Efficient Multilevel Solvers and High Performance Computing Techniques for the Finite Element Simulation of Large-Scale Elasticity Problems

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften

Der Fakultät für Mathematik der
Technischen Universität Dortmund
vorgelegt von

Hilmar Wobker

Efficient Multilevel Solvers and High Performance Computing Techniques for the Finite Element Simulation of Large-Scale Elasticity Problems

Hilmar Wobker

Dissertation eingereicht am: 14.12.2009
Tag der mündlichen Prüfung: 11.03.2010

Mitglieder der Prüfungskommission:

Prof. Dr. S. Turek (1. Gutachter, Betreuer)
Prof. Dr. F.-T. Suttmeier (2. Gutachter)
Prof. Dr. R. Scharlau
Prof. Dr. J. Stöckler
Dr. Ch. Becker

Für
Svea, Jannis & Lili

Acknowledgements

I am sincerely grateful to Professor Stefan Turek, who gave me the opportunity to develop and implement my ideas that finally led to this thesis. He always cared for a nice and friendly working atmosphere at the Institute of Applied Mathematics, he was available for countless discussions and showed me the right directions by asking the decisive questions and pinpointing the essential problems. He was always aware and appreciative of the technical and implementational difficulties that occur when developing a complex software package like FEAST.

I would like to express my gratitude to Professors Franz-Theo Suttmeier, Bob Svendsen and Franz-Joseph Barthold who helped me during the initial phase of my PhD studies to deepen my understanding of the continuum mechanical background and of the algorithmic techniques to treat solid mechanical problems.

I am very grateful to the members of the FEAST group, Christian Becker, Sven Buijssen, and Dominik Göddeke, who helped me finding the path into and out of the 'abyss' of parallel computing, processor architectures, compilers, scripting, and all the other technical details that need to be known to do high performance computing. Especially I would like to thank Dominik Göddeke for proofreading this thesis and providing constructive suggestions for improvement.

I would like to thank Susanne Kilian for fruitful discussions on the basic FEAST concepts, Matthias Grajewski for his help on FEM theory, Michael Köster for performing some reference computations with the FEATFLOW software, and Jaroslav Hron for his valuable help on various issues, ranging from theoretical questions about linear solvers to implementational details of constitutive laws to technical compiler problems.

I am thankful for the support by the NRW Graduate School of Production Engineering and Logistics, the DFG (Deutsche Forschungsgemeinschaft; TU 102/11-3) and the BMBF (Bundesministerium für Bildung und Forschung; 01IH08003D / SKALB).

Finally, I would like to express my deep gratitude to my wife Svea and my children Jannis and Lili. Without their love and patience it would have been much harder to keep up the motivation to finish this thesis. I am very grateful to my parents Klaus and Barbara who facilitated my scientific education and supported me in everything I aimed for.

Dortmund, November 13, 2009

Hilmar Wobker

Contents

1 Introduction

In this thesis we are concerned with efficient solution techniques for equation systems arising in finite element simulations of elasticity problems. In computational solid mechanics (CSM) the need for high performance computing (HPC) techniques has not evolved as early as, e. g., in the field of computational fluid dynamics (CFD): Many practical problems could – and still can – be solved with a simple workstation in a reasonable amount of time. However, there is an increasing need to simulate more complex real-world processes, e. g., industrial manufacturing, structural optimisation, static and dynamic behaviour of buildings or bridges, crash tests in the automotive industry or geological phenomenons like earthquakes. Especially, the treatment of nearly incompressible materials (e. g., rubber) is a demanding task. Such materials are frequently used in manufacturing, e. g., as flexible diaphragms in hydraulic systems to adjust the pressure, as bushings in suspension systems to minimise vibrations or to absorb shocks, and, more generally, to provide damping in complex mechanical systems or to carry large loads (e. g., tires or gaskets). Hence, there is a great need to simulate nearly incompressible materials in industrial applications. For such real-world problems, desktop computers are often not sufficient: Complicated nonlinear material models and time-dependent processes can lead to unreasonable computation times, while large, intricate geometries can result in discrete problem sizes that exceed the computer's main memory.

The simulation of the underlying physical processes which are usually modelled in terms of partial differential equations (PDEs), often requires to repeatedly solve discrete equation systems. For complex real-world applications, these systems are usually much too large to be solved with a direct solution method, such that iterative solution schemes are mandatory. The efficiency of such methods can be crucially influenced by different factors:

- material parameters,
- nonlinear effects,
- the shape of the geometry,
- the size and the quality of the underlying computational mesh,
- algorithmic parameters, and
- the number of processors in a parallel computing system.

We distinguish between three different aspects of 'efficiency': The *processor efficiency* characterises the degree to which a simulation software is able to actually exploit the processor's computational power. The *parallel efficiency* describes the ratio between computation and communication times on distributed computer systems, while the *numerical efficiency* refers to the convergence behaviour of the numerical solution algorithm. The development of 'black-box' solution techniques that gain high efficiencies in all three aspects and that are robust with respect to the perturbing influences listed above is a demanding task. Additionally, these solution techniques should be flexible: Although this thesis is confined to the treatment of elasticity problems, the goal is to develop solver strategies that can be applied – without or with only slight modifications – to other physical problem settings, as well. The solution algorithms should exclusively use standard operations that are available in basic PDE software packages; special problem-dependent modifications should be avoided.

We build this solver machinery on top of our parallel finite element solution toolkit FEAST (Finite Element Analysis & Solution Tools). It is specialised to solve scalar elliptic equations in a highly efficient way, exploiting the abilities of modern supercomputers and clusters of commodity based processors. The goal is to design solvers that reduce the solution of multivariate problems to the solution of a series of scalar problems, such that the efficiency of the underlying FEAST library can be fully exploited. The corresponding concepts are realised and explained by means of the program module FEASTsolid, a new application on top of FEAST for solving stationary and transient, small and finite deformation, compressible and (nearly) incompressible elasticity problems.

This thesis mainly focusses on the development of solution techniques for multivariate elasticity problems. However, further important aspects of the finite element discretisation and the overall solution process are treated: A fundamental weakness of FEAST's scalar solver concept is identified, and a possible remedy to overcome this problem is presented. Furthermore, the robustness of FEAST's scalar solvers is improved by incorporating multigrid-Krylov techniques, and the whole solution approach is critically discussed. To treat (nearly) incompressible material we apply a mixed finite element formulation that needs to be stabilised. Standard stabilisation techniques are often not suited for irregular meshes. We present an enhanced stabilisation variant that is efficient for arbitrary meshes, and we modify this variant for the case of nonlinear finite deformation elasticity.

The following outline shows that this thesis covers a wide range of different aspects. Hence, parts of the work naturally have a survey character, and not all topics can be treated in full detail. For the same reason, we use numerical experiments rather than theoretical analysis to examine and test the developed concepts. The thesis is organised as follows:

- In Chapter 2 we describe in detail the basic FEAST library, in particular the idea of *hardware-oriented numerics*. We emphasise that the concept of data locality, which many finite element tools are lacking, is essential to alleviate the so called 'memory wall problem' and to minimise communication overhead. We illustrate the conflicting demands of processor and parallel efficiency on the one hand, and numerical efficiency on the other

hand, and we illustrate how FEAST's solver concept ScaRC tries to resolve these conflicts. ScaRC is a generalised multilevel domain decomposition/multigrid (MLDD/MG) solver with minimal overlap. One of its core ideas is to use a global multilevel method which is smoothed by local schemes that are adapted to the local situation, e. g., local mesh irregularities. When this local scheme is a multigrid solver itself, we call the entire solver 2-layer-ScaRC, when it is an elementary smoother like Jacobi, then the whole solver is referred to as 1-layer-ScaRC. We describe a fundamental deficiency of the local multigrid solver, which is a consequence of how the concept of minimal overlap is realised. We demonstrate that this deficiency especially becomes evident in the case of the elasticity equation, and we present a technique to overcome the problem. Furthermore, we show how the robustness of the ScaRC solvers can be significantly increased by adding Krylov schemes on the global and on the local layer. Finally, we critically compare the 1-layer-ScaRC and the 2-layer-ScaRC approach, and examine under which circumstances 2-layer-ScaRC is superior.

- In Chapter 3 we present the continuum mechanical basics of elasticity, including the important restriction to 2D linearised elasticity. We describe the finite element discretisation in terms of a pure displacement formulation and in terms of a mixed displacement/pressure formulation which is used to treat (nearly) incompressible material. We motivate the use of the low-order bilinear finite element Q_1 and discuss its advantages and disadvantages with respect to software aspects on the one hand, and with respect to typical locking effects that occur in solid mechanical simulations on the other hand. Stabilisation techniques for the unstable Q_1/Q_1 element pair for discretising the mixed formulation are presented. We develop an enhanced stabilisation method which is suitable for highly irregular meshes. All stabilisation variants are extensively compared by means of numerical examples, including nonlinear finite deformation tests. Finally, the deficiencies of the stabilised Q_1/Q_1 formulation are illustrated, especially for the case of transient simulations.

- In Chapter 4 we motivate and describe our main strategy to bring down the process of solving multivariate problems to the solution of a series of scalar problems. The main motivation is to automatically provide the programmer of multivariate physical applications with FEAST's fully parallel and processor-efficient discretisation and numerical linear algebra concepts, and its support of special hardware technologies. The application profits – without being changed – from future improvements and adaptations to forthcoming HPC trends, i. e., the underlying technical details can be completely hidden from the application programmer. The concept is illustrated with the example of linearised elasticity for compressible material, and it is realised by means of the program module FEASTsolid; the general idea, however, is applicable in most general-purpose PDE solution tools.

  We demonstrate the efficiency of using FEAST's scalar ScaRC solvers as preconditioners for multivariate solvers. However, we also show that it is not always sufficient to use a 'simple' outer solver and to fully rely on the efficiency and robustness of the scalar

ScaRC solvers. In some situations, e. g., when the global domain is strongly anisotropic, we have to use more sophisticated multivariate solvers. Similar to Chapter 2, we emphasise the importance of multigrid-Krylov techniques. In extensive numerical tests, we compare various combinations of local/global and scalar/multivariate solver components, and we identify an efficient solver with potential 'black-box' character. We demonstrate that the processor and parallel efficiency of the underlying FEAST library fully transfers to the multivariate solvers, and we show that the possible superiority of the 2-layer-ScaRC approach can also be observed when employed for preconditioning an outer multivariate solver.

In the second part of Chapter 4 we are concerned with the solution of finite deformation elasticity problems for compressible materials. The nonlinear equation is treated by means of the Newton-Raphson method. In each nonlinear iteration, a linear equation system has to be solved for which the same techniques can be applied as in the linearised elasticity case. Hence, also the solution of nonlinear elasticity problems can essentially be reduced to the solution of scalar subsystems, and the whole nonlinear solver automatically runs in parallel. We examine techniques to adaptively determine the accuracy for the linear solves in order to decrease the arithmetic costs of the Newton-Raphson method, and we use line-search methods to significantly increase its robustness.

- In Chapter 5 we examine solution strategies for the important class of saddle point systems. Such systems stem from the mixed displacement/pressure formulation that is used to robustly treat (nearly) incompressible materials. Due to the similarity to the Stokes equation, we can apply techniques that are commonly used in CFD. We distinguish two main solver classes: on the one hand, *segregated* methods, i. e., pressure Schur complement and block preconditioning approaches, and on the other hand, *coupled* methods, i. e., Vanka-type multigrid solvers. The former allow the application of our basic strategy of reducing the solution of multivariate systems to the solution of scalar systems, while the latter fully operate on the basis of local representations of the entire multivariate saddle point system such that a reduction to scalar components is not possible. On the other hand, segregated methods urgently require efficient Schur complement preconditioners that are usually problem dependent, while coupled solvers do not need such preconditioners. Vanka solvers are very popular for solving saddle point problems, especially in the context of CFD. We show that the segregated methods using the underlying FEAST library are much more efficient (provided an efficient Schur complement preconditioner is available) than the coupled solvers. We describe Schur complement preconditioning techniques for stationary and transient linearised elasticity problems. It turns out that the transient preconditioners are not robust with respect to the time step size, which is probably due to the essential deficiency of the stabilised Q_1/Q_1 element pair in the small time step limit which is described in Chapter 3.

  Segregated saddle point solvers need to solve subsystems that have the same structure as those stemming from the pure displacement formulation. However, the solver that we identified as favourite in Chapter 4 is not efficient when applied as a subsolver in

saddle point schemes. It turns out that it is generally difficult to identify *the* best segregated solver. Such a saddle point solver has to nest iterative schemes on up to three different layers. We demonstrate that for complicated configurations, it can be necessary to use multigrid schemes on all three layers in order to obtain convergence at all, while for simple configurations one multigrid scheme (e. g., on the scalar layer) is often sufficient. However, a saddle point solver using multiply nested multigrid scheme exhibits bad parallel efficiency, i. e., it should only be employed if it is actually necessary from a numerical point of view.

The numerical studies of the various saddle point solvers are confined to the case of linearised and stationary elasticity problems. We demonstrate by means of one example that FEASTsolid is able to solve nonlinear and transient saddle point systems, as well. We briefly discuss the possible deterioration of the standard Schur complement preconditioners in more complicated situations, and mention possible alternatives.

2 Multilevel Solvers for Scalar Elliptic Equations

The solution of elliptic partial differential equations (PDEs) is one of the most important, but also one of the most challenging tasks in the numerical simulation of physical and technical processes, especially in the fields of CSM and CFD. While the equations of (linearised) elasticity, representing the most important model in CSM, are elliptic by themselves, scalar elliptic subproblems have to be solved for the (linear) Stokes and (nonlinear) Navier-Stokes equations. The treatment of these subproblems is the most time-consuming part in the course of the iterative process to solve the entire saddle point systems (see Kilian [94], Turek [158]). This observation was the main motivation for the development of the so called ScaRC solvers and their technical realisation within the FEAST (Finite Element Analysis & Solution Tools) project. They represent a class of generalised multilevel domain decomposition/multigrid solvers that are highly specialised for scalar elliptic equations. In this chapter we explain the underlying concepts of ScaRC and develop substantial improvements of important solver components. This work has to be seen as continuation and extension of the theses of Kilian [94] and Becker [16] and as a contribution to the FEAST project.

In Section 2.1 we illustrate the general difficulty of achieving good numerical efficiency and exploiting modern computer hardware at the same time. We describe how FEAST overcomes this problem. Section 2.2 gives a brief overview of domain decomposition methods, while Sections 2.3 and 2.4 describe FEAST's grid and refinement model and the concept of minimal overlap. In Section 2.5 we define and classify FEAST's solver concept ScaRC, whereas we choose a different focus than Kilian [94] and Becker [16]: We are especially interested in the two possible interpretations of the solver scheme as a *block-smoothed multigrid method* or a *multilevel Schwarz method*. This point of view is taken up in Section 2.6.4 where we numerically compare the two concepts. In Section 2.6 we show how important ScaRC components can be significantly improved. In Section 2.6.1, especially, we illuminate an intrinsic weakness of the ScaRC approach, which has been paid only minor attention until now, and we offer a possible remedy. Finally, we describe open problems and future work on ScaRC in Section 2.7.

2.1 Numerical Efficiency vs. Hardware Efficiency

In order to assess the quality of a numerical method, different efficiency aspects have to be considered:

- The **numerical efficiency** characterises the amount of work a numerical method needs to achieve some desired goal. On the one hand, this means the convergence rates and robustness of iterative solution methods, i. e., the required number of iterations and how this number varies with respect to changing algorithmic or physical parameters. On the other hand, it means the ability of the method to employ goal-oriented adaptation techniques, for example the concentration of grid refinement to regions of interest in order to keep the total number of unknowns small.
- The **processor efficiency** describes the ability of an algorithm to exploit the CPU's full capacities. It is mainly determined by the success of the algorithm to circumvent the so called 'memory wall problem' (see Section 2.1.1).
- The **parallel efficiency** determines the ratio of communication and computation, i. e., it relates the number of data exchanges between processors and the amount of exchanged data to the amount of arithmetic work performed on the processors. In view of today's compute clusters consisting of several thousand processors, it is critical how this ratio scales when the number of processors increases.
- The term **hardware efficiency** is used to summarise the two terms 'processor efficiency' and 'parallel efficiency'.

The main motivation for the development of FEAST is the realisation of 'hardware-oriented numerics' [162], i. e., to achieve good efficiency in all these aspects. Due to conflicting requirements this is a demanding task: First, flexible and dynamic grid structures are needed to describe complicated geometries and to facilitate adaptivity techniques ($\rightarrow$ numerical efficiency), while highly regular grid patterns are required to exploit modern processor technology ($\rightarrow$ processor efficiency). Second, global data transfer is needed to achieve good convergence rates ($\rightarrow$ numerical efficiency), while locality is desired for good parallel performance ($\rightarrow$ parallel efficiency). The following two sections describe these conflicts in more detail and explain FEAST's strategy to resolve them. For a general discussion how computer hardware trends influence numerical methodology, especially for PDEs, see, for example, Colella et al. [53], Keyes [93] and Rüde [135].

2.1.1 Processor Efficiency

All efforts to achieve good processor efficiency have to be based on the insight that in modern computer architectures not *data processing*, but *data moving* is critical. This well known fact is generally denoted as the *'memory wall problem'* [174].

The following statistics are taken from Graham et al. [77]. Over the last decades peak processor performance improved at a rate of nearly 60 % per year. While at the end of the 1980s commodity processors showed a peak performance of about one million floating point operations per second (1 MFLOP/s), they were about 1000 times faster in the year 2004. In contrast, improvements in memory bandwidth and memory latency happened much more slowly: While memory

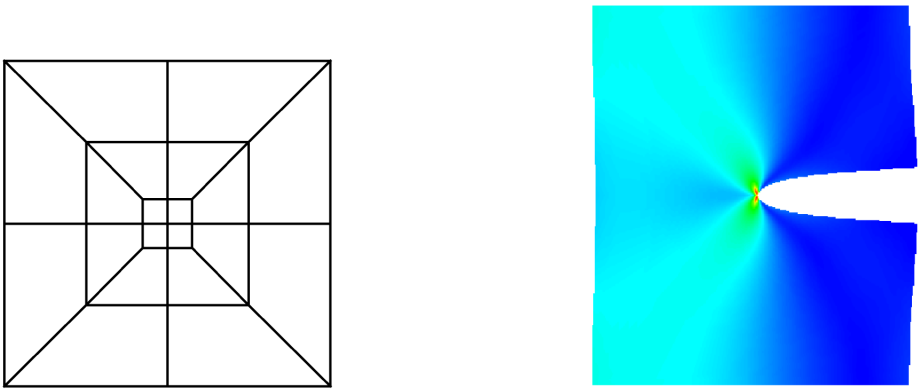
bandwidth increased by roughly 30 %, memory latency improved by only 5.5 % per year. This means, while in 1992 one floating point operation (FLOP) took about the same time as accessing the memory, in 2004 one FLOP was about 100 times faster than memory access. This *'memory gap'* is expected to broaden at a similar rate over the next years and, thus, is considered the *main obstacle for achieving high processor performance* in data intensive applications. Especially FEM codes suffer from this development, since they deal with sparse matrices and thus exhibit a low *arithmetic intensity*, which means that only few arithmetic operations are performed per data transfer. Consequently, the performance of FEM codes is usually not bound by the processor peak, but by the memory bandwidth.

Today's commodity processors try to bridge this memory gap by using a *cache* system – a hierarchy of intermediate memory layers which can be accessed by the processor very fast but have relatively small storage size only. In order to 'keep the processor busy', one of the key properties of modern software has to be seen in the ability to exploit this caching technique. A prerequisite for this is to develop applications that work on *structured data with high spatial and temporal locality*: In order to reach a considerable ratio of the processor's peak performance, an algorithm has to be able to fetch 'the right data at the right moment' into the cache, to sufficiently exploit the cache size (spatial locality), and to perform as many operations as possible on this data (temporal locality, enabling *'data reuse'*).

A major drawback of many FEM codes is that they *do not* fulfil this demand. The reason is the use of grids, which are either initially unstructured or lose their structure in the course of the simulation. The former may stem from automatic grid generation tools that are mainly focussed on geometry approximation but not on creating well structured grids. The latter are a consequence of *adaptive grid refinement*, which is one of the key components in modern simulation tools to achieve high numerical efficiency: Guided by the user or by some automatic error control mechanism, the grid is refined only where necessary. Especially in 3D computations it is prohibitively expensive – in terms of memory consumption and computation times – to uniformly refine the whole grid in order to achieve a desired accuracy. Such adaptive refinement procedures usually result in highly unstructured grids. FEM matrices – being directly connected to the grid data – are often stored in the standard *compact sparse row (CSR)* format [13] which provides the necessary flexibility to account for the missing structure in the grid. However, this enforces the usage of pointer structures ('indirect addressing') which violates the locality paradigm and, thus, prevents the employment of cache-oriented techniques. What follows is that, for instance, matrix vector multiplications – one of the core components of many computational routines – are often executed with less than one percent of the theoretically possible MFLOP/s rate of the processor. Multigrid solvers perform even more slowly, thus often losing the linear dependency of computing time on the problem size (see Köster et al. [102], Turek et al. [161, 162] for more details).

So, the following critical question arises: Is it possible to work with complex geometries and adaptive refinement strategies and, at the same time, maintain (some degree of) grid structure that allows for achieving high processor efficiency?

Actually, these two seemingly conflicting demands – processor efficiency on the one hand and numerical efficiency (in terms of grid adaptation techniques) on the other hand – *can* be successfully combined. The realisation of this goal is one of the main motivations of FEAST. The basic idea is to define the geometry *patch-wise* and to use *locally structured grids* within the patches. This separation of unstructured and structured data on the one hand provides great flexibility in resolving even complicated geometries, and on the other hand facilitates hardware-efficient implementation. Before we describe our approach in detail (see Section 2.3), we want to illustrate it here with the help of a well known simulation example, a square-shaped domain with a slit that is stretched such that the slit opens (see Figure 2.1b). The geometry is described by an unstructured collection of quadrilateral patches (see Figure 2.1a), which is the starting configuration for the computation and is called the *coarse grid*. The coarse grid



(a) Coarse grid consisting of 20 patches. **(b)** Deformed domain, von Mises stresses displayed.

Figure 2.1: Slit configuration.

is then refined in such a way that each patch describes a **generalised tensor product grid**, i. e., each vertex lying in the interior of a patch has exactly four neighbouring vertices. This structure is the key requirement to achieve high processor performance: Applying rowwise numbering in such a tensor product grid results in a matrix with a predefined band structure. In case of the bilinear element Q_1, for instance, the matrix consists of exactly nine bands as illustrated in Figure 2.2. Every patch of the coarse grid corresponds to such a *local matrix*.

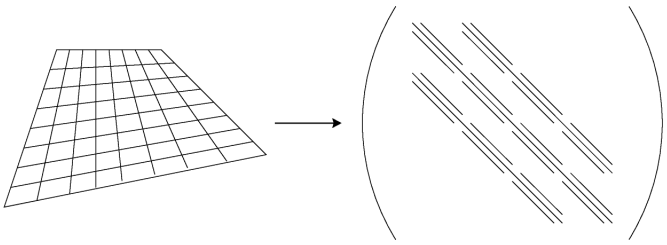


Figure 2.2: Generalised tensor product grid and corresponding band-structured finite element matrix.

A corresponding *global matrix* is only implicitly represented by applying the local matrices and exchanging information on patch boundaries. Knowing the exact matrix structure a priori, highly optimised linear algebra routines can be developed. The crucial advantage compared to the standard sparse storage technique is the possibility of *direct addressing*: matrix entries can be immediately accessed via array indices such that no intermediate pointer structure is necessary. Consequently, FEAST's data structure exhibits high locality and thus fulfils the precondition for cache-oriented programming. The described concept has been realised within the SparseBandedBLAS library and its BlockedBanded version which represent an important part of the FEAST project. Especially, matrix vector multiplications, which consume most of the time in finite element simulations, can be performed at significantly higher MFLOP/s rates than with standard sparse storage techniques. The band structure of the matrices is further exploited by applying specialised line-wise working tridiagonal multigrid smoothers. Due to the 'gaps' in the bands (see Figure 2.2), which are a consequence of the rowwise numbering, the tridiagonal part of the matrix splits into $\sqrt{n}$ blocks where n is the number of vertices in the patch. This blocking allows for highly efficient cache-aware implementations of the corresponding LU decompositions [2]. An alternating direction Gauß-Seidel variant of the tridiagonal smoother (ADITriGS) has been shown to be very robust with respect to anisotropies occurring in the grid or in the operator (see Altieri [2], Turek et al. [161, 162] for details and also Section 2.5.3.1).

FEAST's grid adaptation strategies are out of the scope of this work and described in detail in the dissertation of Grajewski [78]. Here, we only briefly illustrate how to combine the described meshing concept with such grid adaptation techniques. A first possibility is to use different patch refinement levels, i. e., *patch-wise h-adaptivity*. Which patches to refine is decided either by the user (a priori), or automatically by some error control mechanism (a posteriori). Figure 2.3 shows how this concept can be used to better resolve the tip of the slit. FEAST allows for

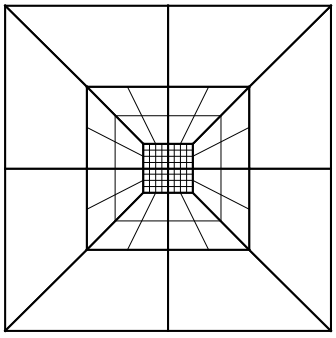
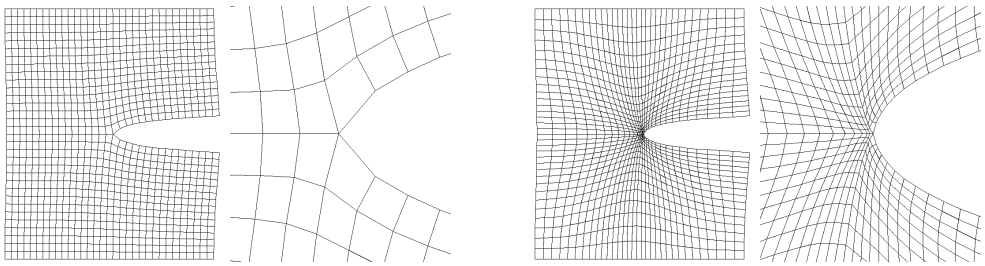


Figure 2.3: Example for *h*-adaptivity: Adaptively refined coarse grid of the slit configuration.

'1-irregular hanging nodes' on patch boundaries, i. e., the refinement levels of two neighbouring patches may differ by at most one. On the one hand, this provides some degree of flexibility in the refinement process, e. g., the introduction of transition elements is not necessary. On the other hand, however, the concept can also be too restrictive since strongly localised effects will finally result in refinement of regions which are of minor interest. If, for instance, the inner patches of Figure 2.3 were refined once more also the outer patch ring would have to be refined.

That is why FEAST additionally allows for *anisotropic refinement* (see Section 2.3) and *grid deformation techniques (r-adaptivity)* (see Figure 2.4) to obtain more flexible adaptive designs. In this manner boundary layers or singularities (like the tip of the slit) can be resolved. It is



(a) before grid deformation **(b)** after grid deformation

Figure 2.4: Example for *r*-adaptivity: Deformed grid of the slit configuration and magnified view of the slit tip. The coarse grid of this configuration consists of four patches (not shown).

important to note that meshes which are subject to anisotropic refinement or grid deformation do not lose the tensor product property, i. e., the before mentioned caching techniques are still applicable.

2.1.2 Parallel Efficiency

Besides processor efficiency, another crucial criterion for modern software with respect to exploitation of today's (super)computer architectures is *parallel efficiency*. The main problem of parallelising a numerical algorithm is that *parallel efficiency* and *numerical efficiency* are often two contrary properties. To achieve good convergence rates *global information* is needed, which is especially true for elliptic problems: The solution in an interior mesh node is influenced by all boundary values, i. e., information has to 'travel' at least once through the whole mesh. In contrast, *locality* is urgently required to minimise communication between processors. A similar argument as for the 'memory gap' (see Section 2.1.1) holds for the performance increase of the interconnecting network: Advances in global bandwidth and global latency are significantly slower than in processor technology. Some facts presented by Graham et al. [77] show this: The improvement rates of network technology over the last years were roughly 30 % per year (compared to 60 % for processor performance). However, these rates are not expected to be sustainable due to physical constraints like the speed of light. As an example, when in 2004 one inter-processor communication took about as long as 4000 FLOPs, then in the year 2020 roughly 670,000 FLOPs can be performed during one communication. So, the overall parallel performance of an algorithm clearly is and will be determined by the amount of data exchange.

Another trend observable in parallel computer architectures is an increasing number of processors. While high-end supercomputers in the year 2004 employed up to 4000 processors, this number is expected to rise by a rate of about 20 % per year such that supercomputers in

the year 2020 will be likely equipped with more than 70,000 processors.[1] This trend enforces parallel algorithms to be *scalable*: The runtime should decrease inversely proportional to the number of processors (*'strong scaling'*), or, when increasing the problem size and the number of processors by the same factor, the runtime should (ideally) remain constant (*'weak scaling'*). In the context of finite element simulations a higher degree of parallelism is typically achieved by dividing the computational domain into more smaller subdomains. In doing so, the number of boundary nodes per subdomain decreases by a lower factor than the total number of nodes per subdomain, i. e., the ratio between communication and computation naturally deteriorates. Only when exhibiting a high degree of locality, one has the chance to weaken this loss of scalability. This means that data locality has to be considered again as *the* critical requirement for the solution algorithm to achieve high parallel efficiency.

In summary, parallel solution methods have to meet the following requirements:

- low inter-processor communication
- high processor loads
- good scalability
- good convergence behaviour
- robustness with respect to complicated geometries, mesh refinement level, mesh irregularities, and number/size of subdomains

The first three points determine the solver's parallel efficiency, the latter two the numerical efficiency, where all aspects, of course, influence each other and can thus not be considered completely disjunct. The critical question is now how to design such a solution method.

State-of-the-art solvers for elliptic problems that are able to meet the requirements concerning the numerical efficiency are *multilevel domain decomposition* (MLDD) and *multigrid methods* (MG). Both approaches employ coarse grids and are thus able to 'transport information faster through the domain'. For elliptic problems, this feature is mandatory for achieving good convergence rates that do not depend on the refinement level of the mesh. So, the question is how the two solution approaches can be parallelised. In multilevel domain decomposition methods, which are described in more detail in Section 2.2, the computational domain is subdivided into overlapping subdomains. Parts of the overall solution process are performed independently on these subdomains, such that MLDD methods can be naturally parallelised by distributing the subdomains and the corresponding operations to the available processors.

Multigrid methods act on a hierarchy of nested meshes, employing a combination of smoothers, grid transfer operators and coarse grid solvers. Their success is based on the fact that elementary iterative methods like Jacobi or Gauß-Seidel have poor solving abilities, though, but smooth oscillations in the defect vector after only few iterations. This smoothed vector can then be

[1] The current 'Top500' list (`http://www.top500.org/list/2009/06/100`) of the most powerful computer systems in the world shows, that reality already caught up with this forecast: There already exist systems with more than 100,000 processors.

well approximated on a coarser grid. The alternating procedure of smoothing and restricting the defect vector is repeated over all refinement levels until the problem on the coarsest mesh is solved exactly. The resulting coarse grid correction is prolongated to the finer level and used to update the solution vector there, possibly followed by a postsmoothing process. This procedure is repeated until the finest level is reached. Due to the highly recursive nature of smoothing operations like Gauß-Seidel or ILU, multigrid algorithms exhibit a very poor parallelisation potential. A typical way to overcome this problem is to subdivide the mesh, such that the smoother can be defined *block-wise* and perform concurrently on the single subdomains, while the other MG components are still applied globally. The resulting algorithm is referred to as *block-smoothed multigrid method*.

Actually, multilevel domain decomposition methods and block-smoothed multigrid methods have much in common. FEAST's solution scheme exploits this fact to combine the two methods in a generalised MLDD/MG concept called ScaRC. The specific features that distinguish the two methods are discussed in Sections 2.5.2 and 2.5.3. They play an important role for the development and the evaluation of the ScaRC concept. Section 2.5 shows that ScaRC is able to largely fulfil above listed requirements of a modern parallel solution method.

2.2 A Brief Survey of Domain Decomposition Methods

Domain decomposition methods are used to solve linear systems of equations, $\mathbf{Ax} = \mathbf{b}$. The core idea of these methods is a classical 'divide and conquer' strategy: The computational domain is decomposed into several (eventually overlapping) subdomains, the (large) global problem is restricted to (smaller) local ones, which are solved to obtain local corrections for the global solution vector. The process is repeated until the coupling between the local parts of the solution is sufficiently resolved. Hence, domain decomposition methods are always embedded into some 'outer' iterative solution procedure. The simplest iterative algorithm is the following defect correction scheme,

$$\mathbf{x} \leftarrow \mathbf{x} + \omega\tilde{\mathbf{A}}(\mathbf{b} - \mathbf{Ax}), \tag{2.1}$$

where the operator $\tilde{\mathbf{A}}$ is a **preconditioner** of the matrix $\mathbf{A}$ and ω a damping parameter. In general, $\tilde{\mathbf{A}}$ is not available in closed matrix form, but rather stands for a possibly complicated algorithm to calculate the application of the preconditioner. Here, it represents the application of a domain decomposition method. Procedure (2.1) is also called **basic iteration** or **preconditioned Richardson iteration**. The aim of preconditioning is to significantly decrease the number of iterations that are necessary to satisfy a predefined termination criterion and, thus, reduce the overall costs of the solution procedure. A good preconditioner has to fulfil two requirements:

1. The matrix $\tilde{\mathbf{A}}\mathbf{A}$ should be 'close to identity', i. e., its condition number should be much smaller than that of the matrix $\mathbf{A}$.
2. The application of $\tilde{\mathbf{A}}$ should be inexpensive compared to the computation of $\mathbf{A}^{-1}$.

The first requirement is necessary for improving the convergence rate of the iteration, but it is not sufficient to make the overall algorithm more efficient. This becomes clear when considering the extremal case $\tilde{\mathbf{A}} := \mathbf{A}^{-1}$. Although method (2.1) will terminate after one iteration in this case, the overall costs are not reduced but simply shifted to the preconditioning phase. Hence, preconditioning offers the potential to actually reduce the total solving time *only* if the second requirement is met, as well. Domain decomposition methods fulfil both requirements: On the one hand, the global operator is approximated by a combination of local restrictions of this operator, and, on the other hand, the solution of several local problems is cheaper than the solution of one global problem, especially when they can be solved in parallel. Consequently, it makes sense to consider decomposition methods as preconditioners.

There are two classes of domain decomposition algorithms, *nonoverlapping* and *overlapping methods*. The former are often denoted as *(iterative) substructuring* or *Schur complement methods*, popular representatives are *Neumann-Neumann*, *Dirichlet-Neumann* and *FETI methods*. Overlapping domain decomposition methods go back to the alternating Schwarz method on overlapping subdomains [142] and are thus often referred to as *Schwarz methods*. FEAST's solver approach belongs to the latter class. An overview of the numerous variants of domain decomposition algorithms, or a comparison between overlapping and nonoverlapping methods is clearly out of the scope of this work. We refer to the monographs by Smith et al. [150], Toselli and Widlund [156] for a general overview, and to the dissertation of Kilian [94] for a extensive comparison of those domain decomposition methods that are important for the derivation of FEAST's solving concept. The computational overhead and the additional communication effort is considered to be the main disadvantage of the Schwarz methods compared to nonoverlapping methods. FEAST alleviates this effect by using a concept of minimal overlap (see Section 2.3). On the other hand, Schwarz methods are considered to be applicable to a wider class of problems, while substructuring methods often need special preconditioners to treat the arising interface problems. In terms of data structures, nonoverlapping methods need to take extra care of interfaces and Schur complement systems, especially the extension from 2D to 3D is much more complicated than for overlapping methods.

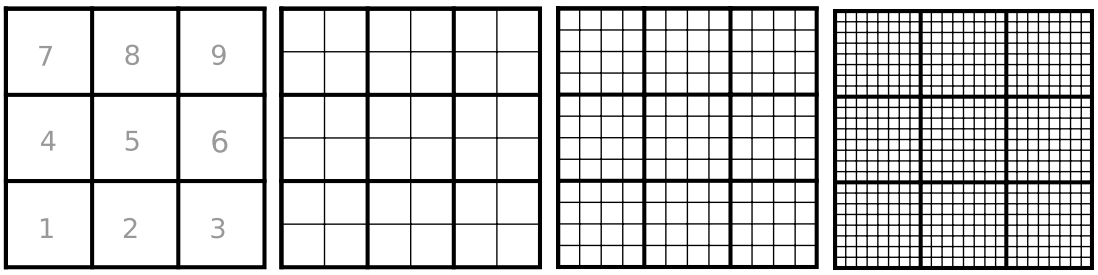
Schwarz methods are divided into one-level, two-level and multilevel methods. The critical drawback of the classical **one-level Schwarz methods**, which work on a single grid, is that the convergence rate of the outer iterative algorithm depends on the number of subdomains. This is due to the nature of elliptic problems: The solution in an interior point depends on all boundary values, i. e., information has to traverse the domain completely from one end to the other at least once. Each subdomain exchanges data only with its neighbouring subdomains, hence the pace at which information spreads depends on the number of subdomains. This can be overcome by solving the equation additionally on a much coarser grid, thus providing a global exchange of information within one iteration step. Although the costs for solving this *coarse grid problem* are usually low compared to other components of the overall algorithm, its addition suffices to render the convergence rates of the resulting **two-level Schwarz methods** independent of the number of subdomains. When there are many subdomains, the coarse grid problem might be too large to be solved directly. In this case, it can be solved iteratively by using another

two-level Schwarz method. Applying this idea recursively leads to the concept of **multilevel Schwarz methods** (see Section 2.5).

There are two ways to combine the local operators in overlapping domain decomposition methods: While **additive Schwarz methods** solve the local problems independently of each other and update the global solution vector simultaneously, **multiplicative Schwarz methods** treat these subproblems after each other such that recent solution updates can immediately be incorporated into the right hand sides of the local subproblems which are to be solved next. Intuitively, it is clear that multiplicative methods exhibit better convergence properties since they propagate information faster through the grid. The relation between additive and multiplicative Schwarz methods is comparable to the relation between (block-)Jacobi and (block-)Gauß-Seidel methods. Actually, the latter two can be interpreted as additive and multiplicative Schwarz methods, resp., with zero overlap. Two-level and multilevel methods offer the additional degree of freedom how to combine operators from different grid levels, i. e., whether they work *additively* or *multiplicatively between levels*. Thus, we obtain purely additive (additive within and between levels), purely multiplicative (multiplicative within and between levels) methods, and so called *hybrid methods* [150]. FEAST's solver concept belongs to the category of hybrid multilevel methods that are additive within levels and multiplicative between levels.

2.3 Grid Partitioning and Refinement

The starting point is a **macro decomposition** of the domain $\bar{\Omega}$ into $M = m^{(0)}$ quadrilateral patches $\Omega_i = \Omega_i^{(0)}$. These subdomains build a triangulation $\mathcal{T}^{(0)} := \{\Omega_i^{(0)} \mid i = 1,\dots,m^{(0)}\}$ (on level 0) with $\bar{\Omega} = \bar{\Omega}^{(0)} := \bigcup_{i=1}^{M} \bar{\Omega}_i^{(0)}$ in case the boundary of Ω is captured exactly, and $\bar{\Omega} \approx \bar{\Omega}^{(0)}$ otherwise (e. g., in case of curved boundaries). The intersection of $\bar{\Omega}_i^{(0)}$ and $\bar{\Omega}_j^{(0)}$ with $j \neq i$ is either empty, a single vertex or an edge. Figure 2.5a illustrates a macro decomposition of the



(a) Level 0 $\bar{\Omega}^{(0)}$. **(b)** Level 1 $\bar{\Omega}^{(1)}$. **(c)** Level 2 $\bar{\Omega}^{(2)}$. **(d)** Level 3 $\bar{\Omega}^{(3)}$.

Figure 2.5: Example for grid partitioning and refinement: The unitsquare $\bar{\Omega}$ decomposed into $M = m^{(0)} = 9$ quadrilateral patches, showing four refinement levels $\bar{\Omega}^{(i)} = \bar{\Omega}_1^{(i)} \cup \ldots \cup \bar{\Omega}_9^{(i)}, i = 0,\dots,3$

unitsquare into nine patches of equal size. By successive regular subdivision (see below) a hierarchy of $L+1$ grid levels is created such that the level l version $\Omega_i^{(l)}$ ($l = 0,\dots,L$) of each

subdomain Ω_i contains $m_{\text{loc}}^{(l)} := 4^l$ elements $e_{i,j}^{(l)}$ $\left(j = 1,\dots,m_{\text{loc}}^{(l)}\right)$ and $n_{\text{loc}}^{(l)} := (2^l+1)^2$ vertices $v_{i,k}^{(l)}$ $\left(k = 1,\dots,n_{\text{loc}}^{(l)}\right)$ (see Figures 2.5b, 2.5c and 2.5d).[2] Finer-grid vertices that correspond to curved boundaries are projected to the real boundary of Ω, such that $\bar{\Omega}^{(l)} \to \bar{\Omega}$ for $l \to \infty$. We obtain the (level l) grid

$$\bar{\Omega}^{(l)} := \bigcup_{i=1}^{M} \bar{\Omega}_i^{(l)}, \qquad l = 1,\dots,L, \tag{2.2}$$

which is decomposed into M subgrids $\Omega_i^{(l)}$ and consists of $m^{(l)} := M \cdot m_{\text{loc}}^{(l)}$ elements $e_j^{(l)}$ ($j = 1,\dots,m^{(l)}$) and $n^{(l)} < M \cdot n_{\text{loc}}^{(l)}$ vertices $v_k^{(l)}$ ($k = 1,\dots,n^{(l)}$). We denote the number of vertices and elements on the finest grid with $n := n^{(L)}$ and $m := m^{(L)}$, respectively. Correspondingly, we define the (level l) triangulation[3]

$$\mathcal{T}^{(l)} := \{e_j^{(l)} \mid j = 1,\dots,m^{(l)}\}. \tag{2.3}$$

The initial grid $\bar{\Omega}^{(0)}$ only serves for defining the macro decomposition of the domain, but is not used as computational grid. For implementational reasons, the actual coarse grid of all our multigrid/multilevel algorithms is $\bar{\Omega}^{(1)}$.

We apply two different kinds of refinement. In case of **isotropic refinement**, the midpoints of two opposite element edges are connected. In case of **anisotropic refinement** we move the points to be connected along the edges to create four elements with varying aspect ratios. In detail, we use a function $\boldsymbol{s}^{(l)} : \mathbb{R}^2 \times \mathbb{R}^2 \mapsto \mathbb{R}^2$ that defines the refinement from level $l-1$ to level l and is given by

$$\boldsymbol{s}^{(l)}(\boldsymbol{z}_1,\boldsymbol{z}_2) := \begin{cases} \boldsymbol{z}_1 + a_{\text{F}}\frac{1}{2}(\boldsymbol{z}_2-\boldsymbol{z}_1), & l = 1 \\ \boldsymbol{z}_1 + a_{\text{I}}\frac{1}{2}(\boldsymbol{z}_2-\boldsymbol{z}_1), & l = 2,\dots,L-1 \\ \boldsymbol{z}_1 + a_{\text{L}}\frac{1}{2}(\boldsymbol{z}_2-\boldsymbol{z}_1), & l = L. \end{cases} \tag{2.4}$$

Here, $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are start and end point of an edge on level $l-1$, and the three factors $0 < a_{\text{F}}, a_{\text{L}}, a_{\text{I}} < 2$ determine the vertex positions for the first, the last and all intermediate refinements, respectively. Two opposite edges are refined equally such that at the end six factors $a_{\text{F}}^x, a_{\text{M}}^x, a_{\text{L}}^x$ and $a_{\text{F}}^y, a_{\text{M}}^y, a_{\text{L}}^y$ are necessary to completely define the anisotropic refinement. Figure 3.8 on page 158, for example, illustrates various refinements of the unitsquare. Setting all factors to 1.0 coincides with isotropic refinement. For two neighbouring subdomains the refinements along the common boundary have to match to obtain a valid triangulation.

[2] In principle, FEAST is able to handle grids consisting of subdomains with different refinement levels (cf. Section 2.1.1). In this thesis, however, we only work with equal refinement levels.

[3] To ease notation, the symbol e simultaneously denotes the 'topological' element (as part of the triangulation $e \in \mathcal{T}$ consisting of edges and vertices) and the 'geometric' element (as part of the domain $e \subset \bar{\Omega}$).

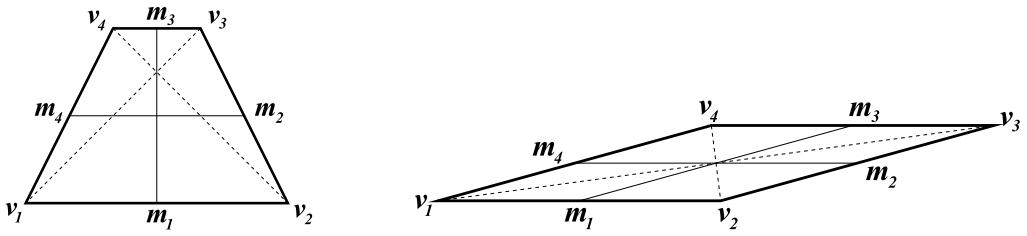
The aspect ratio of an element e is defined as follows. We denote the four vertices of the element (ordered counterclockwise) with $\boldsymbol{v}_1,\dots,\boldsymbol{v}_4$ and the midpoints of the edges with $\boldsymbol{m}_1,\dots,\boldsymbol{m}_4$, where $\boldsymbol{m}_i = \frac{1}{2}(\boldsymbol{v}_i + \boldsymbol{v}_{\mathrm{mod}(i,4)+1})$ (see Figure 2.6). The four values

$$\sigma_1 := \max\left\{\frac{|\boldsymbol{m}_3-\boldsymbol{m}_1|}{|\boldsymbol{m}_4-\boldsymbol{m}_2|}, \frac{|\boldsymbol{m}_4-\boldsymbol{m}_2|}{|\boldsymbol{m}_3-\boldsymbol{m}_1|}\right\} \qquad \sigma_2 := \max\left\{\frac{|\boldsymbol{v}_2-\boldsymbol{v}_1|}{|\boldsymbol{v}_4-\boldsymbol{v}_3|}, \frac{|\boldsymbol{v}_4-\boldsymbol{v}_3|}{|\boldsymbol{v}_2-\boldsymbol{v}_1|}\right\}$$
$$\sigma_3 := \max\left\{\frac{|\boldsymbol{v}_3-\boldsymbol{v}_2|}{|\boldsymbol{v}_4-\boldsymbol{v}_1|}, \frac{|\boldsymbol{v}_4-\boldsymbol{v}_1|}{|\boldsymbol{v}_3-\boldsymbol{v}_2|}\right\} \qquad \sigma_4 := \max\left\{\frac{|\boldsymbol{v}_3-\boldsymbol{v}_1|}{|\boldsymbol{v}_4-\boldsymbol{v}_2|}, \frac{|\boldsymbol{v}_4-\boldsymbol{v}_2|}{|\boldsymbol{v}_3-\boldsymbol{v}_1|}\right\},$$

describe the ratio of lengths of the two lines connecting the edge midpoints (σ_1), the ratio of lengths of opposite edges (σ_2 and σ_3) and the ratio of lengths of the two element diagonals (σ_4). With these values we define the **element aspect ratio** as

$$\sigma := \max\{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}. \tag{2.5}$$

Figure 2.6a illustrates, why σ_1 is not sufficient to characterise the anisotropy of an element, and Figure 2.6b shows why σ_4 is necessary. The **(degree of) anisotropy of a mesh** is determined



(a) $\sigma_1 = \sigma_3 = \sigma_4 = 1, \sigma_2 = 3$. **(b)** $\sigma_1 = \sigma_2 = \sigma_3 = 1, \sigma_4 \approx 7.4$.

Figure 2.6: Two examples illustrating the calculation of the element aspect ratio.

by the maximal element aspect ratio occurring in the mesh.

The grid hierarchy is used to discretise continuous operators with Q_1 bilinear finite elements, resulting in a hierarchy of matrices and vectors corresponding to the bilinear and linear forms of the respective boundary value problem. For details about the finite element discretisation, see Section 3.2.

2.4 Minimal Overlap and Extended Dirichlet Boundary Conditions

The grid decomposition described in the previous section defines on each level a **domain decomposition with minimal overlap** in the sense that the only vertices that neighbouring subdomains share are the vertices on the common boundary. More precisely, let $\mathcal{V}^{(l)}$ be the index set of all grid vertices on level l, and $\mathcal{V}_i^{(l)}$ the indices of all vertices in $\bar{\Omega}_i^{(l)}$. Then, with $|\mathcal{V}^{(l)}| = n^{(l)}$ and $|\mathcal{V}_i^{(l)}| = n_{\mathrm{loc}}^{(l)}$, we define the **prolongation matrix** $\mathbf{P}_i^{(l)} \in \mathbb{R}^{n^{(l)} \times n_{\mathrm{loc}}^{(l)}}$. It takes the entries of

a local node vector $\mathbf{x}_i^{(l)}$ defined on $\bar{\Omega}_i^{(l)}$, distributes them to the corresponding positions of a global vector $\mathbf{x}^{(l)}$ and sets all remaining entries to zero, i. e.,

$$(\mathbf{x}^{(l)})_k = (\mathbf{P}_i^{(l)}\mathbf{x}_i^{(l)})_k := \begin{cases} (\mathbf{x}_i^{(l)})_{\mathrm{loc}_i^{(l)}(k)} & k \in \mathcal{V}_i^{(l)} \\ 0 & k \in \mathcal{V}^{(l)} \setminus \mathcal{V}_i^{(l)}, \end{cases}$$

where $\mathrm{loc}_i^{(l)}(k)$ is the local index within $\bar{\Omega}_i^{(l)}$ of the vertex with global index k. The corresponding **restriction matrix** $\mathbf{R}_i^{(l)} := (\mathbf{P}_i^{(l)})^{\mathsf{T}} \in \mathbb{R}^{n_{\mathrm{loc}}^{(l)} \times n^{(l)}}$ extracts the entries belonging to a subdomain $\bar{\Omega}_i^{(l)}$ from a global vector. Thus, we can define the matrix $\mathbf{A}_i^{(l)}$ that corresponds to the subdomain $\bar{\Omega}_i^{(l)}$ via

$$\mathbf{A}_i^{(l)} := \mathbf{R}_i^{(l)}\mathbf{A}^{(l)}\mathbf{P}_i^{(l)}, \tag{2.6}$$

i. e., $\mathbf{A}_i^{(l)} \in \mathbb{R}^{n_{\mathrm{loc}}^{(l)} \times n_{\mathrm{loc}}^{(l)}}$ is exactly the submatrix of $\mathbf{A}^{(l)}$ corresponding to the vertex indices $\mathcal{V}_i^{(l)}$. Since the matrix entries corresponding to the boundary vertices of the subdomain $\Omega_i^{(l)}$ are the sum of integrals over all surrounding elements, an overlap of exactly one element layer results. We say the local matrix contains *full entries on subdomain boundaries* and denote the (open) subdomain that results from extending $\Omega_i^{(l)}$ by this adjacent element layer (being part of the neighbouring subdomains) with $\hat{\Omega}_i^{(l)}$. Figure 2.7 illustrates this with the help of the unitsquare example from Section 2.3. This procedure allows for the following interpretation:

(a) level 2 grid $\bar{\Omega}^{(2)} = \bar{\Omega}_1^{(2)} \cup \ldots \cup \bar{\Omega}_9^{(2)}$. **(b)** Extended subdomain $\hat{\Omega}_5^{(2)}$ (grey).

Figure 2.7: Illustration of the concept of *full entries on subdomain boundaries*.

The integrals are computed over this extended subdomain $\hat{\Omega}_i^{(l)}$, Dirichlet boundary conditions are set on the boundary $\partial\hat{\Omega}_i^{(l)}$ of the extended subdomain, and all matrix rows and columns corresponding to the boundary vertices are eliminated. The matrix resulting from this process is exactly the local matrix $\mathbf{A}_i^{(l)}$. A direct conclusion of this interpretation is that $\mathbf{A}_i^{(l)}$ is regular. Thus, for the solution of the corresponding local linear systems no special boundary treatment is necessary and no pure Neumann problems have to be solved. This is a significant advantage over other domain decomposition approaches that need such special measures. On the other hand, special care has to be taken when applying local multigrid solvers which will be examined in Section 2.6.1.

For the following description of FEAST's solver concept, we usually speak of the subdomain $\Omega_i^{(l)}$ itself and not of the extended one $\hat{\Omega}_i^{(l)}$, which is just used to illustrate the concept of full entries on subdomain boundaries. However, to remind of this interpretation we will define the term **extended Dirichlet boundary conditions**, which is from now on used to describe the treatment of boundaries on inner subdomain boundaries. Consequently, a subdomain boundary exhibits *Dirichlet* or *Neumann boundary conditions*, if it lies on the geometric boundary of the domain, and *extended Dirichlet boundary conditions* when it is an interior boundary.

2.5 ScaRC: A Multilevel Domain Decomposition Method with Minimal Overlap

We now have all notations at hand to define FEAST's domain decomposition solver. As already mentioned, it is a *hybrid multilevel Schwarz method that works additively within levels and multiplicatively between levels*.

2.5.1 The Global Multiplicative Scheme

First we describe the multiplicative multilevel part of the overall solution algorithm. Since this is the outermost scheme, we introduce it as solver and not as preconditioner. We need to define grid operators that transfer information between levels. The **prolongation matrix** $\mathbf{P}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$ transfers a vector from grid level $l-1$ to level l via bilinear interpolation. The **restriction matrix** $\mathbf{R}^{(l-1)} \in \mathbb{R}^{n^{(l-1)} \times n^{(l)}}$ is defined as the adjoint operation, $\mathbf{R}^{(l-1)} := \left(\mathbf{P}^{(l)}\right)^{\mathsf{T}}$, and restricts a vector from level l to level $l-1$. So, the superscript always refers to the destination level of the operation. Let $\tilde{\mathbf{A}}^{(l)}$ denote a generic preconditioner for the level l matrix $\mathbf{A}^{(l)}$. This preconditioner will be described in more detail in Section 2.5.5. One iteration of the **multiplicative multilevel Schwarz solver** to solve the equation system $\mathbf{A}^{(L)}\mathbf{x}^{(L)} = \mathbf{b}^{(L)}$ consists of the following substeps:

$$\mathbf{x}^{(L)} \leftarrow \mathbf{x}^{(L)} + \tilde{\mathbf{A}}^{(L)}\left(\mathbf{b}^{(L)} - \mathbf{A}^{(L)}\mathbf{x}^{(L)}\right) \tag{2.7a}$$

For $l = L-1, \ldots, 2$:

$$\mathbf{b}^{(l)} \leftarrow \mathbf{R}^{(l)}\left(\mathbf{b}^{(l+1)} - \mathbf{A}^{(l+1)}\mathbf{x}^{(l+1)}\right), \quad \mathbf{x}^{(l)} \leftarrow \tilde{\mathbf{A}}^{(l)}\mathbf{b}^{(l)} \tag{2.7b}$$

$l = 1$:

$$\mathbf{b}^{(1)} \leftarrow \mathbf{R}^{(2)}\left(\mathbf{b}^{(2)} - \mathbf{A}^{(2)}\mathbf{x}^{(2)}\right), \qquad \mathbf{x}^{(1)} \leftarrow \left(\mathbf{A}^{(1)}\right)^{-1}\mathbf{b}^{(1)} \tag{2.7c}$$

For $l = 2, \ldots, L$:

$$\mathbf{x}^{(l)} \leftarrow \mathbf{x}^{(l)} + \mathbf{P}^{(l)}\mathbf{x}^{(l-1)}, \qquad \mathbf{x}^{(l)} \leftarrow \mathbf{x}^{(l)} + \tilde{\mathbf{A}}^{(l)}\left(\mathbf{b}^{(l)} - \mathbf{A}^{(l)}\mathbf{x}^{(l)}\right) \tag{2.7d}$$

So, in the first step (2.7a) the preconditioner $\tilde{\mathbf{A}}^{(L)}$ is applied to the defect on the finest level L. Then, in the restriction phase (2.7b) each substep consists of computing the current defect,

restricting it to the next coarser level and applying the generic preconditioner $\tilde{\mathbf{A}}^{(l)}$. This is repeated until the coarsest level is reached, where we do not apply the preconditioner $\tilde{\mathbf{A}}^{(1)}$ (which would also be possible), but solve the resulting system (see equation (2.7c)). If it is 'small enough' (e. g., less than 20000 unknowns), then we use the direct solver UMFPACK [55], otherwise some Krylov solver.[4] Then, in the prolongation phase (2.7d) each substep consists of prolongating the correction to the next finer level, updating the solution, computing the defect and applying the preconditioner $\tilde{\mathbf{A}}^{(l)}$ again. This is repeated until the finest level is reached. The whole process is iterated until a prescribed stopping criterion is met.

Scheme (2.7) resembles a classical multigrid approach. It can be seen as a *block-smoothed V-cycle multigrid*, where the block-smoother is given by $\tilde{\mathbf{A}}^{(l)}$. The question whether to interpret the method as multilevel Schwarz or as multigrid with a block-smoother is discussed in Section 2.5.3.

2.5.2 The Global Additive Preconditioner

The crucial task of the preconditioner $\tilde{\mathbf{A}}^{(l)}$ used within the multilevel solver (2.7) is to realise the connection between the global and the local layer. This is done via an additive Schwarz method which we want to express again in terms of a preconditioning operator $\tilde{\mathbf{A}}^{(l)}_{\mathrm{AS}}$. So, the preconditioner $\tilde{\mathbf{A}}^{(l)}_{\mathrm{AS}}$ does not equal the preconditioner $\tilde{\mathbf{A}}^{(l)}$, but is only a part of it. In Sections 2.5.5 and 2.6.3 it will become clear, why we make this distinction. We further denote with

$$\tilde{\mathbf{A}}_i^{(l)} \tag{2.8}$$

a local preconditioner for the local matrix $\mathbf{A}_i^{(l)}$ corresponding to subdomain $\Omega_i^{(l)}$, i. e., the operator $\tilde{\mathbf{A}}_i^{(l)}$ in some sense approximates the inverse $(\mathbf{A}_i^{(l)})^{-1}$. In Section 2.5.3 we will examine different possibilities how the operator can be realised. The **additive one-level Schwarz method with minimal overlap** is now defined as

$$\tilde{\mathbf{A}}^{(l)}_{\mathrm{AS}} := \widetilde{\sum_{i=1}^{M}} \mathbf{P}_i^{(l)} \tilde{\mathbf{A}}_i^{(l)} \mathbf{R}_i^{(l)}. \tag{2.9}$$

It takes a global (defect-)vector, restricts it to the i-th subdomain, applies the local preconditioner (2.8) and extends the resulting local correction to the global layer again. The special sum symbol $\widetilde{\sum}$ indicates that after summation values on subdomain boundaries have to be averaged, i. e., divided by the number of adjacent subdomains: Due to the minimal overlap concept described in the Section 2.4 the local solution vectors already contain full entries on the boundaries (see Kilian [94] for details).

The numerical efficiency of *one-level* (and also two-level) domain decomposition methods crucially depends on the size of the overlap. If it is chosen too small (especially if only minimal overlap is used), the convergence rates of such methods deteriorate significantly when a critical

[4] Note that the level 1 grid is the coarsest computational grid in the hierarchy (see Section 2.3).

parameter (refinement level, number of subdomains) increases [94, 150]. *Multilevel* domain decomposition methods, however, show robust convergence behaviour also in case of minimal overlap [94]. Hence, the usage of minimal overlap is justified, and for three essential reasons, FEAST is actually restricted to it:

1. Local submeshes preserve the tensor product structure.
2. High parallel efficiency can be achieved due to a minimal amount of data exchange.
3. Implementation and data structures are greatly simplified.

Consequently, FEAST's domain decomposition method strongly resembles a block-smoothed multigrid method, which is usually characterised by minimal overlap. However, in the next section another feature is identified that distinguishes the two solution approaches.

2.5.3 The Local Preconditioner

While the two previous sections explained the global structure of FEAST's solution algorithm, i. e., the transfer between the levels and the interaction of global and local layers, we will now take a closer look at the local operations. We want to distinguish between two main strategies how to realise the local preconditioner (2.8) within the additive Schwarz preconditioner (2.9). The choice of the strategy also determines whether the multiplicative scheme (2.7) is interpreted as multigrid method with a block-smoother or as multilevel Schwarz method.

2.5.3.1 First strategy: Apply one step of some elementary iterative scheme like Jacobi or Gauß-Seidel

The strategy to employ such elementary schemes within a multilevel method is well known from classical multigrid methods. Due to their property to rapidly dampen high frequencies in the error, they are called *smoothers* in this context. In standard multigrid schemes, these smoothers are applied to the entire matrix at once. To reduce the highly recursive character of this procedure, they can also be applied blockwise, i. e., simultaneously and independently to submatrices corresponding to local subdomains of a decomposed grid. This is exactly what is done within the additive preconditioner (2.9). Hence, when the local preconditioner (2.8) represents the application of a standard smoother like Jacobi, Gauß-Seidel, etc., then the overall method is usually not denoted as multilevel Schwarz method, but as *multigrid method with a block-smoother*. These methods have been studied in detail by Kilian [94].

In this thesis, we use two different block-smoothers, Jacobi and an alternating direction Gauß-Seidel like smoother called ADITriGS [2, 16, 94]. Omitting the superscript for the refinement level, we represent the local matrix $\mathbf{A}_i$ as decomposition of its three 'lower bands', three 'centre bands' and three 'upper bands',

$$\mathbf{A}_i = (\mathbf{L}_i^L + \mathbf{L}_i^C + \mathbf{L}_i^U) + (\mathbf{C}_i^L + \mathbf{C}_i^C + \mathbf{C}_i^U) + (\mathbf{U}_i^L + \mathbf{U}_i^C + \mathbf{U}_i^U),$$

each triple itself consisting of a lower, a centre and an upper single band. With this notation, the local Jacobi smoother is simply given by

$$\tilde{\mathbf{A}}_i^{\text{Jacobi}} := (\mathbf{C}_i^C)^{-1}, \tag{2.10}$$

i. e., its action consists of scaling a given vector with the inverted main diagonal C_C. A standard Gauß-Seidel smoother is correspondingly defined as

$$\tilde{\mathbf{A}}_i^{\text{GS}} := (\mathbf{L}_i^L + \mathbf{L}_i^C + \mathbf{L}_i^U + \mathbf{C}_i^L + \mathbf{C}_i^C)^{-1},$$

i. e., it takes the complete lower triangular matrix (including the main diagonal). An extension is the so called TriGS smoother that involves the complete triple of centre bands, i. e.,

$$\tilde{\mathbf{A}}_i^{\text{TriGS}} := (\mathbf{L}_i^L + \mathbf{L}_i^C + \mathbf{L}_i^U + \mathbf{C}_i^L + \mathbf{C}_i^C + \mathbf{C}_i^U)^{-1}.$$

In case of rowwise numbering the three centre diagonals stem from nodes coupling with themselves and with their left and right neighbours. Mesh or operator anisotropies 'in horizontal direction' are thus well caught, while this is not the case for anisotropies 'in vertical direction'. For this reason, we use a variant of the TriGS smoother in which we implicitly change the numbering of the local subdomain into a columnwise one. This *mirrored* TriGS smoother (MTriGS), is then able to deal with the anisotropies 'in vertical direction'. In realistic simulations, however, a mixture of all kinds of anisotropies is common, hence, we need a smoother that is able to robustly treat them all at the same time. A simple and successful strategy is to use TriGS and MTriGS in an alternating fashion, leading to the alternating direction Gauß-Seidel like smoother ADITriGS. This smoother has been shown to be extremely robust with respect to grid anisotropies [2, 16, 94]. Further significant advantages of this smoother are that it yields best results with a damping parameter of 1.0 and that it can be implemented in a very efficient, hardware-oriented manner (see Section 2.1.1). In Section 2.6 we examine advantages and disadvantages of Jacobi and ADITriGS as block-smoothers.

2.5.3.2 Second strategy: Apply some iterative or direct solution method to (approximately) solve the local system

One of the basic ideas of Schwarz methods is to replace the solution of one large global system by the solution of several smaller local systems. In practice this often means that the (eventually specialised) solver which usually treats the global system is then simply employed to solve the local systems, while the global coupling is provided by some iterative process, e. g., a Krylov method. Hence, when the local preconditioner (2.8) is realised by a 'full' solution method, then the overall algorithm (2.7) can be considered as *multilevel Schwarz method*.

We now discuss the choice of the local solver. It is intuitively clear that the convergence of the global iterative scheme depends – to some degree – on the accuracy the local systems are solved with. From this point of view, the best possible local solution method is a direct solver

that solves the system exactly (up to machine precision). In this case, we obtain a 'perfect' local preconditioner (2.8), which can be written in closed form as

$$\tilde{\mathbf{A}}_i^{\text{direct}} := (\mathbf{A}_i)^{-1}. \tag{2.11}$$

However, the computational costs and storage requirements of direct solvers increase rapidly with the local system size. Hence, there is a threshold where the application of direct solvers is too costly or simply not possible anymore. This threshold depends on different factors:

- The computational costs of direct solvers are usually dominated by the initial LU decomposition. Consequently, it makes a difference whether the LU decomposition has to be done once per linear solve (e. g., within a Newton iteration) or if several linear solves can be performed with one LU decomposition (e. g., within a time stepping scheme or a projection method).
- The size of the available memory physically limits the maximal problem size.
- Direct solvers cannot be applied at all when the system is not uniquely solvable, e. g., when it results from a pure Neumann problem.

So, there are situations where it is advantageous or even mandatory to switch to iterative methods to solve the local systems. One possibility is to employ a suitable Krylov method. However, for these methods it is well known that the convergence rate depends on the refinement level of the mesh. So, there might be a range in the problem size for which Krylov methods can be efficiently used and perform better than direct solvers, but beyond this range the performance will be unsatisfying. In this case, an iterative method has to be employed whose convergence rate does not deteriorate with increasing problem sizes, i. e., a multigrid method. *The strategy to use multigrid for solving the local systems within a global multilevel Schwarz method, is one of the core ideas of FEAST's solution concept.*

A crucial advantage of direct solution methods is the fact that they are 'black box' solvers: Independent of the physical problem or the quality of the mesh, they return the correct solution[5] of the local systems in constant time (i. e., only depending on the problem size), while the user does not have to adjust any parameter. When it is unavoidable to switch to an iterative solution method, then the user immediately has to deal with quite a number of parameters and settings, particularly in case of a multigrid method (stopping criterion, type of cycle, type of smoother, number of smoothing steps, coarse grid solver, etc.).

On the one hand this is clearly a disadvantage: Unsuitable settings might lead to bad convergence behaviour or even failure of the solution method; settings that are optimal for one problem configuration or in one part of the mesh, might be completely unsuited for the next problem configuration or in another part of the mesh. Furthermore, the user has to decide how accurately to solve the local systems in order to optimally balance local costs and number of global iterations and thus obtain minimal runtime of the overall algorithm. On the other hand

[5] Of course, accumulating round-off errors resulting from badly conditioned system matrices might deteriorate the quality of the solution.

this freedom of choice can also be seen as an advantage which will be explained in the next section.

As local multigrid smoothers we use the two elementary schemes, Jacobi and ADITriGS, that have been introduced as block-smoothers for the first strategy in Section 2.5.3.1. In Section 2.6.2 we describe a method that automatically switches between direct solvers and multigrid solvers.

2.5.4 The ScaRC Concept

In order to motivate FEAST's solver concept, we discuss advantages and disadvantages of the two strategies for local preconditioning, which have been introduced in the previous section, with respect to different aspects. For the second strategy we assume that the local systems are too large for the application of direct solvers and confine ourselves to the case of using multigrid as local solver. We can then characterise the difference between the first and the second strategy as follows: *Where the block-smoothed multigrid applies just one step of an elementary iterative scheme* (e. g., Jacobi), *the multilevel Schwarz method uses a complete multigrid solver with this elementary scheme as smoother* (also compare Listings 2.4 and 2.5 in Section 2.5.5). The critical revision of FEAST's solution concept in Section 2.6 will focus on this difference, so we examine it here in detail.

Storage requirements At first glance the difference in terms of storage requirements may appear significant since for block-smoothed multigrid just an elementary smoother (acting on one level) has to be prepared for each subdomain, while for multilevel Schwarz entire local multigrid solvers (acting on several levels) have to be provided. However, the level hierarchy of matrices and vectors needed for local multigrid has been allocated anyway in order to realise the global multilevel method: As described in Section 2.1.1, there exist no global matrices and vectors. Instead, global operations and objects are represented by local ones, combined with according data exchange operations. These local data objects can be reused – with slight modifications [94] – for the local multigrid operations. So, the only additional storage requirement – compared to the first strategy – are the auxiliary vectors needed for the local multigrid scheme. Hence, when assuming the same elementary iterative scheme as (block-)smoother, then the storage requirements for the second strategy are only slightly higher than for the first strategy.

Arithmetic costs Where the first strategy only does one step of an elementary iterative scheme, the second strategy performs one or more entire multigrid cycles including smoothing, defect calculations, grid transfer operations and eventually norm calculations on each level. So, it is clear that the arithmetic costs per outer iteration are significantly higher for the second strategy.

Communication vs. computation From the point of view of parallel execution, the previous point can be formulated in a positive way: Each application of the local preconditioner is

followed by a global defect computation which involves communication between neighbouring subdomains. Hence, the ratio between communication and floating point operations is much more favourable for the second strategy. This, however, can only turn into an advantage when the second strategy is able to significantly reduce the number of global iterations.

Flexibility For both strategies, it is possible to select different smoothers on different subdomains. For example, one can use Jacobi for 'easy subdomains' and ADITriGS for 'difficult' ones. The second strategy offers the additional possibility to determine for each subdomain how accurately to solve the local system. Furthermore, one can potentially adjust all the other multigrid parameters per subdomain to adapt to the local situation. In practice, however, this is useless as long as the user has to perform these settings manually. So, this additional flexibility can only turn into actual gain of performance or robustness when it is accompanied by some 'automatic control system'.

'Black box' character The higher number of parameters in case of the second strategy and the resulting increase of flexibility automatically means to abandon the 'black box' character of the solution method to some degree. Each additional parameter the user has to adjust bears the risk of deteriorated solver performance due to unsuitable settings. Hence, as long as these additional parameters are not set automatically (also see the previous point), the first strategy is more favourable in this regard as there are simply less parameters to set.

Influence on the global iteration A crucial disadvantage of the first strategy is that irregularities in only one subdomain can influence the convergence behaviour of the overall algorithm. The block-smoothed multigrid performs the same number of smoothing steps in each subdomain. While specific features of 'easy subdomains' may be 'sufficiently resolved', this does not have to be the case for 'difficult subdomains' that exhibit irregularities in the mesh or in the operator. Hence, the global scheme has to iterate until the error in these 'difficult subdomains' is sufficiently reduced, as well. Thereby, it does not play a role whether the fraction of 'difficult subdomains' is close to 5% or close to 95%.

With the second strategy, however, one has the possibility to reduce the residual norm of the local problem by a certain factor (e. g., 'gain one digit'). The specific features of the different subdomains can thus be resolved in a homogeneous way such that from the point of view of the outer iterative scheme all subdomains return similar error reductions. Consequently, the global convergence rate does not suffer from local irregularities.

Load balancing In case of the first strategy the same amount of work is performed on each subdomain. Load balancing can thus be realised by 'simply counting subdomains', i. e., trying to assign an equal number of subdomains to each processor (which is – depending on the topology of the domain – already difficult enough under the condition that the set of subdomains residing on the same processor should be connected in order to minimise communication). The flexibility of the second strategy to reduce the residual norm of the local problems by a certain factor instead of performing a fixed number of iterations,

however, prevents this simple way of load balancing. The solution of one 'difficult' local problem potentially requires more work than solving several 'easy' local problems. It is extremely complicated – if not impossible – to reasonably estimate the required work a priori and correspondingly distribute subdomains to processors. This is especially true when the 'difficulty' of the local problem does not result from local mesh irregularities (which would still be predictable somehow), but, for example, from localised operator nonlinearities (which are not readily assessable). Hence, significant conceptual and implementational work is necessary to develop an adequate load balancer that facilitates a satisfactory parallel efficiency for the second strategy (cf. Section 2.7.4).

Employment of co-processors When using co-processors to outsource parts of the arithmetic work (see Göddeke et al. [73, 75] and references therein) this always means that some amount of data has to be transferred to the co-processor (e. g., the current defect vector) and some data has to be returned to the main processor (e. g., the local correction). These transfer operations – eventually together with changing the precision of the data (from single to double or vice versa) – are much slower compared to the actual computation and thus usually constitute a severe bottleneck. Consequently, the more work is performed on the co-processor with the given data (i. e., the higher the arithmetic intensity is), the higher the benefit of the overall procedure will be. In this regard, there is a significant advantage for the second strategy: Performing one or several complete multigrid cycles clearly justifies the expensive movement of data to the co-processors. On the other hand, when performing just one step of an elementary iterative scheme as in the first strategy, the time for moving data will likely dominate the overall runtime.

Table 2.1 summarises the advantages and disadvantages of the two local preconditioning strategies. From this table it becomes clear that it is not possible to give a global answer to the

Storage requirements	1	○●○○ 2
Arithmetic costs	1	●○○○ 2
Communication vs. computation	1	○○○● 2
Flexibility	1	○○●○ 2
'Black box'character	1	○●○○ 2
Influence on global iteration	1	○○○● 2
Load balancing	1	●○○○ 2
Co-processors	1	○○○● 2

Table 2.1: Comparison of the two strategies for local preconditioning. The position of the black dot indicates how strong the first strategy (left) or the second strategy (right) is favoured.

question which of the two preconditioning strategies should be preferred. The answer depends on several factors like regularity and homogeneity of the computational mesh or amount and type of compute resources. Since these factors change from simulation to simulation, or even within one computation, the solution method must be able to dynamically adapt to the current situation. This is the core idea of FEAST's solution concept ScaRC. 'Sca' stands for 'scal-

able'[6] , which means that *solver components can be adjusted independently of each other and thus optimised with respect to different aspects of the current computation*. This especially involves that ScaRC does not represent explicitly one of the two concepts discussed above, but it comprises both: It can switch between the two strategies or even use both at the same time in different parts of the domain. This means on the one hand that ScaRC can reduce to a standard block-smoothed multigrid scheme, when the configuration allows this. On the other hand, ScaRC can represent a highly diversified combination of different local preconditioners, potentially ranging from an elementary Jacobi iteration to direct solvers to multigrid schemes which are smoothed by ADITriGS and achieve different accuracies.

In this sense, ScaRC can be interpreted as generalisation of the two concepts 'block-smoothed multigrid' and 'multilevel domain decomposition'. From now on, we want to distinguish the two concepts within ScaRC as following: The multilevel domain decomposition concept is characterised by the fact that on subdomains 'full' solvers are used to treat the local problems. In this case, ScaRC consists of two nested (iterative) solution schemes – on the one hand the outer multilevel algorithm (2.7) solving the global problem, on the other hand an inner iterative multigrid scheme (or a direct solver) for the local subdomain problems. Hence, the algorithm starts solvers on two different hierarchical layers which we simply call the **global layer** and the **local layer**. The restriction and prolongation matrices $\mathbf{R}_i$ and $\mathbf{P}_i$ introduced in Section 2.4 formally 'toggle' between these two layers. The overall algorithm is then referred to as **2-layer-ScaRC**. Correspondingly, when the solver exclusively applies the first strategy for local preconditioning (i. e., it reduces to a block-smoothed multigrid) then we denote it as **1-layer-ScaRC**.

The advantages of the generalised ScaRC approach can be described as follows (also see Becker [16], Kilian [94]):

- Due to the multilevel ansatz the convergence rates are independent of the refinement level, and for meshes with only mild macro anisotropies they are also independent of the number of subdomains.
- The domain decomposition approach allows for the possibility to define the local preconditioners completely independent of each other: The additive Schwarz preconditioner (2.9) simply 'collects and combines' the local corrections, no matter how these have been computed.
- Both 1-layer- and 2-layer-ScaRC can choose the elementary smoother corresponding to the local situation. For instance, Jacobi is sufficient on isotropic submeshes, while ADITriGS can be used for anisotropic ones. When a subdomain does not fulfil the tensor product property, then some general smoother like ILU can be applied (which is not done in this thesis).
- Local irregularities in the mesh or in the operator can be 'hidden' from the outer iterative scheme when applying the 2-layer-ScaRC strategy: Reducing the local residuum norms

[6] For an explanation of the acronym 'RC' see Section 2.7.2.

by a certain factor (e. g., 'gain one digit') lets the subdomains appear 'equally difficult' to the outer solver. Hence, the number of global iterations is not affected by (increasing) local irregularities and can be held constant, thus resulting in a high numerical efficiency of the overall algorithm. This is especially important in case the mesh is adaptively – and eventually anisotropically (cf. Section 2.3) – refined in the course of the simulation.

- If such irregularities are strongly localised (e. g., they appear in only one subdomain), then ScaRC can be configured to treat the uncritical subdomains with a simple local preconditioner. So, in the extreme case the local preconditioner on a 'difficult subdomain' is a multigrid smoothed by ADITriGS gaining one digit, and on an 'easy subdomain' merely one single Jacobi step. Several of these 'easy subdomains' can then be treated in the same time as one 'difficult subdomain'. Applying the complex preconditioner to *all* subdomains might eventually lead to a slightly better global convergence, though, but it will likely be much too expensive in terms of arithmetic work and storage requirements. Hence, by applying strong local preconditioners *only where necessary* the ScaRC ansatz bears the potential to optimally balance convergence behaviour and computational costs.

- Due to the tensor product property of the subdomains, local operations can exploit the cache-optimised SparseBandedBLAS library (see Section 2.1.1) and thus achieve high processor efficiency in terms of MFLOP/s rates.

- As a consequence of the previous points, ScaRC is able to achieve a high degree of data locality and a good ratio between computation and communication, overall resulting in a high parallel efficiency.

This list of advantages shows that ScaRC indeed has the potential to successfully combine sophisticated solution strategies with high performance computing techniques and to thus achieve both numerical and hardware efficiency at the same time. Kilian [94] and Becker [16] confirm this with the help of extensive numerical studies. However, ScaRC also (still) exhibits some disadvantages:

- The ratio between computation and communication necessarily deteriorates when performing operations on smaller refinement levels. This, however, is inherent to the global multilevel approach – which is indispensable from a numerical point of view – and can thus not be avoided.

- The additive domain decomposition approach shows the typical block-Jacobi property to be very sensitive with respect to macro anisotropies. The solver especially loses its independency of number of subdomains when too large macro anisotropies are present. This problem can partially be resolved by combining the multilevel scheme with a Krylov solver (see Becker [16], Kilian [94] and Section 2.6.3). Another promising approach is to add a third intermediate layer ('3-layer-ScaRC') and/or to merge subdomains (see Becker [16] and Section 2.7.2).

- The local multigrid solvers within the 2-layer-ScaRC scheme are *nonconforming* and thus require special treatment. In Section 2.6.1 we describe this problem in detail and present a feasible solution.
- Due to the complexity of the ScaRC solvers – especially of the 2-layer variant – there are many parameters to adjust. Especially the two damping parameters are critical: They decisively influence the convergence behaviour and can even cause divergence if not properly set. In Section 2.6.3 we describe how to set these parameters automatically and to thus considerably increase the robustness of the overall solution method. We try to identify configurations that could serve as a sort of 'black box' solver.
- ScaRC's ability to adapt solver components to locally varying problem complexities is only useful when this adaptation happens automatically. First steps in this direction are taken by Becker [16]. We will further discuss this topic in Section 2.7.3.
- The strategy to let the local solvers 'gain one digit' makes load balancing in a parallel computation extremely challenging. This problem is discussed in Section 2.7.4.

2.5.5 Algorithmic Formulation

In this section we try to give a concise algorithmic description of the ScaRC solvers which is aggravated by their complexity and versatility. To ease this task we will consider the global multilevel scheme (2.7) on page 30 as *multigrid* solver (and not as multilevel Schwarz method) – regardless of which strategy is used for local preconditioning. In case of 2-layer-ScaRC we will then speak of the *global multigrid* and *local multigrid* to refer to the outer and the inner multigrid solver, respectively. This enables us to use the same algorithmic description for both solvers. For the sake of clarity we split the overall algorithm into different subroutines.[7] The following listings use Fortran-like pseudo code, which should be comprehensible for non-Fortran-programmers, too.

We begin with the outer multigrid scheme, which is represented in a non-recursive form in Listing 2.1. Also compare the definition of the multiplicative multilevel Schwarz solver (2.7) in Section 2.5.1. For simplicity, we only show here the V-cycle; for a schematic representation of F- and W-cycles, see Figure 2.10 on page 48. The difference to standard multigrid implementations is the additional variable Λ that determines the layer the solver acts on. This can be the local layer ($\Lambda =$ `'LOCAL'`), i. e., one subdomain, or the global one ($\Lambda =$ `'GLOBAL'`), i. e., the whole domain (cf. Section 2.5.4). The choice of the local preconditioner T, which is passed to the multigrid algorithm, to some degree depends on the layer, i. e., not every combination of layer and preconditioner makes sense (cf. Listing 2.3). For simplicity, we assume that on all subdomains the same preconditioner T is used. The input variables $\mathbf{A}, \mathbf{b}$ and $\mathbf{x}$ denote the entire hierarchy of matrices and vectors. The input variable L (or l) then determines which level

[7] The presentation resembles the actual implementation in FEAST: There is only one multigrid routine to realise both the global and the local variant, and the nested iterative solution schemes are represented by distinct, recursively called subroutines.

```
1  subroutine MG(A, b, x, Λ, L, T)
2    !Input:       matrix A, RHS b, start vector x, layer Λ,
3    !             max. level L, type of local preconditioner T
4    !Output:      solution x
5    !Parameters:  tolerance ε, maximal number of iterations K,
6    !             number of smoothing steps K_pre, K_post
7
8    δ ← ||b^(L) − A^(L) x^(L)||                  !norm of initial residuum
9    do k = 1,...,K                               !main loop
10     do l = L,...,2                             !restriction loop
11       call RICH(A, b, x, Λ, l, K_pre, T)       !presmoothing
12       b^(l−1) ← R^(l−1) (b^(l) − A^(l) x^(l))
13       x^(l−1) ← 0
14     enddo
15     x^(1) ← (A^(1))^(−1) b^(1)                 !solve coarse grid problem
16     do l = 2,...,L                             !prolongation loop
17       x^(l) ← x^(l) + P^(l) x^(l−1)
18       call RICH(A, b, x, Λ, l, K_post, T)      !postsmoothing
19     enddo
20     if (||b^(L) − A^(L) x^(L)|| ⩽ δε) then     !convergence control
21       exit
22     endif
23   enddo
24 end subroutine MG
```

Listing 2.1: Multigrid (resp., multilevel domain decomposition) algorithm working on global or local layer. The smoother is realised by a Richardson iteration which is described in detail in Listing 2.2.

to choose from the hierarchy. Some of the variables that are necessary for the definition of a multigrid solver are declared as 'parameters' in the subroutine description. This is only to keep the routine interface short – the variables could just as well be declared as input variables.

```
1  subroutine RICH(A, b, x, Λ, l, K, T)
2    !Input:       matrix A, RHS b, start vector x, layer Λ, current level l,
3    !             number of iterations K, type of local preconditioner T
4    !Output:      solution x
5    !Parameters:  damping parameter ω
6
7    do k = 1,2,...,K              !main loop
8      c^(l) ← 0
9      r^(l) ← b^(l) − A^(l) x^(l)  !current defect
10     call PRECON(A, r, c, Λ, l, T) !preconditioning
11     x^(l) ← x^(l) + ωc           !damped update of the solution
12   enddo
13 end subroutine RICH
```

Listing 2.2: Preconditioned Richardson scheme performing a fixed number of iterations. This algorithm realises the smoother within the multigrid implementation in Listing 2.1.

The pre- and postsmoothing process within the multigrid scheme is realised by a preconditioned Richardson iteration which is described in Listing 2.2. In Section 2.6.3 it will become clear why we use this rather unusual representation for smoothing. The number of smoothing steps in the multigrid algorithm ($K_{\text{pre}}, K_{\text{post}}$) defines the number of Richardson iterations. The two most important ingredients of the Richardson scheme are the damping parameter ω and the preconditioner (represented by a call of the routine defined in Listing 2.3). They are linked in the sense that the choice of a 'good' damping parameter crucially depends on the choice of the preconditioner.

```
1  subroutine PRECON(A,b,x,Λ,l,T)
2    !Input:       matrix A, RHS b, start vector x=0, layer Λ, current level l,
3    !             type of local preconditioner T
4    !Output:      solution x
5
6    if (Λ .eq. "GLOBAL") then
7      do i=1,2,...,M                          !add. Schwarz loop over subdomains
8        b_i^(l) ← R_i^(l) b^(l)               !restrict defect to i-th subdomain
9        call PRECON(A_i,b_i,x_i,"LOCAL",l,T)  !local precond. on i-th  subdomain
10     enddo
11     x^(l) ← Σ̃_{i=1}^M P_i^(l) x_i^(l)       !combine local corrections
12   else if (Λ .eq. "LOCAL") then
13     if (T .eq. "JACOBI") then               !1-layer-ScaRC
14       x^(l) ← (Ã^Jacobi)^(l) b^(l)          !apply Jacobi preconditioner
15     else if (T .eq. "MG-JACOBI") then       !2-layer-ScaRC
16       call MG(A,b,x,"LOCAL",l,"JACOBI")     !apply local multigrid
17     else if (T .eq. "DIRECT") then          !2-layer-ScaRC
18       x^(l) ← (Ã^direct)^(l) b^(l)          !apply local direct solver
19     endif
20   endif
21 end subroutine PRECON
```

Listing 2.3: Generic preconditioning routine that realises the additive Schwarz preconditioner and all local preconditioners/smoothers. (Jacobi is used representatively.)

The algorithm depicted in Listing 2.3 represents the core preconditioning routine within ScaRC. It is used for both local and global preconditioning. On the one hand, when called on the subdomain layer, it implements the three types of local preconditioners described in Section 2.5.3:

1. application of one Jacobi step $\tilde{\mathbf{A}}^{\text{Jacobi}}$ (see equation (2.10) on page 33)
2. call of a local multigrid solver smoothed by Jacobi
3. call of a direct solver $\tilde{\mathbf{A}}^{\text{direct}}$ (see equation (2.11) on page 34)

Jacobi is used here only as representative for all the other elementary smoothers. On the other hand, when called on the global layer, it realises the additive Schwarz preconditioner (2.9) on page 31 which establishes the connection between the two layers. For each subdomain, the routine recursively calls itself and applies the corresponding local preconditioner. After

the subdomain loop the local corrections are summarised as described in Section 2.5.2. So, this routine is responsible to algorithmically distinguish between 1-layer-ScaRC and 2-layer-ScaRC solvers. Of course, not every parameter combination makes sense. For example, there seems to be no point in smoothing a local multigrid scheme with another local multigrid (which would be possible from the implementational point of view).

We now briefly turn back to the definition of the multiplicative multilevel Schwarz solver (2.7) in Section 2.5.1. There, we introduce the generic preconditioning operator $\tilde{\mathbf{A}}^{(l)}$ that 'contains', but is not equal to the additive Schwarz preconditioner (2.9) defined in Section 2.5.2. The reason why we make this distinction is clear now: The preconditioner $\tilde{\mathbf{A}}^{(l)}$ is represented by the Richardson iteration (see Listing 2.1, lines 11 and 18, and Listing 2.2) which enables us to apply several smoothing steps. The additive Schwarz preconditioner is called only within this iteration, where the operator $\tilde{\mathbf{A}}^{(l)}_{\mathrm{AS}}$ is represented by lines 7–11 of the algorithm in Listing 2.3.

2.5.5.1 The ScaRC File Format

In FEAST, the ScaRC solvers are defined in an ASCII file format. We use it in the following sections to describe the different solvers we are working with. The format naturally imitates the recursive structure of the ScaRC solvers and is thus intuitively comprehensible. For example, the ScaRC file in Listing 2.4 describes a 1-layer-ScaRC solver performing at most 128 iterations, stopping the solution process when the initial residuum is reduced by a factor of 10^{-6} ('gain 6 digits'), using a V-cycle with 2 pre- and 2 postsmoothing steps. The smoother is

```
1 solver=MG, maxiter=128, tol=REL:1e-6, cycle=V:2:2
2 smoother=JACOBI, damp=0.7
3 coarse=UMFPACK
```

Listing 2.4: Typical 1-layer-ScaRC solver (operations on the local layer highlighted in grey).

Jacobi with a damping parameter of $\omega = 0.7$. Note that in all descriptions of ScaRC files those portions of the algorithm that work on the local layer are highlighted in grey.[8] UMFPACK is used for solving the global coarse grid problem.

A typical 2-layer-ScaRC solver can be seen in Listing 2.5. It replaces the elementary smoother

```
1 solver=MG, maxiter=128, tol=REL:1e-6, cycle=V:2:2
2 smoother=LOCAL, damp=0.8
3   solver=MG, maxiter=32, tol=REL:1e-1, cycle=F:1:1
4   smoother=JACOBI, damp=0.7
5   coarse=UMFPACK
6 coarse=UMFPACK
```

Listing 2.5: Typical 2-layer-ScaRC solver (operations on the local layer highlighted in grey).

[8] In FEAST, *all* elementary iterative schemes like Jacobi, Gauß-Seidel, etc. are working block-wise. So, when such a scheme is used in ScaRC files, there is no need to explicitly mention this.

by an indirect one acting on the local layer (‘smoother=LOCAL’) damped with $\omega = 0.8$. This indirect smoother is realised by a local multigrid scheme, performing at most 32 iterations, gaining 1 digit and applying a F-cycle with 1 pre- and 1 postsmoothing step of Jacobi damped with $\omega = 0.7$. UMFPACK is employed for both the local and the global coarse grid problem. Note, that the definition of the local solver (lines 3-5) formally equals that of the 1-layer-ScaRC solver in Listing 2.4. When the local problems are to be solved by direct solvers, the 2-layer-ScaRC has to be modified as shown in Listing 2.6.

```
1 solver=MG, maxiter=128, tol=REL:1e-6, cycle=V:2:2
2 smoother=LOCAL, damp=0.8
3   solver=UMFPACK
4 coarse=UMFPACK
```

Listing 2.6: 2-layer-ScaRC scheme with UMFPACK solving the local systems (highlighted in grey).

2.5.5.2 Short Notation of ScaRC Solvers

We now introduce a short notation that will be used in the following sections to identify and distinguish different ScaRC solvers within the running text without being forced to always provide or refer to a detailed ScaRC file description. The short notation adheres to the following rules:

- As already mentioned in the previous section, we refer to the global multilevel algorithm simply as *global* or *outer multigrid*. So, we can use the shortcut ‘MG’ which also applies to the local multigrid schemes.
- The shortcut ‘MG’ is followed by the indication of the smoother and the coarse grid solver.
- The elementary iteration schemes Jacobi and ADITriGS are denoted with ‘JAC’ and ‘ADI’, respectively. By definition, they act on the local layer.
- The application of a direct solver is denoted with a ‘D’.
- Switching between local and global layer is denoted with a double underscore ‘__’.
- Solver components on the same layer are separated by a single hyphen ‘-’.
- Optionally, solving parameters can be added in parentheses, separated by commas:
  - number of iterations (‘2’)
  - stopping criterion (‘1e-6’)
  - damping parameter (‘0.7’)
  - multigrid cycle and pre- and postsmoothing steps (‘V11’)

Note that the number of iterations, stopping criterion and damping parameter are always written in integer, exponent and decimal notation, respectively, as to uniquely distinguish them.

- Singlegrid (Krylov) solvers (see Section 2.6.3) are abbreviated as follows:
  - Richardson: 'RICH'
  - Conjugate Gradients: 'CG'
  - Stabilised Bi-conjugate gradients (BiCGstab): 'BICG'
  - (Flexible) Generalised Minimal Residuals (GMRes, FGMRes) without restart parameter: 'GMRES', 'FGMRES'
  - GMRes, FGMRes with restart parameter 20: 'GMRES20', 'FGMRES20'

The ScaRC solvers in Listings 2.4, 2.5 and 2.6 thus read in minimalistic short notation:

```
MG__JAC__D
MG__MG-JAC-D__D
MG__D__D
```

In a more verbose short notation they exemplarily read:

```
MG(128,V22,0.7)__JAC__D
MG(1e-6,0.8)__MG(F11,1e-1,0.7)-JAC-D__D
MG(0.8)__D__D
```

2.6 ScaRC Revisited: Improvements and a Critical Review

Kilian [94] introduces the 2-layer-ScaRC approach and generally demonstrates its efficiency by means of extensive numerical studies. Becker [16] realises the '**R**ecursive **C**lustering' aspect (see Section 2.7.2) and examines the entire solution strategy from a hardware-oriented point of view. In this thesis we want to concentrate on three different aspects:

1. In Sections 2.6.1 and 2.6.2 we take a closer look at the local multigrid solver.
2. In Section 2.6.3 we present possibilities to increase the robustness of ScaRC.
3. In Section 2.6.4 we critically compare 1-layer-ScaRC and 2-layer-ScaRC.

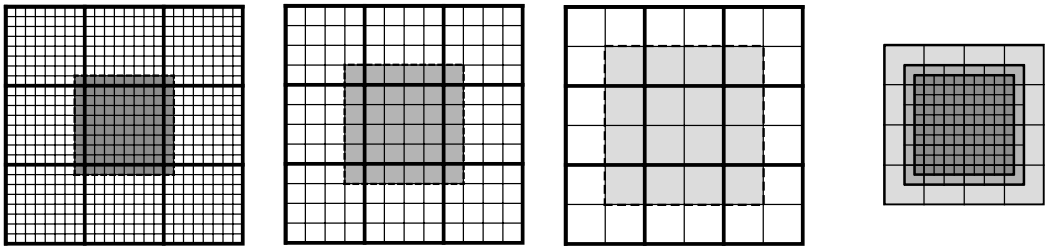
In view of the immense arithmetic costs of employing multigrid schemes on the local layer, we especially want to discuss the question if and when the 2-layer-ScaRC approach is actually beneficial.

2.6.1 A Modified Multigrid Method for Solving Local Subdomain Problems

When in a 2-layer-ScaRC solver the local subproblems become too large for the application of a direct solution method (cf. Section 2.5.3.2), then a multigrid solver has to be employed. It acts on a hierarchy of local meshes based on the partition of the global mesh hierarchy (cf. Section 2.3). Due to the regular tensor product property of the local submeshes, all multigrid components can be implemented in a very hardware-efficient way (cf. Sections 2.1.1 and 2.5.3). So, at first glance we seem to have perfect conditions at hand to apply conventional, yet highly efficient multigrid solvers. Unfortunately, this is not the case as we will illustrate now with the help of an example.

2.6.1.1 The Fundamental Problem

We consider the unitsquare introduced in Figure 2.5 on page 26 and assume that the finest grid level is $L = 3$. We take a look at the central subdomain $\bar{\Omega}_1^{(5)}$. Due to the concepts of minimal overlap and extended Dirichlet boundary conditions described in Sections 2.4 and 2.5.2, the local subproblem to be solved is actually defined over the *extended* subdomain $\hat{\Omega}_5^{(3)}$ (see Figure 2.8a). When we want to apply a multigrid scheme to solve this problem, we need grids



(a) level 3 grid $\bar{\Omega}^{(3)}$; extend. subdomain $\hat{\Omega}_5^{(3)}$. **(b)** level 2 grid $\bar{\Omega}^{(2)}$; extend. subdomain $\hat{\Omega}_5^{(2)}$. **(c)** level 1 grid $\bar{\Omega}^{(1)}$; extend. subdomain $\hat{\Omega}_5^{(1)}$. **(d)** Nonconforming grids $\hat{\Omega}_5^{(1)}, \hat{\Omega}_5^{(2)}, \hat{\Omega}_5^{(3)}$.

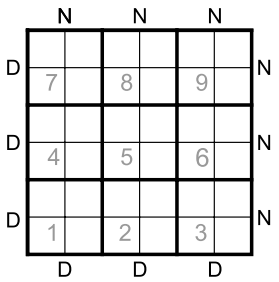
Figure 2.8: Example grid in three refinement levels with extended subdomains highlighted.

on level 2 and level 1. These are naturally given by the corresponding *extended* subdomains $\hat{\Omega}_5^{(2)}$ and $\hat{\Omega}_5^{(1)}$ (see Figures 2.8b and 2.8c). Displaying the three extended subdomains on top of each other as in Figure 2.8d reveals the basic problem: *The size of the computational domain increases as the grid level decreases.* Consequently, we are faced with a *nonconforming multigrid method.*

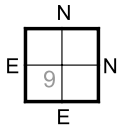
The basic procedure of multigrid methods is to smooth the defect, restrict it to a coarser grid, compute a defect correction there, and prolongate it back to update the iteration vector. The problem is now, that this coarse grid correction is actually based on a different configuration since it is computed on a larger domain. Hence, the coarse grid correction cannot be optimal and might even be harmful for the convergence process. The higher the number of levels in the

grid hierarchy, the greater the difference in size between the domain on the finest grid and that on the coarsest grid will be. Consequently, we have to expect that *the convergence behaviour of the multigrid solver is not level independent*, i. e., one of the most important properties of typical multigrid methods is lost. Furthermore, standard multigrid theory does not fully apply such that convergence cannot be guaranteed.

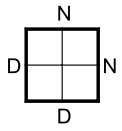
We will use the example grid of Figure 2.8 to show that these are not only theoretical considerations, but that they indeed cause severe problems in practice. We solve the standard Poisson equation, discretised by Q_1 bilinear finite elements, with constant right hand side, $-\Delta u = 1$, applying zero Dirichlet conditions on the left and the bottom boundary and zero Neumann boundary conditions on the remaining two boundaries (see Figure 2.9a). We focus on the upper



(a) Geometric Neumann ('N') and Dirichlet ('D') boundary conditions.



(b) Top right subdomain with two Neumann and two extended Dirichlet ('E') boundaries.



(c) Reference configuration with two Neumann and two (standard) Dirichlet boundaries.

Figure 2.9: Test configurations.

right subdomain Ω_9, which consequently exhibits two boundaries with zero Neumann conditions and two boundaries with extended Dirichlet conditions (see Figure 2.9b). In order to assess the negative influence of the extended Dirichlet boundary conditions, we will use a reference configuration for comparison in which these extended Dirichlet boundary conditions are replaced by standard zero Dirichlet boundary conditions (see Figure 2.9c).

We use the solver depicted in Listing 2.7 to solve the local problems. The outer Richardson scheme is just a 'dummy' needed to start the local multigrid solver on each subdomain. We also consider F-cycles (`MG(64,1e-6,F22,0.7)-JAC-D`) and W-cycles (`MG(64,1e-6,W22,0.7)-JAC-D`) (see Figure 2.10).

```
1 solver=RICH, maxiter=1, tol=IGNORE
2 prec=LOCAL, damp=1.0
3   solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:2:2
4   smoother=JACOBI, damp=0.7
5   coarse=UMFPACK
```

Listing 2.7: Local solver `MG(64,1e-6,V22,0.7)-JAC-D` (highlighted in grey) as part of a 'dummy' global Richardson scheme.

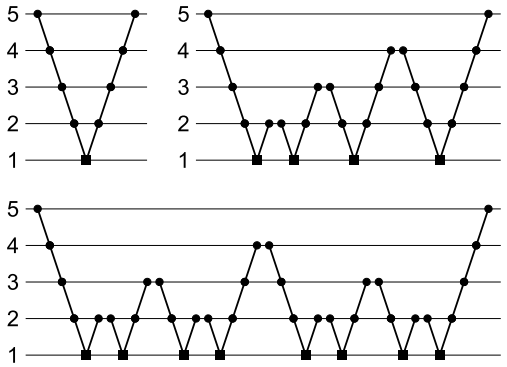


Figure 2.10: Schematic representations of V-, F- and W-cycle multigrid over 5 levels. Black dots indicate smoothing, while black squares stand for coarse grid solving.

To show the influence of the increasing domain size on coarser levels, we fix the maximal grid level and vary the minimal level (on which the direct coarse grid solver is applied). We exemplarily show the convergence rates of the local multigrid solvers for maximal grid level 8 and minimal level varying between 7 (2-grid method) and 1 (8-grid method). The convergence rate is computed as

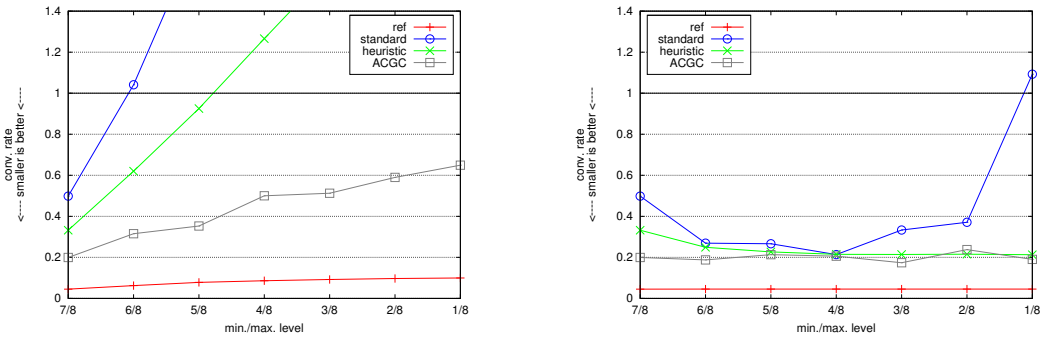
$$c := \left(\frac{\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|}\right)^{1/k}, \tag{2.12}$$

where $\|\mathbf{r}_0\|$ is the norm of the initial residual and $\|\mathbf{r}_k\|$ is the residual norm after the k-th iteration. If a method diverges ($c > 1.0$), we display the convergence rate after the last iteration, where we cut off the graphs appropriately. Note that the y-axis (denoting the convergence rate) of the following graphs is varied in order to better recognise and compare the features of the different methods.

First of all, the results obtained on the reference configuration (see the graphs labelled 'ref' in Figure 2.11) show that using 2 pre- and postsmoothing steps with Jacobi damped by $\omega = 0.7$ is suitable – for both V- and F-cycle the convergence rates are below $c = 0.1$. Exchanging the Dirichlet boundary conditions (Figure 2.9c) with extended Dirichlet boundary conditions (Figure 2.9b) has dramatic consequences: The graph labelled 'standard' in Figure 2.11a shows that the standard V-cycle 2-grid method still converges with a rate of about $c = 0.5$, though, but as soon as 3 or more grid levels are used the method *diverges*. The slope of the curve clearly shows that the divergence gets worse the more grid levels are used. Using the F-cycle we obtain convergence at least up to 7 grid levels (minimal level 2), where the rates are already quite bad in comparison to the reference computation. For 8 grid levels, however, also the F-cycle fails (see Figure 2.11b).[9]

This simple example already shows that *standard multigrid techniques are not suited to solve subproblems with extended Dirichlet boundary conditions*. Until now, FEAST applied some heuristic to circumvent this deficiency of the local multigrid schemes: In grid transfer operations between grid levels, the values corresponding to subdomain boundaries were treated in

[9] The graphs labelled 'ACGC' are explained in Section 2.6.1.3.

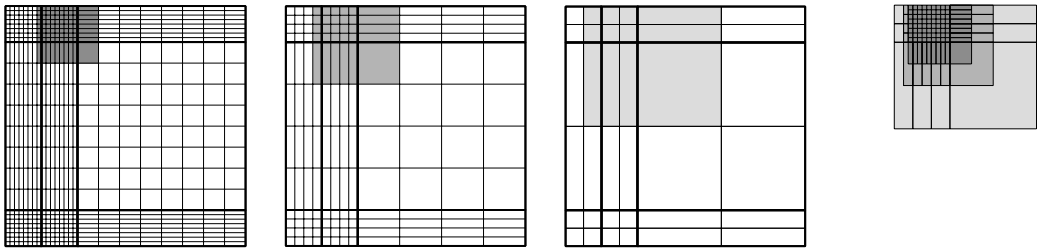


(a) V-cycle . **(b)** F-cycle .

Figure 2.11: Convergence rates of local multigrid `MG(64,1e-6,[V|F]22,0.7)-JAC-D` (maximal grid level 8, varying minimal level) on the subdomains in Figures 2.9c ('ref') and 2.9b ('standard', 'heuristic', 'ACGC').

a special way (see Kilian [94] and Becker [16]). Looking at the graphs labelled 'heuristic' in Figure 2.11 one can see that this indeed helps to improve the convergence behaviour. At least, the F-cycle now shows robust convergence for this configuration. The V-cycle, however, still diverges when more than 5 grid levels are involved. Consequently, we have to state that *the V-cycle multigrid method with heuristically modified grid transfer operations is not suited to solve subproblems with extended Dirichlet boundary conditions*. It is mandatory to use the F-cycle instead.

Unfortunately, for more complicated configurations the heuristic fails even when using the F-cycle. To illustrate this we change the shape of the subdomains and introduce slight anisotropies (see Figure 2.12). The difficulty of this configuration is not only caused by the higher



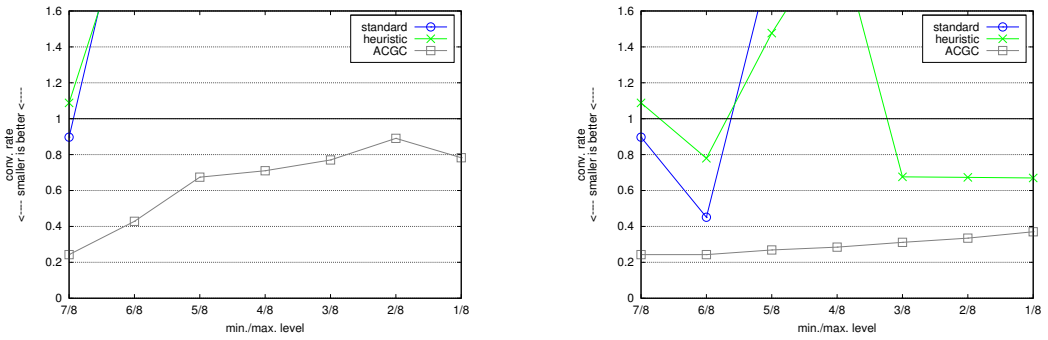
(a) level 3 grid $\bar{\Omega}^{(3)}$; extend. subdomain $\hat{\Omega}_8^{(3)}$. **(b)** level 2 grid $\bar{\Omega}^{(2)}$; extend. subdomain $\hat{\Omega}_8^{(2)}$. **(c)** level 1 grid $\bar{\Omega}^{(1)}$; extend. subdomain $\hat{\Omega}_8^{(1)}$. **(d)** Nonconforming grids $\hat{\Omega}_8^{(1)}, \hat{\Omega}_8^{(2)}, \hat{\Omega}_8^{(3)}$.

Figure 2.12: Anisotropic grid in three refinement levels with extended top centre subdomain highlighted.

element aspect ratios, another issue is the varying size of neighbouring subdomains. We focus on the top centre subdomain Ω_8. Although the subdomain itself is square shaped and refined in an isotropic way, anisotropies enter the subproblem through the minimal overlap. The corresponding matrix entries depend on the size and the shape of the neighbouring subdomains. In other words, although the subdomain Ω_8 is perfectly regular, the extended subdomain $\hat{\Omega}_8$ ac-

tually exhibits an irregular refinement pattern. The negative effect of the increasing subdomain sizes on coarser grid levels is correspondingly aggravated as Figure 2.12d illustrates.

We now examine the local multigrid method in this top centre subdomain Ω_8. We perform the same tests as for the previous configuration 2.9b, only omitting a reference computation which yields similar results as the one in configuration 2.9c. This time we use $\omega = 0.6$ as damping parameter for the local multigrid smoother. The graphs 'standard' and 'heuristic' in Figure 2.13a show that the standard V-cycle multigrid and the V-cycle multigrid with modified grid transfer operations fail even more drastically than in the isotropic configuration. Furthermore, Fig-

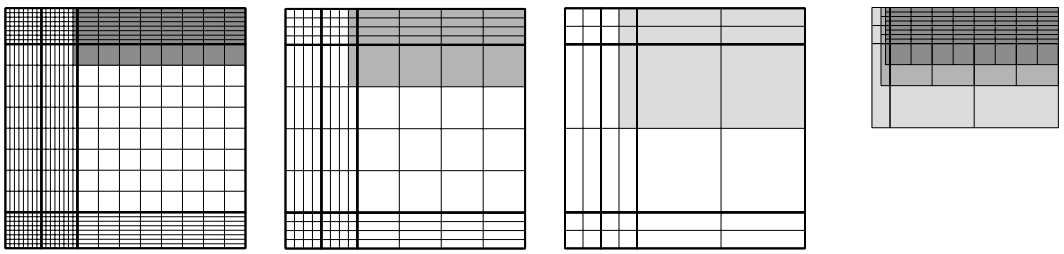


(a) V-cycle . **(b)** F-cycle .

Figure 2.13: Convergence rates of local multigrid `MG(64,1e-6,[V|F]22,0.6)-JAC-D` (maximal grid level 8, varying minimal level) on the top centre subdomain in Figure 2.12.

ure 2.13b shows that already 3 grid levels are enough to let the standard F-cycle multigrid scheme diverge (on the isotropic configuration divergence occurred only for 8 grid levels). The heuristic F-cycle multigrid, which converged robustly for the isotropic configuration, now behaves inconsistently: It surprisingly fails for problems with only few grid levels, but solves the 6-, 7- and 8-grid problems.

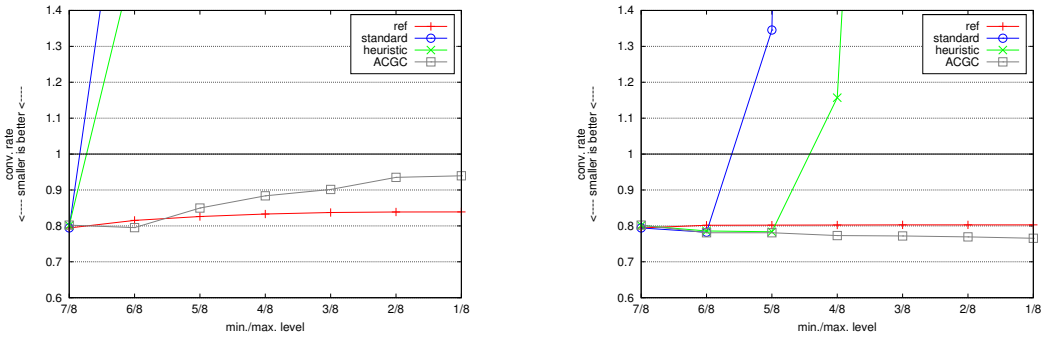
This behaviour changes for the local multigrid scheme acting on the top right subdomain (see Figure 2.14). Now, the subdomain itself exhibits higher element aspect ratios of $\sigma \approx 4.67$ (cf.



(a) level 3 grid $\bar{\Omega}^{(3)}$; extended subdomain $\hat{\Omega}_9^{(3)}$. **(b)** level 2 grid $\bar{\Omega}^{(2)}$; extended subdomain $\hat{\Omega}_9^{(2)}$. **(c)** level 1 grid $\bar{\Omega}^{(1)}$; extended subdomain $\hat{\Omega}_9^{(1)}$. **(d)** Nonconforming grids $\hat{\Omega}_9^{(1)}, \hat{\Omega}_9^{(2)}, \hat{\Omega}_9^{(3)}$.

Figure 2.14: Anisotropic grid in three refinement levels with extended top right subdomain highlighted.

equation (2.5) on page 28), which in general impairs the solution process. To show this we perform a reference computation corresponding to that in Figure 2.9c, i. e., with two Neumann- and two Dirichlet boundaries, on the anisotropic subdomain. Comparing the resulting convergence rates (graphs 'ref' in Figure 2.15) with those of the isotropic configuration (graphs 'ref' in Figure 2.11), clearly shows that this local problem is generally much more complicated to solve. Figure 2.15a demonstrates that standard and heuristic V-cycle multigrid again fail completely.



(a) V-cycle . **(b)** F-cycle .

Figure 2.15: Convergence rates of local multigrid `MG(64,1e-6,[V|F]22,0.6)-JAC-D` on top right subdomain in Figure 2.14 (maximal grid level 8, varying minimal level).

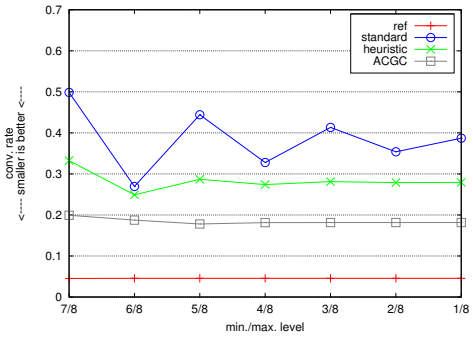
The standard and heuristic F-cycle multigrid schemes now behave somehow contrarily to the case of the top centre subdomain (see Figure 2.12): They solve the problems exhibiting only few grids, but fail for those with more than 4 and 5 grids, respectively (see Figure 2.15b).

We tried to 'rescue' the F-cycle multigrid solver by increasing the number of smoothing steps. In order to let the F-cycle 8-grid solver converge on the top right subdomain (see Figure 2.14), we had to invest *16 pre- and postsmoothing steps* which is, of course, much too expensive for the problem at hand. Furthermore, this number of smoothing steps was again insufficient to let the 9-grid solver converge. So, we have to conclude that also *the F-cycle multigrid scheme with modified grid transfer operations is not suited to solve subproblems with extended Dirichlet boundary conditions.*

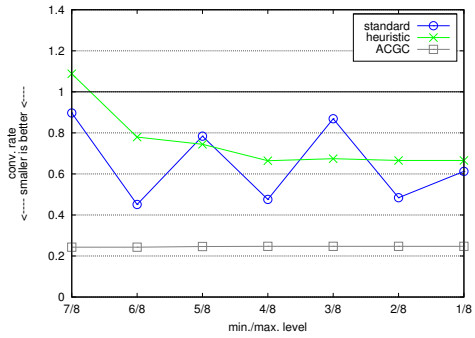
We tested other heuristics to improve the robustness of the local V- and F-cycle multigrid method. For example, we used artificial zero Dirichlet boundary conditions on coarser grid levels in order to 'switch off' the problematic terms coming from the increasing subdomain size, or we moved grid points on coarser levels to align domain sizes with each other. However, all these heuristics quite drastically violate standard multigrid methodology. Furthermore they are inflexible, i. e., they do not allow adjustment to locally varying problem configurations, and the implementation is intricate and requires non-standard modifications of basic finite element routines (like matrix assembly, grid organisation, etc.). Finally, although these heuristics yield partially better results than those of Kilian [94] and Becker [16], they are also not reliable: For each heuristic we found configurations for which the solver failed.

2.6.1.2 W-Cycle Multigrid

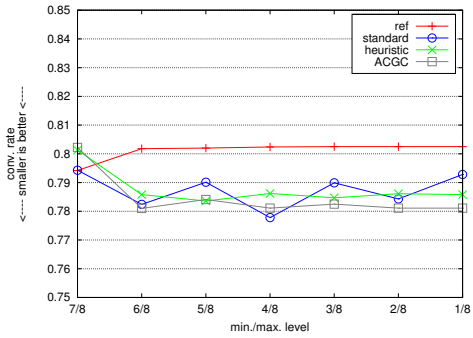
Kilian [94] and Becker [16] state that the F-cycle usually behaves similarly to the W-cycle. Since the latter is more expensive, they neglect it and exclusively use the F-cycle for their numerical tests. However, in view of the deficiencies of V- and F-cycle demonstrated in the previous paragraphs, we have to repeat the experiments employing the W-cycle this time. The results are presented in Figure 2.16 and have to be compared to those in Figures 2.11, 2.13 and 2.15, respectively. We can make some interesting observations. First, the reference com-



(a) Isotropic configuration (compare with Figure 2.11).



(b) Anisotropic configuration, top centre subdomain (compare with Figure 2.13).



(c) Anisotropic configuration, top right subdomain (compare with Figure 2.15).

Figure 2.16: Convergence rates of local W-cycle multigrid `MG(64,1e-6,W22,0.6)-JAC-D` (maximal grid level 8, varying minimal level).

putations (graphs labelled 'ref') show that F- and W-cycle indeed behave similarly on default configurations. However, when considering local problems with extended Dirichlet conditions, we see a clear advantage of the W-cycle multigrid: It converges for all three configurations, also where the F-cycle failed (graphs labelled 'standard'). However, the convergence behaviour is strangely oscillatory in the sense that the convergence rates are clearly better when the number of grid levels is odd. This is some consequence of the extended Dirichlet boundary conditions, which is not quite understood yet. We tested the standard W-cycle for many more configu-

rations and grid levels and found these results confirmed: These strange oscillations occur to some extent, but apart from that, the method yields robust convergence behaviour. The W-cycle multigrid scheme with modified grid transfer operations (graphs labelled 'heuristic') seemingly behaves quite robustly, too. However, the divergence of the 2-grid method on the top centre subdomain (see Figure 2.13) shows that this is not the case. Such 'glitches' also occurred for other grid levels – rarely, though, but sufficiently often to actually render the method unreliable.

After all, the standard W-cycle scheme is the only one of all multigrid methods tested so far that solves local problems with extended Dirichlet boundary conditions in a relatively robust way. All other methods, i. e., the standard V- and F-cycle multigrid schemes and the V-, F- and W-cycle multigrid schemes with modified grid transfer operations, are prone to diverge and thus have to be rejected. This situation is quite unsatisfying: First, the standard W-cycle multigrid method shows the described oscillations, and second, W-cycles are expensive. So, the question is if there is a technique that on domains with extended Dirichlet boundary conditions either restores robustness of standard V- and F-cycle multigrid schemes, or that at least removes the oscillatory behaviour of the W-cycle multigrid scheme. The positive answer is given in the following section.

2.6.1.3 Adaptive Coarse Grid Correction

The problem caused by the extended Dirichlet boundary conditions is that the coarse grid solution obtained on level $l-1$ is not an optimal correction for the solution vector on level l since it actually has been computed on a larger domain. So, the idea is to *adaptively damp the coarse grid correction* with the aim to sufficiently suppress its unfavourable parts. Therefore, we do not add the full prolongated coarse grid correction vector, but determine an 'optimal' steplength for this update. This technique is also employed by other authors to increase the robustness of multigrid methods. While Braess [30] uses it to alleviate locking effects (also compare Section 3.2.2), Turek [158] and John and Tobiska [89] identify the technique as essential for solving linear systems stemming from nonconforming finite element discretisations. Reusken [134] examines theoretical aspects of steplength optimisation in multigrid methods.

We focus on the update of the iteration vector by the prolongated coarse grid correction within the local multigrid method (compare equation (2.7d) on page 30 for the analogous global multilevel method and line 17 in Listing 2.1 on page 41). On each level $l = 2,\dots,L$, we damp this update with a steplength parameter α:

$$\mathbf{x}^{(l)} \leftarrow \mathbf{x}^{(l)} + \alpha \mathbf{P}^{(l)} \mathbf{x}^{(l-1)}.$$

(We omit the subscripts denoting the subdomain here.) This damping parameter is chosen in such a way that the updated vector $\mathbf{x}^{(l)}$ minimises the energy norm

$$(\mathbf{x}^{(l)} - \mathbf{x}^{*})^{\mathsf{T}} \mathbf{A}^{(l)} (\mathbf{x}^{(l)} - \mathbf{x}^{*}),$$

where $\mathbf{x}^*$ is the exact solution. When $\mathbf{A}^{(l)}$ is symmetric this leads to

$$\alpha = \frac{\mathbf{c}^{(l)\mathsf{T}}\mathbf{d}^{(l)}}{\mathbf{c}^{(l)\mathsf{T}}\mathbf{A}^{(l)}\mathbf{c}^{(l)}}, \tag{2.13}$$

where $\mathbf{c}^{(l)} := \mathbf{P}^{(l)}\mathbf{x}^{(l-1)}$ is the prolongated coarse grid correction and $\mathbf{d}^{(l)} := \mathbf{b}^{(l)} - \mathbf{A}^{(l)}\mathbf{x}^{(l)}$ the current defect. One could also optimise with respect to the defect norm, $\mathbf{d}^{(l)\mathsf{T}}\mathbf{d}^{(l)}$, but numerical experiments (which we do not present here) showed that energy minimisation yields better results (also compare Turek [158]).

We enhance the standard multigrid cycles by applying the described adaptive coarse grid correction (ACGC) and repeat the tests above. The graphs labelled 'ACGC' in Figures 2.11, 2.13, 2.15 and 2.16 make the success of this technique apparent. The V-cycle converges for all configurations now, though it shows a dependence on the number of grid levels. The F-cycle behaves even more robustly, only on the top centre subdomain a slight dependence on the number of grid levels can be observed. The W-cycle converges smoothly now, the oscillations that are characteristic for the standard W-cycle vanish completely. Furthermore, the W-cycle with ACGC exhibits better convergence rates: On the isotropic configuration (Figure 2.16a) and on the top centre subdomain of the anisotropic configuration (Figure 2.16b) the convergence rate is about $c = 0.2$ while that of the standard W-cycle oscillates between $c = 0.3$ and $c = 0.5$ and between $c = 0.4$ and $c = 0.8$, respectively. On the top right subdomain of the anisotropic configuration (Figure 2.16c) the differences are not as crucial since the convergence behaviour in this subdomain is dominated by the high aspect ratios (and not by the influence of the extended Dirichlet boundary conditions).

In summary, enhancing multigrid with adaptive coarse grid correction exactly achieves the desired goals on domains with extended Dirichlet conditions: It prevents divergence of the V- and F-cycle multigrid methods and it suppresses the oscillations of the standard W-cycle multigrid method. Compared to the heuristics discussed in the previous sections, the adaptive coarse grid correction has some further advantages:

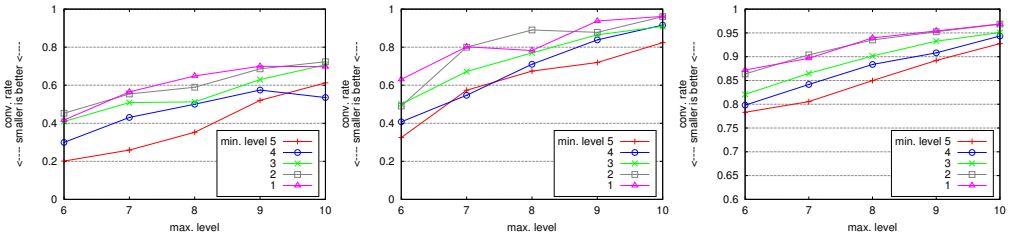
- It is an established technique whose functionality has also been examined from the theoretical point of view [134].
- It can be implemented very easily with standard techniques, i. e., matrix vector multiplications and scalar products with data that is readily available. Non-standard modifications of kernel routines like grid transfer or matrix assembly are not necessary.
- Its arithmetic costs (one matrix vector multiplication, two scalar products) are comparatively low (cf. page 58).
- The damping value is adaptively computed per subdomain. It will be small on subdomains that strongly suffer from the extended Dirichlet boundary conditions, and it will be close to 1.0 on those subdomains that are only mildly affected. This way the overall solution process is not unnecessarily impaired in general.

Above plots only show exemplary results for maximal multigrid level 8. For sake of completeness, we now want to present the results for other grid levels, as well. Therefore, we vary the minimal grid level between 1 and 5, and the maximal grid level between 6 and 10, such that the solvers vary between 2-grid (minimal level 5, maximal level 6) and 10-grid methods (minimal level 1, maximal level 10). Figure 2.17 shows the results for those methods that were identified as robust: V-, F- and W-cycle multigrid with ACGC and standard W-cycle multigrid, each of them applied to the three configurations used above. Reading the plots 'horizontally' shows how the convergence rates vary when the maximal multigrid level, i. e., the number of unknowns, is increased. Reading the plots 'vertically' shows the influence of varying the minimal multigrid level (with fixed number of unknowns). Overall, the plots confirm all the observations made before. First of all, all tests converge. The V-cycle multigrid with ACGC exhibits the strongest dependencies 'in horizontal and vertical direction', while the W-cycle multigrid with ACGC behaves almost perfectly independent of maximal and minimal grid level and yields the best convergence rates. The F-cycle multigrid with ACGC lies in between, with respect to both, variations and absolute convergence rate values. The convergence rates of the standard W-cycle multigrid (without ACGC) are worse than those of the F- and W-cycle multigrid with ACGC, but better than those of the V-cycle multigrid with ACGC. Additionally, the unfavourable oscillations of the standard W-cycle consistently occur for all multigrid levels.

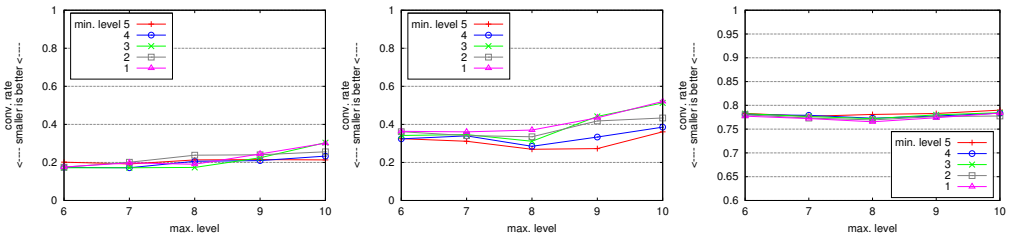
In summary, the W-cycle multigrid method with ACGC clearly shows the best convergence behaviour. However, this method also is the most expensive one in terms of number of floating point operations (FLOPs) per iteration. We compare the four methods now in this regard. First, we estimate the costs of the three multigrid cycles without ACGC. We denote the number of unknowns in a subdomain on level l with $n_{\text{loc}}^{(l)}$ (cf. Section 2.3).[10] Due to the regular refinement strategy we have $n_{\text{loc}}^{(l)} = (2^l+1)^2 = 4^{l-L} n_{\text{loc}}^{(L)}$, where L denotes the maximal level (see Section 2.3). The costs of one complete Jacobi smoothing step (including defect calculation, application of the smoothing operator and update of the iteration vector) on level l sum up to $23 n_{\text{loc}}^{(l)}$, while for the additional defect calculation and grid transfer operations within the multigrid cycle about $24 n_{\text{loc}}^{(l)}$ operations are necessary [16]. Assuming that the minimal grid level is $l = 1$ and denoting the total number of smoothing steps with s, the arithmetic costs of complete multigrid cycles on maximal level $L \geqslant 2$ are:

$$\begin{aligned}
\text{FLOP}_{\text{V}}(L,s) &= \text{FLOP}_{\text{UMF}}(n_{\text{loc}}^{(1)}) + (23s+24) n_{\text{loc}}^{(L)} \sum_{l=2}^{L} 4^{l-L}, \\
\text{FLOP}_{\text{F}}(L,s) &= (L-1)\,\text{FLOP}_{\text{UMF}}(n_{\text{loc}}^{(1)}) + (23s+24) n_{\text{loc}}^{(L)} \sum_{l=2}^{L} (L-l+1) 4^{l-L}, \\
\text{FLOP}_{\text{W}}(L,s) &= 2^{L-2}\,\text{FLOP}_{\text{UMF}}(n_{\text{loc}}^{(1)}) + (23s+24) n_{\text{loc}}^{(L)} \sum_{l=2}^{L} 2^{L-l} 4^{l-L}.
\end{aligned} \tag{2.14}$$

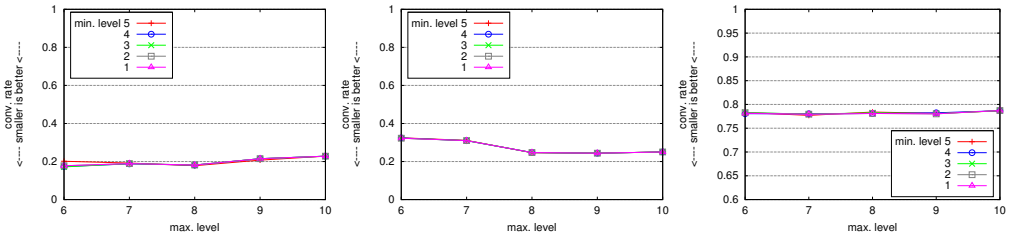
[10] Due to the finite element discretisation with Q_1 bilinear elements, the number of unknowns coincides with the number of mesh nodes (disregarding Dirichlet boundary nodes).



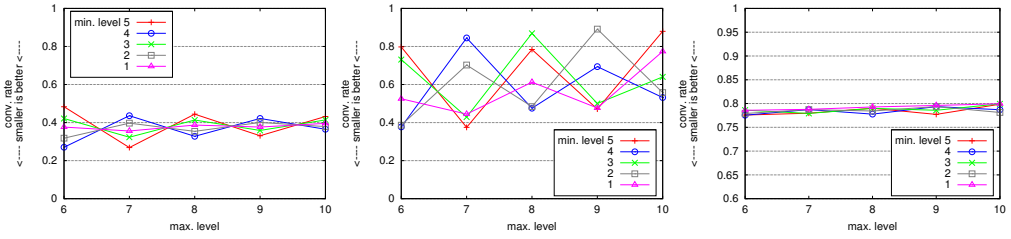
(a) V-cycle with ACGC.



(b) F-cycle with ACGC.



(c) W-cycle with ACGC.



(d) standard W-cycle.

Figure 2.17: Different multigrid methods on isotropic configuration 2.9b (left), on top centre subdomain of anisotropic configuration 2.12 (centre), and top right subdomain of anisotropic configuration 2.14 (right), respectively. The x-axis shows the maximal multigrid level (6 – 10), and each of the five graphs represents a minimal multigrid level (1 – 5). (Note the varied y-axis of the right hand plots.)

The term $\mathrm{FLOP}_{\mathrm{UMF}}(n)$ represents the number of FLOPs UMFPACK needs for forward and backward substitution of a system with n unknowns.[11] In the literature, this number is usually specified as $O(n^2)$. In FEAST, however, the local systems always have the same special band structure, i. e., independent of the equation and the grid shape, UMFPACK will always need

[11] The costs for the LU decomposition are neglected here since it is only performed once in the initialisation phase.

about the same number of FLOPs on a given level. So, we can simply determine this number by enquiring the UMFPACK statistics after solving. We did so and considered UMFPACK solves on levels 1 to 10. Figure 2.18 shows the resulting number of FLOPs per degree of freedom (DOF). Without knowing the exact functional dependency on the number of unknowns, we can at least estimate $\mathrm{FLOP}_{\mathrm{UMF}}(n) = O(n\log^p(n))$ with $1.5 < p < 2$.

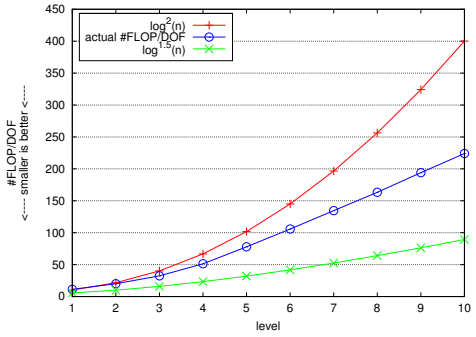


Figure 2.18: Actual number of UMFPACK FLOPs per DOF on different multigrid levels and reference curves $\log^{1.5}(n)$ and $\log^{2.0}(n)$. The number of unknowns on level l is given by $n_{\mathrm{loc}}^{(l)} = (2^l+1)^2$.

We denote the sums at the end of the formulas (2.14) with $\Sigma_{\mathrm{V}}(L), \Sigma_{\mathrm{F}}(L)$ and $\Sigma_{\mathrm{W}}(L)$, respectively. The factors $L-l+1$ and 2^{L-l} for the F- and W-cycle become clear when looking at the schematic representation of the cycles in Figure 2.10 on page 48. Neglecting the coarse grid solver, these sums represent the factors by which the work performed on the finest level (here $(23s+24)n_{\mathrm{loc}}^{(L)}$) has to be multiplied to obtain the costs of the complete cycle. Applying index shifts and formulas for geometric series[12], these sums can be computed precisely:

$$\Sigma_{\mathrm{V}}(L) = \sum_{l=0}^{L-2} 4^{-l} = \frac{1}{3}(4-4^{2-L}) \xrightarrow{L\to\infty} \frac{4}{3},$$

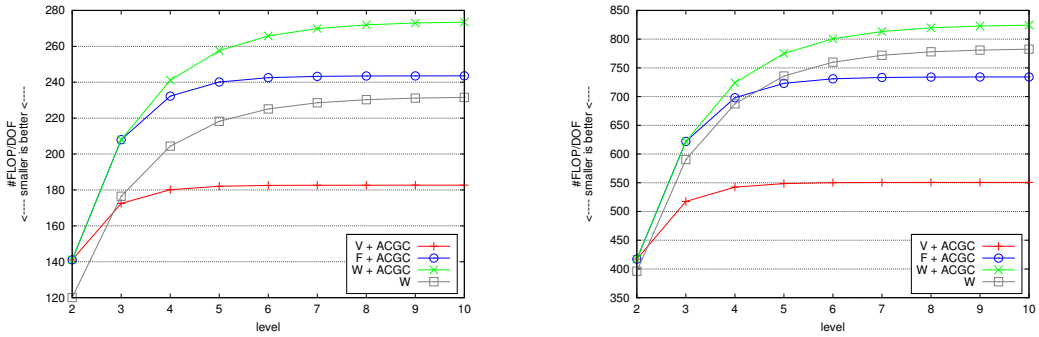
$$\Sigma_{\mathrm{F}}(L) = \sum_{l=0}^{L-2} 4^{-l}(l+1) = \Sigma_{\mathrm{V}}(L) + \frac{1}{9}\big((2-3L)4^{2-L}+4\big) \xrightarrow{L\to\infty} \frac{16}{9},$$

$$\Sigma_{\mathrm{W}}(L) = \sum_{l=0}^{L-2} 2^{-l} = 2-2^{2-L} \xrightarrow{L\to\infty} 2.$$

We see that the W-cycle is about 1.5 (1.125) times more expensive than the V-cycle (F-cycle) when considering mere FLOP counts. The addition of UMFPACK FLOPs (see equation (2.14) and Figure 2.18) does not significantly change these ratios. However, for the preparation of UMFPACK solves, RHS and solution vectors have to be converted between FEAST's and UMFPACK's format. Consequently, since UMFPACK is called 2^{L-2} times in case of the W-cycle, in practice one has to expect some degradation of performance compared to the V-cycle (F-cycle) where UMFPACK is only called once ($L-1$ times).

[12] The two formulas are $\sum_{k=0}^{n} q^k = \frac{1-q^{n+1}}{1-q}$ and $\sum_{k=0}^{n} kq^k = \frac{nq^{n+2}-(n+1)q^{n+1}+q}{(1-q)^2}$ for $q \neq 1$.

We now consider the additional costs for the adaptive coarse grid correction. In detail, after each prolongation from level $l-1$ to level l one matrix vector multiplication ($17n_{\text{loc}}^{(l)}$ FLOPs) and two scalar products ($4n_{\text{loc}}^{(l)}$ FLOPs) have to be performed (cf. equation (2.13)). This means, that the factor $(23s+24)n_{\text{loc}}^{(L)}$ in formulas (2.14) increases to $(23s+45)n_{\text{loc}}^{(L)}$. Figure 2.19 compares the arithmetic costs for the four types of multigrid cycles, i. e., V-, F- and W-cycle with ACGC and W-cycle without ACGC. The plots display the number of FLOPs per DOF needed for a



(a) 2 pre- and 2 postsmoothing steps. **(b)** 8 pre- and 8 postsmoothing steps.

Figure 2.19: Number of FLOPs per DOF for one multigrid cycle with Jacobi smoothing.

complete cycle performing 2+2 or 8+8 Jacobi smoothing steps with minimal level 1. One can see that despite the additional work for the computation of the optimal step length, the V-cycle with ACGC is less costly than the W-cycle without ACGC. When more work is invested into the smoothing process, then the share of the step length computation in the total costs decreases: When applying only 2+2 Jacobi smoothing steps (Figure 2.19a), the F-cycle with ACGC is more expensive than the W-cycle without ACGC, though, but when the number of smoothing steps is increased to 8+8 (Figure 2.19b), this is the other way around. A similar effect has to be expected when instead of Jacobi a more expensive smoother like ADITriGS is used.

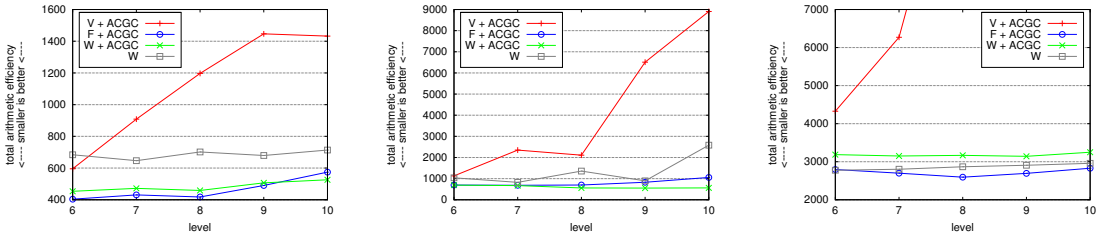
There is obviously a relation between costs per cycle and convergence behaviour. The W-cycle with ACGC yields the best convergence rates, but is also the most expensive method. So, we consider the *total arithmetic efficiency* of the four multigrid methods now. We use FEAST's built-in mechanism for FLOP counting to get the exact number of FLOPs that the four methods need to obtain the desired 6 digits in the above configurations. The **total arithmetic efficiency** in terms of FLOPs is then defined by

$$\text{total arithmetic efficiency} = -\frac{\#\text{FLOPs}}{\#\text{DOF} \times \#\text{iter} \times \log_{10}(c)}, \tag{2.15}$$

i. e., the number of FLOPs scaled with the number of DOF and with the digits the iteration actually gained (cf. equation (2.12) on page 48). So, *the total arithmetic efficiency tells how*

many FLOPs are needed per DOF to gain one digit and, hence, represents a suitable measure to compare the arithmetic costs of different solution algorithms.[13]

Figure 2.20 shows that the costs of the V-cycle with ACGC clearly rise with the increasing multigrid level. The results that are out of range of Figure 2.20c are 12419 (level 8), 20570



(a) Isotropic configuration.

(b) Anisotropic configuration, top centre subdomain.

(c) Anisotropic configuration, top right subdomain.

Figure 2.20: Total arithmetic efficiency of the methods `MG(1024,1e-6,[V|F|W]22,[0.6|0.7])-JAC-D` (with or without ACGC) to solve the local systems, minimal level 1.

(level 9) and 35096 (level 10) FLOPs, respectively. So, it can clearly not compete with the three other methods whose costs show no or only mild dependence on the number of unknowns. The W-cycle with ACGC exhibits the smallest dependencies, though, but it is not always the least expensive one. Especially on the top right subdomain where the convergence behaviour is dominated by the high element aspect ratios, the F-cycle with ACGC and the W-cycle without ACGC are less expensive.

These examples already show that it is hardly possible to give a general answer to the question which of the methods is best suited to solve local problems with extended Dirichlet boundary conditions. In terms of robustness, the W-cycle with ACGC seems to be clearly the best choice, but it is not always the best candidate in terms of total costs.

Local Multigrid Schemes within Global Solves

To better assess the presented local methods, we will now actually examine them in their role as local smoothers in a global 2-layer-ScaRC solver. Until now, we rather considered them as 'stand-alone' solvers in order to compare their general convergence behaviour, neglecting the global problem. Unfortunately, we cannot simply transfer the obtained results to the case where the local solvers are used as preconditioners in a 2-layer-ScaRC scheme: Here, the local solvers are applied repeatedly to local systems with possibly varying degree of difficulty, and these

[13] One has to be aware, however, that the amount of arithmetic work does not directly translate into corresponding runtimes, not even for serial computations on a single commodity processor. Different hardware characteristics, especially the size of the cache, influence the actual runtimes. For example, operations on smaller grid levels, for which the necessary data completely fits into the cache, potentially perform significantly faster since expensive memory access is minimised. (See, for example, Becker [16] for more detailed hardware-specific discussions and examples.) In Section 4.2.7.4 we will also introduce the *total runtime efficiency*, for which the number of FLOPs in equation (2.15) is replaced by the runtime in microseconds.

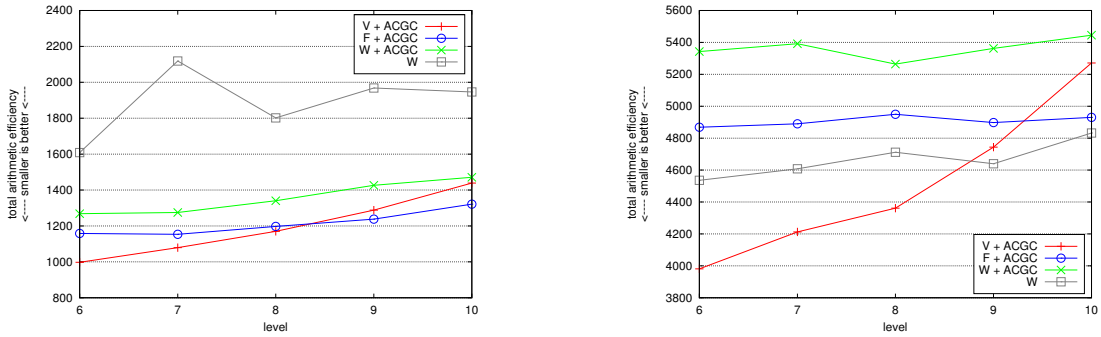
systems are usually not solved as accurately as it is done in the previous tests. Hence, we now solve the isotropic configuration depicted in Figure 2.9 and the anisotropic one (Figure 2.12) globally with a 2-layer-ScaRC solver. We use a global V-cycle multigrid with 1 pre- and 1

```
1 solver=MG, maxiter=32, tol=REL:1e-6, cycle=V:1:1
2 smoother=LOCAL, damp=0.7
3   solver=MG, maxiter=64, tol=REL:1e-1, cycle=V:2:2, cgcdamp=ADAP
4   smoother=JACOBI, damp=0.7
5   coarse=UMFPACK
6 coarse=UMFPACK
```

Listing 2.8: 2-layer-ScaRC solver `MG(1e-6,V11)__MG(64,1e-1,V22,0.7)-JAC-D__D` with adaptive damping of the coarse grid correction '`cgcdamp=ADAP`' (operations on the local layer highlighted in grey).

postsmoothing step and damping $\omega = 0.7$. As local solvers we use the four multigrid schemes examined above, with the only difference that we now gain only 1 digit. Listing 2.8 exemplarily shows the version with a local V-cycle with ACGC.

Figure 2.21 displays the total arithmetic efficiency of the four schemes to solve the global systems on varying maximal level and fixed minimal level 1. On the isotropic configuration all



(a) Isotropic configuration.

(b) Anisotropic configuration.

Figure 2.21: Total arithmetic efficiency of the 2-layer-ScaRC solver in Listing 2.8 to solve the global systems, minimal level 1.

variants with ACGC need 6 or 7 global iterations to achieve the desired accuracy, while the number of global iterations for the standard W-cycle varies between 7 and 9. On the anisotropic configuration all variants need 11 or 12 iterations. Figure 2.21a shows that on the isotropic configuration the standard W-cycle is clearly the most expensive one and shows a certain degree of irregularity and dependency with regard to the maximal grid level. F- and W-cycle with ACGC behave similarly to each other, where the F-cycle is consistently less expensive. Both show a mild dependency on the grid level and behave quite regularly. The V-cycle with ACGC, however, shows a stronger level dependency, such that it is cheaper for levels 6 – 8, though, but already more expensive than the F-cycle with ACGC on levels 9 and 10, and even nearly as expensive as the W-cycle with ACGC on level 10. On the anisotropic configuration (Figure 2.21b) the behaviour is similar. Here, the V-cycle also shows the strongest level dependency and is the

cheapest method on levels 6 – 9. On level 10, however, it is more expensive than the F-cycle with ACGC and than the standard W-cycle. The W-cycle with ACGC is the most expensive method on the anisotropic configuration. The standard W-cycle behaves surprisingly well, but it also shows a slight level dependency. F- and W-cycle with ACGC behave slightly irregularly, though, but in total the costs per DOF seem not to rise significantly for increasing grid levels (cf. level 6 and level 10).

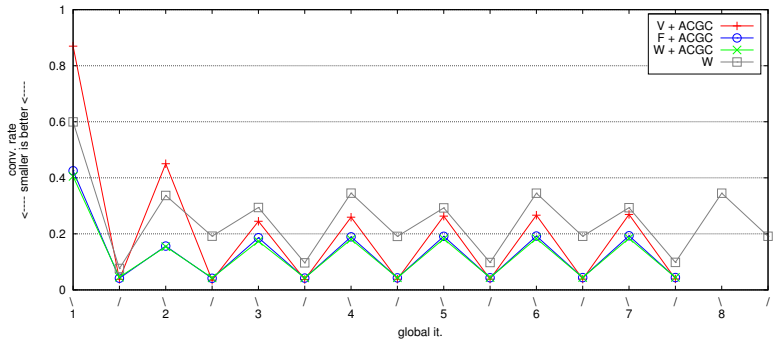
The most remarkable observation, however, is that the costs of the global solution process are not consistent with the results we obtained for the local solvers. In view of the immense costs of the local V-cycle multigrid with ACGC (cf. Figure 2.20), it is especially surprising how comparatively well it performs, when the whole global solution process is considered. So, it is obvious that the previous results for local solvers can not simply be transferred to the global situation. In order to better understand this circumstance, we now examine how the local solvers behave in the course of the global iteration. We use a global V-cycle multigrid performing 1 pre- and 1 postsmoothing step (see Listing 2.8), i. e., in each global iteration the local solvers are called two times per level, once in the restriction phase and once in the prolongation phase. In Figure 2.22 we show the corresponding convergence rates for the local solvers in the top right subdomain of the isotropic configuration (Figure 2.22a) and the top centre and top right subdomain of the anisotropic configuration (Figures 2.22b and 2.22c). For the sake of a clearer representation, we restrict ourselves to the convergence rates of maximum level 10 and minimum level 1.

Looking at the three plots, the discrepancy between local and global results becomes apparent immediately: The convergence behaviour of the very first local solves (presmoothing in the first global iteration) is – in most cases – significantly worse than that for the following solves. Especially the difference of the convergence rates in the presmoothing phase (denoted with '\') and the postsmoothing phase (denoted with '/') of the first global iteration is remarkable: The convergence rate of the V-cycle with ACGC in the isotropic configuration, for instance, improves from $c \approx 0.9$ for presmoothing to $c \approx 0.05$ for postsmoothing (see Figure 2.22a). While the other local methods perform clearly better in the first global presmoothing step ($c \approx 0.4$ for F- and W-cycle with ACGC, $c \approx 0.6$ for standard W-cycle), their convergence rate in the first postsmoothing phase is similar to (or even slightly worse than) that of the V-cycle.

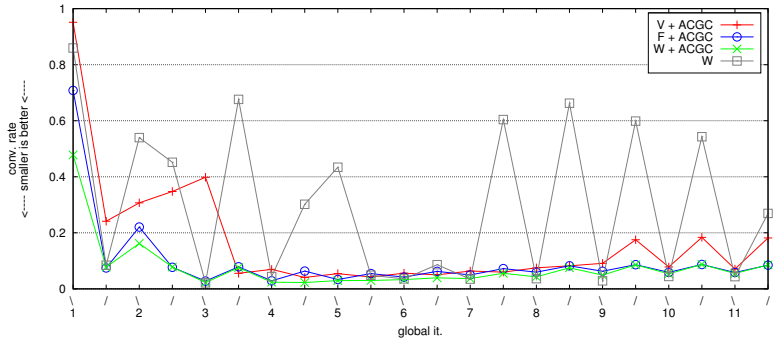
We observed the phenomenon, that the first local solves are much more difficult, on all subdomains and in all computations we performed. In this sense, our detailed tests of local solvers (see Figures 2.11, 2.13, 2.15, 2.16, 2.17 and 2.20) are not representative for the overall solution process. However, solving the local systems in the first global presmoothing step in a robust and efficient way is a prerequisite for the success of the overall solution method. Hence, the local solver tests remain valid and still make sense, since they cover exactly this 'worst case' configuration.

We can observe further aspects in Figure 2.22:

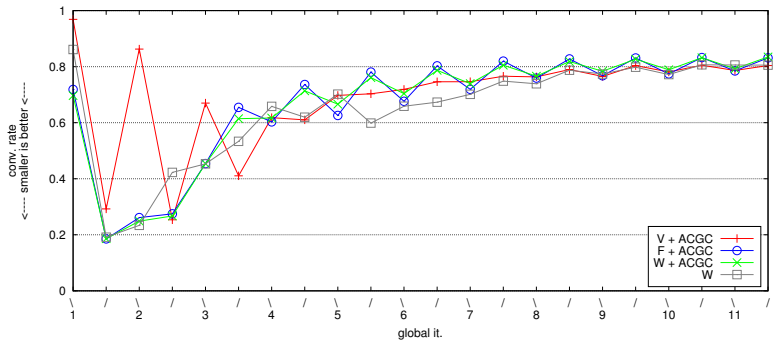
- In the isotropic configuration (Figure 2.22a) we see a regular pattern: The local solver performance in the global presmoothing phase is clearly worse than in the postsmoothing



(a) Isotropic configuration, top right subdomain.



(b) Anisotropic configuration, top centre subdomain.



(c) Anisotropic configuration, top right subdomain.

Figure 2.22: Convergence rates of local multigrid schemes on level 10 in the course of the global iteration. '\' denotes the local solve in the restriction phase of the global multigrid, '/ ' the local solve in the prolongation phase.

phase. In the anisotropic configuration (Figures 2.22b and 2.22c) this is no longer true in general. It partially even seems to be the other way around.

- The standard W-cycle shows strong irregularities on the top centre subdomain of the anisotropic configuration (Figure 2.22b).

- Apart from the standard W-cycle the local convergence rates on the top centre subdomain of the anisotropic configuration (Figure 2.22b) are on average even better than those of the isotropic configuration (Figure 2.22a).

- The convergence behaviour on the top right subdomain of the anisotropic configuration (Figure 2.22c) is dominated by the high element aspect ratios. That is why the convergence rates of all four methods group around $c \approx 0.8$ in later global iterations. It is interesting that here the V-cycle with ACGC and the standard W-cycle are partially better than the F- and W-cycle with ACGC.

Currently, we are not able to explain every feature of the convergence curves in Figure 2.22. In summary, however, the plots show that the performance of the local multigrid methods on subdomains with extended Dirichlet boundary conditions crucially depends on the given right hand side data. The influence of the extended Dirichlet boundary conditions decreases due to the residuals getting smaller and smoother in the course of the global iteration.

2.6.1.4 Extended Dirichlet Boundary Conditions with ADITriGS

In order to show that above deficiencies of (standard) multigrid grid schemes are not due to the simple Jacobi smoother, we will now examine ADITriGS. We use the solver scheme depicted in Listing 2.9, where the global Richardson scheme is again just a dummy to call the local solvers. We perform two pre- and two postsmoothing steps, the first smoothing step being

```
1 solver=RICH, maxiter=1, tol=IGNORE
2 prec=LOCAL, damp=1.0
3   solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:2:2
4   smoother=ADITRIGS, damp=1.0
5   coarse=UMFPACK
```

Listing 2.9: Local solver `MG(64,1e-6,V22)-ADI-D` (highlighted in grey) as part of a 'dummy' global Richardson scheme.

TriGS, respectively, and the second one MTriGS (cf. Section 2.5.3.1). We also consider F- and W-cycles. To justify the usage of ADITriGS we take the grid in Figure 2.14 on page 50 and increase the anisotropies (see Figure 2.23). We only examine the top right subdomain, which

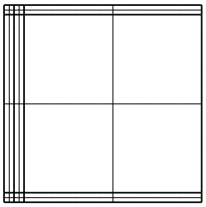
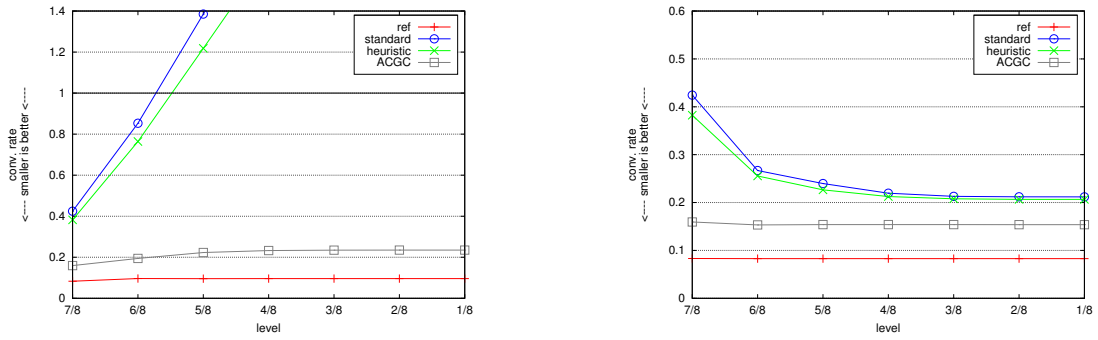


Figure 2.23: Anisotropic grid used for the ADITriGS tests.

exhibits two Neumann boundaries, two extended Dirichlet boundaries and element aspect ratios of $\sigma = 18$. We perform reference computations on this subdomain (denoted with 'ref' in the following plots) where we exchange the two extended Dirichlet boundaries with standard Dirichlet boundaries (cf. Figure 2.9c on page 47).

Figure 2.24 exemplarily shows level 8 computations with varying minimal level. One can see



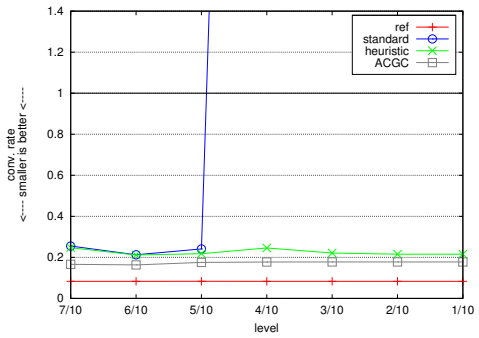
(a) V-cycle. **(b)** F-cycle.

Figure 2.24: Convergence rates of local multigrid `MG(64,1e-6,[V|F]22)-ADI-D` on top right subdomain in Figure 2.23 (maximal grid level 8, varying minimal level).

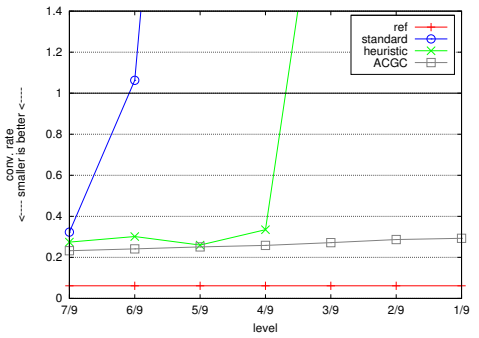
that the standard V-cycle and the V-cycle with modified grid transfer operations (graphs labelled 'standard'and 'heuristic', respectively) fail when 4 or more grid levels are involved (see Figure 2.24a). The corresponding F-cycles, however, work well (see Figure 2.24b), although they show some strange dependency on the number of grid levels. For both cycles, the ACGC version clearly yields the best results, both in terms of absolute convergence rate and in terms of independency of the number of grid levels. In the case of Jacobi, the corresponding configuration was already sufficient to let the standard F-cycle fail (cf. Figure 2.15b on page 51). In case of ADITriGS, however, level 8 and also level 9 still perform well. Only on level 10, the method suddenly diverges when 6 or more grid levels are involved (see Figure 2.25a).

The F-cycle multigrid with modified grid transfer operations using ADITriGS as smoother performs well on all levels. It was also successful in other configurations we tested. Unfortunately, this robustness seems to be confined to the simple case of the standard Poisson operator $-\Delta u$ we used for our tests so far. In Section 4.2.4 we will see that the scalar operators arising from the elasticity problem are elliptic, though, but not isotropic. They have the form $-a\partial_{xx}u - b\partial_{yy}u$ with varying degree of anisotropy $\frac{a}{b}$. We repeated the previous tests with $a = 10$ and $b = 1$ (instead of $a = b = 1$). Figure 2.25b shows that now also the F-cycle with modified grid transfer operations fails on level 9 when 5 or more grid levels are used. (For maximal level 8 the method still converges robustly for all numbers of grid levels.)

So, we have to conclude that the F-cycle multigrid with modified grid transfer operations using ADITriGS as smoother may be robust for the standard Poisson operator (which was exclusively used in the numerical studies of Kilian [94] and Becker [16]), but it is clearly not robust in case



(a) Isotropic operator $-\Delta u$ (maximal grid level 10, varying minimal level).

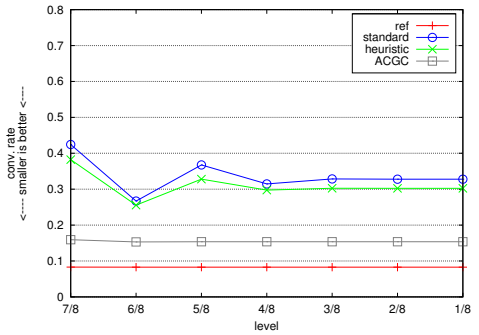


(b) Anisotropic operator $-10\partial_{xx}u - \partial_{yy}u$ (maximal grid level 9, varying minimal level).

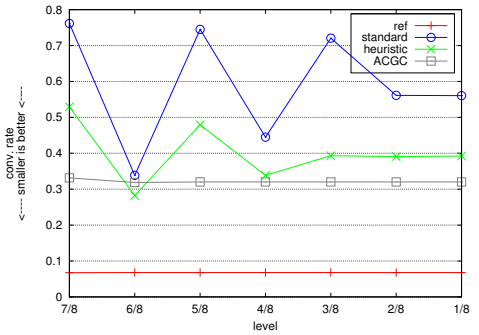
Figure 2.25: Convergence rates of local F-cycle multigrid `MG(64,1e-6,F22)-ADI-D` on top right subdomain in Figure 2.23.

of the operators that arise, for instance, in the context of solid mechanics. The F-cycle with ACGC, however, converges robustly and independently of the number of grid levels also in case of such more complicated operators. The V-cycle with ACGC also solves these problems robustly, though, but it exhibits some dependency on the number of grid levels (results not presented here).

Figure 2.26 shows the convergence behaviour of the different W-cycles for the isotropic and the anisotropic operator. We see that both the standard and the heuristic W-cycle converge for all



(a) Isotropic operator $-\Delta u$.



(b) Anisotropic operator $-10\partial_{xx}u - \partial_{yy}u$.

Figure 2.26: Convergence rates of local W-cycle multigrid `MG(64,1e-6,W22)-ADI-D` on top right subdomain in Figure 2.23 (maximal grid level 8, varying minimal level).

numbers of grid levels, though, but they show a similar oscillating behaviour as in the Jacobi case (cf. Figure 2.16 on page 52). The W-cycle with ACGC again shows the best convergence behaviour. The results on level 9 and 10 (not presented here) are qualitatively equal to those of level 8.

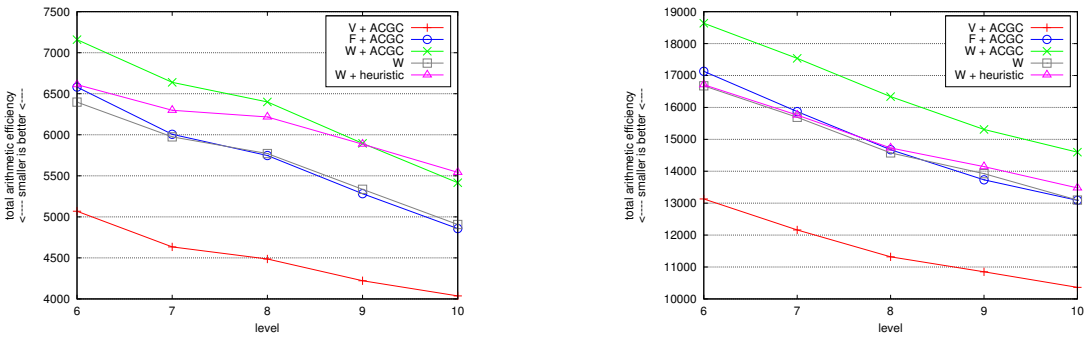
Local Multigrid Schemes within Global Solves

Just like in the Jacobi case, it is not feasible to directly deduce the global solver costs from the local solver results. So, we treat the global problem now with the solver depicted in Listing 2.10. Due to the high macro anisotropies in the grid, we perform a global F-cycle with 2

```
1 solver=MG, maxiter=64, tol=REL:1e-6, cycle=F:2:2
2 smoother=LOCAL, damp=0.7
3   solver=MG, maxiter=64, tol=REL:1e-1, cycle=V:2:2, cgcdamp=ADAP
4   smoother=ADITRIGS, damp=1.0
5   coarse=UMFPACK
6 coarse=UMFPACK
```

Listing 2.10: 2-layer-ScaRC solver `MG(1e-6,F22)__MG(64,1e-1,V22)-ADI-D__D`, here with adaptive damping of the coarse grid correction (operations on the local layer highlighted in grey).

pre- and 2 postsmoothing steps. Figure 2.27 shows the total arithmetic efficiency of the five methods that behaved robustly in the previous tests. We fix the minimal level 1 and vary the maximal level. Figure 2.27a displays the results for the isotropic operator and Figure 2.27b those of the anisotropic one.[14] We can see that the V-cycle with ACGC is clearly the cheapest



(a) Isotropic operator $-\Delta u$. **(b)** Anisotropic operator $-10\partial_{xx}u - \partial_{yy}u$.

Figure 2.27: Total arithmetic efficiency of the 2-layer-ScaRC solver in Listing 2.10 to solve the global systems, minimal level 1.

method and the W-cycle with ACGC the most expensive one. The other three methods lie in between where the F-cycle with ACGC and the standard W-cycle are quite close to each other and in general slightly less costly than the W-cycle with modified grid transfer operations. It is interesting to observe that the total arithmetic efficiency of the V-cycle with ACGC does not deteriorate with increasing grid level as it was the case with Jacobi as smoother (cf. Figure 2.20 on page 59). Consequently, the corresponding global 2-layer-ScaRC solver is the least expensive one over all grid levels. With Jacobi as smoother this was only the case on the smaller grid levels (cf. Figure 2.21 on page 60).

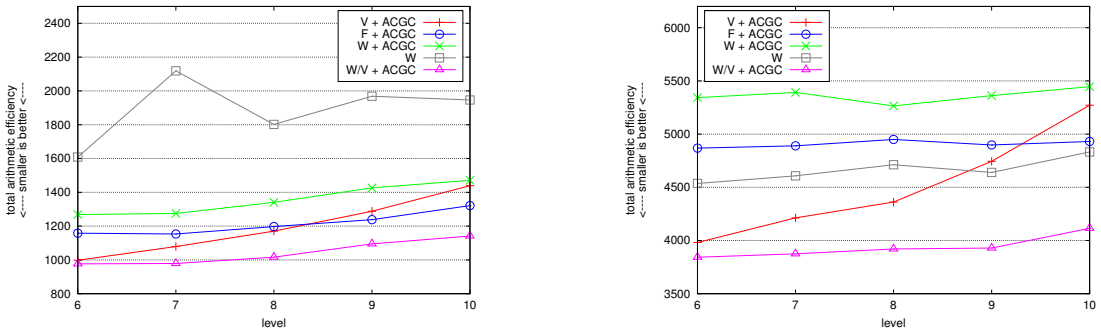
[14] In the example at hand, the number of global iterations decreases with increasing multigrid level, which explains the negative slope of the curves.

2.6.1.5 Varying Local Cycles

Considering the phenomenon that the first local solves seem to be much more difficult than all the following ones, we want to briefly discuss a simple, yet obvious strategy:

> *In the first global iteration, use W-cycle with ACGC for the local multigrid. Then, switch to local V-cycle multigrid with ACGC.*

So, the first (difficult) local solves are robustly treated with the expensive W-cycle, which is, however, oversized for the following (easier) solves. These can be performed much more efficiently with the inexpensive V-cycle. To show the success of this strategy we repeat the tests above (isotropic Poisson operator, grids depicted in Figure 2.9a on page 47 and Figure 2.12 on page 49). Figure 2.28 shows the total arithmetic efficiency of the global solver with Jacobi smoothing. It contains the same results as Figure 2.21 on page 60, extended with the results for



(a) Isotropic configuration.

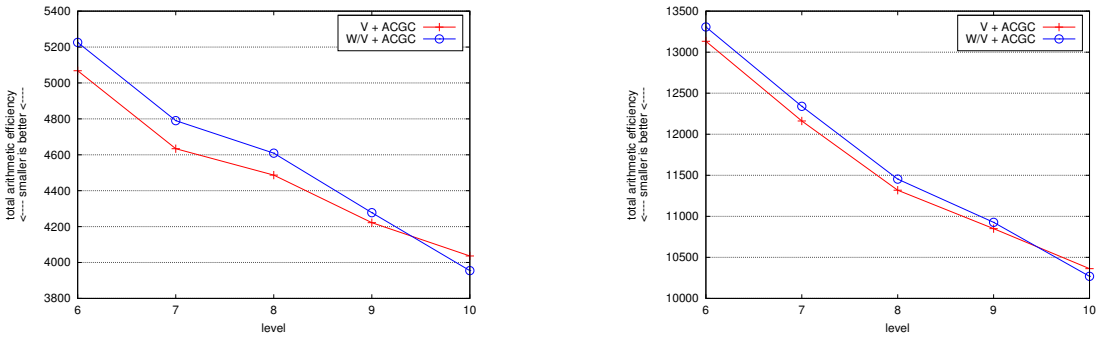
(b) Anisotropic configuration.

Figure 2.28: Total arithmetic efficiency of the 2-layer-ScaRC solver in Listing 2.8 to solve the global systems, minimal level 1.

the new strategy (denoted with 'W/V + ACGC'). As expected, the new strategy leads to the most efficient solver. The strong level dependency of the V-cycle with ACGC is clearly weakened and now comparable to that of the other methods. This especially shows that the level dependency of the V-cycle with ACGC is mainly caused by its bad performance within the very first local solves.

Figure 2.29 is the corresponding extension of Figure 2.27 for ADITriGS smoothing. Here, we only compare with the V-cycle with ACGC, which was the least expensive one for these tests. We can observe, that here the effect of the new strategy is not so drastic as in the case of Jacobi smoothing. Only on level 10, the global solution scheme can actually benefit from the strategy and outperform the pure V-cycle with ACGC. This is due to the fact that the overall problem needs comparatively many iterations such that the impact of the local solves in the very first global iteration is not as large as in the Jacobi tests.

The idea of varying the local multigrid cycle in the course of the global iteration is promising, though, but the question is how it behaves in other situations. For example, in the next sections



(a) Isotropic operator $-\Delta u$. **(b)** Anisotropic operator $-10\partial_{xx}u - \partial_{yy}u$.

Figure 2.29: Total arithmetic efficiency of the 2-layer-ScaRC solver in Listing 2.10 to solve the global systems, minimal level 1.

we will examine the global multigrid scheme not as a solver but as a preconditioner for an outer Krylov solver. It is not clear how the degree of difficulty of the local multigrid solves varies then. The situation is even more unclear when we use ScaRC solvers as preconditioner for treating multivariate (possibly nonlinear) solid mechanical problems (see Chapters 4 and 5). Do we then still have use the W-cycle only in the very first local solves? Or in the first local solves per Krylov-step, or per iteration of the CSM solver or per nonlinear iteration, respectively? Another question is how the initial conditions (boundary conditions, RHS, etc.) influence the local multigrid solvers.

Especially in view of these open questions, there are many further studies necessary to satisfyingly assess the strategy of varying the local multigrid cycles in the course of the global iteration. Such studies, however, are beyond the scope of this thesis and will be part of future work. As long as the effect of the varying performance of the local solves is not fully understood, we will ignore this strategy and stick to the non-varying local multigrid cycles examined in the previous sections.

2.6.1.6 Summary

We want to briefly summarise the obtained results about the performance of multigrid solvers on subdomains with extended Dirichlet boundary conditions:

- The standard V- and F-cycle and the V-cycle with modified grid transfer operations are generally unsuited.
- The F-cycle with modified grid transfer operations is not suited in connection with the Jacobi smoother. When using ADITriGS, the method still fails for more complicated operators as they arise, for instance, in solid mechanics.

- The W-cycle with modified grid transfer operations does not diverge in connection with ADITriGS, but it is unreliable when using Jacobi.
- The standard W-cycle converges in all cases, though, but it also shows strange oscillations depending on the number of grid levels.
- V-, F- and W-cycle with ACGC converge in all cases.
- The performance of the V-cycle with ACGC and Jacobi as smoother deteriorates with increasing grid level. With ADITriGS, however, this dependency is weakened or even vanishes completely.
- In terms of total arithmetic efficiency, the F-cycle with ACGC seems to be the best choice in case Jacobi is used as smoother. In case of ADITriGS, the V-cycle with ACGC is clearly favourable.

Especially motivated by the last item in this list, we will from now on stick to the following rules for the local multigrid scheme within a 2-layer-ScaRC solver:

1. *In case of Jacobi as elementary smoother, we use the F-cycle with ACGC.*
2. *In case of ADITriGS as elementary smoother, we use the V-cycle with ACGC.*

Of course, there are configurations where one of the other local schemes will perform better (e. g., the standard W-cycle in Figure 2.21b). But overviewing the results we obtained on the different configurations suggests that the two rules are a good compromise. In forthcoming numerical tests we will examine further aspects of multilevel solvers. Sticking to the two rules above greatly helps to keep the amount of possible parameter combinations manageable and presentable.

2.6.2 Truncation of Multigrid Cycles

All the computations we perform are based on a macro decomposition of the domain which is then regularly refined several times (cf. Section 2.3). Multigrid solvers act on this hierarchy of nested grids. If possible, the size of the coarsest grid problem should be small enough such that its solution (by UMFPACK or some Krylov solver) does not dominate the overall solution time. Usually, the level 1 grid represents the coarsest grid in the multigrid hierarchy. However, when the corresponding coarse grid problem is very small, FEAST offers the possibility to define the grid on a higher level to be the coarsest grid in the hierarchy, such that the coarser grids are completely ignored during the computation. We used this functionality in the tests of the previous section where we varied the minimal multigrid level.

In domains that consist of many subdomains, however, this functionality is not very useful. The smallest possible problem size per subdomain is $n_{\text{loc}}^{(1)} = 9$ unknowns (level 1 grid consisting of four elements), i. e., the size of a global coarse grid problem consisting of $M \times M$ subdomains is given by $n^{(1)} = (2M+1)^2$. When the grid consists of, for instance, $M \times M = 64^2 = 4096$

subdomains, the coarse grid system already has a size of $n^{(1)} = 129^2 = 16641$ unknowns, which is clearly efficiently solvable with a direct solver. Declaring the level 2 grid as coarsest grid is, however, already questionable since the direct solver would then have to deal with a system size of $n^{(2)} = 257^2 = 66049$ unknowns. This means, depending on the number of subdomains one may be forced to actually go down to level 1 in order to efficiently solve the resulting coarse grid problem.

Unfortunately, this means that in a 2-layer-ScaRC solver also the local multigrid schemes go down to the level 1 grid, where then coarse grid problems consisting of 9 unknowns have to be solved. Treating such tiny amounts of data clearly results in an unfavourable ratio between the time that is actually spent on arithmetic work and the time that is spent on data organisation and code instructions like function calls. Hence, it makes sense to cut off the restriction process of the local multigrid solvers already on a higher grid level, a technique we call *truncation of the local multigrid cycle*. In Figure 2.30 we schematically depict global 5-grid V-cycles (solid lines) that use local V-cycle multigrid for smoothing (dashed lines). In the top figure,

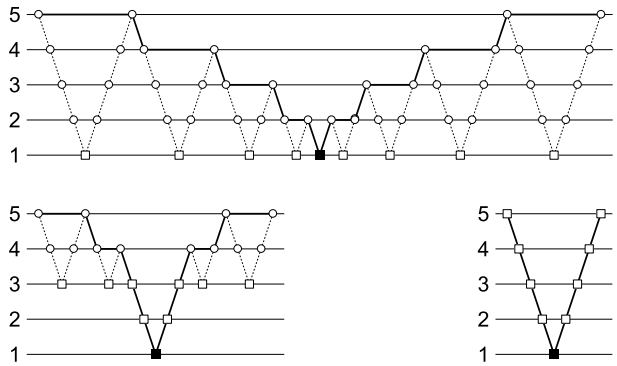


Figure 2.30: Schematic representation of truncation of local multigrid cycles: No truncation (top figure), truncation on level 3 (bottom left), truncation on level 5 (bottom right). Solid lines represent the global V-cycle, dashed lines local V-cycles. The black square stands for the solution of the global coarse grid problem on level 1, while white squares represent the local coarse grid solves. White circles indicate local smoothing (Jacobi or ADITriGS).

no truncation is applied (the truncation level is 1, respectively), such that each local V-cycle goes down to level 1 and applies the coarse grid solver there (white squares). In the bottom left figure, the truncation level is 3. Consequently, when the local smoothers are called on levels lower than or equal to 3, no local multigrid cycle is applied, but the solver which usually works as coarse grid solver of the local multigrid is started directly. This especially means, that local LU decompositions for direct solution methods do not only have to be prepared for level 3, but also for level 2. In the extreme case, the truncation level equals the maximal level (bottom right figure). Then, actually no local multigrid is performed but all local systems are directly solved by the corresponding coarse grid solver. This method then exactly represents what is usually referred to as classical multilevel Schwarz method, in which the local systems are treated by singlegrid solvers like UMFPACK or Krylov methods (cf. Section 2.5.3.2). Instead of setting

the truncation level equal to the maximal level, one could also directly choose, e. g., UMFPACK for local solving as depicted in Listing 2.6 on page 44.

Truncating local multigrid cycles on some level l, $1 < l \leqslant L$, can have further advantages: The convergence behaviour of the multigrid method usually improves since the problem on level l is already solved exactly instead of only approximately by means of a multigrid cycle starting on this level. This improvement is expected to be significant when the multigrid solver is not optimally suited for the problem at hand, e. g., when on an anisotropic grid Jacobi is used as smoother. Furthermore, by omitting the coarsest grid levels $1,\ldots,l-1$, the special problem of extended Dirichlet boundary conditions, i. e., the increasing subdomain sizes on coarser grid levels (see previous section), is automatically alleviated. On the other hand, using UMFPACK as coarse grid solver on finer grid levels bears the already discussed disadvantages, i. e., the increasing storage requirements (see Section 2.5.3.2) and the additional time needed for computing the LU decompositions and for converting data formats (see page 57). We will perform some comparative tests in Section 2.6.4 that show the possible benefit of truncating multigrid cycles.

2.6.3 Using Krylov Methods to enhance ScaRC

The idea to improve multigrid performance by employing Krylov methods has been applied by many authors (see, e. g., Elman et al. [64] and the references therein). We distinguish between two variants: On the one hand, multigrid as preconditioner of a Krylov method, and, on the other hand, a Krylov method as smoother of multigrid.[15]

2.6.3.1 Multigrid as Preconditioner

The most common way to combine multigrid and Krylov schemes is to use multigrid not as stand-alone solver but as preconditioner in an outer Krylov iteration. This may help to improve the robustness, especially when the multigrid iteration is not guaranteed to be a convergent process. However, when the multigrid method itself is already robust and efficient, then the embedding into an outer Krylov method usually does not significantly improve the overall performance. Kilian [94] and Becker [16] successfully apply this technique to improve ScaRC's convergence behaviour on certain configurations. They usually perform one ScaRC iteration to precondition an outer CG or BiCGstab method.

To illustrate the mechanism, we consider a standard 2-layer-ScaRC solver (see Listing 2.11) and embed it as preconditioner into an outer BiCGstab solver (see Listing 2.12). The term '`prec=GLOBAL`' indicates that a solver working on the global layer is used as preconditioner.

[15] Pflaum [128] introduces another approach to combine multigrid and Krylov techniques: The orthogonalisation procedure is applied to the space of restricted residuals, i. e., on each level an orthogonal correction direction is computed. This technique is shown to be superior for some Poisson problems with jumping coefficients. However, in this thesis we do not consider this approach.

```
solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:4:4
smoother=LOCAL, damp=0.8
  solver=MG, maxiter=64, tol=REL:5e-1, cycle=F:4:4, cgcdamp=ADAP
  smoother=JACOBI, damp=0.7
  coarse=UMFPACK
coarse=UMFPACK
```

Listing 2.11: Standard 2-layer-ScaRC solver `MG__MG-JAC-D__D` (operations on the local layer highlighted in grey).

```
solver=BICG, maxiter=64, tol=REL:1e-6
prec=GLOBAL, damp=1.0
  solver=MG, maxiter=1, tol=IGNORE, cycle=V:4:4
  smoother=LOCAL, damp=0.8
    solver=MG, maxiter=64, tol=REL:5e-1, cycle=F:4:4, cgcdamp=ADAP
    smoother=JACOBI, damp=0.7
    coarse=UMFPACK
  coarse=UMFPACK
```

Listing 2.12: Standard 2-layer-ScaRC as preconditioner of BiCGstab (`BICG-MG__MG-JAC-D__D`).

The notation '`tol=IGNORE`' means that no convergence control is necessary since a fixed number of iterations is performed. Applying the rules listed in Section 2.5.5.2, the solver in Listing 2.12 reads in short notation

`BICG(1e-6)-MG(1,V44,0.8)__MG(F44,5e-1,0.7)-JAC-D__D,`

and in minimalistic notation

`BICG-MG__MG-JAC-D__D.`

Note that BiCGstab and the outer MG are acting on the same layer and are thus separated by a single hyphen '`-`' instead of two underscores.

Until now, the strategy to use multigrid not as solver, but as preconditioner was restricted to the global solver in FEAST. However, it is feasible and makes sense to apply the same idea also to the *local* multigrid scheme within a 2-layer-ScaRC solver. We implemented this feature such that FEAST now also offers solver definitions as the one in Listing 2.13.

```
solver=BICG, maxiter=64, tol=REL:1e-6
prec=GLOBAL, damp=1.0
  solver=MG, maxiter=1, tol=IGNORE, cycle=V:4:4
  smoother=LOCAL, damp=0.8
    solver=BICG, maxiter=64, tol=REL:5e-1
    prec=LOCAL, damp=1.0
      solver=MG, maxiter=1, tol=IGNORE, cycle=F:4:4, cgcdamp=ADAP
      smoother=JACOBI, damp=0.7
      coarse=UMFPACK
  coarse=UMFPACK
```

Listing 2.13: Embedding the global and the local multigrid scheme as preconditioners into BiCGstab solvers (`BICG-MG__BICG-MG-JAC-D__D`).

2.6.3.2 Krylov as Smoother

Another possibility to enhance multigrid with the help of Krylov methods is to use the latter as smoothers. To illustrate how this works, we again consider the standard 2-layer-ScaRC solver in Listing 2.11. In Section 2.5.5 we explained that the smoothing process is actually realised by a preconditioned Richardson iteration (see Listing 2.2 on page 41). So, we can interpret this process as application of a 'stand-alone' solver and write the ScaRC scheme correspondingly in the equivalent form depicted in Listing 2.14. So, the Richardson solver takes the role of

```
1  solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:1:1
2  smoother=GLOBAL, damp=1.0
3    solver=RICH, maxiter=4, tol=IGNORE
4    prec=LOCAL, damp=0.8
5      solver=MG, maxiter=64, tol=REL:5e-1, cycle=F:1:1, cgcdamp=ADAP
6      smoother=LOCAL, damp=1.0
7        solver=RICH, maxiter=4, tol=IGNORE
8        prec=JACOBI, damp=0.7
9      coarse=UMFPACK
10 coarse=UMFPACK
```

Listing 2.14: Standard 2-layer-ScaRC solver, smoothing process written as Richardson iteration (`MG-RICH__MG-RICH-JAC-D__D`).

the multigrid smoother, and what has been the multigrid smoother becomes the preconditioner for the Richardson solver. Correspondingly, the damping parameter for smoothing turns into the damping parameter for the preconditioner within the Richardson iteration. The damping parameter for smoothing can consequently be ignored, i. e., set to 1.0. Note that the `V:4:4` (`F:4:4`) cycle in the simple notation in Listing 2.11 turned into an `V:1:1` (`F:1:1`) cycle and the instruction to perform 4 smoothing steps is now contained in the definition of the Richardson solver ('`maxiter=4`'). In short notation the solver can be written as

```
MG(V11)-RICH(4,0.8)__MG(F11)-RICH(4,0.7)-JAC-D__D.
```

Inflating the notation of the standard 2-layer-ScaRC solver like this does, of course, not make sense on its own. But it is perfectly suited to explain how Krylov solvers are employed as multigrid smoothers. Actually, we only have to replace the Richardson solvers in Listing 2.14 by the desired Krylov solver. When, for instance, the global multigrid scheme is to be smoothed by FGMRes and the local multigrid scheme by BiCGstab, this is realised by the solver scheme depicted in Listing 2.15 which in short notation reads

```
MG(V11)-FGMRES4(4)__MG(F11)-BICG(2)-JAC-D__D.
```

It is well known that the Jacobi smoother of the inner multigrid in Listing 2.14 is extremely sensitive with respect to the damping parameter. Its optimal value depends on several factors (e. g., the degree of mesh anisotropy), it can hardly be set automatically a priori, but has to be adjusted manually by the user. When the value is chosen too large, divergence of the overall method is likely. A similar sensitivity has to be expected with respect to the *global* damping parameter in Listing 2.14: In Section 2.5.2 we explained that the Schwarz preconditioner within one level works *additively*, i. e., it exhibits a *block-Jacobi* character. Since the Schwarz preconditioner

```
1  solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:1:1
2  smoother=GLOBAL, damp=1.0
3    solver=FGMRES, maxiter=4, tol=IGNORE, restart=4
4    prec=LOCAL, damp=1.0
5      solver=MG, maxiter=64, tol=REL:5e-1, cycle=F:1:1, cgcdamp=ADAP
6      smoother=LOCAL, damp=1.0
7        solver=BICG, maxiter=2, tol=IGNORE
8        prec=JACOBI, damp=1.0
9      coarse=UMFPACK
10 coarse=UMFPACK
```

Listing 2.15: 2-layer-ScaRC solver with global multigrid scheme smoothed by FGMRes and the local one by BiCGstab (`MG-FGMRES4__MG-BICG-JAC-D__D`).

is integrated in terms of a Richardson process, a corresponding sensitivity with respect to the damping parameter has to be expected.

Comparing Listings 2.14 and 2.15, one can see the crucial advantage of the Krylov smoothers: *There are no damping parameters to adjust.*[16] It is a typical feature of Krylov methods like CG, BiCGstab and GMRes to implicitly compute optimal step lengths for the update vector obtained in the preconditioning step. The process is similar to the computation of the adaptive coarse grid correction in equation (2.13) on page 54. We will see in the following numerical tests that Krylov smoothers indeed significantly increase the robustness of the overall solution algorithm.

With ADITriGS as smoother the local damping parameter can be set to 1.0 anyway (see Section 2.5.3.1), so we do not expect significant improvements when using a Krylov smoother in this situation. However, in a 2-layer-ScaRC scheme we still have to deal with the *global* damping parameter such that a Krylov smoother on the global layer makes sense.

When using Krylov methods as smoothers, one cannot in general expect an improvement of the multigrid convergence, since the smoothing property of Krylov solvers is not necessarily better than that of the Richardson scheme. However, Krylov solvers can lead to more robust multigrid methods in the sense that they are able to prevent divergence of the smoothing process.

The employment of Krylov schemes is actually standard in the context of domain decomposition methods. In practice, 1-level additive Schwarz preconditioners are never embedded into a global Richardson solver, since the resulting solver is not guaranteed to converge [150]. Instead, suitable Krylov methods are used to provide the global coupling. Considering 2-layer-ScaRC solvers as multilevel domain decomposition methods, we simply apply the strategy used for 1-level Schwarz methods *on each level*.

[16] The notation of the 4 damping parameters in Listing 2.15 can just as well be omitted completely.

2.6.3.3 Combining both Strategies

We will see in the numerical tests below that both strategies of combining multigrid and Krylov methods have the ability to improve the efficiency of the overall solution method. So, an obvious idea is to use both strategies at the same time, i. e., an outermost Krylov method is preconditioned by a multigrid scheme which is smoothed on each level by another Krylov scheme. Such techniques have, for example, been employed by Köster [100] to improve standard multigrid techniques. In FEAST we can go a step further and apply this idea to both the local and the global layer, such that we end up with quite complex solver definitions as the one in Listing 2.16. The question is, of course, whether or not the positive effects of both strategies

```
1  solver=BICG, maxiter=64, tol=REL:1e-6
2  prec=GLOBAL, damp=1.0
3    solver=MG, maxiter=1, tol=IGNORE, cycle=V:1:1
4    smoother=GLOBAL, damp=1.0
5      solver=FGMRES, maxiter=2, tol=IGNORE, restart=2
6      prec=LOCAL, damp=1.0
7        solver=BICG, maxiter=64, tol=REL:5e-1
8        prec=LOCAL, damp=1.0
9          solver=MG, maxiter=1, tol=IGNORE, cycle=F:1:1, cgcdamp=ADAP
10         smoother=LOCAL, damp=1.0
11           solver=BICG, maxiter=2, tol=IGNORE
12           prec=JACOBI, damp=1.0
13         coarse=UMFPACK
14   coarse=UMFPACK
```

Listing 2.16: Exemplary 2-layer-ScaRC solver with both multigrid schemes used as preconditioner of and smoothed by Krylov schemes (`BICG-MG-FGMRES__BICG-MG-BICG-JAC-D__D`).

actually accumulate and thus further improve the overall solver efficiency, especially in view of the enormous storage requirements and arithmetic work per iteration. We will treat this issue in the following numerical tests.

2.6.3.4 Suitable Krylov Methods

The classical representative of Krylov space methods is the *Conjugate Gradient* (CG) method by Hestenes and Stiefel [79]. It is applicable to symmetric positive definite systems only. The main difference compared to elementary iterative methods like Jacobi or Gauß-Seidel is that CG minimises the error between the approximate $\mathbf{x}^k$ and the real solution $\mathbf{x}^*$ over a growing subspace, the *Krylov space*

$$\mathcal{K}^k := \text{span}\{\mathbf{r}^0, \mathbf{A}\mathbf{r}^0, \ldots, \mathbf{A}^{k-1}\mathbf{r}^0\},$$

where $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$ is the initial residual. More precisely, we have

$$\|\mathbf{x}^k - \mathbf{x}^*\|_{\mathbf{A}} = \min_{\mathbf{x} \in \mathcal{K}^k} \|\mathbf{x} - \mathbf{x}^*\|_{\mathbf{A}}$$

with $\|\cdot\|_{\mathbf{A}}$ being the norm induced by the inner product $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}$. Thus, when neglecting roundoff errors, convergence within n steps (n being the problem size) is guaranteed. In real-life simulations where n is large, this is actually of no practical relevance. Nevertheless, the method can be very efficient when provided with a good preconditioner.

Based on the CG method, many further Krylov methods have been developed for various kinds of linear systems. A recent overview can be found in Simoncini and Szyld [149]. For implementational considerations, see Barrett et al. [13]. We confine ourselves to Krylov solvers that have the ability to

- solve nonsymmetric systems,
- do this without the transpose of the system matrix,
- employ variable preconditioners.

Even if the system matrix is symmetric, the application of a nonsymmetric preconditioner (e. g., ADITriGS) forbids the usage of standard CG. Variable preconditioners are, for example, inner solvers that gain one digit instead of performing a fixed number of iterations. Also local multigrid solvers that employ adaptive coarse grid correction (see Section 2.6.1.3) have to be considered as variable preconditioners. As representatives of more general Krylov space methods that fulfil the desired requirements we only consider BiCGstab [167] and FGMRes [136] (the flexible version of GMRes [137]), probably the most widely used iterative methods for nonsymmetric and/or indefinite systems. Both schemes are briefly described now in terms of their advantages and disadvantages.

GMRes [137] (Generalised Minimal Residual) minimises the residual in the standard l_2-norm over the Krylov subspace, i. e.,

$$\|\mathbf{r}^k\| = \min_{\mathbf{x}\in\mathcal{K}^k} \|\mathbf{f}-\mathbf{A}\mathbf{x}\|.$$

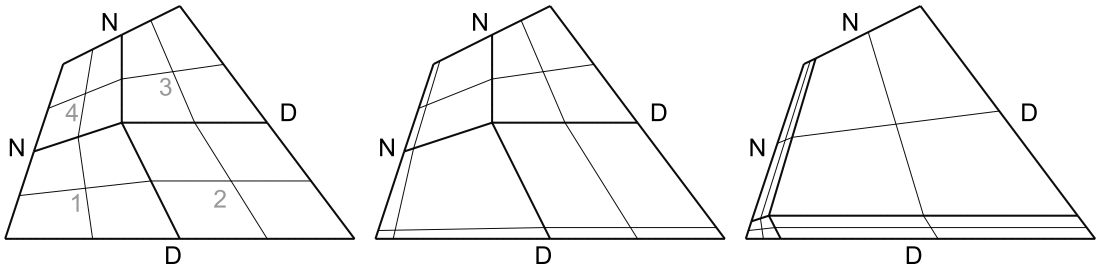
To guarantee this property, the basis of the growing space $\mathcal{K}^k$ has to be stored which becomes prohibitively expensive (array of size $n\cdot k$) for increasing iteration numbers k and large problem sizes n. One remedy to overcome this is to use a *restarted* version, GMRes(m). The idea is to omit the Krylov basis after m iterations and restart the process based on the current iterate and residual. Unfortunately, convergence of the process is no longer guaranteed then. Furthermore, the user has to adjust yet another parameter, the maximum Krylov subspace dimension m. See Simoncini and Szyld [149] for an overview of more sophisticated restarting strategies and alternatives to limit the memory consumption. A further disadvantage of GMRes(m) is that it is only applicable for fixed preconditioners. However, once implemented, it is simply extendable to deal with variable preconditioners, leading to the *flexible* version FGMRes [136]. The flexibility comes with the cost of roughly doubled memory consumption which again has to be circumvented by using a restarted version, FGMRes(m). GMRes(m) needs $m+3$ auxiliary vectors, while FGMRes needs $2m+3$ auxiliary vectors. Hence, when m is large the storage costs are enormous compared to a standard multigrid solver which needs about 3 auxiliary vectors. For this reason we employ FGMRes(m) only as inner Krylov smoother, where we perform a fixed number of iterations.

As outermost Krylov solver we use BiCGstab (Bi-Conjugate Gradient stabilised), which can be viewed as an extension of BiCG and CGS (Conjugate Gradient Squared). From these three methods, which are all able to deal with nonsymmetric matrices, BiCGstab is considered to be the most robust one [92]. However, there is no convergence theory available and the algorithm can break down in certain situations. The norm of the residual is known to oscillate although these oscillations are significantly weakened compared to the other two methods (which explains the term 'stabilised'). The crucial advantage of BiCGstab is that by exploiting two coupled short-term recurrences, only seven auxiliary vectors are necessary, i. e., the memory consumption does not increase as the iteration proceeds. This renders the method suitable as outermost solver, for which the number of iterations is not known a priori. The implementation of BiCGstab is very similar to that of CG, 'non-standard operations' like solving a least squares problem as in GMRes do not have to be provided. BiCGstab is able to deal with varying preconditioners (see Vogel [170]).[17]

Beside the additional storage requirements, a further disadvantage of *global* Krylov methods has to be seen in the significant increase of communication in parallel computations. Especially, the calculation of scalar products and norms is expensive as it requires *global communication* (see Kilian [94, Section A.2.2, p. 221]). The following numerical tests, however, show that it is justified to employ global Krylov schemes despite these disadvantages.

2.6.3.5 Numerical Tests

In order to avoid special convergence effects which may arise on purely isotropic unitsquare configurations with isotropic operators, we use the distorted geometry in Figure 2.31. The domain consists of four subdomains (refinement level 1 displayed), and is partitioned/refined in four different ways:



(a) Isotropic macro decomp., isotropic refinement (ISO/ISO).

(b) Isotropic macro decomp., anisotropic refinement (ISO/ANISO).

(c) Anisotropic macro decomp., isotropic refinement (ANISO/ISO).

Figure 2.31: Distorted test grid consisting of four subdomains, three different decompositions/refinements, level 1 displayed.

[17] Notay [118] examines a flexible version of CG that is able to deal with variable preconditioners. The method, however, lacks the key property of short-term recurrence, such that the storage requirements are, in principle, the same as for GMRes.

1. *Isotropic macro decomposition, isotropic refinement* (ISO/ISO, see Figure 2.31a). The maximal element aspect ratio is $\sigma \approx 2.2$ (cf. equation (2.5) on page 28).

2. *Isotropic macro decomposition, anisotropic refinement* (ISO/ANISO, see Figure 2.31b). Subdomains 1 and 2 are anisotropically refined towards the bottom, subdomains 1 and 4 are anisotropically refined towards the left with the anisotropy factors $a_F = 0.2, a_L = a_I = 1.0$ (cf. equation (2.4) on page 27). Subdomain 3 is isotropically refined. The maximal element aspect ratio is $\sigma \approx 17.0$.

3. *Anisotropic macro decomposition, isotropic refinement* (ANISO/ISO, see Figure 2.31c). The inner point and the central boundary points of the macro decomposition are moved to introduce macro anisotropies. The subdomains are refined isotropically. The maximal element aspect ratio is $\sigma \approx 12.7$.

4. *Anisotropic macro decomposition, anisotropic refinement* (ANISO/ANISO, not displayed in Figure 2.31). The same macro decomposition as in Figure 2.31c, combined with the anisotropic refinement of Figure 2.31b. The maximal element aspect ratio is $\sigma \approx 63.6$.

To further aggravate the problem configuration, we use the equation

$$-\partial_{xx}u - 4\partial_{yy}u = 1 \tag{2.16}$$

instead of the isotropic Poisson operator. Bottom and right side of the domain exhibit zero Dirichlet boundary conditions, while zero Neumann boundary conditions are applied to top and left side (see Figure 2.31).

In view of the possible complexity of the 2-layer-ScaRC solvers, it is impossible to examine the whole parameter space. We performed excessive numerical studies on different configuration (which are not presented here) in order to confine ourselves to a manageable subset of parameter combinations. We use the following settings:

- When no Krylov smoother is used, the global multigrid scheme performs a `V:4:4` cycle.

- As global Krylov smoother we perform 4 iterations of FGMRes(4) (which are then embedded into a `V:1:1` cycle).

- When Jacobi (ADITriGS) is used as elementary smoother (without local Krylov smoothing), the local multigrid scheme performs an `F:4:4` (`V:2:2`) cycle. Remember, that in case of ADITriGS the first smoothing step performs TriGS and the second one MTriGS (cf. Section 2.5.3.1).

- In case of local Krylov smoothing with Jacobi as elementary preconditioner, we perform 2 BiCGstab smoothing iterations, embedded into a local `F:1:1` cycle. For ADITriGS no local Krylov smoothing is performed.

- The local problems are solved with a relative stopping criterion of $\varepsilon = 0.5$. This seems to be quite inaccurate, but it turned out to be a good compromise between the amount of local arithmetic work on the one hand and number of global iterations on the other hand. Gaining, e. g., one digit ($\varepsilon = 0.1$) often decreases the global number of iterations only marginally or not at all, while it significantly increases the amount of arithmetic work (see Section 2.6.4).

- The local multigrid schemes always perform the adaptive coarse grid correction (ACGC) technique (see Section 2.6.1.6).

To further reduce the amount of data, we present in a first stage only computations with minimal level 1 and maximal level 8, which means 66189 unknowns per subdomain and 263169 unknowns for the global problem.

We begin with examining the global multigrid solver within the 2-layer-ScaRC scheme. Therefore, we use a direct solver to solve the local problems in order to exclude dependencies of the results on the way how the local problems are solved. In a first test we only consider standard multigrid (see Listing 2.17) and multigrid as preconditioner for an outer BiCGstab scheme (see Listing 2.18). For both variants, the global damping parameter has to be set by the user (denoted in bold face). Figure 2.32 shows the total arithmetic efficiency of the two solver

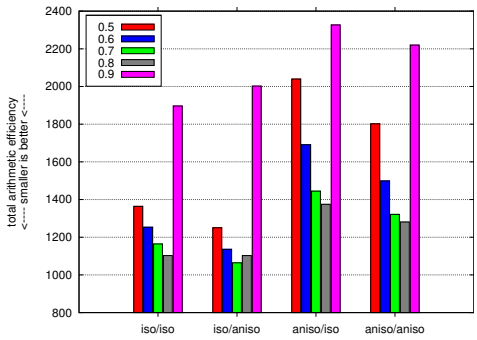
```
1 solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:4:4
2 smoother=LOCAL, damp=0.7
3   solver=UMFPACK
4 coarse=UMFPACK
```

Listing 2.17: 2-layer-ScaRC solver with UMFPACK as local solver `MG(64,1e-6,V44,0.7)__D__D` (operations on the local layer highlighted in grey).

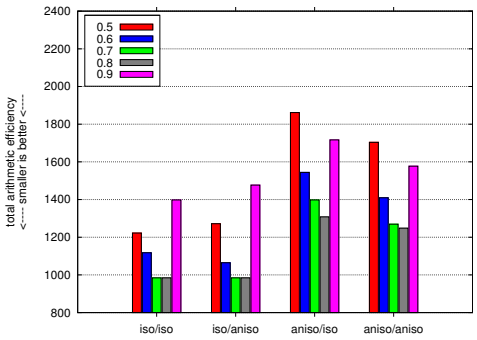
```
1 solver=BICG, maxiter=64, tol=REL:1e-6
2 prec=GLOBAL, damp=1.0
3   solver=MG, maxiter=1, tol=IGNORE, cycle=V:4:4
4   smoother=LOCAL, damp=0.7
5     solver=UMFPACK
6   coarse=UMFPACK
```

Listing 2.18: Solver of Listing 2.17 used as preconditioner of an outer BiCGstab scheme `BICG(64,1e-6)-MG(1,V44,0.7)__D__D`.

schemes on the four test grids, where the global damping parameter varies between $\omega = 0.5$ and $\omega = 0.9$. The results for $\omega = 1.0$ are not displayed since the performance deteriorated by a factor of 4–10 compared to $\omega = 0.9$. The plots show that for the ISO/ANISO configuration the damping parameter $\omega = 0.7$ is the best choice, while for the other configurations $\omega = 0.8$ yields the best results. However, the difference between the two is relatively small compared to the other values of ω. Figure 2.32b shows that the outer BiCGstab scheme clearly weakens the sensitivity with respect to the damping parameter. When considering only the best damping pa-



(a) `MG(64,1e-6,V44,ω)__D__D` (Listing 2.17).



(b) `BICG(64,1e-6)-MG(1,V44,ω)__D__D` (Listing 2.18).

Figure 2.32: Total arithmetic efficiency of two solver schemes with varying global damping parameter ω on the four test grids.

rameter per grid, one can see that the total arithmetic efficiency is slightly improved compared to Figure 2.32a.

To get rid of the necessity to adjust the global damping parameter, we now use 4 FGMRes(4) iterations as Krylov smoother. The corresponding enhanced versions of the two previous solvers are depicted in Listings 2.19 and 2.20. These enhanced solvers are now compared to those

```
1 solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:1:1
2 smoother=GLOBAL, damp=1.0
3   solver=FGMRES, maxiter=4, tol=IGNORE, restart=4
4   prec=LOCAL, damp=1.0
5     solver=UMFPACK
6 coarse=UMFPACK
```

Listing 2.19: Solver of Listing 2.17 enhanced by a Krylov smoother `MG(64,1e-6,V11)-FGMRES4(4)__D__D`.

```
1 solver=BICG, maxiter=64, tol=REL:1e-6
2 prec=GLOBAL, damp=1.0
3   solver=MG, maxiter=1, tol=IGNORE, cycle=V:1:1
4   smoother=GLOBAL, damp=1.0
5     solver=FGMRES, maxiter=4, tol=IGNORE, restart=4
6     prec=LOCAL, damp=1.0
7       solver=UMFPACK
8   coarse=UMFPACK
```

Listing 2.20: Solver of Listing 2.19 used as preconditioner of an outer BiCGstab scheme `BICG(64,1e-6)-MG(1,V11)-FGMRES4(4)__D__D`.

versions of the simple solvers, which performed best on the respective grid. The results are summarised in Figure 2.33. The most important observation is that on each of the four grids the multigrid solver using Krylov smoothing (`MG-FGMRES__D__D`) is the most efficient one. It not only frees the user from setting the global damping parameter, it also clearly outperforms the

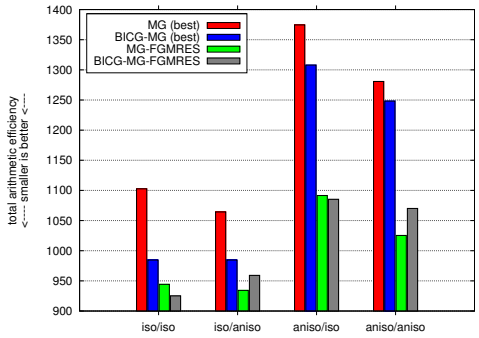


Figure 2.33: Total arithmetic efficiency of the solver schemes in Listings 2.17, 2.18 (the respective best one per grid), 2.19 and 2.20).

manually optimised multigrid solvers without Krylov smoothing. The second important observation is that additionally enclosing the solver as preconditioner into an outer BiCGstab scheme (`BICG-MG-FGMRES__D__D`), does not necessarily improve the total arithmetic efficiency – for the two configurations ISO/ISO and ISO/ANISO it does (only slightly, though), for the other two it does not.

We now examine the local multigrid solvers within the 2-layer-ScaRC scheme. Therefore, we fix the global solver to be `MG-FGMRES`. For the local solver we choose, on the one hand, a standard multigrid for which we have to adjust the damping parameter (see Listing 2.21) and, on the other hand, a multigrid smoothed by BiCGstab for which no damping parameter has to be set (see Listing 2.15 on page 74). We vary the local damping parameter in Listing 2.21

```
1 solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:1:1
2 smoother=GLOBAL, damp=1.0
3   solver=FGMRES, maxiter=4, tol=IGNORE, restart=4
4   prec=LOCAL, damp=1.0
5     solver=MG, maxiter=64, tol=REL:5e-1, cycle=F:4:4, cgcdamp=ADAP
6     smoother=JACOBI, damp=0.7
7     coarse=UMFPACK
8 coarse=UMFPACK
```

Listing 2.21: 2-layer-ScaRC solver with global multigrid scheme smoothed by FGMRes (`MG-FGMRES4(4)__MG-JAC-D__D`).

between $\omega = 0.5$ and $\omega = 1.0$. For $\omega \geqslant 0.8$ *divergence occurs on all four grids*, the solver with $\omega = 0.7$ only converges on the ISO/ISO grid. In Figure 2.34 the results of the converged solvers are compared to the solver using BiCGstab as local Krylov smoother (see Listing 2.15). One can observe the crucial advantage of the latter solver: It converges on all four grids and yields a significantly better total arithmetic efficiency on the three anisotropic grids. On the ISO/ISO grid, it is only slightly more expensive than the other solvers.

Analogously to the global solver, we additionally embedded the local multigrid scheme as preconditioner into an outer local BiCGstab solver (see the local part of the solver in Listing 2.16 on page 75). This, however, did not improve the total arithmetic efficiency of the overall solver

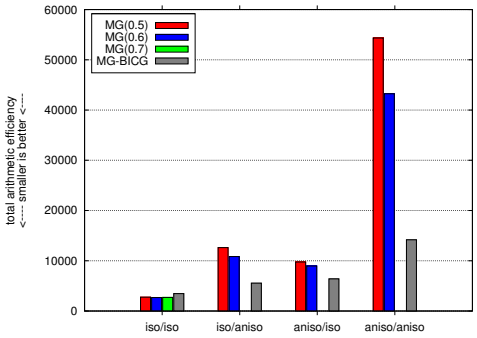


Figure 2.34: Total arithmetic efficiency of the solver schemes in Listings 2.21 and 2.15.

scheme, neither for the four configurations in Figure 2.31, nor for the more complicated configurations in Figures 2.38 and 2.39. So, we will neglect this solver configuration and confine ourselves to use as local solver a multigrid scheme smoothed by BiCGstab.

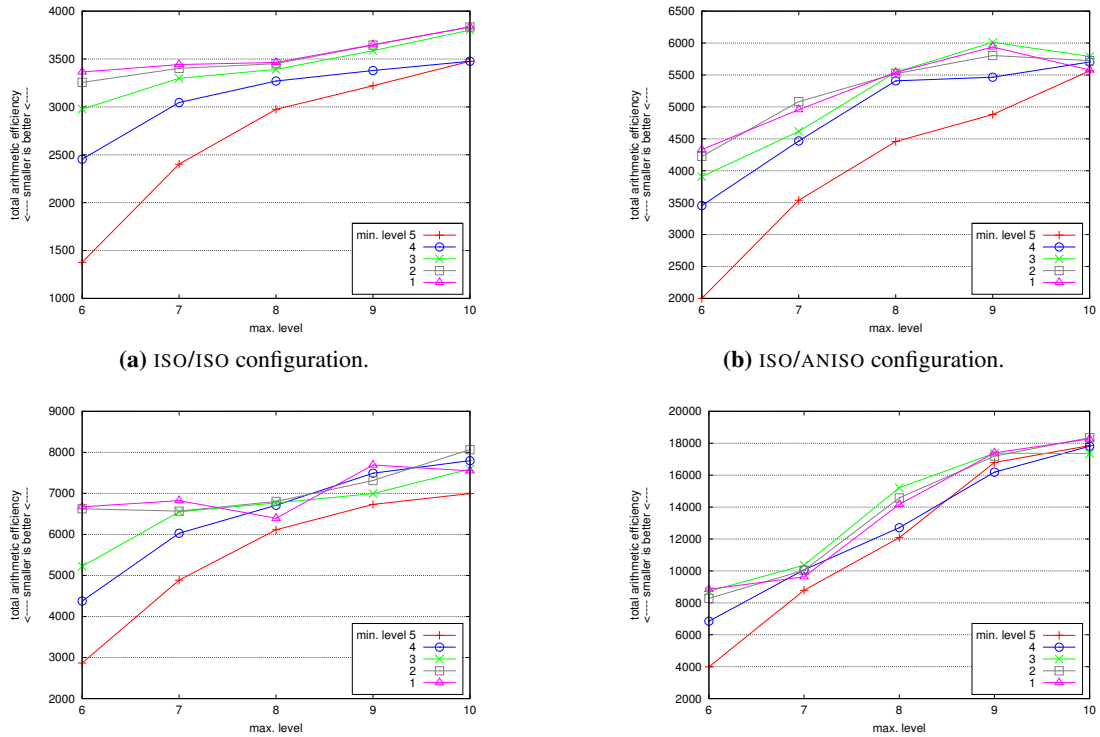
In order to get a better picture of the performance of our favourite solver `MG-FGMRES4(4)__-MG-BICG-JAC-D__D` (see Listing 2.15 on page 74), we show in Figure 2.35 results for varying minimum and maximum grid level on the four test grids. One can see that all tests converge and that the results are fairly consistent with each other. A dependence on both the minimum and the maximum multigrid level is observable, which is most prominent in case of the ANISO/ANISO configuration. This, however, is mainly due to an increase of local work – the global number of iterations is constant or rises only mildly within one configuration. For example, on the ISO/ISO configuration all tests need exactly 3 iterations, and on the ANISO/ANISO configuration the number of iterations increases from 4 on level 6 to 5 on level 10 (for all minimum levels). The increase of local work is a consequence of the extended Dirichlet boundary conditions (cf. Sections 2.4 and 2.6.1.1) and of the grid/operator anisotropies which are quite a challenge for the simple Jacobi smoother (cf. the following ADITriGS tests).

For sake of completeness, we repeat the previous tests with ADITriGS as elementary smoother. The results are as expected: A damping parameter of $\omega = 1.0$ usually yields the best

```
1 solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:4:4
2 smoother=LOCAL, damp=0.7
3   solver=MG, maxiter=64, tol=REL:5e-1, cycle=V:2:2, cgcdamp=ADAP
4   smoother=ADITRIGS, damp=1.0
5   coarse=UMFPACK
6 coarse=UMFPACK
```

Listing 2.22: 2-layer-ScaRC solver with ADITriGS as elementary smoother (`MG(V44,0.7)__MG-ADI-D__D`).

results (only in rare cases, $\omega = 0.9$ leads to marginally better results). Consequently, the usage of BiCGstab as local Krylov smoother does not improve the performance. However, *global* Krylov smoothing is beneficial as Figure 2.36 demonstrates. It shows the results of the solver in Listing 2.22 with varying global damping parameter and of the solver in Listing 2.23 on the four configurations in Figure 2.31. Similar to the Jacobi case, the simple multigrid solver



(a) ISO/ISO configuration.

(b) ISO/ANISO configuration.

(c) ANISO/ISO configuration.

(d) ANISO/ANISO configuration.

Figure 2.35: Total arithmetic efficiency of the solver `MG-FGMRES4__MG-BICG-JAC-D__D` (see Listing 2.15 on page 74) on the four test grids with varying multigrid levels. (Note the different y-scales.)

```
1 solver=MG, maxiter=64, tol=REL:1e-6, cycle=V:1:1
2 smoother=GLOBAL, damp=1.0
3   solver=FGMRES, maxiter=4, tol=IGNORE, restart=4
4   prec=LOCAL, damp=1.0
5     solver=MG, maxiter=64, tol=REL:5e-1, cycle=V:2:2, cgcdamp=ADAP
6     smoother=ADITRIGS, damp=1.0
7     coarse=UMFPACK
8 coarse=UMFPACK
```

Listing 2.23: 2-layer-ScaRC solver with global multigrid scheme smoothed by FGMRes and ADITriGS as elementary smoother (`MG-FGMRES4__MG-ADI-D__D`).

shows strong variations with respect to the global damping parameter, and the solver using Krylov smoothing performs best on each configuration. The additional enhancement by using the multigrid schemes as preconditioner for BiCGstab does not yield any performance gain on the four test grids (results not shown here). In Figure 2.37 we see the results of the solver `MG-FGMRES4__MG-ADI-D__D` (see Listing 2.23) for varying minimum and maximum grid levels on the four test configurations. One can see that ADITriGS behaves much more robust with respect to the anisotropies. While the total arithmetic efficiency in case of Jacobi ranges from 1300 to 18500, the ADITriGS results lie between 850 and 2500. Furthermore the level dependency is

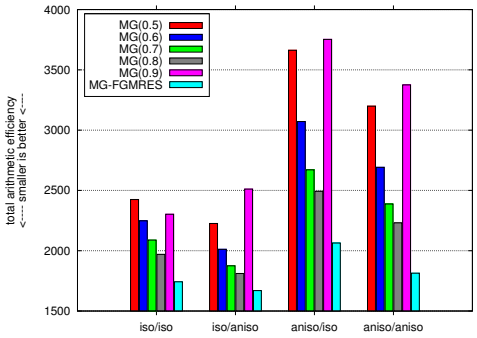
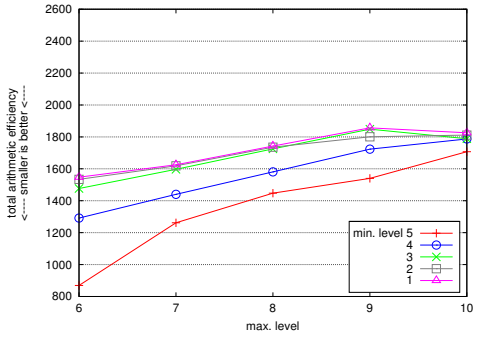
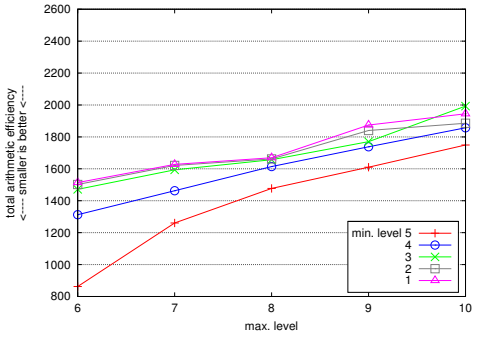


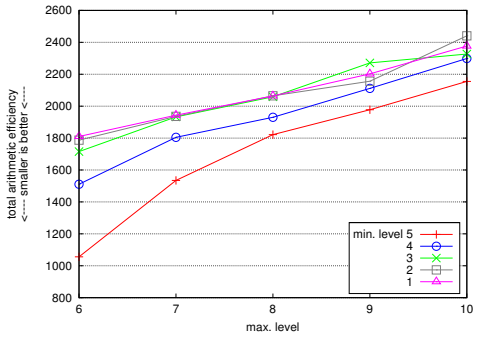
Figure 2.36: Total arithmetic efficiency of the solver schemes `MG(V44,ω)__MG-ADI-D__D` and `MG-FGMRES4__MG-ADI-D__D` (see Listings 2.22 and 2.23).



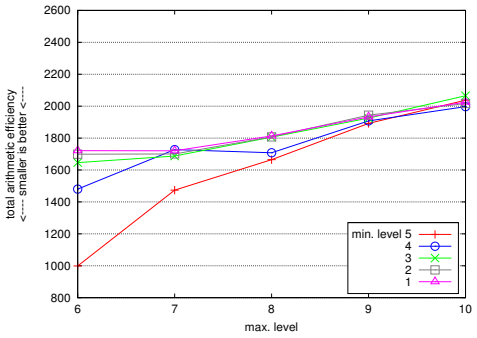
(a) ISO/ISO configuration.



(b) ISO/ANISO configuration.



(c) ANISO/ISO configuration.

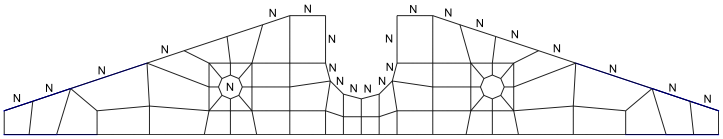


(d) ANISO/ANISO configuration.

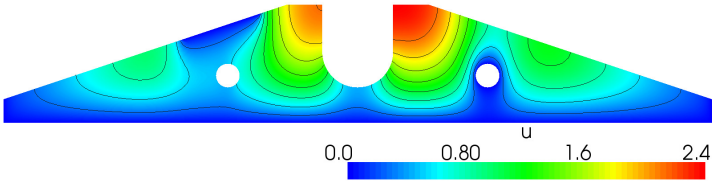
Figure 2.37: Total arithmetic efficiency of the solver `MG-FGMRES4__MG-ADI-D__D` (see Listing 2.23) on the four test grids with varying multigrid levels.

much weaker in case of ADITriGS and, again, mainly driven by an increase of local work: The global iteration numbers are between 3 and 4 for all tests.

To further demonstrate the strength of the Krylov-smoothed ScaRC solvers, we will now consider two more complicated geometries: On the one hand, the 2D cross section of a crossover



(a) Macro decomposition (level 0) consisting of 64 subdomains. The boundary segments with zero Neumann boundary conditions are labelled 'N', the remainder of the boundary exhibits zero Dirichlet boundary conditions.

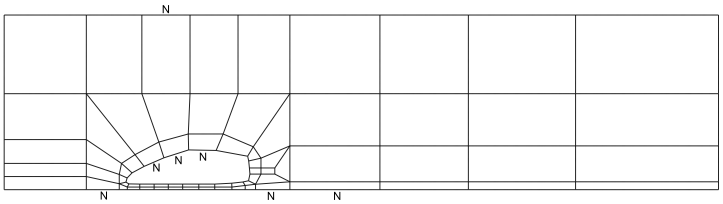


(b) Computed solution.

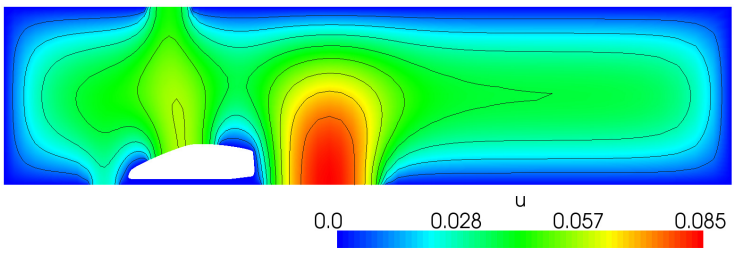
Figure 2.38: CROSSOVER configuration.

which is used to slow down street traffic, on the other hand, the so called ASMO configuration which represents a car shape in a wind tunnel. The former configuration is used in Wobker and Turek [175], the latter configuration in Kilian [94] and Becker [16]. Figures 2.38a and 2.39a display the macro decompositions and the arbitrarily chosen boundary conditions of the two configurations. The computed solutions of the anisotropic equation (2.16) on these two configurations are shown in Figures 2.38b and 2.39b. For our numerical tests we choose for both configurations three different kinds of grid refinement (see Table 2.2 for detailed statistics and equation (2.4) on page 27 for the definition of the anisotropy factors):

1. Isotropic refinement with anisotropy factors $a_F = a_L = a_I = 1.0$ (denoted as ISO).



(a) Macro decomposition (level 0) consisting of 70 subdomains.



(b) Computed solution.

Figure 2.39: ASMO configuration.

2. Refinement with anisotropy factors $a_F = 0.05, a_L = a_I = 1.0$ (denoted as ANISO1). In the CROSSOVER configuration all subdomains around the holes are anisotropically refined towards the holes, in the ASMO configuration the subdomains are anisotropically refined towards the upper and lower boundary of the channel and towards the car shape.

3. Refinement with anisotropy factors $a_F = 0.25, a_L = 0.35, a_I = 0.5$ (denoted as ANISO2). The anisotropic refinement is applied to the same subdomains as in the case of the ANISO1 refinement. Note, that the ANISO2 refinement leads to level dependent maximum element aspect ratios with maximum values of $\sigma \approx 12200$ in the case of the CROSSOVER configuration and $\sigma \approx 226000$ in the case of the ASMO configuration.

	CROSSOVER (64 subdomains)				ASMO (70 subdomains)			
		max. AR				max. AR		
lev	#DOF	ISO	ANISO1	ANISO2	#DOF	ISO	ANISO1	ANISO2
5	6.66e+4	2.91	20.4	1.83e+2	7.26e+4	18.2	3.64e+2	3.39e+3
6	2.64e+5			5.23e+2	2.89e+5			9.69e+3
7	1.05e+6			1.50e+3	1.15e+6			2.77e+4
8	4.20e+6			4.27e+3	4.59e+6			7.91e+4
9	1.68e+7			1.22e+4	1.84e+7			2.26e+5

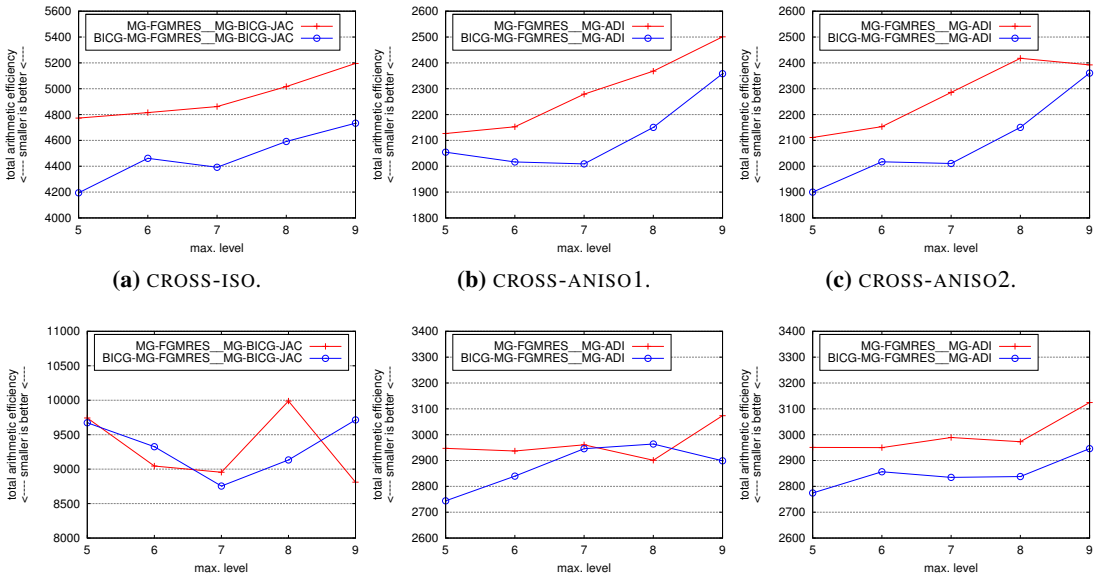
Table 2.2: Number of DOF and maximal element aspect ratios (AR) of the different CROSSOVER and ASMO configurations.

On the two isotropic configurations we use the 2-layer-ScaRC solver with Jacobi as smoother (see Listing 2.15), while ADITriGS is employed for the four anisotropic configurations (see solver in Listing 2.23). We furthermore embed the two solvers as preconditioners into an outer BiCGstab scheme and compare the results with the help of the six plots in Figure 2.40. We vary the maximum grid level from 5 to 9, the minimum level is always 1.

In total, the results show that all configurations are robustly solved. Only the solution of the ASMO-ISO configuration is comparatively expensive which is due to the usage of Jacobi for the relatively high aspect ratios of $\sigma \approx 18.2$. In most (but not in all) cases the usage of an outer BiCGstab scheme is beneficial, the differences, however, are not dramatic. Hence, in view of the additional storage requirements, the employment of this additional BiCGstab solver is questionable. The fact that the huge element aspect ratios in case of the ANISO2 configurations affect the total arithmetic efficiency only mildly or even not at all (compared to the ANISO1 configurations), shows the enormous robustness and efficiency of the ADITriGS smoother.

2.6.3.6 Summary and Open Problems

In summary, the Krylov-smoothed 2-layer-ScaRC methods represent powerful solution mechanisms that are able to robustly solve even complicated configurations as those in Figures 2.38 and 2.39. The most important advantage of the solvers is that the user is relieved from adjusting the critical damping parameters.



(a) CROSS-ISO. **(b)** CROSS-ANISO1. **(c)** CROSS-ANISO2.

(d) ASMO-ISO. **(e)** ASMO-ANISO1. **(f)** ASMO-ANISO2.

Figure 2.40: Total arithmetic efficiency of the solver `MG-FGMRES4__MG-BICG-JAC-D__D` (see Listing 2.15) and its enhanced variant `BICG-MG-FGMRES4__MG-BICG-JAC-D__D` on the two isotropic configurations (Figures 2.40a and 2.40d) and of the solver `MG-FGMRES4__MG-ADI-D__D` (see Listing 2.23) and its enhanced variant `BICG-MG-FGMRES4__MG-ADI-D__D` on the anisotropic configurations (Figures 2.40b, 2.40c, 2.40e and 2.40f).

However, the presented solution schemes still exhibit sources of instability. For example, it is impossible to solve the ASMO-ISO configuration with `MG-FGMRES2(2)__MG(5e-1)-BICG-JAC-D__D`, i. e., when performing only 2 instead of 4 global FGMRes smoothing steps. The solver simply stagnates in this case. Also the results on the four test grids (see Figure 2.35) are much more irregular when using only 2 global FGMRes smoothing steps, in rare cases the solver stagnates as in the case of the ASMO configuration. Using an outer global BiCGstab scheme helps to let the iteration converge, though, but still leads to very irregular behaviour. Another possibility to 'rescue' the solver is to use a relative tolerance of $\varepsilon = 0.2$ instead of $\varepsilon = 0.5$ for the local problems (`MG-FGMRES2(2)__MG(2e-1)-BICG-JAC-D__D`).[18]

Another problem we encountered is that embedding the local multigrid schemes into an outer (local) BiCGstab solver, sometimes negatively influences the global multigrid-Krylov iteration. When solving the local problems more accurately (e. g., gaining 2 digits), these problems vanish since it does not play a crucial role then *how* the two digits are gained. When solving the local problems less accurately (as we do in the numerical tests above), the way *how* the systems are solved (e. g., if and which Krylov solvers are used) seems to play a bigger role.

[18] When using ADITriGS it seems to be no problem to only perform 2 instead of 4 global FGMRes smoothing steps. This is possibly due to the fact that the local solves gain more than the desired tolerance of $\varepsilon = 0.5$ already in the first iteration.

The solvers

```
MG-FGMRES4(4)__MG(5e-1)-BICG-JAC-D__D
BICG-MG-FGMRES4(4)__MG(5e-1)-BICG-JAC-D__D
```

perform quite robustly in our numerical tests, though, and also do so for many other configurations we tried. But there is no guarantee that the solvers converge for all configurations. It may be necessary to further increase the number of global Krylov smoothing steps in certain situations. Another issue is whether we have to expect negative consequences when nesting up to four (global and local) Krylov schemes as in the solver in Listing 2.16 on page 75. Do the different Krylov schemes somehow influence each other in a negative way, or can they be safely nested in an arbitrary manner?

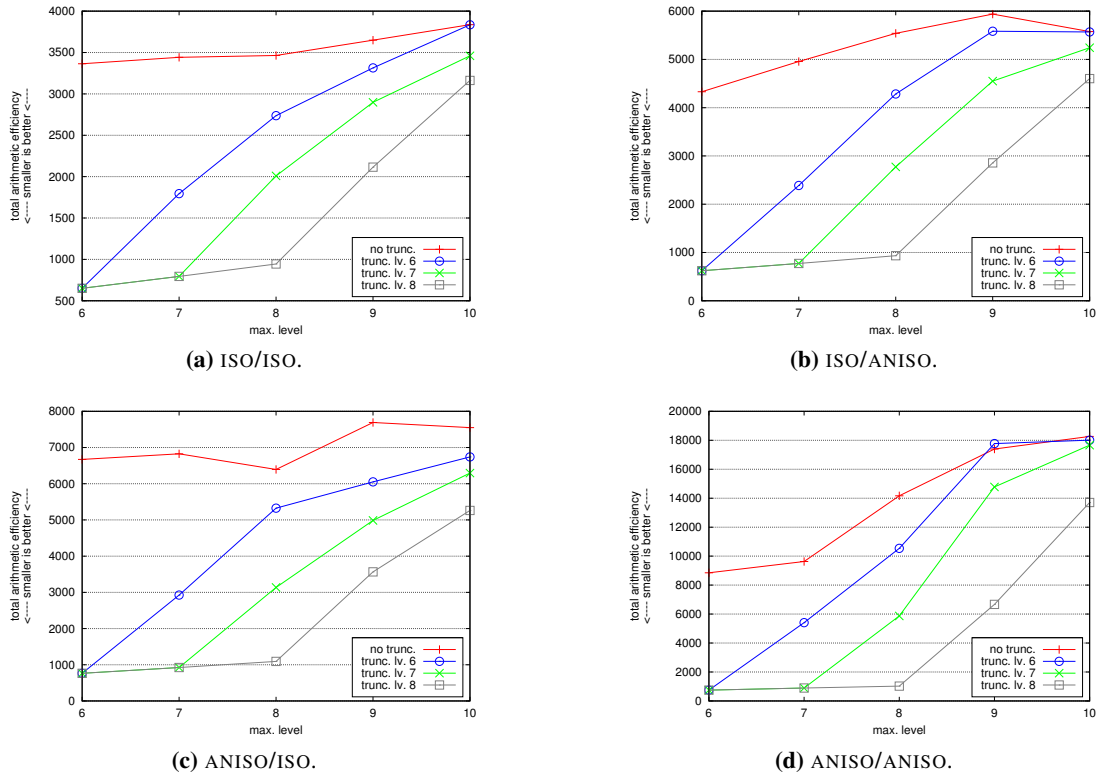
We see that there are still some uncertainties and unanswered questions. However, we want to stress that the ScaRC-Krylov solvers introduced in this section – although not fully understood and not being perfect – already behave significantly more robust than the standard 2-layer-ScaRC schemes without Krylov smoothing.

2.6.4 1-layer-ScaRC vs. 2-layer-ScaRC

In Section 2.5.3 we presented two general possibilities for local preconditioning: On the one hand, applying single steps of an elementary iterative method like Jacobi, leading to the interpretation *block-smoothed multigrid* or *1-layer-ScaRC*, on the other hand, applying 'full' solution methods to solve the local problems, leading to the interpretation *multilevel Schwarz* or *2-layer-ScaRC*. In Section 2.5.4 we discussed and compared the two strategies with respect to different aspects. We now want to verify some of these aspects with the help of some numerical examples. We are especially interested in the question whether and when the application of the (expensive!) 2-layer-ScaRC schemes is beneficial. We examine this issue separately for Jacobi and ADITriGS as elementary smoother.

2.6.4.1 Jacobi as Elementary Smoother

In a first step, we try to reduce the arithmetic costs of the 2-layer-ScaRC schemes by applying the truncation strategy introduced in Section 2.6.2. Therefore, we use our favourite 2-layer-ScaRC solver `MG-FGMRES4__MG-BICG-JAC-D__D` (see Listing 2.15 on page 74) and truncate the local multigrid schemes. We compare truncation level 1 (i. e., no truncation), level 6 (4225 DOF), level 7 (16641 DOF) and level 8 (66089 DOF). Figure 2.41 shows the total arithmetic efficiency of the corresponding solvers. One can see that in those cases where the truncation level is greater than or equal to the maximum level, the total arithmetic efficiency is by far better. It especially does not depend on the degree of anisotropy, i. e., it is nearly the same for all four configurations. This is of course due to the fact, that *all* local problems are solved by UMFPACK (cf. Figure 2.30 on page 70), which does not suffer from the increasing anisotropies. But also in those cases where the truncation level is smaller than the maximum level, the total



(a) ISO/ISO. **(b)** ISO/ANISO.

(c) ANISO/ISO. **(d)** ANISO/ANISO.

Figure 2.41: Total arithmetic efficiency of the solver `MG-FGMRES4__MG-BICG-JAC-D__D` (see Listing 2.15) for varying truncation levels.

arithmetic efficiency in most cases improves with increasing truncation level. In view of the increasing storage requirement and initialisation time for UMFPACK's LU decomposition (cf. the discussion in Section 2.5.3.2), we consider the level 7 truncation as good compromise for all the following tests. We abbreviate the corresponding solver with `MG-FGMRES4__MG(T7)-BICG-JAC-D__D`, where '`T7`' stands for 'truncation on level 7'.

```
1 solver=MG, maxiter=2048, tol=REL:1e-6, cycle=V:1:1
2 smoother=GLOBAL, damp=1.0
3   solver=FGMRES, maxiter=4, tol=IGNORE, restart=4
4   prec=JACOBI, damp=1.0
5 coarse=UMFPACK
```

Listing 2.24: 1-layer-ScaRC solver with multigrid scheme smoothed by FGMRes and Jacobi as elementary preconditioner (`MG-FGMRES4__JAC__D`).

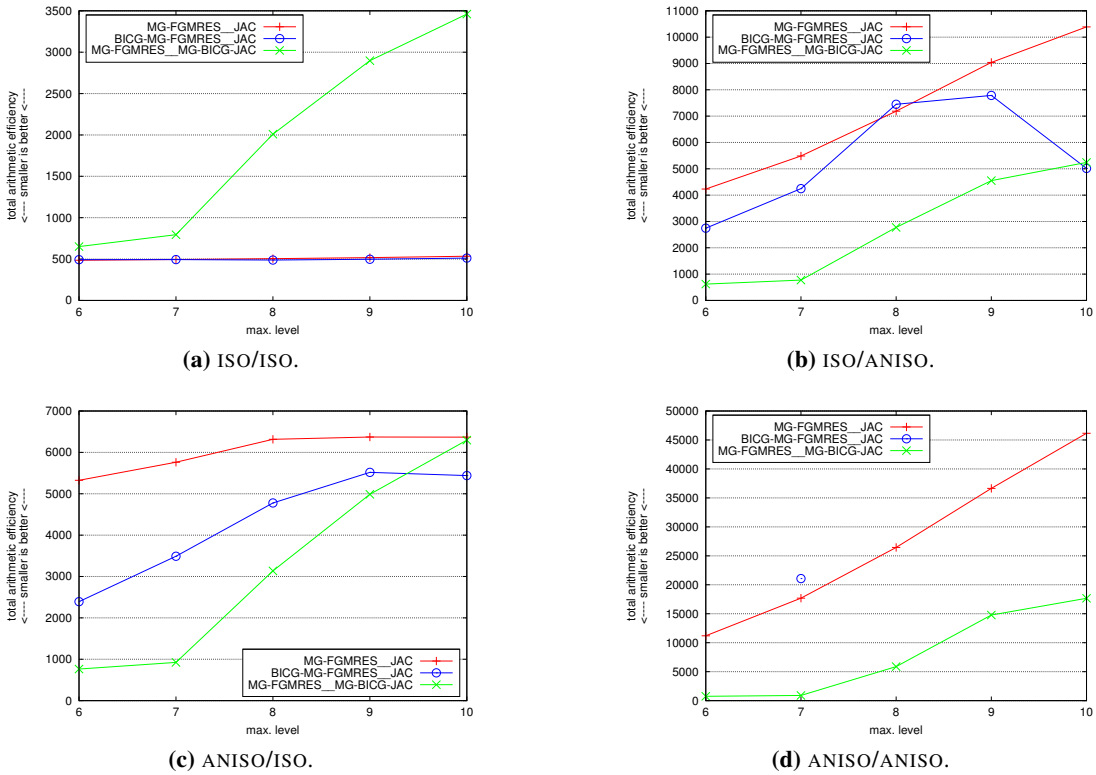
We compare this solver to the 1-layer-ScaRC solver depicted in Listing 2.24, for which we use the same strategy as in Section 2.6.3.2 to get rid of the critical damping parameter: The multigrid scheme is not smoothed by Jacobi directly, but by four iterations of a global FGMRes solver which uses Jacobi as local preconditioner. We again use the four test grids of the previous

section (see Figure 2.31 on page 77). We already found out that for these configurations the 2-layer-ScaRC solver does not necessarily benefit from enclosing it into an outer BiCGstab scheme, so we neglect this solver variant for these tests. For the 1-layer-ScaRC solver, however, we still have to check how an additional outer BiCGstab scheme influences the overall solver performance. So, we additionally consider the solver depicted in Listing 2.25.

```
1 solver=BICG, maxiter=1024, tol=REL:1e-6
2 prec=GLOBAL, damp=1.0
3   solver=MG, maxiter=1, tol=IGNORE, cycle=V:1:1
4   smoother=GLOBAL, damp=1.0
5     solver=FGMRES, maxiter=4, tol=IGNORE, restart=4
6     prec=JACOBI, damp=1.0
7   coarse=UMFPACK
```

Listing 2.25: 1-layer-ScaRC solver of Listing 2.24 as preconditioner for BiCGstab (`BICG-MG-FGMRES4__JAC__D`).

Figure 2.42 shows the total arithmetic efficiency of the three solvers on the four test grids. We want to analyse these plots in combination with Table 2.3 which shows the *number of global*



(a) ISO/ISO. **(b)** ISO/ANISO.

(c) ANISO/ISO. **(d)** ANISO/ANISO.

Figure 2.42: Comparison of the total arithmetic efficiency of the 1-layer-ScaRC solvers `MG-FGMRES4__JAC__D` and `BICG-MG-FGMRES4__JAC__D` and the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-BICG-JAC-D__D` where the local multigrid is truncated on level 7 (see Listing 2.15).

	MG-FGMRES4				BICG-MG-FGMRES4				MG-FGMRES4__MG			
lev	I/I	I/A	A/I	A/A	I/I	I/A	A/I	A/A	I/I	I/A	A/I	A/A
6	48	392	488	1024	48	240	208	-	24	24	32	24
7	48	512	536	1640	48	384	304	1856	24	24	32	24
8	48	672	592	2472	48	656	432	-	24	32	40	32
9	56	848	600	3432	48	688	496	-	24	32	40	40
10	56	976	600	4336	48	464	496	-	24	32	40	40

Table 2.3: Total number of global smoothing steps for the 1-layer-ScaRC solvers `MG-FGMRES4__JAC__D` and `BICG-MG-FGMRES4__JAC__D` and for the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-BICG-JAC-D__D` on the four test configurations ISO/ISO (denoted as I/I), ISO/ANISO (I/A), ANISO/ISO (A/I) and ANISO/ANISO (A/A). Dividing the numbers by 8, 16 and 8, respectively, yields the number of outer iterations of the three solvers.

smoothing steps that are performed by the outer solver on the finest level. This quantity is chosen for two reasons:

1. Simply displaying the number of iterations of the outermost scheme is inappropriate for a fair comparison between different solvers. The arithmetic costs per iteration of the solver `MG(V11)__JAC__D` are, for instance, roughly half as high as those of the solver `BICG-MG(V11)__JAC__D` and roughly a quarter of the costs of the solver `MG(V11)-FGMRES4(4)__JAC__D`. On the other hand, using the number of global smoothing steps per iteration on the finest grid level (which are 2, 4 and 8, respectively, for these three solvers) facilitates a fair solver comparison.

2. Each call of the global smoother is accompanied by (at least) one global defect correction and a call of the additive Schwarz preconditioner (cf. Section 2.5.2 and Listing 2.3 on page 42). These are operations that require communication in a parallel computation. Of course, there are other communicating operations like global norm calculations or – in case of Krylov methods – scalar products, but the number of their calls is usually directly proportional to the number of calls of the global smoother. Hence, the number of global smoothing steps is well suited to estimate the amount of communication the algorithm requires.

We can make several interesting observations:

- While the 1-layer-ScaRC solver `MG-FGMRES4` converges on the ANISO/ANISO configuration, the corresponding `BICG-MG-FGMRES4` variant fails on this configuration. It only converges on level 7, while it stagnates on the other levels. Also its relatively good performance on level 10 of the ISO/ANISO configuration has to be taken with a grain of salt: On level 8 and 9 it behaves much worse and on level 11 (not shown here) the solver stagnates again. This shows that it is not always safe to arbitrarily combine several Krylov methods: The addition of the outer BiCGstab scheme significantly impairs the robustness of the solver instead of improving it (also see Section 2.6.3.6).[19] Due to

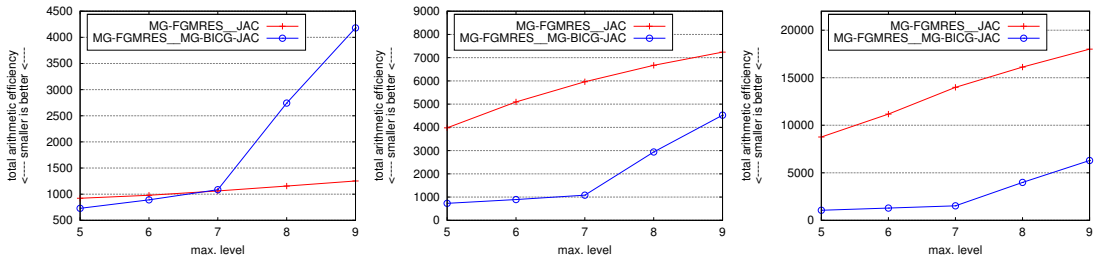
[19] Remember, that the 2-layer-ScaRC variant with an outer BiCGstab solver is not necessarily more efficient than the 2-layer-ScaRC variant without BiCGstab, though, but it behaves at least as robustly.

its unreliable behaviour we will neglect the solver BICG-MG-FGMRES4__JAC__D for the further discussion.

- The performance of the 2-layer-ScaRC solver abruptly deteriorates when switching from level 7 to level 8, which is due to the truncation of the local multigrid schemes on level 7 (cf. Figure 2.41). From level 8 on, the total arithmetic efficiency of the 2-layer-ScaRC solver is clearly level dependent. Table 2.3, however, shows that this is mainly due to an increase of local work: The number of global iterations rises only mildly, or even stays constant in the case of the ISO/ISO configuration.
- On the ISO/ISO configuration, the 1-layer-ScaRC solver behaves independently of the grid level and exhibits a much better total arithmetic efficiency than the (level dependent) 2-layer-ScaRC solver. The number of global smoothing steps (and consequently, the number of global iterations) is roughly twice as high as for the 2-layer-ScaRC solver.
- The results change drastically in the presence of anisotropies. There, also the 1-layer-ScaRC solver exhibits level dependent behaviour, and the 2-layer-ScaRC solver shows a clearly better total arithmetic efficiency. The discrepancy is biggest on the ANISO/ANISO configuration.
- Beside its superiority in terms of total arithmetic efficiency, another essential advantage of the 2-layer-ScaRC solver is revealed in Table 2.3. The number of global smoothing steps of the 1-layer-ScaRC directly depends on the maximum element aspect ratio of the underlying grid: On level 10 it performs 56, 976, 600 and 4336 global smoothing steps for aspect ratios of 2.2, 17.0, 12.7 and 63.6, respectively. The 2-layer-ScaRC solver, on the other hand, is able to hide these micro anisotropies from the outer solver: Its number of global smoothing steps is mainly influenced by the macro anisotropies of the grid and ranges between 24 and 40. This becomes apparent, when we exemplarily compare the ANISO/ISO and the ANISO/ANISO configuration: Although the maximum element aspect ratio increases by a factor of 5 (from 12.7 to 63.6), the number of global iterations stays constant (namely, 5 for both configurations). The number of global iterations of the 1-layer-ScaRC solver increases by a factor of 7 on these two configurations (from 75 to 542). Hence, the communication effort of the 1-layer-ScaRC solver is more than 100 times bigger than that of the 2-layer-ScaRC solver on the ANISO/ANISO configuration.

We now want to check whether these results are valid for the more realistic configurations CROSSOVER and ASMO (see Figures 2.38 and 2.39), as well. In Figure 2.43 the total efficiencies of the 1-layer-ScaRC solver MG-FGMRES4__JAC__D and the 2-layer-ScaRC solver MG-FGMRES4__MG(T7)-BICG-JAC-D__D are compared.[20] Table 2.4 lists the number of global smoothing calls. We see that all the results obtained on the four test grids are basically confirmed: The performance jump of the 2-layer-ScaRC solver from level 7 to level 8 and its level dependency (in terms of total arithmetic efficiency) from level 8 on (due to increase of

[20] The corresponding variants with BiCGstab as outer solver are omitted: The 1-layer-ScaRC variant shows similar deficiencies as on the four test grids in the previous tests, and the 2-layer-ScaRC variant, though behaving robustly, does not yield a better efficiency than the simple multigrid variant.



(a) CROSS-ISO. **(b)** CROSS-ANISO1. **(c)** ASMO-ISO.

Figure 2.43: Comparison of the total arithmetic efficiency of the 1-layer-ScaRC solver `MG-FGMRES4__JAC__D` and the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-BICG-JAC-D__D`.

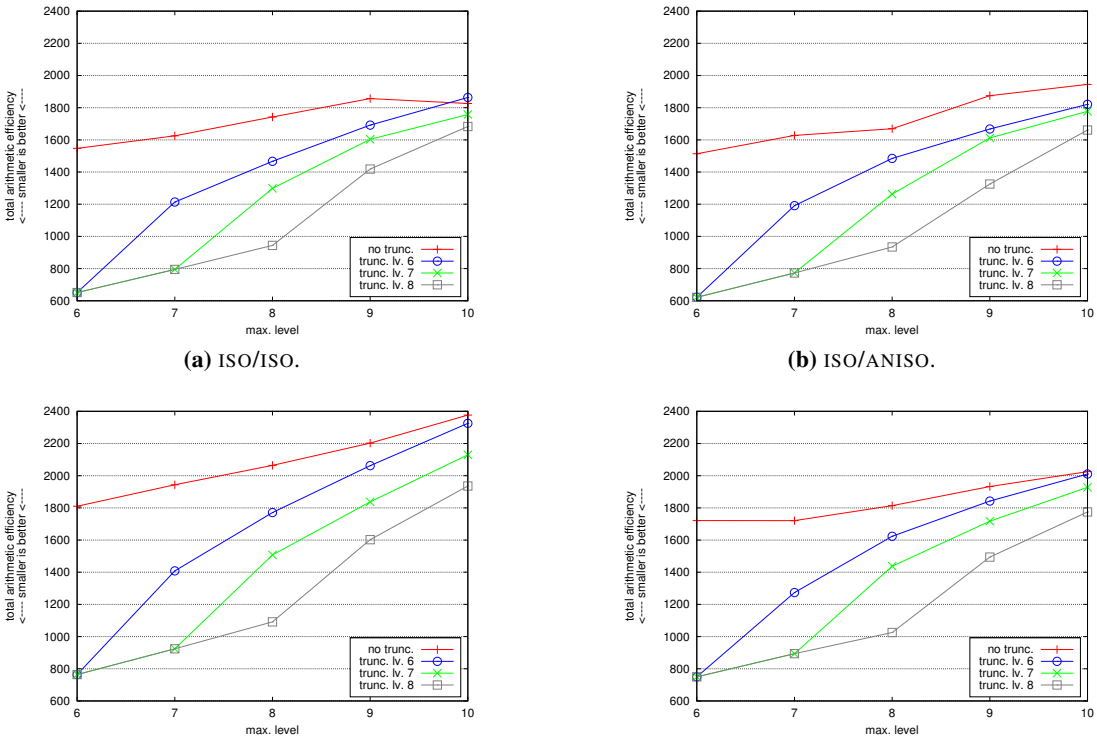
	MG-FGMRES4			MG-FGMRES4__MG		
lev	CR-I	CR-A1	AS-I	CR-I	CR-A1	AS-I
5	88	352	776	32	32	48
6	96	464	1016	32	32	48
7	104	552	1296	32	32	48
8	112	624	1504	32	32	48
9	120	680	1688	32	32	56

Table 2.4: Total number of global smoothing steps for the 1-layer-ScaRC solver `MG-FGMRES4__JAC__D` and the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-BICG-JAC-D__D` on the configurations CROSS-ISO (denoted as CR-I), CROSS-ANISO1 (CR-A1) and ASMO-ISO (AS-I). Dividing the numbers by 8 yields the number of outer iterations of the two solvers.

local work); the superiority of the 1-layer-ScaRC solver in case of the CROSS-ISO configuration (maximum aspect ratio of 2.91) and the superiority of the 2-layer-ScaRC solver in case of the other two configurations CROSS-ANISO1 (maximum aspect ratio of 20.4) and ASMO-ISO (maximum aspect ratio of 18.2); 1-layer-ScaRC's dependence of the number of global smoothing steps on the micro anisotropies (increase from 120 for the CROSS-ISO configuration to 680 for the CROSS-ANISO1 configuration) and 2-layer-ScaRC's ability to hide these local anisotropies from the global solver (same number of global smoothing steps on the CROSS-ISO and the CROSS-ANISO1 configuration).

2.6.4.2 ADITriGS as Elementary Smoother

We now repeat above tests with ADITriGS as elementary smoother. We begin with examining the truncation of the local multigrid solvers. Figure 2.44 shows the total arithmetic efficiency of the solver `MG-FGMRES4__MG-ADI-D__D` (see Listing 2.23 on page 83) for truncation levels 1 (no truncation), 6, 7 and 8. The results are very similar to those of the Jacobi smoother (see Figure 2.41 and its evaluation). The only difference is that the solvers using ADITriGS exhibit a much weaker level dependency. Correspondingly, the results for the different truncation levels



(a) ISO/ISO. **(b)** ISO/ANISO.

(c) ANISO/ISO. **(d)** ANISO/ANISO.

Figure 2.44: Total arithmetic efficiency of the solver `MG-FGMRES4__MG-ADI-D__D` (see Listing 2.23 on page 83) for varying truncation levels.

do not vary as strongly as in the Jacobi case. Again, we use the level 7 truncation, i. e., the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D`, for all the following tests.

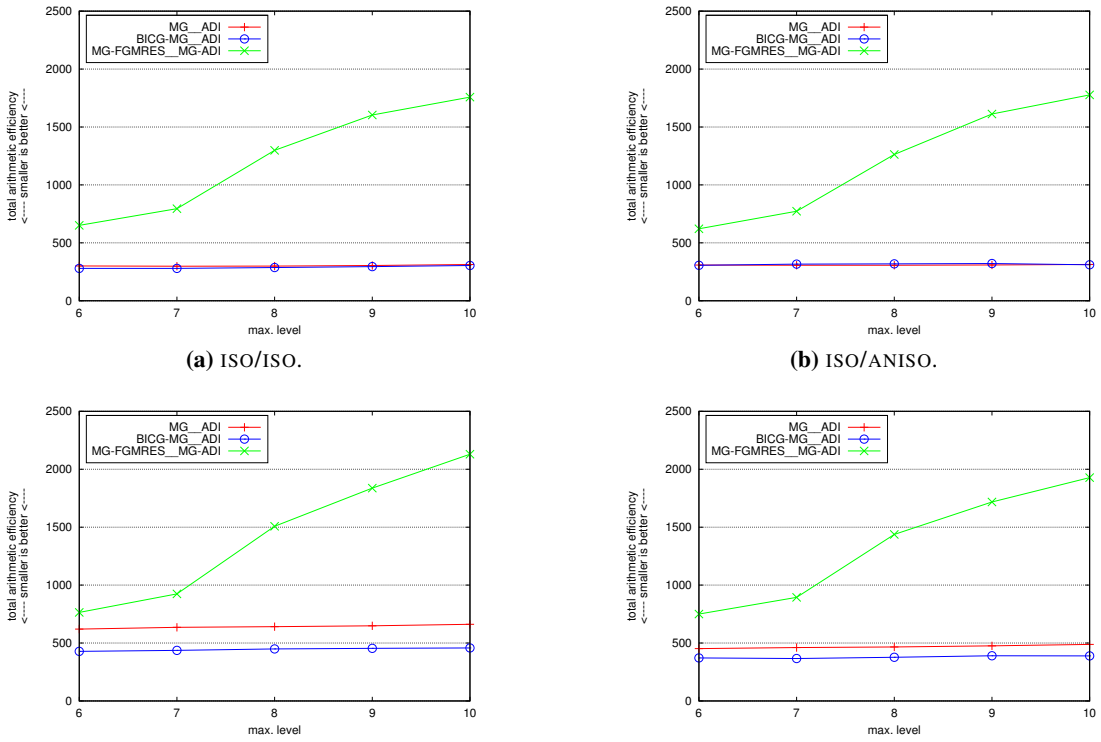
We compare this solver to the two 1-layer-ScaRC solvers depicted in Listings 2.26 and 2.27. In contrast to the 1-layer-ScaRC solver with Jacobi smoothing (see Listing 2.24), ADITriGS

```
1 solver=MG, maxiter=2048, tol=REL:1e-6, cycle=V:2:2
2 smoother=ADITRIGS, damp=1.0
3 coarse=UMFPACK
```

Listing 2.26: 1-layer-ScaRC solver with ADITriGS as elementary smoother (`MG(V22)__ADI__D`).

```
1 solver=BICG, maxiter=1024, tol=REL:1e-6
2 prec=GLOBAL, damp=1.0
3   solver=MG, maxiter=1, tol=IGNORE, cycle=V:2:2
4   smoother=ADITRIGS, damp=1.0
5   coarse=UMFPACK
```

Listing 2.27: 1-layer-ScaRC solver of Listing 2.26 as preconditioner for BiCGstab (`BICG-MG(V22)__ADI__D`).



(a) ISO/ISO. **(b)** ISO/ANISO.

(c) ANISO/ISO. **(d)** ANISO/ANISO.

Figure 2.45: Comparison of the total arithmetic efficiency of the 1-layer-ScaRC solvers `MG(V22)__ADI__D` and `BICG-MG(V22)__ADI__D` and the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D` (see Listing 2.23) on the four prototypical configurations ISO/ISO, ISO/ANISO, ANISO/ISO and ANISO/ANISO. The local multigrid of the 2-layer-ScaRC solver is truncated on level 7.

usually performs best with the damping parameter $\omega = 1.0$, such that a global Krylov smoother, which would 'automatically adjust' this parameter, is not necessary.[21] Figure 2.45 shows the total arithmetic efficiency of the three solvers on the four prototypical test configurations. Table 2.5 displays the number of global smoothing steps. We can make the following observations:

- The total arithmetic efficiency of the 2-layer-ScaRC solver deteriorates with increasing multigrid level. This level dependent deterioration is not as drastic as in the Jacobi case and it is fairly independent of the anisotropies in the grid.
- The 1-layer-ScaRC solvers show a level independent behaviour.
- The 1-layer-ScaRC solvers suffer a little bit stronger from the macro anisotropy than the 2-layer-ScaRC solver (compare ISO/ISO with ANISO/ISO and ISO/ANISO with ISO/ANISO). This is especially true for the solver `MG(V22)__ADI__D`, whose number of global

[21] We tested the solver `MG-FGMRES2(2)__ADI__D` and indeed found that it does not perform better than the simple solver `MG(V22)__ADI__D`.

	MG(V22)				BICG-MG(V22)				MG-FGMRES4__MG			
lev	I/I	I/A	A/I	A/A	I/I	I/A	A/I	A/A	I/I	I/A	A/I	A/A
6	32	32	64	44	24	32	40	32	24	24	32	24
7	32	32	64	48	24	32	40	40	24	24	32	24
8	32	32	64	48	32	32	40	40	24	24	32	24
9	32	32	68	48	32	32	48	40	24	24	32	32
10	32	32	68	52	32	32	48	40	24	24	32	32

Table 2.5: Total number of global smoothing steps for the 1-layer-ScaRC solvers `MG(V22)__ADI__D` and `BICG-MG(V22)__ADI__D` and for the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D` on the prototypical configurations ISO/ISO (denoted as I/I), ISO/ANISO (I/A), ANISO/ISO (A/I) and ANISO/ANISO (A/A). Dividing the numbers by 4, 8 and 8, respectively, yields the number of outer iterations of the three solvers.

smoothing steps shows the strongest increase from the ISO/... to the ANISO/... configurations.

- On the two configurations without macro anisotropies (ISO/...), the two 1-layer-ScaRC solvers show the same efficiency. On the other two configurations (ANISO/...), however, the BiCGstab is able to weaken the negative influence of the macro anisotropies.

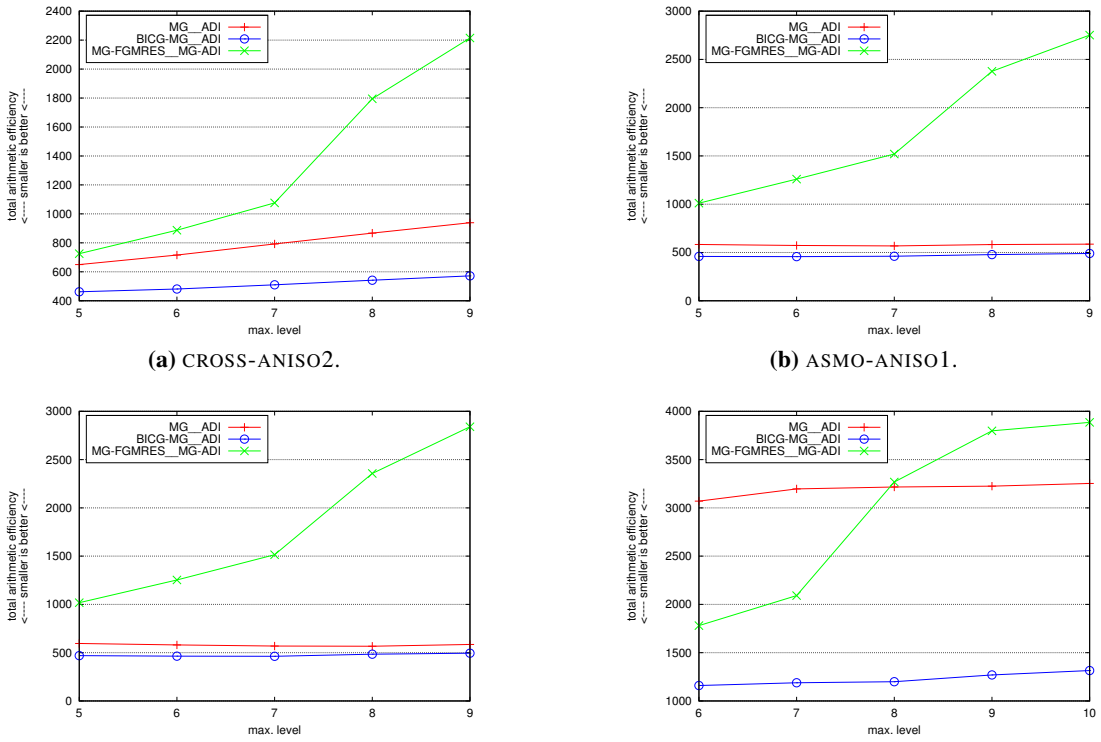
- On all four configurations, both 1-layer-ScaRC solvers are clearly more efficient than the 2-layer-ScaRC solver.

- The 2-layer-ScaRC solver exhibits the smallest number of global smoothing steps, but the differences are much less dramatic compared to the Jacobi case.

Figure 2.46 and Table 2.6 show that these results are essentially true for the CROSS-ANISO2, ASMO-ANISO1 and ASMO-ANISO2 configurations, too. A difference is, that on the CROSS-ANISO2 configuration the number of global smoothing steps of the two 1-layer-ScaRC solvers rise with increasing multigrid level. This does not happen for the 2-layer-ScaRC solver. Nevertheless, the two 1-layer-ScaRC solvers still exhibit a much better total arithmetic efficiency.

	MG(V22)				BICG-MG(V22)				MG-FGMRES4__MG			
lev	CR2	AS1	AS2	AC1	CR2	AS1	AS2	AC1	CR2	AS1	AS2	AC1
5	64	56	56	300	40	40	40	100	32	40	40	56
6	72	56	56	316	44	40	40	108	32	40	40	56
7	80	56	56	320	44	40	40	116	32	40	40	56
8	88	60	56	320	48	44	44	116	32	40	48	56
9	96	60	60	324	52	44	44	116	32	48	48	56

Table 2.6: Total number of global smoothing steps for the 1-layer-ScaRC solvers `MG(V22)__ADI__D` and `BICG-MG(V22)__ADI__D` and for the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D` on the configurations CROSS-ANISO2 (denoted as CR2), ASMO-ANISO1 (AS1), ASMO-ANISO2 (AS2) and ACCORDION-ANISO1 (AC1). Dividing the numbers by 4, 8 and 8, respectively, yields the number of outer iterations of the three solvers.

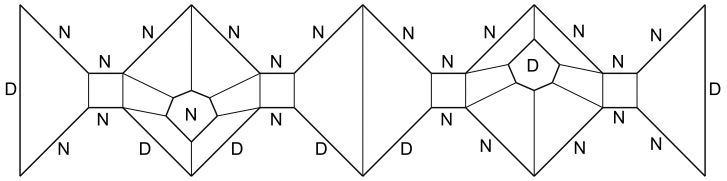


(a) CROSS-ANISO2. **(b)** ASMO-ANISO1.

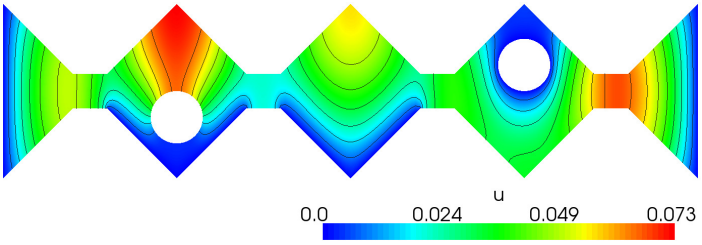
(c) ASMO-ANISO2. **(d)** ACCORDION-ANISO1.

Figure 2.46: Comparison of the total arithmetic efficiency of the 1-layer-ScaRC solvers `MG(V22)__ADI__D` and `BICG-MG(V22)__ADI__D` and the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D` on the configurations CROSS-ANISO2, ASMO-ANISO1, ASMO-ANISO2 and ACCORDION-ANISO1.

So, we see that on the configurations where the 2-layer-ScaRC Jacobi solver outperforms the 1-layer-ScaRC Jacobi solver, the 1-layer-ScaRC ADITriGS solvers are still clearly more efficient than the 2-layer-ScaRC ADITriGS solver. In search of configurations, where this situation changes, we found that the simple 1-layer-ScaRC solver `MG(V22)__ADI__D` does have some problems on the domain depicted in Figure 2.47, which we want to denote – due to its shape and for lack of a better name – with ACCORDION. It consists of 20 subdomains and contains 21.0 M vertices on level 10. We anisotropically refine towards the holes, towards the vertical boundaries above and below the holes, and towards the vertical boundary in the centre of the domain, using the ANISO1 refinement (see page 85). Figure 2.46d and the fifth column in Table 2.6 show that the solver `MG(V22)__ADI__D` is indeed quite inefficient compared to the other three configurations. As a consequence, the 2-layer-ScaRC solver at least shows a better total arithmetic efficiency on level 6 and 7 (where all local problems are solved by UMFPACK). From level 8 on, however, the 1-layer-ScaRC solver is more efficient again. On the other hand, the 1-layer-ScaRC solver needs roughly 5 times more global smoothing steps which is a clear advantage for the 2-layer-ScaRC solver. However, using an outer BiCGstab scheme improves the performance significantly here: The solver `BICG-MG(V22)__ADI__D` is roughly



(a) Macro decomposition (level 0) consisting of 20 subdomains.



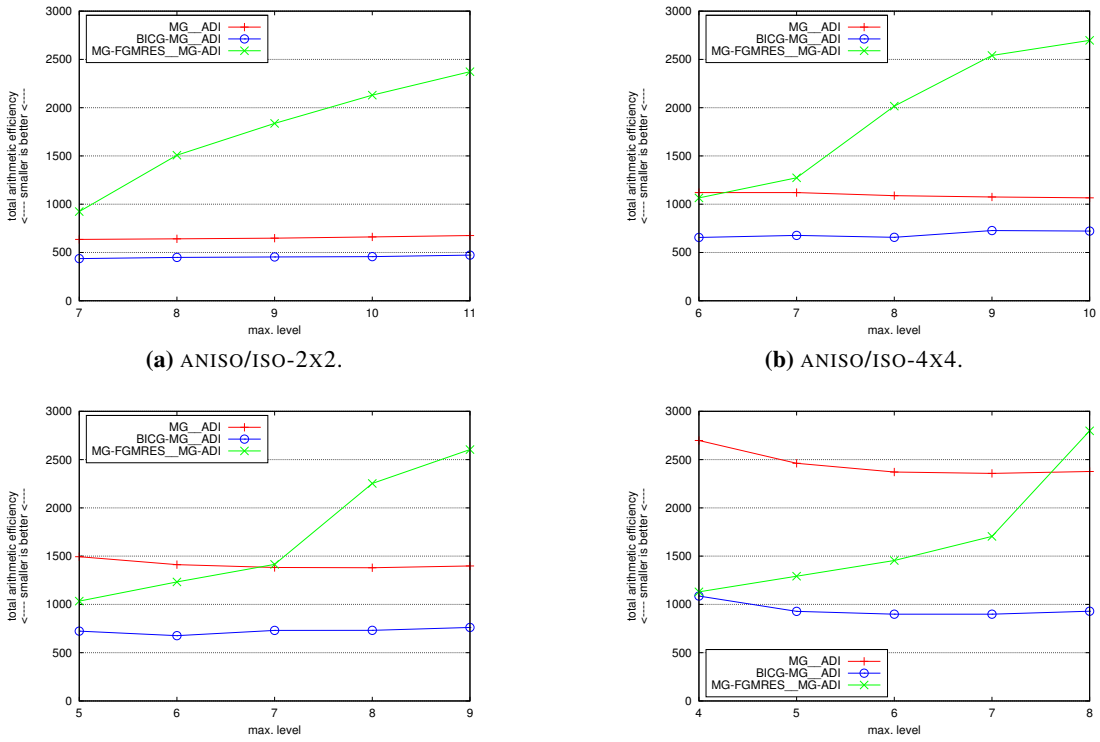
(b) Computed solution.

Figure 2.47: ACCORDION configuration.

2.5 times better (in terms of total arithmetic efficiency and number of smoothing steps) than the solver `MG(V22)__ADI__D`. Consequently, it clearly outperforms the 2-layer-ScaRC solver again in terms of total arithmetic efficiency (1300 vs. 3900), where it needs roughly twice as many smoothing steps (116 vs. 56).

We perform a final test, that confirms these results. In Section 2.5.4 we explained that the ScaRC solvers in principle suffer from strong macro anisotropies which is a consequence of the block-Jacobi character of the additive Schwarz preconditioner (also see Kilian [94]). We want to examine how the two 1-layer-ScaRC solvers and the 2-layer-ScaRC solver manage this difficulty. Therefore, we consider again the ANISO/ISO configuration which consists of 2×2 subdomains (cf. Figure 2.31c on page 77). We increase the number of subdomains by simply subdividing each subdomain regularly into four. We do this three times, such that we obtain the configurations ANISO/ISO-2X2, ANISO/ISO-4X4, ANISO/ISO-8X8 and ANISO/ISO-16X16 with 2×2, 4×4, 8×8 and 16×16 subdomains, respectively. We adapt the multigrid level correspondingly such that we always deal with the same number of unknowns ranging from 66.0 k (level 7 for the 2X2 configuration, level 4 for the 16X16 configuration) to 16.8 M (level 11 for the 2X2 configuration, level 8 for the 16X16 configuration).

Figure 2.48 shows the total arithmetic efficiency of the three solvers on the four grids and Table 2.7 the corresponding number of global smoothing steps. One can see that the simple 1-layer-ScaRC solvers suffers most from the increasing number of subdomains – its total arithmetic efficiency deteriorates by a factor of roughly 3.5 (from 700 to 2400) and the number of global smoothing steps increases by the same factor (from 68 to 236). The total arithmetic efficiency of the 2-layer-ScaRC solver, however, does not vary so strongly: It deteriorates from 2400 on the 2X2 configuration to 2900 on the 16X16 configuration. The number of smoothing steps increases by a factor of 1.5 from 32 to 48. Consequently, the 2-layer-ScaRC solver can



(a) ANISO/ISO-2X2. **(b)** ANISO/ISO-4X4.

(c) ANISO/ISO-8X8. **(d)** ANISO/ISO-16X16.

Figure 2.48: Comparison of the total arithmetic efficiency of the 1-layer-ScaRC solvers `MG(V22)__ADI__D` and `BICG-MG(V22)__ADI__D` and the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D` on the ANISO/ISO configuration with increasing number of subdomains.

	MG(V22)				BICG-MG(V22)				MG-FGMRES4__MG			
DOF	2x2	4x4	8x8	16x16	2x2	4x4	8x8	16x16	2x2	4x4	8x8	16x16
6.60e+4	64	108	140	236	40	56	60	88	32	40	40	56
2.63e+5	64	112	136	228	40	60	60	80	32	40	40	48
1.05e+6	68	108	136	228	44	64	68	84	32	40	40	48
4.20e+6	68	108	136	232	44	64	64	84	32	40	40	48
1.68e+7	68	108	140	236	44	64	68	84	32	40	40	48

Table 2.7: Total number of global smoothing steps for the 1-layer-ScaRC solvers `MG(V22)__ADI__D` and `BICG-MG(V22)__ADI__D` and for the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D` on the configurations ANISO/ISO-2X2 (denoted as 2X2), ANISO/ISO-4X4, ANISO/ISO-8X8 and ANISO/ISO-16X16. Dividing the numbers by 4, 8 and 8, respectively, yields the number of outer iterations of the three solvers.

compete with the simple 1-layer-ScaRC solver at least up to level 7 where all local problems are solved by UMFPACK. Beyond level 7 the 2-layer-ScaRC solver is still more expensive.

However, the solver `BICG-MG(V22)__ADI__D` again clearly outperforms the other two solvers. Its number of global smoothing steps only doubles from the 2X2 to the 16X16 configuration,

and the total arithmetic efficiency only deteriorates by a factor of 2, such that at the end the total arithmetic efficiency of the 1-layer-ScaRC BiCGstab solver is roughly 2.5 times better than that of the 2-layer-ScaRC solver, and its number of global smoothing steps is roughly twice as high. Figure 2.49 displays the results on the finest level of the four grids (1.68e+7 DOF) in one diagram. It once again shows that the simple 1-layer-ScaRC solver suffers strongest from the

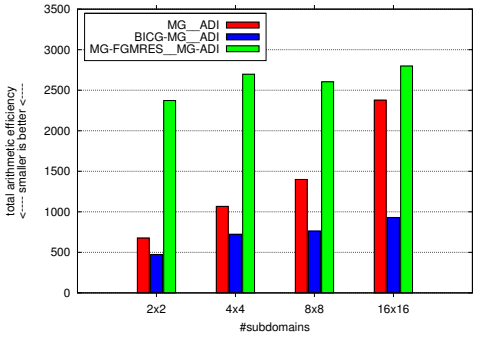


Figure 2.49: Comparison of the total arithmetic efficiency of the 1-layer-ScaRC solvers `MG(V22)__ADI__D` and `BICG-MG(V22)__ADI__D` and the 2-layer-ScaRC solver `MG-FGMRES4__MG(T7)-ADI-D__D` on the finest level (1.68e+7 DOF) of the ANISO/ISO configuration with increasing number of subdomains.

increasing number of subdomains. The 1-layer-ScaRC BiCGstab and the 2-layer-ScaRC solver do not vary as strongly such that the 1-layer-ScaRC BiCGstab solver is the clear favourite due to its much lower costs.[22]

2.6.4.3 Summary and Critical Discussion

On the configurations we tested, the employment of 2-layer-ScaRC solvers is only beneficial in the case that Jacobi is used as elementary smoother. With ADITriGS as smoother the 1-layer-ScaRC schemes are clearly more efficient, especially when using BiCGstab as outer solver. The reason is that the 2-layer-ScaRC methods can only tap their full potential when there are local irregularities to 'hide'. In case of the Jacobi smoother such local irregularities are given by localised mesh anisotropies. The ADITriGS smoother, however, is designed to robustly deal with anisotropies. Consequently, the ADITriGS smoother itself already fulfils the task to hide these local irregularities from the outer solver. In this sense, the embedding into an 2-layer-ScaRC scheme can yield no (or only marginal) further improvement since there is, plainly spoken, 'nothing to hide'.

This situation is unsatisfactory since the employment of local multigrid schemes is one of the core ideas of FEAST. However, we believe that the ADITriGS smoother itself is simply 'too strong' such that there is no chance to show the superiority of the 2-layer-ScaRC strategy with the help of the configurations we considered for our tests. It needs more than anisotropic

[22] Kilian [94] and Becker [16] always compared the 2-layer-ScaRC solvers with the simple `MG__ADI__D` solver, but never with the corresponding BiCGstab variant `BICG-MG__ADI__D`.

elliptic linear operators in 2D and irregular grids with high aspect ratios to sufficiently impair the efficiency of 1-layer-ScaRC ADITriGS solvers, such that 2-layer-ScaRC can eventually win the comparison. We believe that ADITriGS will lose its efficiency to some degree when applied in the context of more complicated physical equations. It is also unclear, how ADITriGS behaves in 3D computations.[23] However, a configuration where 1-layer-ScaRC ADITriGS is clearly outperformed by 2-layer-ScaRC ADITriGS still has to be found.

Now, one could argue that for the linear elliptic case we are currently dealing with, the 2-layer-ScaRC approach is unnecessary; the 1-layer-ScaRC ADITriGS solver embedded into an outer BiCGstab solver treats all problems more efficiently and at least as robustly. However, there are situations where the employment of ADITriGS is unsuited or even not possible:

- ADITriGS needs about 9 times more additional storage than Jacobi. Hence, changing ADITriGS for Jacobi is usually not beneficial in terms of arithmetic work, but it clearly is in terms of storage requirements. This means, even if Jacobi performs worse it might nevertheless be advantageous (or even necessary) to switch to Jacobi on systems where the amount of RAM is critical. In such cases one can readily benefit from the 2-layer-ScaRC approach as our numerical tests show. However, this argument is weakened by the fact that Jacobi needs an additional Krylov-smoother for sufficient robustness as the tests in Section 2.6.3.5 show.

- ADITriGS is only applicable on subdomains that fulfil the tensor product property (see Section 2.1.1), while Jacobi can be used on arbitrarily structured meshes. If such sub-meshes exhibit strong anisotropies, then other smoothers like ILU have to be employed. For such smoothers, the 1-layer-ScaRC and 2-layer-ScaRC approaches still have to be compared.

- The efficiency of ADITriGS decisively depends on the hardware-oriented implementation within the SparseBandedBLAS library (see Section 2.1.1). The implementation has to be adapted and optimised for each finite element; in FEAST, it has been realised for Q_1 until now. It is not clear yet how the ADITriGS approach will perform in 3D or even how it has to be realised for non-standard finite elements like the rotated bilinear element $\tilde{Q}_1$ (see Turek [158]). The Jacobi smoother, however, is always available and its efficient implementation is straightforward since it only needs to scale vectors with the inverse of the main diagonal of the system matrix.

- An efficient implementation of ADITriGS on vector machines like the NEC-SX maintained by the HLRS Stuttgart (`http://www.hlrs.de`) is extremely complicated. The MFLOP/s rates of a multigrid solver using ADITriGS are roughly 5 times worse than those of a multigrid solver using Jacobi (see Becker [16, p. 118]). On meshes which exhibit not too strong anisotropies, the Jacobi smoother consequently has the chance to outperform the

[23] In 2D, ADITriGS is realised by implicitly changing the rowwise numbering scheme into a columnwise one in every second step to catch mesh anisotropies in both spatial directions (see Section 2.5.3.1). It is not clear yet how this concept is efficiently transferred to 3D and if it is able to treat all kinds of anisotropies that can occur in 3D in an equally robust way.

ADITriGS smoother, thereby exploiting the 2-layer-ScaRC concept. On standard architectures, however, the MFLOP/s rates of the two smoothers are similar to each other (also see Becker [16, p. 118]).

- The recent trend to outsource parts of the computational work to fast co-processors like GPUs (see Göddeke et al. [73, 75] and references therein) requires great implementational efforts, despite the availability of programming tools like NVIDIA's cuda (`http://www.nvidia.com/cuda`). Algorithms and data structures have to be adapted and tuned for the specific kind of hardware at hand. Consequently, the implementation of such highly sophisticated and complicated procedures like ADITriGS smoothing is a tedious and time-consuming task. Furthermore, there is no guarantee that algorithms and data structures exist that efficiently realise a specific procedure and, at the same time, fully exploit the strengths of the specific co-processor. Hence, in a first phase a co-processor library most likely offers only simple procedures (like Jacobi smoothing), and scientific or economic constraints can significantly extend the length of this initial phase.

These arguments show that, despite of the advantages of the 1-layer-ScaRC ADITriGS approach, the 2-layer-ScaRC strategy is fully justified.

The strongest argument for 2-layer-ScaRC, however, has to be seen in its ability to minimise the number of global iterations (respectively, global smoothing steps) and, thus, the amount of communication. In view of the development of future hardware systems, especially the increasing gap in the improvement of network and processor technology (cf. Section 2.1.2), this property will become more and more important. Due to the trend to distribute computational problems to ever more parallel processors, communication will be the decisive factor for the overall efficiency of an algorithm. At the same time, the pure amount of (local) arithmetic work will become less critical, especially when extremely fast co-processors can be employed. This prospective also alleviates the definite deficiency of the 2-layer-ScaRC solvers to exhibit level dependent arithmetic costs. In the tests we presented in previous sections, it seems, at first glance, that we pay quite a high price to minimise the number global iterations: In some cases, the reduction of number of iterations by only 50% is accompanied by tripled arithmetic costs. On a massively parallel compute cluster, though, these 50% may already be sufficient to obtain a shorter total runtime. This, however, is only possible in combination with a sufficiently sophisticated load balancing system, which is not yet available and whose development has to be seen as one of the most critical future tasks (cf. Section 2.7.4). The (already) good scalability properties of 2-layer-ScaRC are demonstrated, for example, by Becker [16] and Göddeke et al. [71, 75].

Basing on the findings of this chapter, we want to specify which ScaRC solvers are to be favoured for the solution of elliptic equations. When the choice of the elementary smoother is confined to Jacobi, then the solver `MG-FGMRES4__MG-BICG-JAC-D__D` (see Listing 2.15 on page 74) is the best choice for a wide range of configurations. Despite its complexity, it exhibits a relatively high 'black box'character, which is due to the fact that there are no damping parameters to adjust. For highly challenging configurations the four global FGMRes smoothing steps may turn out to be insufficient (cf. Section 2.6.3.6), but on the (already non-trivial)

configurations we tested they yielded good results. The modification of other parameters (e. g., the type of the global multigrid cycle) turned out to have no significant influence on the overall performance. When ADITriGS is available as elementary smoother, than the favourite solver is `BICG-MG__ADI__D` (see Listing 2.27 on page 94). While ADITriGS is able to robustly treat even extreme micro anisotropies, the outer BiCGstab scheme alleviates the negative effects of complicated geometries or macro anisotropies. In Chapter 4 we examine whether our results can be transferred to the elasticity solvers that use ScaRC for preconditioning.

2.7 Future Work on ScaRC

In this final section we want to describe open problems and possible further studies on ScaRC, that have emerged from the examinations and results within this chapter. All these topics offer various directions for future work, ranging from theoretical to software-specific aspects. In order to make further substantial progress with respect to FEAST's flexibility, robustness and usability, it is required to find adequate solutions to the problems indicated in the following subsections.

2.7.1 Theoretical Aspects

The extended Dirichlet boundary conditions described in Sections 2.4 and 2.6.1.1 render the local multigrid schemes nonconforming. In order to better understand their convergence behaviour, theoretical examinations are necessary. Literature about multigrid methods on non-nested grids, or, more generally, about nonconforming multigrid methods could be helpful in this context. The adaptive coarse grid correction represents a suitable strategy to overcome the problems arising with the extended Dirichlet boundary conditions, but a deeper theoretical understanding could yield a more specific and eventually more efficient solution.

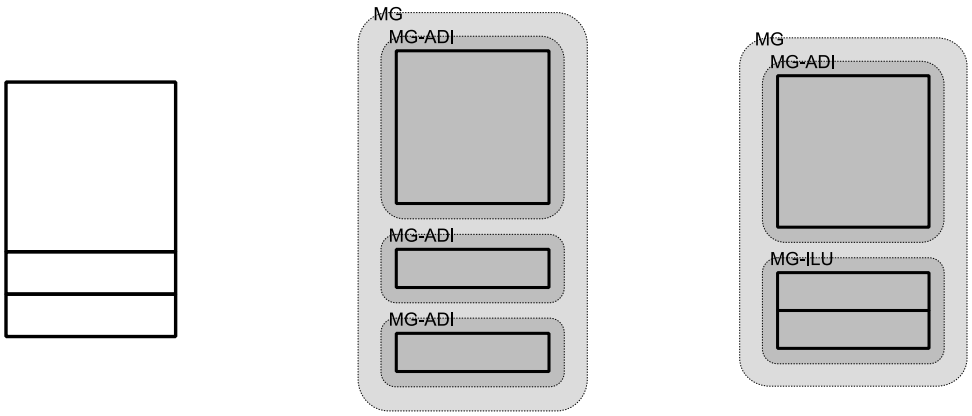
Another issue is the employment of Krylov schemes as outer solvers or as smoothers. In Section 2.6.3 we have seen that the usage of several nested Krylov schemes is not necessarily beneficial, or can even by disadvantageous. Furthermore, applying too few Krylov smoothing steps can lead to instabilities. A deeper understanding is necessary how nested Krylov schemes influence each other and how they interact with multigrid solvers, especially when combined on the different layers.

2.7.2 Recursive Clustering

In this thesis, we only discuss the 'Sca' part of the acronym 'ScaRC', which stands for 'scalable' (see Sections 2.5.4 and 2.7.3). The acronym 'RC' means 'recursive clustering'. The idea is outlined by Kilian [94], while Becker [16] realises the concept within FEAST and performs extensive numerical studies to examine it. The approach can be regarded as an extension of the

2-layer-ScaRC strategy, with the goal to alleviate the negative effects of strong macro anisotropies. We want to briefly describe the concept.

The term 'clustering' refers to the process of combining several subdomains. There are two different ways how to do this. On the one hand, the meshes of k neighbouring subdomains can be *merged* in the sense that instead of k local matrices only *one* local matrix is assembled. The resulting submesh (merged of k tensor product meshes) does in general not have the tensor product property anymore. This means, that specialised smoothers like ADITriGS can not be applied to the corresponding local problem. Instead, general smoothers like ILU can be used. Figure 2.50 illustrates the concept with the help of a simple example grid consisting of three subdomains, two of them being slightly anisotropic. Figure 2.50b represents a standard 2-



(a) Domain consisting of three subdomains.

(b) Standard 2-layer-ScaRC approach.

(c) 2-layer-ScaRC approach with merged subdomains.

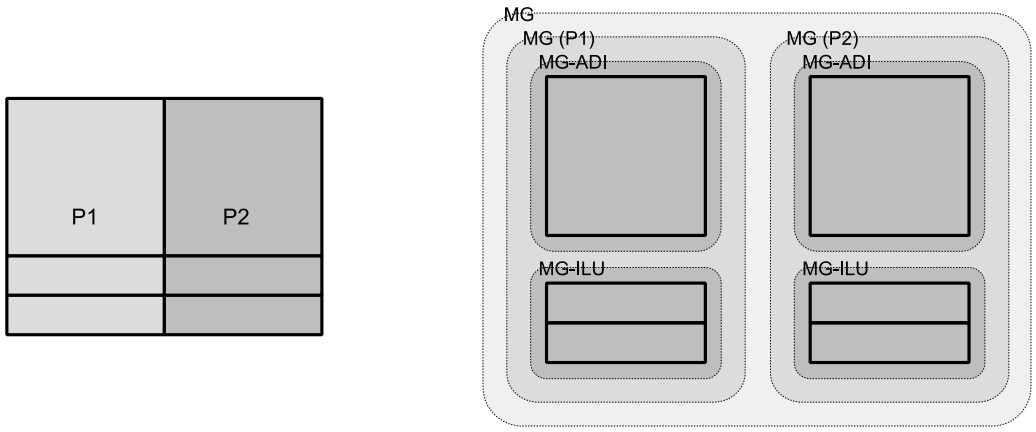
Figure 2.50: Schematic illustration of the first clustering strategy 'merging subdomains'.

layer-ScaRC solver which applies three local multigrid solvers to three subproblems, while Figure 2.50c shows a 2-layer-ScaRC solver that applies one local `MG-ILU` solver to the two merged anisotropic subdomains. The advantage of this approach is that the macro anisotropies are now 'hidden' from the outer solver which only 'sees' the (isotropic) merged subdomain. The convergence rate of the global solver improves accordingly. The disadvantage is the lack of the tensor product property and the corresponding deterioration of the processor efficiency (cf. Section 2.1.1).

The second way to cluster subdomains is to introduce an intermediate layer between the global and the local ones and to define an additional multigrid solver there, leading to the concept of 3-layer-ScaRC. This approach is especially meaningful in parallel computations: In general, the number of subdomains does not equal the number of available processors such that several subdomains have to be scheduled to one processor. In order to minimise inter-processor communication it is beneficial to define a multigrid solver on the cluster of subdomains that reside on the same processor (which defines the intermediate layer then). When this solver gains, e. g., one digit, then the number of iterations of the global solver is minimised. The

intermediate layer does not necessarily have to be defined by the distribution of subdomains to the processors, its definition can also be based on geometric considerations. The local subdomains can (but do not have to) be treated by local multigrid solvers. To decide whether this is advantageous or not, the same guidelines can be applied as for the question whether to use 1-layer-ScaRC or 2-layer-ScaRC (see Section 2.6.4). If on the local layer multigrid is used, as well, there are three nested multigrid schemes in total, spread over three layers, which justifies the name 3-layer-ScaRC. This hierarchical layer structure and the correspondingly nested multigrid schemes motivate the term 'recursive' and explain the 'R' in the acronym 'ScaRC'.

Figure 2.51 illustrates the concept with the help of an extension of the grid depicted in Figure 2.50a. It consists of six subdomains, distributed to two processors (P1 and P2). The global



(a) Domain consisting of six subdomains distributed to two processors (P1 and P2).

(b) 3-layer-ScaRC approach.

Figure 2.51: Schematic illustration of 3-layer-ScaRC, the second clustering strategy.

multigrid solver is smoothed by the two intermediate multigrid schemes (each one acting on one processor), which are smoothed again by local multigrid schemes acting on single (`MG-ADI`) or merged (`MG-ILU`) subdomains. This example also demonstrates that the two kinds of clustering can be combined.

Becker [16] shows that the clustering strategies indeed alleviate the dependence on macro anisotropies and thus increase the robustness of ScaRC solvers. But he also states that the deficiencies of the two approaches – the lack of the tensor product property in case of the 'merging' strategy and the considerable overhead of arithmetic work in case of 3-layer-ScaRC – lead to significantly longer total (parallel) execution times. Hence, he concludes that clustering is not advantageous.

However, in view of the ScaRC improvements we achieved in this chapter (adaptive coarse grid correction, truncation of local multigrid methods, employment of Krylov smoothers), clustering has to be reevaluated. Especially, ACGC and Krylov smoothing have to be employed *additionally* for the intermediate multigrid solver acting on the clustered subdomains. The question to

be answered is whether the expected gain of robustness and performance is sufficient to render the clustering approach superior.

Another issue that is worth examining in this context is the following: When several subdomains reside on one processor, then the corresponding local solves have to be performed *subsequently*. Hence, one could combine the local contributions in a *multiplicative* way (block-Gauß-Seidel), i. e., immediately incorporate the local solutions into the right hand sides of the local systems to be treated next. This does not influence the parallel efficiency of the overall solver, but can considerably improve the numerical efficiency.

2.7.3 ScaRC with Automatic Scaling

One of the advantages of ScaRC is its ability to locally adapt solving parameters. This potential, however, can only be exploited, when there is an automatic system that performs this adaptation. When the user has to make these settings manually, the concept is plainly not practicable in the case of realistic simulation scenarios. Becker [16] realises first steps towards such an automatic system. He shows how the total storage requirements can be significantly reduced by using Jacobi as smoother where possible and ADITriGS only where necessary, instead of using ADITriGS everywhere. The decision process is automated by evaluating the shape of the subdomains and the refinement factors (see equation (2.4) on page 27) and choosing the smoother accordingly.

This automatic system has to be extended. We want to list some possible aspects:

- The automatic choice of the elementary smoother has to be refined. Not only the shape of the subdomain itself has to be considered, but also the shapes of the neighbouring subdomains which influence the local system via the minimal overlap. For example, the bottom left subdomain in the grid depicted in Figure 2.31c on page 77 itself is fairly isotropic, but due to its strongly elongated and larger neighbours, the simple Jacobi smoother performs badly.

- Whether to use 1-layer-ScaRC or 2-layer-ScaRC can be decided basing on the speed of the network connection in a parallel computing environment and on the geometric properties of underlying mesh (also compare Section 2.7.4). The network quality can also influence the choice how many digits to gain in local solves: The slower the network (in terms of bandwidth and latency), the more important the minimisation of the number of global smoothing steps.

- There has to be a mechanism to detect strong macro anisotropies and to automatically apply clustering techniques if necessary (see Section 2.7.2).

- The results of Section 2.6.4 show that the higher the truncation level for the local multigrid solvers is, the more robust the overall solver behaves. Hence, the truncation level

should be chosen automatically as high as possible, thereby respecting the kind of simulation (how often can one LU decomposition be used?) and the available memory which represents a hard limit (cf. the discussion in Section 2.5.3.2).

The requirements for such an automatic system rise drastically when dynamic grid adaptation techniques are employed. Finally, the realisation of a 'perfect expert system' that relieves the user from all decisions is very unlikely. However, the goal should be to automate as many processes as possible.

2.7.4 Load Balancing

One of the most challenging tasks in parallel computing is to equally distribute the work to the available processors in order to achieve a satisfying parallel efficiency ('load balancing'). In case of 1-layer-ScaRC the amount of local arithmetic work is constant over all subdomains, i. e., one 'only' has to count subdomains and distribute them equally to the available processors. In FEAST, this already extremely difficult task is performed by the partitioning tool METIS [90].

The problem is significantly aggravated when 2-layer-ScaRC solvers are used: The crucial advantage of the 2-layer-ScaRC approach is its ability to 'hide' local irregularities and to minimise the number of global iterations. This is only made possible by 'gaining digits' in the local solves instead of performing a fixed number of iterations. Consequently, the local solvers do different amounts of arithmetic work, depending on the difficulty of the local situation. For example, we consider the solver `MG-FGMRES4__MG-BICG-JAC-D__D` (see Listing 2.15 on page 74) applied to the ANISO/ANISO configuration (see Figure 2.31 on page 77). The amount of arithmetic work it spends on the bottom right subdomain is roughly 8 (2.5, 5) times bigger than that on the top right (top left, bottom left) subdomain. So, when the four subdomains are to be distributed to two processors, the best solution would be to schedule the bottom right subdomain to one processor and the other three subdomains to the second one. The obvious solution to use two subdomains per processor would deteriorate the parallel efficiency significantly.

This trivial example is, of course, not representative for realistic configurations consisting of dozens, hundreds or even thousands of subdomains, where a balanced distribution to processors is much more complicated. Knowing the strengths and weaknesses of the solver components at hand, one might be able to do some a priori estimates of the amount of local arithmetic work. For example, one could use information about mesh anisotropies to estimate the performance of Jacobi smoothers. This, however, is complicated by the fact that – due to the minimal overlap – the geometry of neighbouring subdomains also plays an important role, i. e., it is not sufficient to consider the shape of the respective subdomain alone.[24] It gets even more complicated when

[24] A possible remedy for this problem would be to not use the real minimal overlap, but so called 'mirror boundary conditions' for the local problems. An artificial overlap is created by (implicitly) 'mirroring' the outer cell layer to the outside of the subdomain and to apply Dirichlet boundary conditions to this layer of virtual cells. This corresponds to multiplicating matrix and vector entries that belong to boundary points by the number of subdomains that meet in the respective boundary point. The shape of the neighbouring subdomains will not influence the local system anymore, though, but this technique will surely impair the convergence of the

the operator of the underlying equation is anisotropic or even nonlinear. In the latter case, it is extremely hard to predict where in the mesh nonlinearities rise up and to estimate local solver performance correspondingly.

Hence, instead of using a priori estimates, it might be a better strategy to perform some sort of 'trial' computation (e. g., performing one solver iteration on a coarser grid level), to gather solver statistics, to distribute subdomains to processors correspondingly, and to start the actual computation only then. In a dynamic or nonlinear simulation one can exploit statistics of the previous time step or Newton step to improve load balancing. Similar to the issue of automatic scaling described in Section 2.7.3, dynamic grid adaptation techniques will further aggravate the task of equally distributing work.

We consider the problem of load balancing as one of the most critical ones. As long as this is not adequately solved, the 2-layer-ScaRC strategy can not be fully exploited in practice and has no chance to be superior over 1-layer-ScaRC on (massively) parallel compute systems.

global iteration, especially, when neighbouring subdomains strongly variate in shape and size. In this sense, this approach is inconsistent with ScaRC's design goal to minimise the number of global iterations.

3 Elasticity

Elasticity is a *continuum mechanical* discipline that examines the deformation of solid elastic bodies under external loads. A body is considered as *solid* and *deformable* when mechanical forces change its shape, but not its continuous coherence. Furthermore, the body is *elastic* when the loading process is reversible, that means the body returns to its initial state after removal of the loads. Here, only 'purely' elastic materials are considered, extensions like *thermoelasticity* (including thermal effects), *viscoelasticity* (time-dependent elastic behaviour) or *elastoplasticity* (critical loading states result in changes of the material's micro-structure, rendering the deformation process irreversible) go beyond the scope of this study.

Solid material in general shows heterogeneous structures on a microscopic level, but to enable a mathematical description a body is considered in an idealised way as *continuous set of material points*. Throughout this work we further assume, that the material is *homogeneous* and *isotropic*, i. e., all material points have identical properties which are independent of the spatial direction. The latter is, for instance, not true for wood which shows different properties along its fibres than across them. All considerations are based on the *deterministic theory* saying that each material point is uniquely identifiable at each point in time. This is different, for example, in the analysis of gases where a statistical approach might be more appropriate. Finally, we assume that deformation does not destroy the continuous coherence of the body, i. e., the material does not break.

The chapter is organised as follows: In Section 3.1 we present the basics of (finite deformation) elasticity, including kinematics (Section 3.1.1), balance equations (Section 3.1.2), constitutive laws (Section 3.1.3) and dynamics (Section 3.1.4). After collecting and characterising the nonlinear boundary value problems (Section 3.1.5), the important restrictions to linearised (small deformation) elasticity (Section 3.1.6) and to two spatial dimensions (Section 3.1.7) are discussed.

Section 3.2 deals with the spatial discretisation of the continuous problems by means of the finite element method (FEM), beginning with the standard displacement formulation for the linearised elasticity case (Section 3.2.1) and a description of its drawbacks (Section 3.2.2). Then, the mixed displacement/pressure formulation is motivated and described (Section 3.2.3). Both formulations are then presented in terms of the finite deformation setting (Section 3.2.4).

Section 3.3 covers the important topic of stabilising equal-order finite element pairs. We introduce standard methods (Sections 3.3.1 and 3.3.3) and present a new stabilisation technique for unstructured meshes (Section 3.3.2). We extend our considerations to the nonlinear case

(Section 3.3.4) and compare different stabilisation methods by means of several numerical tests (Section 3.3.5).

Section 3.4 deals with the discretisation of time dependent problems. A method commonly used in CSM is briefly introduced in Section 3.4.1. Finally, a severe drawback of stabilised mixed finite element methods occurring in the case of small time steps is described in Section 3.4.2.

3.1 Continuous Formulation

In this section, which is mainly based on the monograph of Ciarlet [52], we derive the basic equations of elasticity.

3.1.1 Kinematics

We consider a body as coherent continuum of material points, occupying the area $\bar{\Omega} = \Omega \cup \partial\Omega$ with Ω being a bounded, open, connected subset of $\mathbb{R}^3$ with Lipschitz-continuous boundary[1] $\partial\Omega$, i. e., Ω is a *domain* (in the mathematical sense). $\bar{\Omega}$, the so called **undeformed** or **reference configuration**, represents the initial state of the body before any deformation has occurred. A **deformation** results from exposing the body to external loads and is defined as smooth, orientation-preserving vector field

$$\boldsymbol{\varphi} : \begin{cases} \bar{\Omega} \to \mathbb{R}^3 \\ \boldsymbol{X} \mapsto \boldsymbol{x} := \boldsymbol{\varphi}(\boldsymbol{X}), \end{cases} \tag{3.1}$$

which is injective on Ω.[2] The deformed state $\boldsymbol{\varphi}(\bar{\Omega})$ of the body is called the **deformed** or **current configuration**.[3] The properties defining a deformation mapping mathematically describe the aforementioned assumptions of uniquely identifiable material points and non-breaking material. With the help of the partial derivatives $\frac{\partial \varphi_i}{\partial X_j}, i, j \in \{1,2,3\}$, we can define the **deformation gradient**

$$\boldsymbol{F} := \mathbf{Grad}(\boldsymbol{\varphi}) = \left(\frac{\partial \varphi_i}{\partial X_j}\right)_{ij} = \begin{pmatrix} \frac{\partial \varphi_1}{\partial X_1} & \frac{\partial \varphi_1}{\partial X_2} & \frac{\partial \varphi_1}{\partial X_3} \\ \frac{\partial \varphi_2}{\partial X_1} & \frac{\partial \varphi_2}{\partial X_2} & \frac{\partial \varphi_2}{\partial X_3} \\ \frac{\partial \varphi_3}{\partial X_1} & \frac{\partial \varphi_3}{\partial X_2} & \frac{\partial \varphi_3}{\partial X_3} \end{pmatrix} \tag{3.2}$$

[1] A *Lipschitz-continuous boundary* can be represented locally as a graph of a Lipschitz-continuous function with Ω lying on one side of this graph. For a precise definition and examples of boundaries being *not* Lipschitz-continuous see, for example, Ciarlet [52].

[2] Since *self-contact* of the body is allowed, the deformation mapping $\boldsymbol{\varphi}$ does not necessarily have to be injective on the boundary $\partial\Omega$.

[3] Following continuum mechanics literature, quantities corresponding to the reference and to the current configuration are denoted with uppercase letters ($\boldsymbol{X}, \mathbf{Div}, \mathbf{Grad}, \ldots$) and lowercase letters ($\boldsymbol{x}, \mathbf{div}, \mathbf{grad}, \ldots$), respectively. In some cases, however, when this distinction is not appropriate, quantities of the current configuration are denoted with the superscript $(\cdot)^{\mathsf{c}}$.

which can also be denoted as $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} = (\frac{\partial x_i}{\partial X_j})_{ij}$.[4][5] The postulation of a deformation being orientation-preserving is equivalent to the property

$$J := \det(\boldsymbol{F}) > 0 \qquad \text{in } \bar{\Omega}.$$

The deformation gradient $\boldsymbol{F}$ allows for the following interpretation: Let the point $\boldsymbol{X}$ be contained in a line segment lying parallel to the j-th coordinate vector $\boldsymbol{e}_j$ in the undeformed configuration. Then, the j-th column vector of $\boldsymbol{F}(\boldsymbol{X})$, i. e., $\frac{\partial \boldsymbol{\varphi}(\boldsymbol{X})}{\partial X_j}$, is the tangent vector of the deformed image of this line segment in the point $\boldsymbol{x}$. In this respect, the deformation gradient completely describes the deformation up to first order.

Closely connected to the deformation $\boldsymbol{\varphi}$ is the vector field of **displacements** $\boldsymbol{u} : \bar{\Omega} \to \mathbb{R}^3$, determined by

$$\boldsymbol{\varphi} = \mathbf{id} + \boldsymbol{u}. \tag{3.3}$$

The relation between the deformation gradient and the **displacement gradient**

$$\boldsymbol{H} := \mathbf{Grad}(\boldsymbol{u}) = \left(\frac{\partial u_i}{\partial X_j} \right)_{ij}$$

is obviously given by

$$\boldsymbol{F} = \boldsymbol{I} + \boldsymbol{H},$$

with $\boldsymbol{I}$ denoting the identity matrix in $\mathbb{R}^{3\times 3}$.

Basically, there are two possibilities to describe the motion of a body, the *Eulerian view* and the *Lagrangian view*. Generally speaking, they differ by the point of view from which the physical process is observed. In the **Eulerian view** movement is described with respect to the coordinates of the *current configuration*. It can be interpreted as the observer residing at a fixed location in space, measuring the physical properties of material points moving by in the course of time. This description is commonly used in computational fluid dynamics (CFD). In the **Lagrangian view**, however, the basis of description are the coordinates of the *reference configuration*. Here, the observer follows the material points through space and measures their location and properties over time. This description is mostly used in solid mechanics, as it will be throughout this work.

In order to employ the Lagrangian approach we must be able to express quantities of the deformed configuration (describing the 'real' physical state of the body) with respect to the reference configuration, and vice versa. The corresponding transformations are called *'push forward'* (from reference to current configuration) and *'pull back'* (from current to reference configuration). Here, we are interested in transforming *volume elements, area elements*, and *length elements*.

[4] The components of a vector $\boldsymbol{v} \in \mathbb{R}^3$ are written as v_i or $(\boldsymbol{v})_i$ and always refer to the canonical Cartesian basis $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$ with $\boldsymbol{e}_i$ being 1 in the i-th component, and 0 otherwise, e. g., $\boldsymbol{X} = \sum_{i=1}^{3} X_i \boldsymbol{e}_i$.

[5] Although the deformation gradient is computed for a point $\boldsymbol{X} \in \bar{\Omega}$, we preferably omit the argument and write $\boldsymbol{F}$ instead of $\boldsymbol{F}(\boldsymbol{X})$. The same holds for similar quantities to be introduced later.

3.1.1.1 Transformation of Volume Elements

With $\boldsymbol{\varphi}$ being a deformation mapping we can apply the rule of change of variables in integrals in the following special form: With $V \subset \bar{\Omega}$ and $V^{\mathsf{c}} := \boldsymbol{\varphi}(V)$ holds

$$\operatorname{vol} V^{\mathsf{c}} := \int_{V^{\mathsf{c}}} \mathrm{d}x = \int_V \left|\det\left(\mathbf{Grad}(\boldsymbol{\varphi})\right)\right| \mathrm{d}X = \int_V J \,\mathrm{d}X. \tag{3.4}$$

This equation shows how to compute volumes in the current configuration and especially reveals the relation between the **volume element** $\mathrm{d}X$ of the reference configuration and the volume element $\mathrm{d}x$ of the current configuration:

$$\mathrm{d}x = J \,\mathrm{d}X. \tag{3.5}$$

3.1.1.2 Transformation of Area Elements

Area elements can be associated similarly via *surface integrals*, whose definition requires Ω to be a *domain*, i. e., a open, bounded, connected subset with Lipschitz-continuous boundary.[6] For simplicity, we assume in the following that the deformation $\boldsymbol{\varphi}$ is smooth enough and injective on $\bar{\Omega}$ such that $\Omega^{\mathsf{c}} := \boldsymbol{\varphi}(\Omega)$ is a domain, as well, and $\bar{\Omega}^{\mathsf{c}} = \boldsymbol{\varphi}(\bar{\Omega})$. See Ciarlet [52] for details in this regard. Surface integrals can be related to volume integrals via the well-known *Green's formula*. Before we do this we need two fundamental definitions.

With $\mathbb{M}^3$ denoting the set of square matrices $\mathbb{R}^{3\times 3}$, a **tensor field of second order** (short: **tensor**) is given by the mapping

$$\boldsymbol{T} : \begin{cases} \bar{\Omega} \to \mathbb{M}^3 \\ \boldsymbol{X} \mapsto \boldsymbol{T}(\boldsymbol{X}). \end{cases}$$

Since all quantities (of both the reference and the current configuration) are defined with respect to the canonical Cartesian coordinate system, we can identify a tensor $\boldsymbol{T}$ with its representing coordinate matrix

$$(T_{ij}) = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix},$$

where $T_{ij} : \bar{\Omega} \to \mathbb{R}$. In continuum mechanics literature tensors are often used in a more general sense, i. e., a specific matrix notation is avoided by not restricting to a fixed basis (see, e. g.,

[6] This explains why Lipschitz-continuity is a suitable postulation for the solid body's boundary: All necessary integrals are defined, although the body's surface does not have to be 'too smooth'. Especially, polyhedrals are feasible, which makes discretisation and numerical treatment of the elasticity problem possible in the first place.

Stein and Barthold [151]). When the tensor field is smooth enough, its **divergence** is defined as[7]

$$\mathbf{Div}(\boldsymbol{T}) := \begin{pmatrix} \frac{\partial T_{11}}{\partial X_1} + \frac{\partial T_{12}}{\partial X_2} + \frac{\partial T_{13}}{\partial X_3} \\ \frac{\partial T_{21}}{\partial X_1} + \frac{\partial T_{22}}{\partial X_2} + \frac{\partial T_{23}}{\partial X_3} \\ \frac{\partial T_{31}}{\partial X_1} + \frac{\partial T_{32}}{\partial X_2} + \frac{\partial T_{33}}{\partial X_3} \end{pmatrix} = \sum_{i=1}^{3} \Big(\sum_{j=1}^{3} \frac{\partial T_{ij}}{\partial X_j} \Big) \boldsymbol{e}_i.$$

Tensors in the current configuration are defined correspondingly. As an example, we compute the divergence of the displacement gradient in the current configuration,

$$\mathbf{div}\big(\mathbf{grad}(\boldsymbol{u})\big) = \mathbf{div} \begin{pmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} & \frac{\partial u_1}{\partial x_3} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} & \frac{\partial u_2}{\partial x_3} \\ \frac{\partial u_3}{\partial x_1} & \frac{\partial u_3}{\partial x_2} & \frac{\partial u_3}{\partial x_3} \end{pmatrix} = \begin{pmatrix} \frac{\partial^2 u_1}{\partial x_1 \partial x_1} + \frac{\partial^2 u_1}{\partial x_2 \partial x_2} + \frac{\partial^2 u_1}{\partial x_3 \partial x_3} \\ \frac{\partial^2 u_2}{\partial x_1 \partial x_1} + \frac{\partial^2 u_2}{\partial x_2 \partial x_2} + \frac{\partial^2 u_2}{\partial x_3 \partial x_3} \\ \frac{\partial^2 u_3}{\partial x_1 \partial x_1} + \frac{\partial^2 u_3}{\partial x_2 \partial x_2} + \frac{\partial^2 u_3}{\partial x_3 \partial x_3} \end{pmatrix} = \boldsymbol{\Delta u},$$

yielding the Laplacian of $\boldsymbol{u}$. In CFD literature, this relation is often written as $\boldsymbol{\nabla} \cdot \boldsymbol{\nabla u}$.

With these definitions, the *divergence theorem for tensor fields* (in the reference configuration), a consequence of the fundamental Green's formula, can be formulated:

$$\int_\Omega \mathbf{Div}(\boldsymbol{T}) \, \mathrm{d}X = \sum_{i=1}^{3} \Big(\sum_{j=1}^{3} \int_\Omega \frac{\partial T_{ij}}{\partial X_j} \, \mathrm{d}X \Big) \boldsymbol{e}_i = \sum_{i=1}^{3} \Big(\sum_{j=1}^{3} \int_{\partial\Omega} T_{ij} n_j \, \mathrm{d}A \Big) \boldsymbol{e}_i = \int_{\partial\Omega} \boldsymbol{T}\boldsymbol{n} \, \mathrm{d}A. \tag{3.6}$$

Here, $\boldsymbol{n}$ and $\mathrm{d}A$ denote the **unit outer normal vector** and the **area element** along $\partial\Omega$, respectively. The aim is now to connect $\mathrm{d}A$ to the area element $\mathrm{d}a$ of the current configuration via the equation

$$\int_{\partial V} \boldsymbol{T}\boldsymbol{n} \, \mathrm{d}A = \int_{\partial V^{\mathsf{c}}} \boldsymbol{T}^{\mathsf{c}} \boldsymbol{n}^{\mathsf{c}} \, \mathrm{d}a, \tag{3.7}$$

where V is a subdomain of Ω, $V^{\mathsf{c}} := \boldsymbol{\varphi}(V)$, and $\boldsymbol{n}^{\mathsf{c}}$ the unit outer normal vector along $\partial V^{\mathsf{c}} = \boldsymbol{\varphi}(\partial V)$. In order to derive relation (3.7) we choose the current configuration (representing the 'real' physical state of the body) as starting point and assume that the tensor $\boldsymbol{T}^{\mathsf{c}}$ is given. The important **Piola transform** then determines how $\boldsymbol{T}^{\mathsf{c}}$ translates into the corresponding tensor $\boldsymbol{T}$ of the reference configuration:

$$\boldsymbol{T} = J \boldsymbol{T}^{\mathsf{c}} \boldsymbol{F}^{-\mathsf{T}}. \tag{3.8}$$

Here, we use the short notation $\boldsymbol{F}^{-\mathsf{T}} := (\boldsymbol{F}^{\mathsf{T}})^{-1}$. With the help of the divergence theorem for tensor fields (3.6) and the relation

$$\mathbf{Div}(\boldsymbol{T}) = J \, \mathbf{div}(\boldsymbol{T}^{\mathsf{c}}) \tag{3.9}$$

(see Ciarlet [52, Theorem 1.7-1]), one can derive relation (3.7) via

$$\begin{aligned} \int_{\partial V} \boldsymbol{T}\boldsymbol{n} \, \mathrm{d}A &\overset{(3.6)}{=} \int_V \mathbf{Div}(\boldsymbol{T}) \, \mathrm{d}X \overset{(3.9)}{=} \int_V J \, \mathbf{div}(\boldsymbol{T}^{\mathsf{c}}) \, \mathrm{d}X \overset{(3.4)}{=} \int_{V^{\mathsf{c}}} \mathbf{div}(\boldsymbol{T}^{\mathsf{c}}) \, \mathrm{d}x \\ &\overset{(3.6)}{=} \int_{\partial V^{\mathsf{c}}} \boldsymbol{T}^{\mathsf{c}} \boldsymbol{n}^{\mathsf{c}} \, \mathrm{d}a. \end{aligned}$$

[7] The last term $\sum_{i=1}^{3} \Big(\sum_{j=1}^{3} \frac{\partial T_{ij}}{\partial X_j} \Big) \boldsymbol{e}_i$ is often written as $\frac{\partial T_{ij}}{\partial X_j} \boldsymbol{e}_i$, assuming the *repeated index convention for summation*. This convention, however, is not used in this work.

Since this equation holds for every subdomain V, we have

$$\boldsymbol{T}\boldsymbol{n}\,\mathrm{d}A = \boldsymbol{T}^{\mathsf{c}}\boldsymbol{n}^{\mathsf{c}}\,\mathrm{d}a. \tag{3.10}$$

For the special case $\boldsymbol{T}^{\mathsf{c}} = \boldsymbol{I}$ this equation reads $J\boldsymbol{F}^{-\mathsf{T}}\boldsymbol{n}\,\mathrm{d}A = \boldsymbol{n}^{\mathsf{c}}\,\mathrm{d}a$, leading to

$$J\|\boldsymbol{F}^{-\mathsf{T}}\boldsymbol{n}\|\,\mathrm{d}A = \mathrm{d}a. \tag{3.11}$$

This can be used to establish – corresponding to relation (3.4) – a rule for *transforming areas*:

$$\operatorname{area} S^{\mathsf{c}} := \int_{S^{\mathsf{c}}} \mathrm{d}a = \int_S J\|\boldsymbol{F}^{-\mathsf{T}}\boldsymbol{n}\|\,\mathrm{d}A.$$

Here, S is a measurable subset of ∂V and $S^{\mathsf{c}} := \boldsymbol{\varphi}(S)$.

3.1.1.3 Transformation of Length Elements

To see how length elements change under the deformation $\boldsymbol{\varphi}$, we consider a fixed point $\boldsymbol{X} \in \bar{\Omega}$ and an arbitrary line element $\boldsymbol{Y}$ with $\boldsymbol{X}+\boldsymbol{Y} \in \bar{\Omega}$. Assuming that $\boldsymbol{\varphi}$ is differentiable in $\boldsymbol{X}$, we can write[8]

$$\boldsymbol{\varphi}(\boldsymbol{X}+\boldsymbol{Y}) - \boldsymbol{\varphi}(\boldsymbol{X}) = \boldsymbol{F}(\boldsymbol{X})\boldsymbol{Y} + o(\|\boldsymbol{Y}\|),$$

such that the squared Euclidean distance between the two points $\boldsymbol{\varphi}(\boldsymbol{X})$ and $\boldsymbol{\varphi}(\boldsymbol{X}+\boldsymbol{Y})$ in the current configuration can be expressed as follows:

$$\begin{aligned}\|\boldsymbol{\varphi}(\boldsymbol{X}+\boldsymbol{Y}) - \boldsymbol{\varphi}(\boldsymbol{X})\|^2 &= \|\boldsymbol{F}(\boldsymbol{X})\boldsymbol{Y}\|^2 + o(\|\boldsymbol{Y}\|^2)\\ &= \big(\boldsymbol{F}(\boldsymbol{X})\boldsymbol{Y}\big)^{\mathsf{T}}\boldsymbol{F}(\boldsymbol{X})\boldsymbol{Y} + o(\|\boldsymbol{Y}\|^2)\\ &= \boldsymbol{Y}^{\mathsf{T}}\boldsymbol{F}(\boldsymbol{X})^{\mathsf{T}}\boldsymbol{F}(\boldsymbol{X})\boldsymbol{Y} + o(\|\boldsymbol{Y}\|^2).\end{aligned}$$

The tensor

$$\boldsymbol{C}(\boldsymbol{X}) := \boldsymbol{F}(\boldsymbol{X})^{\mathsf{T}}\boldsymbol{F}(\boldsymbol{X}) \tag{3.12}$$

is known as the **right Cauchy-Green strain tensor**. It is symmetric and – due to the properties of $\boldsymbol{\varphi}$ – induces a positive definite quadratic form

$$(\boldsymbol{Y},\boldsymbol{Y}) \mapsto \boldsymbol{Y}^{\mathsf{T}}\boldsymbol{C}(\boldsymbol{X})\boldsymbol{Y} = \|\boldsymbol{F}(\boldsymbol{X})\boldsymbol{Y}\|^2.$$

Thus, for the **length element** $\mathrm{d}L$ of the reference configuration,

$$\mathrm{d}L = \sqrt{\boldsymbol{Y}^{\mathsf{T}}\boldsymbol{Y}},$$

we can express the corresponding length element $\mathrm{d}l$ of the current configuration as

$$\mathrm{d}l = \sqrt{\boldsymbol{Y}^{\mathsf{T}}\boldsymbol{C}(\boldsymbol{X})\boldsymbol{Y}}.$$

In this regard, the tensor $\boldsymbol{C}$ can be interpreted as describing the local change of length under the deformation $\boldsymbol{\varphi}$.

[8] The 'o-notation' $o(h), h \in \mathbb{R}$, means that $o(h) = h\varepsilon(h)$ with $\lim_{h\to 0}\varepsilon(h) = 0$.

3.1.1.4 Rigid Body Motions

The right Cauchy-Green strain tensor $\boldsymbol{C}$ is suited to characterise **rigid body motions**. These are defined by

$$\exists \boldsymbol{b} \in \mathbb{R}^3, \boldsymbol{Q} \in \mathbb{O}^3_+ : \quad \boldsymbol{\varphi}(\boldsymbol{X}) = \boldsymbol{Q}\boldsymbol{X} + \boldsymbol{b}, \qquad \boldsymbol{X} \in \bar{\Omega}, \tag{3.13}$$

where $\mathbb{O}^3_+ \subset \mathbb{M}^3$ denotes the set of orthogonal matrices with positive determinant $+1$. This means that the body is *translated and rotated*, but it does not change its shape. For such rigid body motions obviously $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{Q}$ holds, and hence

$$\boldsymbol{C}(\boldsymbol{X}) = \boldsymbol{F}(\boldsymbol{X})^{\mathsf{T}} \boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{I}, \tag{3.14}$$

indicating that no strains occur. Actually, the latter relation suffices to determine a rigid body motion: If for a smooth enough mapping $\boldsymbol{\varphi}$, equation (3.14) holds with $\det(\boldsymbol{\varphi}) > 0$, then $\boldsymbol{\varphi}$ fulfils condition (3.13). For a proof, see Ciarlet [52, Theorem 1.8-1]. In summary, a deformation $\boldsymbol{\varphi}$ describes a rigid body motion, if and only if its right Cauchy-Green strain tensor $\boldsymbol{C}$ is the identity matrix.

Another strain measure, the so called **Green-St. Venant strain tensor**

$$\boldsymbol{E} := \tfrac{1}{2}\big(\boldsymbol{C} - \boldsymbol{I}\big),$$

is suited to estimate how a deformation *deviates* from a rigid body motion. From a mechanical point of view this is the 'natural' strain measure since it is zero in the case of a rigid body motion, which induces no strain and no stress. The Green-St. Venant strain tensor is often expressed in terms of displacements,

$$\boldsymbol{E}(\boldsymbol{u}) = \tfrac{1}{2}\big(\mathbf{Grad}(\boldsymbol{u}) + \mathbf{Grad}(\boldsymbol{u})^{\mathsf{T}} + \mathbf{Grad}(\boldsymbol{u})^{\mathsf{T}}\,\mathbf{Grad}(\boldsymbol{u})\big), \tag{3.15}$$

revealing the connection to the linearised strain tensor (cf. Section 3.1.6).

3.1.2 Equilibrium Equations

A solid body deforms when it is exposed to some external load. We consider two kinds of load, namely *body forces* and *surface forces*. *Contact forces*, which occur when the body touches itself or some other object, are not taken into account. For simplicity, also *inertial forces* are neglected for the time being. They will be introduced in Section 3.1.4. Since the deformed configuration describes the actual physical state of the body, we begin our examinations there.

Body forces, acting on the interior of the deformed body, are given by a vector field

$$\boldsymbol{f}^{\mathsf{c}} : \Omega^{\mathsf{c}} \to \mathbb{R}^3. \tag{3.16}$$

More precisely, $\boldsymbol{f}^{\mathsf{c}}$ represents the **(density of the) applied body force per unit volume in the current configuration**. An example of such a body force is *gravity*,

$$\boldsymbol{f}^{\mathsf{c}}(\boldsymbol{x}) = \big(0, 0, -g\rho^{\mathsf{c}}(\boldsymbol{x})\big)^{\mathsf{T}},$$

where $\rho^c(\boldsymbol{x}) \in \mathbb{R}$ is the body's *mass density* in the point $\boldsymbol{x}$ of the current configuration, and g denotes the gravity constant.

Surface forces act on parts of the deformed body's boundary and can be described by a vector field

$$\boldsymbol{g}^c : \Gamma_N^c \to \mathbb{R}^3, \tag{3.17}$$

the **(density of the) applied surface force per unit area in the current configuration**. The boundary of the body, denoted by $\Gamma^c := \partial\Omega^c$, decomposes into two parts

$$\Gamma^c = \Gamma_D^c + \Gamma_N^c, \qquad \Gamma_D^c \cap \Gamma_N^c = \emptyset.$$

Γ_D^c is the part of the boundary where the body is fixed (corresponding to **Dirichlet boundary conditions** in the language of boundary value problems), while Γ_N^c is the free part where surface forces apply and which is allowed to move (corresponding to **Neumann boundary conditions**).

Loads that do not depend on the deformation $\boldsymbol{\varphi}$ are called **dead loads**. Gravity, for instance, is such a dead load since it does not change its direction in the course of the deformation process. For surface forces, however, it is in general unrealistic to assume dead loads – the direction of a surface force usually changes when the body deforms. Nevertheless, for simplicity we neglect this and will throughout this work apply surface forces as dead loads.

With the definition of applied forces at hand we can now derive the fundamental equilibrium conditions which are expressed in terms of partial differential equations (PDEs). Their derivation is based on the following central axiom of continuum mechanics, going back to Euler and Cauchy:

AXIOM 3.1 (stress principle of Euler and Cauchy)

For a body Ω^c loaded by forces $\boldsymbol{f}^c : \Omega^c \to \mathbb{R}^3$ and $\boldsymbol{g}^c : \Gamma_N^c \to \mathbb{R}^3$, there exists a vector field

$$\boldsymbol{t}^c : \bar{\Omega}^c \times S_1 \to \mathbb{R}^3, \quad (\boldsymbol{x}, \boldsymbol{n}^c) \mapsto \boldsymbol{t}^c(\boldsymbol{x}, \boldsymbol{n}^c) \tag{3.18}$$

where $S_1 := \left\{ \boldsymbol{v} \in \mathbb{R}^3 \;\middle|\; \|\boldsymbol{v}\| = 1 \right\}$, such that:

1. For arbitrary $V^c \subset \bar{\Omega}^c$ and for any $\boldsymbol{x} \in \Gamma_N^c \cap \partial V^c$ where the unit outer normal vector $\boldsymbol{n}^c$ exists,
$$\boldsymbol{t}^c(\boldsymbol{x}, \boldsymbol{n}^c) = \boldsymbol{g}^c(\boldsymbol{x}).$$

2. For arbitrary $V^c \subset \bar{\Omega}^c$,
$$\int_{V^c} \boldsymbol{f}^c(\boldsymbol{x})\,\mathrm{d}x + \int_{\partial V^c} \boldsymbol{t}^c(\boldsymbol{x}, \boldsymbol{n}^c)\,\mathrm{d}a = \boldsymbol{0},$$
which is the **axiom of force balance**.

3. For arbitrary $V^{\mathsf{c}} \subset \Omega^{\mathsf{c}}$,

$$\int_{V^{\mathsf{c}}} \boldsymbol{x} \times \boldsymbol{f}^{\mathsf{c}}(\boldsymbol{x}) \, \mathrm{d}x + \int_{\partial V^{\mathsf{c}}} \boldsymbol{x} \times \boldsymbol{t}^{\mathsf{c}}(\boldsymbol{x}, \boldsymbol{n}^{\mathsf{c}}) \, \mathrm{d}a = \boldsymbol{0},$$

which is the **axiom of moment balance**. Here, '$\times$' denotes the vector product in $\mathbb{R}^3$.

The axiom states that any subdomain of a solid body loaded by volumetric forces and, possibly, surface forces at points $\boldsymbol{x} \in \Gamma_{\mathrm{N}}^{\mathsf{c}} \cap \partial V^{\mathsf{c}}$, where the unit outer normal vector exists, is in static equilibrium due to the existence of additional surface forces $\boldsymbol{t}^{\mathsf{c}}$ arising on the remaining boundary part. Furthermore, these surface forces depend on the subdomain only via the normal vector, and not, for instance, via the curvature of the subdomain's boundary. The vector $\boldsymbol{t}^{\mathsf{c}}(\boldsymbol{x}, \boldsymbol{n}^{\mathsf{c}})$ is called the **Cauchy stress vector**

A famous theorem of Cauchy states that the Cauchy stress vector field depends linearly on the normal vector and expresses the relation between the stresses and the applied surface forces in terms of a PDE:

THEOREM 3.2 (Cauchy's theorem)

Let the applied body force $\boldsymbol{f}^{\mathsf{c}} : \Omega^{\mathsf{c}} \to \mathbb{R}^3$ be continuous, let the Cauchy stress vector field $\boldsymbol{t}^{\mathsf{c}}$ be continuously differentiable with respect to the variable $\boldsymbol{x} \in \bar{\Omega}^{\mathsf{c}}$ for each $\boldsymbol{n}^{\mathsf{c}} \in S_1$ and continuous with respect to the variable $\boldsymbol{n}^{\mathsf{c}} \in S_1$ for each $\boldsymbol{x} \in \bar{\Omega}^{\mathsf{c}}$. Then, it follows from the axioms of force and momentum balance (Axiom 3.1) that there exists a continuously differentiable tensor field

$$\boldsymbol{T}^{\mathsf{c}} : \bar{\Omega}^{\mathsf{c}} \to \mathbb{M}^3$$

that satisfies

1. the following relation to the Cauchy stress vector

$$\boldsymbol{t}^{\mathsf{c}}(\boldsymbol{x}, \boldsymbol{n}^{\mathsf{c}}) = \boldsymbol{T}^{\mathsf{c}}(\boldsymbol{x}) \boldsymbol{n}^{\mathsf{c}}, \qquad \boldsymbol{x} \in \bar{\Omega}^{\mathsf{c}}, \boldsymbol{n}^{\mathsf{c}} \in S_1, \tag{3.19}$$

2. the PDE

$$-\operatorname{\mathbf{div}}\big(\boldsymbol{T}^{\mathsf{c}}(\boldsymbol{x})\big) = \boldsymbol{f}^{\mathsf{c}}(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega^{\mathsf{c}}, \tag{3.20a}$$

3. the symmetry properties

$$\boldsymbol{T}^{\mathsf{c}}(\boldsymbol{x}) = \boldsymbol{T}^{\mathsf{c}}(\boldsymbol{x})^{\mathsf{T}}, \qquad \boldsymbol{x} \in \bar{\Omega}^{\mathsf{c}}, \tag{3.20b}$$

4. and the boundary conditions

$$\boldsymbol{T}^{\mathsf{c}}(\boldsymbol{x}) \boldsymbol{n}^{\mathsf{c}} = \boldsymbol{g}^{\mathsf{c}}(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Gamma_{\mathrm{N}}^{\mathsf{c}}. \tag{3.20c}$$

For a proof, see Ciarlet [52, Theorem 2.3-1].

The tensor $\boldsymbol{T}^{\mathsf{c}}(\boldsymbol{x})$ introduced in this theorem is called the **Cauchy stress tensor** at the point $\boldsymbol{x}$. Equations (3.20) are known as the **equations of equilibrium in the deformed configuration**. The boundary value problem given by equations (3.20a) and (3.20c) can be transformed into an equivalent **weak** or **variational formulation**

$$\int_{\Omega^{\mathsf{c}}} \boldsymbol{T}^{\mathsf{c}} : \mathbf{grad}(\boldsymbol{\theta}^{\mathsf{c}})\,\mathrm{d}x = \int_{\Omega^{\mathsf{c}}} \boldsymbol{f}^{\mathsf{c}} \cdot \boldsymbol{\theta}^{\mathsf{c}}\,\mathrm{d}x + \int_{\Gamma_{\mathrm{N}}^{\mathsf{c}}} \boldsymbol{g}^{\mathsf{c}} \cdot \boldsymbol{\theta}^{\mathsf{c}}\,\mathrm{d}a, \qquad \boldsymbol{\theta}^{\mathsf{c}} = \mathbf{0} \text{ on } \Gamma_{\mathrm{D}}^{\mathsf{c}},$$

with smooth enough functions $\boldsymbol{\theta}^{\mathsf{c}} : \Omega^{\mathsf{c}} \to \mathbb{R}^3$, $\boldsymbol{u} \cdot \boldsymbol{v}$ denoting the Euclidean vector inner product and $\boldsymbol{A} : \boldsymbol{B} = \operatorname{trace}(\boldsymbol{A}^{\mathsf{T}}\boldsymbol{B})$ the matrix inner product. For a proof of equivalence, see Ciarlet [52, Theorem 2.4-1]. This equation is known as the **principle of virtual work in the deformed configuration**.

To actually compute the deformation of a loaded body, neither the equilibrium equations in the deformed configuration, nor their variational formulation are directly applicable since they implicitly use exactly this unknown deformation field as a variable, e. g., in terms of the Euler variable $\boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X})$. Therefore, the equations have to be expressed with respect to the reference configuration, which is possible with the help of the relations introduced in Section 3.1.1. First, we build the Piola transform (cf. equation (3.8)) of the Cauchy stress tensor:

$$\boldsymbol{T} = J\boldsymbol{T}^{\mathsf{c}}\boldsymbol{F}^{-\mathsf{T}}. \tag{3.21}$$

The tensor $\boldsymbol{T} : \bar{\Omega} \to \mathbb{M}^3$ is called the **first Piola-Kirchhoff stress tensor**. Correspondingly, we define the **first Piola-Kirchhoff stress vector** $\boldsymbol{t}(\boldsymbol{X}, \boldsymbol{n})$ via

$$\boldsymbol{t}(\boldsymbol{X}, \boldsymbol{n})\,\mathrm{d}A = \boldsymbol{t}^{\mathsf{c}}(\boldsymbol{x}, \boldsymbol{n}^{\mathsf{c}})\,\mathrm{d}a,$$

which together with

$$\boldsymbol{T}\boldsymbol{n}\,\mathrm{d}A = \boldsymbol{T}^{\mathsf{c}}\boldsymbol{n}^{\mathsf{c}}\,\mathrm{d}a$$

(cf. relation (3.10)) leads to the counterpart of equation (3.19) in the reference configuration:

$$\boldsymbol{t}(\boldsymbol{X}, \boldsymbol{n}) = \boldsymbol{T}\boldsymbol{n}, \qquad \boldsymbol{X} \in \bar{\Omega}, \boldsymbol{n} \in S_1.$$

Then, due to equation (3.9), we have the following simple relation for the divergence:

$$\mathbf{Div}(\boldsymbol{T}) = J\,\mathbf{div}(\boldsymbol{T}^{\mathsf{c}}).$$

Finally, we define the vector field $\boldsymbol{f} : \Omega \to \mathbb{R}^3$ as the **(density of the) applied body force per unit volume in the reference configuration** and the vector field $\boldsymbol{g} : \Gamma_{\mathrm{N}} \to \mathbb{R}^3$ as the **(density of the) applied surface force per unit area in the reference configuration** by relating them to the applied forces of the deformed configuration via

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{X})\,\mathrm{d}X &= \boldsymbol{f}^{\mathsf{c}}(\boldsymbol{x})\,\mathrm{d}x, \qquad \boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X}) \in \Omega^{\mathsf{c}}, \\ \boldsymbol{g}(\boldsymbol{X})\,\mathrm{d}A &= \boldsymbol{g}^{\mathsf{c}}(\boldsymbol{x})\,\mathrm{d}a, \qquad \boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X}) \in \Gamma_{\mathrm{N}}^{\mathsf{c}}. \end{aligned} \tag{3.22}$$

Due to equations (3.5) and (3.11), resp., the following relations between the force vectors of the reference and the current configuration hold:

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{X}) &= J\boldsymbol{f}^{\mathsf{c}}(\boldsymbol{x}), \\ \boldsymbol{g}(\boldsymbol{X}) &= J\|\boldsymbol{F}^{-\mathsf{T}}\boldsymbol{n}^{\mathsf{c}}\|\boldsymbol{g}^{\mathsf{c}}(\boldsymbol{x}). \end{aligned}$$

With these definitions at hand, we can provide the analogon of Cauchy's theorem 3.2 for the reference configuration:

THEOREM 3.3

Let the forces $\boldsymbol{f}$ and $\boldsymbol{g}$ defined by equations (3.22) be given. Then the first Piola-Kirchhoff stress tensor $\boldsymbol{T}$ defined by relation (3.21) fulfils

$$-\mathbf{Div}\big(\boldsymbol{T}(\boldsymbol{X})\big) = \boldsymbol{f}(\boldsymbol{X}), \qquad \boldsymbol{X} \in \Omega, \tag{3.23a}$$

$$\boldsymbol{F}\boldsymbol{T}(\boldsymbol{X})^{\mathsf{T}} = \boldsymbol{T}(\boldsymbol{X})\boldsymbol{F}^{\mathsf{T}}, \qquad \boldsymbol{X} \in \Omega, \tag{3.23b}$$

$$\boldsymbol{T}(\boldsymbol{X})\boldsymbol{n} = \boldsymbol{g}(\boldsymbol{X}), \qquad \boldsymbol{X} \in \Gamma_{\mathrm{N}}. \tag{3.23c}$$

Equations (3.23) are called the **equations of equilibrium in the reference configuration**. With $\theta : \bar{\Omega} \to \mathbb{R}^3$ being smooth enough vector fields, equations (3.23a) and (3.23c) can be transformed into the equivalent variational formulation

$$\int_\Omega \boldsymbol{T} : \mathbf{Grad}(\boldsymbol{\theta})\,\mathrm{d}X = \int_\Omega \boldsymbol{f}\cdot\boldsymbol{\theta}\,\mathrm{d}X + \int_{\Gamma_{\mathrm{N}}} \boldsymbol{g}\cdot\boldsymbol{\theta}\,\mathrm{d}A, \qquad \boldsymbol{\theta} = \boldsymbol{0} \text{ on } \Gamma_{\mathrm{D}}, \tag{3.24}$$

known as the **principle of virtual work in the reference configuration**. For proofs of this equivalence and of Theorem 3.3, see Ciarlet [52, Theorem 2.6-1].

Relation (3.23b) shows that the first Piola-Kirchhoff stress tensor is in general not symmetric. This motivates the definition of the **second Piola-Kirchhoff stress tensor**

$$\boldsymbol{S} = \boldsymbol{F}^{-1}\boldsymbol{T}.$$

Substituting $\boldsymbol{T}$ with $\boldsymbol{F}\boldsymbol{S}$ in equation (3.23b) yields

$$\boldsymbol{S}^{\mathsf{T}} = \boldsymbol{S}, \tag{3.25}$$

showing the symmetry of the second Piola-Kirchhoff stress tensor. The equations of equilibrium in the reference configuration and the corresponding weak formulation can now be expressed in terms of the second Piola-Kirchhoff stress tensor, as well: We simply replace equation (3.23b) by equation (3.25) and every occurrence of $\boldsymbol{T}$ in equations (3.23) and (3.24) by $\boldsymbol{F}\boldsymbol{S}$.

3.1.3 Constitutive Laws

It is intuitively clear that a body consisting of steel reacts differently to an applied force than an equally shaped body consisting of rubber. The model equations presented in the previous section do not contain any information about the material and are consequently incomplete. This is also evident from a mathematical point of view. There are nine unknowns in the equilibrium equations (3.23) – the three components of the deformation $\boldsymbol{\varphi}$ and six components of the first Piola-Kirchhoff stress tensor $\boldsymbol{T}$ (taking into consideration relation (3.23b) or the symmetry of the second Piola-Kirchhoff stress tensor). The equilibrium conditions (3.23a), however, only consist of three equations, thus representing an undetermined system. The six missing equations are now provided by specifying the dependence of the stress tensor on the deformation. The underlying continuum mechanical model, which this thesis confines to, has been described at the beginning of this chapter. The properties and restrictions accompanying this model are now expressed mathematically, finally resulting in concrete forms of the boundary value problem (3.23) which will serve as basis for the discretisation (cf. Section 3.2).

3.1.3.1 Elastic Material

A material is **elastic** if the second Piola-Kirchhoff stress tensor[9] in a point $\boldsymbol{X}$ only depends on the deformation gradient $\boldsymbol{F}(\boldsymbol{X}) = \mathbf{Grad}\big(\boldsymbol{\varphi}(\boldsymbol{X})\big)$ and the point $\boldsymbol{X}$ itself, i. e., if there is a mapping

$$\hat{\boldsymbol{S}} : \bar{\Omega} \times \mathbb{M}^3_+ \to \mathbb{M}^3,$$

called the **response function for the second Piola-Kirchhoff stress tensor**, such that the **constitutive equation**

$$\boldsymbol{S}(\boldsymbol{X}) = \hat{\boldsymbol{S}}\big(\boldsymbol{X}, \boldsymbol{F}(\boldsymbol{X})\big), \qquad \boldsymbol{X} \in \bar{\Omega}$$

holds.[10] If the response function is independent of the specific point $\boldsymbol{X}$ of the reference configuration, i. e., if $\boldsymbol{S}(\boldsymbol{X}) = \hat{\boldsymbol{S}}\big(\boldsymbol{F}(\boldsymbol{X})\big), \boldsymbol{X} \in \bar{\Omega}$, then the material is called *homogeneous*. To simplify notations we assume from now on the material to be homogeneous and consequently omit the first variable of the response function. The following results, however, are also valid for inhomogeneous materials.

The constitutive equation is significantly simplified when the material is *isotropic*, which intuitively means that a material point responds identically in all directions. The *axiom of material frame-indifference*, which holds for *all* materials, further restricts the form of the response function: When a deformed configuration is rotated around the origin, then the axiom says that the Cauchy stress vector is transformed by the same rotation. For the mathematical definitions

[9] Elastic material can analogously be defined via the Cauchy or the first Piola-Kirchhoff stress tensor.

[10] While the axioms of force and momentum balance and the resulting equilibrium conditions are widely accepted and used as the basis for continuum mechanical problems, there are possibilities to vary or extend the characterisation of elastic materials. For example, in *nonlocal elasticity* the stress tensor $\boldsymbol{S}(\boldsymbol{X})$ is assumed to additionally depend on all other points $\boldsymbol{Y} \in \bar{\Omega}$, while the theory of *elastic materials of second grade* incorporates higher-order derivatives of the deformation tensor. For references in this regard, see Ciarlet [52, p. 93].

of frame-indifference and isotropy, see Ciarlet [52, pages 104 and 106]. One can show [52, Theorem 3.6-2] that the response function of a frame-indifferent, homogeneous and isotropic material can be expressed in the simple form

$$\hat{\boldsymbol{S}}(\boldsymbol{F}) = \tilde{\boldsymbol{S}}(\boldsymbol{C}) = \gamma_0(\mathfrak{u}_{\boldsymbol{C}})\boldsymbol{I} + \gamma_1(\mathfrak{u}_{\boldsymbol{C}})\boldsymbol{C} + \gamma_2(\mathfrak{u}_{\boldsymbol{C}})\boldsymbol{C}^2,$$

where γ_i, $i = 0,1,2$, are real-valued functions of the three principal invariants[11] of the matrix $\boldsymbol{C} = \boldsymbol{F}^{\mathsf{T}}\boldsymbol{F}$. To further concretise the representation of the stress tensor we assume that the reference configuration of the body is *stress-free*, i. e., that the *residual stress tensor* $\boldsymbol{T}_{\mathrm{R}} := \hat{\boldsymbol{S}}(\boldsymbol{I}) = \tilde{\boldsymbol{S}}(\boldsymbol{I})$ is zero in all points. In this case the reference configuration is called a *natural state*. Note that a reference configuration subjected to a rotation is still in a natural state. Now we take a closer look at the response function $\tilde{S}$ in the case of a deformation 'near a natural state'. The Green-St. Venant strain tensor $\boldsymbol{E} = \frac{1}{2}(\boldsymbol{C} - \boldsymbol{I})$ (see definition (3.15) on page 115) measures the deviation of the deformation from a rigid body transformation. Accordingly, it makes sense to consider the deviation $\tilde{S}(\boldsymbol{I}+2\boldsymbol{E}) - \tilde{S}(\boldsymbol{I})$ by computing a Taylor expansion around $\tilde{S}(\boldsymbol{I})$ and considering the first order terms. Doing so results in the following theorem:

THEOREM 3.4

Let there be a solid body which consists of frame-indifferent, homogeneous, isotropic, elastic material and whose reference configuration is in natural state. If the functions γ_i, $i = 0,1,2$, are differentiable at the point $\mathfrak{u}_{\boldsymbol{I}} = (3,3,1)$ (the three principal invariants of the identity matrix), then there are two constants μ and λ such that the response function can be written as[12]

$$\hat{\boldsymbol{S}}(\boldsymbol{F}) = \tilde{S}(\boldsymbol{C}) = \check{S}(\boldsymbol{E}) = 2\mu\boldsymbol{E} + \lambda\,\mathrm{tr}(\boldsymbol{E})\boldsymbol{I} + o(\boldsymbol{E}). \tag{3.26}$$

For a proof, see Ciarlet [52, Theorems 3.7-1 and 3.8-1].

Remarkable about this representation is the reduction to only *two* material constants, which are called the **Lamé constants**. Furthermore, the constant μ is called the **shear modulus** of the material. 'Ideal' experiments, which are realised by performing special deformations (pure shear, pure traction, pure compression), show that both constants must be positive. Such experiments are actually used in practice to determine numerical values of the two constants for

[11] The **three principal invariants** $\mathfrak{u}_{\boldsymbol{A}} = \big(\mathfrak{u}_1(\boldsymbol{A}), \mathfrak{u}_2(\boldsymbol{A}), \mathfrak{u}_3(\boldsymbol{A})\big)$ of a matrix $\boldsymbol{A} \in \mathbb{M}^3$ are the coefficients of the characteristic polynomial $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = -\lambda^3 + \mathfrak{u}_1(\boldsymbol{A})\lambda^2 - \mathfrak{u}_2(\boldsymbol{A})\lambda + \mathfrak{u}_3(\boldsymbol{A})$. They can be expressed in terms of the eigenvalues λ_1, λ_2 and λ_3 of the matrix $\boldsymbol{A}$:

$$\begin{aligned}
\mathfrak{u}_1(\boldsymbol{A}) &= \mathrm{tr}(\boldsymbol{A}) = \lambda_1 + \lambda_2 + \lambda_3 \\
\mathfrak{u}_2(\boldsymbol{A}) &= \frac{1}{2}\big(\mathrm{tr}(\boldsymbol{A})^2 - \mathrm{tr}(\boldsymbol{A}^2)\big) = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1 \\
&= \det(\boldsymbol{A})\,\mathrm{tr}(\boldsymbol{A}^{-\mathsf{T}}) \quad \text{(if } \boldsymbol{A} \text{ invertible)} \\
\mathfrak{u}_3(\boldsymbol{A}) &= \det(\boldsymbol{A}) = \lambda_1\lambda_2\lambda_3.
\end{aligned}$$

[12] The definition of the 'o-notation' $o(\boldsymbol{E})$ is analogous to that in footnote 8 on page 114.

a given material. The properties of an elastic material can also be specified in terms of two other constants, the **Poisson ratio** ν and the **Young modulus** E. The relation between the four constants is given by the following equations:

$$\begin{aligned} E &= \frac{\mu(3\lambda+2\mu)}{\lambda+\mu}, & \mu &= \frac{E}{2(1+\nu)}, \\ \nu &= \frac{\lambda}{2(\lambda+\mu)}, & \lambda &= \frac{E\nu}{(1+\nu)(1-2\nu)}. \end{aligned} \tag{3.27}$$

From $\mu, \lambda > 0$, $E > 0$ and $0 < \nu < 0.5$ follows. The limit case of $\nu = 0.5$ is considered at the end of this section.

Equation (3.26) straightforwardly leads to one of the most popular materials in actual computations, the **St. Venant-Kirchhoff material**. Its response function is given by simply omitting the remainder in equation (3.26),

$$\check{S}(\boldsymbol{E}) = \tilde{S}(\boldsymbol{I}+2\boldsymbol{E}) = 2\mu\boldsymbol{E} + \lambda\,\mathrm{tr}(\boldsymbol{E})\boldsymbol{I}, \tag{3.28}$$

thus defining a frame-indifferent, homogeneous, isotropic, elastic material with reference configuration at natural state. Due to its definition, it is clear that this material model is only valid for 'small' strains. It is noteworthy that, although the relation $\boldsymbol{E} \to \check{S}(\boldsymbol{E})$ is obviously linear, the St. Venant-Kirchhoff material model is still nonlinear in terms of the displacements since the kinematic relation (3.15) is nonlinear.

3.1.3.2 Hyperelasticity

To overcome the limitation to small strains we take a different view at the deformation problem, namely as an energy minimisation problem. We once again consider the applied forces (cf. equations (3.16), (3.17), (3.22)). If the body force is a dead load, we define the functional

$$F : \{\boldsymbol{\psi} : \bar{\Omega} \to \mathbb{R}^3\} \to F(\boldsymbol{\psi}) = \int_\Omega \hat{F}(\boldsymbol{X}, \boldsymbol{\psi}(\boldsymbol{X}))\,\mathrm{d}X$$

where

$$\hat{F} : \Omega \times \mathbb{R}^3 \to \mathbb{R}, \qquad \hat{F}(\boldsymbol{X}, \boldsymbol{\eta}) = \boldsymbol{f}(\boldsymbol{X}) \cdot \boldsymbol{\eta}$$

is the **potential of the applied body force**.[13] Then, we can write the volume integral on the right hand side of the principle of virtual work (3.24) as the Gâteaux derivative[14] of the functional F:

$$F'(\boldsymbol{\varphi})\boldsymbol{\theta} = \int_\Omega \hat{F}(\boldsymbol{X}, \boldsymbol{\theta}(\boldsymbol{X}))\,\mathrm{d}X = \int_\Omega \boldsymbol{f}(\boldsymbol{X}) \cdot \boldsymbol{\theta}(\boldsymbol{X})\,\mathrm{d}X.$$

[13] This definition can be extended to forces that are not dead loads, leading to the definition of *conservative forces* (see Ciarlet [52, pp. 81f]).

[14] The *Gâteaux* (or *directional*) *derivative* of a function f at the point $\boldsymbol{a}$ in direction of the vector $\boldsymbol{h}$ is given by

$$f'(\boldsymbol{a})\boldsymbol{h} = \frac{\mathrm{d}}{\mathrm{d}t} f(\boldsymbol{a}+t\boldsymbol{h})\Big|_{t=0}$$

Analogously, let G be a functional whose Gâteaux derivative yields the surface integral on the right hand side of the principle of virtual work. Then it is desirable to also express the left hand side in terms of a functional W, which motivates the following definition: A homogeneous elastic material with response function $\hat{\boldsymbol{T}}$ is **hyperelastic** if there is a differentiable function $\hat{W} : \mathbb{M}^3_+ \to \mathbb{R}$ such that

$$\hat{\boldsymbol{T}}(\boldsymbol{F}) = \frac{\partial \hat{W}}{\partial \boldsymbol{F}}(\boldsymbol{F}), \qquad \boldsymbol{F} \in \mathbb{M}^3_+.$$

The potential $\hat{W}$ is called the **stored energy function** and describes the locally stored density of elastic energy during the deformation process. Deriving the response function from a potential allows for the following physical interpretation: The energy needed to deform a hyperelastic body from state A into state B is independent of the deformation path. Consequently, a cyclic deformation process (e. g., from state A over state B back to state A) does not dissipate energy.

The desired functional W can then be defined by

$$W(\boldsymbol{\psi}) = \int_\Omega \hat{W}(\mathbf{Grad}(\boldsymbol{\psi}))\,\mathrm{d}X$$

and is denoted as **strain energy**. Physically, it can be interpreted as the amount of work which is necessary to bring the hyperelastic body into its current state. In conjunction with the functionals F and G it determines the **total energy**

$$I(\boldsymbol{\psi}) = W(\boldsymbol{\psi}) - \big(F(\boldsymbol{\psi}) + G(\boldsymbol{\psi})\big).$$

Now, it can be shown that the principle of virtual work (3.24) is equivalent to the condition

$$I'(\boldsymbol{\varphi})\boldsymbol{\theta} = 0, \qquad \boldsymbol{\theta} = \mathbf{0} \text{ on } \Gamma_\mathrm{D}$$

with $\boldsymbol{\theta} : \Omega \to \mathbb{R}^3$ being smooth enough (see Ciarlet [52, Theorem 4.1-1]). This leads to the following important theorem:

THEOREM 3.5

Let a hyperelastic, homogeneous material be subjected to body and surface forces that are dead loads. A sufficiently smooth mapping $\boldsymbol{\varphi}$ that satisfies

$$\boldsymbol{\varphi} \in \boldsymbol{\Phi} := \{\boldsymbol{\psi} : \bar{\Omega} \to \mathbb{R}^3 \mid \boldsymbol{\psi} = \boldsymbol{\varphi}_0 \text{ on } \Gamma_\mathrm{D}\}$$

and

$$I(\boldsymbol{\varphi}) = \inf_{\boldsymbol{\psi} \in \boldsymbol{\Phi}} I(\boldsymbol{\psi}),$$

solves the following boundary value problem:

$$-\mathbf{Div}\left[\frac{\partial \hat{W}}{\partial \boldsymbol{F}}\Big(\mathbf{Grad}\big(\boldsymbol{\varphi}(\boldsymbol{X})\big)\Big)\right] = \boldsymbol{f}(\boldsymbol{X}), \qquad \boldsymbol{X} \in \Omega, \tag{3.29a}$$

$$\boldsymbol{\varphi}(\boldsymbol{X}) = \boldsymbol{\varphi}_0(\boldsymbol{X}), \qquad \boldsymbol{X} \in \Gamma_\mathrm{D}, \tag{3.29b}$$

$$\frac{\partial \hat{W}}{\partial \boldsymbol{F}}\Big(\mathbf{Grad}\big(\boldsymbol{\varphi}(\boldsymbol{X})\big)\Big)\boldsymbol{n} = \boldsymbol{g}(\boldsymbol{X}), \qquad \boldsymbol{X} \in \Gamma_\mathrm{N}. \tag{3.29c}$$

For a proof, see Ciarlet [52, Theorems 4.1-1 and 4.1-2].

This theorem states the intuitive idea that the solution of a deformation problem describes a state of minimal energy.

Corresponding to the definitions for response functions of elastic materials (cf. Section 3.1.3.1), isotropy and frame-indifference of the stored energy function lead to the representation $\hat{W}(\boldsymbol{F}) = \tilde{W}(\boldsymbol{F}^{\mathsf{T}}\boldsymbol{F})$. This form corresponds to the response function of the symmetric second Piola-Kirchhoff stress tensor via

$$\hat{\boldsymbol{S}}(\boldsymbol{F}) = \tilde{\boldsymbol{S}}(\boldsymbol{C}) = 2\frac{\partial \tilde{W}}{\partial \boldsymbol{C}}(\boldsymbol{C}), \qquad \boldsymbol{F} \in \mathbb{M}^3_+$$

(see Ciarlet [52, Theorem 4.2-2]). The stored energy function can equivalently be expressed in terms of the Green-St. Venant strain tensor $\boldsymbol{E} = \frac{1}{2}(\boldsymbol{F}^{\mathsf{T}}\boldsymbol{F} - \boldsymbol{I})$, i. e.,

$$\hat{\boldsymbol{S}}(\boldsymbol{F}) = \check{\boldsymbol{S}}(\boldsymbol{E}) = \frac{\partial \check{W}}{\partial \boldsymbol{E}}(\boldsymbol{E}), \qquad \boldsymbol{F} \in \mathbb{M}^3_+.$$

Analogously to Section 3.1.3.1 it can be shown that a frame-indifferent, isotropic stored energy function of a hyperelastic material only depends on the three principal invariants of the matrix $\boldsymbol{C} = \boldsymbol{F}^{\mathsf{T}}\boldsymbol{F}$, i. e.,

$$\hat{W}(\boldsymbol{F}) = \dot{W}(\mathfrak{i}_{\boldsymbol{C}}), \qquad \boldsymbol{F} \in \mathbb{M}^3_+.$$

The response function of the second Piola-Kirchhoff stress tensor can then be written as

$$\begin{aligned}
\tilde{\boldsymbol{S}}(\boldsymbol{C}) &= 2\frac{\partial \tilde{W}}{\partial \boldsymbol{C}}(\boldsymbol{C}) = 2\frac{\partial \dot{W}}{\partial \boldsymbol{C}}(\mathfrak{i}_1(\boldsymbol{C}), \mathfrak{i}_2(\boldsymbol{C}), \mathfrak{i}_3(\boldsymbol{C})) \\
&= 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_1}\frac{\partial \mathfrak{i}_1}{\partial \boldsymbol{C}} + 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_2}\frac{\partial \mathfrak{i}_2}{\partial \boldsymbol{C}} + 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_3}\frac{\partial \mathfrak{i}_3}{\partial \boldsymbol{C}} \\
&= 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_1}\boldsymbol{I} + 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_2}\big(\mathfrak{i}_1(\boldsymbol{C})\boldsymbol{I} - \boldsymbol{C}\big) + 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_3}\mathfrak{i}_3(\boldsymbol{C})\boldsymbol{C}^{-1} \\
&= 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_1}\boldsymbol{I} + 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_2}\big(\operatorname{tr}(\boldsymbol{C})\boldsymbol{I} - \boldsymbol{C}\big) + 2\frac{\partial \dot{W}}{\partial \mathfrak{i}_3}\det(\boldsymbol{C})\boldsymbol{C}^{-1},
\end{aligned} \tag{3.30}$$

where we use $\frac{\partial \dot{W}}{\partial \mathfrak{i}_i}$ as short form of $\frac{\partial \dot{W}}{\partial \mathfrak{i}_i}(\mathfrak{i}_{\boldsymbol{C}})$ (cf. Ciarlet [52, Theorems 4.4-1 and 4.4-2]). With this representation at hand, we can construct, analogously to Theorem 3.4, a simple form of the stored energy function for isotropic, hyperelastic materials in the case of deformations 'near a natural state':

$$\check{W}(\boldsymbol{E}) = \mu \operatorname{tr}(\boldsymbol{E}^2) + \frac{\lambda}{2}\operatorname{tr}(\boldsymbol{E})^2 + o(\|\boldsymbol{E}\|^2).$$

Omitting the remainder[15] leads to the stored energy function of the St. Venant-Kirchhoff material (3.28),

$$\check{W}(\boldsymbol{E}) = \mu \operatorname{tr}(\boldsymbol{E}^2) + \frac{\lambda}{2}\operatorname{tr}(\boldsymbol{E})^2, \tag{3.31}$$

which shows that this material is hyperelastic (see Ciarlet [52, Theorem 4.4-3 and 4.5-1]).

[15] For the definition of the 'o-notation' $o(\|\boldsymbol{E}\|^2)$ see footnote 8 on page 114.

We now want to express the already stated limitation of the St. Venant-Kirchhoff material to *small strains* in terms of the stored energy function. This motivates the definition of a constitutive equation which is suitable for *large strains*, as well. Apparently, states of extreme strains should involve extreme stresses. For example, when compressing a material in such a way that $\det(\boldsymbol{F}) \to 0$, then the stress should tend to infinity. In other words, infinite energy should be necessary to cancel volumes. This intuitive idea has to be reflected by the stored energy function. Mathematically, we can express this via the following two assumptions how the stored energy function should behave for large strains:[16]

$$\begin{aligned} \det(\boldsymbol{F}) \to 0 \quad &\Rightarrow \quad \hat{W}(\boldsymbol{F}) \to +\infty, \qquad \boldsymbol{F} \in \mathbb{M}^3_+ \\ \|\boldsymbol{F}\| + \|\mathbf{Cof}\,\boldsymbol{F}\| + \det(\boldsymbol{F}) \to +\infty \quad &\Rightarrow \quad \hat{W}(\boldsymbol{F}) \to +\infty, \qquad \boldsymbol{F} \in \mathbb{M}^3_+ \end{aligned}$$

Considering the St. Venant-Kirchhoff material, it is obvious that it is not able to meet the first assumption as its stored energy function (3.31) does not explicitly depend on $\det(\boldsymbol{F})$. An example of a hyperelastic material which fulfils both assumptions and which still has a quite simple form and is thus often used in practice, is the **Neo-Hooke material**. It does not depend on the second principal invariant $\mathrm{I}_2(\boldsymbol{C})$ and is usually expressed in terms of $J = \det(\boldsymbol{F})$ instead of $\det(\boldsymbol{C})$. Representation (3.30) thus simplifies to

$$\tilde{\boldsymbol{S}}(\boldsymbol{C}) = 2\frac{\partial \ddot{W}}{\partial \boldsymbol{C}}(\mathrm{I}_1(\boldsymbol{C}), J) = 2\frac{\partial \ddot{W}}{\partial \mathrm{I}_1}\boldsymbol{I} + \frac{\partial \ddot{W}}{\partial J}J\boldsymbol{C}^{-1}.$$

We choose a variant of Neo-Hooke material which is frequently used in the literature [133, 145, 176]. Its stored energy function is given by

$$\ddot{W}(\mathrm{I}_1(\boldsymbol{C}), J) = \frac{\mu}{2}\big(\mathrm{tr}(\boldsymbol{C}) - 3 - 2\ln(J)\big) + \frac{\lambda}{4}\big(J^2 - 1 - 2\ln(J)\big), \tag{3.32}$$

yielding the following form of the second Piola-Kirchhoff stress tensor:

$$\boldsymbol{S} = \mu(\boldsymbol{I} - \boldsymbol{C}^{-1}) + \frac{\lambda}{2}(J - \frac{1}{J})J\boldsymbol{C}^{-1}. \tag{3.33}$$

Due to the logarithmic terms, the stored energy function tends to infinity when $\det \boldsymbol{F} \to 0$, and additionally it provides a stress-free reference configuration ($\boldsymbol{S} = \boldsymbol{0}$ for $\boldsymbol{C} = \boldsymbol{I}$).

3.1.3.3 (Nearly) Incompressible Materials

Some materials react to volume changes with a large increase of stored energy. Such materials are called **nearly incompressible**. In the limit case, when the material does not allow any change of volume, it is called **incompressible**. Mathematically, this constraint is expressed in terms of the determinant of the deformation gradient,

$$\det \boldsymbol{F} = 1. \tag{3.34}$$

[16] $\mathbf{Cof}\,\boldsymbol{F} := \det(\boldsymbol{F})\boldsymbol{F}^{-\mathsf{T}}$ is the **cofactor matrix** of $\boldsymbol{F}$.

For nearly incompressible material, i. e., when $J = \det \boldsymbol{F}$ is close to 1, it is obvious from the second term of equation (3.32) that a large increase of energy due to volume changes is only possible if the material constant λ is large. Consequently, this constant can be interpreted as reflecting how the material reacts to compression – the larger λ, the 'less compressible' the material. The degree of incompressibility can also be expressed in terms of the Poisson ratio ν. From equation (3.27) follows (for fixed shear modulus μ) the relation

$$\nu \to 0.5 \Leftrightarrow \lambda \to \infty.$$

Consequently, a Poisson ratio of $\nu = 0.5$ characterises incompressible material which makes it very convenient in a computer program to switch from nearly incompressible material, e. g., $\nu = 0.49999$ to incompressible material, $\nu = 0.5$.

In the case of the incompressible limit, corresponding to $J = 1$ and '$\lambda = \infty$', the stress tensor reduces to $\boldsymbol{S} = \mu(\boldsymbol{I} - \boldsymbol{C}^{-1})$, and the computation has to be accompanied by a Lagrange multiplier p to enforce the constraint $J = 1$. This additional unknown can be interpreted as *pressure*. In the nearly incompressible case, however, imposing the condition $J \approx 1$ is also a severe constraint to the class of admissible deformations, and it is intuitively clear that the problem of finding a solution for the boundary value problem (3.29) is overconstrained. When not accounting for this problem, the effect is clearly observable in actual computations and is widely known as *volume locking*: The loaded body behaves much stiffer than it actually is and consequently shows too small deformations in response. A mathematical explanation of this effect will be given for the linear case in Section 3.2.1.

As in the incompressible case, a possibility to overcome this problem is to use an additional pressure variable

$$p := -\frac{\lambda}{2}(J - \frac{1}{J}) \tag{3.35}$$

and replace the 'compressibility part' of the second Piola-Kirchhoff stress tensor (3.33) accordingly:[17]

$$\boldsymbol{S} = \mu(\boldsymbol{I} - \boldsymbol{C}^{-1}) - pJ\boldsymbol{C}^{-1} \tag{3.36}$$

The boundary value problem (3.29) is extended by relation (3.35), the so called **continuity equation** (see equation (3.43b) on page 130). We call the resulting problems (3.44) and (3.49) **mixed $\boldsymbol{u}/p$ formulation**, or simply **mixed formulation**.[18] Accordingly, the problems (3.40) and (3.47) containing the displacements as only unknowns are denoted as **displacement formulation**. The advantage of the mixed formulation is, that for actual computations one does

[17] The Cauchy stress tensor corresponding to the second Piola-Kirchhoff stress tensor (3.36) has the form

$$\boldsymbol{T}^{\mathsf{c}} = \mu(\boldsymbol{F}\boldsymbol{F}^{\mathsf{T}} - \boldsymbol{I}) - p\boldsymbol{I}.$$

This shows that the definition (3.35) represents the 'real' pressure in the deformed configuration. When regarding the variable p merely as a Lagrange multiplier (ignoring its physical interpretation), then one could, for example, also set $p = -\frac{\lambda}{2}(J^2 - 1)$ or $p = -\lambda(J - 1)$.

[18] A similar strategy can be applied to treat (nearly) incompressible St. Venant-Kirchhoff material: One can, e. g., define $p := -\lambda \operatorname{tr}(\boldsymbol{E})$ and replace the second Piola-Kirchhoff stress tensor (3.28) by $\boldsymbol{S} = 2\mu\boldsymbol{E} - p\boldsymbol{I}$.

not have to distinguish anymore between nearly incompressible and incompressible material – in the latter case, all terms containing the constant λ simply vanish from the equations. In addition, the formulation can just as well be applied to the compressible case ($\nu \approx 0.3$). Of course, due to the additional unknowns it is computationally more expensive, but in terms of accuracy it might be advantageous over the displacement formulation. This has been observed, for instance, by Suttmeier [153] and will be confirmed in Section 3.3.5.

3.1.3.4 Examples of Elastic Materials

We want to finish this section about constitutive laws by listing the physical properties of some common elastic materials (see Table 3.1). Some of the numerical simulation examples in this

Material	E [Pa]	ν [-]	μ [Pa]	λ [Pa]	ρ [$\frac{\text{kg}}{\text{m}^3}$]
Steel/Iron	$2.1 \cdot 10^{11}$	0.29	$8.14 \cdot 10^{10}$	$1.1 \cdot 10^{11}$	$7.8 \cdot 10^{3}$
Aluminium	$7.0 \cdot 10^{10}$	0.35	$2.6 \cdot 10^{10}$	$6.0 \cdot 10^{10}$	$2.7 \cdot 10^{4}$
Lead	$1.6 \cdot 10^{10}$	0.44	$5.6 \cdot 10^{9}$	$4.1 \cdot 10^{10}$	$1.34 \cdot 10^{4}$
Rubber	$2.5 \cdot 10^{7}$	0.499–0.5	$8.2 \cdot 10^{6}$	$4.1 \cdot 10^{9}$	$1.0 \cdot 10^{3}$

Table 3.1: Some average material constants. Poisson ratio ν, shear modulus μ and density ρ serve as input for the program FEASTsolid.

work use these materials. The rubber material is representative for the wide class of natural and synthetic elastomers like caoutchouc or polybutadiene with varying degrees of compressibility. Table 3.2 lists some physical quantities and relations.

velocity	$\frac{\text{m}}{\text{s}}$	length per time
acceleration	$\frac{\text{m}}{\text{s}^2}$	length per time squared
force	$\text{N} = \text{kg}\frac{\text{m}}{\text{s}^2}$	mass $\times$ acceleration (Newton)
density	$\frac{\text{kg}}{\text{m}^3}$	mass per volume
body force	$\frac{\text{N}}{\text{m}^3} = \frac{\text{m}}{\text{s}^2}\frac{\text{kg}}{\text{m}^3}$	force per volume *or* acceleration $\times$ density
surface force (pressure)	$\text{Pa} = \frac{\text{N}}{\text{m}^2}$	force per area (Pascal)

Table 3.2: Physical quantities and their units.

3.1.4 Transient Problems

The physical model considered so far completely neglects dynamic effects which occur due to external loads that strongly variate in time. Actually, each deformation process is time-dependent, but when loads are applied 'very slowly' and do not further change over time, then

one can neglect the 'intermediate path' of the deformation and only compute the final static state. This justifies the elastostatic formulation we considered in the previous sections.

In elastodynamics the deformation function $\boldsymbol{\varphi}$ additionally depends on the time variable t, i. e.,

$$\boldsymbol{\varphi} : [0,T] \times \bar{\Omega} \to \mathbb{R}^3, \qquad (t, \boldsymbol{X}) \mapsto \boldsymbol{\varphi}(t, \boldsymbol{X}),$$

where $I := [0,T]$ is the time interval of interest and the reference configuration coincides with the body at time $t = 0$. Inertial forces enter the equilibrium equation (3.29a) as additional body forces (D'Alembert's principle):

$$\rho \ddot{\boldsymbol{\varphi}} - \mathbf{Div}\left(\frac{\partial \hat{W}}{\partial \boldsymbol{F}}\right) = \boldsymbol{f} \qquad \text{in } I \times \Omega,$$

where $\ddot{\boldsymbol{\varphi}}$ is the acceleration and ρ the density of the body in the reference configuration. The weak formulation is extended accordingly (see equations (3.47) and (3.49)).

3.1.5 Definition and Characterisation of the Boundary Value Problems

We now summarise the boundary value problems for the various continuous models described in the previous sections. We provide the according variational problem, which will serve as basis for the discretisation process (see Sections 3.2 and 3.4). From now on we shift the focus from the deformations $\boldsymbol{\varphi}$ to the displacements $\boldsymbol{u} = \boldsymbol{\varphi} - \mathbf{id}$ and consider the latter as the unknowns which are to be computed. Let there be given prescribed displacements $\bar{\boldsymbol{u}}$ imposed on the boundary part Γ_{D}, dead surface forces $\boldsymbol{g}$ acting on the boundary part Γ_{N}, and dead body forces $\boldsymbol{f}$. For transient problems, let $I = [0,T]$ be the time interval of interest, $\bar{\boldsymbol{u}}_0$ and $\bar{\boldsymbol{v}}_0$ the initial values for displacements and velocities, respectively, at time $t = 0$, and $\boldsymbol{u}_t(\boldsymbol{X}) := \boldsymbol{u}(t, \boldsymbol{X})$. With $\boldsymbol{V}$ and M we denote functional spaces for displacements and pressure, respectively, where we assume that they are suitably defined for each of the following problems (for details in this regard, see for example Ciarlet [52], Le Tallec [105] and Braess and Ming [34]). Furthermore, we define the following products

$$\big(\boldsymbol{A}, \boldsymbol{B}\big)_{\Omega} := \int_{\Omega} \boldsymbol{A}(\boldsymbol{X}) : \boldsymbol{B}(\boldsymbol{X}) \, \mathrm{d}X, \qquad \boldsymbol{A}, \boldsymbol{B} \in L^2(\Omega, \mathbb{M}^3) \tag{3.37a}$$

$$(\boldsymbol{a}, \boldsymbol{b})_{\Omega} := \int_{\Omega} \boldsymbol{a}(\boldsymbol{X}) \cdot \boldsymbol{b}(\boldsymbol{X}) \, \mathrm{d}X, \qquad \boldsymbol{a}, \boldsymbol{b} \in L^2(\Omega, \mathbb{R}^3) \tag{3.37b}$$

$$(\boldsymbol{a}, \boldsymbol{b})_{\Gamma_{\mathrm{N}}} := \int_{\Gamma_{\mathrm{N}}} \boldsymbol{a}(\boldsymbol{X}) \cdot \boldsymbol{b}(\boldsymbol{X}) \, \mathrm{d}A, \qquad \boldsymbol{a}, \boldsymbol{b} \in L^2(\Gamma_{\mathrm{N}}, \mathbb{R}^3) \tag{3.37c}$$

and the functional of external loads

$$L(\boldsymbol{v}) := (\boldsymbol{f}, \boldsymbol{v})_{\Omega} + (\boldsymbol{g}, \boldsymbol{v})_{\Gamma_{\mathrm{N}}}. \tag{3.38}$$

For the sake of a compact presentation we briefly collect the basic definitions used in the following problem formulations:

- Deformation field (equation (3.1) on page 110): $\boldsymbol{\varphi} : \bar{\Omega} \to \mathbb{R}^3$
- Displacement field (equation (3.3) on page 111): $\boldsymbol{u} = \boldsymbol{\varphi} - \mathbf{id}$
- Deformation gradient (equation (3.2) on page 110): $\boldsymbol{F} = \mathbf{Grad}(\boldsymbol{\varphi})$
- Determinant of the deformation gradient: $J = \det(\boldsymbol{F})$
- Green-St. Venant strain tensor (equation (3.15) on page 115):

$$\boldsymbol{E} = \frac{1}{2}(\boldsymbol{F}^{\mathsf{T}}\boldsymbol{F} - \boldsymbol{I}) = \tfrac{1}{2}\big(\mathbf{Grad}(\boldsymbol{u}) + \mathbf{Grad}(\boldsymbol{u})^{\mathsf{T}} + \mathbf{Grad}(\boldsymbol{u})^{\mathsf{T}}\,\mathbf{Grad}(\boldsymbol{u})\big)$$

- Pressure field (equation (3.35) on page 126): $p = -\frac{\lambda}{2}(J - \frac{1}{J})$
- First Piola-Kirchhoff stress tensor (equation (3.21) on page 118): $\boldsymbol{T} = J\boldsymbol{T}^{\mathsf{c}}\boldsymbol{F}^{-\mathsf{T}}$ with $\boldsymbol{T}^{\mathsf{c}}$ denoting the Cauchy stress tensor (see Theorem 3.2 on page 117)

3.1.5.1 The static compressible case

Boundary value problem:

$$-\mathbf{Div}(\boldsymbol{T}) = \boldsymbol{f}, \qquad \text{in } \Omega \tag{3.39a}$$
$$\boldsymbol{u} = \bar{\boldsymbol{u}}, \qquad \text{on } \Gamma_{\mathrm{D}} \tag{3.39b}$$
$$\boldsymbol{T}\boldsymbol{n} = \boldsymbol{g}, \qquad \text{on } \Gamma_{\mathrm{N}} \tag{3.39c}$$

Variational problem:

Find $\boldsymbol{u} - \bar{\boldsymbol{u}} \in \boldsymbol{V}$ such that

$$\big(\boldsymbol{T}, \mathbf{Grad}(\boldsymbol{v})\big)_{\Omega} = L(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V} \tag{3.40}$$

St. Venant-Kirchhoff material:

$$\boldsymbol{T} = \boldsymbol{F}\boldsymbol{S} = 2\mu\boldsymbol{F}\boldsymbol{E} + \lambda\,\mathrm{tr}(\boldsymbol{E})\boldsymbol{F} \tag{3.41}$$

Neo-Hooke material:

$$\boldsymbol{T} = \mu(\boldsymbol{F} - \boldsymbol{F}^{-\mathsf{T}}) + \frac{\lambda}{2}(J^2 - 1)\boldsymbol{F}^{-\mathsf{T}} \tag{3.42}$$

3.1.5.2 The static (nearly) incompressible case

Boundary value problem:

$$-\mathbf{Div}(\boldsymbol{T}) = \boldsymbol{f}, \qquad \text{in } \Omega \tag{3.43a}$$
$$-\tfrac{1}{2}\big(J-\tfrac{1}{J}\big) - \tfrac{1}{\lambda}p = 0 \qquad \text{in } \Omega, \tag{3.43b}$$
$$\boldsymbol{u} = \bar{\boldsymbol{u}}, \qquad \text{on } \Gamma_{\mathrm{D}} \tag{3.43c}$$
$$\boldsymbol{T}\boldsymbol{n} = \boldsymbol{g}, \qquad \text{on } \Gamma_{\mathrm{N}} \tag{3.43d}$$

Variational problem:

Find $(\boldsymbol{u}-\bar{\boldsymbol{u}}, p) \in \boldsymbol{V} \times M$ such that

$$\big(\boldsymbol{T}, \mathbf{Grad}(\boldsymbol{v})\big)_{\Omega} = L(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V} \tag{3.44a}$$
$$-\big(\tfrac{1}{2}\big(J-\tfrac{1}{J}\big), q\big)_{\Omega} - \tfrac{1}{\lambda}(p,q)_{\Omega} = 0, \qquad q \in M \tag{3.44b}$$

Neo-Hooke material:

$$\boldsymbol{T} = \mu(\boldsymbol{F} - \boldsymbol{F}^{-\mathsf{T}}) - pJ\boldsymbol{F}^{-\mathsf{T}} \tag{3.45}$$

Note, that the terms $\frac{1}{\lambda}p$ in equation (3.43b) and $\frac{1}{\lambda}(p,q)_{\Omega}$ in equation (3.44b) vanish in the case of incompressible material.

3.1.5.3 The transient compressible case

Initial boundary value problem:

$$\rho\ddot{\boldsymbol{u}} - \mathbf{Div}(\boldsymbol{T}) = \boldsymbol{f} \qquad \text{in } I \times \Omega \tag{3.46a}$$
$$\boldsymbol{u}_0 = \bar{\boldsymbol{u}}_0, \dot{\boldsymbol{u}}_0 = \bar{\boldsymbol{v}}_0 \qquad \text{in } \Omega \tag{3.46b}$$
$$\boldsymbol{u} = \bar{\boldsymbol{u}}, \qquad \text{in } I \times \Gamma_{\mathrm{D}} \tag{3.46c}$$
$$\boldsymbol{T}\boldsymbol{n} = \boldsymbol{g}, \qquad \text{in } I \times \Gamma_{\mathrm{N}} \tag{3.46d}$$

Variational problem:

Find $\boldsymbol{u} - \bar{\boldsymbol{u}} \in L^2(I, \boldsymbol{V})$ such that

$$\int_0^T \Big[\rho(\ddot{\boldsymbol{u}}, \boldsymbol{v})_{\Omega} + \big(\boldsymbol{T}, \mathbf{Grad}(\boldsymbol{v})\big)_{\Omega}\Big]\,\mathrm{d}t = \int_0^T L(\boldsymbol{v})\,\mathrm{d}t, \qquad \boldsymbol{v} \in L^2(I, \boldsymbol{V}) \tag{3.47}$$

(first Piola-Kirchhoff stress tensor $\boldsymbol{T}$ given by equation (3.41) or (3.42))

3.1.5.4 The transient (nearly) incompressible case

Initial boundary value problem:

$$\rho\ddot{\boldsymbol{u}} - \mathbf{Div}(\boldsymbol{T}) = \boldsymbol{f} \qquad \text{in } I\times\Omega \tag{3.48a}$$
$$-\tfrac{1}{2}\big(J-\tfrac{1}{J}\big) - \tfrac{1}{\lambda}p = 0 \qquad \text{in } I\times\Omega \tag{3.48b}$$
$$\boldsymbol{u}_0 = \bar{\boldsymbol{u}}_0, \dot{\boldsymbol{u}}_0 = \bar{\boldsymbol{v}}_0 \qquad \text{in } \Omega \tag{3.48c}$$
$$\boldsymbol{u} = \bar{\boldsymbol{u}}, \qquad \text{in } I\times\Gamma_{\mathrm{D}} \tag{3.48d}$$
$$\boldsymbol{T}\boldsymbol{n} = \boldsymbol{g}, \qquad \text{in } I\times\Gamma_{\mathrm{N}} \tag{3.48e}$$

Variational problem:

Find $(\boldsymbol{u}-\bar{\boldsymbol{u}}, p) \in L^2(I,\boldsymbol{V})\times L^2(I,M)$ such that

$$\int_0^T \big[\rho(\ddot{\boldsymbol{u}},\boldsymbol{v})_\Omega + \big(\boldsymbol{T},\mathbf{Grad}(\boldsymbol{v})\big)_\Omega\big]\,\mathrm{d}t = \int_0^T L(\boldsymbol{v})\,\mathrm{d}t, \qquad \boldsymbol{v}\in L^2(I,\boldsymbol{V}) \tag{3.49a}$$
$$\int_0^T \big[-\big(\tfrac{1}{2}\big(J-\tfrac{1}{J}\big),q\big)_\Omega - \tfrac{1}{\lambda}(p,q)_\Omega\big]\,\mathrm{d}t = 0, \qquad q\in L^2(I,M) \tag{3.49b}$$

(first Piola-Kirchhoff stress tensor $\boldsymbol{T}$ given by equation (3.45))

3.1.5.5 Characterisation of the Partial Differential Equations

The equilibrium equations occurring in the boundary value problems (3.39), (3.43), (3.46) and (3.48) build a *system of three nonlinear partial differential equations of second order* with respect to the displacements $u_i, i = 1,2,3$. Due to the divergence structure, the PDEs are *quasi-linear*, i. e., they depend linearly on the second-order derivatives (see Ciarlet [52, p. 251]), but they are *not semilinear* since the coefficients of the second-order derivatives depend on the unknown solutions.

Actually, the PDEs exhibit a highly nonlinear character. First, there is the **geometric nonlinearity** between the displacements and the strains in form of the Green-St. Venant strain tensor (3.15) (or, equivalently, between the deformations and the right Cauchy-Green strain tensor (3.12)). Then, the relation between the strains and the stresses is nonlinear in the case of the Neo-Hooke constitutive law (3.33), which is called **material nonlinearity**. Furthermore, the constraints $\det(\boldsymbol{F}) > 0$, specifying a deformation mapping, and $\det(\boldsymbol{F}) = 1$, characterising incompressible material, are nonlinear.[19]

Such nonlinear PDEs are not necessarily uniquely solvable. There are many classical examples of simple deformation problems which possess multiple solutions. However, the numerical

[19] Other sources of nonlinearity (which are not considered in this thesis) are external forces that depend on the deformation (i. e., that are not dead loads), and certain kinds of boundary conditions, for example, when contact with an obstacle occurs.

tests performed in this study are such that the difficulty of non-uniqueness does not arise. For details in this regard and for theoretical existence results we refer to Ciarlet [52] and Le Tallec [105].

3.1.6 Linearised Elasticity

Considering the highly nonlinear character of the elasticity problem, it seems to be physically questionable to treat it in a completely linear fashion. As suggested by Ciarlet [52, p. 286], one should avoid the term *linear elasticity*, since elasticity itself *is nonlinear*. The linear equations which are examined in this section result from a linearisation of the nonlinear problem near the origin. Accordingly, one should rather speak of *linearised elasticity* and keep in mind that the computed solutions only represent approximations to the 'real' displacements. Nevertheless, linearised elasticity is widely used in engineering applications to numerically approximate the solutions of solid mechanical problems. Likewise, our examinations of robust solving methods (see Chapters 4 and 5) will mainly focus on the linearised theory. This is justified, since the numerical strategy to solve nonlinear problems (see Section 4.3) in fact consists of solving a series of linear problems that represent linearisations around the current solution. These linear problems have the same structure as the ones to be introduced in this section. So, even if the following explanations might seem as a mere 'by-product' of the nonlinear theory and might be unrealistic from a physical point of view, they serve as basis for the development of our solution strategy and are thus of major importance for this work.

First, we linearise the kinematic relation between displacements and strains: We omit the quadratic terms of the Green-St. Venant strain tensor (3.15) and thus obtain the **linearised strain tensor**[20]

$$\boldsymbol{\varepsilon}(\boldsymbol{u}) := \tfrac{1}{2}\left(\boldsymbol{\nabla u} + \boldsymbol{\nabla u}^{\mathsf{T}}\right),$$

which is also referred to as *symmetric gradient* of $\boldsymbol{u}$. For details about this linearisation and the following ones we refer to Stein and Barthold [151, chapter 6]. Replacing the real strain tensor by its linearised version is only feasible when the displacements are small. Consequently, linearised elasticity requires the *assumption of small deformations*.[21]

Next, we need a linear stress-strain relation. The St. Venant-Kirchhoff constitutive law (3.28) fulfils this requirement, but is still nonlinear in terms of the displacements. So, we simply replace the Green-St. Venant strain tensor by the linearised strain tensor, which leads to **Hooke's law** for isotropic material:

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda\,\mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}. \tag{3.50}$$

The symmetric tensor $\boldsymbol{\sigma}$ is the linearisation of the second Piola-Kirchhoff stress tensor $\boldsymbol{S}$ defined by the response function (3.28). Linearising the corresponding first Piola-Kirchhoff stress

[20] For linearised elasticity, we use the symbol $\boldsymbol{\nabla}$ to denote the gradient. Furthermore, we do not distinguish between the coordinates of reference and deformed configuration, and use small letters ($\boldsymbol{x}$, **div**, etc.) only.

[21] When the small deformation assumption is dropped, this is then often emphasised in the literature by using the terms *large deformation* or *finite deformation*.

tensor $\boldsymbol{T}$ or the Cauchy stress tensor $\boldsymbol{T}^{\mathrm{c}}$ leads to the same result. So, we can interpret the tensor $\boldsymbol{\sigma}$ as an approximation of the 'real' physical Cauchy stresses occurring in the deformed body. The corresponding stored energy function is given by

$$\check{W}(\boldsymbol{\varepsilon}) = \mu \operatorname{tr}(\boldsymbol{\varepsilon}^2) + \tfrac{\lambda}{2} \operatorname{tr}(\boldsymbol{\varepsilon})^2$$

Finally, for the treatment of (nearly) incompressible material we have to translate the nonlinear continuity equation (3.43b) into a linear relation. The linearisation of the incompressibility constraint, $\det \boldsymbol{F} = 1$, results in the condition

$$\operatorname{div}(\boldsymbol{u}) = 0,$$

which is well known from the (Navier-)Stokes equations describing the flow of incompressible fluids [158]. Again, we emphasise that the condition $\operatorname{div}(\boldsymbol{u}) = 0$ characterises incompressibility only approximately. Nevertheless, for the sake of convenience we relax the notation for the linearised case and still use the terms *incompressible* for $\nu = 0.5$ and *nearly incompressible* when ν is close to 0.5.

With the definitions given at the beginning of Section 3.1.5 we can now define the boundary value problem of linearised elasticity for the compressible case, the so called **Lamé equation**:

$$-2\mu \operatorname{\mathbf{div}}(\boldsymbol{\varepsilon}) - \lambda \boldsymbol{\nabla} \operatorname{div}(\boldsymbol{u}) = \boldsymbol{f} \qquad \text{in } \Omega \tag{3.51a}$$

$$\boldsymbol{u} = \bar{\boldsymbol{u}} \qquad \text{on } \Gamma_{\mathrm{D}} \tag{3.51b}$$

$$\boldsymbol{\sigma n} = \boldsymbol{g} \qquad \text{on } \Gamma_{\mathrm{N}}. \tag{3.51c}$$

Defining the bilinear form

$$k(\boldsymbol{u}, \boldsymbol{v}) := 2\mu \big(\boldsymbol{\varepsilon}(\boldsymbol{u}), \boldsymbol{\varepsilon}(\boldsymbol{v})\big)_{\Omega} + \lambda \big(\operatorname{div}(\boldsymbol{u}), \operatorname{div}(\boldsymbol{v})\big)_{\Omega} \tag{3.52}$$

and the space

$$\boldsymbol{V} := \big\{ \boldsymbol{v} \in H^1(\Omega)^3 \;\big|\; \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_{\mathrm{D}} \big\}, \tag{3.53}$$

the corresponding variational formulation reads:

Find $\boldsymbol{u} - \bar{\boldsymbol{u}} \in \boldsymbol{V}$ such that

$$k(\boldsymbol{u}, \boldsymbol{v}) = L(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V}, \tag{3.54}$$

where the right hand side $L(\boldsymbol{v})$ is defined by equation (3.38). The bilinear form $k(\cdot,\cdot)$ is $\boldsymbol{V}$-elliptic and continuous,

$$\exists\, \alpha, C > 0 : \; \alpha \|\boldsymbol{v}\|_1^2 \leqslant k(\boldsymbol{v}, \boldsymbol{v}) \leqslant C \|\boldsymbol{v}\|_1^2, \qquad \boldsymbol{v} \in \boldsymbol{V}, \tag{3.55}$$

and the variational problem (3.54) has a unique solution [32].

For (nearly) incompressible material we apply the same strategy as in the finite elasticity case: The part of the stress tensor that responds sensitively to volume changes is replaced by a pressure-like variable. With the definition

$$p := -\lambda \operatorname{div}(\boldsymbol{u})$$

the boundary value problem for (nearly) incompressible material is then given by

$$-2\mu\,\mathbf{div}(\boldsymbol{\varepsilon}) + \boldsymbol{\nabla} p = \boldsymbol{f} \qquad \text{in } \Omega \tag{3.56a}$$
$$-\operatorname{div}(\boldsymbol{u}) - \tfrac{1}{\lambda} p = 0 \qquad \text{in } \Omega \tag{3.56b}$$
$$\boldsymbol{u} = \bar{\boldsymbol{u}} \qquad \text{on } \Gamma_{\mathrm{D}} \tag{3.56c}$$
$$\boldsymbol{\sigma n} = \boldsymbol{g} \qquad \text{on } \Gamma_{\mathrm{N}}. \tag{3.56d}$$

With $\boldsymbol{V}$ defined by (3.53), $M := L^2(\Omega)$ and the three bilinear forms

$$a(\boldsymbol{u},\boldsymbol{v}) := 2\mu\big(\boldsymbol{\varepsilon}(\boldsymbol{u}),\boldsymbol{\varepsilon}(\boldsymbol{v})\big)_\Omega, \tag{3.57a}$$
$$b(\boldsymbol{v},p) := -(\operatorname{div}(\boldsymbol{v}),p)_\Omega, \tag{3.57b}$$
$$c(p,q) := -(p,q)_\Omega, \tag{3.57c}$$

the corresponding variational formulation reads:

Find $(\boldsymbol{u}-\bar{\boldsymbol{u}},p) \in \boldsymbol{V}\times M$ such that

$$a(\boldsymbol{u},\boldsymbol{v}) + b(\boldsymbol{v},p) \;= L(\boldsymbol{v}), \qquad \boldsymbol{v}\in\boldsymbol{V} \tag{3.58a}$$
$$b(\boldsymbol{u},q) + \tfrac{1}{\lambda} c(p,q) = 0, \qquad q \in M. \tag{3.58b}$$

Again, for incompressible material the terms containing the constant λ vanish from equations (3.56b) and (3.58b). This representation reveals the structure of a *saddle point problem with a penalty term*, where the penalty term is given by $\frac{1}{\lambda}$, tending to zero as the degree of incompressibility increases. For the existence of a unique solution the $\boldsymbol{V}$-ellipticity of the bilinear form $a(\cdot,\cdot)$ is not sufficient. Additionally, the two spaces $\boldsymbol{V}$ and M have to fulfil a compatibility condition. Together, these two requirements form the well-known **Babuška-Brezzi conditions** [32]:

1. The continuous bilinear form $a(\cdot,\cdot)$ is $\boldsymbol{V}_0$-elliptic:

$$\exists \alpha > 0 : a(\boldsymbol{v},\boldsymbol{v}) \geqslant \alpha \|\boldsymbol{v}\|_1^2, \quad \boldsymbol{v}\in\boldsymbol{V}_0, \tag{3.59}$$

   where $\boldsymbol{V}_0 := \{\boldsymbol{v}\in\boldsymbol{V} \mid b(\boldsymbol{v},q) = 0, \quad q\in M\}$.

2. The continuous bilinear form $b(\cdot,\cdot)$ fulfils the inf-sup condition:

$$\exists \beta > 0 : \inf_{0\neq q\in M} \sup_{\mathbf{0}\neq\boldsymbol{v}\in\boldsymbol{V}} \frac{b(\boldsymbol{v},q)}{\|\boldsymbol{v}\|_1 \|q\|_0} \geqslant \beta. \tag{3.60}$$

Here, $\|\cdot\|_0$ and $\|\cdot\|_1$ are the usual norms induced by the scalar products of the spaces $L^2(\Omega)$ and $H^1(\Omega)^3$, respectively. The bilinear forms $a(\cdot,\cdot)$ and $b(\cdot,\cdot)$ and the spaces $\boldsymbol{V}$ and M fulfill these conditions. Additionally, $c(\cdot,\cdot)$ is continuous and $c(q,q) \leqslant 0, q \in M$. So, problem (3.58) has a unique solution [32].

For the sake of completeness, we present the transient linearised elasticity equations, as well. For the compressible case we have the initial boundary value problem

$$\rho\ddot{\boldsymbol{u}} - 2\mu\,\mathbf{div}(\boldsymbol{\varepsilon}) - \lambda\boldsymbol{\nabla}\operatorname{div}(\boldsymbol{u}) = \boldsymbol{f} \qquad \text{in } I\times\Omega \tag{3.61a}$$

$$\boldsymbol{u}_0 = \bar{\boldsymbol{u}}_0, \dot{\boldsymbol{u}}_0 = \bar{\boldsymbol{v}}_0 \qquad \text{in } \Omega \tag{3.61b}$$

$$\boldsymbol{u} = \bar{\boldsymbol{u}}, \qquad \text{in } I\times\Gamma_{\mathrm{D}} \tag{3.61c}$$

$$\boldsymbol{T}\boldsymbol{n} = \boldsymbol{g}, \qquad \text{in } I\times\Gamma_{\mathrm{N}} \tag{3.61d}$$

and the according variational formulation:

Find $\boldsymbol{u} - \bar{\boldsymbol{u}} \in L^2(I,\boldsymbol{V})$ such that

$$\int_0^T \Big[\rho(\ddot{\boldsymbol{u}},\boldsymbol{v})_\Omega + 2\mu\big(\boldsymbol{\varepsilon}(\boldsymbol{u}),\boldsymbol{\varepsilon}(\boldsymbol{v})\big)_\Omega + \lambda\big(\operatorname{div}(\boldsymbol{u}),\operatorname{div}(\boldsymbol{v})\big)_\Omega\Big]\,\mathrm{d}t = \int_0^T L(\boldsymbol{v})\,\mathrm{d}t, \qquad \boldsymbol{v}\in L^2(I,\boldsymbol{V}). \tag{3.62}$$

For the (nearly) incompressible case we have the initial boundary value problem

$$\rho\ddot{\boldsymbol{u}} - 2\mu\,\mathbf{div}(\boldsymbol{\varepsilon}) + \boldsymbol{\nabla}p = \boldsymbol{f} \qquad \text{in } I\times\Omega \tag{3.63a}$$

$$-\operatorname{div}(\boldsymbol{u}) - \tfrac{1}{\lambda}p = 0 \qquad \text{in } I\times\Omega \tag{3.63b}$$

$$\boldsymbol{u}_0 = \bar{\boldsymbol{u}}_0, \dot{\boldsymbol{u}}_0 = \bar{\boldsymbol{v}}_0 \qquad \text{in } \Omega \tag{3.63c}$$

$$\boldsymbol{u} = \bar{\boldsymbol{u}}, \qquad \text{in } I\times\Gamma_{\mathrm{D}} \tag{3.63d}$$

$$\boldsymbol{T}\boldsymbol{n} = \boldsymbol{g}, \qquad \text{in } I\times\Gamma_{\mathrm{N}} \tag{3.63e}$$

and the according variational formulation:

Find $(\boldsymbol{u} - \bar{\boldsymbol{u}}, p) \in L^2(I,\boldsymbol{V}) \times L^2(I,M)$ such that

$$\int_0^T \big[\rho(\ddot{\boldsymbol{u}},\boldsymbol{v})_\Omega + 2\mu\big(\boldsymbol{\varepsilon}(\boldsymbol{u}),\boldsymbol{\varepsilon}(\boldsymbol{v})\big)_\Omega - (\operatorname{div}(\boldsymbol{v}),p)_\Omega\big]\,\mathrm{d}t = \int_0^T L(\boldsymbol{v})\,\mathrm{d}t, \qquad \boldsymbol{v}\in L^2(I,\boldsymbol{V}) \tag{3.64a}$$

$$\int_0^T \big[-(\operatorname{div}(\boldsymbol{u}),q)_\Omega - \tfrac{1}{\lambda}(p,q)_\Omega\big]\,\mathrm{d}t = 0, \qquad q\in L^2(I,M). \tag{3.64b}$$

3.1.7 Restriction to Two-Dimensional Elasticity

The elasticity problem naturally applies to the three-dimensional physical world. However, consider a solid body which is long in X_3-direction and whose geometry does not vary along

this direction. Additionally, assume the applied forces to be independent of X_3 and to have zero X_3-components, i. e.,

$$\boldsymbol{f}(\boldsymbol{X}) = \big(f_1(\boldsymbol{X}), f_2(\boldsymbol{X}), f_3(\boldsymbol{X})\big) = \big(\tilde{f}_1(X_1, X_2), \tilde{f}_2(X_1, X_2), 0\big).$$

Then it is justified to only consider a cross section in the (X_1, X_2)-plane and to assume the displacements and strains in the X_3-direction to be zero. This describes the so called *plane strain* model, which can be considered as the actual 2D analogon to the three-dimensional case in the following sense: All the physical quantities (force vectors, strain tensors, stress tensors etc.) introduced in this chapter can be simply restricted to their 2D-versions by removing all components that are connected to the X_3-component. All relations and computations remain valid.[22]

The remainder of this work is confined to the two-dimensional plane strain model. One motivation for this is to simplify notation at some places, thus keeping the presentation clear and avoiding unnecessary complications. The second motivation is that also our numerical examples will be two-dimensional. The reason for this is that the 3D-version of FEAST as underlying library for FEASTsolid is work in progress and not finished yet. *However, all the concepts, equations and algorithms described in the following sections can be extended to the three-dimensional case without any complications.*

3.2 Discretisation in Space

In order to solve the elasticity problem numerically, the *continuous* equations derived in the previous sections have to be transformed into *discrete* representations. In this section we address the spatial discretisation, while the temporal discretisation is treated in Section 3.4.

The most popular spatial discretisation method, especially in CSM, is the finite element method (FEM). For a general introduction, with emphasis on solid mechanics, we refer to basic textbooks [15, 32, 176, 178, 179] and the references therein. Starting from a variational formulation (see Sections 3.1.5 and 3.1.6), the finite element method consists, roughly speaking, of two ingredients – the partition of the continuous body $\bar{\Omega}$ into 'elements' and the restriction of the variational solution spaces $\boldsymbol{V}$ and M to finite dimensional subspaces.

The domain discretisation has been described in Section 2.3. For the following, we neglect the multilevel representation of the domain, as well as its division into subdomains, and only

[22] This is different for the two-dimensional *plane stress* model which applies to structures that are thin in X_3-direction and only loaded in the (X_1, X_2) plane (e. g., a vertically loaded thin wall). Here, the stresses in X_3-direction can be neglected and are thus assumed to be zero. This assumption, however, changes the stress-strain relation such that, for instance, Hooke's law (3.50) for the two-dimensional plane stress case reads:

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda\big(1 - \tfrac{\lambda}{2\mu+\lambda}\big)\,\mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}.$$

consider the finest-level grid $\Omega^{(L)}$ (see definition (2.2) on page 27) and the corresponding triangulation $\mathcal{T}^{(L)}$ (see definition (2.3)). We will from now on omit the superscript denoting the grid level and use instead the superscript h which refers to the characteristic size of the quadrilateral elements e that build the triangulation. In summary, we have

$$\bar{\Omega} \approx \bar{\Omega}^h = \bigcup_{e \in \mathcal{T}^h} e,$$

where the intersection of two elements $e_i, e_j, i \neq j$, is either empty, a vertex, or an edge (also see footnote 3 on page 27.) In the following sections we introduce the space discretisations in terms of linearised elasticity and give additional remarks about the nonlinear case.

3.2.1 Displacement Formulation

The standard procedure in Galerkin finite element methods is to use polynomial basis functions with local support to build the discrete function spaces. Throughout this work, we use the conforming, bilinear finite element, denoted as Q_1. This particular choice has some severe disadvantages as we will show in the following, but nevertheless it is very popular in practical applications. The main reason for this is that such low-order elements are very convenient from an implementational point of view. The data structures can be kept simple, especially as the four nodes corresponding to the four nodal basis functions per element coincide with the geometric grid vertices, which, for example, facilitates boundary condition structures and communication structures in a parallel program environment (data exchange over inner subdomain boundaries). Additionally, the size of the resulting algebraic systems is significantly smaller than for higher-order elements (especially in 3D), and the matrices usually have a smaller bandwidth (which is especially important when direct solving techniques are applied). Of course, for higher-order elements a much coarser grid representation might suffice to achieve the same accuracy, such that at the end the total number of unknowns might be equal (or even smaller). This argument, however, is weakened by two points: First, complicated geometries often require a minimum grid resolution which would then lead to unnecessarily large algebraic systems when using higher-order elements. Second, the high accuracy of higher-order elements is often lost in regions where the solution is irregular, e. g., near the boundary or near a singularity. Another implementational issue is the necessary adaption of certain algorithmic components to the element type. Here, we want to exemplarily mention grid transfer operations in a multigrid solver (see Köster and Turek [101]) and FEAST's SparseBandedBLAS package. As described in Section 2.1.1, the latter is highly specialised to exploit the matrix band structure resulting from the mesh's tensor product property. Such specialisation often comes with a certain loss of flexibility, such that an extension to higher-order elements means reimplementation of significant parts of the library. Similar to FEAST's ongoing extension to 3D, this is again just a 'technical software issue', nevertheless it is extremely time consuming. Currently, this is work in progress, and the SparseBandedBLAS library does not support higher-order elements yet.

Let

$$Q_1(e^{\mathrm{b}}) := \left\{ q \;\middle|\; q(\xi,\eta) = \sum_{0 \leqslant i,j \leqslant 1} c_{ij} \xi^i \eta^j \right\}$$

be the space of bilinear polynomials over the basis element[23] $e^{\mathrm{b}} = [-1,1]^2$. Following the isoparametric concept, each element e of the triangulation $\mathcal{T}^h$ can be represented as a bilinear transformation of the basis element, i. e.,

$$e = \boldsymbol{\psi}_e(e^{\mathrm{b}}), \qquad \boldsymbol{\psi}_e \in Q_1(e^{\mathrm{b}})^2. \tag{3.65}$$

Using the definition

$$Q_1(e) := \{ q \circ \boldsymbol{\psi}_e^{-1} \mid q \in Q_1(e^{\mathrm{b}}) \},$$

we can describe the finite dimensional subspace $\boldsymbol{V}^h \subset \boldsymbol{V}$ formed by Q_1 finite elements as

$$\boldsymbol{V}^h := \left\{ \boldsymbol{v} \in C(\Omega)^2 \;\middle|\; \boldsymbol{v}|_e \in Q_1(e)^2 \; \forall e \in \mathcal{T}^h, \, \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_{\mathrm{D}} \right\}. \tag{3.66}$$

The dimension of this space is $\dim(\boldsymbol{V}^h) = 2n$ where n is the number of vertices in the discrete mesh $\bar{\Omega}^h$ (see Section 2.3). Denoting the n grid vertices with $\boldsymbol{x}_i, i = 1, \ldots, n$, then the basis of $\boldsymbol{V}^h$ is given by the nodal basis functions

$$\phi_i : \bar{\Omega}^h \to \mathbb{R}, \qquad \phi_i(\boldsymbol{x}_j) = \delta_{ij}, \tag{3.67}$$

which are defined element-wise by functions from $Q_1(e)$ (sometimes called 'shape functions'). Here, δ_{ij} is the Kronecker symbol. The discrete version of the variational problem (3.54) now simply reads:

Find $\boldsymbol{u}^h - \bar{\boldsymbol{u}} \in \boldsymbol{V}^h$ such that

$$k(\boldsymbol{u}^h, \boldsymbol{v}) = L(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V}^h. \tag{3.68}$$

The finite element solution $\boldsymbol{u}^h = \left(u_1^h, u_2^h\right) \in \boldsymbol{V}^h$ is given in terms of the nodal basis functions,

$$u_j^h(\boldsymbol{x}) = \sum_{i=1}^{n} (\mathbf{u}_j)_i \phi_i(\boldsymbol{x}), \qquad j = 1,2, \tag{3.69}$$

with $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2)^{\mathsf{T}} \in \mathbb{R}^{2n}$ being the (unknown) coefficient vector. Correspondingly, the bilinear form $k(\cdot,\cdot)$ leads to the discrete stiffness matrix $\mathbf{K} \in \mathbb{R}^{2n \times 2n}$ and the linear form $L(\cdot)$ to the discrete load vector $\mathbf{f} \in \mathbb{R}^{2n}$, resulting in the linear system of equations

$$\mathbf{K}\mathbf{u} = \mathbf{f}. \tag{3.70}$$

For details about the structure of this system and about the solution process see Chapter 4. Throughout this work we use bold upright letters to denote discrete vectors and matrices.

[23] We do not use the term 'reference element' here in order to avoid confusion with 'elements in the reference configuration' which have to be distinguished from those in the current configuration in the context of finite deformation.

The discrete variational problem (3.68) naturally raises the question about the error between the finite element solution $\boldsymbol{u}^h$ and the exact solution $\boldsymbol{u}$. The *Céa-Lemma* relates this error to the approximation error in terms of the H^1-norm:

$$\|\boldsymbol{u}-\boldsymbol{u}^h\|_1 \leqslant \frac{C}{\alpha} \inf_{\boldsymbol{v}\in\boldsymbol{V}^h} \|\boldsymbol{u}-\boldsymbol{v}\|_1. \tag{3.71}$$

The constants $\alpha > 0$ and $C > 0$ are defined by the $\boldsymbol{V}$-ellipticity and continuity of the bilinear form $k(\cdot,\cdot)$ (see equation (3.55)). The Céa-Lemma confirms the intuitive idea that 'enlarging' the discrete function space leads to a better approximation of the exact solution. This improvement can also be quantified. Under certain conditions regarding the domain Ω, the triangulation $\mathcal{T}^h$ and the regularity of the solution $\boldsymbol{u}$, the following error estimates can be derived [32]:

$$\begin{aligned} \|\boldsymbol{u}-\boldsymbol{u}^h\|_1 &\leqslant ch\|\boldsymbol{u}\|_2, \\ \|\boldsymbol{u}-\boldsymbol{u}^h\|_0 &\leqslant ch^2\|\boldsymbol{u}\|_2. \end{aligned} \tag{3.72}$$

Here, c is a generic positive constant, containing the factor $\frac{C}{\alpha}$ from the Céa-Lemma (3.71) which plays a role in the next sections. The estimates (3.72) are, for example, no longer guaranteed, when Ω is not convex (e. g., the slit domain in Figure 2.1 on page 20).

3.2.2 Shear Locking and Volume Locking

Although the error estimates (3.72) guarantee convergence towards the exact solution as the mesh is refined ($h \to 0$), standard polynomial finite element approximations might nevertheless show unexpectedly large errors in certain situations. This is especially true for low-order Q_1 elements. Generally speaking, the deficiency is caused by some parameter approaching zero or infinity which leads to a corresponding increase of the error constants in relations (3.72); the problem does not converge uniformly with respect to this degenerating parameter. Consequently, on coarser grids the constants dominate, and excessive grid refinement is necessary for compensation. Usually, finite element computations show too small displacements in such situations which coined the term 'locking'. Mathematical explanations of locking effects can be found in Babuška and Suri [9] and Braess [32]; mechanically motivated interpretations are presented, for example, by Koschnick [99]. We now present two kinds of locking effects, *shear* and *volume* locking. Both significantly influence the condition of the resulting linear systems and therefore play an important role with respect to our solving strategies (see Chapters 4 and 5). Furthermore, it turns out that the volume locking effect can only be treated robustly if a more advanced finite element discretisation is applied (see Section 3.2.3).

3.2.2.1 Shear Locking

The classical example for demonstrating the shear locking effect is a *cantilever beam*. A rectangular body with length L and height H, where $L \gg H$, is fixed on one of its narrow ends

and vertically loaded by some body or surface force. We use a special configuration here which stems from a fluid structure interaction benchmark [80, 159]: The beam is attached to a cylinder positioned in a channel (i. e., the fixed end is rounded; see Figure 3.4). A fluid moving through this channel imposes traction forces on the beam's surface, resulting in elastic deformations. The benchmark aims at global aspect ratios of more than $\frac{L}{H} = 1000$.

The difficulties arising with such configurations can be measured mathematically in terms of *Korn's inequality*. For simplicity, we assume zero Dirichlet boundary conditions on Γ_D, and Γ_D having a positive one-dimensional measure (which is true for the beam configuration). Korn's inequality then amounts to

$$\exists c_K = c_K(\Omega, \Gamma_D) > 0 : \|\boldsymbol{\varepsilon}(\boldsymbol{v})\|_0^2 \geqslant c_K \|\boldsymbol{v}\|_1^2, \qquad \boldsymbol{v} \in \boldsymbol{V}. \tag{3.73}$$

This relation is used to prove the ellipticity of the bilinear form $k(\cdot,\cdot)$ (see equation (3.55) on page 133), i. e., the ellipticity constant is simply given by $\alpha = 2\mu c_K$. The dependence of the Korn constant c_K on the domain Ω and on the boundary Γ_D now plays the decisive role for the explanation of the shear locking effect. Imagine the beam is fixed on its narrow left end and loaded in such a way that it undergoes the deformation

$$\boldsymbol{v} = \begin{pmatrix} x_1 x_2 \\ -\frac{1}{2} x_1^2 \end{pmatrix}, \qquad \boldsymbol{\varepsilon}(\boldsymbol{v}) = \begin{pmatrix} x_2 & 0 \\ 0 & 0 \end{pmatrix}, \tag{3.74}$$

which describes, due to $\boldsymbol{\varepsilon}_{12} = 0$, a state of *pure bending*. Using this deformation to estimate

$$\frac{1}{c_K} = \sup_{\boldsymbol{v} \in \boldsymbol{V}, \boldsymbol{\varepsilon}(\boldsymbol{v}) \neq \boldsymbol{0}} \frac{\|\boldsymbol{v}\|_1}{\|\boldsymbol{\varepsilon}(\boldsymbol{v})\|_0},$$

it is easy to see that

$$\frac{L}{H} \to \infty \quad \Rightarrow \quad \frac{1}{c_K} \to \infty,$$

i. e., the inverse of the Korn constant grows with the global aspect ratio of the beam. The quality of the finite element approximation can be affected correspondingly since the inverse of the Korn constant enters the Céa-Lemma (3.71) and the error estimates (3.72). In numerical computations the effect arises particularly when a relatively small number of vertices is used in horizontal direction. To illustrate this we consider three versions of the beam with dimensions $H = 0.02$ and $L \in \{0.04, 0.16, 0.64\}$, i. e., global anisotropies of $\frac{L}{H} = 2, 8$ and 32, where each coarse grid consists of four elements (see Figure 3.1). As analytic solution we use the defor-

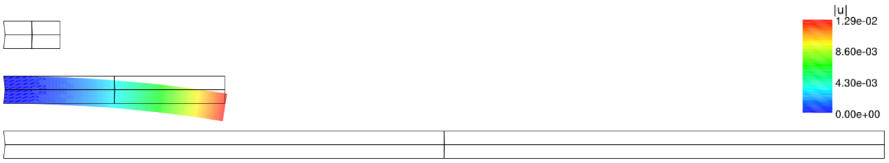
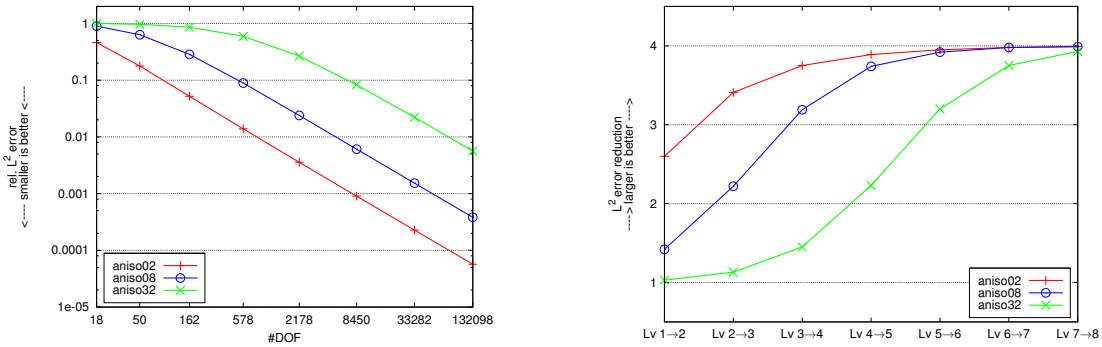


Figure 3.1: Three beam configurations BEAM-LH[2,8,32]-ANISO. Deformed state of the second beam, $\|\boldsymbol{u}\|$ displayed. Prescribed analytical solution $\boldsymbol{v} = (x_1 x_2, -\frac{1}{2} x_1^2)^T$, material steel.

mation $\boldsymbol{v}$ as given in example (3.74). The right hand side terms (body and surface forces) are computed according to the prescribed solution such that we are able to compute the L^2-error $\|\boldsymbol{u}-\boldsymbol{u}^h\|_0$. Figure 3.1 exemplarily displays the resulting deformation of the middle beam. The plots in Figure 3.2 show that for the longer beams the expected L^2 error reduction of 4 is only attained on higher refinement levels. This confirms that on coarser grids the error constant dominates.



(a) Relative L^2 error over number of DOF.

(b) Reduction of the L^2 error.

Figure 3.2: Results of the tests with analytical solutions for the three BEAM configurations.

The shear locking problem is particularly severe for the Q_1 element, which is not well suited to perform such bending dominated deformations. This is often illustrated by the following simple example. The two opposite sides of the basis element $e^{\mathrm{b}} = [-1,1]^2$ are 'tilted' as shown in Figure 3.3. When denoting the displacements of the four corners with $\big((\mathbf{u}_1)_i,(\mathbf{u}_2)_i\big), i =$

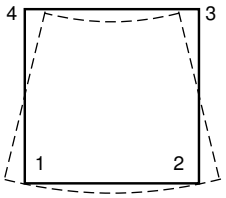


Figure 3.3: The basis element under pure bending.

$1,\ldots,4$ (at the same time representing the coefficients of the finite element function $\boldsymbol{u}^h$, cf. equation (3.69)), this corresponds to

$$\begin{aligned}(\mathbf{u}_1)_1 &= -(\mathbf{u}_1)_2 = (\mathbf{u}_1)_3 = -(\mathbf{u}_1)_4,\\ (\mathbf{u}_2)_1 &= (\mathbf{u}_2)_2 = -(\mathbf{u}_2)_3 = -(\mathbf{u}_2)_4.\end{aligned} \tag{3.75}$$

Such a pure bending situation is characterised by zero shear strains, i. e.,

$$\boldsymbol{\varepsilon}_{12}(\boldsymbol{u}^h) = \frac{1}{2}\Big(\frac{\partial u_1^h}{\partial \eta} + \frac{\partial u_2^h}{\partial \xi}\Big) = 0. \tag{3.76}$$

Defining the four bilinear basis functions

$$\phi_1(\xi,\eta) = \tfrac{1}{4}(1-\xi)(1-\eta), \qquad \phi_2(\xi,\eta) = \tfrac{1}{4}(1+\xi)(1-\eta),$$
$$\phi_3(\xi,\eta) = \tfrac{1}{4}(1+\xi)(1+\eta), \qquad \phi_4(\xi,\eta) = \tfrac{1}{4}(1-\xi)(1+\eta),$$

and considering relations (3.75), the finite element function is given by

$$u_1^h(\xi,\eta) = \sum_{i=1}^{4} (\mathbf{u}_1)_i \phi_i(\xi,\eta) = (\mathbf{u}_1)_1 \xi\eta, \tag{3.77}$$

$$u_2^h(\xi,\eta) = \sum_{i=1}^{4} (\mathbf{u}_2)_i \phi_i(\xi,\eta) = -(\mathbf{u}_2)_1 \eta, \tag{3.78}$$

yielding

$$\boldsymbol{\varepsilon}_{12}(\boldsymbol{u}^h) = \tfrac{1}{2}(\mathbf{u}_1)_1 \xi.$$

Condition (3.76) thus leads to $(\mathbf{u}_1)_1 = 0$, implying that in a single element only zero displacements can fulfil the pure bending condition (3.76), or – to put it differently – non-zero displacements 'produce non-physical shear strains', which has to be compensated for by increasing the number of grid cells. Higher-order elements like Q_2 do not exhibit this deficiency as they have, roughly speaking, 'enough degrees of freedom' to resolve such bending deformations. Unfortunately, FEAST does not support higher-order elements, so a direct comparison is not possible here. Instead, we exemplarily illustrate the effect by applying two different loading states to the BEAM-LH2-ISO configuration (see Figure 3.4) with material parameters $\mu = 8.14 \cdot 10^4$ MPa and $\nu = 0.29$ (steel/iron). The first loading state is a horizontal traction force stretching the beam (see Figures 3.4a and 3.4c), the second one is a 'rotational' force causing a bending moment (see Figures 3.4b and 3.4d). For these tests the analytical solutions are not known, so we take the finite element solutions obtained on the level 11 grid (16.8 M DOF) as reference solution, denoted as $\boldsymbol{u}^*$. Figure 3.5 shows the relative error (in percent) of the displacement's magnitude in the upper right corner $\boldsymbol{A} = (0.04, 0.01)$ of the beam, i. e.,

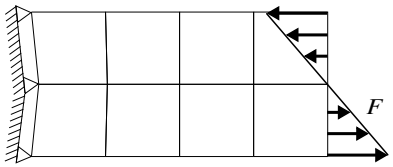
$$e_{\text{rel}} := 100 \frac{\|\boldsymbol{u}^h(\boldsymbol{A}) - \boldsymbol{u}^*(\boldsymbol{A})\|}{\|\boldsymbol{u}^*(\boldsymbol{A})\|}.$$

The left plot shows that on coarser meshes the approximation of the bending situation is extremely poor, and the right plot reveals that about three extra grid refinements are necessary to achieve a similar accuracy as in the stretching case.[24] We want to emphasise that this test only serves as illustration to hint at Q_1's deficiencies in bending dominated configurations. A further example is presented in Section 4.3.2 for the case of finite deformation elasticity (see Figure 4.23 on page 236). For more elaborate examinations and comparisons with Q_2, we refer to Brink and Stein [41], Koschnick [99] and Papoulia [124].

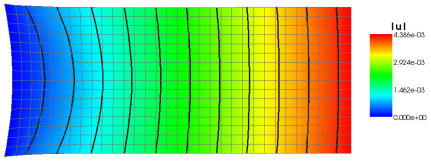
In summary, whenever long, thin structures and/or bending-like deformations occur, the Q_1 element might produce less accurate solutions than expected, eventually demanding a finer

[24] The latter statement does only make sense if we assume that the reference solutions are 'similarly close' to the real solutions.

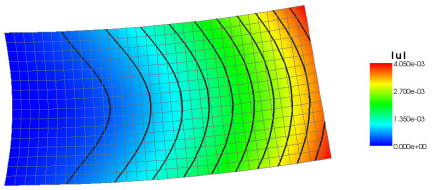
(a) Stretched beam: coarse grid, reference point $\boldsymbol{A} = (0.04, 0.01)$ and load $F = 2.5 \cdot 10^4$ MPa.



(b) Bended beam: coarse grid and load $F = -10^6 y$ MPa.

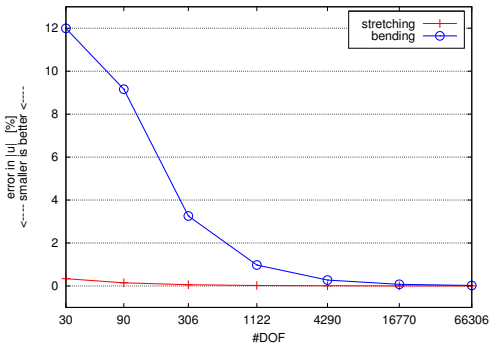


(c) Stretched beam: deformed state (Level 4), $\|\boldsymbol{u}\|$ displayed.

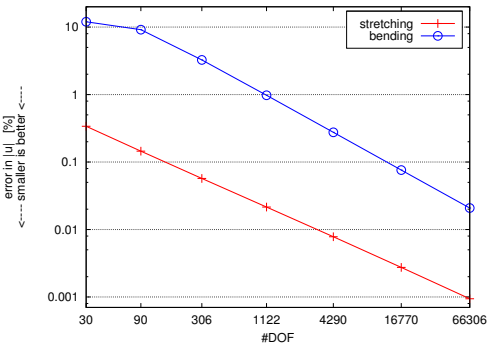


(d) Bended beam: deformed state (Level 4), $\|\boldsymbol{u}\|$ displayed.

Figure 3.4: Two different loading states of the BEAM-LH2-ISO configuration.



(a) Normal y-scale.



(b) Logarithmic y-scale.

Figure 3.5: Beam stretching vs. beam bending: Relative error of the displacements in the upper right corner $\boldsymbol{A} = (0.04, 0.01)$ over varying number of degrees of freedom (level 1–7).

grid resolution. Since the poor behaviour is triggered by geometric factors (shape of the body, boundary conditions, bending-like deformation), the effect is also referred to as *geometric locking*.

3.2.2.2 Volume Locking

In contrast to shear locking, the *volume locking* phenomenon, which has already been introduced in Section 3.1.3 on page 125, emanates from a local property of the material, and is therefore also denoted as *material locking*. It occurs when standard polynomial finite elements

are used to compute deformations of nearly incompressible material, which leads to an over-constrained problem setting. The degenerating parameter is the Lamé constant λ tending to infinity as the degree of incompressibility increases. From equation (3.55) follows $\alpha \lesssim \mu$ and $C \gtrsim \mu + \lambda$ [32], i. e.,

$$\lambda \to \infty \quad \Rightarrow \quad \frac{C}{\alpha} \gtrsim \frac{\mu+\lambda}{\mu} \to \infty.$$

This means that the constants in the Céa-Lemma (3.71) and in the error estimates (3.72) increase with λ, and the finite element solution does not converge uniformly with respect to this parameter. Especially for coarser grid resolutions large errors can be expected, as the following illustrating example shows. We consider the stretched beam configuration (see Figure 3.4a), a traction force of $F = 5.0$ MPa and rubber materials with Poisson ratios between $\nu = 0.4$ and $\nu = 0.49999$ and shear modulus $\mu = 8.2$ MPa. Again, we evaluate the relative displacement error (in percent) in the upper right corner of the beam, where we use the level 11 solution obtained with a locking-free formulation (see Section 3.2.3) as reference. The plots in Figure 3.6

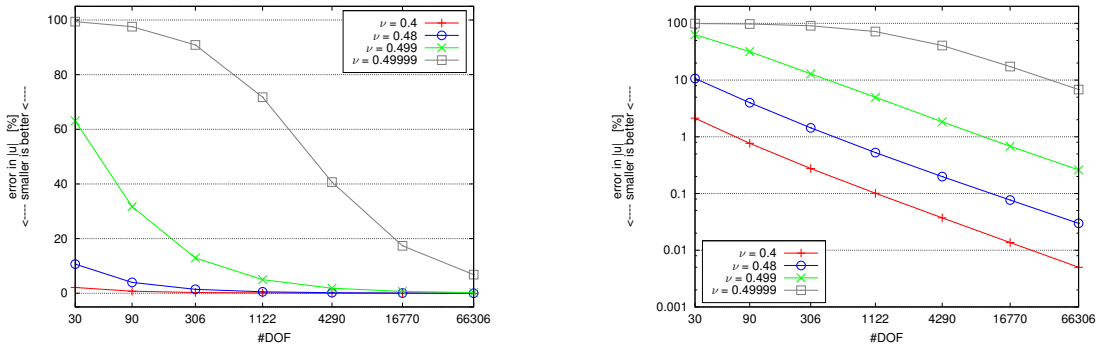


Figure 3.6: Volume locking: Relative error of the displacements in the upper right corner $\boldsymbol{A} = (0.04, 0.01)$ of the stretched beam (Figure 3.4a) over varying number of degrees of freedom (level 1–7) for different degrees of compressibility. Left normal y-scale, right logarithmic y-scale.

clearly show how the approximation quality of the displacement formulation deteriorates with increasing degree of incompressibility. The less compressible the material is, the worse are especially the coarse grid approximations and the slower is the convergence towards the exact solution. In the case of the most incompressible material ($\nu = 0.49999$) the relative error on the level 7 grid (66.3 k DOF) is still about 7%. In contrast to shear locking, the usage of higher-order elements like Q_2 does not significantly alleviate the volume locking problem (see, e. g., Brink and Stein [41]).

In Chapters 4 and 5 we show that standard linear solvers severely suffer from the described locking situations. It will turn out that the impact of shear locking on iterative solver performance is clearly observable, though, but that it can be significantly weakened by using robust solution methods (see Section 4.2.7.3). The effect of volume locking, however, is much more severe, and it is very difficult to solve the resulting linear systems at all (see Section 5.2.1.7). When additionally considering the need for excessively fine meshes to achieve reasonable small

discretisation errors, this renders the standard Q_1 displacement formulation useless for the treatment of nearly incompressible material.

3.2.3 Mixed Displacement-Pressure Formulation

Many techniques have been developed to counteract locking effects. In view of the shear locking problem, however, we consider it justified to still use the standard Q_1 displacement discretisation, even if 'non-standard' versions might perform better [107, 146, 154]. In contrast, it is *mandatory* to switch to an advanced finite element technique in order to overcome the volume locking problem. Some popular approaches in this regard are mixed displacement-pressure formulations (see below), nonconforming methods, especially *Enhanced Assumed Strain (EAS)* methods [146], and *selective reduced integration (SRI)* techniques [107]. EAS and SRI can be shown to be equivalent to certain mixed methods [31, 107], but have the advantage that the displacements remain the only unknowns in the formulation and that the resulting linear systems remain positive definite. This is different for mixed formulations where additional unknowns are introduced, leading to eventually indefinite saddle point systems. However, for both EAS and SRI special care has to be taken to avoid instabilities of the volumetric terms ('checkerboard modes'), also known from the Q_1/P_0 mixed element [31, 107]. A countermeasure is, for example, to consider *macro elements* consisting of four elements [31, 32]. Such techniques, however, are (currently) prohibitive in the FEAST context since they introduce additional coupling and thus change the band structure of the resulting discrete matrices (cf. Section 2.1.1). Furthermore, the linear systems arising from EAS and SRI are not suited for standard multigrid methods [33, 140]. To avoid degeneration of convergence rates, special transfer and smoothing operations are necessary, which is clearly unfavourable in view of the desired 'black box' solver (see Chapter 1). Additionally, standard EAS methods may suffer from nonphysical instabilities ('hourglassing'), especially in finite deformation computations, requiring special non-trivial stabilisation methods [132, 133, 176].

These considerations are one motivation for our decision to use the mathematical well-established *mixed displacement-pressure formulation* to handle nearly incompressible material. Another crucial motivation is that highly developed discretisation and solution techniques from the field of computational fluid dynamics (CFD) can be employed. This is possible since the discrete systems arising from the (Navier-)Stokes equations for incompressible fluids exhibit a similar structure as the systems stemming from the mixed formulation of (nearly) incompressible elasticity. Especially the fact, that for the solution only 'standard' multigrid components are needed (in contrast to the necessary modifications for displacement formulations; see previous paragraph), is very advantageous for the development of 'black box' solvers. Using the mixed displacement-pressure formulation enables us to treat incompressible material ($\nu = 0.5$), for which pure displacement formulations degenerate. However, the mixed formulation also comes along with some disadvantages. Due to the additional pressure variable the linear systems to be solved are larger, and their saddle point structure requires special solving techniques (see Section 5.1), such that the computational work compared to one of the robust pure displacement

formulations described above is clearly higher. Another disadvantage is the restrictive choice of finite element spaces for displacements and pressure, especially the need for stabilisation in the case of equal-order elements (see Section 3.3).

Many authors have considered the (stabilised) mixed displacement-pressure formulation in the context of CSM. For example, Sussmann and Bathe [152] develop models for finite elasticity based on a mixed displacement-pressure formulation. Also in the framework of finite elasticity, Brink and Stein [41] compare different stable mixed methods, which is pursued by Klaas et al. [95] for the stabilised Q_1/Q_1 pair and by Maniatty et al. [108] for stable and stabilised higher-order pairs. In the same context, Papoulia [124] draws comparisons to SRI methods. Suttmeier [153], Cervera et al. [49] and Commend et al. [54] apply stabilised mixed displacement-pressure methods to elastoplastic materials under small deformations. Ramesh and Maniatty [131] and Agelet de Saracibar et al. [1] extend this to finite deformation elastoplasticity, while Pastor et al. [125] consider applications of elastic and viscoplastic failure. In the framework of linearised elasticity, Klawonn [96, 97] develops block-preconditioners for the saddle point systems arising from the mixed formulation, Braess and Blömer [33] compare multigrid performance for mixed and SRI methods, and Wieners [173] analyses a multigrid solver applied to the pure displacement formulation with a smoother based on the mixed formulation. Pantuso and Bathe [122] and Auricchio et al. [5] combine the mixed displacement-pressure formulation with the EAS method and systematically study this approach for the case of finite elasticity [4, 123]. Chiumenti, Valverde, Agelet de Saracibar, and Cervera apply the mixed formulation with orthogonal sub-scale stabilisation to linearised elasticity [51], elastoplasticity [49] and finite deformation elastoplasticity [1].

The discrete form of the variational mixed displacement-pressure problem (cf. equation (3.58) on page 134) reads:

Find $(\boldsymbol{u}^h - \bar{\boldsymbol{u}}, p^h) \in \boldsymbol{V}^h \times M^h$ such that

$$a(\boldsymbol{u}^h, \boldsymbol{v}) + b(\boldsymbol{v}, p^h) = L(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V}^h \tag{3.79a}$$

$$b(\boldsymbol{u}^h, q) + \tfrac{1}{\lambda} c(p^h, q) = 0, \qquad q \in M^h. \tag{3.79b}$$

Corresponding to the displacement formulation (3.70), the discretisation process finally leads to a linear system of equations

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{C}^{\mathrm{c}} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}. \tag{3.80}$$

The matrix $\mathbf{C}^{\mathrm{c}}$ contains the 'compressibility terms'[25], i. e., we have $\mathbf{C}^{\mathrm{c}} = 0$ in the incompressible limit, revealing the saddle point structure of the system.

In order to obtain a robust formulation, the choice of the finite element spaces $\boldsymbol{V}^h$ and M^h is crucial. This choice is guided by the observation, that problem (3.56) on page 134 bears

[25] The superscript 'c' is added to distinguish this matrix from the stabilisation matrix $\mathbf{C}^{\mathrm{s}}$ which is introduced in Section 3.3.

resemblance to the *Stokes equation for viscous incompressible fluids*: When considering the limit case $\nu = 0.5$, replacing the strain tensor $\boldsymbol{\varepsilon}(\boldsymbol{u})$ by the gradient $\boldsymbol{\nabla u}$ and using the relation $\boldsymbol{\Delta u} = \mathbf{div}(\boldsymbol{\nabla u})$, this results in

$$-2\mu\boldsymbol{\Delta u} + \boldsymbol{\nabla} p = \boldsymbol{f} \qquad \text{in } \Omega \tag{3.81a}$$

$$\operatorname{div}(\boldsymbol{u}) = 0 \qquad \text{in } \Omega. \tag{3.81b}$$

When interpreting Ω as fluid, $\boldsymbol{u}$ as the fluid's velocity, p as pressure and 2μ as dynamic viscosity, this is the classical form of the stationary Stokes equation. Due to this equivalence, element pairs which have been found to be efficient for the Stokes equation, are also suitable for the discretisation of the mixed displacement-pressure formulation [32].

If the finite element spaces $\boldsymbol{V}^h$ and M^h fulfil a discrete analogon of the Babuška-Brezzi conditions (3.59) and (3.60) (see equation (3.99) on page 152), then the finite element solution converges robustly, i. e., independently of the parameter λ, to the exact solution [32]. Such element combinations are called *stable*. An example is the Q_k/P_{k-1} pair for $k \geqslant 2$, where especially the case $k = 2$ is very popular in CSM [15, 41, 80, 105, 152]. For other stable elements see, e. g., Braess [32].

FEAST's current restriction concerning the finite element library limits our choice of element combinations to Q_1/P_0 and Q_1/Q_1. The first is ineligible due to the considerations at the beginning of this section, such that only Q_1/Q_1 remains. Hence, we choose the space $\boldsymbol{V}^h$ as in equation (3.66) on page 138, and the space M^h likewise:

$$M^h := \left\{ q \in C(\Omega) \;\middle|\; q|_e \in Q_1(e)\ \forall e \in \mathcal{T}^h \right\}. \tag{3.82}$$

It is well known, that equal-order combinations Q_k/Q_k, $k \geqslant 1$, are unstable [32], such that some sort of stabilisation procedure is necessary, which will be introduced in Section 3.3.[26]

3.2.4 Finite Deformation

The discrete variational formulations of the boundary value problems of compressible and (nearly) incompressible finite deformation (see equations (3.39) and (3.43) on page 129) are obtained analogously to the linearised case, choosing the same discrete spaces $\boldsymbol{V}^h$ and M^h (see equation (3.66) on page 138 and equation (3.82)). For theoretical convergence results see, e. g., Le Tallec [105] and Braess and Ming [34].

3.2.4.1 Displacement Formulation

First, we consider the nonlinear displacement formulation for compressible material. Replacing in equation (3.40) on page 129 the function space $\boldsymbol{V}$ by the discrete one, $\boldsymbol{V}^h$, the discrete variational formulation reads:

[26] With 'stabilisation' we always refer to the Babuška-Brezzi stabilisation. Other kinds of stabilisation as they are used in CFD (e. g., to handle convective terms) are not treated in this thesis.

Find $\boldsymbol{u}^h - \bar{\boldsymbol{u}} \in \boldsymbol{V}^h$ such that

$$\big(\boldsymbol{T}(\boldsymbol{u}^h), \mathbf{Grad}(\boldsymbol{v})\big)_\Omega = L(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V}^h, \tag{3.83}$$

where the first Piola-Kirchhoff stress tensor $\boldsymbol{T}(\boldsymbol{u}^h)$ is given by relation (3.41) or (3.42) on page 129. The argument $\boldsymbol{u}^h$ emphasises the tensor's nonlinear dependence on the solution. Defining the functional

$$\mathcal{R}_{\boldsymbol{u}^h}(\boldsymbol{v}) := \big(\boldsymbol{T}(\boldsymbol{u}^h), \mathbf{Grad}(\boldsymbol{v})\big)_\Omega - L(\boldsymbol{v}), \tag{3.84}$$

we can express problem (3.83) in residual form as:

$$\mathcal{R}_{\boldsymbol{u}^h}(\boldsymbol{v}) = 0, \qquad \boldsymbol{v} \in \boldsymbol{V}^h. \tag{3.85}$$

We take a closer look at the resulting algebraic system of nonlinear equations. To ease the notation, we split the linear form (3.38) on page 128,

$$L(\boldsymbol{v}) := L_1(v_1) + L_2(v_2),$$

where

$$L_j(v_j) := (f_j, v_j)_\Omega + (g_j, v_j)_{\Gamma_\mathrm{N}}, \qquad j = 1,2. \tag{3.86}$$

Then we define the discrete vectors

$$\mathbf{f}^{\mathrm{ex}} := (\mathbf{f}_1^{\mathrm{ex}}, \mathbf{f}_2^{\mathrm{ex}})^\mathsf{T}, \quad \mathbf{f}^{\mathrm{in}}(\mathbf{u}) := \big(\mathbf{f}_1^{\mathrm{in}}(\mathbf{u}), \mathbf{f}_2^{\mathrm{in}}(\mathbf{u})\big)^\mathsf{T} \quad \in \mathbb{R}^{2n}$$

of *external* and *internal forces*, respectively, with

$$\begin{aligned} (\mathbf{f}_j^{\mathrm{ex}})_i &:= L_j(\phi_i), && i = 1,\dots,n, \quad j = 1,2, \\ \big(\mathbf{f}_j^{\mathrm{in}}(\mathbf{u})\big)_i &:= \big(\boldsymbol{T}_{j\bullet}(\boldsymbol{u}^h), \mathbf{Grad}(\phi_i)\big)_\Omega, && i = 1,\dots,n, \quad j = 1,2. \end{aligned}$$

Here, $\boldsymbol{T}_{j\bullet}(\boldsymbol{u}^h)$ denotes the j-th row of the first Piola-Kirchhoff stress tensor $\boldsymbol{T}(\boldsymbol{u}^h)$, ϕ_i the nodal basis functions and $\mathbf{u}$ the unknown coefficient vector of the finite element solution $\boldsymbol{u}^h$ (cf. relations (3.67) and (3.69) on page 138). Then, we can define – corresponding to equation (3.84) – the residual vector

$$\mathbf{r}(\mathbf{u}) := \mathbf{f}^{\mathrm{in}}(\mathbf{u}) - \mathbf{f}^{\mathrm{ex}} \in \mathbb{R}^{2n}, \tag{3.87}$$

which measures the *imbalance of internal and external forces*. The nonlinear problem (3.85) can then be represented in compact, algebraic form as

$$\mathbf{r}(\mathbf{u}) = \mathbf{0}. \tag{3.88}$$

In order to solve this nonlinear problem, we need to consider its linearisation around the current solution state $\boldsymbol{u}^h$. Performing the linearisation leads to the linear variational problem:

Find $\hat{\boldsymbol{u}}^h \in \boldsymbol{V}^h$ such that

$$\tilde{k}(\hat{\boldsymbol{u}}^h, \boldsymbol{v}) = -\mathcal{R}_{\boldsymbol{u}^h}(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V}^h, \tag{3.89}$$

where the bilinear form $\tilde{k}(\cdot,\cdot)$ is defined by[27]

$$\tilde{k}(\hat{\boldsymbol{u}}^h, \boldsymbol{v}) := \frac{\partial \mathcal{R}_{\boldsymbol{u}^h}}{\partial \boldsymbol{u}} \hat{\boldsymbol{u}}^h. \tag{3.90}$$

Corresponding to the small deformation case (see equation (3.70) on page 138), the variational problem (3.89) can be represented as linear algebraic equation system

$$\tilde{\mathbf{K}}\hat{\mathbf{u}} = -\mathbf{r}(\mathbf{u}),$$

where $\hat{\mathbf{u}}$ is the unknown coefficient vector of the finite element function $\hat{\boldsymbol{u}}^h$, $\mathbf{r}(\mathbf{u})$ the residual vector defined by relation (3.87), and $\tilde{\mathbf{K}}\hat{\mathbf{u}}$ the algebraic representation of the bilinear form defined in equation (3.90). Note, that the matrix $\tilde{\mathbf{K}}$ depends on the current solution state $\mathbf{u}$, i. e., $\tilde{\mathbf{K}} = \tilde{\mathbf{K}}(\mathbf{u})$ and is called *tangent matrix*, or simply *tangent*.

3.2.4.2 Mixed Formulation

Now, we turn to the nonlinear mixed formulation for (nearly) incompressible material. Replacing in equation (3.44) the function spaces $\boldsymbol{V}$ and M by discrete ones, $\boldsymbol{V}^h$ and M^h, the discrete weak mixed formulation reads:

Find $(\boldsymbol{u}^h - \bar{\boldsymbol{u}}, p^h) \in \boldsymbol{V}^h \times M^h$ such that

$$\big(\boldsymbol{T}(\boldsymbol{u}^h, p^h), \mathbf{Grad}(\boldsymbol{v})\big)_\Omega = L(\boldsymbol{v}), \qquad \boldsymbol{v} \in \boldsymbol{V}^h \tag{3.91a}$$

$$-\big(\tfrac{1}{2}\big(J(\boldsymbol{u}^h) - \tfrac{1}{J(\boldsymbol{u}^h)}\big), q\big)_\Omega - \tfrac{1}{\lambda}(p^h, q)_\Omega = 0, \qquad q \in M^h, \tag{3.91b}$$

where $\boldsymbol{T}(\boldsymbol{u}^h, p^h)$ is defined in equation (3.45) on page 130. Corresponding to the displacement formulation, we define the functionals

$$\mathcal{R}_{\boldsymbol{u}^h, p^h}(\boldsymbol{v}) := \big(\boldsymbol{T}(\boldsymbol{u}^h, p^h), \mathbf{Grad}(\boldsymbol{v})\big)_\Omega - L(\boldsymbol{v}) \tag{3.92a}$$

$$\mathcal{S}_{\boldsymbol{u}^h, p^h}(q) := -\big(\tfrac{1}{2}\big(J(\boldsymbol{u}^h) - \tfrac{1}{J(\boldsymbol{u}^h)}\big), q\big)_\Omega - \tfrac{1}{\lambda}(p^h, q)_\Omega, \tag{3.92b}$$

and express problem (3.91) in residual form as:

$$\mathcal{R}_{\boldsymbol{u}^h, p^h}(\boldsymbol{v}) = 0, \qquad \boldsymbol{v} \in \boldsymbol{V}^h, \tag{3.93a}$$

$$\mathcal{S}_{\boldsymbol{u}^h, p^h}(q) = 0, \qquad q \in M^h. \tag{3.93b}$$

The resulting nonlinear algebraic equation system can be derived analogously to the displacement formulation. Let the discrete vectors

$$\mathbf{f}^{\text{ex}} = \big(\mathbf{f}_1^{\text{ex}}, \mathbf{f}_2^{\text{ex}}, \mathbf{f}_3^{\text{ex}}\big)^{\mathsf{T}}, \quad \mathbf{f}^{\text{in}}(\mathbf{u}, \mathbf{p}) = \big(\mathbf{f}_1^{\text{in}}(\mathbf{u}, \mathbf{p}), \mathbf{f}_2^{\text{in}}(\mathbf{u}, \mathbf{p}), \mathbf{f}_3^{\text{in}}(\mathbf{u}, \mathbf{p})\big)^{\mathsf{T}} \quad \in \mathbb{R}^{3n}$$

[27] We use the tilde notation $\tilde{k}(\cdot,\cdot)$ to distinguish the bilinear form from its small deformation counterpart (see equation (3.52) on page 133).

of external and internal forces be given by

$$\begin{aligned}
(\mathbf{f}_j^{\text{ex}})_i &:= L_j(\phi_i), && i=1,\dots,n, \quad j=1,2,\\
\mathbf{f}_3^{\text{ex}} &:= \mathbf{0},\\
\big(\mathbf{f}_j^{\text{in}}(\mathbf{u},\mathbf{p})\big)_i &:= \big(\boldsymbol{T}_{j\bullet}(\boldsymbol{u}^h,p^h),\mathbf{Grad}(\phi_i)\big)_\Omega, && i=1,\dots,n, \quad j=1,2,\\
\big(\mathbf{f}_3^{\text{in}}(\mathbf{u},\mathbf{p})\big)_i &:= -\big(\tfrac{1}{2}\big(J(\boldsymbol{u}^h)-\tfrac{1}{J(\boldsymbol{u}^h)}\big),\phi_i\big)_\Omega - \tfrac{1}{\lambda}(p^h,\phi_i)_\Omega, && i=1,\dots,n.
\end{aligned}$$

Then we define – corresponding to relation (3.92) – the residual vector

$$\mathbf{r}(\mathbf{u},\mathbf{p}) := \mathbf{f}^{\text{in}}(\mathbf{u},\mathbf{p}) - \mathbf{f}^{\text{ex}} \in \mathbb{R}^{3n}, \tag{3.94}$$

such that problem (3.93) can be represented in the compact, algebraic form

$$\mathbf{r}(\mathbf{u},\mathbf{p}) = \mathbf{0}.$$

The linearisation of the nonlinear equation (3.93) leads to following saddle point problem:

Find $(\hat{\boldsymbol{u}}^h,\hat{p}^h)\in \boldsymbol{V}^h\times M^h$ such that

$$\tilde{a}(\hat{\boldsymbol{u}}^h,\boldsymbol{v})+\tilde{b}(\boldsymbol{v},\hat{p}^h) = -\mathcal{R}_{\boldsymbol{u}^h,p^h}(\boldsymbol{v}), \qquad \boldsymbol{v}\in\boldsymbol{V}^h \tag{3.95a}$$

$$\tilde{d}(\hat{\boldsymbol{u}}^h,q)+\tilde{c}(\hat{p}^h,q) = -\mathcal{S}_{\boldsymbol{u}^h,p^h}(q), \qquad q\in M^h. \tag{3.95b}$$

The bilinear forms are given by

$$\begin{aligned}
\tilde{a}(\hat{\boldsymbol{u}}^h,\boldsymbol{v}) &:= \frac{\partial \mathcal{R}_{\boldsymbol{u}^h,p^h}}{\partial \boldsymbol{u}}\hat{\boldsymbol{u}}^h, \quad \tilde{b}(\boldsymbol{v},\hat{p}^h) := \frac{\partial \mathcal{R}_{\boldsymbol{u}^h,p^h}}{\partial p}\hat{p}^h,\\
\tilde{d}(\hat{\boldsymbol{u}}^h,q) &:= \frac{\partial \mathcal{S}_{\boldsymbol{u}^h,p^h}}{\partial \boldsymbol{u}}\hat{\boldsymbol{u}}^h, \quad \tilde{c}(\hat{p}^h,q) := \frac{\partial \mathcal{S}_{\boldsymbol{u}^h,p^h}}{\partial p}\hat{p}^h.
\end{aligned} \tag{3.96}$$

Finally, problem (3.95) can be written as algebraic linear equation system

$$\begin{pmatrix}\tilde{\mathbf{A}} & \tilde{\mathbf{B}}\\ \tilde{\mathbf{D}} & \tilde{\mathbf{C}}^{\text{c}}\end{pmatrix}\begin{pmatrix}\hat{\mathbf{u}}\\ \hat{\mathbf{p}}\end{pmatrix} = -\mathbf{r}(\mathbf{u},\mathbf{p}), \tag{3.97}$$

whose block matrix depends on the current solution state $(\mathbf{u},\mathbf{p})^{\mathsf{T}}$. In contrast to the small deformation case (see equation (3.80) on page 146), this system is not necessarily symmetric since, in general, $\tilde{\mathbf{D}}\neq\tilde{\mathbf{B}}^{\mathsf{T}}$ holds.

3.2.4.3 Computation of the Tangent Matrix

The calculations that are needed to build the gradients (3.90) and (3.96) are elementary, though, but their implementation is technical and tedious, especially for the fourth order tensor, that

results from differentiating the first Piola-Kirchhoff stress tensor.[28] Additionally, implementations have to be redone, whenever a new constitutive law or a new stabilisation method (see Section 3.3) is added to the code. This motivated us to compute the tangent matrix not analytically, but numerically via finite differences. We only present the mixed formulation, the displacement formulation is treated analogously. For theoretical considerations in this regard see Kelley [92].

Let $\mathbf{e}_j$ denote the vector of generic length with the j-th component equal to 1 and the other components equal to 0. Furthermore, the value ε_j describes the step size of the finite difference scheme and is given by

$$\varepsilon_j := \gamma \max\{1, |\mathbf{u}_j|\},$$

where $\gamma > 0$ is a perturbation parameter. Then, the single matrix blocks of the saddle point system (3.97) can be approximated as follows:

$$\begin{aligned}
\tilde{\mathbf{A}}_{ij} &\approx \frac{\mathbf{r}_i(\mathbf{u}+\varepsilon_j\mathbf{e}_j,\mathbf{p}) - \mathbf{r}_i(\mathbf{u}-\varepsilon_j\mathbf{e}_j,\mathbf{p})}{2\varepsilon_j}, && i,j = 1,\dots,2n,\\
\tilde{\mathbf{B}}_{ij} &\approx \frac{\mathbf{r}_i(\mathbf{u},\mathbf{p}+\varepsilon_j\mathbf{e}_j) - \mathbf{r}_i(\mathbf{u},\mathbf{p}-\varepsilon_j\mathbf{e}_j)}{2\varepsilon_j}, && i = 1,\dots,2n, \quad j = 1,\dots,n,\\
\tilde{\mathbf{D}}_{ij} &\approx \frac{\mathbf{r}_{i+2n}(\mathbf{u}+\varepsilon_j\mathbf{e}_j,\mathbf{p}) - \mathbf{r}_{i+2n}(\mathbf{u}-\varepsilon_j\mathbf{e}_j,\mathbf{p})}{2\varepsilon_j}, && i = 1,\dots,n, \quad j = 1,\dots,2n,\\
\tilde{\mathbf{C}}^{\mathrm{c}}_{ij} &\approx \frac{\mathbf{r}_{i+2n}(\mathbf{u},\mathbf{p}+\varepsilon_j\mathbf{e}_j) - \mathbf{r}_{i+2n}(\mathbf{u},\mathbf{p}-\varepsilon_j\mathbf{e}_j)}{2\varepsilon_j}, && i = 1,\dots,n, \quad j = 1,\dots,n.
\end{aligned} \tag{3.98}$$

It is worth noting that due to the sparse matrix structure resulting from the finite element discretisation, the assembly process can be realised in $O(n)$ time and does, consequently, not slow down the overall solving process. In Section 4.3.4 we examine how to choose the perturbation parameter γ.

3.3 Stabilisation

Equal-order element pairs, especially the low-order Q_1/Q_1 pair, are very popular from an implementational point of view since data structures can be kept simple: Only one set of basis functions has to be managed and the resulting system matrix only consists of square submatrices which all exhibit the same band structure. Element pairs like Q_k/P_{k-1}, for example, lead to 'rectangular' submatrices with different band patterns. Especially in view of such highly specialised linear algebra libraries like FEAST's SparseBandedBLAS (see Section 2.1.1), the use of equal-order pairs reduces the implementational effort tremendously. Furthermore, if a finite element software package, which is not yet able to treat mixed problems, is planned to be extended in this respect, this is relatively simple when – at least in a first stage – equal-order

[28] For a detailed representation of the linearised tensors, see, e. g., Brink and Stein [41], Klaas et al. [95], Le Tallec [105] and Wriggers [176].

pairs can be employed, since many parts of the code can be reused in this case (matrix assembly, solver components, etc.). Another advantage of stabilised equal-order pairs is their efficiency, i. e., the ratio between computational costs and accuracy. Norburn and Silvester [117], for example, show that the stabilised (triangular) P_1/P_1 element is more than competitive with its stable counterparts in this regard. It can be assumed that this is analogously true for the stabilised (quadrilateral) Q_1/Q_1 element, for which to our knowledge a corresponding study is not available, though.

The widespread popularity and application of the equal-order Q_1/Q_1 pair justifies a detailed examination in the context of CSM. Our aim is to discover the specific disadvantages accompanying this approach in terms of the finite element discretisation (see this section) and in terms of iterative solution methods (see Chapter 5). We try to evaluate how severe these disadvantages are and give some hints for possible improvements. This study can be considered as preparation for later comparisons to stable element pairs like Q_2/P_1 on the one hand, and to EAS or SRI methods (see Section 3.2.3) on the other hand. Such comparisons first require the mentioned extensions of the FEAST library and are not part of this thesis.

The lack of stability of the equal-order ansatz can be illustrated with the help of the discrete version of the inf-sup condition (3.60) on page 134:

$$\exists \beta > 0 : \inf_{0 \neq q^h \in M^h} \sup_{\mathbf{0} \neq \boldsymbol{v}^h \in \boldsymbol{V}^h} \frac{b(\boldsymbol{v}^h, q^h)}{\|\boldsymbol{v}^h\|_1 \|q^h\|_0} \geqslant \beta. \tag{3.99}$$

This can also be read as: For each $q^h \in M^h$ there must be a $\boldsymbol{v}^h \in \boldsymbol{V}^h$ such that $b(\boldsymbol{v}^h, q^h) \geqslant \beta \|\boldsymbol{v}^h\|_1 \|q^h\|_0$. In the case of the incompressible limit, the term $b(\boldsymbol{v}^h, p^h)$ is the only one in which the pressure is tested, namely by functions from $\boldsymbol{V}^h$ (see equation (3.79a) on page 146). If condition (3.99) is not fulfilled, there might be a $p^h \neq 0$ with $b(\boldsymbol{v}, p^h) = 0$, i. e., a non-vanishing function that does not contribute to the equation. In other words: For a given displacement space $\boldsymbol{V}^h$, the corresponding pressure space M^h must not be 'too large', otherwise there might be pressure modes which cannot be tested by functions from $\boldsymbol{V}^h$ and, hence, 'remain undetected', eventually leading to nonphysical oscillations of the pressure field, the pressure tending to infinity or even prohibiting the solution of the problem at all.

For more than twenty years now, great efforts have been made to stabilise low-order finite element pairs, pushed by CFD researchers seeking efficient solution methods for the incompressible Navier-Stokes equations. Two of the first contributions in this regard are those of Brezzi and Pitkäranta [40] and Hughes et al. [82]. The vast number of publications forbids a concise survey of the different approaches within this thesis, so we exemplarily refer to the overviews in Barrenechea and Blasco [12], Barth et al. [14], Bochev et al. [27], Braack et al. [29], Wall [171] and the references therein. Here, we examine two approaches, one going back to the classical Galerkin/Least-Squares scheme [68], and a more recent pressure projection method [27].

3.3.1 A Galerkin/Least-Squares Approach

One possibility to account for the disproportion between the two discretisation spaces $\boldsymbol{V}^h$ and M^h described in the previous section is to regularise the discrete equations and thus coarsening the pressure approximation: The continuity equation is equipped with an additional bilinear form testing the pressure field, which – together with the tests provided by the form $b(\boldsymbol{v}^h, p^h)$ – helps to cancel the spurious pressure modes. In *Galerkin/Least-Squares (GLS)* methods these additional terms are least-squares forms of the momentum equation's residual. When the exact solution is inserted, these additional terms vanish, rendering the method *consistent*. While the term *Galerkin/Least-Squares* was introduced by Hughes et al. [83], the approach is also denoted as *residual-based stabilisation* or *Pressure-Stabilising/Petrov-Galerkin (PSPG)*[29]. We now introduce the method of Franca and Stenberg [68], which extends the method of Hughes and Franca [81] to the case of (nearly) incompressible linearised elasticity.

The starting point is the discrete weak formulation (3.79) on page 146, which can equivalently be written as:

Find $(\boldsymbol{u}^h - \bar{\boldsymbol{u}}, p^h) \in \boldsymbol{V}^h \times M^h$ such that

$$Q(\boldsymbol{u}^h, p^h; \boldsymbol{v}, q) = L(\boldsymbol{v}), \qquad (\boldsymbol{v}, q) \in \boldsymbol{V}^h \times M^h, \tag{3.100}$$

where $Q(\cdot,\cdot;\cdot,\cdot)$ is a bilinear form on the product space $\boldsymbol{V}^h \times M^h$ defined by

$$Q(\boldsymbol{u}^h, p^h; \boldsymbol{v}, q) := a(\boldsymbol{u}^h, \boldsymbol{v}) + b(\boldsymbol{v}, p^h) + b(\boldsymbol{u}^h, q) + \tfrac{1}{\lambda} c(p^h, q).$$

The forms $a(\cdot,\cdot)$, $b(\cdot,\cdot)$ and $c(\cdot,\cdot)$ are defined in relation (3.57) on page 134. The variational problem (3.100) is now extended in the following manner:

Find $(\boldsymbol{u}^h - \bar{\boldsymbol{u}}, p^h) \in \boldsymbol{V}^h \times M^h$ such that

$$Q(\boldsymbol{u}^h, p^h; \boldsymbol{v}, q) + c_{\mathsf{GLS}}(\boldsymbol{u}^h, p^h; \boldsymbol{v}, q) = L(\boldsymbol{v}) + L_{\mathsf{GLS}}(\boldsymbol{v}, q), \qquad (\boldsymbol{v}, q) \in \boldsymbol{V}^h \times M^h, \tag{3.101}$$

where the GLS stabilisation terms are given by

$$c_{\mathsf{GLS}}(\boldsymbol{u}^h, p^h; \boldsymbol{v}, q) := -\frac{\alpha}{2\mu} \sum_e h_e^2 \Big(-2\mu \operatorname{\mathbf{div}}\big(\boldsymbol{\varepsilon}(\boldsymbol{u}^h)\big) + \boldsymbol{\nabla} p^h, -2\mu \operatorname{\mathbf{div}}\big(\boldsymbol{\varepsilon}(\boldsymbol{v})\big) + \boldsymbol{\nabla} q \Big)_e, \tag{3.102a}$$

$$L_{\mathsf{GLS}}(\boldsymbol{v}, q) := -\frac{\alpha}{2\mu} \sum_e h_e^2 \Big(\boldsymbol{f}, -2\mu \operatorname{\mathbf{div}}\big(\boldsymbol{\varepsilon}(\boldsymbol{v})\big) + \boldsymbol{\nabla} q \Big)_e. \tag{3.102b}$$

Here, α is a stabilisation parameter, h_e describes the characteristic size of element e (see Section 3.3.2), and $(\cdot,\cdot)_e$ denotes the L^2 inner product over the element e, i. e.,

$$(a,b)_e := \int_e a(\boldsymbol{x}) b(\boldsymbol{x}) \,\mathrm{d}x, \qquad a, b \in L^2(e, \mathbb{R}),$$

$$(\boldsymbol{a}, \boldsymbol{b})_e := \int_e \boldsymbol{a}(\boldsymbol{x}) \cdot \boldsymbol{b}(\boldsymbol{x}) \,\mathrm{d}x, \qquad \boldsymbol{a}, \boldsymbol{b} \in L^2(e, \mathbb{R}^2).$$

[29] In analogy to *Streamline-Upwind/Petrov-Galerkin (SUPG)* methods used in CFD.

The variational system (3.101) allows for the following interpretation: Considering the boundary value problem (3.56) on page 134, one can derive the weak formulation (3.58) by multiplying the momentum equation (3.56a) with test functions $\boldsymbol{v} \in \boldsymbol{V}^h$ and integrating over the domain Ω. The variational formulation (3.101) including the GLS terms, however, can formally be obtained by using 'perturbed' test functions of the form

$$\tilde{\boldsymbol{v}} := \boldsymbol{v} - \frac{\alpha}{2\mu} h_e^2 \Big(-2\mu \, \mathbf{div}\big(\boldsymbol{\varepsilon}(\boldsymbol{v})\big) + \boldsymbol{\nabla} q \Big),$$

which justifies the notation *Petrov-Galerkin method* (see also Hughes et al. [82]). Considering the GLS terms (3.102), one can easily see that the stabilised weak problem (3.101) actually results from adding the residual of the momentum equation (3.56a), i. e.,

$$-2\mu \, \mathbf{div}\big(\boldsymbol{\varepsilon}(\boldsymbol{u}^h)\big) + \boldsymbol{\nabla} p^h - \boldsymbol{f},$$

weighted by the above stabilisation terms. This explains the notation *residual-based stabilisation*. As a consequence, when inserting the exact solution $(\boldsymbol{u}, p)$ into the equation, the residual is zero and all stabilisation contributions vanish. In this sense, the method can be considered as being *consistent*. Finally, the term *Galerkin/Least-Squares* results from interpreting the sum (3.102a) as the first variation of the least-squares term

$$-\frac{\alpha}{2\mu} \sum_e \frac{1}{2} h_e^2 \big\| 2\mu \, \mathbf{div}\big(\boldsymbol{\varepsilon}(\boldsymbol{u}^h)\big) + \boldsymbol{\nabla} p^h - \boldsymbol{f} \big\|_e^2,$$

which represents the penalty term of the penalised Lagrangian functional whose optimality system is the variational equation (3.101) (see Barth et al. [14] for details).

Franca and Stenberg prove stability and optimal convergence of the method [68, Theorem 3.1]. For the Q_1/Q_1 case, they show that under the usual regularity assumptions on the exact solution $(\boldsymbol{u}, p)$ and for $0 < \alpha < C_{\mathrm{I}}$ problem (3.101) has a unique solution $(\boldsymbol{u}^h, p^h)$ satisfying

$$\|\boldsymbol{u} - \boldsymbol{u}^h\|_1 + \|p - p^h\|_0 \leqslant C(h\|\boldsymbol{u}\|_2 + h^2\|p\|_2), \tag{3.103a}$$

$$\|\boldsymbol{u} - \boldsymbol{u}^h\|_0 \leqslant C(h^2\|\boldsymbol{u}\|_2 + h^3\|p\|_2). \tag{3.103b}$$

Here, C is a generic constant independent of h and λ, and the upper bound C_{I} for the stabilisation parameter α follows from an inverse inequality which is needed to prove the stability of the method (for details, see Franca and Stenberg [68]). This upper bound depends on the subdivision of the mesh and renders the method *conditionally stable* (see below and Barth et al. [14]).

A simplified version of the presented GLS scheme is that of Hughes et al. [82], which we denote – following the nomenclature of Barth et al. [14] – as *simplified Galerkin/Least-Squares*

(SGLS). The method can be defined by replacing the stabilisation terms $c_{\mathsf{GLS}}, L_{\mathsf{GLS}}$ in equation (3.101) by[30]

$$c_{\mathsf{SGLS}}(\boldsymbol{u}^h, p^h; q) := -\frac{\alpha}{2\mu}\sum_e h_e^2\Big(-2\mu\,\mathbf{div}\big(\boldsymbol{\varepsilon}(\boldsymbol{u}^h)\big) + \boldsymbol{\nabla} p^h, \boldsymbol{\nabla} q\Big)_e, \tag{3.104a}$$

$$L_{\mathsf{SGLS}}(q) := -\frac{\alpha}{2\mu}\sum_e h_e^2(\boldsymbol{f}, \boldsymbol{\nabla} q)_e\,. \tag{3.104b}$$

The approach is also known as *pressure-Poisson stabilised Galerkin method* or simply as *pressure-Poisson stabilisation*. In Brezzi and Douglas [38], the method is analysed and similar error estimates as (3.103) are derived, but using a mesh-dependent norm for the pressure. Barth et al. [14] observe that SGLS, although considered as conditionally stable, actually behaves like an absolutely stable method: Numerical studies show that there seems to be no upper bound for the stabilisation parameter, while the choice of this parameter is indeed critical in the case of the GLS method of Franca and Stenberg [68]. In Bochev and Gunzburger [25], an enhanced version of the SGLS method is developed which turns out to fulfil the above error estimates in the natural (mesh-independent) norm and is absolutely stable (i. e., there is no upper bound for the stabilisation parameter). The question, whether the original SGLS method is absolutely stable, is still unanswered.

The presented stabilisation approaches (3.102) and (3.104) contain second derivative terms of the form

$$\mathbf{div}\big(\boldsymbol{\varepsilon}(\boldsymbol{u}^h)\big). \tag{3.105}$$

If $\boldsymbol{V}^h$ consists of bilinear functions, then these terms cannot be represented correctly. Furthermore, in standard Q_1 finite element implementations second derivatives are not calculated at all, such that the stabilisation terms (3.105) can consequently be omitted right from the beginning. The resulting method does not belong to the class of residual-based stabilisation and is *not consistent*. In fact, it is a *penalty method* introducing a penalty error of $\|\mathrm{div}(\boldsymbol{u})\|_0 = O(h)$ (see Brezzi and Douglas [38]). The method, which we want to call *penalty pressure-Poisson (PPP)*, is given by replacing the stabilisation terms $c_{\mathsf{GLS}}, L_{\mathsf{GLS}}$ in equation (3.101) by

$$c_{\mathsf{PPP}}(p^h, q) := -\frac{\alpha}{2\mu}\sum_e h_e^2(\boldsymbol{\nabla} p^h, \boldsymbol{\nabla} q)_e\,, \tag{3.106a}$$

$$L_{\mathsf{PPP}}(q) := -\beta\frac{\alpha}{2\mu}\sum_e h_e^2(\boldsymbol{f}, \boldsymbol{\nabla} q)_e\,. \tag{3.106b}$$

Since this method is not consistent anyway, it is questionable whether the addition of the right hand side terms (3.106b) is reasonable at all. The same question arises for the stabilisation methods (3.109), (3.112) and (3.115) introduced in the next section. Indeed, numerical experiments (which we do not present here) for all these stabilisation methods show: When the right hand side stabilisation terms are added, then the displacement L^2 error slightly deteriorates, the displacement H^1 error is nearly unaffected, and the pressure L^2 error sometimes improves and

[30] Actually, Hughes et al. [82] use a formulation in which the continuity equation and, consequently, the stabilisation term ∇q are multiplied by -1. For a discussion in this regard, see Barth et al. [14].

sometimes deteriorates (depending on the grid and the stabilisation method). However, these differences are only marginal and the convergence rate of the finite element solution towards the exact solution is qualitatively not affected. As a consequence, we consider the right hand side terms as negligible and set $\beta = 0$ for the remainder of this thesis.[31] Hence, method (3.106) coincides with the method of Brezzi and Pitkäranta [40] for the Stokes equation (see also Brezzi and Douglas [38]). As pointed out by Hughes and Franca [81], the upper bound on the stabilisation parameter α in the GLS method is only necessary to control the terms (3.105) on element interiors. Consequently, since these terms are not existent in the PPP method, the upper bound can be dropped. Actually, the penalty method introduced by Brezzi and Pitkäranta [40] works without any parameter. The analysis in Brezzi and Douglas [38] shows, that the method exhibits the same asymptotic error estimates as SGLS.

REMARK 3.6

The degradation of the consistent GLS and SGLS schemes to actually inconsistent penalty methods in the case of bilinear elements motivated some authors to make improvements in this regard. Droux and Hughes [59] add boundary integrals to the SGLS scheme to account for the vanishing divergence terms; a similar approach, which is only applicable in the 2D case, is pursued by Pierre [129] for the penalty method of Brezzi and Pitkäranta [40]. This strategy is justified since the additional term $(-\boldsymbol{\nabla} p + \boldsymbol{f}, \boldsymbol{\nabla} q)_\Omega$ can be interpreted as imposing the natural boundary condition $(\boldsymbol{\nabla} p - \boldsymbol{f}) \cdot \boldsymbol{n} = 0$ for the pressure [171]. Hence, when the divergence terms are neglected, the pressure tends to adopt to this non-physical boundary condition, which is shown in a numerical example in Section 3.3.5 (see Figure 3.18 on page 176). Jansen et al. [85] and Bochev and Gunzburger [25] describe methods to reconstruct the second order terms to retain consistency of the stabilisation approach in the case of low-order elements. This, however, requires the costly introduction of an extra variable and a modification of the standard solution process.

In this thesis we confine ourselves to the 'standard' method, i. e., for all our numerical tests and the following considerations we will use the PPP method. The implementation of one of the 'advanced' methods mentioned in Remark 3.6 is a topic for future work.

Finite element discretisation of the bilinear form $c_{\mathsf{PPP}}(\cdot,\cdot)$ leads to a corresponding stabilisation matrix $\mathbf{C}^{\mathrm{s}}$. Defining the matrix

$$\mathbf{C} := \mathbf{C}^{\mathrm{c}} + \mathbf{C}^{\mathrm{s}},$$

consisting of the compressibility and stabilisation contributions, the stabilised version of the linear saddle point problem (3.80) on page 146 is given by

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}. \tag{3.107}$$

In the incompressible limit we have $\mathbf{C} = \mathbf{C}^{\mathrm{s}}$.

[31] For the stabilisation methods (3.109), (3.112) and (3.115) introduced in the next section the notation of the right hand side stabilisation terms is omitted completely.

3.3.2 Modification for Unstructured Meshes

The PPP method makes use of the characteristic element size h_e. Possibilities to choose this value are for example

$$h_e^{\text{area}} := \sqrt{\text{area}(e)}, \tag{3.108a}$$

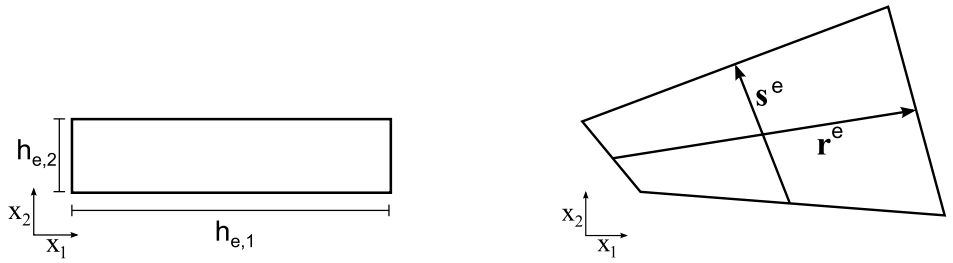
$$h_e^{\min} := \min_{i=1,\dots,4} \{h_e^i\}, \tag{3.108b}$$

$$h_e^{\max} := \max_{i=1,\dots,4} \{h_e^i\}, \tag{3.108c}$$

where $h_e^1, \dots, h_e^4$ are the lengths of the four element edges. For square-shaped elements the three values coincide, and on irregular, but isotropic elements with moderate aspect ratios (cf. definition (2.5) on page 28), they differ only slightly. The situation is different on highly anisotropic meshes, where the three definitions differ significantly, depending on the degree of anisotropy. To retain robustness of the stabilised formulation on anisotropic meshes, Becker and Rannacher [19] suggest a split of the stabilisation terms with respect to the spatial directions (see also Becker [17]). Their approach is restricted to Cartesian grids (i. e., element edges are aligned to the coordinate axes). With $h_{e,1}$ and $h_{e,2}$ denoting the element extension in x_1- and x_2-direction, respectively (see Figure 3.7a), the stabilisation terms are given by

$$c_{\text{PPPBR}}(p^h, q) := -\frac{\alpha}{2\mu} \sum_e h_{e,1}^2 (\partial_1 p^h, \partial_1 q)_e + h_{e,2}^2 (\partial_2 p^h, \partial_2 q)_e \,. \tag{3.109}$$

In the special case of square shaped elements, the scheme coincides with the PPP method (see



(a) Anisotropic element aligned to the coordinate axes. **(b)** Irregular element with local coordinate system.

Figure 3.7: Two element types.

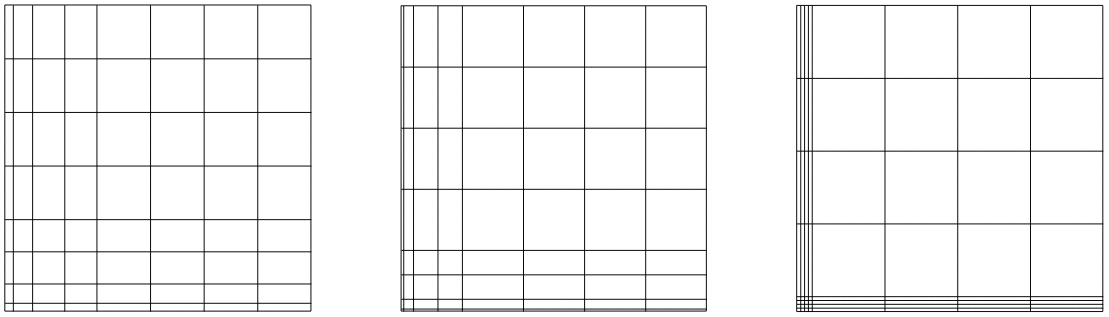
equation (3.106)). Becker [17] proves stability and optimal convergence of the approach on anisotropic meshes.

REMARK 3.7

Becker [17] emphasises the importance of a constant κ entering the estimates. It is given by

$$\kappa^{-1} h_{e,1} \leqslant h_{\hat{e},1} \leqslant \kappa h_{e,1} \quad \text{and} \quad \kappa^{-1} h_{e,2} \leqslant h_{\hat{e},2} \leqslant \kappa h_{e,2}, \quad \hat{e} \in \mathcal{N}(e),$$

where $\mathcal{N}(e)$ is the set of all neighbour elements of e. This constant describes how the aspect ratios of neighbouring elements vary. Consequently, in those areas of the mesh where these variations are large, the solution quality might suffer. We confirm this in the numerical examples in Section 3.3.5. In view of the objective to develop methods which are robust with respect to mesh anisotropies (see Chapter 1), this has to be seen as a severe drawback. We especially have to be careful when using the anisotropic refinement routine (2.4) on page 27: The more the factors $a_\mathrm{F}, a_\mathrm{I}, a_\mathrm{L}$ deviate from 1.0, the larger the critical variations become, which can be observed in the examples in Figure 3.8. Additionally, when two neighbouring cells of the initial macro decomposition (see Section 2.3 on page 27) exhibit strongly deviating aspect ratios, this directly transfers to the two element layers at the corresponding subdomain boundary on the finest grid. So, even when applying isotropic refinement, such undesirable effects can occur.



(a) $(a_\mathrm{F}, a_\mathrm{I}, a_\mathrm{L}) = (0.6, 0.6, 0.6)$. **(b)** $(a_\mathrm{F}, a_\mathrm{I}, a_\mathrm{L}) = (0.4, 0.4, 0.4)$. **(c)** $(a_\mathrm{F}, a_\mathrm{I}, a_\mathrm{L}) = (0.1, 1.0, 1.0)$.

Figure 3.8: Three examples of anisotropic refinement with increasing value κ (refinement level 3 shown).

The main drawback of the PPPBR method is that it is restricted to Cartesian meshes. To overcome this restriction we now introduce an extended version which is applicable to arbitrary meshes. Consider an irregular quadrilateral and its *local coordinate system* as shown in Figure 3.7b. Now, instead of using partial derivatives $\partial_1 p^h, \ldots$, we employ *directional derivatives* $\boldsymbol{r}^e \cdot \boldsymbol{\nabla} p^h, \ldots$, along the axes $\boldsymbol{r}^e = (r_1^e, r_2^e)$ and $\boldsymbol{s}^e = (s_1^e, s_2^e)$ of the element's local coordinate system. Precisely, $\boldsymbol{r}^e$ and $\boldsymbol{s}^e$ are the vectors connecting the midpoints of two opposing element edges, respectively (see Figure 3.7b). The resulting PPP stabilisation method using local coordinates (denoted as PPPLC) is then given by

$$c_{\mathsf{PPPLC}}(p^h, q) := -\frac{\alpha}{2\mu} \sum_e (\boldsymbol{r}^e \cdot \boldsymbol{\nabla} p^h, \boldsymbol{r}^e \cdot \boldsymbol{\nabla} q)_e + (\boldsymbol{s}^e \cdot \boldsymbol{\nabla} p^h, \boldsymbol{s}^e \cdot \boldsymbol{\nabla} q)_e \,. \tag{3.110}$$

Written in terms of the partial derivatives and the three constants

$$c_1 := (r_1^e)^2 + (s_1^e)^2, \quad c_2 := (r_2^e)^2 + (s_2^e)^2, \quad c_3 := r_1^e r_2^e + s_1^e s_2^e, \tag{3.111}$$

we obtain

$$c_{\mathsf{PPPLC}}(p^h,q) := -\frac{\alpha}{2\mu}\sum_e \quad c_1(\partial_1 p^h,\partial_1 q)_e + c_2(\partial_2 p^h,\partial_2 q)_e \\ + c_3\left[(\partial_1 p^h,\partial_2 q)_e + (\partial_2 p^h,\partial_1 q)_e\right]. \tag{3.112}$$

Our method can be regarded as 'natural extension' of the PPPBR method: On Cartesian meshes the two methods coincide, since $r_1^e = h_{e,1}, s_2^e = h_{e,2}, r_2^e = s_1^e = 0$ holds and the partial derivatives are nothing else but the directional derivatives along the coordinate axes $(1,0)$ and $(0,1)$. As a consequence, the drawback described in Remark 3.7 also applies to PPPLC.

Alternatively, the PPPLC method can be expressed with the help of the transformation $\boldsymbol{\psi}_e$ between the basis element $e^{\mathrm{b}} = [-1,1]^2$ and the element e (cf. equation (3.65) on page 138). The Jacobi matrix of this mapping, which we denote by $\boldsymbol{J}^e$, describes the mapping up to first order.[32] Consequently, when applying the Jacobi matrix evaluated in the element centre to the local coordinate axes $(2,0)$ and $(0,2)$ of the basis element e^{b}, the local coordinate axes of the element e result:

$$\begin{pmatrix} J_{11}^e(0,0) & J_{12}^e(0,0) \\ J_{21}^e(0,0) & J_{22}^e(0,0) \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} r_1^e \\ r_2^e \end{pmatrix}, \qquad \begin{pmatrix} J_{11}^e(0,0) & J_{12}^e(0,0) \\ J_{21}^e(0,0) & J_{22}^e(0,0) \end{pmatrix} \begin{pmatrix} 0 \\ 2 \end{pmatrix} = \begin{pmatrix} s_1^e \\ s_2^e \end{pmatrix}.$$

So, we have the relation

$$\begin{pmatrix} r_1^e & s_1^e \\ r_2^e & s_2^e \end{pmatrix} = 2\begin{pmatrix} J_{11}^e(0,0) & J_{12}^e(0,0) \\ J_{21}^e(0,0) & J_{22}^e(0,0) \end{pmatrix} = 2\boldsymbol{J}^e(0,0), \tag{3.113}$$

i. e., the local coordinate axes $\boldsymbol{r}^e$ and $\boldsymbol{s}^e$ of the element e build the columns of the (scaled) Jacobi matrix evaluated in the element centre. Reformulating the summands in definition (3.110),

$$\begin{aligned} &(\boldsymbol{r}^e\cdot\boldsymbol{\nabla}p^h, \boldsymbol{r}^e\cdot\boldsymbol{\nabla}q)_e + (\boldsymbol{s}^e\cdot\boldsymbol{\nabla}p^h, \boldsymbol{s}^e\cdot\boldsymbol{\nabla}q)_e \\ =&\left(\begin{pmatrix} \boldsymbol{r}^e\cdot\boldsymbol{\nabla}p^h \\ \boldsymbol{s}^e\cdot\boldsymbol{\nabla}p^h \end{pmatrix}, \begin{pmatrix} \boldsymbol{r}^e\cdot\boldsymbol{\nabla}q \\ \boldsymbol{s}^e\cdot\boldsymbol{\nabla}q \end{pmatrix}\right)_e = \left(\begin{pmatrix} r_1^e & r_2^e \\ s_1^e & s_2^e \end{pmatrix}\boldsymbol{\nabla}p^h, \begin{pmatrix} r_1^e & r_2^e \\ s_1^e & s_2^e \end{pmatrix}\boldsymbol{\nabla}q\right)_e, \end{aligned}$$

and using the definition $\boldsymbol{J}_0^e := \boldsymbol{J}^e(0,0)$, the PPPLC method can consequently be written as

$$c_{\mathsf{PPPLC}}(p^h,q) = -\frac{2\alpha}{\mu}\sum_e (\boldsymbol{J}_0^{e\mathsf{T}}\boldsymbol{\nabla}p^h, \boldsymbol{J}_0^{e\mathsf{T}}\boldsymbol{\nabla}q)_e = -\frac{2\alpha}{\mu}\sum_e (\boldsymbol{J}_0^e\boldsymbol{J}_0^{e\mathsf{T}}\boldsymbol{\nabla}p^h, \boldsymbol{\nabla}q)_e, \tag{3.114}$$

i. e., the relation to the three constants (3.111) is given by

$$4\boldsymbol{J}_0^e\boldsymbol{J}_0^{e\mathsf{T}} = \begin{pmatrix} c_1 & c_3 \\ c_3 & c_2 \end{pmatrix}.$$

When the element e is a parallelogram, then $\boldsymbol{\psi}_e$ is an affine mapping and the Jacobi matrix is constant over the element, i. e., $\boldsymbol{J}^e = \boldsymbol{J}_0^e$. In this case, our method is similar to the recently

[32] Also compare the interpretation of the deformation gradient $\boldsymbol{F}$ on page 111.

published work of Blasco [23], who solves incompressible flow problems with linear triangular elements. The difference to our method is discussed in Remark 3.8. Blasco proves stability and convergence of his method in a mesh-dependent norm and shows its superiority over the standard GLS method and over the method of Micheletti et al. [111] by means of numerical examples using anisotropic irregular grids.

In the case of an arbitrary quadrilateral element e, the mapping $\boldsymbol{\psi}_e$ is not necessarily affine and the Jacobi matrix is not constant over the element. Consequently, the Jacobi matrix evaluated in the element centre, $\boldsymbol{J}_0^e$, only represents parts of the element's geometry information. Consider, for example, the two elements in Figure 3.9: Although the shapes of the elements differ significantly, they share the same local coordinate system. To incorporate more geometric in-

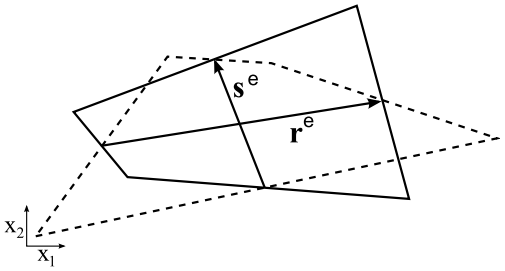


Figure 3.9: Two differently shaped elements with the same local coordinate system.

formation of an arbitrarily shaped element, one would have to use the locally varying Jacobi matrix $\boldsymbol{J}^e$, instead of the constant one, $\boldsymbol{J}_0^e$. The resulting stabilisation method, which we want to denote with PPPJ, is then given by

$$c_{\mathsf{PPPJ}}(p^h,q) := -\frac{2\alpha}{\mu}\sum_e (\boldsymbol{J}^{e\mathsf{T}}\boldsymbol{\nabla} p^h, \boldsymbol{J}^{e\mathsf{T}}\boldsymbol{\nabla} q)_e\,. \tag{3.115}$$

However, numerical tests (which are not presented here) show that there is practically no difference between PPPLC and PPPJ: For all the configurations that are used in Section 3.3.5, the error graphs of the two methods are indistinguishable. Furthermore, iterative solvers show equal convergence behaviour. We conclude that it is sufficient to use the constant Jacobian $\boldsymbol{J}_0^e$ also for elements that are not parallelograms: Incorporating more details of the element geometry by means of the PPPJ method has no impact on the quality of the stabilisation. Therefore, we abandon this idea and use the PPPLC method for all kinds of elements.

REMARK 3.8

The difference between our PPPLC stabilisation method (3.114) and the one described in Blasco [23] (which we want to denote with PPPBL) is that the latter uses the *non-transposed* Jacobian where we use the *transposed*, i. e., the stabilisation terms are given by

$$c_{\mathsf{PPPBL}}(p^h,q) = -\frac{2\alpha}{\mu}\sum_e (\boldsymbol{J}_0^e\boldsymbol{\nabla} p^h, \boldsymbol{J}_0^e\boldsymbol{\nabla} q)_e = -\frac{2\alpha}{\mu}\sum_e (\boldsymbol{J}_0^{e\mathsf{T}}\boldsymbol{J}_0^e\boldsymbol{\nabla} p^h, \boldsymbol{\nabla} q)_e\,. \tag{3.116}$$

As counterparts of the three constants (3.111) we thus obtain

$$\tilde{c}_1 := (r_1^e)^2 + (r_2^e)^2, \quad \tilde{c}_2 := (s_1^e)^2 + (s_2^e)^2, \quad \tilde{c}_3 := r_1^e s_1^e + r_2^e s_2^e. \tag{3.117}$$

Both methods can be interpreted as extension of the PPPBR method of Becker and Rannacher [19] (which takes only the diagonal of the Jacobian into account), but only our version using the transpose can be motivated geometrically (see page 159). The motivation to use the non-transposed Jacobian is simply given by the fact that this method often leads to smaller pressure L^2 errors [24]. We confirm this observation in Section 3.3.5.1. However, in the same section we also show that iterative solvers have severe problems to solve the linear equation systems that result from using the non-transposed Jacobian.

We further illustrate the difference between the stabilisation methods PPPLC and PPPBL by means of an example. Consider a Cartesian anisotropic element e of width $h_{e,1} = a$ and height $h_{e,2} = 1$ with $a \gg 1$ (cf. Figure 3.7a on page 157). Then consider the rotation of this element around its centre by the angle δ, which can be represented by the Jacobi matrix

$$\boldsymbol{J}^e = \frac{1}{2}\begin{pmatrix} r_1^e & s_1^e \\ r_2^e & s_2^e \end{pmatrix} = \frac{1}{2}\begin{pmatrix} a\cos(\delta) & -\sin(\delta) \\ a\sin(\delta) & \cos(\delta) \end{pmatrix}.$$

The three constants (3.111) of our PPPLC method (3.114) and the three constants (3.117) of the PPPBL method (3.116) are then respectively given by

$$\begin{aligned} c_1 &= a^2\cos^2(\delta) + \sin^2(\delta) & \tilde{c}_1 &= a^2, \\ c_2 &= a^2\sin^2(\delta) + \cos^2(\delta) & \tilde{c}_2 &= 1, \\ c_3 &= (a^2-1)\cos(\delta)\sin(\delta) & \tilde{c}_3 &= 0. \end{aligned}$$

One can see that method (3.116) completely ignores the rotation of the element, i. e., in this case it reduces to the method of Becker and Rannacher [19] which is actually intended for Cartesian meshes only. Currently, we are not able to explain why the PPPBL method yields in many cases slightly better pressure approximations than PPPLC. However, the fact that standard iterative solving schemes have extreme problems with the linear systems resulting from PPPBL (see Section 3.3.5.1) indicates that the approach might be unfeasible.

3.3.3 A Pressure Projection Approach

Bochev et al. [27] introduced a stabilisation procedure for the Stokes problem that is based on pressure projections instead of residuals. To our knowledge, this approach has not been applied to CSM problems in the literature yet. The authors derive the method by characterising the instabilities of the Q_1/Q_1 pair in terms of a *weak* inf-sup *condition* (cf. relation (3.99)):

$$\exists \beta_1, \beta_2 > 0 : \sup_{\mathbf{0} \neq \boldsymbol{v} \in \boldsymbol{V}^h} \frac{b(\boldsymbol{v}, q)}{\|\boldsymbol{v}\|_1} \geqslant \beta_1 \|q\|_0 - \beta_2 h \|\nabla q\|_0, \qquad q \in M^h. \tag{3.118}$$

For a proof, see Bochev et al. [27, Lemma 2.1]. The term $h\|\nabla q\|_0$ can be interpreted as the 'destabilising' contribution which has to be 'balanced'. The GLS and PPP methods introduced in the previous sections do this by directly incorporating this term into the stabilisation. Bochev et al., however, use a different characterisation of the critical term. They define the projection operator

$$\Pi : L^2 \to Q_0,$$

where Q_0 denotes the space of piecewise constant functions, and prove that there exists a constant $C > 0$ such that

$$Ch\|\nabla q\|_0 \leqslant \|q - \Pi q\|_0, \qquad q \in M^h.$$

Then, the authors deduce the alternative form of the weak inf-sup condition,

$$\exists C_1, C_2 > 0 : \sup_{\mathbf{0} \neq \boldsymbol{v} \in \boldsymbol{V}^h} \frac{b(\boldsymbol{v}, q)}{\|\boldsymbol{v}\|_1} \geqslant C_1 \|q\|_0 - C_2 \|q - \Pi q\|_0, \qquad q \in M^h,$$

in which the constants do not depend on the parameter h. Based on this characterisation of the Q_1/Q_1 instability, the authors suggest to add the (suitably scaled) bilinear form $c(p^h, q) = (p^h - \Pi p^h, q - \Pi q)_\Omega$ to the variational form of the Stokes equation. Under two mild assumptions concerning the projection operator Π, stability and optimal convergence of the method is proven (see Bochev et al. [27, Theorem 4.1, Theorem 5.1]). The authors choose the projection operator defined by

$$\Pi q|_e := \frac{1}{\operatorname{area}(e)} \int_e q \, \mathrm{d}x,$$

i. e., the mean value of q over the element e. This choice fulfils all necessary requirements, and has the additional advantages that it can be computed with standard techniques available in each finite element code, that it can be computed locally on element level, and that the resulting stabilisation matrix is symmetric. When adapting the scaling factor to the elasticity case, the variational problem (see equation (3.100) on page 153) can finally be stabilised by adding the bilinear form

$$c_{\mathsf{PROJ}}(p^h, q) := -\frac{1}{2\mu} \sum_e (p^h - \Pi p^h, q - \Pi q)_e \,. \tag{3.119}$$

Obviously, this method has some advantages – beside the already mentioned ones – compared to the GLS stabilisation methods: It is absolutely stable, i. e., there is no need for a stabilisation parameter, and it does not depend on any mesh-dependent parameters (like the characteristic element size). However, similar to the PPP method of Brezzi and Douglas [38], it is not consistent. Interestingly, from their analysis the authors deduce that their method and the PPP method have similar stability properties. On anisotropic/irregular meshes, however, the PPP method has to be adapted to retain stability (PPPBR, PPPLC, PPPJ). It is not clear a priori if the presented projection method can robustly deal with such mesh irregularities, or if an adaption is necessary, as well. We want to examine this by means of detailed numerical tests in Section 3.3.5.

3.3.4 Finite Deformation

In this section we want to apply the stabilisation methods introduced in the previous sections to finite deformation elasticity. To our knowledge, analytical convergence and stability results are not available for this case. However, many authors successfully applied stabilised equal-order element pairs to nonlinear solid mechanics problems (see the overview on page 146).

As described in the previous sections, the stabilisation terms take the geometry of the element into account, for example by defining one constant value characterising the element size (PPP) or by using the element's local coordinate system (PPPLC). In finite deformation computations one has to distinguish between the current and the reference configuration (cf. Section 3.1.1). So, the question arises how to treat the stabilisation terms and whether the geometry of the deformed or of the undeformed element should be considered.

Before we discuss two different variants, we introduce some further notations. For an element $e \subset \Omega$ in the reference configuration we denote its deformed counterpart in the current configuration as $e^{\mathsf{c}} \subset \Omega^{\mathsf{c}}$. Correspondingly, $(\cdot,\cdot)_{e^{\mathsf{c}}}$ denotes the L^2 inner product over the deformed element e^{c}, i. e.,

$$(a,b)_{e^{\mathsf{c}}} := \int_{e^{\mathsf{c}}} a(\boldsymbol{x})b(\boldsymbol{x})\,\mathrm{d}x, \qquad a,b \in L^2(e^{\mathsf{c}},\mathbb{R}),$$
$$(\boldsymbol{a},\boldsymbol{b})_{e^{\mathsf{c}}} := \int_{e^{\mathsf{c}}} \boldsymbol{a}(\boldsymbol{x})\cdot\boldsymbol{b}(\boldsymbol{x})\,\mathrm{d}x, \qquad \boldsymbol{a},\boldsymbol{b} \in L^2(e^{\mathsf{c}},\mathbb{R}^2),$$

while

$$(a,b)_e := \int_e a(\boldsymbol{X})b(\boldsymbol{X})\,\mathrm{d}X, \qquad a,b \in L^2(e,\mathbb{R}),$$
$$(\boldsymbol{a},\boldsymbol{b})_e := \int_e \boldsymbol{a}(\boldsymbol{X})\cdot\boldsymbol{b}(\boldsymbol{X})\,\mathrm{d}X, \qquad \boldsymbol{a},\boldsymbol{b} \in L^2(e,\mathbb{R}^2)$$

is the L^2 inner product over the element e in the reference configuration. The characteristic size h_e and the local coordinate system $(\boldsymbol{r}^e,\boldsymbol{s}^e)$ of the undeformed element are correspondingly denoted by $h_{e^{\mathsf{c}}}$ and $(\boldsymbol{r}^{e^{\mathsf{c}}},\boldsymbol{s}^{e^{\mathsf{c}}})$, respectively, for the deformed element. The pull back (cf. Section 3.1.1) of $\mathbf{grad}(q)$ is given by $\boldsymbol{F}^{-\mathsf{T}}\,\mathbf{Grad}(q)$ (see, e. g., Stein and Barthold [151, chapter 3]).

3.3.4.1 Stabilisation Based on the Current Configuration

In the first variant the stabilisation is applied to the current configuration. This means on the one hand, that the stabilisation constants are based on the deformed geometry, and on the other hand, that a pull back to the reference configuration is performed. The motivation for this approach is that the current configuration describes the 'real' physical state of the body, and consequently its actual shape. We want to denote this approach by appending '-C' (for 'current') to the name of the stabilisation method (e. g., PPP-C). Klaas et al. [95] apply this variant to the SGLS

stabilisation (see equation (3.104) on page 155). They include the second order stabilisation terms and also derive the corresponding linearisation, though, but for the numerical examples they only use the stabilised linear/linear element, such that their method actually reduces to PPP-C. Only in Maniatty et al. [108], the authors extend their examinations to higher order methods.

The PPP stabilisation terms of the current configuration transform into the reference configuration as follows (cf. Klaas et al. [95]):

$$\begin{aligned}
\hat{c}_{\text{PPP-C}}(p^h, q) &:= -\frac{\alpha}{2\mu} \sum_e h_{e^c}^2 \big(\mathbf{grad}(p^h), \mathbf{grad}(q) \big)_{e^c} \\
&= -\frac{\alpha}{2\mu} \sum_e h_{e^c}^2 \big(J \boldsymbol{F}^{-\mathsf{T}} \mathbf{Grad}(p^h), \boldsymbol{F}^{-\mathsf{T}} \mathbf{Grad}(q) \big)_e \\
&= -\frac{\alpha}{2\mu} \sum_e h_{e^c}^2 \big(J \boldsymbol{F}^{-1} \boldsymbol{F}^{-\mathsf{T}} \mathbf{Grad}(p^h), \mathbf{Grad}(q) \big)_e \\
&= -\frac{\alpha}{2\mu} \sum_e h_{e^c}^2 \big(J \boldsymbol{C}^{-1} \mathbf{Grad}(p^h), \mathbf{Grad}(q) \big)_e .
\end{aligned}$$

To derive the PPPLC-C method for finite deformation, we have to consider the local coordinate system of the deformed element, $(\boldsymbol{r}^{e^c}, \boldsymbol{s}^{e^c})$. Analogously to definition (3.110) we obtain

$$\begin{aligned}
\hat{c}_{\text{PPPLC-C}}(p^h, q) &:= -\frac{\alpha}{2\mu} \sum_e \big(\boldsymbol{r}^{e^c} \cdot \mathbf{grad}(p^h), \boldsymbol{r}^{e^c} \cdot \mathbf{grad}(q) \big)_{e^c} + \big(\boldsymbol{s}^{e^c} \cdot \mathbf{grad}(p^h), \boldsymbol{s}^{e^c} \cdot \mathbf{grad}(q) \big)_{e^c} \\
&= -\frac{\alpha}{2\mu} \sum_e \left(\underbrace{\begin{pmatrix} r_1^{e^c} & r_2^{e^c} \\ s_1^{e^c} & s_2^{e^c} \end{pmatrix}}_{=:\boldsymbol{H}} \mathbf{grad}(p^h), \begin{pmatrix} r_1^{e^c} & r_2^{e^c} \\ s_1^{e^c} & s_2^{e^c} \end{pmatrix} \mathbf{grad}(q) \right)_{e^c} \\
&= -\frac{\alpha}{2\mu} \sum_e \big(\boldsymbol{H}\, \mathbf{grad}(p^h), \boldsymbol{H}\, \mathbf{grad}(q) \big)_{e^c} \\
&= -\frac{\alpha}{2\mu} \sum_e \big(J \underbrace{\boldsymbol{H}\boldsymbol{F}^{-\mathsf{T}}}_{=:\tilde{\boldsymbol{H}}} \mathbf{Grad}(p^h), \boldsymbol{H}\boldsymbol{F}^{-\mathsf{T}}\, \mathbf{Grad}(q) \big)_e \\
&= -\frac{\alpha}{2\mu} \sum_e \big(J \tilde{\boldsymbol{H}}\, \mathbf{Grad}(p^h), \tilde{\boldsymbol{H}}\, \mathbf{Grad}(q) \big)_e \\
&= -\frac{\alpha}{2\mu} \sum_e \big(J \tilde{\boldsymbol{H}}^{\mathsf{T}} \tilde{\boldsymbol{H}}\, \mathbf{Grad}(p^h), \mathbf{Grad}(q) \big)_e .
\end{aligned} \tag{3.120}$$

Note, that the matrix $\boldsymbol{H}$ is constant, i. e., it is not affected by the pull back operation and in the integral term $\big(\boldsymbol{H}\boldsymbol{F}^{-\mathsf{T}}\, \mathbf{Grad}(p^h), \boldsymbol{H}\boldsymbol{F}^{-\mathsf{T}}\, \mathbf{Grad}(q) \big)_e$ only the part $\boldsymbol{F}^{-\mathsf{T}}\, \mathbf{Grad}(\cdot)$ has to be evaluated in the cubature points.

In the linear case we presented an alternative formulation of the PPPLC method using the Jacobi matrix $\boldsymbol{J}_0^e$. This is also possible for the finite deformation case. Therefore, we consider the composed mapping $\boldsymbol{\varphi} \circ \boldsymbol{\psi}_e : e^b \to e^c$ from the basis element into the deformed element (cf. equation (3.65) on page 138), whose Jacobi matrix is simply given by $\boldsymbol{F}\boldsymbol{J}^e$. The local coordinate axes of the deformed element can be obtained by applying the transpose of this matrix to the

local coordinate axes of the basis element. So, corresponding to relation (3.113) on page 159 we have

$$\boldsymbol{H} = \begin{pmatrix} r_1^{e^c} & r_2^{e^c} \\ s_1^{e^c} & s_2^{e^c} \end{pmatrix} = 2\Big(\boldsymbol{F}\big(\boldsymbol{\psi}_e(0,0)\big)\boldsymbol{J}_0^e\Big)^{\mathsf{T}} = 2\boldsymbol{J}_0^{e\mathsf{T}}\boldsymbol{F}\big(\boldsymbol{\psi}_e(0,0)\big)^{\mathsf{T}}.$$

For a consistency check, consider the special case that the deformation $\boldsymbol{\varphi}$ is affine over the element. Then the deformation tensor $\boldsymbol{F}$ is constant, i. e., $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{F}\big(\boldsymbol{\psi}_e(0,0)\big), \boldsymbol{X} \in e$, and the matrix $\tilde{\boldsymbol{H}}$ in equation (3.120) reduces to

$$\tilde{\boldsymbol{H}} = \boldsymbol{H}\boldsymbol{F}^{-\mathsf{T}} = 2\boldsymbol{J}_0^{e\mathsf{T}}\boldsymbol{F}^{\mathsf{T}}\boldsymbol{F}^{-\mathsf{T}} = 2\boldsymbol{J}_0^{e\mathsf{T}}.$$

This means, when the deformation mapping is linear, then the stabilisation is consistent with the stabilisation for the linearised small deformation case (see equation (3.114) on page 159).

To adapt the pressure projection stabilisation of Bochev et al. [27] (see Section 3.3.3) to the finite deformation setting based on the current configuration, we introduce the projection operator

$$\Pi^{\mathsf{c}} q|_{e^c} := \frac{1}{\operatorname{area}(e^{\mathsf{c}})} \int_{e^{\mathsf{c}}} q \,\mathrm{d}x,$$

which calculates the mean value of q over the deformed element e^{c}. Observing that the term $q - \Pi^{\mathsf{c}} q$ is invariant under the pull back operation, we obtain the stabilisation

$$\begin{aligned} \hat{c}_{\text{PROJ-C}}(p^h, q) &:= -\frac{1}{2\mu}\sum_e (p^h - \Pi^{\mathsf{c}} p^h, q - \Pi^{\mathsf{c}} q)_{e^{\mathsf{c}}} \\ &= -\frac{1}{2\mu}\sum_e \big(J(p^h - \Pi^{\mathsf{c}} p^h), q - \Pi^{\mathsf{c}} q\big)_e . \end{aligned}$$

3.3.4.2 Stabilisation Based on the Reference Configuration

The second variant is a ‘naive’ approach: We simply ignore the difference between reference and current configuration and directly apply the stabilisation to the Lagrangian form. This means that no pull back of the stabilisation terms is performed and the geometry of the undeformed element is taken into account. On the one hand this seems to be a ‘numerical crime’, on the other hand, however, one can argue as follows:

Due to the nonlinear nature of the finite deformation setting, there is no guarantee that the stabilisation methods adapted from the linearised setting are successful at all – for some deformations they might work, for others they might not. Also compare the discussion in Le Tallec [105, Section 14] about the use of stable finite elements for finite deformation. The stabilisation terms of the first variant based on the current configuration are themselves nonlinear and might consequently ‘add to this uncertainty’. To avoid this, one should use the linear stabilisation ‘as it is’ and apply it directly to the (nonlinear) variational equation in Lagrangian formulation – even if this approach ignores the physical interpretation on which the first variant is based.

We want to denote this approach by appending '-R' (for 'reference') to the name of the stabilisation method (e. g., PPP-R). The stabilisation terms for the PPP-R method are

$$\hat{c}_{\text{PPP-R}}(p^h, q) := -\frac{\alpha}{2\mu} \sum_e h_e^2 \big(\mathbf{Grad}(p^h), \mathbf{Grad}(q) \big)_e ,$$

the PPPLC-R method is given by

$$\begin{aligned} \hat{c}_{\text{PPPLC-R}}(p^h, q) :=& -\frac{\alpha}{2\mu} \sum_e \big(\boldsymbol{r}^e \cdot \mathbf{Grad}(p^h), \boldsymbol{r}^e \cdot \mathbf{Grad}(q) \big)_e + \big(\boldsymbol{s}^e \cdot \mathbf{Grad}(p^h), \boldsymbol{s}^e \cdot \mathbf{Grad}(q) \big)_e \\ =& -\frac{2\alpha}{\mu} \sum_e (\boldsymbol{J}_0^{e\mathsf{T}} \mathbf{Grad}(p^h), \boldsymbol{J}_0^{e\mathsf{T}} \mathbf{Grad}(q))_e , \end{aligned}$$

and finally the projection method PROJ-R reads

$$\hat{c}_{\text{PROJ-R}}(p^h, q) := -\frac{1}{2\mu} \sum_e (p^h - \Pi p^h, q - \Pi q)_e .$$

In Section 3.3.5 we examine whether this 'naive' approach is successful.

3.3.5 Numerical Tests

In this section we want to examine some aspects of the spatial discretisation treated in the previous sections by means of numerical examples. We consider the following issues in the context of linearised elasticity:

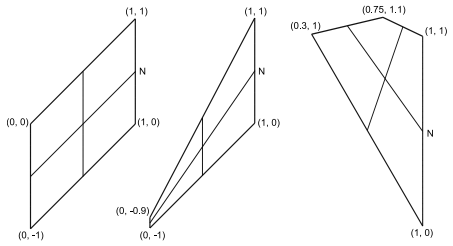
- comparison of the presented stabilisation methods (Section 3.3.5.1)
- comparison of pure displacement and mixed formulation for increasing incompressibility (Section 3.3.5.2)
- effect of the stabilisation methods' inconsistency (Section 3.3.5.3)

Finally, we compare the four stabilisation variants in the context of finite deformation elasticity (Section 3.3.5.4). In all the following tests we are mainly interested in the accuracy of the computed solution. Only in Section 3.3.5.1 we also consider aspects of iterative solver performance.

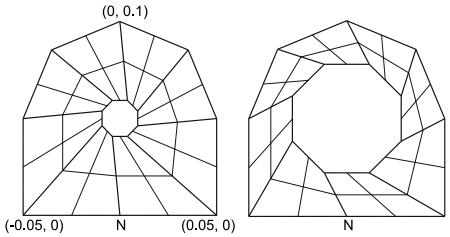
3.3.5.1 Comparison of Stabilisation Methods

We first compare different stabilisation methods in the context of linearised elasticity:

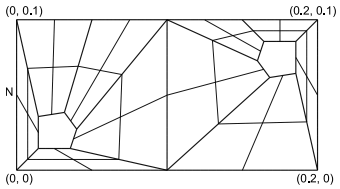
- PPPmin, PPPmax, PPParea - the simple PPP stabilisation method (3.106a) considering the three possibilities to compute the characteristic element size (see equation (3.108))



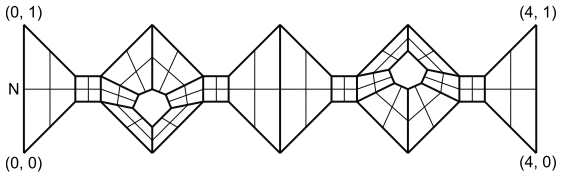
(a) QUAD1, QUAD2 and QUAD3; 1 subdomain.



(b) RUBBERBUSHING2 and RUBBERBUSHING6, holes with diameter 0.02 m and 0.06 m, resp.; 8 subdomains.



(c) BLOCKWITHHOLES; 10 subdomains.



(d) ACCORDION; 20 subdomains.

Figure 3.10: Level 1 macro decompositions of the configurations for the stabilisation tests. 'N' marks boundary segments with Neumann conditions, unmarked boundary segments exhibit Dirichlet conditions.

- PPPLC - our enhanced variant (3.112) of PPP considering the element's local coordinate system (the transposed Jacobian matrix, resp.)[33]
- PPPBL - Blasco's variant (3.116) of PPP for triangular elements considering the non-transposed Jacobian matrix, applied to quadrilaterals
- PROJ - the pressure projection method (3.119) of Bochev et al.

		max. AR								
		QUAD1			QUAD2			QUAD3		
lev	#DOF	ISO	ANISO1	ANISO2	ISO	ANISO1	ANISO2	ISO	ANISO1	ANISO2
3	8.67e+2	1.41	26.9	88.4	21.5	1.90e+2	3.16e+2	3.71	52.1	1.70e+2
4	3.27e+3			2.21e+2			7.89e+2			4.26e+2
5	1.27e+4			5.52e+2			1.97e+3			1.06e+3
6	4.99e+4			1.38e+3			4.93e+3			2.66e+3
7	1.98e+5			3.45e+3			1.23e+4			6.65e+3

Table 3.3: Number of DOF and maximal element aspect ratios (AR) of the different QUAD configurations.

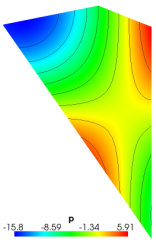
For all PPP variants we choose $\alpha = 0.1$ as stabilisation parameter. We want to assess the stabilisation methods for a wide variety of domain geometries and element shapes and thus perform tests on the seven configurations depicted in Figure 3.10. The three QUAD configurations are solved with three different refinement modes: ISO with anisotropy factors $a_F = a_L = a_I = 1.0$ (cf. equation (2.4) on page 27), ANISO1 with $a_F = 0.1, a_L = a_I = 1.0$ and ANISO2 with $a_F = a_L = a_I = 0.4$. All other configurations are refined isotropically. The resulting maximal element

[33] For an application of the PPPLC method to the Stokes- and Navier-Stokes equation, see Buijssen [44].

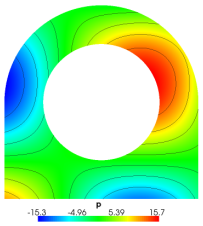
	RUBBUSH2		RUBBUSH6	BLOCKWITHHOLES		ACC	
lev	#DOF	max. AR	max. AR	#DOF	max. AR	#DOF	max. AR
3	1.73e+3	7.89	2.01	2.13e+3	8.49	4.29e+3	5.71
4	6.53e+3			8.11e+3		1.63e+4	
5	2.53e+4			3.16e+4		6.33e+4	
6	9.98e+4			1.25e+5		2.49e+5	
7	3.96e+5			4.95e+5		9.90e+5	

Table 3.4: Number of DOF and maximal element aspect ratios (AR) of the RUBBERBUSHING, BLOCKWITHHOLES and ACCORDION configurations.

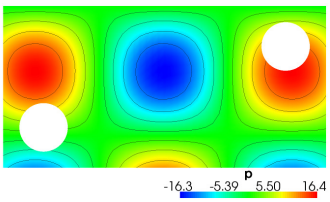
aspect ratios and the number of DOF are collected in Tables 3.3 and 3.4. The inner circles of the two rubber bushings[34] are slightly rotated to introduce some additional element irregularities.



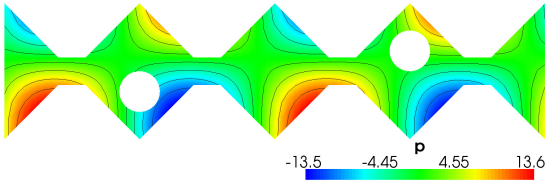
(a) QUAD3.



(b) RUBBERBUSHING6.



(c) BLOCKWITHHOLES.



(d) ACCORDION.

Figure 3.11: Computed pressure solutions of selected configurations, corresponding to analytical test functions.

At first we solve problems with right hand sides corresponding to known analytical functions and examine L^2 errors. For the QUAD and ACCORDION configurations we use the functions

$$\begin{aligned} \boldsymbol{u}(x_1,x_2) &= \begin{pmatrix} \sin(\pi x_1 - 0.7)\sin(\pi x_2 + 0.2) \\ \cos(\pi x_1 - 0.7)\cos(\pi x_2 + 0.2) \end{pmatrix}, \\ p(x_1,x_2) &= 2\mu \sin(\pi x_1 + 0.4)\cos(\pi x_2 - 0.3), \end{aligned} \tag{3.121}$$

and for the RUBBERBUSHING and BLOCKWITHHOLES configuration the functions

$$\begin{aligned} \boldsymbol{u}(x_1,x_2) &= \begin{pmatrix} \sin(2\pi(x_1 + 0.02))\sin(2\pi(x_2 + 0.02)) \\ \cos(2\pi(x_1 + 0.02))\cos(2\pi(x_2 + 0.02)) \end{pmatrix}, \\ p(x_1,x_2) &= -2\mu \sin(4\pi^2(x_1 + 0.02))\cos(4\pi^2(x_2 + 0.02)). \end{aligned} \tag{3.122}$$

[34] Rubber bushings connect two solid parts (e. g., metal) and facilitate flexible movement of the two parts. They are, for example, employed in suspension systems to minimise vibrations or to absorb shocks.

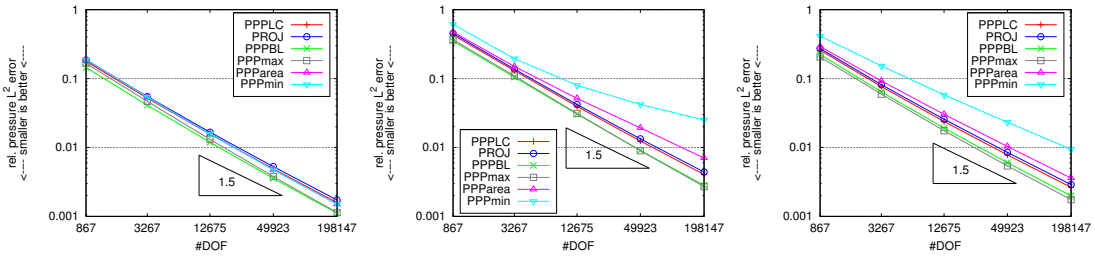
The pressure solutions are depicted in Figure 3.11. We use an incompressible rubber material with Poisson ratio $\nu = 0.5$ and shear modulus $\mu = 8.2$ MPa. All linear systems are treated with the direct solver UMFPACK [55]. Due to the a priori error estimates the pressure L^2 error converges at least with a rate of $O(h)$ (see Section 3.3.1). In practice, however, often a rate of order $O(h^{3/2})$ can be observed (see Becker and Braack [18] for a discussion and further references). This is also the case in most of our numerical experiments. Figure 3.12 shows the results for the nine QUAD configurations, and Figure 3.13 those of the other four configurations; Figures 3.12j and 3.13e, respectively, show the results on the highest level in more detail.

The most important observation is that all stabilisation methods – except for PPPmin and PPPareа – exhibit similar and smooth convergence behaviour; neither irregular element shapes, nor high element aspect ratios lead to significant deterioration of L^2 errors. The convergence behaviour of PPPmin and PPPareа, however, is clearly worse in some cases (see, e. g., Figure 3.12h). Ignoring PPPmin and PPPareа, it is interesting that the other methods show L^2 errors so close to each other, especially on the four more realistic geometries. The simple PPPmax stabilisation method (which does not account for element shapes or aspect ratios) performs surprisingly well: Figures 3.12j and 3.13e show that for all configurations it yields either the best or the second best result. A direct comparison of PPPLC and PPPBL shows that the latter yields (slightly) better results in many cases (as already mentioned in Remark 3.8 on page 160), but in some cases the former performs better (e. g., on the QUAD3-ISO configuration). The PROJ stabilisation is quantitatively slightly worse than PPPLC and PPPBL on the QUAD configurations, but qualitatively the three methods yield equal results.

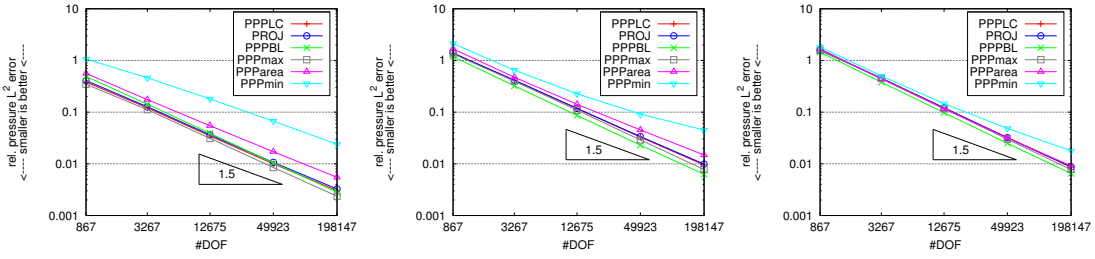
Now we examine real loading situations for the configurations RUBBERBUSHING and BLOCKWITHHOLES. We move the inner solid (which we assume to be rigid) of the rubber bushing, i. e., we apply a non-zero Dirichlet boundary condition on the inner circle. The outer boundary is fixed on the upper part and can move freely on its bottom part (see Figure 3.14a). For the small hole and large hole rubber bushing we apply a movement of $\boldsymbol{u} = (-0.01\text{ m}, -0.01\text{ m})^{\mathsf{T}}$ and $\boldsymbol{u} = (-0.003\text{ m}, -0.003\text{ m})^{\mathsf{T}}$, respectively. The block is loaded by a vertical line force of $F = -10$ MPa, i. e., we apply a non-zero Neumann boundary condition. The top side of the block is fixed in x_1-direction, while the bottom side is fixed in x_2-direction (see Figure 3.14d). We use a rubber material with Poisson ratio of $\nu = 0.49999$ and shear modulus $\mu = 8.2$ MPa. The resulting deformation and the pressure distribution are displayed in Figures 3.14b, 3.14c and 3.14e. We enquire the computed pressure solution p^h in the points $\boldsymbol{A}$ marked in Figures 3.14a and 3.14d. Note that these points lie in the interior of the domain; values on the boundary are not reliable due to the inconsistency of the stabilisation (cf. Remark 3.6 on page 156 and Section 3.3.5.3). Lacking a better reference solution, we compare the values to the finite element solution p^* obtained on the level 10 grid exhibiting 25.2 M DOF and 31.5 M DOF, respectively. The plots in Figure 3.15 display the relative error in percent, i. e.,

$$e_{\text{rel}} := 100\left|\frac{p^h(\boldsymbol{A}) - p^*(\boldsymbol{A})}{p^*(\boldsymbol{A})}\right|.$$

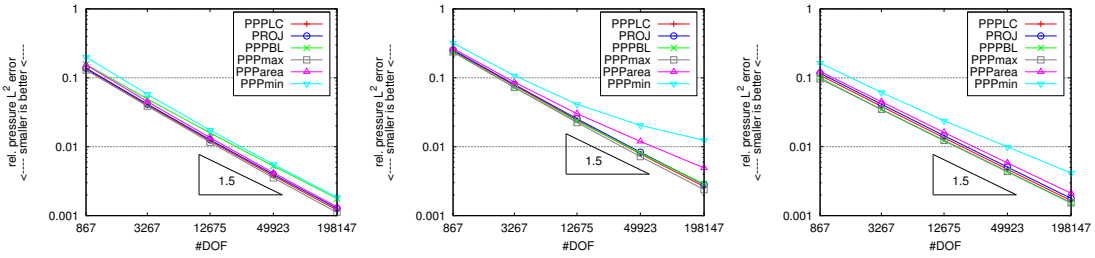
We can observe that the results partially differ from those obtained in the analytical tests above. On the RUBBERBUSHING6 configuration (Figure 3.15b) the PPPmax method, for example,



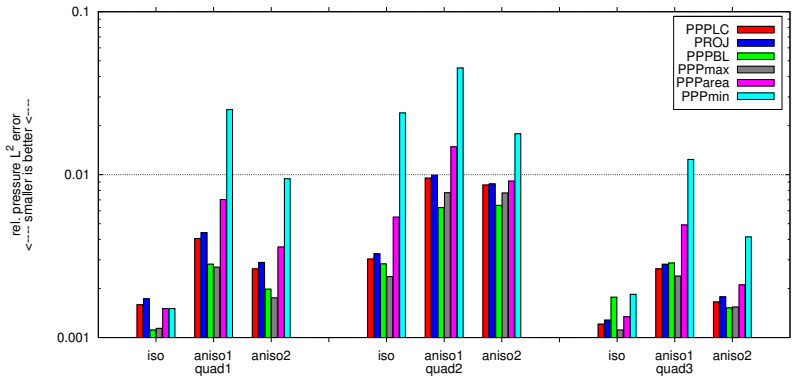
(a) QUAD1-ISO. **(b)** QUAD1-ANISO1. **(c)** QUAD1-ANISO2.



(d) QUAD2-ISO. **(e)** QUAD2-ANISO1. **(f)** QUAD2-ANISO2.

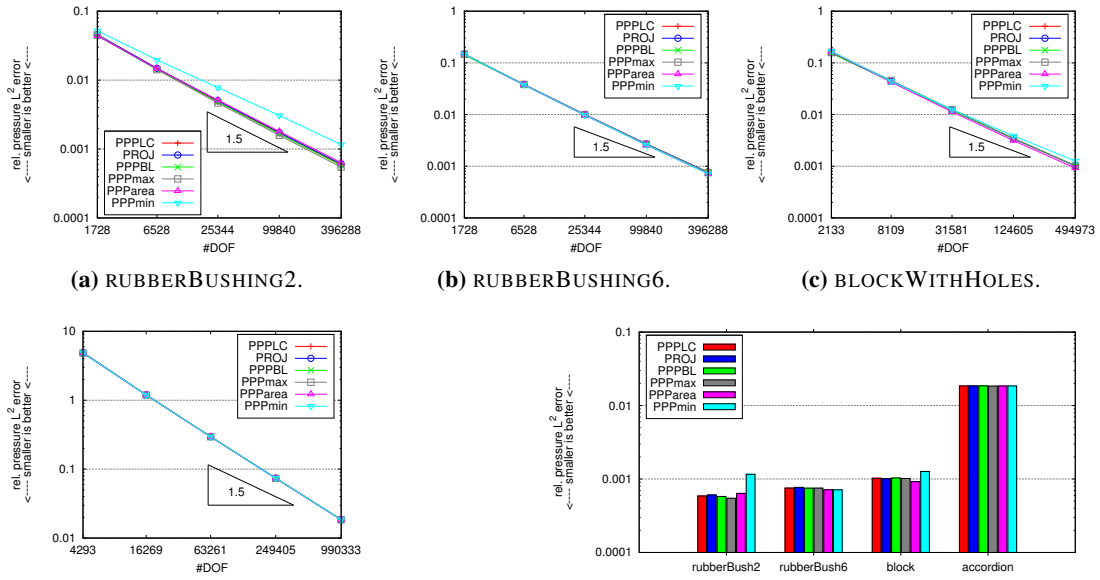


(g) QUAD3-ISO. **(h)** QUAD3-ANISO1. **(i)** QUAD3-ANISO2.



(j) Results of the nine configurations above on level 7 (198147 DOF).

Figure 3.12: L^2 error between the computed and the analytical pressure solution.



(a) RUBBERBUSHING2. **(b)** RUBBERBUSHING6. **(c)** BLOCKWITHHOLES.

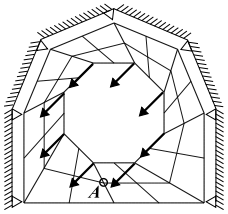
(d) ACCORDION. **(e)** Results of the four configurations on level 7.

Figure 3.13: L^2 error between the computed and the analytical pressure solution.

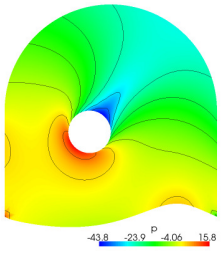
performs worst now, on the BLOCKWITHHOLES configuration (Figure 3.15c) it performs best. While PPPmin and PPParea show very large errors on the coarser RUBBERBUSHING6 grids, they yield the smallest error on the highest grid level. PPPLC and PROJ are very close together on all three configurations, PPPBL performs slightly worse. It is interesting that these three methods seem to exhibit a slightly worse convergence rate than the three simple PPP methods on the BLOCKWITHHOLES configuration (Figure 3.15c). In summary, a clear favourite can not be identified; it clearly depends on the configuration, which of the six methods performs best, and the differences are not significant.

We now want to show that there *are* crucial differences between the six stabilisation methods when considering iterative solving behaviour. Saddle point solvers are not introduced before Chapter 5, but we use three typical representatives already now, two segregated and one coupled method. For details about the these solvers, see Section 5.1. Figure 3.16 shows the convergence rates of the three solvers for six of the QUAD configurations (see Figure 3.10a), using the known analytical solution (3.121). The iterative solvers stop the iteration when the initial residual norm is reduced by the factor 10^{-12}. The coupled solver `BiCG-MG-VankaP` is not efficient for strongly anisotropic meshes, so it is applied to only two of the grids (Figure 3.16c). Missing bars mean that the corresponding solution process diverged.

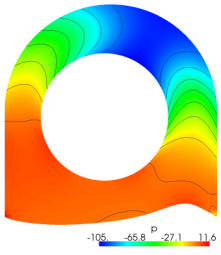
We can see that on the simple QUAD1-ISO grid, all stabilisation methods result in satisfying convergence rates: The solver `BiCG-SCBTria[BSor]` shows only slight variations for this grid (see Figure 3.16a), while for the other two solvers the convergence rates vary a little bit stronger, but are still below 0.15 (Figures 3.16b and 3.16c). For all the other grids, there are large



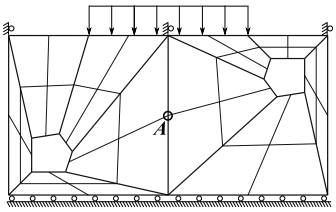
(a) RUBBERBUSHING6, $\boldsymbol{A} \approx (-0.0057, 0.0111)$ ($\boldsymbol{A} \approx (-0.0019, 0.0204)$ for RUBBERBUSHING2).



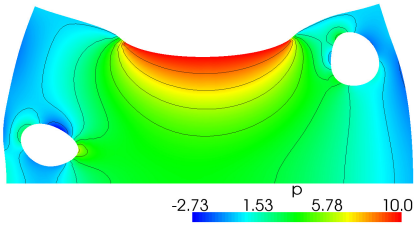
(b) RUBBERBUSHING2.



(c) RUBBERBUSHING6.

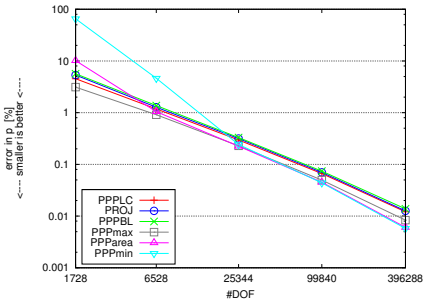


(d) BLOCKWITHHOLES, $\boldsymbol{A} = (0.1, 0.05)$.

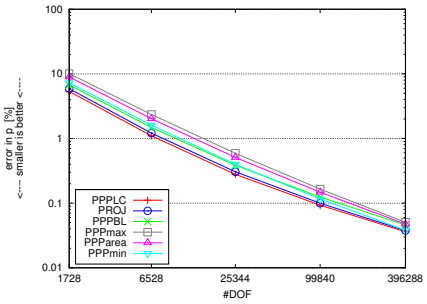


(e) BLOCKWITHHOLES.

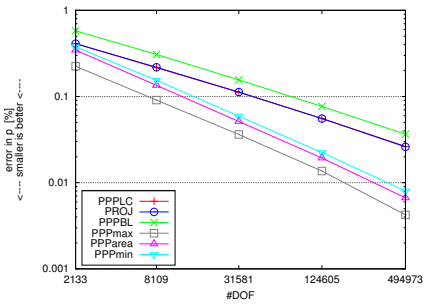
Figure 3.14: Boundary conditions and reference points $\boldsymbol{A}$ (left), deformation and pressure solution (right).



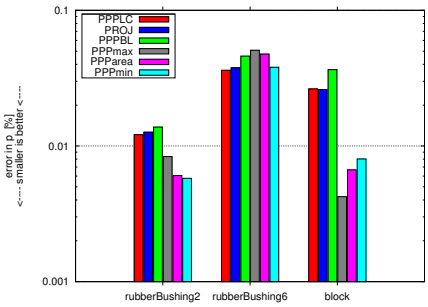
(a) RUBBERBUSHING2.



(b) RUBBERBUSHING6.

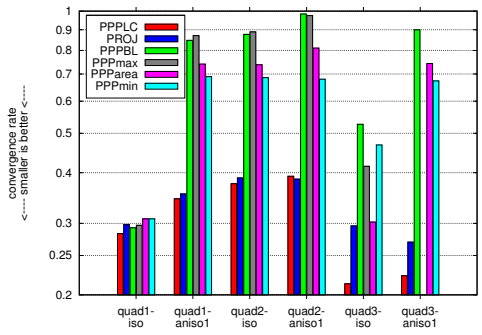


(c) BLOCKWITHHOLES.

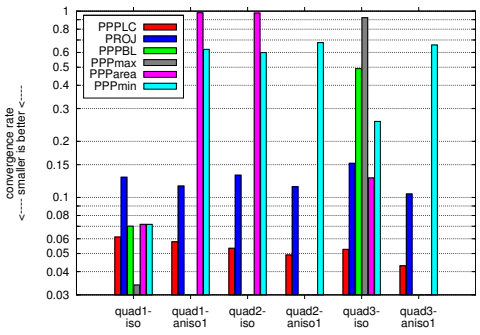


(d) Results of the configurations on level 7.

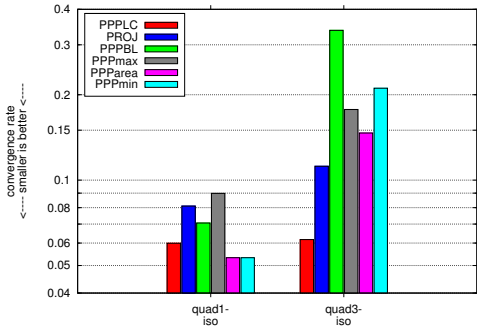
Figure 3.15: Relative error (%) between the computed pressure solutions and the level 10 reference solution.



(a) `BiCG-SCBTria[BSor]`.



(b) `BiCG-MG-SCBTria[BSor]`.



(c) `BiCG-MG-VankaP`.

Figure 3.16: Convergence rates of three saddle point solvers for selected grids. Note the varying, logarithmic scale.

differences: The only stabilisation methods that are successful for all solvers on all grids, are PPPLC, PROJ, and PPPmin. For the solver `BiCG-SCBTria[BSor]` the two methods PPPLC and PROJ are quite close together, with a slight advantage for PPPLC (especially on the two QUAD3 grids). For the other two solvers, PPPLC is clearly superior to PROJ. PPPmin exhibits by far worse convergence rates than PPPLC and PROJ for all three solvers.

The performance of the three stabilisation methods PPPBL, PPPmax and PPPareа is not robust: For the solver `BiCG-SCBTria[BSor]`, the convergence rates are above 0.7 on four of the six grids, partially even very close to 1.0. On the grid QUAD3-ANISO1, the method PPPmax even leads to divergence. The additional multigrid scheme in the solver `BiCG-MG-SCBTria[BSor]` seems to react more sensitively to the underlying stabilisation: Here, PPPBL and PPPmax result in divergence on four grids; PPPareа lets the solver diverge in two cases, while in the other two cases it converges with a rate very close to 1. For the solver `BiCG-MG-VankaP` the situation is not so dramatic on the two ISO grids, but on QUAD3-ISO we see again a clear deterioration for all methods but PPPLC, where the PPPBL method performs worst.

We want to remark that the described behaviour is qualitatively the same for all the other configurations used in this section, and also for all the other solvers that are described in Chapter 5. Hence, we can state that our stabilisation method PPPLC is clearly superior. PPPLC and PROJ

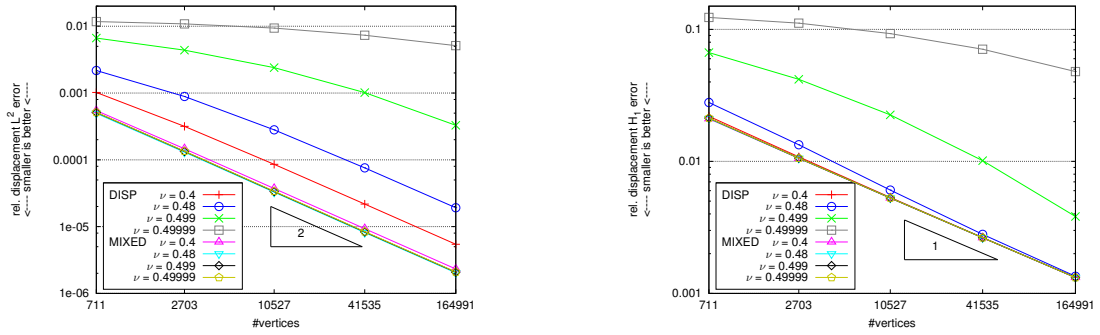
turn out to be the only ones that lead to a satisfying iterative solver performance for a wide range of grids and solvers. The three simple PPP methods and especially PPPBL produce linear systems that seem to be unfavourable for standard linear solving methods.

Summary

In principle, all the presented stabilisation methods exhibit (more or less) satisfying convergence behaviour. Only PPPmin and PPParea show clear deficiencies in single configurations (see, for example, Figure 3.12h). The PPPmax method behaves surprisingly well overall, partially even better than the three more complicated methods PPPLC, PPPBL and PROJ. The results show that none of the methods is superior in general, it depends on the specific configuration (geometry, loading state, boundary conditions, ...) which method performs best. So, when only considering error convergence rates, we have to conclude that the choice of the stabilisation method is not as crucial as one might believe – the simple variants PPPmin/max/area perform comparatively well, even on highly irregular and anisotropic grids (despite the noted deficiencies of PPPmin and PPParea). Realising this, one could refrain from implementing the more complicated methods PPPLC, PPPBL and PROJ, and stick to the simple variants. This, however, is only advisable when using direct solvers to treat the linear systems: In Figure 3.16 we showed that all stabilisation methods except for PPPLC and PROJ produce linear equation systems which are *extremely* difficult to solve for standard saddle point solvers. This is especially surprising for the PPPBL stabilisation. We can conclude that *from the set of presented stabilisation methods,* PPPLC *and* PROJ *are the only ones that can be used in practice for solving large-scale simulations with (nearly) incompressible material, where our* PPPLC *stabilisation method seems to result in better iterative solving behaviour.*

3.3.5.2 Pure Displacement vs. Mixed Formulation

In Section 3.2.2.2 we already showed the deficiencies of the pure displacement formulation when applied to nearly incompressible material in terms of a real loading situation (see Figure 3.6 on page 144). We now want to confirm these results in terms of an analytical test case and show the superiority of the mixed formulation. Therefore, we repeat the analytical test on the BLOCKWITHHOLES configuration (see equation (3.122) on page 168 and Figure 3.10c on page 167) for different degrees of compressibility, i. e., the Poisson ratio ranging between $\nu = 0.4$ and $\nu = 0.49999$. Figure 3.17 shows the errors for the pure displacement and the mixed formulation on different grid refinement levels. Figure 3.17a displays the relative displacement L^2 error, which is expected to decrease by $O(h^2)$, and Figure 3.17b the relative displacement H^1 error, which should decrease by $O(h)$ (see estimates (3.72) on page 139 and (3.103) on page 154). One can see that the pure displacement formulation clearly fails to achieve these convergence rates for higher degrees of incompressibility. The mixed formulation, however, shows the expected behaviour – nearly independently of the value of ν such that the graphs are hardly distinguishable. The L^2 error on the finest level, for example, is consequently three orders of magnitude smaller than that of the pure displacement formulation. Another interesting



(a) Relative displacement L^2 error.

(b) Relative displacement H^1 error.

Figure 3.17: Comparison of displacement ('DISP') and mixed formulation ('MIXED') for varying compressibility.

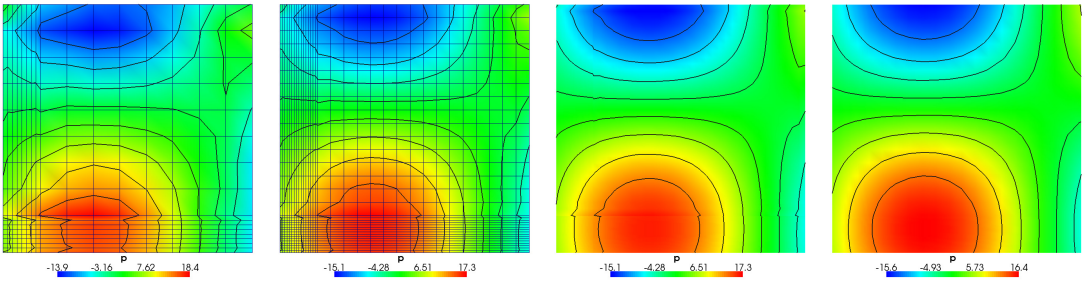
observation is that even for the compressible material ($\nu = 0.4$) the mixed formulation yields a smaller L^2 error. (The H^1 is also smaller, but the differences are only marginal.) This has already been observed by Suttmeier [153]; also compare Section 3.1.3.3. In Section 5.2.1.7 we show that the bad convergence of the pure displacement formulation for nearly incompressible material is accompanied by a strong degradation of linear solver performance. In summary, we can confirm that the pure displacement formulation is absolutely unsuited to treat nearly incompressible material and the switch to a more advanced finite element discretisation – like the mixed $\boldsymbol{u}/p$ formulation – is mandatory.

3.3.5.3 The Effect of Inconsistency

In Sections 3.3.1 and 3.3.3 we explained that the stabilisation methods used in this work are inconsistent. Although this fact does not impair the general convergence quality of the finite element solution, it nevertheless influences the pressure solution on a given grid. We want to illustrate two effects that result from the inconsistency: The degraded accuracy at the boundary (cf. Remark 3.6 on page 156) and the influence of jumps in element sizes (cf. Remark 3.7 on page 157).

We perform tests using the analytical solution (3.121) on the unitsquare which is slightly anisotropically refined towards the lower left corner with anisotropy factors $a_{\mathrm{F}} = 0.3, a_{\mathrm{L}} = a_{\mathrm{I}} = 1.0$ (cf. equation (2.4) on page 27). We tested our PPPLC method (see equation (3.112)) and the pressure projection method PROJ (see equation (3.119)). The results are qualitatively the same, so we only present those of the PPPLC method.

Figure 3.18 shows the pressure solution for four different refinement levels, additionally displaying the computational mesh for the two coarser levels (Figures 3.18a and 3.18b). On the first three grids we can clearly see the irregular behaviour of the contour lines; on the one hand, in the vicinity of the boundaries, and, on the other hand, where the element sizes jump due to the anisotropic refinement. We can also observe, that in both cases the distortion is limited to



(a) Lv. 4. **(b)** Lv. 5. **(c)** Lv. 6. **(d)** Lv. 10 (lv. 5 visualised).

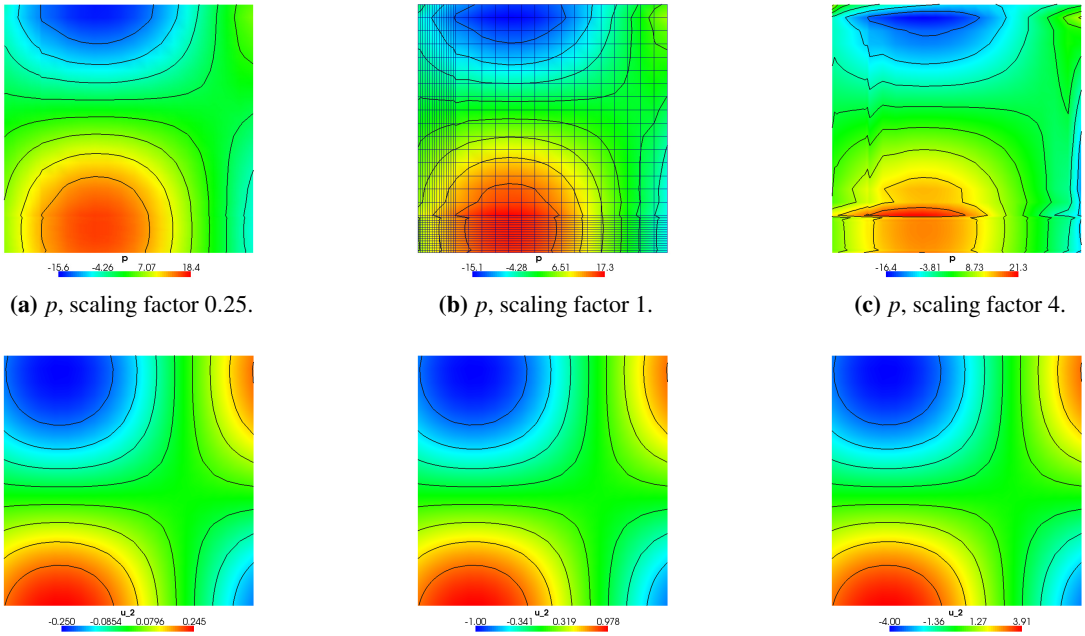
Figure 3.18: Computed pressure solution for four different refinement levels.

the adjacent element layers, such that with increasing refinement level the effect weakens. For the level 10 solution (Figure 3.18d), no artifacts are visible anymore, which is, of course, also due to the fact that the computed solution is interpolated on a coarser grid for visualisation. A representation on the level 10 grid would reveal correspondingly tiny irregularities in the pressure solution, too. Hence, the inconsistency error decreases with further grid refinement but will never vanish completely, which is exactly what has to be expected for inconsistent penalty methods.

We now want to illustrate that the negative effects of inconsistency are further influenced by the displacement solution. This is, of course, not surprising: The second derivative displacement terms (see equation (3.105) on page 155) are omitted in the stabilisation residual. Hence, one has to expect that the 'larger' these terms are (compared to the pressure), the larger the resulting error will be. We confirm this with a simple test: In the analytic reference solution (3.121) we scale the displacement function $\boldsymbol{u}$ with the factors 0.25 and 4, respectively, while the pressure function p remains unchanged. Figures 3.19a and 3.19c show the computed pressure solutions for these two cases, while Figure 3.19b shows the pressure solution for the case of the unscaled displacement function. One can clearly see that the irregularities of the computed pressure solution aggravate while the scaling factor of the analytical displacement function increases.

The illustrated inaccuracy of the pressure solution resulting from the inconsistency of the stabilisation methods *does not influence the displacement solution itself*, which exhibits perfectly smooth contour lines in the examples above (see Figures 3.19d, 3.19e and 3.19f). Hence, one could argue that this problem is irrelevant for real computations since the pressure only serves as auxiliary variable to solve the deformation problem efficiently. However, the material stresses resulting from the elastic deformation – usually the quantities of major interest in solid mechanical applications – directly depend on the pressure solution and will thus be affected, as well, by these irregularities.

Especially, when performing simulations on rather coarse meshes, one has to be aware that the computed results near boundaries or in regions with strongly varying element sizes might not be as accurate as in other regions of the mesh. *This has to be seen as major disadvantage of*



(a) p, scaling factor 0.25. **(b)** p, scaling factor 1. **(c)** p, scaling factor 4.

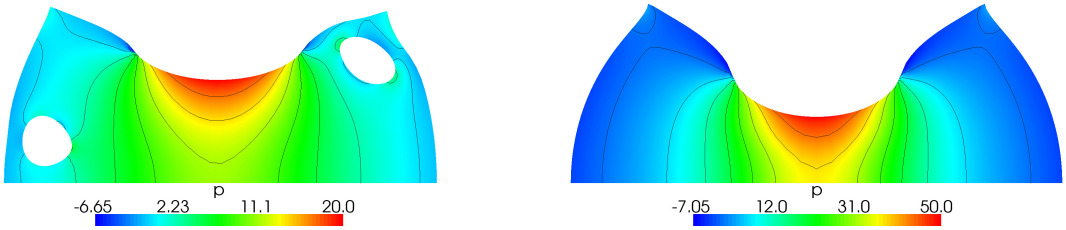
(d) u_2, scaling factor 0.25. **(e)** u_2, scaling factor 1. **(f)** u_2, scaling factor 4.

Figure 3.19: Computed level 5 solution for pressure (top row) and x_2-displacement (bottom row); analytical displacement function scaled with factors 0.25 (left), 1 (centre) and 4 (right).

the stabilised Q_1/Q_1 formulation, which can only be alleviated by using finer mesh resolutions and/or by avoiding too large jumps in neighbouring element shapes.

3.3.5.4 Comparison of Stabilisation Methods for Finite Deformation

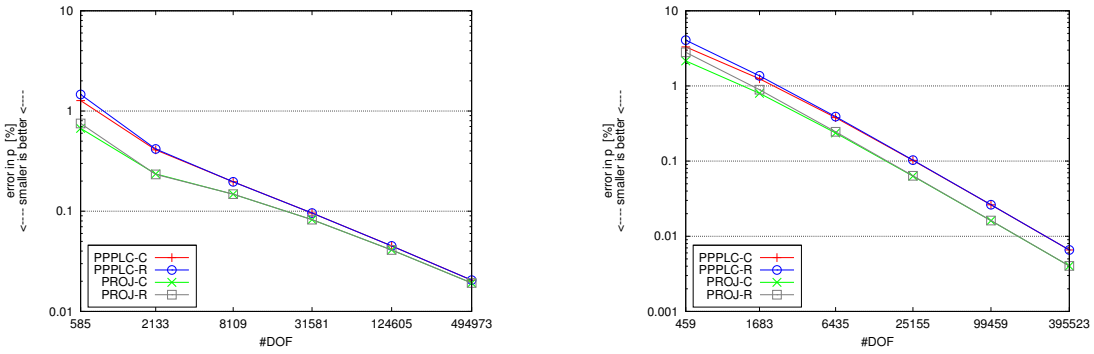
In Section 3.3.5.1 we claimed that the stabilisation methods PPPLC and PROJ are the only efficient ones for large scale computations. We now want to compare the finite deformation variants of these two methods developed in Section 3.3.4, i. e., we consider the four methods PPPLC-R, PPPLC-C, PROJ-R and PROJ-C. We use the real loading configuration of the BLOCKWITHHOLES (see Figures 3.14d and 3.14e) and apply a vertical line load of $F = -20$ MPa. Additionally, we use a variant of the block without holes, consisting of 8 subdomains (see Figure 4.1 on page 200), for which we increase the load to $F = -50$ MPa to obtain stronger deformations. This configuration is simply denoted by BLOCK. For both we use a nearly incompressible rubber material with $\nu = 0.49999$ and $\mu = 8.2$ MPa, and we apply the Neo-Hooke constitutive law (see equation (3.36) on page 126). The deformed bodies are depicted in Figure 3.20. The nonlinear iteration is stopped when the norm of the nonlinear residual is reduced by the factor 10^{-10}. See Section 4.3 for details on the nonlinear solving method. Lacking a better reference solution, we use the finite element pressure solution on level 10 exhibiting 31.5 M DOF in case



(a) BLOCKWITHHOLES, load $F = -20$. **(b)** BLOCK, load $F = -50$.

Figure 3.20: Computed deformation and pressure solution for the two block configurations.

of the BLOCKWITHHOLES and 25.2 M DOF in case of the BLOCK configuration. This reference solution is evaluated in the point $\boldsymbol{A} = (0.1, 0.05)$ and compared to the pressure solutions obtained with the four stabilisation methods on lower levels. Figure 3.21 shows the relative error (in percent). We can make the following observations:



(a) BLOCKWITHHOLES. **(b)** BLOCK.

Figure 3.21: Relative error (in percent) between the computed pressure solutions and the level 10 reference solution in the reference point $\boldsymbol{A} = (0.1, 0.05)$.

- All four stabilisation methods are successful in the sense that the nonlinear problems are solved without instabilities and they converge towards the same solution.
- Since the deformations are relatively large, the stabilisation terms of the two variants basing on the reference ('R') and the current configuration ('C') differ significantly. These differences, however, do not crucially affect the finite element solution: Only on the coarser grid levels, one can observe a slight advantage of the stabilisation methods based on the current configuration. On the finer grid levels, the differences vanish since the stabilisation terms tend to zero with the decreasing element sizes.
- The projection method PROJ yields slightly better results than the PPPLC method.

In summary, the computed examples indicate that it does not play an important role whether to use the stabilisation variant based on the reference configuration ('R') or that based on the cur-

rent configuration ('C'). This remains true when considering iterative solver behaviour (results are not presented here): We observe no significant differences in nonlinear and linear solver convergence. Hence, we state that both variants seem to be feasible. The variant based on the current configuration has the advantage that it corresponds to the actual physical state of the body and can thus be motivated more intuitively. It also seems to yield slightly better results on coarser grids. The variant based on the reference configuration, however, is rather naive and lacks a physical interpretation, but it is easier to implement.

The two stabilisation methods PPPLC and PROJ show qualitatively the same convergence behaviour, while PROJ yields slightly smaller errors. There are also no qualitative differences in terms of linear and nonlinear solver behaviour (results are not presented here). So, we state that both methods seem to be suitable to be employed for finite deformation elasticity problems.

We want to emphasise, that it is difficult to derive general statements from a few numerical example computations when dealing with nonlinear problems. The efficiency and the success of a numerical method often depends on the specific configuration at hand. Hence, we want the results presented above to be understood only as starting point for further examinations. For a more reliable assessment of the stabilisation methods more extensive studies are necessary. These, however, are not performed within this thesis but are part of future work.

3.4 Discretisation in Time

3.4.1 The Newmark Scheme

The time discretisation of transient problems is exemplarily explained by means of the mixed $\boldsymbol{u}/p$ formulation in the linear small deformation case (see equation (3.64) on page 135 and equation (3.107) on page 156). The remaining cases (pure displacement formulation and/or finite deformation; see equation (3.62) on page 135 and equations (3.47) and (3.49) on page 130) can be derived analogously. In a first step (ignoring stabilisation terms for the time being), the equation is discretised in space for a point in time, $t \in I = [0,T]$, resulting in the *semi-discretisation*

$$\begin{aligned} \rho\mathbf{M}\ddot{\mathbf{u}}(t) + \mathbf{A}\mathbf{u}(t) + \mathbf{B}\mathbf{p}(t) &= \mathbf{f}(t) \\ \mathbf{B}^{\mathsf{T}}\mathbf{u}(t) + \mathbf{C}^{\mathrm{c}}\mathbf{p}(t) &= \mathbf{0}, \end{aligned} \tag{3.123}$$

where $\mathbf{M}$ is the displacement mass matrix and the time-dependent coefficient vector $\mathbf{u}(t) = \big(\mathbf{u}_1(t), \mathbf{u}_2(t)\big)$ defines the finite element function

$$u_j^h(t,\boldsymbol{x}) = \sum_{i=1}^{n} (\mathbf{u}_j)_i(t)\phi_i(\boldsymbol{x}), \qquad j = 1,2.$$

$\mathbf{p}(t)$ and $\mathbf{f}(t)$ are defined analogously. In a second step, the semi-discrete equation (3.123), which represents a system of ordinary differential equations, is discretised in time. We use an implicit time integration scheme, which is very popular in CSM, the *Newmark scheme* [116].

Therefore, we partition the time interval I into K subintervals $I_k = [t_{k-1}, t_k]$, $k = 1, \ldots, K$, of equal length $\tau := t_k - t_{k-1}$ with $0 = t_0 < t_1 < \ldots < t_K = T$ and define $\mathbf{u}^{(k)} := \mathbf{u}(t_k)$. When the state of the system at time t_k is known, the Newmark scheme approximates displacement and velocity at time t_{k+1} by

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \tau\dot{\mathbf{u}}^{(k)} + \tau^2\Big(\big(\tfrac{1}{2} - \beta\big)\ddot{\mathbf{u}}^{(k)} + \beta\ddot{\mathbf{u}}^{(k+1)}\Big) \tag{3.124a}$$

$$\dot{\mathbf{u}}^{(k+1)} = \dot{\mathbf{u}}^{(k)} + \tau\Big((1-\gamma)\ddot{\mathbf{u}}^{(k)} + \gamma\ddot{\mathbf{u}}^{(k+1)}\Big), \tag{3.124b}$$

where β and γ are algorithmic parameters (see below). Solving equation (3.124a) for $\ddot{\mathbf{u}}^{(k+1)}$ (see equation (3.126a)) and insertion into equation (3.123) yields the dynamic equilibrium at time t_{k+1}:

$$\begin{aligned} &\begin{pmatrix} \frac{\rho}{\tau^2\beta}\mathbf{M} + \mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{C}^{\mathrm{c}} \end{pmatrix} \begin{pmatrix} \mathbf{u}^{(k+1)} \\ \mathbf{p}^{(k+1)} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{f}^{(k+1)} + \frac{\rho}{\tau^2\beta}\mathbf{M}\Big(\mathbf{u}^{(k)} + \tau\dot{\mathbf{u}}^{(k)} + \tau^2\big(\frac{1}{2} - \beta\big)\ddot{\mathbf{u}}^{(k)}\Big) \\ \mathbf{0} \end{pmatrix}. \end{aligned} \tag{3.125}$$

After solving this linear system the following updates have to be performed:

$$\ddot{\mathbf{u}}^{(k+1)} = \tfrac{1}{\tau^2\beta}\Big(\mathbf{u}^{(k+1)} - \mathbf{u}^{(k)} - \tau\dot{\mathbf{u}}^{(k)} - \tau^2\big(\tfrac{1}{2} - \beta\big)\ddot{\mathbf{u}}^{(k)}\Big), \tag{3.126a}$$

$$\dot{\mathbf{u}}^{(k+1)} = \dot{\mathbf{u}}^{(k)} + \tau\Big((1-\gamma)\ddot{\mathbf{u}}^{(k)} + \gamma\ddot{\mathbf{u}}^{(k+1)}\Big). \tag{3.126b}$$

In the case of linearised elasticity, the Newmark scheme is unconditionally stable for $\gamma \geqslant \frac{1}{2}$ and $\beta \geqslant \frac{1}{4}(\gamma + \frac{1}{2})^2$ and it conserves energy for $\gamma = \frac{1}{2}$ (see, e. g., Bathe [15]). Usually the parameters are chosen to be $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$.

It is known, that the stability and conservation properties of the Newmark scheme do not necessarily hold in the case of finite elasticity. However, since time integration schemes are not in the focus of this thesis, we neglect this fact and, for the sake of simplicity, use the Newmark scheme to solve transient finite deformation problems. For more sophisticated approaches in this regard see, e. g., Gonzalez [76].

3.4.2 Stabilisation in the Small Time Step Limit

For the case of the Stokes equation, Bochev et al. [28] make the crucial observation that, when using equal-order element pairs stabilised by residual-based Galerkin/Least-Squares or penalty pressure-Poisson methods, *the pressure approximation becomes unstable when the time step gets too small*. We will briefly sketch the authors' line of arguments and adapt it for the mixed $\boldsymbol{u}/p$ formulation in linearised elasticity.

In order to maintain the consistency property, the residual based stabilisation methods GLS (see equation (3.102) on page 153) and SGLS (see equation (3.104) on page 155) have to incorporate

the accelerations $\ddot{\boldsymbol{u}}^h$ into the residual stabilisation terms. Reduction to the inconsistent PPP method (see equation (3.106) on page 155) then results in the following stabilisation terms:

$$c_{\mathsf{PPP}}(\ddot{\boldsymbol{u}}^h, p^h; q) = -\frac{\alpha}{2\mu} \sum_e h_e^2 \Big(\ddot{\boldsymbol{u}}^h + \boldsymbol{\nabla} p^h, \boldsymbol{\nabla} q \Big)_e . \tag{3.127}$$

The semi-discrete matrix equation (3.123) expands correspondingly,

$$\begin{aligned} \rho \mathbf{M} \ddot{\mathbf{u}}(t) + \mathbf{A}\mathbf{u}(t) + \mathbf{B}\mathbf{p}(t) &= \mathbf{f}(t) \\ -\tilde{\mathbf{B}}^{\mathsf{T}} \ddot{\mathbf{u}}(t) + \mathbf{B}^{\mathsf{T}} \mathbf{u}(t) + (\mathbf{C}^{\mathrm{c}} + \mathbf{C}^{\mathrm{s}}) \mathbf{p}(t) &= \mathbf{0}, \end{aligned} \tag{3.128}$$

where the matrix $\tilde{\mathbf{B}}^{\mathsf{T}}$ results from the time-dependent stabilisation terms in equation (3.127). For simplicity, the material is assumed to be incompressible ($\mathbf{C}^{\mathrm{c}} = 0$), the grid to be regular with $h_e = h$. In this case, we have $\tilde{\mathbf{B}}^{\mathsf{T}} = \delta \mathbf{B}^{\mathsf{T}}$ and $\mathbf{C}^{\mathrm{s}} = -\delta \mathbf{L}$, where $\delta := \frac{\alpha h^2}{2\mu}$ and $\mathbf{L}$ is the discrete Laplacian. Inserting these quantities into equation (3.128) and additionally assuming $\rho = 1$ leads to the simplified system

$$\begin{aligned} \mathbf{M} \ddot{\mathbf{u}}(t) + \mathbf{A}\mathbf{u}(t) + \mathbf{B}\mathbf{p}(t) &= \mathbf{f}(t) \\ -\mathbf{B}^{\mathsf{T}} \ddot{\mathbf{u}}(t) + \tfrac{1}{\delta} \mathbf{B}^{\mathsf{T}} \mathbf{u}(t) - \mathbf{L}\mathbf{p}(t) &= \mathbf{0}, \end{aligned} \tag{3.129}$$

Eliminating $\ddot{\mathbf{u}}(t)$ then yields the semi-discrete pressure equation

$$\big(\mathbf{B}^{\mathsf{T}} \mathbf{M}^{-1} \mathbf{B} - \mathbf{L} \big) \mathbf{p}(t) = \mathbf{B}^{\mathsf{T}} \mathbf{M}^{-1} \mathbf{f}(t) - \big(\tfrac{1}{\delta} \mathbf{B}^{\mathsf{T}} + \mathbf{B}^{\mathsf{T}} \mathbf{M}^{-1} \mathbf{A} \big) \mathbf{u}(t). \tag{3.130}$$

Therein, the operator $\mathbf{B}^{\mathsf{T}} \mathbf{M}^{-1} \mathbf{B}$ can be interpreted as the discretised mixed formulation of the Laplacian. Bochev et al. show that the operator $\mathbf{B}^{\mathsf{T}} \mathbf{M}^{-1} \mathbf{B} - \mathbf{L}$ is unstable in the sense that it is not uniformly (with respect to h) invertible.

Applying the Newmark time discretisation (cf. equation (3.125)) to the simplified semi-discrete system (3.129) leads to the fully discrete problem

$$\begin{pmatrix} \frac{1}{\tau^2 \beta} \mathbf{M} + \mathbf{A} & \mathbf{B} \\ (\frac{1}{\delta} - \frac{1}{\tau^2 \beta}) \mathbf{B}^{\mathsf{T}} & -\mathbf{L} \end{pmatrix} \begin{pmatrix} \mathbf{u}^{(k+1)} \\ \mathbf{p}^{(k+1)} \end{pmatrix} = \mathrm{rhs}, \tag{3.131}$$

where we omit the detailed representation of the right hand side terms. Solving this system includes the solution of the following fully discrete pressure Schur complement equation:

$$\Big(\big(1 - \tfrac{\tau^2 \beta}{\delta}\big) \mathbf{B}^{\mathsf{T}} (\mathbf{M} + \tau^2 \beta \mathbf{A})^{-1} \mathbf{B} - \mathbf{L} \Big) \mathbf{p}^{(k+1)} = \mathrm{rhs}. \tag{3.132}$$

Bochev et al. then show that this pressure operator is stable if $\frac{\tau^2 \beta}{\delta} > \varepsilon$ for some $\varepsilon > 0$, independent of h and τ. If, however, $\frac{\tau^2 \beta}{\delta} \to 0$ as $\tau \to 0$, then the operator (3.132) tends to the semi-discrete pressure operator in equation (3.130) and thus becomes unstable. We refer to the original paper [28] for numerical examples that illustrate how the quality of the pressure

solution can actually deteriorate with decreasing time step sizes. In order to guarantee a stable formulation, the mesh size h and time step size τ have to fulfil the relation

$$\tau > ch, \qquad \text{with } c := \sqrt{\tfrac{\alpha\varepsilon}{2\mu\beta}},\ \varepsilon > 0. \tag{3.133}$$

This constitutes a *severe restriction on the choice of time step and/or mesh sizes:* If the computational grid is fixed or is not allowed to be further refined (e. g., due to storage or compute time constraints), then the time step must not be chosen too small. To put it differently: If the time step *has* to be reduced below the critical bound, then this is only possible when the mesh is refined accordingly, which automatically leads to a significant rise of computational work.[35] Highly dynamic CSM simulations that span a very short time interval only (like simulations of metal-forming processes or impact tests) and thus automatically require very short time steps, are extremely expensive or might not be practicable at all. *This has to be seen as a critical drawback of the stabilised Q_1/Q_1 discretisation.* Stable finite element pairs like Q_2/P_1 do not exhibit such deficiencies.

[35] The problem is more critical for the elastodynamic equation than for the time-dependent Stokes (or Navier-Stokes) equation: For the latter, the relation between time-step and mesh size is given by $\tau > ch^2$, i. e., compared to constraint (3.133), the time step can be chosen much smaller before instabilities have to be expected.

4 Multilevel Solvers for Compressible Elasticity Problems

In this chapter we describe our strategy to solve solid mechanical problems. The guiding idea is to adapt the solution process in such a way that the powerful concepts of FEAST can be efficiently exploited and transferred to multivariate linear and nonlinear elasticity problems.

In Section 4.1 we motivate our solution approach. In Section 4.2 the fundamental problem of linearised elasticity for compressible materials under small loads is utilised to explain the strategy and its practical realisation in detail. The technique is then applied to nonlinear finite elasticity problems in Section 4.3. We assess our solution approaches with the help of extensive numerical experiments. These are performed by the application FEASTsolid, which is built on top of the basic FEAST library and represents the practical implementation of the described concepts.

4.1 The Basic Strategy: Reduction to Scalar Components

In Chapter 2 we examined FEAST's ScaRC solvers on the basis of the prototypical scalar Poisson equation. Another important example of a scalar equation is the heat equation, in which the unknown quantity is the temperature. Most physical problems, however, can not be expressed in terms of a single scalar equation:

- A d-dimensional elasticity problem (usually, $d = 2, 3$), for example, has to be solved for d unknown scalar quantities, namely the displacements in the d spatial directions.
- Employing the mixed u/p formulation, the pressure enters the equation as additional unknown scalar quantity.
- The resulting equation system resembles that stemming from the standard formulation of the Stokes and Navier-Stokes equations in computational fluid dynamics (CFD), where the d displacements of a solid material point are replaced by the d components of the fluid's velocity in a point of the domain.
- The latter two physical equations can be coupled, leading to the important fluid structure interaction (FSI) problem, exhibiting $2d + 1$ unknown scalar quantities.

- Each of the above examples can be additionally coupled with a thermal component. When, for instance, an elastic body is deformed, this process produces heat (thermoelasticity).
- When using the Lattice Boltzmann method (LBM) to solve CFD problems, one has to deal, for instance in case of the 'D2Q9' model in two dimensions, with nine scalar transport equations.
- In computational electromagnetics (CEM), the quantities of interest are magnetic and electric fields in one or more spatial directions, depending on the specific model used.

These are only few out of many examples to illustrate that most physical problems are expressed in terms of *multivariate* systems of equations. FEAST's data structures and concepts, however, are laid out and optimised for *scalar* equations (see Section 2.1.1 for details in this regard) and can thus not be applied directly. *So, the fundamental question arises how to treat multivariate problems with* FEAST.

It is obvious that the discretisation of these various multivariate equations results in differently structured matrices – even if the underlying discrete mesh is the same. This is due to the fact that the diverse physical quantities within the multivariate equation couple in different ways. The tensor product property of the local submeshes (see Section 2.1.1) still creates some special, a priori known structure in parts of the matrix, but the structure of the entire matrix heavily depends on the physical problem we are dealing with, especially when the degrees of freedom are not sorted component-wise but per mesh node (see Section 4.2.1 and equation (4.3)).

Hence, when FEAST's basic data structures were to be adapted and extended to directly support multivariate systems as a whole, this would have to be redone for each and every physical problem that is to be solved with FEAST. In practice, that would mean that an application programmer who, for example, wants to couple the elasticity problem with heat transfer, would have to modify some of the most basic kernel modules in order to provide efficient implementations of such core routines as matrix vector multiplication or Gauß-Seidel like preconditioners. Under the typical scientific or economic circumstances, the necessary implementational effort is often too great an obstacle, especially in large and well-established software packages. The situation aggravates significantly when taking into account different hardware technologies, e. g., specialised co-processors like GPUs. The efficient programming of such devices demands special knowledge that is usually not shared by all programmers in a company or in a scientific group.

Our goal is to hide such technical details from the application programmer, but – at the same time – to provide him with the full spectrum of FEAST's advanced and specialised discretisation and solution techniques, including processor-efficient numerical linear algebra, full parallelism and support of special hardware technologies. Since FEAST is meant to be a general-purpose finite element and solution toolkit, this requires that we provide solving techniques that have the potential to run 'out of the box' (or, at least, with only slight modifications) for different physical applications.

The key to achieve this is to reduce the solution of multivariate systems to the solution of a series of scalar systems, and – more general – to realise multivariate operators (operations) as a set (series) of scalar operators (operations). Section 4.2 explains in detail how this is realised with the example of the linearised elasticity equation. The concept has two important advantages:

- It facilitates a strict separation of low-level kernel functionalities and high-level application code. The application programmer, simply spoken, builds a multivariate equation system (sticking to a equation-wise ordering scheme, see Section 4.2.1) and merely calls kernel functions to operate on the system. He does not have to care for matrix structures or processor architectures. He automatically profits from ongoing enhancements of low-level kernel functionalities without modifying the high-level application.

- The kernel programmer can concentrate on the scalar equation case and does not have to care for several different physical applications. All improvements, extensions and adaptions of the low-level kernel libraries are readily available to all applications that stick to the proposed concept. Such enhancements can be the adaption or development of basic routines for different or new processor technologies (e. g., the necessary modification of ADITriGS to vector architectures [16] or the use of GPUs as co-processors [70]), or the introduction of a new finite element and the corresponding necessary adaptions to the resulting matrix fill-in pattern.

In Göddeke et al. [74, 75] we demonstrate how FEASTsolid and FEASTflow[1] are accelerated – without any change to the application code – by outsourcing scalar solves to GPU co-processors. In Buijssen et al. [43], we show the good performance of the two applications on a NEC SX-8 supercomputer using vector processors.[2] In the following sections we show in detail using the example of linearised elasticity how the proposed concept can be successfully realised. We do not claim that the strategy is the most efficient one for each and every physical problem; specialised approaches that exploit particular features of the underlying physical equation are usually superior in terms of numerical efficiency. However, such specialised approaches are expensive to develop and maintain (in terms of programming time) and do not automatically profit from low-level code improvements.

[1] FEASTflow is an application built on top of the FEAST library for solving CFD problems. It is developed by Buijssen [44].

[2] http://www.hlrs.de/systems/platforms/nec-sx8/

4.2 Pure Displacement Formulation for Linearised Elasticity

4.2.1 Operator Splitting

To understand how the scalar ScaRC solvers described in Chapter 2 can be exploited to solve multivariate elasticity problems, we rewrite the left hand side of the Lamé equation (3.51a) on page 133 for the two-dimensional case $d = 2$,

$$-2\mu\,\mathbf{div}(\boldsymbol{\varepsilon}) - \lambda\boldsymbol{\nabla}\operatorname{div}(\boldsymbol{u}) = \begin{pmatrix} (2\mu+\lambda)\partial_{11} + \mu\partial_{22} & \mu\partial_{21} + \lambda\partial_{12} \\ \mu\partial_{12} + \lambda\partial_{21} & \mu\partial_{11} + (2\mu+\lambda)\partial_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix},$$

where $\partial_{jk}u_i = \frac{\partial^2 u_i}{\partial x_j \partial x_k}$. The left hand side of the variational formulation (3.54) on page 133 can be split correspondingly (using $\partial_j u_i := \frac{\partial u_i}{\partial x_j}$):

$$\begin{aligned}
& k(\boldsymbol{u},\boldsymbol{v}) \\
= \;& 2\mu\big(\boldsymbol{\varepsilon}(\boldsymbol{u}),\boldsymbol{\varepsilon}(\boldsymbol{v})\big)_\Omega + \lambda\big(\operatorname{div}(\boldsymbol{u}),\operatorname{div}(\boldsymbol{v})\big)_\Omega \\
= \;& \int_\Omega 2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) + \lambda\operatorname{div}(\boldsymbol{u})\operatorname{div}(\boldsymbol{v})\,\mathrm{d}x \\
= \;& \int_\Omega (2\mu+\lambda)\partial_1 u_1\partial_1 v_1 + \mu\partial_2 u_1\partial_2 v_1\,\mathrm{d}x + \int_\Omega \mu\partial_1 u_2\partial_2 v_1 + \lambda\partial_2 u_2\partial_1 v_1\,\mathrm{d}x \\
& + \int_\Omega \mu\partial_2 u_1\partial_1 v_2 + \lambda\partial_1 u_1\partial_2 v_2\,\mathrm{d}x + \int_\Omega \mu\partial_1 u_2\partial_1 v_2 + (2\mu+\lambda)\partial_2 u_2\partial_2 v_2\,\mathrm{d}x \\
= \;& \underbrace{\int_\Omega \left[\left(\begin{smallmatrix} 2\mu+\lambda & 0 \\ 0 & \mu \end{smallmatrix}\right)\boldsymbol{\nabla}u_1\right]\cdot\boldsymbol{\nabla}v_1\,\mathrm{d}x}_{=:k_{11}(u_1,v_1)} + \underbrace{\int_\Omega \mu\partial_1 u_2\partial_2 v_1 + \lambda\partial_2 u_2\partial_1 v_1\,\mathrm{d}x}_{=:k_{12}(u_2,v_1)} \\
& + \underbrace{\int_\Omega \mu\partial_2 u_1\partial_1 v_2 + \lambda\partial_1 u_1\partial_2 v_2\,\mathrm{d}x}_{=:k_{21}(u_1,v_2)} + \underbrace{\int_\Omega \left[\left(\begin{smallmatrix} \mu & 0 \\ 0 & 2\mu+\lambda \end{smallmatrix}\right)\boldsymbol{\nabla}u_2\right]\cdot\boldsymbol{\nabla}v_2\,\mathrm{d}x}_{=:k_{22}(u_2,v_2)}.
\end{aligned} \tag{4.1}$$

Separating the integrals in this manner reveals the bilinear forms k_{11} and k_{22} to be weak forms of anisotropic, elliptic equations which are scalar in u_1 and u_2, respectively. This separation of variables can be transferred to the discrete operators by ordering the degrees of freedom (DOF) *equation wise*, i. e., corresponding to the displacements in x_1- and x_2-direction (also called *displacement decomposition* [22], *separate displacement ordering* [7] or *component decomposition* [113]). Doing so, the linear system $\mathbf{Ku} = \mathbf{f}$ (see equation (3.70) on page 138) exhibits the following block structure:

$$\begin{pmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{pmatrix}. \tag{4.2}$$

The single matrix blocks $\mathbf{K}_{rs} \in \mathbb{R}^{n\times n}$ are given by

$$(\mathbf{K}_{rs})_{ij} = k_{rs}(\phi_j,\phi_i), \qquad i,j = 1,\ldots,n, \quad r,s = 1,2,$$

the (unknown) coefficient vector $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2)^\mathsf{T} \in \mathbb{R}^{2n}$ defines the finite element solution $\boldsymbol{u}^h = \left(u_1^h, u_2^h\right) \in \boldsymbol{V}^h$ via

$$u_s^h(\boldsymbol{x}) = \sum_{i=1}^{n} (\mathbf{u}_s)_i \phi_i(\boldsymbol{x}), \qquad s = 1,2$$

(see equation (3.69) on page 138), and the vector of external loads $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2)^\mathsf{T} \in \mathbb{R}^{2n}$ is given by

$$(\mathbf{f}_r)_i = L_r(\phi_i) \qquad i = 1, \ldots, n, , \quad r = 1,2$$

(see equation (3.86) on page 148), with ϕ_i denoting the Q_1 nodal basis functions (see equation (3.67) on page 138). We stress the usage of this ordering scheme for two reasons. First, it is the fundamental key to realise our strategy of reducing solving processes to the treatment of scalar systems (see Section 4.1). Second, in finite element software it is often more common to order the unknowns *per node*, i. e., the coefficient vector $\hat{\mathbf{u}} \in \mathbb{R}^{2n}$, for instance, is given by

$$\hat{\mathbf{u}}_i = \begin{cases} (\mathbf{u}_1)_{(i+1)/2}, & i \text{ odd}, \\ (\mathbf{u}_2)_{i/2}, & i \text{ even}. \end{cases} \tag{4.3}$$

In the CSM community, where concepts and algorithms are often developed and explained on the basis of a single element, this ordering scheme may be considered to be the more 'natural' one since the local data (per node) is clustered within the global array structures. The well known program FEAP (see Zienkiewicz and Taylor [178, 179]), for instance, uses this scheme.[3] But also in the CFD literature, this kind of ordering is sometimes considered the 'common' one [163]. See Bank et al. [10] for a comparison of the two ordering schemes.

4.2.2 Modular Solution Strategy

The task is now to efficiently and robustly solve the block structured linear equation system (4.2). Since it usually is too large to be treated by a direct solver, iterative solution methods are mandatory. The *basic iteration* to solve the system is a *preconditioned Richardson method*

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \tilde{\mathbf{K}}_\mathsf{B}^{-1}(\mathbf{f} - \mathbf{K}\mathbf{u}^k), \tag{4.4}$$

where $\tilde{\mathbf{K}}_\mathsf{B}$ denotes the preconditioner. (See Section 2.2 for general comments on preconditioning.) Although this iteration scheme is very simple, it suffices to explain the basics of our solution strategy.

We follow a *modular concept* which mainly consists of two building blocks: On the one hand, there is an *outer solver* which acts on the whole system, e. g., the Richardson iteration (4.4). It does not mind the block structure of the system (4.2) and shall therefore be called *block independent*. On the other hand, there is the preconditioner or smoother $\tilde{\mathbf{K}}_\mathsf{B}$, which crucially relies on the block structure of the system (indicated by the index 'B'). Operators of this type

[3] http://www.ce.berkeley.edu/projects/feap/

shall therefore be called *block dependent*, *block preconditioner* or *block smoother*. The term $\tilde{\mathbf{K}}_{\mathsf{B}}$ does usually not define an explicit matrix, but stands for performing certain operations with (parts of) the matrix $\mathbf{K}$. These operations involve the solution of subsystems, such that it is also justified to regard the operator as *inner solver*. This combination of outer and inner solver can be described as a *two-stage* solving procedure.

4.2.3 The Block Independent Outer Solver

The outer iterative solving scheme needs to perform standard operations like matrix vector multiplications, vector additions, scalar products and norm calculations. Although the outer solver is independent of the block structure, such operations *are*, in fact, performed blockwise. We want to illustrate this exemplarily by means of the defect calculation within the basic iteration (4.4):

$$\mathbf{f} - \mathbf{K}\mathbf{u}^k = \begin{pmatrix} \mathbf{f}_1 - \mathbf{K}_{11}\mathbf{u}_1^k - \mathbf{K}_{12}\mathbf{u}_2^k \\ \mathbf{f}_2 - \mathbf{K}_{21}\mathbf{u}_1^k - \mathbf{K}_{22}\mathbf{u}_2^k \end{pmatrix}. \tag{4.5}$$

Listing 4.1 shows the blockwise defect calculation in algorithmic form. One can see that the

```
1  subroutine defectCalc(K, f, u, q, d)
2    !Input:       (q × q) block structured matrix K,
3    !             block structured vectors f and u,
4    !             number of block components q
5    !Output:      defect vector d = f − Ku
6
7    do r = 1,...,q
8      d_r ← f_r
9      do s = 1,...,q
10       d_r ← d_r − K_rs u_s
11     enddo
12   enddo
13 end subroutine defectCalc
```

Listing 4.1: Defect calculation for block structured matrix and vectors.

defect calculation (4.5) for the multivariate system is reduced to four corresponding operations for scalar systems (line 10 in Listing 4.1 for $q = 2$). For these operations highly tuned subroutines exist in FEAST's SparseBandedBLAS library, such that the multivariate routines can directly benefit from them. The special treatment of the block structure is completely hidden from the outer solving scheme and thus from the application programmer who does not want to deal with submatrices and subvectors.

Another crucial advantage is that FEAST's built-in parallel data structures for scalar components / operations is directly inherited without any further complication. The initial macro domain decomposition of the computational domain leads to local representations of vectors and matrices, the corresponding global entities exist only virtually (see Section 2.3). This means, that the block structured system matrix (4.2) consists of four such global virtual matrices $\mathbf{K}_{rs}$,

each of them represented by M local matrices $(\mathbf{K}_{rs})_i, i = 1,\ldots,M$, where M is the number of subdomains.[4]

So, we have two kinds of blocking here which we have to distinguish strictly: On the one hand the *component blocking* that corresponds to the separate displacement ordering, i. e., the splitting of the multivariate operators into two scalar components; on the other hand the *geometric blocking* corresponding to the domain decomposition. In both cases, the matrix blocks result from restricting the total set of degrees of freedom with size $2n$ to subsets of size n for the case of component blocking and of size $O(\frac{n}{M})$ for the case of geometric blocking. In the latter case, the subsets are associated with subsets of grid vertices. An important difference is that component blocking leads to truly disjunct subsets of unknowns, while geometric blocking (as performed in FEAST) does not, due to the common vertices on subdomain boundaries resulting from the concept of minimal overlap (see Section 2.4).

We briefly want to assume that the two blockings are performed *one after the other*.[5] Then there are two possibilities:

1. First perform the geometric blocking (i. e., domain decomposition), then the component blocking (i. e., operator splitting):

$$\begin{aligned} & \mathbf{K} && (4.6a)\\ \rightarrow\ & \mathbf{K}_i, \quad i = 1,\ldots,M && (4.6b)\\ \rightarrow\ & \begin{pmatrix} (\mathbf{K}_{11})_i & (\mathbf{K}_{12})_i \\ (\mathbf{K}_{21})_i & (\mathbf{K}_{22})_i \end{pmatrix}, \quad i = 1,\ldots,M && (4.6c) \end{aligned}$$

2. First perform the component blocking (i. e., operator splitting), then the geometric blocking (i. e., domain decomposition):

$$\begin{aligned} & \mathbf{K} && (4.7a)\\ \rightarrow\ & \begin{pmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{pmatrix} && (4.7b)\\ \rightarrow\ & \begin{pmatrix} (\mathbf{K}_{11})_i & (\mathbf{K}_{12})_i \\ (\mathbf{K}_{21})_i & (\mathbf{K}_{22})_i \end{pmatrix}, \quad i = 1,\ldots,M && (4.7c) \end{aligned}$$

The initial and final states are in both cases the same, only the intermediate states differ. It is important to know that *in our solution concept we only use the second variant*. This means, *on a local subdomain we never consider the whole multivariate system* (as in equation (4.6b)); before restricting the total set of DOF geometrically, it must be divided corresponding to its scalar components (as in equation (4.7b)). We will get back to this issue in Section 4.2.4.

[4] We want to point out not to confuse the two similar notations $(\mathbf{K}_{rs})_i$ and $(\mathbf{K}_{rs})_{ij}$. The former refers to the M local matrices the (virtual) global matrix $\mathbf{K}_{rs}$ consists of, while the latter refers to the $n \times n$ entries of $\mathbf{K}_{rs}$.

[5] Actually, there is no order in this sense. The only existing matrix realisation (in terms of data structures) is the final one (see equation (4.6c) or (4.7c)).

To accelerate the basic iteration (4.4) for solving the block structured system (4.2), we use *Krylov space methods*. In order to preserve the flexibility of our modular solver concept it must be possible to apply unsymmetric and/or variable preconditioners (see Section 4.2.4), hence, the basic CG method is not an option as outer solver. Corresponding to the scalar case described in Section 2.6.3.4 we use BiCGstab instead. The main drawback of (not properly preconditioned) BiCGstab (and every other Krylov space method), however, is the dependence of the convergence rate on the grid size parameter h; halving h roughly doubles the number of iterations. Such behaviour clearly renders the simple Krylov methods unsuitable for large scale problems. Therefore, we also consider multigrid (MG) methods to solve the block structured system (4.2). Since these methods will employ FEAST's ScaRC solvers to treat scalar subsystems (see Section 4.2.4) and thus automatically make use of the underlying minimally overlapping grid partitioning, they can also be considered as multilevel domain decomposition (MLDD) methods.

However, in Section 4.2.4 we also show that Krylov methods for *multivariate* systems can show level independent behaviour if they are properly preconditioned by *scalar* multilevel methods. Consequently, the questions arise: Do we really need a multilevel solver for the multivariate system? If so, are the (scalar) ScaRC solvers – involving multilevel schemes themselves – still necessary? Or can they be replaced by simpler smoothers? In Section 4.2.7 it will be shown that there actually *are* situations where robust solving behaviour can only be achieved if we use *both* stages of multilevel methods.

The basic principles of multigrid and multilevel domain decomposition methods have already been described in Chapter 2, so we only make some additional comments for the multivariate case here:

- *Restriction* and *prolongation* routines can be directly taken from the FEAST library, due to the separate displacement ordering. A defect vector $\mathbf{d} = (\mathbf{d}_1, \mathbf{d}_2)^\mathsf{T}$, for instance, is restricted to the coarser level by simply restricting its component vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ – both corresponding to scalar equations – one after the other. Hidden in a proper routine, the block independent multigrid algorithm only deals with the global vector $\mathbf{d}$ and never 'sees' the single blocks $\mathbf{d}_i$.

- Also the *direct coarse grid solver* can be realised with only small effort. Due to the separate displacement ordering, the four coarse grid (level 1) matrix blocks $\mathbf{K}_{rs}^{(1)}$ (corresponding to scalar operators, see equation (4.2)) can be assembled using the standard FEAST routines which collect and prepare the necessary data from all subdomains and all processors. So, the only task is to assemble these four blocks according to the UMFPACK data structures [55].

- The *smoothing* operation algorithmically resembles the procedure of preconditioning a Krylov method, i. e., we can in principle use the same block dependent inner solvers $\tilde{\mathbf{K}}_\mathsf{B}$ for smoothing (see Section 4.2.4). In Section 4.2.7 we examine if this approach is efficient.

We described the combination of outer and inner solvers as a *two-stage* solution process. Actually, our modular strategy goes one step further: The outer solver can itself act as preconditioner/smoother within another iteration scheme.[6] In Section 2.6.3 we explained the possible advantages of combining multigrid and Krylov methods. Also for multivariate system solvers, the user can define a *three-stage* procedure like

> 'Use BiCGstab as solver, precondition it by performing one multigrid V-cycle which is smoothed by $\tilde{\mathbf{K}}_{\mathsf{B}}$',

or a *four-stage* procedure like

> 'Use BiCGstab as solver, precondition it by performing one multigrid F-cycle which is smoothed by one iteration of FGMRes preconditioned by $\tilde{\mathbf{K}}_{\mathsf{B}}$'.

Our implementation allows for an arbitrary nesting of iterative schemes ('modules'). This facilitates the development of robust solution strategies, that can be flexibly adjusted to the specific problem at hand. In our tests, however, we will only use BiCGstab as outer and inner Krylov solver/smoother, respectively; comparative tests with FGMRes as inner Krylov smoother are not performed.

4.2.4 The Block Dependent Preconditioner/Smoother

Our aim is to find efficient preconditioners $\tilde{\mathbf{K}}_{\mathsf{B}}$ that make use of FEAST's ScaRC solvers. The basic problem is that all data structures and solution concepts in FEAST are laid out for *scalar* equations. Only then, the resulting matrices have the specific 9-band structure the SparseBandedBLAS library depends on (see Section 2.1.1). The matrix $\mathbf{K}$ stemming from the 2D elasticity problem clearly lacks this structure, preventing a direct application of the ScaRC solvers. That is why the separate displacement ordering (4.1) has to be applied: The matrices $\mathbf{K}_{11}$ and $\mathbf{K}_{22}$ of the resulting block structured system (4.2) correspond to scalar elliptic anisotropic equations and are thus perfectly suited to be treated by the ScaRC solvers. In order to exploit this, we introduce the splitting $\mathbf{K} = \mathbf{D} + \mathbf{L} + \mathbf{U}$, where

$$\mathbf{D} := \begin{pmatrix} \mathbf{K}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_{22} \end{pmatrix}, \qquad \mathbf{L} := \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{K}_{21} & \mathbf{0} \end{pmatrix}, \qquad \mathbf{U} := \begin{pmatrix} \mathbf{0} & \mathbf{K}_{12} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{4.8}$$

and define the **block Jacobi preconditioner**

$$\tilde{\mathbf{K}}_{\mathsf{BJac}} := \mathbf{D} = \begin{pmatrix} \mathbf{K}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_{22} \end{pmatrix}. \tag{4.9}$$

Applying this preconditioner, which was first described already 30 years ago by Axelsson and Gustafsson [8], means to solve the two separated equation systems

$$\mathbf{K}_{11}\mathbf{c}_1 = \mathbf{r}_1 \qquad \text{and} \qquad \mathbf{K}_{22}\mathbf{c}_2 = \mathbf{r}_2, \tag{4.10}$$

[6] Consequently, the solver can be considered as inner *and* outer solver at the same time – depending on the point of view.

where $(\mathbf{r}_1, \mathbf{r}_2)^\mathsf{T} = \mathbf{r} = \mathbf{f} - \mathbf{K}\mathbf{u}^k$ denotes the current residual and $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)^\mathsf{T}$ the correction vector for updating the iterate $\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{c}$ (see basic iteration (4.4)).

Blaheta [22] shows that $\mathbf{K}$ and the block diagonal part $\tilde{\mathbf{K}}_{\mathsf{BJac}}$ are spectrally equivalent and that the condition number of the preconditioned system matrix can be bounded from above via

$$\kappa(\tilde{\mathbf{K}}_{\mathsf{BJac}}^{-1}\mathbf{K}) \leqslant \frac{1}{c_\mathrm{K}}\Big(1 + \frac{1}{1-2\nu}\Big) = \frac{1}{c_\mathrm{K}}\Big(\frac{2\mu+\lambda}{\mu}\Big), \tag{4.11}$$

where $c_\mathrm{K} = c_\mathrm{K}(\Omega, \Gamma_\mathrm{D})$ is the Korn constant (see equation (3.73) on page 140) and ν is the Poisson ratio (see equation (3.27) on page 122). We want to draw three conclusions from this estimate:

- *The condition number κ does not depend on the mesh size parameter h.*
  Consequently, the convergence of the preconditioned solver is independent of the mesh refinement level. When, additionally, the scalar systems (4.10) can be solved with a convergence rate that is independent of h, this property is valid for the overall solution algorithm, which is one of the key requirements for solving large scale problems.

- *The condition number depends on Korn's constant.*
  Consequently, the number of iterations of the outer solver is influenced by the domain shape and the boundary conditions. This is especially true for structures that are *thin* in one direction and fixed only on a small portion of the boundary. Ovtchinnikov and Xanthis [121] introduce variations of Korn's inequality for thin domains to overcome this problem. The theoretical insight is used to modify an iterative solution scheme which is then shown to robustly deal with thin structures. However, these modifications are very specialised and, thus, unsuitable within our desired 'black box' solver concept. It may be a topic for future work to adapt these ideas, but for the time being, we tackle the problem solely by applying a robust combination of our standard solution modules. In Section 4.2.7.3 we show that the influence of Korn's constant can be, at least, considerably weakened this way.

- *The condition number depends on the Poisson ratio ν.*
  When this constant tends to 0.5, i. e., when the material becomes less compressible, κ tends to infinity, resulting in extremely ill-conditioned systems. In contrast to the previous point, this deficiency can not be mitigated by using more efficient solving methods, but only by using a more sophisticated discretisation technique (see Sections 3.2.2.2 and 3.2.3).

We will thoroughly illustrate these conclusions in Section 4.2.7.

Instead of the block Jacobi preconditioner (4.9), a Gauß-Seidel variant

$$\tilde{\mathbf{K}}_{\mathsf{BGS}} := \mathbf{D} + \mathbf{L} = \begin{pmatrix} \mathbf{K}_{11} & \mathbf{0} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{pmatrix} \tag{4.12}$$

can be used, as well. With $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2)^T$ denoting the current residual, the application of this preconditioner means to *subsequently* solve the two systems

$$\mathbf{K}_{11}\mathbf{c}_1 = \mathbf{r}_1 \qquad \text{and} \qquad \mathbf{K}_{22}\mathbf{c}_2 = \mathbf{r}_2 - \mathbf{K}_{21}\mathbf{c}_1. \tag{4.13}$$

So, the only difference compared to the application of the block Jacobi preconditioner (4.10) is that the right hand side for the second system solve is updated with the solution of the first one. To our knowledge an estimate corresponding to equation (4.11) is not available, but numerical examinations show that the block Gauß-Seidel version usually results in fewer outer iterations [113]. We confirm this in Section 4.2.7. However, the application of the block Gauß-Seidel preconditioner is also more expensive due to the additional matrix vector multiplication. So, we have to examine if it is still superior when total efficiency is measured. A disadvantage of the block Gauß-Seidel preconditioner is that it is unsymmetric; iterative methods like CG, for instance, are not applicable anymore as outer solver (see Section 4.2.3). Of course, one can resolve this by defining a symmetrised variant of block Gauß-Seidel, but this increases the arithmetic costs significantly. Furthermore, the preconditioner is still unsymmetric if one of the solvers for the scalar subsystems (4.10) is unsymmetric. For these reasons, we do not consider a symmetrised variant.

Another important issue is that for the block Jacobi preconditioner the two systems (4.10) could be solved *in parallel* which is not possible for the block Gauß-Seidel preconditioner (4.13). FEAST's parallelisation concept, however, is strictly based on domain decomposition. Enabling such an additional 'parallelisation layer' would require non-trivial extensions of FEAST's core communication routines, which is clearly possible, though, but is not part of this thesis.

In special situations the application of the block Gauß-Seidel preconditioner is problematic: Let us assume that a material body is only loaded in vertical direction, i. e., $\mathbf{f}_1 = \mathbf{0}$ (which is a typical loading situation). Let us further assume that we use a standard implementation of BiCGstab (see Barrett et al. [13] and Listing 4.2) with zero start vector $\mathbf{u} = \mathbf{0}$ and block Gauß-Seidel preconditioning where the scalar blocks are solved exactly. Then the following happens: In the first iteration, we have

$$\mathbf{r} = \tilde{\mathbf{r}} = \mathbf{p} = (\mathbf{0}, \mathbf{f}_2)^T, \quad \rho = \|\mathbf{f}_2\|^2$$

(Listing 4.2, lines 1, 2, 4 and 7) such that the application of the block Gauß-Seidel preconditioner (line 13) yields

$$\mathbf{c} = (\mathbf{0}, \mathbf{K}_{22}^{-1}\mathbf{f}_2)^T,$$

leading to

$$\mathbf{v} = (\mathbf{K}_{12}\mathbf{K}_{22}^{-1}\mathbf{f}_2, \mathbf{f}_2)^T, \quad \alpha = \rho/\|\mathbf{f}_2\|^2 = 1$$

(lines 14 and 15). The updated residual vector (line 17) is then given by

$$\mathbf{r} = (-\mathbf{v}_1, \mathbf{0})^T.$$

The second preconditioning step (line 19) yields

$$\mathbf{c}_1 = \mathbf{K}_{11}^{-1}\mathbf{r}_1, \quad \mathbf{c}_2 = -\mathbf{K}_{22}^{-1}\mathbf{K}_{21}\mathbf{c}_1,$$

leading to

$$\mathbf{t} = (\star, \mathbf{K}_{21}\mathbf{c}_1 - \mathbf{K}_{22}\mathbf{K}_{22}^{-1}\mathbf{K}_{21}\mathbf{c}_1)^{\mathsf{T}} = (\star, \mathbf{0})^{\mathsf{T}} \tag{4.14}$$

(line 20) with $\mathbf{t}_1 = \star$ denoting some nonzero value. Consequently, also the second component of the final residual vector (line 23) is zero, $\mathbf{r} = (\star, \mathbf{0})^{\mathsf{T}}$. So, we have $\rho = 0$ in the second iteration (line 5) causing the algorithm to fail.

```
1  r ← f − Ku,   δ ← ‖r‖      ! initial residual and its norm for some start vector u
2  r̃ ← r
3  do i = 1,2,3,...                ! begin iteration
4    ρ ← r̃ᵀr
5    if (ρ .eq. 0) method fails
6    if (i .eq. 1) then
7      p ← r
8    else
9      β ← ρα/(ρ_prev ω)
10     p ← r + β(p − ωv)
11   endif
12   ρ_prev = ρ
13   solve K̃c = p                ! first preconditioning step
14   v ← Kc
15   α ← ρ/r̃ᵀv
16   u ← u + αc                  ! update solution
17   r ← r − αv                  ! update residual
18   if (‖r‖ ⩽ δε) exit          ! 'early exit'
19   solve K̃c = r                ! second preconditioning step
20   t ← Kc
21   ω ← tᵀr/tᵀt
22   u ← u + ωc                  ! update solution
23   r ← r − ωt                  ! update residual
24   if (‖r‖ ⩽ δε) then
25     exit
26   endif
27 enddo
```

Listing 4.2: Preconditioned BiCGstab iteration to solve the system $\mathbf{Ku} = \mathbf{f}$ with preconditioner matrix $\tilde{\mathbf{K}}$

In practice, the value will not be accurately zero, but the more exactly the scalar systems are solved in the block Gauß-Seidel preconditioner the closer to zero the value will be. This can lead to instabilities and to stagnation of the convergence process. This problem could, for example, be avoided by using a nonzero start vector $\mathbf{u}$. However, the BiCGstab algorithm is often used in the context of an outer defect correction, where the zero start vector is usually the best choice. Another possibility would be to choose the vector $\tilde{\mathbf{r}}$ differently. We tried this, but found that the problem still occurred in some situations. A third possibility would be to apply at least two block Gauß-Seidel iterations for preconditioning. This, however, would increase the computational costs significantly.

We solve the problem by using block SOR (successive overrelaxation) instead of block Gauß-Seidel. The block SOR preconditioner is given by

$$\tilde{\mathbf{K}}_{\mathsf{BSOR}} := \frac{1}{\delta}(\mathbf{D}+\delta\mathbf{L}) = \frac{1}{\delta}\begin{pmatrix}\mathbf{K}_{11} & \mathbf{0}\\ \delta\mathbf{K}_{21} & \mathbf{K}_{22}\end{pmatrix}, \tag{4.15}$$

where δ is the relaxation parameter. With $\delta = 1$ block SOR reduces to block Gauß-Seidel. The block SOR preconditioner is applied by solving the two systems

$$\mathbf{K}_{11}\mathbf{c}_1 = \mathbf{r}_1 \quad \text{and} \quad \mathbf{K}_{22}\mathbf{c}_2 = \mathbf{r}_2 - \delta\mathbf{K}_{21}\mathbf{c}_1 \tag{4.16}$$

and finally scale the correction vector, $\mathbf{c} \leftarrow \delta\mathbf{c}$. This final scaling can be omitted when the solver is scale invariant (which is the case for BiCGstab; cf. lines 14–16). When applying block SOR in the example above, equation (4.14) is replaced by

$$\mathbf{t} = (\star, \mathbf{K}_{21}\mathbf{c}_1 - \delta\mathbf{K}_{21}\mathbf{c}_1)^{\mathsf{T}},$$

i. e., with $\delta \neq 1$ the second component of $\mathbf{t}$ is guaranteed to be nonzero, avoiding the breakdown of the algorithm in the second iteration.

In all the following numerical tests, we use the relaxation parameter $\delta = 1.1$. On the one hand, this solves the special problem with BiCGstab described above, and on the other hand it generally leads to – partially significant – improvements of the convergence. Only in very rare cases, the convergence is marginally worse than for $\delta = 1.0$.

Axelsson [7] discusses the problem that the solver which is used to treat the scalar systems (see equation (4.10)) may suffer from the anisotropy in the operators $k_{ii}, i = 1,2$, (see equation (4.1) on page 186). This operator anisotropy depends on the value of ν (respectively, λ) and is given by

$$a_{\text{op}} := \frac{2\mu+\lambda}{\mu} = 1 + \frac{1}{1-2\nu}. \tag{4.17}$$

Hence, the operator anisotropy $a_{\text{op}} \in (2,\infty)$ increases with the material becoming more incompressible ($\nu \to 0.5$). Table 4.1 lists the corresponding values for the example materials from Table 3.1 on page 127. Axelsson suggests to circumvent the operator anisotropy by replacing

	Steel/Iron	Aluminium	Lead	Rubber-like			
ν	0.29	0.35	0.44	0.49	0.499	0.49999	0.5
a_{op}	3.4	4.3	9.3	51	501	50001	∞

Table 4.1: Operator anisotropies for the materials listed in Table 3.1 on page 127.

the anisotropic weak forms k_{ii} by those corresponding to the isotropic Laplace operator (also see Axelsson and Gustafsson [8] and Blaheta [22]),

$$\hat{k}_{ii}(u_i, v_i) = (\boldsymbol{\nabla} u_i, \boldsymbol{\nabla} v_i)_\Omega.$$

The solution of the corresponding subsystems is significantly easier then, i. e., one step of the isotropic Laplace preconditioner is usually less expensive than one step of the anisotropic preconditioner that is based on the anisotropic forms k_{ii}. However, it also exhibits worse preconditioning abilities, especially for values of ν closer to 0.5 for which the forms k_{ii} and $\hat{k}_{ii}$ differ more clearly. The estimate of the condition number of the isotropically preconditioned system is similar to the estimate (4.11), though, but numerical experiments indeed show that the isotropic preconditioner leads to larger outer iteration numbers [22]. Another disadvantage is that the two corresponding Laplace matrices would additionally have to be kept in memory, resulting in a significant increase of storage requirements. In addition to operator anisotropies, the problem of mesh anisotropies arises in a simulation. This problem, however, is a fundamental one which also affects the isotropic Laplace preconditioner.

We want to emphasise that FEAST's scalar solvers are perfectly suited to evade exactly these issues: They are capable to robustly deal with both operator and mesh anisotropies (see Chapter 2 and Section 4.2.7).[7] Consequently, it is not necessary to switch to the Laplace preconditioner which would result in additional storage requirements and loss of numerical efficiency. Instead, by using the powerful ScaRC solvers, we can safely and efficiently exploit the capabilities of the anisotropic block preconditioner.

4.2.5 Extending the 2-layer-ScaRC Concept

When applying the block Jacobi or block Gauß-Seidel preconditioner the basic question arises how accurately to solve the scalar systems (4.10) and (4.13). Solving them exactly usually results in the best possible convergence rate of the outer method, though, but in practice, this is much too expensive in terms of arithmetic work. Instead, a quite rough inner solution (e. g., gaining only one digit) is often a good compromise between amount of inner work and number of outer iterations. We will examine this issue in Section 4.2.7.

In Chapter 2 we distinguished 1-layer-ScaRC and 2-layer-ScaRC solvers. The term 'layer' is motivated geometrically: A solver can be defined on the whole domain (global layer), or on a subdomain (local layer). A 2-layer-ScaRC solver consists of (at least) two nested solvers (usually multigrid), one acting on the global layer and one on the local layer. We emphasised the importance of keeping the number of global iterations as small as possible in order to minimise the amount of communication on parallel computers. The 2-layer-ScaRC concept tries to achieve this goal by 'shifting' global work to local subdomain solves. This, however, does not apply to the case, when balancing the number of outer iterations of a multivariate method to solve the block structured system (4.2) and the accuracy of the scalar inner solves (4.10). The reason is that also the scalar solves act on the *global* layer and have to communicate. Hence, 'shifting' as much work as possible to the scalar solves does usually not decrease the amount of communication.

[7] This does not eliminate the underlying deficiencies of the overall iterative method for nearly incompressible material. Its convergence will clearly deteriorate with increasing values of λ, even if the blocks on the diagonal are efficiently treated by ScaRC.

For this reason we want to introduce a variant of the block preconditioners that can be viewed as extension of the scalar 2-layer-ScaRC concept to the multivariate case. Instead of the matrices $\mathbf{K}_{11}$ and $\mathbf{K}_{22}$ in the block Jacobi preconditioner (4.9), we use additive Schwarz preconditioners as defined in equation (2.9) on page 31, i. e., with M denoting the number of subdomains:

$$(\tilde{\mathbf{K}}_{rr}^{\mathrm{AS}})^{-1} := \widetilde{\sum_{i=1}^{M}} \mathbf{P}_i \big((\mathbf{K}_{rr})_i \big)^{-1} \mathbf{R}_i, \quad r = 1,2.$$

(For an explanation of the restriction and prolongation operators $\mathbf{R}_i$ and $\mathbf{P}_i$ and of the special sum symbol, see Sections 2.4 and 2.5.2.) Hence, the two *global* scalar system solves (4.10) are replaced by two sweeps of M *local* scalar system solves:

$$\begin{aligned} &\text{solve } (\mathbf{K}_{11})_i \mathbf{x}_i = \mathbf{R}_i \mathbf{r}_1 \text{ for } i = 1,\dots,M, \qquad && \mathbf{c}_1 \leftarrow \widetilde{\sum_{i=1}^{M}} \mathbf{P}_i \mathbf{x}_i, \\ &\text{solve } (\mathbf{K}_{22})_i \mathbf{x}_i = \mathbf{R}_i \mathbf{r}_2 \text{ for } i = 1,\dots,M, \qquad && \mathbf{c}_2 \leftarrow \widetilde{\sum_{i=1}^{M}} \mathbf{P}_i \mathbf{x}_i. \end{aligned} \tag{4.18}$$

The auxiliary vectors $\mathbf{x}_i$ have only been introduced for sake of clarity and can actually be identified with $\mathbf{R}_i \mathbf{c}_1$ and $\mathbf{R}_i \mathbf{c}_2$, respectively. The resulting **block Jacobi additive Schwarz preconditioner** can be represented in matrix form as

$$\tilde{\mathbf{K}}_{\mathsf{BJacAS}} := \begin{pmatrix} \tilde{\mathbf{K}}_{11}^{\mathrm{AS}} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{K}}_{22}^{\mathrm{AS}} \end{pmatrix}. \tag{4.19}$$

When the outer multivariate solver is a multigrid method and also the scalar local systems (4.18) are solved by multigrid, then the resulting solution scheme can indeed be regarded as extension of the scalar 2-layer-ScaRC solver: The only difference is that the outer multigrid scheme acts on the multivariate system and that two local solves per subdomain are performed.

Correspondingly, we can define the **block Gauß-Seidel additive Schwarz preconditioner**

$$\tilde{\mathbf{K}}_{\mathsf{BGSAS}} := \begin{pmatrix} \tilde{\mathbf{K}}_{11}^{\mathrm{AS}} & \mathbf{0} \\ \mathbf{K}_{21} & \tilde{\mathbf{K}}_{22}^{\mathrm{AS}} \end{pmatrix}. \tag{4.20}$$

This preconditioner is applied as follows:

$$\begin{aligned} &\text{solve } (\mathbf{K}_{11})_i \mathbf{x}_i = \mathbf{R}_i \mathbf{r}_1 \text{ for } i = 1,\dots,M, \qquad && \mathbf{c}_1 \leftarrow \widetilde{\sum_{i=1}^{M}} \mathbf{P}_i \mathbf{x}_i, \\ &\text{solve } (\mathbf{K}_{22})_i \mathbf{x}_i = \mathbf{R}_i (\mathbf{r}_2 - \mathbf{K_{21}} \mathbf{c}_1) \text{ for } i = 1,\dots,M, \qquad && \mathbf{c}_2 \leftarrow \widetilde{\sum_{i=1}^{M}} \mathbf{P}_i \mathbf{x}_i. \end{aligned} \tag{4.21}$$

It differs from the block Jacobi additive Schwarz preconditioner (4.18) only in the second step. Additionally, we use a block SOR additive Schwarz preconditioner, which is defined analogously to the block Gauß-Seidel additive Schwarz preconditioner (4.20).

The advantage of the block additive Schwarz preconditioners $\tilde{\mathbf{K}}_{\mathsf{BJacAS}}$, $\tilde{\mathbf{K}}_{\mathsf{BGSAS}}$, and $\tilde{\mathbf{K}}_{\mathsf{BSorAS}}$ is clearly the comparatively small amount of communication. On the other hand, it is also obvious that such a preconditioning step is only a very rough approximation to the actual solution of the corresponding scalar system. Hence, the preconditioning qualities are worse than those of the standard block preconditioners $\tilde{\mathbf{K}}_{\mathsf{BJac}}$ and $\tilde{\mathbf{K}}_{\mathsf{BGS}}$. We will examine this issue in Section 4.2.7.

4.2.6 Solver Variants

The various ways to combine individual solver components (global / local, multivariate / scalar) result in several different, possibly quite complex solution algorithms. We now describe those variants which are used and compared in Section 4.2.7. We introduce a short notation that extends that of Section 2.5.5.2.

We only consider *global*[8] multivariate solvers: As explained in Section 4.2.3 we never treat the entire multivariate system restricted to one subdomain (as in equation (4.6b) on page 189). In other words, before restricting the set of unknowns to a subdomain, it first has to be restricted to one scalar component (as in equation (4.7)). A prerequisite for the solvers in the following enumeration is to contain at least *one* multigrid scheme (local, global, scalar or multivariate). Only then the solver has the chance to show convergence rates that are independent of the grid refinement level.

1. **Multivariate solver: block preconditioned singlegrid Krylov method**
   Scalar solver: global 1-layer-ScaRC or 2-layer-ScaRC
   In this variant the multivariate system is treated with a singlegrid Krylov solver, while the scalar systems within the preconditioner are solved by a global multigrid scheme (1-layer-ScaRC or 2-layer-ScaRC). The short notation of an exemplary solver from this category is

   ```
   BICG-BJac[MG__MG-ADI].
   ```

   A BiCGstab solver is preconditioned by block Jacobi, and the scalar systems are solved by a 2-layer-ScaRC solver with ADITriGS as elementary smoother. The solver for the scalar systems is written in square brackets directly after the notation of the block preconditioner. More details about the solver components can again be added in parentheses (see Section 2.5.5.2), e. g.,

   ```
   BICG(128,1e-6)-BJac[MG(1,V11)__MG(1e-1,V22)-ADI].
   ```

2. **Multivariate solver: block preconditioned singlegrid Krylov method**
   Scalar solver: additive Schwarz preconditioner with local multigrid
   This variant differs from the previous variant in that it uses a block Jacobi or Gauß-Seidel Schwarz preconditioner. In short notation, such a solver is exemplarily written as

   ```
   BICG-BJacAS[MG-ADI].
   ```

   The scalar solver in square brackets must be a local solver. In the example it is a multigrid method with ADITriGS as smoother.

3. **Multivariate solver: block smoothed multigrid method**
   Scalar solver: global 1-layer-ScaRC or 2-layer-ScaRC

[8] To avoid confusion we want to note that the term 'global' is strictly used in the geometric sense, and *not* to distinguish multivariate from scalar solvers. Both multivariate and scalar solvers can be global, i. e., act on the whole domain.

In this variant the multivariate system is treated by a multigrid solver. This means that the block preconditioner is called on each level. Examples of such solvers are

```
MG-BJac[MG__ADI],
MG-BJac[MG__MG-ADI].
```

In the first case, the scalar solver is a 1-layer-ScaRC scheme, which means that the overall solver consists of two nested global multigrid schemes. In the second case, the scalar solver is a 2-layer-ScaRC scheme, i. e., the overall solver comprises three multigrid schemes, two global ones and a local one.

4. **Multivariate solver: block smoothed multigrid method**
   Scalar solver: additive Schwarz preconditioner with local multigrid
   This variant is the natural extension of the 2-layer-ScaRC concept to multivariate systems as described in Section 4.2.5. Such a solver is exemplarily given by

   ```
   MG-BJacAS[MG-ADI].
   ```

   The global multivariate multigrid uses a block Jacobi additive Schwarz smoother, which treats the local systems with a multigrid method using ADITriGS as elementary smoother.

5. **Multivariate solver: block smoothed multigrid method**
   Scalar solver: local elementary smoother
   The difference of this variant to the previous one is that the local systems within the block additive Schwarz smoother are not solved by a local multigrid. Instead, a simple elementary smoother is applied, as for example ADITriGS:

   ```
   MG-BJacAS[ADI].
   ```

   This solver can be viewed as the multivariate extension of a 1-layer-ScaRC solver.

6. **Multivariate solver: multigrid method with pointwise Jacobi smoother**
   Scalar solver: -
   For sake of comparison we also consider a multivariate multigrid with a standard pointwise Jacobi smoother,

   ```
   MG-Jac,
   ```

   for which no scalar solves are performed. This solver completely ignores the block structure of the multivariate system. It can be regarded as default multigrid solver as it is used by other software packages.

The specific solver examples in the list are just representatives for the respective category. In practice they will partially be enhanced by adding Krylov schemes as outer solver or as smoother (cf. Section 2.6.3). The file format for multivariate solvers is similar to that of the

scalar ScaRC solvers (see Section 2.5.5.1), so we omit a detailed explanation here.[9] We will use the short notation introduced above to define and distinguish different solvers.

4.2.7 Numerical Tests

In this section we thoroughly investigate the block preconditioning technique with respect to various aspects. We examine and compare the different solver approaches described in Section 4.2.6. At first, we concentrate on the block preconditioning concept itself (Sections 4.2.7.1, 4.2.7.2 and 4.2.7.3), i. e., we solve all (local and global) scalar subsystems exactly in order to suppress the influence of approximate system solves. In these tests we are only interested in outer iteration numbers. Then, in Section 4.2.7.4 we examine inexact subsystem solves and measure the *total arithmetic efficiency* and the *total runtime efficiency* of the various solution strategies. In Section 4.2.7.5 we compare 1-layer-ScaRC and 2-layer-ScaRC solvers and investigate whether the possible superiority of the latter class, that we found for scalar systems, can be transferred to multivariate elasticity systems. Finally, we demonstrate the good parallel efficiency of the overall solution scheme by means of some large scale multiprocessor simulations. All tests are performed for several refinement levels, such that also the question whether the convergence rates are independent of the grid refinement can be examined.

4.2.7.1 Dependence on Compressibility

In this section we examine how the solvers react to an increasing degree of incompressibility. We consider the BLOCK configuration depicted in Figure 4.1 (taken from Reese et al. [133]), using materials with shear modulus $\mu = 8.2$ MPa and Poisson ratios ranging between $\nu = 0.4$ and $\nu = 0.499$. The block is vertically loaded by a line force of -10 MPa. Since the various

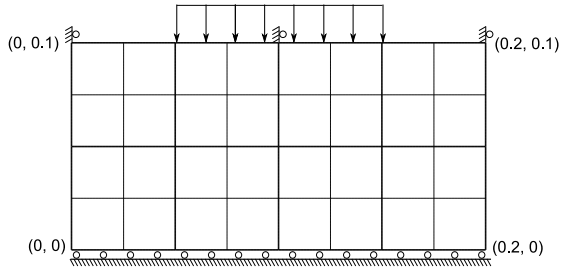


Figure 4.1: BLOCK configuration consisting of 8 subdomains, level 1 depicted. The bottom side is fixed in vertical direction, the top side is fixed in horizontal direction.

[9] Currently, scalar and multivariate solvers are strictly separated in the FEAST code; there are, for instance, two separated implementations of BiCGstab. This has to be seen as intermediate stage in the code development: In future versions of FEAST the two modules will be combined, and scalar solvers will simply be considered as 1×1 block systems. Consequently, scalar and multivariate solvers are currently defined by two different solver files.

solvers do different amounts of work per iteration, it does not make sense to compare iteration numbers. Instead, we count how often the block preconditioner/smoother is called. A BiCGstab solver, for example, performs two preconditioning steps per iteration while a multigrid scheme with two pre- and two postsmoothing steps calls the block smoother four times per iteration. When the solver does not achieve the relative tolerance of $\varepsilon = 10^{-6}$ within 256 block preconditioner/smoother calls, the corresponding bar is labelled by '>>' in the following bar charts; when the solver diverges, the bar is omitted completely.

We now list the solvers to be compared in the following tests. When the damping parameter is 1.0, it is omitted in the notation. If two solvers are nested (e. g., `BiCG-MG`), the inner solver always performs exactly one iteration, which is also omitted in the short notation. All solvers use either block Jacobi or block SOR (additive Schwarz) preconditioning/smoothing.

- `BiCG-BJac, BiCG-BSor`
  BiCGstab solver with block preconditioner (2 preconditioner calls per iteration (PCPI))
- `BiCG-BJacAS, BiCG-BSorAS`
  BiCGstab solver with block additive Schwarz preconditioner (2 PCPI)
- `MG(V22)-BJac, MG(V22)-BSor`
  V-cycle multigrid solver with block smoother (4 PCPI)
- `MG(V22,0.8)-BJacAS, MG(V22,0.8)-BSorAS`
  V-cycle multigrid solver with block additive Schwarz smoother, damping parameter $\omega = 0.8$ (4 PCPI)
- `MG(V22)-BJacAS, MG(V22)-BSorAS`
  V-cycle multigrid solver with block additive Schwarz smoother (4 PCPI)
- `MG(V11)-BiCG-BJac, MG(V11)-BiCG-BSor`
  V-cycle multigrid solver, smoothed by 1 BiCGstab iteration with block preconditioner (4 PCPI)
- `MG(V11)-BiCG-BJacAS, MG(V11)-BiCG-BSorAS`
  V-cycle multigrid solver, smoothed by 1 BiCGstab iteration with block additive Schwarz preconditioner (4 PCPI)
- `BiCG-MG(V11)-BJac, BiCG-MG(V11)-BSor`
  BiCGstab solver, preconditioned by 1 V-cycle multigrid iteration with block smoother (4 PCPI)
- `BiCG-MG(V11)-BiCG-BJac, BiCG-MG(V11)-BiCG-BSor`
  BiCGstab solver, preconditioned by 1 V-cycle multigrid iteration, smoothed by 1 BiCGstab iteration with block preconditioner (8 PCPI)
- `BiCG-MG(V11)-BiCG-BJacAS, BiCG-MG(V11)-BiCG-BSorAS`
  BiCGstab solver, preconditioned by 1 V-cycle multigrid iteration, smoothed by 1 BiCGstab iteration with block additive Schwarz preconditioner (8 PCPI)

We begin with excluding those solvers of the list, that are already inefficient for the compressible case $\nu = 0.4$, $\mu = 8.2$ MPa. Figure 4.2 shows that the convergence rate of BiCGstab with block additive Schwarz preconditioning (`BiCG-BJacAS`, `BiCG-BSorAS`) clearly deteriorates with increasing multigrid level. Hence, estimate (4.11), which guarantees the level independency for the block Jacobi preconditioner, does obviously not hold anymore when only using block additive Schwarz preconditioners. Furthermore, when block additive Schwarz preconditioners are applied as multigrid smoothers, damping is clearly necessary: Without damping ($\omega = 1.0$), the multigrid solver diverges (`MG(V22)-BJacAS`) or shows strong level dependency (`MG(V22)-BSorAS`), while it behaves well with damping $\omega = 0.8$. For other configurations, however, this damping value may be too large and lead to divergence again. In summary, we see that block additive Schwarz preconditioners neither can be efficiently used for plain BiCGstab preconditioning, nor for plain multigrid smoothing. Consequently, the only remaining possibility is to employ them within a nested multigrid-Krylov solver, i. e., `MG(V11)-BiCG-BJacAS`, `BiCG-MG(V11)-BiCG-BJacAS` and the two corresponding `BSorAS` variants.

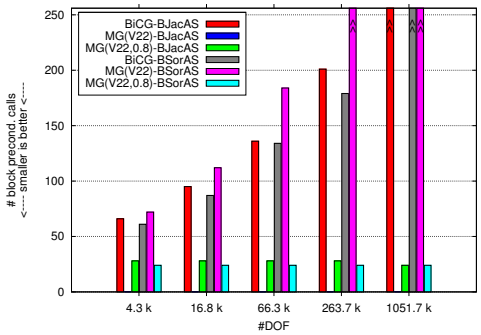
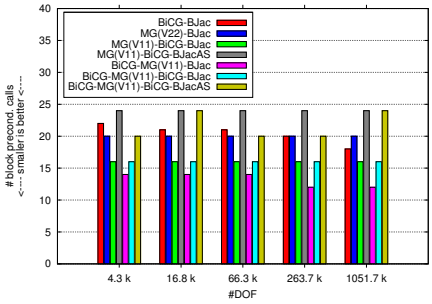


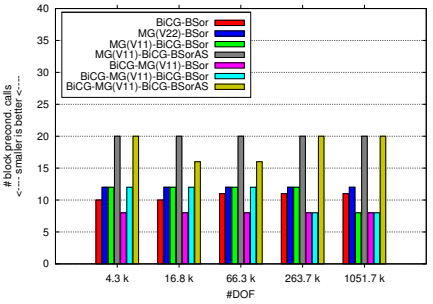
Figure 4.2: Results on the BLOCK configuration (Figure 4.1), $\nu = 0.4$, $\mu = 8.2$ MPa.

Figure 4.3 shows the results for all solvers except for those listed in Figure 4.2. The four rows of plots show the results for the four Poisson ratios $\nu \in \{0.4, 0.48, 0.49, 0.499\}$. The left column of plots shows the solvers with block Jacobi (additive Schwarz) preconditioning/smoothing, the right column all those with block SOR (additive Schwarz). We can make several observations:

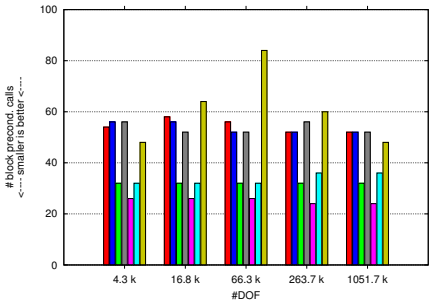
- In principle, all solver variants show level independent iteration counts, which confirms the h independency of the estimate (4.11). Of course, there are some more or less distinct oscillations, but definitely no general deterioration with increasing multigrid level.
- All solver variants suffer from increasing incompressibility, which confirms the dependency of the estimate (4.11) on the Poisson ratio ν.
- The dependence on the degree of incompressibility differs – some solvers are able to, at least, weaken its influence, others diverge when the material becomes too incompressible. Solvers that suffer strongest are
  - those that have no Krylov scheme involved at all (`MG-BJac/BSor`),



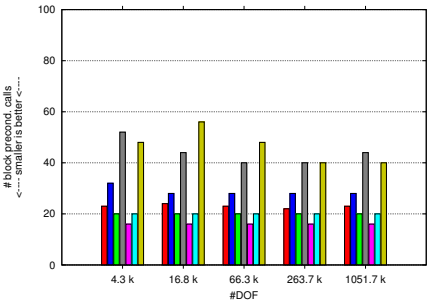
(a) $\nu = 0.4$, `BJac(AS)`.



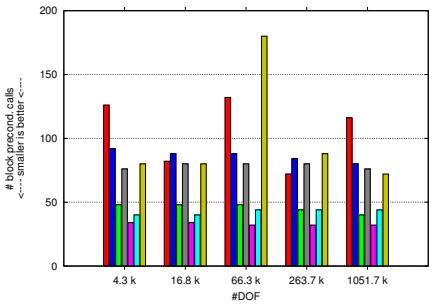
(b) $\nu = 0.4$, `BSor(AS)`.



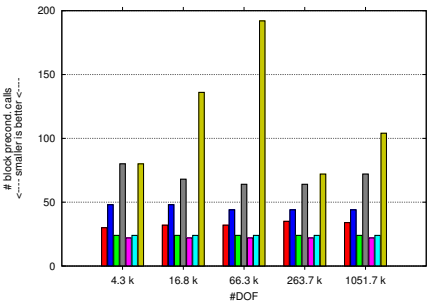
(c) $\nu = 0.48$, `BJac(AS)`.



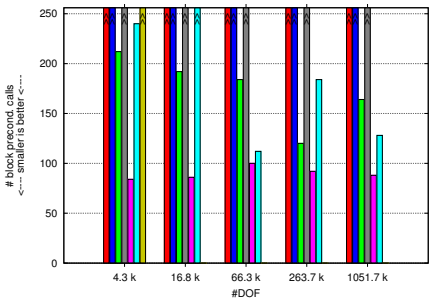
(d) $\nu = 0.48$, `BSor(AS)`.



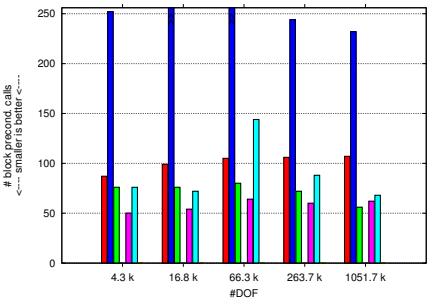
(e) $\nu = 0.49$, `BJac(AS)`.



(f) $\nu = 0.49$, `BSor(AS)`.



(g) $\nu = 0.499$, `BJac(AS)`.



(h) $\nu = 0.499$, `BSor(AS)`.

Figure 4.3: Results on the BLOCK configuration (Figure 4.1), $\mu = 8.2$ MPa, varying ν. All plots on the left (right) side share the colour key of Figure 4.3a (Figure 4.3b).

  - those that use block additive Schwarz preconditioners/smoothers (MG(V11)-BiCG-BJacAS/BSorAS, BiCG-MG(V11)-BiCG-BJacAS/BSorAS), and
  - BiCG-BJac.

  The solver BiCG-MG(V11)-BSor performs best in the majority of all cases, though, but the comparatively good performance of BiCG-BSor shows that for alleviating the effect of incompressibility it is not mandatory to include multivariate multigrid solvers.

- The block SOR preconditioner/smoother is indeed able to lower the number of iterations considerably compared to the block Jacobi preconditioner/smoother. This is especially the case for the plain BiCGstab solver BiCG-BSor, while the difference is not so drastic when a multivariate multigrid is involved.
- The two block additive Schwarz preconditioners/smoothers BJacAS and BSorAS differ only slightly. The additional matrix vector multiplication for BSorAS sometimes lowers the number of iterations (see $\nu = 0.4$ and $\nu = 0.48$), but sometimes it does the opposite (see $\nu = 0.49$ and $\nu = 0.499$).
- The block Jacobi additive Schwarz preconditioner/smoother BJacAS is competitive with the block Jacobi preconditioner/smoother BJac in some cases (compare, for example, MG(V11)-BiCG-BJacAS with BiCG-BJac and MG(V22)-BJac in Figure 4.3c), but the direct comparison, i. e., when in a given solver BJac is replaced by BJacAS, the latter clearly loses (compare MG(V11)-BiCG-BJac with MG(V11)-BiCG-BJacAS and BiCG-MG(V11)-BiCG-BJac with BiCG-MG(V11)-BiCG-BJacAS for all values of ν). The difference between BSor and BSorAS is even more distinct, which simply follows from the fact, that BSor is often clearly better than BJac, while BSorAS is only slightly better or sometimes even worse than BJacAS. Furthermore, the block additive Schwarz preconditioners/smoothers show the most irregular behaviour with respect to the refinement level, and they are the only ones that actually diverge in case of $\nu = 0.499$; the other solvers partially converge very slowly (especially not within 256 preconditioner calls), though, but at least they do not diverge. The worse performance of the block additive Schwarz preconditioners/smoothers has its origin in the level dependency that showed up when used as preconditioners for plain BiCGstab, and in the need for damping when used as smoother for plain multigrid (see Figure 4.2). Hence, embedding these block additive Schwarz preconditioners in strong Krylov-multigrid solvers clearly alleviates these problems, but cannot completely hide them.
- Comparing the MG(V22) solvers with the corresponding BiCG-MG(V11) solvers shows the positive effect of using multigrid as preconditioner of an outer Krylov iteration. In general, one can say: The more 'complicated' the situation (e. g., a higher degree of incompressibility), the stronger the benefit from such an outer Krylov scheme is. This statement will also be confirmed in the following sections. When employing the Krylov scheme as smoother for an outer multigrid (MG(V11)-BiCG), this also leads to a clear improvement compared to the plain multigrid solver. But since BJac and BSor do not require damping when employed as smoothers (what such a Krylov smoother

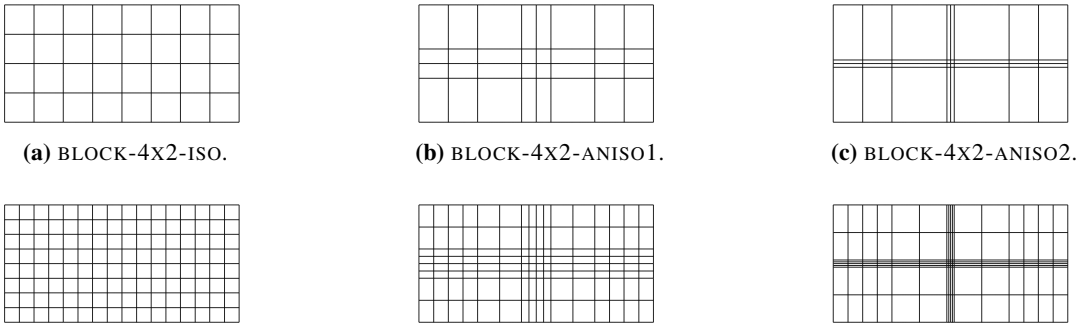
would usually be employed for), the multigrid-Krylov solvers `MG(V11)-BiCG` do not perform as well as the corresponding Krylov-multigrid variants `BiCG-MG(V11)`. Using two stages of BiCGstab (`BiCG-MG(V11)-BiCG`) does usually not improve the overall performance. When the innermost preconditioner is not robust enough, the usage of two nested BiCGstab schemes can even be harmful (compare, for example, `MG(V11)-BiCG-BSorAS` with `BiCG-MG(V11)-BiCG-BSorAS` in Figure 4.3f, or `MG(V11)-BiCG-BJacAS` with `BiCG-MG(V11)-BiCG-BJacAS` in Figure 4.3g).

In summary, it is apparent already now – although we did not investigate total arithmetic efficiencies yet (see Section 4.2.7.4) – that the additional matrix vector multiplication needed for block SOR clearly pays off in most situations. Hence, block SOR is to be preferred over block Jacobi preconditioning. At the same time it seems to be questionable already now, whether the smaller communication requirements of the block additive Schwarz preconditioners (`BJacAS` and `BSorAS`) can sufficiently balance their numerical disadvantages and render them competitive again in parallel computations. This question will be answered in Section 4.2.7.6. The favourite solver from the tests above is clearly `BiCG-MG(V11)-BSor`.

We want to stress that the purpose of these tests is to show the degradation of the solver performance with increasing incompressibility. In practice, the pure displacement formulation is not used for Poisson ratios close to $\nu = 0.5$, especially due to the known loss of accuracy in the finite element approximation (see Section 3.2.2.2 and Figure 3.6 on page 144). In Chapter 5 on page 251 we show that solvers based on the mixed formulation converge robustly with respect to the Poisson ratio; for a direct comparison between pure displacement and mixed formulation in terms of solver behaviour, see Section 5.2.1.7. For the remainder of Section 4.2.7, we only deal with compressible material.

4.2.7.2 Dependence on Mesh Anisotropies

In this section we examine the solvers' dependence on mesh anisotropies. Additionally, we vary the number of subdomains. Again, we use the BLOCK configuration depicted in Figure 4.1, but employ different refinements and macro decompositions (see Figure 4.4). First, the block is decomposed into 4×2 subdomains, each refined isotropically (Figure 4.4a) or anisotropically with anisotropy factors (cf. equation (2.4) on page 27) $a_F = 0.5, a_L = a_I = 1.0$ (BLOCK-4X2-ANISO1, Figure 4.4b), factors $a_F = 0.125, a_L = a_I = 1.0$ (BLOCK-4X2-ANISO2, Figure 4.4c) and factors $a_F = 0.03125, a_L = a_I = 1.0$ (BLOCK-4X2-ANISO3, not displayed). The same meshes are then created using 8×4 subdomains (Figures 4.4d–4.4f), where the anisotropies are already introduced in the macro decomposition and not via anisotropic refinement. The level n finite element mesh of a 4×2 block exactly equals the level $n-1$ mesh of the corresponding 8×4 block. The maximal element aspect ratios are $\sigma = 3$ (ANISO1), $\sigma = 15$ (ANISO2) and $\sigma = 63$ (ANISO3), respectively. As material we use aluminium (see Table 3.1 on page 127) with $\nu = 0.35$ and $\mu = 2.6 \cdot 10^4$ MPa, and the vertical force is -10^4 MPa. The selected refinement is somehow artificial for this configuration. However, if, e. g., the block



(a) BLOCK-4X2-ISO. **(b)** BLOCK-4X2-ANISO1. **(c)** BLOCK-4X2-ANISO2.

(d) BLOCK-8X4-ISO. **(e)** BLOCK-8X4-ANISO1. **(f)** BLOCK-8X4-ANISO2.

Figure 4.4: Different refinements/decompositions of the BLOCK configuration, level 1 depicted.

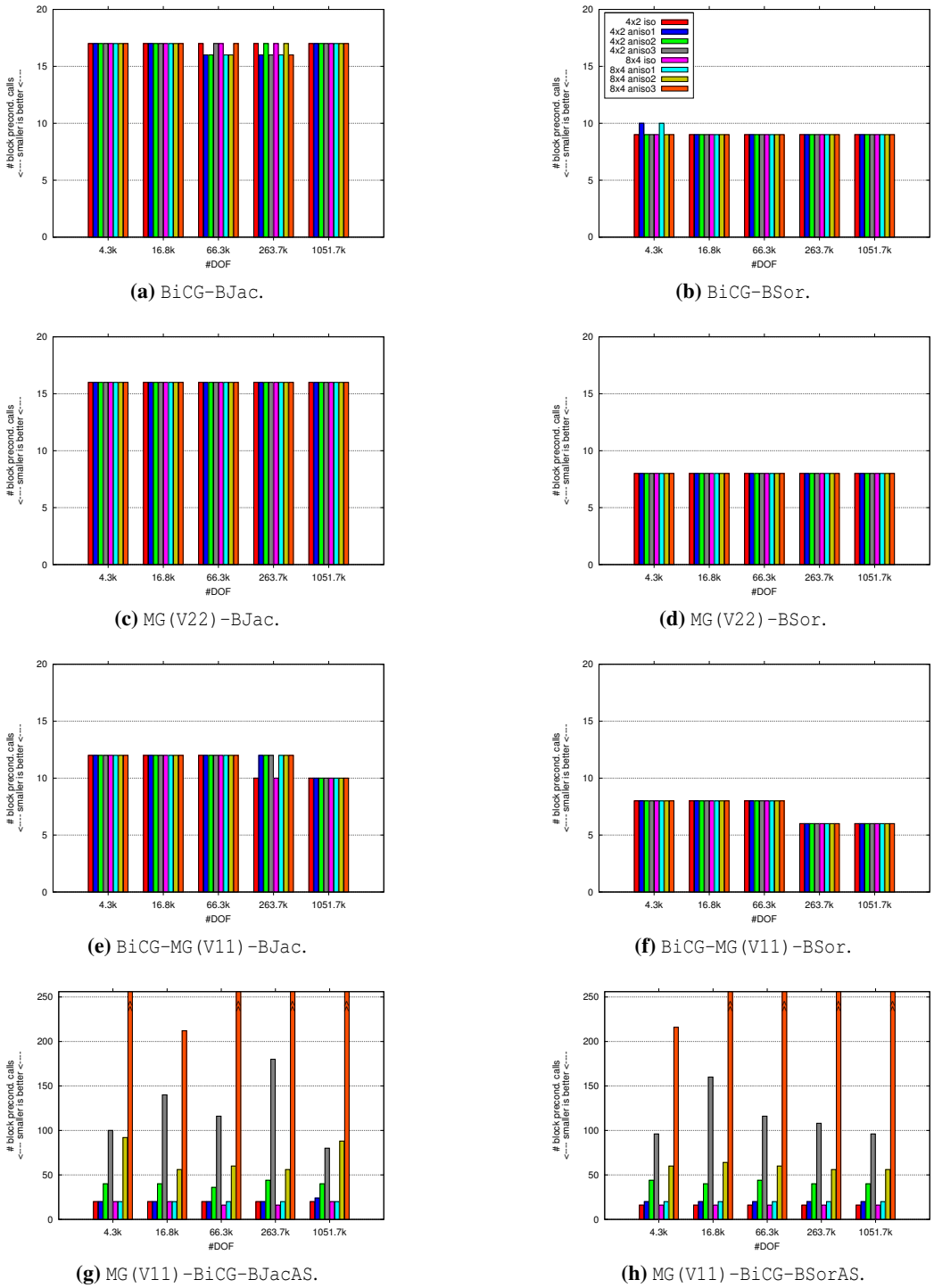
consisted of different materials such that there is a jump of material coefficients at the horizontal and vertical centre axis of the block, then large stresses would have to be expected in these regions and the refinement would make sense. In this thesis, however, the problem of jumping coefficients is not taken into account.

In Figure 4.5 we show the results for selected solvers. This time, we display in one plot the number of preconditioning calls of one solver for all eight BLOCK configurations. We can make the following observations:

- All solvers using the block preconditioners/smoothers `BJac` and `BSor` converge independently of the number of subdomains, i. e., there is no difference whether the domain is decomposed into 4×2 or 8×4 blocks. This is not surprising: In Section 4.2.3 we explained, that the multivariate solvers do not 'see' the decomposition of the domain. The only part of the entire solver for which the decomposition plays a role is within the scalar solves of the block preconditioner. This means, the robustness with respect to number of subdomains is exclusively determined by the scalar solvers.

- All solvers using the block preconditioners/smoothers `BJac` and `BSor` converge independently of the anisotropy in the mesh. Again, the scalar solves within the block preconditioner completely 'absorb' the possible difficulties arising from high element aspect ratios. This, however, is not self-evident: Since the offdiagonal matrix blocks $\mathbf{K}_{12}$ and $\mathbf{K}_{21}$ are also affected by mesh anisotropies, one could expect a negative influence on the convergence of the multivariate solver. The numerical experiments, however, show that this is not the case.

- For the block additive Schwarz preconditioners `BJacAS` and `BSorAS` the two previous statements do not apply. We can clearly observe a strong sensitivity with respect to mesh anisotropies. In the case of stronger anisotropies we additionally see a dependence on the number of subdomains, which is typical for the additive Schwarz approach (cf. Section 2.5.4). Of course, the comparison to the block preconditioners `BJac` and `BSor` is not quite fair: The latter actually suffer from the additive Schwarz approach, namely



(a) `BiCG-BJac`.

(b) `BiCG-BSor`.

(c) `MG(V22)-BJac`.

(d) `MG(V22)-BSor`.

(e) `BiCG-MG(V11)-BJac`.

(f) `BiCG-MG(V11)-BSor`.

(g) `MG(V11)-BiCG-BJacAS`.

(h) `MG(V11)-BiCG-BSorAS`.

Figure 4.5: Results on the different BLOCK configurations (Figure 4.4) for selected solvers. All plots share the colour key of Figure 4.5b.

in terms of the scalar solvers. In the tests above, however, we neglect the behaviour and the costs of the scalar solves completely. So, we have to reevaluate the situation in Section 4.2.7.4, where we examine total costs.

- Using `BSor` instead of `BJac` reduces the number of preconditioning calls considerably in most cases.

We briefly comment on the solvers omitted in Figure 4.5. The `MG(V11)-BiCG-BJac/BSor` solvers yield the same results as the corresponding `MG(V22)` solvers in Figures 4.5c and 4.5d, i. e., Krylov smoothing does not yield any improvement here. Equipping the `BiCG-MG(V11)-BJac/BSor` solvers in Figures 4.5e and 4.5f with an additional Krylov smoother even slightly increases the number of preconditioner calls in some cases, but in general shows the same convergence behaviour. This is different for block additive Schwarz smoothers: On those configurations where the solvers `MG(V11)-BiCG-BJacAS/BSorAS` already exhibit irregularities and bad convergence, the situation even deteriorates when an additional outer BiCGstab solver is employed, i. e., the number of preconditioner calls is higher, the general behaviour more irregular and the solver sometimes even diverges.

4.2.7.3 Dependence on Geometry

The condition number of the block preconditioned system depends on Korn's constant (see equation (4.11) on page 192 in Section 4.2.4). In Section 3.2.2.1 it was shown that Korn's constant grows with the anisotropy of the cantilever beam configuration. So, we use three variants of this configuration with global anisotropies of $\frac{L}{H} = 4, 16, 64$ to illustrate how the solver performance is affected. We furthermore use two different macro decompositions: On the one

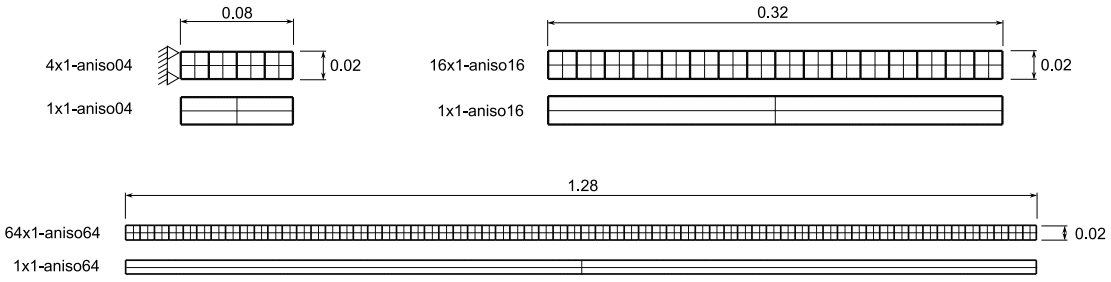


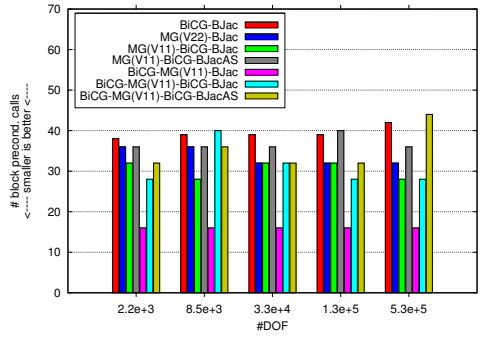
Figure 4.6: BEAM configurations, level 1 depicted. The left side is fixed (only depicted for BEAM-4X1-ANISO04), the other sides can move freely.

hand, the three beams consist of one layer of 4, 16 and 64 isotropic subdomains, respectively, and on the other hand they consist of only one anisotropic subdomain. The six resulting variants are depicted in Figure 4.6.[10] As material we use aluminium (see Table 3.1 on page 127)

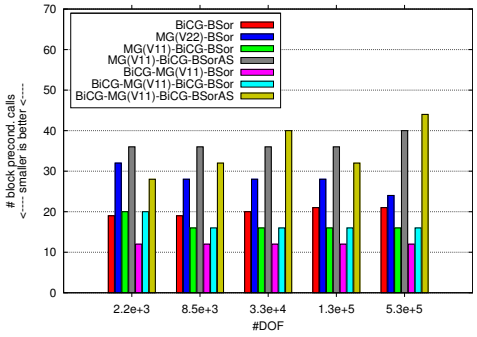
[10] Actually, the beam is rounded on its left side, which is not displayed in the coarse grids in Figure 4.6. Instead, see Figure 3.1 on page 140 and Figure 3.4 on page 143.

with $\nu = 0.35$ and $\mu = 2.6 \cdot 10^4$ MPa. The three beams are loaded by vertical body forces of $-1.6 \cdot 10^4 \frac{\mathrm{N}}{\mathrm{m}^3}$, $-4 \cdot 10^3 \frac{\mathrm{N}}{\mathrm{m}^3}$ and $-10^3 \frac{\mathrm{N}}{\mathrm{m}^3}$, respectively.

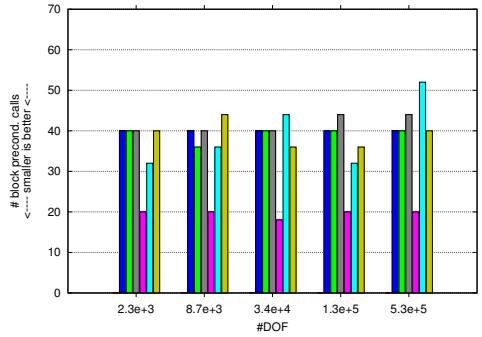
Figure 4.7 shows the number of preconditioning calls for the different solvers on the BEAM configurations consisting of isotropic subdomains, while Figure 4.8 shows those on the BEAM configurations consisting of one anisotropic subdomain. The plots on the left side show the



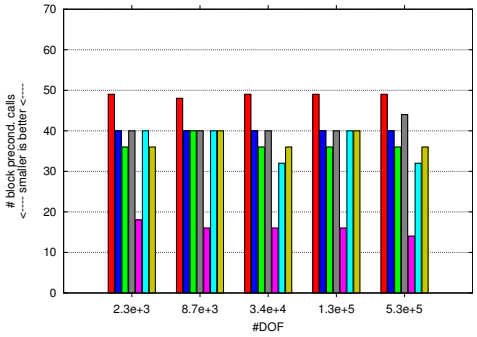
(a) BEAM-4X1-ANISO4, `BJac(AS)`.



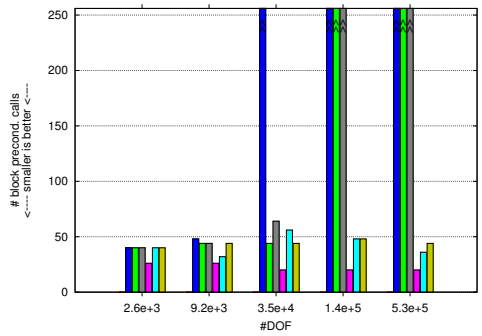
(b) BEAM-4X1-ANISO4, `BSor(AS)`.



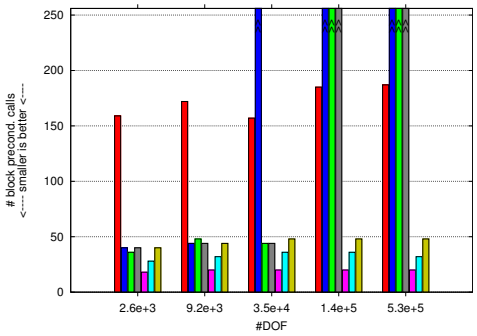
(c) BEAM-16X1-ANISO16, `BJac(AS)`.



(d) BEAM-16X1-ANISO16, `BSor(AS)`.

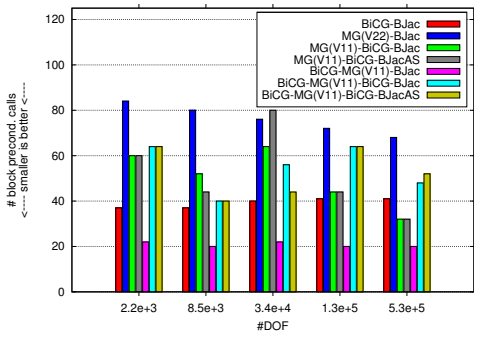


(e) BEAM-64X1-ANISO64, `BJac(AS)`.

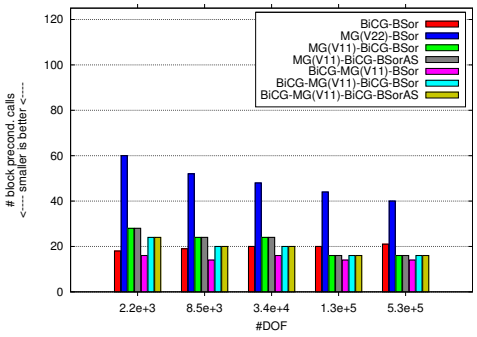


(f) BEAM-64X1-ANISO64, `BSor(AS)`.

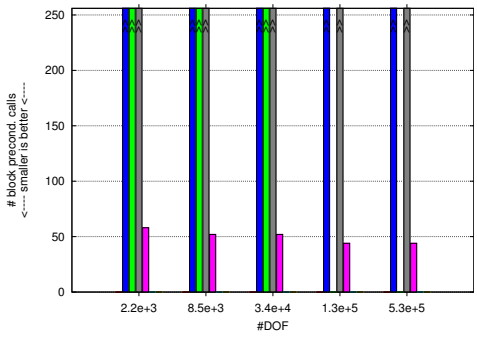
Figure 4.7: Results on the BEAM configurations consisting of isotropic subdomains (see Figure 4.6). Plots on the left share the colour key of Figure 4.7a, plots on the right that of Figure 4.7b.



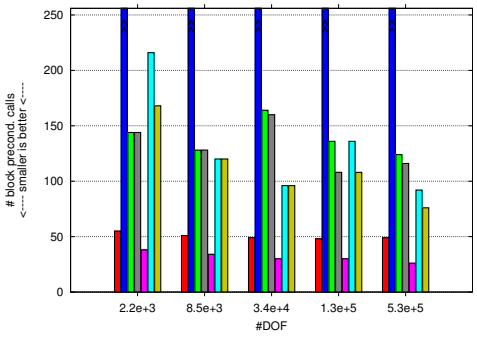
(a) BEAM-1X1-ANISO4, `BJac(AS)`.



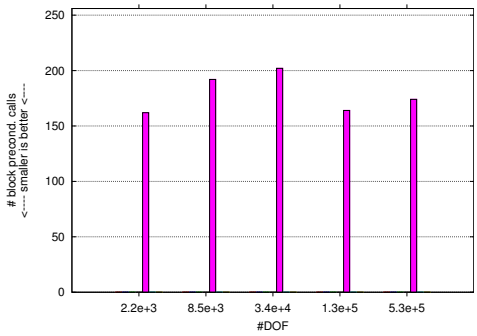
(b) BEAM-1X1-ANISO4, `BSor(AS)`.



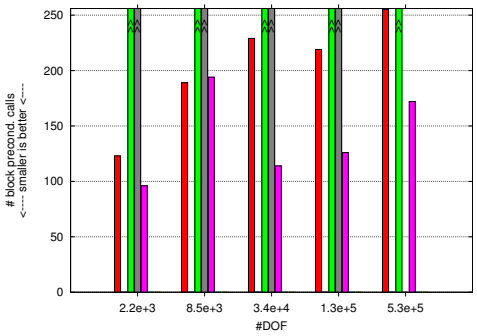
(c) BEAM-1X1-ANISO16, `BJac(AS)`.



(d) BEAM-1X1-ANISO16, `BSor(AS)`.



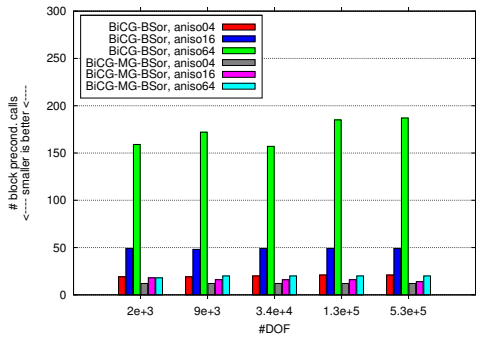
(e) BEAM-1X1-ANISO64, `BJac(AS)`.



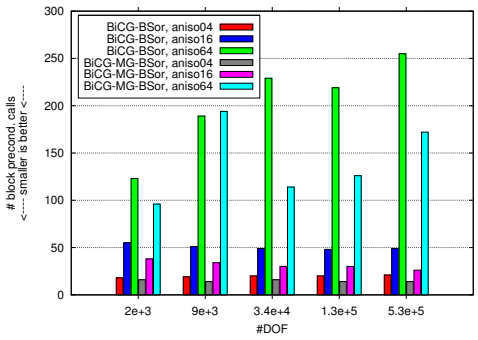
(f) BEAM-1X1-ANISO64, `BSor(AS)`.

Figure 4.8: Results on the BEAM configurations consisting of one anisotropic subdomain (see Figure 4.6). Plots on the left share the colour key of Figure 4.8a, plots on the right that of Figure 4.8b.

solvers using block Jacobi (additive Schwarz) preconditioners, while those on the right side show the solvers using block SOR (additive Schwarz) preconditioners. In Figure 4.9 we explicitly again compare the two solvers `BiCG-BSor` and `BiCG-MG-BSor`. We can make the following observations:



(a) 4X1-ANISO04, 16X1-ANISO16, 64X1-ANISO64.



(b) 1X1-ANISO04, 1X1-ANISO16, 1X1-ANISO64.

Figure 4.9: `BiCG-BSor` vs. `BiCG-MG(V11)-BSor` on the BEAM configurations consisting of isotropic subdomains (Figure 4.9a) and of one anisotropic subdomain (Figure 4.9b).

- The solver `BiCG-BJac` is only able to solve the ANISO04 configuration (Figures 4.7a and 4.8a), on the other configurations it diverges for all refinement levels. This shows the strong influence of Korn's constant. Using `BiCG-BSor` instead improves the situation drastically. There is a clear dependence on the anisotropy of the beam, though, but the solver converges for all configurations.

- Using BiCGstab as outer solver is mandatory for the anisotropic configurations. On both ANISO64 beams and on the 1X1-ANISO16 beam, the solvers `MG(V22)` and `MG(V11)-BiCG` do not converge within 256 preconditioning calls or even diverge, no matter which block preconditioner is used.

- Using BiCGstab not only as outer solver (`BiCG-MG`), but additionally as Krylov smoother (`BiCG-MG-BiCG`) does never yield improvements for the case of `BJac` or `BSor` as block preconditioner. Sometimes it even leads to divergence, where the solver `BiCG-MG` converges (see Figures 4.8c, 4.8e and 4.8f). In case of `BJacAS` or `BSorAS` as block preconditioner it is sometimes beneficial to use two stages of BiCGstab (e. g., Figures 4.7f and 4.7f), sometimes it is not (e. g., Figure 4.8c).

- In most cases, the best solver variants using the block additive Schwarz preconditioners `BJacAS/BSorAS` perform clearly worse than the best variants using block preconditioners `BJac/BSor`.

- The 1X1-ANISO64 configuration is so badly conditioned, that finally only three schemes remain that are able to solve it reliably: `BiCG-MG(V11)-BJac`, `BiCG-BSor` and `BiCG-MG(V11)-BSor`. Taking a closer look at the latter two in Figure 4.9, reveals the following: On the BEAM configurations consisting of isotropic subdomains (Figure 4.9a), `BiCG-BSor` shows the (expected) strong dependency on the degree of anisotropy, while `BiCG-MG(V11)-BSor` is able to weaken this dependency significantly. Hence, for such configurations it is very advantageous to apply an additional multigrid scheme to solve the multivariate system – the scalar multigrid solvers used for block preconditioning are

not sufficient to resolve the numerical difficulties that arise in case of the BEAM configuration. This statement is also supported by the fact, that `BiCG-MG` is the only solver that works robustly with the block Jacobi preconditioner as Figure 4.8e shows. Figure 4.9b, however, reveals that on the BEAM configurations consisting of one anisotropic subdomain, also the `BiCG-MG-BSor` solver begins to show stronger sensitivities with respect to the length of the beam. But also there it still performs clearly better than `BiCG-BSor`.

In summary, we can again state that the solver `BiCG-MG(V11)-BSor` performs best. Especially its ability to weaken the negative influence of the beam's increasing anisotropy is remarkable. In some cases the use of `BSor` instead of `BJac` does not only lower the iteration number, but it renders the solver convergent in the first place. Again, the block additive Schwarz preconditioners `BJacAS/BSorAS` are clearly less efficient and robust than the block preconditioners `BJac/BSor`.

4.2.7.4 Approximate Subsystem Solves

In the previous sections we solved all (local and global) subsystems exactly in order to investigate the block preconditioners themselves. In practice, this is much too expensive, usually the systems can be solved only approximately without sacrificing too much preconditioning capability. In this section we want to examine how the convergence behaviour of the outermost solver is influenced by approximate system solves, and we try to find a good balance between arithmetic costs for inner iterations on the one hand and number of outer iterations on the other hand. Furthermore, we investigate whether the numerical advantage of block SOR over block Jacobi does translate into smaller total runtimes, as well. Finally, we answer the question whether the block additive Schwarz preconditioner, which clearly showed worse numerical efficiency in the tests of the previous Sections, performs better in comparison to the other preconditioners when arithmetic costs are considered.

First we examine how accurately the subsystems have to be solved. We vary the inner tolerance between $\varepsilon = 10^{-8}$ and $\varepsilon = 0.5$. In order to achieve these tolerances as sharply as possible, we use a simple CG solver to treat the scalar (global or local) subsystems; a more efficient multigrid solver (which will be used in later tests) often gains one digit or more already in the first iteration, such that it would be impossible to reliably assess the coarser inner tolerances. Due to the inefficiency and level dependency of the simple CG solver, we do not evaluate arithmetic costs yet, but are only interested in the convergence behaviour of the outer solver. To limit the amount of data, we only show results for the solvers

`BiCG-BSor[CG(1e-8)]`, ..., `BiCG-BSor[CG(5e-1)]`,
`BiCG-MG-BSor[CG(1e-8)]`, ..., `BiCG-MG-BSor[CG(5e-1)]`,
`MG-BiCG-BSorAS[CG(1e-8)]`, ..., `MG-BiCG-BSorAS[CG(5e-1)]`.

Using a different outer solver or block Jacobi instead of block SOR does not affect the following observations.

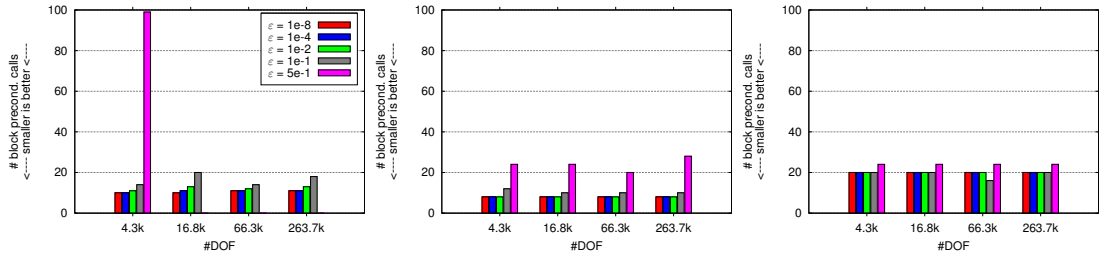
Figure 4.10 depicts the number of preconditioning calls for these three solvers on five different refinement levels of four different configurations with varying inner tolerance. We can observe, that for all solvers the inner tolerance of $\varepsilon = 0.5$ is clearly insufficient. For the solver `BiCG-BSor` it leads to divergence in most cases, while for `BiCG-MG-BSor` the outer iteration number abruptly rises from $\varepsilon = 0.1$ to $\varepsilon = 0.5$. For the block additive Schwarz preconditioner there are configurations where the iteration numbers for $\varepsilon = 0.5$ are about the same as for $\varepsilon = 0.1$ (Figures 4.10c and 4.10l), but on the other configurations a clear deterioration can be observed (Figures 4.10f and 4.10i). The solver `BiCG-BSor` on the BEAM configuration (Figure 4.10j) is the only combination where we can observe a clear difference between $\varepsilon = 10^{-8}$, $\varepsilon = 10^{-4}$ and $\varepsilon = 0.01$, for the latter the solver even diverges. This confirms the problems this solver has for the BEAM configuration (cf. Section 4.2.7.3). For all other solvers, there are practically no or only slight differences in outer convergence when the inner tolerance is varied between $\varepsilon = 10^{-8}$ and $\varepsilon = 0.01$. Also the step from $\varepsilon = 0.01$ to $\varepsilon = 0.1$ lets the outer iteration count only increase slightly or not at all in some configurations, in other configurations, however, a deterioration is clearly visible (e. g., Figures 4.10d and 4.10k).

In summary, the solver `BiCG-BSor` is most sensitive with respect to the inner tolerance, the solver `MG-BiCG-BSorAS` the least sensitive one (which is not surprising), while `BiCG-MG-BSor` lies in between. It is impossible to identify a value for the inner tolerance that is the best for every solver and every configuration. However, the tests show that $\varepsilon = 0.01$ and $\varepsilon = 0.1$ are the most promising candidates: They lead to reliable convergence behaviour (ignoring `BiCG-BSor` on the BEAM configuration) and are, of course, much cheaper than setting $\varepsilon = 10^{-4}$. A more detailed comparison of $\varepsilon = 0.01$ and $\varepsilon = 0.1$ (which we do not present here) using more efficient scalar solvers than CG reveals that in the vast majority of all cases, clearly less arithmetic work is performed when setting $\varepsilon = 0.1$ instead of $\varepsilon = 0.01$. *Consequently, for all subsequent tests we will use the inner tolerance* $\varepsilon = 0.1$*, representing a good compromise between reliability and arithmetic costs.* In some cases, the setting $\varepsilon = 0.5$ yields less arithmetic costs, though, but due to its unreliability (cf. Figure 4.10) it is not an option in view of our general goal to create robust solution schemes.

Now, that we identified a suitable value for the inner tolerance we compare the different multivariate solver and preconditioning schemes with respect to their total efficiency. For the solution of the global scalar systems we use the 1-layer-ScaRC solver defined in Listing 2.26 on page 94, a global V-cycle multigrid performing two ADITriGS pre- and postsmoothing steps and gaining one digit, such that the complete solver schemes read

```
BiCG-BJac[MG(1e-1)__ADI],
BiCG-BSor[MG(1e-1)__ADI],
BiCG-MG(V11)-BJac[MG(1e-1)__ADI],
BiCG-MG(V11)-BSor[MG(1e-1)__ADI].
```

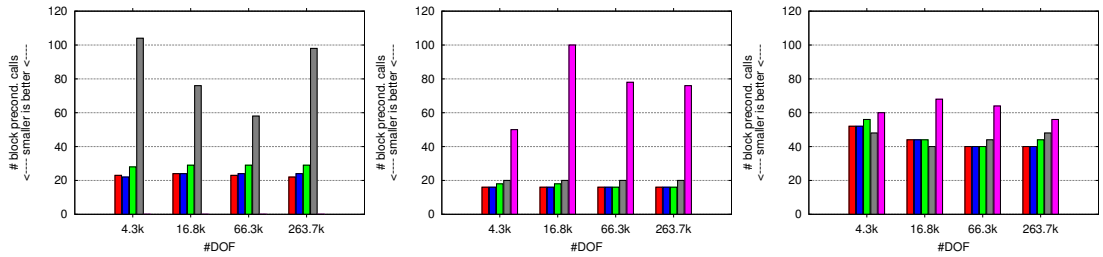
For the solution of the *local* scalar systems in case of the block additive Schwarz preconditioners, we use exactly the same solver scheme, only working on the local layer: a local V-cycle multigrid with two ADITriGS pre- and postsmoothing steps. The two corresponding solvers we use for our tests are



(a) BiCG-BSor, BLOCK-ISO, $\nu = 0.4$, $\mu = 8.2$ MPa.

(b) BiCG-MG-BSor, BLOCK-ISO, $\nu = 0.4$, $\mu = 8.2$ MPa.

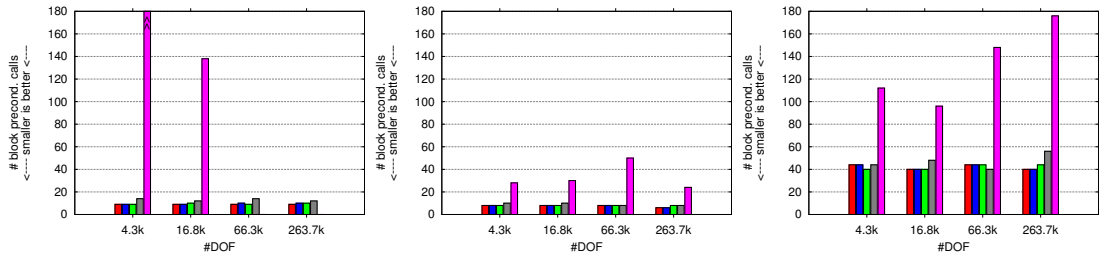
(c) MG-BiCG-BSorAS, BLOCK-ISO, $\nu = 0.4$, $\mu = 8.2$ MPa.



(d) BiCG-BSor, BLOCK-ISO, $\nu = 0.48$, $\mu = 8.2$ MPa.

(e) BiCG-MG-BSor, BLOCK-ISO, $\nu = 0.48$, $\mu = 8.2$ MPa.

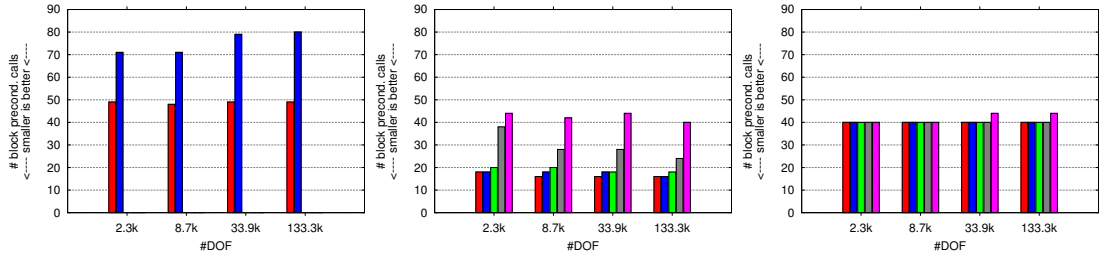
(f) MG-BiCG-BSorAS, BLOCK-ISO, $\nu = 0.48$, $\mu = 8.2$ MPa.



(g) BiCG-BSor, BLOCK-4X2- ANISO2, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(h) BiCG-MG-BSor, BLOCK-4X2- ANISO2, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(i) MG-BiCG-BSorAS, BLOCK-4X2- ANISO2, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.



(j) BiCG-BSor, BEAM-16X1- ANISO16, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(k) BiCG-MG-BSor, BEAM-16X1- ANISO16, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(l) MG-BiCG-BSorAS, BEAM-16X1- ANISO16, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

Figure 4.10: Number of block preconditioning calls for varying inner tolerance. The three columns show three different solvers, the four rows show four different configuration. All plots share the colour key of Figure 4.10a.

```
MG-BiCG-BJacAS[MG(1e-1)-ADI],
MG-BiCG-BSorAS[MG(1e-1)-ADI].
```

Hence, for the block preconditioners `BJac` and `BSor`, the scalar solver within the brackets always acts on the global layer, while for the block additive Schwarz preconditioners `BJacAS` and `BSorAS` it always acts on the local layer.

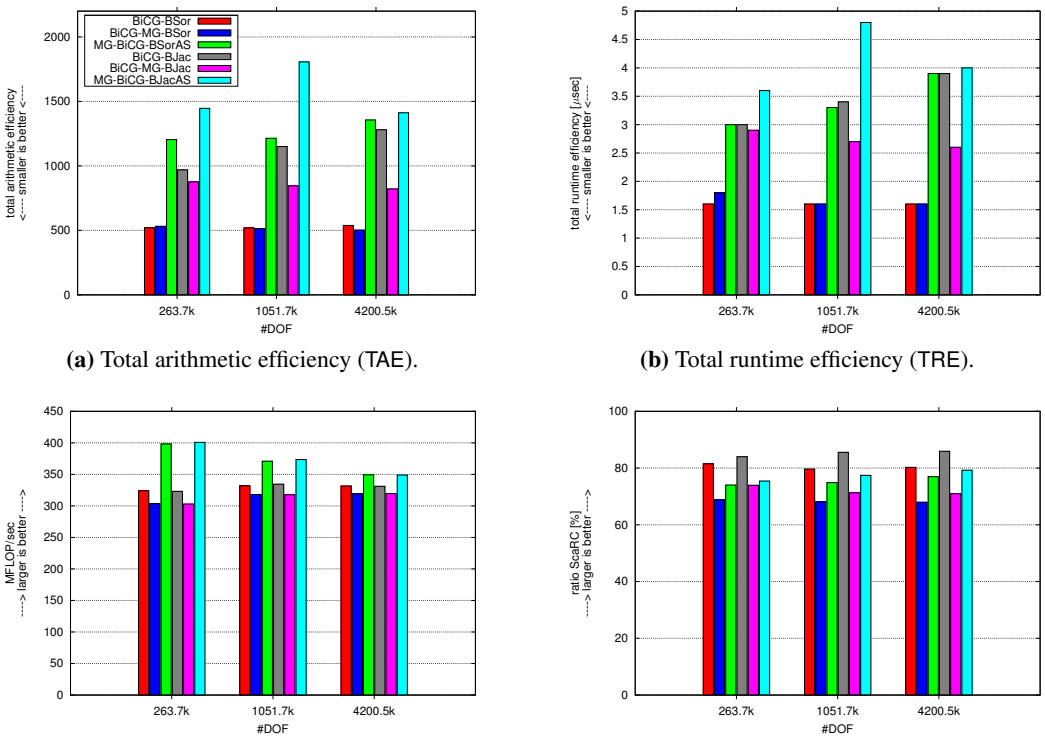
For these six solver schemes we now compare the MFLOP/s rate, the ratio the solution schemes spend in scalar ScaRC solves (measured in percent of the total solution time), the total arithmetic efficiency (TAE; see definition (2.15) on page 58), and the **total runtime efficiency** (TRE), which is defined by

$$\text{total runtime efficiency} = -\frac{T[\mu\text{sec}]}{\#\text{DOF} \times \#\text{iter} \times \log_{10}(c)}. \tag{4.22}$$

It represents the runtime T in microseconds scaled with the number of DOF and with the digits the iteration actually gained (cf. equation (2.12) on page 48), i. e., *the total runtime efficiency tells how many microseconds are needed per DOF to gain one digit* and, hence, represents a suitable measure to compare the runtimes of different solution algorithms. We only perform serial computations, the parallel efficiency of the solvers will be investigated in Section 4.2.7.6. The compute server is an AMD Opteron DP 844 with 1.8 GHz.

We begin with the simple BLOCK-ISO configuration with Poisson ratio $\nu = 0.4$ and shear modulus $\mu = 8.2$ MPa, this time for the higher refinement levels 7, 8 and 9 (see Figure 4.11). We can make some interesting observations:

- While the tests in the previous sections already showed that block SOR reduces the number of iterations compared to block Jacobi, we now see that – despite of the additional matrix vector multiplication per preconditioning call – this also leads to significantly reduced total arithmetic costs and runtimes (compare the respective solver variants in Figures 4.11a and 4.11b). The differences are most distinct in case of the simple outer BiCGstab solver (compare `BiCG-BJac` and `BiCG-BSor`). MFLOP/s rates and ScaRC ratios are nearly equal for block Jacobi and block SOR (see Figures 4.11c and 4.11d). These findings are confirmed for all the other configurations, although the performance difference between block Jacobi and block SOR are not always that clear. Nevertheless, block SOR is superior in the vast majority of all cases. *Consequently, we will neglect block Jacobi from now on and will exclusively use block SOR (additive Schwarz) preconditioning in the subsequent tests.*
- While the solvers `BiCG-BSor` and `BiCG-MG-BSor` exhibit nearly the same efficiency (see Figures 4.11a and 4.11b; 47 s runtime on the finest level), the worse performance of the block SOR additive Schwarz preconditioner (115 s runtime on the finest level), that was already stated in the previous sections, is confirmed here.
- The MFLOP/s rate of `BiCG-BSor` is slightly higher than that of `BiCG-MG-BSor`. This can be explained by the fact that more work is spent on coarser grid levels where the ratio between algorithmic overhead due to function calls and data organisation (which costs time



(a) Total arithmetic efficiency (TAE).

(b) Total runtime efficiency (TRE).

(c) MFLOP/s rate.

(d) ScaRC ratio.

Figure 4.11: Total arithmetic efficiency (TAE), total runtime efficiency (TRE), MFLOP/s rate and ratio of ScaRC solves for six solvers on the configuration BLOCK-ISO, $\nu = 0.4$, $\mu = 8.2$ MPa. All plots share the colour key of Figure 4.11a.

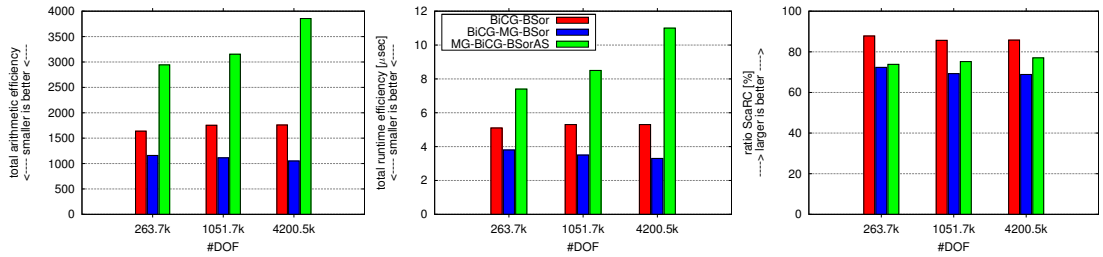
but is not counted for the calculation of the MFLOP/s rate) and pure arithmetic work gets more and more unfavourable. The MFLOP/s rate of the block SOR additive Schwarz preconditioner is clearly the best. The reason is that large parts of the arithmetic work (i. e., the local multigrid solves) can be performed locally on the subdomains without being interrupted by data exchange/alignment along subdomain boundaries (cf. Section 2.1.1). However, we can also observe that the MFLOP/s rate slightly deteriorates with increasing multigrid level. This is due to the fact, that not all the necessary data fits into the cache and more (expensive!) memory access has to be performed. The last argument is, of course, also true for the first two solvers. There, however, the effect is superimposed by the higher number of inter-subdomain data exchanges. We want to mention that the obtained MFLOP/s rates are very close to the optimal value, which is imposed by the available bandwidth, and not by the processor's peak performance.

- An advantage of `BiCG-BSor` is that 80% of the solution time is spent in scalar ScaRC solves (see Figure 4.11d), while for `BiCG-MG-BSor` it is less than 70% (`MG-BiCG-BSorAS` lies in between). This value is especially important for our solver strategy: The basic idea is to reduce the solution of the multivariate system to the solution of scalar systems by

highly efficient ScaRC solvers (see Sections 4.1 and 4.2.4). Hence, the more time a solver actually spends in solving scalar systems (relative to the total solution time), the more it profits from this efficiency. Future numerical improvements or hardware-oriented adaptations of the scalar ScaRC solvers correspondingly translate into performance gains of the multivariate solvers. As a concrete example we want to mention the possibility to accelerate the scalar ScaRC solvers by using GPUs [73]. Currently, there is no way to also accelerate the 20%–30% of the multivariate solvers that are spent outside the scalar solves. Hence, the smaller this ratio is, the more the multivariate solver can benefit from GPU acceleration [72, 75].

We now examine the efficiency of the three solvers in more complicated situations, i. e., on the BLOCK-ISO configuration with less compressible material ($\nu = 0.48$, $\mu = 8.2$ MPa), on the BLOCK-4X2-ANISO2 configuration with mesh anisotropies, and on the BEAM-16X1-ANISO16 configuration exhibiting a long and thin geometry. Figure 4.12 shows the TAE, the TRE and ScaRC ratios. (The MFLOP/s rates are qualitatively the same as in Figure 4.11, so we omit their representation.) We can make the following observations:

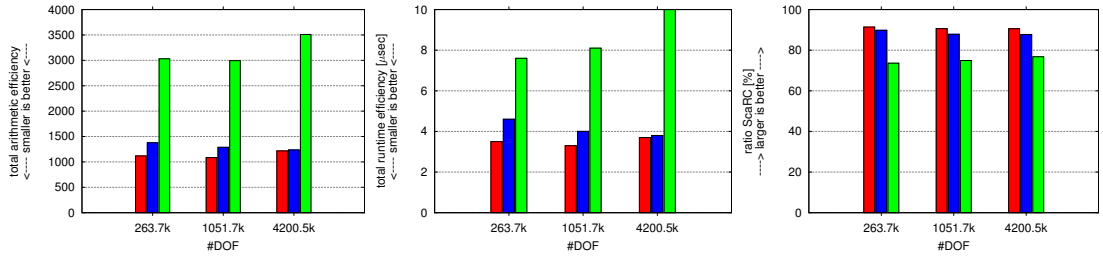
- While `BiCG-MG-BSor` is slightly less efficient than `BiCG-BSor` on the BLOCK-4X2-ANISO2 configuration (see Figures 4.12d and 4.12e), it is significantly more efficient on the other two configurations (see Figures 4.12a, 4.12b, 4.12g and 4.12h), being up to twice as fast (e. g., 44 s vs. 100 s runtime on the finest level of the beam configuration).
- Again, `MG-BiCG-BSorAS` shows the worst performance on all configurations.
- In case of the beam configuration and the BLOCK-4X2-ISO configuration with $\nu = 0.48$, the ScaRC ratios (Figures 4.12i and 4.12c) are comparable to those on the BLOCK-4X2-ISO configuration with $\nu = 0.4$ (Figure 4.11d). On the configuration BLOCK-4X2-ANISO2, however, the two solvers using the `BSor` preconditioner exhibit a clearly higher ratio of ScaRC solving time (about 90%), while the solver `MG-BiCG-BSorAS` exhibits the same ratios as on the other configurations. This shows that the difficulties of the BLOCK-4X2-ANISO configurations do not lie on the local, but on the global layer: Locally on each subdomain, the ADITriGS smoother is able to robustly deal with the occurring mesh anisotropies, as the ScaRC ratio of `MG-BiCG-BSorAS` shows. The actual difficulty is the smaller overlap region due to the anisotropic refinement towards the inner boundaries, which negatively influences the convergence of the *global* domain decomposition method. Hence, only the global ScaRC solvers within the `BSor` preconditioner suffer, but not the local one within the `BSorAS` preconditioner. In detail, the global ScaRC solver `MG(1e-1)__ADI` needs 2.5–3 iterations on average to gain the desired digit, while the local ScaRC solver `MG(1e-1)-ADI` always needs only one iteration. (On the BLOCK-4X2-ISO configuration with $\nu = 0.4$, for example, also the global ScaRC solvers need only one iteration on average.) While in case of the `BSor` preconditioner already the global scalar ScaRC solver copes with the occurring difficulties and thus hides them from the outer multivariate solver, this is not the case for the `BSorAS` preconditioner, where the multivariate solver itself has to deal with these difficulties and exhibits correspondingly worse convergence rates. The general comparison between `BSor` and `BSorAS`, however, shows



(a) TAE, BLOCK-ISO, $\nu = 0.48$, $\mu = 8.2$ MPa.

(b) TRE, BLOCK-ISO, $\nu = 0.48$, $\mu = 8.2$ MPa.

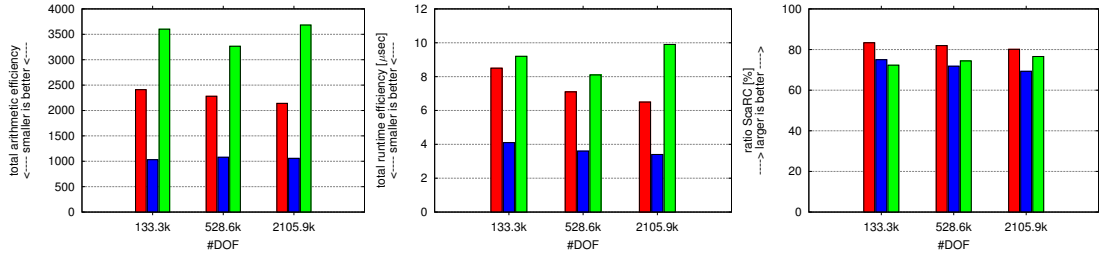
(c) ScaRC ratio, BLOCK-ISO, $\nu = 0.48$, $\mu = 8.2$ MPa.



(d) TAE, BLOCK-4X2-ANISO2, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(e) TRE, BLOCK-4X2-ANISO2, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(f) ScaRC ratio, BLOCK-4X2-ANISO2, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.



(g) TAE, BEAM-16X1-ANISO16, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(h) TRE, BEAM-16X1-ANISO16, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

(i) ScaRC ratio, BEAM-16X1-ANISO16, $\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa.

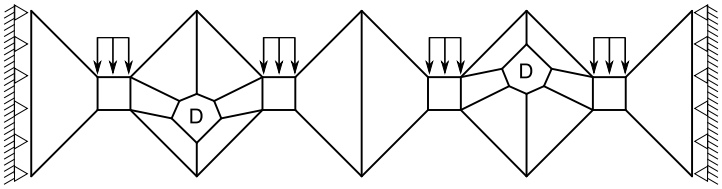
Figure 4.12: Total arithmetic efficiency (TAE), total runtime efficiency (TRE) and ScaRC ratios for the three solvers using block (additive Schwarz) SOR preconditioning on three different configurations. All plots share the colour key of Figure 4.12b.

that it is clearly the better strategy to hide the arising irregularities from the multivariate solver by using global ScaRC solvers, i. e., `BSor`.

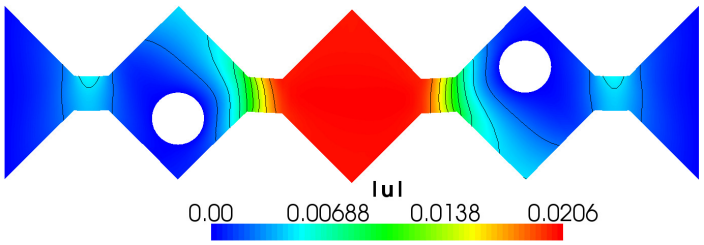
- Overall, the solver `BiCG-MG-BSor` again turns out to be the most efficient one, i. e., its advantages that we already stated in the previous sections directly translate into (partially significantly) smaller total runtimes.

In Section 2.6 we identified the ScaRC solver `BICG-MG__ADI__D` (see Listing 2.27 on page 94) to be the most efficient one. We also computed the tests above with this solver, but found that it led to higher values for TAE and TRE in most cases. The reason is that the simpler solver

MG__ADI__D is absolutely sufficient for the configurations above; the outer BiCGstab does not significantly improve the numerical efficiency, but increases the arithmetic costs. However, in Section 2.6 we used the ACCORDION configuration (see Figure 2.47 on page 98) to show that BICG-MG__ADI__D indeed *can* be clearly superior over MG__ADI__D (cf. Figure 2.46d on page 97). We now want to test whether this result can be transferred to the multivariate elasticity solvers. Therefore, we take the ACCORDION domain, use aluminium as material ($\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa) and apply boundary conditions as depicted in Figure 4.13a. We use the



(a) Macro decomposition (level 0) consisting of 20 subdomains and applied boundary conditions. The left and right side and the holes are fixed (denoted by 'D'), and the horizontal boundary segments are loaded by vertical surface forces of -10^3 MPa.



(b) Deformed body (displacements' magnitude displayed).

Figure 4.13: ACCORDION configuration.

two refinements ACCORDION-ISO and ACCORDION-ANISO1 (see page 85 and the description of Figure 2.47 on page 98) and solve the resulting system with BiCG-MG-BSor as multivariate solver and MG__ADI__D or BICG-MG__ADI__D as scalar solver. Figure 4.14 shows the resulting total arithmetic efficiency for the four combinations. One can see that on the ACCORDION-ISO

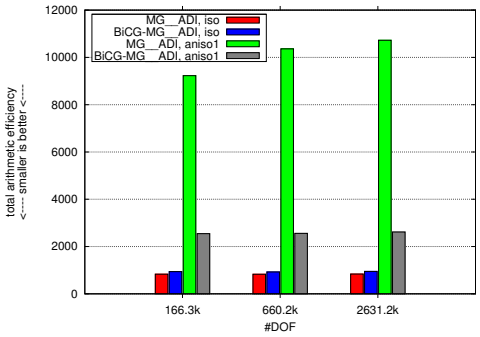


Figure 4.14: Total arithmetic efficiency of the solver BiCG-MG-BSor using two different scalar solvers MG__ADI and BiCG-MG__ADI on the two configurations ACCORDION-ISO and ACCORDION-ANISO1.

configuration it does not make a big difference whether MG__ADI or BiCG-MG__ADI is used to solve the scalar systems; with the latter the overall solver is even slightly more expensive (850 vs. 950). On the ACCORDION-ANISO1 configuration, however, the difference is significant: While for MG__ADI as scalar solver the TAE rises to more than 10000, it only rises to roughly 2500 for BiCG-MG__ADI, i. e., the former is about four times more expensive on the ACCORDION-ANISO1 configuration.[11] *Hence, we can state that the benefit of using* BiCG-MG__ADI *instead of* MG__ADI *does clearly transfer to the case of multivariate elasticity solvers.* We can make another interesting observation when comparing the results for BiCG-MG__ADI in Figure 4.14 with those in Figure 2.46d on page 97: The solution of the elasticity problem in the special case of the ACCORDION-ANISO1 grid with the given boundary conditions is about twice as costly as the solution of the Poisson problem on the same grid.[12]

4.2.7.5 1-layer-ScaRC vs. 2-layer-ScaRC

In Section 2.6.4 we showed that for the solution of the scalar Poisson problem, 2-layer-ScaRC solvers can be superior to 1-layer-ScaRC solvers. We now want to test if this result can be transferred to the elasticity case where the ScaRC solvers are only used for preconditioning. In doing so, we confine ourselves to Jacobi as elementary smoother (cf. the discussion in Section 2.6.4.3). We use the configurations ISO/ISO, ANISO/ISO and ANISO/ANISO described in Section 2.6.4.1 (see Figure 2.31 on page 77), but solve an elasticity problem with boundary conditions as depicted in Figure 4.15a, material aluminium ($\nu = 0.35$, $\mu = 2.6 \cdot 10^4$ MPa), and a horizontal surface force of $-5 \cdot 10^3$ MPa. We compare four different solvers, which differ in how they nest various layers of multigrid schemes:

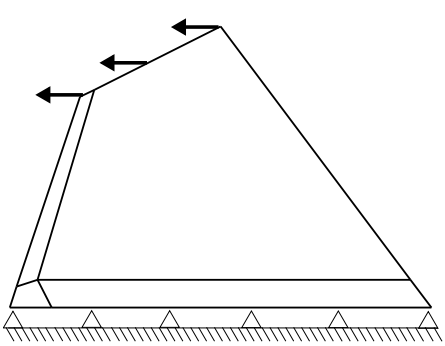
1. BiCG-MG(1,V11)-BSor[MG(1e-1)-FGMRES4__JAC__D]:
   This solver (denoted as '1-layer' in Figure 4.16) comprises two nested multigrid schemes – a global multivariate one and a global scalar one. The scalar 1-layer-ScaRC solver is defined in Listing 2.24 on page 89.

2. BiCG-MG(1,V11)-BSor[MG(1e-1)-FGMRES4__MG-BICG-JAC-D__D]:
   This solver (denoted as '2-layer') comprises three nested multigrid schemes – a global multivariate one, a global scalar one and a local scalar one. The scalar 2-layer-ScaRC solver is defined in Listing 2.15 on page 74 and the local multigrid solver is truncated on level 7 (16641 DOF).

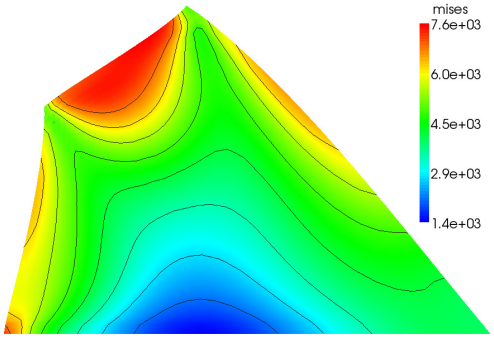
3. MG(V11)-BiCG(1)-BSorAS[MG(5e-1)-BiCG(2)-Jac]:
   This solver (denoted as 'BSorAS') comprises two nested multigrid schemes – a global multivariate one and a local scalar one.

[11] The reason for the general increase compared to the ISO configuration is the anisotropic refinement towards inner boundaries (cf. explanation in the list of observations for Figure 4.12). The additional work is completely performed within the scalar ScaRC solves, the outer multivariate solver performs 2.5 iterations in all four cases.

[12] Note that the total arithmetic efficiency is a well suited measurement to even compare the linear solving costs for two different physical equations.



(a) Macro decomposition (level 0) consisting of four subdomains with applied boundary conditions. The bottom side is fixed, and the upper side is loaded by a horizontal surface force of $-5 \cdot 10^3$ MPa.



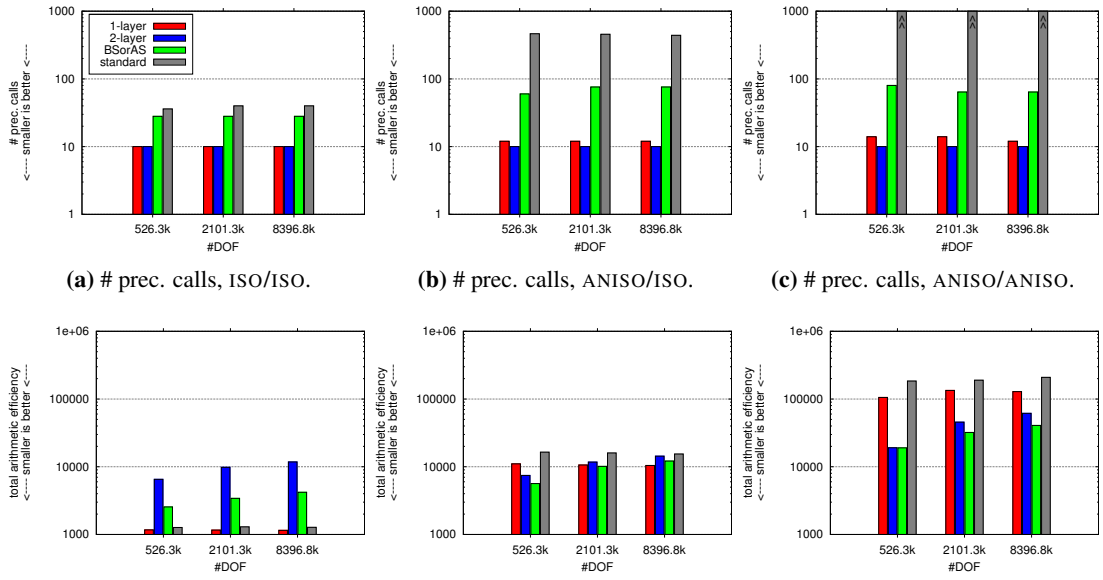
(b) Deformed body (von Mises stress displayed).

Figure 4.15: Configuration from Figure 2.31 modified for elasticity.

4. `MG(F11)-BiCG(2)-Jac`:
   This solver (denoted as 'standard') comprises only one multigrid scheme – a global multivariate one. Instead of a block smoother, it uses standard point Jacobi for smoothing, i. e., no scalar systems are solved, everything happens on the multivariate layer. This solver can be regarded as a 'standard' multigrid scheme which treats the linear system as a whole and neglects its block-structure.

Figure 4.16 compares the number of (block) preconditioning calls and the total arithmetic efficiency for the four solvers on the three configurations ISO/ISO, ANISO/ISO and ANISO/ANISO. Due to the large variations we use a logarithmic scale for the y-axes. The upper row shows that only for the first two solvers – those using global scalar ScaRC solvers – the numbers of preconditioning calls are (nearly) independent of the configuration, namely 10 for the 2-layer-ScaRC solver in all cases and 10–14 for the 1-layer-ScaRC solver. The number of preconditioning calls for the third solver (using the block additive Schwarz preconditioner) is clearly higher on the two ANISO configurations than on the ISO/ISO configuration. For the fourth solver (the standard multigrid), the increase is much more drastic; on the ANISO/ANISO configuration it needs more than 6000 preconditioning calls.

Of course, the arithmetic costs per preconditioning call differ significantly for the four solver schemes, which can be seen in the lower row of Figure 4.16. On the ISO/ISO configuration (Figure 4.16d), for example, the first two solvers perform the same number of preconditioning calls, but the 2-layer-ScaRC variant is roughly ten times more expensive. On the ANISO/ISO configuration (Figure 4.16e), the two solvers perform roughly the same work, while on the ANISO/ANISO configuration (Figure 4.16f) the 2-layer-ScaRC variant is clearly superior. Furthermore, the 2-layer-ScaRC solver needs on all configurations 1 iteration on average to gain the desired digit (i. e., all irregularities are hidden within the local multigrid solves), while the 1-layer-ScaRC solver needs 6 iterations on the ANISO/ISO configuration and 70 iterations on



(a) # prec. calls, ISO/ISO. **(b)** # prec. calls, ANISO/ISO. **(c)** # prec. calls, ANISO/ANISO.

(d) TAE, ISO/ISO. **(e)** TAE, ANISO/ISO. **(f)** TAE, ANISO/ANISO.

Figure 4.16: Number of preconditioning calls (upper row) and total arithmetic efficiency (lower row) for four different solvers on the three configurations ISO/ISO, ANISO/ISO and ANISO/ANISO. All plots share the colour key of Figure 4.16a. Note the logarithmic scale.

the ANISO/ANISO configuration on average. *So, we can state that the superiority of the 2-layer-ScaRC solver that we observed in Section 2.6.4 for the scalar Poisson equation clearly applies to the elasticity case, as well.*[13]

It is interesting to see, that the third solver (using the extended 2-layer-ScaRC concept in form of the `BSorAS` preconditioner; cf. Section 4.2.5) is comparatively successful on the configurations at hand. In all cases it performs less arithmetic work than the second solver (using the classical scalar 2-layer-ScaRC solver). However, it also needs much more preconditioning calls and thus much more communication effort. That is why the second solver is still to be preferred. But the third solver is, at least, superior to the first solver (using 1-layer-ScaRC). The latter needs much less (multivariate) preconditioning calls, though, but its scalar 1-layer-ScaRC solver, working on the global layer, performs up to 70 iterations per call, and exhibits correspondingly high communication requirements.

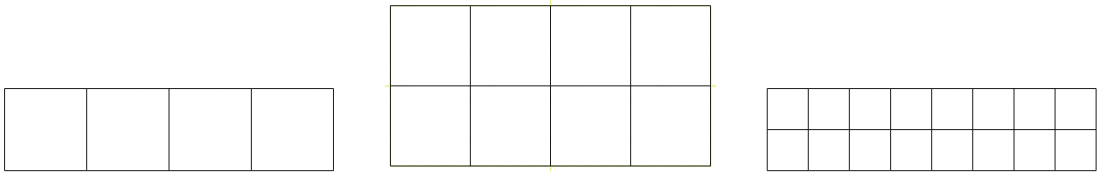
The standard multigrid solver using pointwise Jacobi is only competitive on the ISO/ISO configuration and – at least in terms of the TAE – on the ANISO/ISO configuration. On the ANISO/ANISO configuration, however, it exhibits the worst performance, especially in terms of number

[13] We want to emphasise, that the performed tests only serve the purpose to principally demonstrate the possible superiority of the 2-layer-ScaRC concept (see Section 2.6.4 for a discussion in this regard). If stronger elementary smoothers are available (e. g., ADITriGS), the three configurations can actually be solved with a total arithmetic efficiency of less than 1000.

of preconditioning calls. This result is important since it shows that the block preconditioning concept does not only serve the purpose to facilitate the application of highly specialised concepts, which are often only available for scalar systems (like ADITriGS smoothing), to multivariate systems; it can also significantly improve the performance of the elementary Jacobi smoother (for which the block structure is actually not mandatory).[14]

4.2.7.6 Parallel Efficiency

In this section we want to examine whether the good parallel efficiency of the scalar ScaRC solvers [16, 71, 94] transfers to the multivariate elasticity solvers. In detail, we investigate *weak scalability* of the solution algorithms, i. e., how the runtime changes when the problem size and the number of processors are increased by the same factor. An algorithm that exhibits perfect weak scalability yields equal runtimes, i. e., the ratio between communication and computation times remains constant. We use the BLOCK-ISO configuration and material with Poisson ratio $\nu = 0.4$ and shear modulus $\mu = 8.2$ MPa. We partition the domain with an increasing number of subdomains and vary the geometry to obtain square shaped subdomains in all cases. We



(a) 4×1 subdomains, 4.2 M vertices, 4 processors.

(b) 4×2 subdomains, 8.4 M vertices, 8 processors.

(c) 8×2 subdomains, 16.8 M vertices, 16 processors.

(d) 8×4 subdomains, 33.6 M vertices, 32 processors.

(e) 16×4 subdomains, 67.1 M vertices, 64 processors.

(f) 16×8 subdomains, 134.2 M vertices, 128 processors.

Figure 4.17: Different macro decompositions of two versions of the BLOCK configuration.

schedule one subdomain per processor and vary the number of processors between 4 and 128 (see Figure 4.17). The grid is refined 10 times such that we obtain a problem size of 2.1 million DOF per subdomain, i. e., the total number of DOF varies between 8.4 and 268.5 million. For

[14] While the Jacobi smoother can be applied to multivariate systems very easily (without respecting any block structure), this is not the case for ADITriGS and similar more sophisticated elementary smoothers. The development of a *multivariate* ADITriGS *smoother*, i. e., which does not work blockwise on scalar systems but on the whole multivariate system, is theoretically possible, though, but requires extensions and/or reimplementations of major parts of the SparseBandedBLAS library (see Section 2.1.1).

the calculations we use up to 65 compute nodes of the LiDO cluster of the TU Dortmund. Each node contains two AMD Opteron DP 250 CPUs with 2.4 GHz, the nodes are connected by an InfiniBand switch.[15]

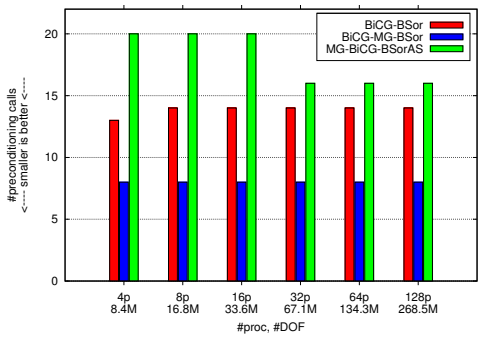
We consider the three solvers we already compared in Figure 4.12 on page 218 (using ADITriGS as elementary smoother again). As explained in Section 4.2.3, all operations are performed blockwise on scalar operators such that FEAST's underlying parallelisation concept is employed automatically without further complication. The only noteworthy additional task is the assembly and communication of the data needed for direct coarse grid solving (see Section 4.2.3). As in the case of scalar ScaRC solvers, the global multivariate coarse grid problems are solved on a separate master process (which 'knows' the complete global coarse grid), while all other operations are performed in parallel on several slave processes assigned to different subdomains (see Kilian [94] and Becker [16] for details on the parallel implementation of FEAST).

Figure 4.18 shows the results for the three solvers. Those plots that contain time measurements use graphs instead of bar charts to better assess the deviation from the case of perfect weak scalability which would be a straight horizontal line. Due to the slight variations in iteration counts, we display the total linear solving time in Figure 4.18c and the linear solving time per iteration in Figure 4.18d. The total runtime efficiency (Figure 4.18f) is multiplied with the number of processors, such that also here a straight horizontal line represents the optimal case of perfect weak scalability.

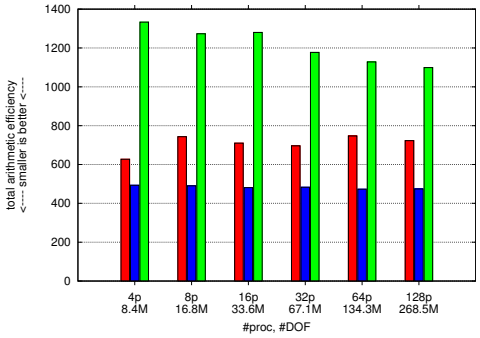
We can make the following observations:

- Figure 4.18a shows that the solver `BiCG-MG-BSor` needs exactly 2 iterations (8 preconditioning calls) in all six computations. For the other two solvers `BiCG-BSor` and `MG-BiCG-BSorAS` the number of iterations varies slightly between 6.5 and 7 iterations (13–14 preconditioning calls) and between 5 and 4 iterations (20–16 preconditioning calls), respectively.
- The total arithmetic efficiency (see Figure 4.18b) is constant for the solver `BiCG-BSor`, while it slightly varies for the solver `BiCG-MG-BSor`. For the solver `MG-BiCG-BSorAS` a slight decrease can be observed.
- Overall, the solver `MG-BiCG-BSorAS` shows a slightly irregular behaviour, while the other two solvers behave quite smoothly.
- In total solution time (Figure 4.18c), the solver `BiCG-MG-BSor` is clearly the best, closely followed by `BiCG-BSor`. However, the runtime of the solver `MG-BiCG-BSorAS` is roughly twice as long.
- The solver `MG-BiCG-BSorAS` showed the worst performance in most of the tests in previous sections. The last remaining question is whether it will do better (compared to

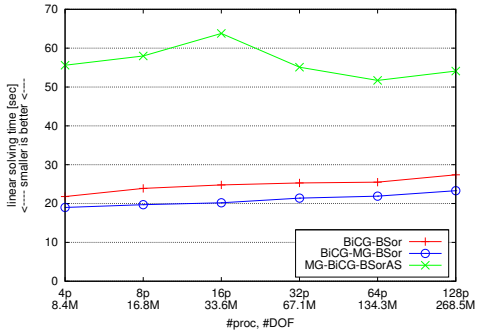
[15] For further details, see `http://www.itmc.tu-dortmund.de/en/hochleistungsrechnen/lido/index.html`.



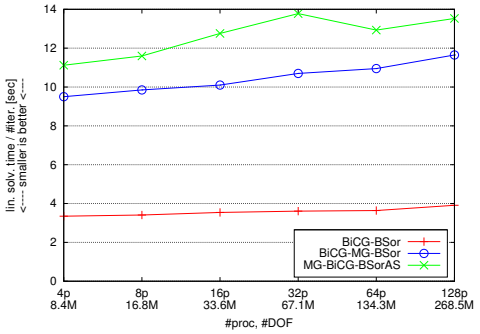
(a) Number of preconditioning calls.



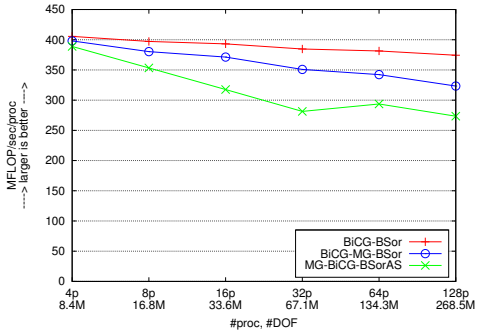
(b) Total arithmetic efficiency.



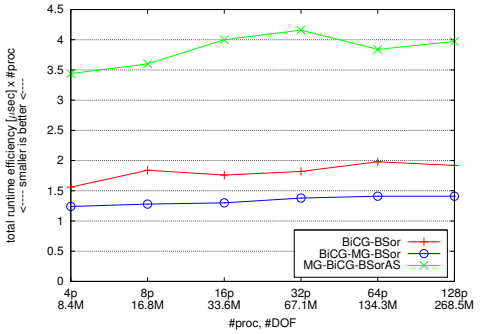
(c) Linear solving time.



(d) Linear solving time per iteration.



(e) MFLOP/s rate per processor.



(f) TRE multiplied with the number of processors.

Figure 4.18: Parallel performance of three different solvers on the six BLOCK configurations displayed in Figure 4.17.

the other two solvers) with increasing number of processors. Figures 4.18d and 4.18e show that this is clearly not the case. With increasing number of processors, the time per iteration slightly increases for *all* three solvers, though, but the solver `BiCG-BSor` clearly shows the smallest increase. The loss of efficiency of `MG-BiCG-BSorAS` is clearly greater, and it is comparable to that of `BiCG-MG-BSor` (ignoring the irregularities of `MG-BiCG-BSorAS`). These trends are confirmed by the MFLOP/s rate per processor, which

decreases most slowly for the BiCG-BSor solver. The reason is that on the multivariate layer the BiCG-BSor solver does only work on the finest grid, while the other two solvers (using multivariate multigrid schemes) work on on all grids (suffering from the deteriorating ratio between arithmetic work and communication effort), especially including the application of a multivariate direct coarse grid solver running on the master process only. The BiCG-BSor solver has to deal with these disadvantages on the scalar layer only.

- The total runtime efficiency (Figure 4.18f) does not only assess mere runtimes, but puts them into relation to the actual reduction of the residual, and can thus be seen as the most meaningful and most important measure. Here, we can see that the solver BiCG-MG-BSor is superior again: It shows the smallest variations and also the smallest increase.

Altogether, all three solvers show good weak scalability, the differences are rather small. Since we cannot perform simulations with more processors on LiDO, we can only extrapolate the results: Due to the trends observable in the various plots, we expect, especially when taking the total runtime efficiency into account, that also for a larger number of processors BiCG-BSor will *not* be superior to BiCG-MG-BSor. *Hence, also in terms of parallel efficiency, we regard* BiCG-MG-BSor *as our favourite solver.* In Göddeke et al. [75] we present further examples of large-scale cluster computations showing the good parallel efficiency of our elasticity solvers and their successful acceleration by using graphics cards.

4.2.8 Summary

We compared the different solver strategies described in Section 4.2.6 for the case of the pure displacement formulation for linearised elasticity. The main results are:

- The block (additive Schwarz) SOR preconditioner is clearly superior to the block (additive Schwarz) Jacobi preconditioner.

- Using the block additive Schwarz preconditioner/smoother within a standard BiCGstab or multigrid solver is not robust; BiCG-BSorAS shows a clear dependency on the refinement level and MG-BSorAS requires manual setting of the damping parameter. The only possibility to use BSorAS is within a MG-BiCG solver which adjusts the damping parameter automatically.

- Using two layers of multivariate BiCGstab, i. e., BiCG-MG-BiCG, is in most cases not more efficient than MG-BiCG or BiCG-MG; in some cases it is even harmful and can lead to a significant loss of robustness and/or efficiency.

- Using BiCG-MG-BSor instead of MG-BSor improves the efficiency and robustness significantly in most cases. In some cases, it is even mandatory (see, e. g., Figure 4.7f on page 209).

- On 'easy' configurations, `BiCG-BSor` and `BiCG-MG-BSor` exhibit similar efficiency. On more complicated ones (like the BEAM configuration), however, `BiCG-MG-BSor` is clearly superior.
- The block additive Schwarz preconditioner `BSorAS` disappoints in all aspects. In most cases, it is by far less efficient than the standard block preconditioner `BSor`. On more complicated configurations it even diverges, while `BSor` does not. Furthermore, the fact that the amount of communication required for one `BSorAS` step is small compared to one step of `BSor`, cannot outweigh the bad numerical behaviour, such that finally `BSorAS` does not excel in terms of parallel efficiency either.
- For solving scalar subsystems within block preconditioning, the best compromise between inner arithmetic costs and outer convergence rate is to gain one digit.
- The superiority of scalar 2-layer-ScaRC over 1-layer-ScaRC solvers in certain situations can also be observed when they are applied as block preconditioners in multivariate solvers, i. e., the 2-layer-ScaRC concept is justified also in the context of elasticity.
- The good parallel efficiency of the scalar ScaRC solvers fully transfers to the multivariate elasticity solvers.

In summary, we can state that the strategy of reducing the solution of multivariate systems to solving scalar systems is successful. The multivariate solver `BiCG-MG-BSor` clearly performs best in all the numerical tests and is therefore our favourite solver. For the solution of the scalar systems a standard `MG__ADI` solver is usually sufficient. Since we want to concentrate on other aspects in the next sections, we will confine ourselves to this combination of multivariate and scalar solver from now on. We will show that it is also able to efficiently and robustly deal with more complicated situations.

4.3 Pure Displacement Formulation for Finite Deformation Elasticity

In this section we are concerned with the solution of the nonlinear equation

$$\mathbf{r}(\mathbf{u}) = \mathbf{0}, \tag{4.23}$$

where $\mathbf{r}(\mathbf{u}) = \mathbf{f}^{\text{in}}(\mathbf{u}) - \mathbf{f}^{\text{ex}}$ is the residual vector (depending on the current discrete solution vector $\mathbf{u}$) measuring the imbalance of external and internal forces (see equation (3.87) on page 148 and its derivation). The equation is the algebraic expression of the nonlinear boundary value problem (3.40) on page 129. We confine ourselves to the Neo-Hooke constitutive law (see equation (3.32) on page 125 and equation (3.42) on page 129), only in one validation test (Figures 4.22 and 4.23 and Table 4.2c) we use the St. Venant-Kirchhoff model (see equation (3.31) on page 124 and equation (3.41) on page 129). In the following subsections we describe the

basic solution technique (Section 4.3.1), present a strategy to significantly improve its robustness, validate the correctness of the computed solutions and present comparative tests (Section 4.3.2). Then, we investigate the effect of solving the linear subsystems iteratively and only approximately (Section 4.3.3). After that, we examine the use of numerically calculated tangents (Section 4.3.4). Finally, we summarise the results and briefly describe further possible improvements (Section 4.3.6).

4.3.1 The Newton-Raphson Method

We solve equation (4.23) by means of the *Newton-Raphson method*, one of the most commonly used methods to solve nonlinear equations, especially in CSM. Listing 4.3 represents the method in algorithmic form. One can see that the solution of the nonlinear equation system is

```
1  subroutine newtonRaphson(f^in, f^ex, u, ε^NL_rel, ε^NL_abs)
2    ! Input:      internal forces f^in (as a function of u),
3    !             external force vector f^ex,
4    !             start vector u,
5    !             rel. and abs. tolerance ε^NL_rel, ε^NL_abs
6    ! Output:     solution u of the equation f^in(u) − f^ex = 0
7
8    r(u) ← f^in(u) − f^ex                          ! initial residual
9    ε ← max{‖r(u)‖ε^NL_rel, ε^NL_abs}              ! termination criterion
10   do while (‖r(u)‖ > ε)
11     K̃ ← ∂r(u)/∂u                                 ! calculate tangent matrix
12     solve K̃û = −r(u)                             ! solve linear system
13     u ← u + û                                    ! update solution
14     r(u) ← f^in(u) − f^ex                        ! compute new residual
15   enddo
16 end subroutine newtonRaphson
```

Listing 4.3: The Newton-Raphson method for solving the nonlinear equation $\mathbf{r}(\mathbf{u}) = \mathbf{0}$.

basically reduced to the solution of a series of linear equation systems (line 12), that exhibit the same 2×2 block structure as in the case of linearised elasticity. Hence, we can efficiently use the multivariate solvers introduced in Section 4.2. Exploiting the block preconditioners explained in that section, means that finally even the solution of a nonlinear multivariate elasticity problem is basically reduced to the solution of several scalar linear problems that can be treated by the parallel ScaRC solvers. All the other basic operations in the Newton-Raphson method work inherently in parallel due to FEAST's domain decomposition approach, such that, finally, the whole nonlinear solution method automatically runs in parallel.

The second time-consuming operation – besides linear solving – within the Newton-Raphson method is the reassembly of the tangent matrix (line 11 in Listing 4.3) in each iteration. If these two operations (assembling and linear solving) are performed exactly, the Newton-Raphson method yields the typical quadratic convergence rate in the vicinity of the solution [92]. For

the analytic computation of the tangent matrix see, for example, Wriggers [176]. If a numeric approximation of the tangent is used (see Section 4.3.4) or if the linear systems are only solved approximately (see Section 4.3.3), the quadratic convergence is not guaranteed. For theoretical results on the Newton-Raphson method see Kelley [92]. The termination criterion for the nonlinear (NL) iteration is given by a combination of relative and absolute tolerance (line 9).

4.3.2 The Global Newton-Raphson Method

It is well known that the standard Newton-Raphson method as defined in Listing 4.3 might fail if the start solution is too far from the real solution of the problem (see, e. g., Kelley [92]). One possibility to overcome this problem is to use **incremental loading**: Instead of applying the full external load $\mathbf{f}^{\text{ex}}$ at once, it is slowly incremented. The standard Newton-Raphson scheme is embedded into an outer loop of L loading steps (see Listing 4.4). The rationale is, simply

```
1  subroutine newtonRaphsonLoadIncr(f^in, f^ex, u, ε^NL_rel, ε^NL_abs, L)
2    ! Input:  internal forces f^in  (as a function of u),
3    !         external force vector f^ex,
4    !         start vector u,
5    !         rel. and abs. tolerance ε^NL_rel, ε^NL_abs,
6    !         number of load increments L
7    ! Output: solution of the equation f^in(u) − f^ex = 0
8
9    do k = 1,2,...,L
10     ! call the Newton-Raphson algorithm with the current u as start vector
11     u ← newtonRaphson(f^in, (k/L) f^ex, u, ε^NL_rel, ε^NL_abs)
12   enddo
13 end subroutine newtonRaphsonLoadIncr
```

Listing 4.4: The Newton-Raphson method with incremental loading, exploiting the standard Newton-Raphson method in Listing 4.3.

spoken, that for sufficiently large L, the elastic deformation caused by the load increment $\frac{1}{L}\mathbf{f}^{\text{ex}}$ is so small that the solution of the previous iteration is close enough to the real solution and the Newton-Raphson scheme is guaranteed to converge. The disadvantages of this concept obviously are that the arithmetic costs significantly increase with larger number of loading steps and that this number has to be chosen a priori by the user. Setting L too small may lead to failure of the overall method, setting L too large will result in unnecessarily high computation times.

A more sophisticated way to increase the robustness of the Newton-Raphson method is the so called **line search**: The update $\hat{\mathbf{u}}$ computed by the standard Newton-Raphson scheme (line 12 in Listing 4.3) is only used as search direction. Instead of applying it fully, one searches along the line $\{\mathbf{u} + s\hat{\mathbf{u}} \mid s \in (0,1]\}$ for a better suited value. The resulting method is also called **damped Newton-Raphson**. The question is how to determine the damping parameter s. Therefore, let $f(\mathbf{u}) \in \mathbb{R}$ be some measure tending to zero in the course of the Newton-Raphson iteration.

Clearly, the update should lead to $|f(\mathbf{u}+s\hat{\mathbf{u}})| < |f(\mathbf{u})|$, but in order to avoid stagnation it has to be ensured that the decrease is not too small. This can be done by fulfilling a **sufficient decrease condition**, the so called **Armijo rule**:

$$|f(\mathbf{u}+s\hat{\mathbf{u}})| < (1-\theta s)|f(\mathbf{u})|.$$

Here, $\theta < 1$ is a small positive constant; following Eisenstat and Walker [60] and Kelley [92] we choose $\theta = 10^{-4}$. For the measure f we use the norm of the current residual[16], i. e.,

$$f(\mathbf{u}) := \|\mathbf{r}(\mathbf{u})\|.$$

A simple line search algorithm using the Armijo rule is presented in Listing 4.5.

```
1  subroutine lineSearch(u, û, θ)
2    ! Input:  previous iterate u,
3    !         search direction û,
4    !         sufficient decrease parameter θ
5    ! Output: new iterate u
6
7    s ← 1                                   ! set initial step size
8    u_tr ← u + sû                           ! compute trial point
9    do while (‖r(u_tr)‖ ⩾ (1 − θs)‖r(u)‖)
10     s ← s/2                               ! current point rejected, decrease step
11     u_tr ← u + sû                         ! compute new trial point
12   enddo
13   u ← u_tr                                ! accept trial point as new iterate
```

Listing 4.5: Simple line search.

Starting with the full update, the step size s is halved (line 10) until the sufficient decrease condition is fulfilled (line 9). This 'trial and error' procedure stepping back along the search line is sometimes also referred to as *backtracking*.

The reduction factor 2 in line 10 could be replaced by any other positive factor greater than 1. Unfortunately, it depends on the given problem which value is well suited, it can hardly be determined a priori. Hence, instead of simply reducing the step size by a constant factor, we consider a more sophisticated approach to calculate a new trial step size. The strategy is to find an 'optimal' value based on the information about $f(\mathbf{u}) = \|\mathbf{r}(\mathbf{u})\|$ that is already available. In detail, when s_{cur} denotes the current step size (i. e., $s_{\text{cur}} = 1$ in the first iteration), then we want to determine an optimal value

$$s_* \in [\sigma_l s_{\text{cur}}, \sigma_r s_{\text{cur}}], \tag{4.24}$$

where $0 < \sigma_l < \sigma_r < 1$ are so called **safeguards** that guarantee sufficient progress within the iterative process [92]. On the one hand, σ_l avoids taking too small step sizes which could lead to

[16] Another possibility is, for instance, $f(\mathbf{u}) := |\hat{\mathbf{u}}^{\mathsf{T}}\mathbf{r}(\mathbf{u})|$.

stagnation of the (outer) Newton-Raphson iteration. On the other hand, σ_r enforces a minimum reduction rate of the trial step size to avoid stagnation of the (inner) line search iteration. In our implementation we set $\sigma_l = 0.1$ and $\sigma_r = 0.95$. To determine s_* we consider the function

$$g(s) := f(\mathbf{u} + s\hat{\mathbf{u}}) = \|\mathbf{r}(\mathbf{u} + s\hat{\mathbf{u}})\|$$

and its derivative

$$\begin{aligned} g'(s) &= \frac{d}{ds}\sqrt{\mathbf{r}(\mathbf{u}+s\hat{\mathbf{u}})^\mathsf{T}\mathbf{r}(\mathbf{u}+s\hat{\mathbf{u}})} \\ &= \frac{1}{2\|\mathbf{r}(\mathbf{u}+s\hat{\mathbf{u}})\|} 2\mathbf{r}(\mathbf{u}+s\hat{\mathbf{u}})^\mathsf{T}\frac{\partial \mathbf{r}(\mathbf{u}+s\hat{\mathbf{u}})}{\partial \mathbf{u}}\hat{\mathbf{u}} \\ &= \frac{1}{g(s)}\mathbf{r}(\mathbf{u}+s\hat{\mathbf{u}})^\mathsf{T}\frac{\partial \mathbf{r}(\mathbf{u}+s\hat{\mathbf{u}})}{\partial \mathbf{u}}\hat{\mathbf{u}}. \end{aligned}$$

The aim is now to approximate this function by a quadratic polynomial $p(s) = a_2 s^2 + a_1 s + a_0$ whose minimiser should represent a good approximation of the optimal step size. We can uniquely determine the polynomial with the help of the three values

$$\begin{aligned} p(0) &= g(0) = \|\mathbf{r}(\mathbf{u})\|, \\ p(s_{\text{cur}}) &= g(s_{\text{cur}}) = \|\mathbf{r}(\mathbf{u}+s_{\text{cur}}\hat{\mathbf{u}})\|, \\ p'(0) &= g'(0) = \frac{1}{g(0)}\mathbf{r}(\mathbf{u})^\mathsf{T}\frac{\partial \mathbf{r}(\mathbf{u})}{\partial \mathbf{u}}\hat{\mathbf{u}}. \end{aligned}$$

The derivative value can be further simplified by using the relation

$$\frac{\partial \mathbf{r}(\mathbf{u})}{\partial \mathbf{u}}\hat{\mathbf{u}} = \tilde{\mathbf{K}}\hat{\mathbf{u}} = -\mathbf{r}(\mathbf{u}),$$

assuming that the linear system within the Newton-Raphson iteration (see line 12 in Listing 4.3) has been solved exactly; if not, it is still a good approximation.[17] Hence, we obtain

$$p'(0) = -\frac{1}{g(0)}\mathbf{r}(\mathbf{u})^\mathsf{T}\mathbf{r}(\mathbf{u}) = -g(0) = -\|\mathbf{r}(\mathbf{u})\|.$$

All three values $g(0)$, $g(s_{\text{cur}})$ and $g'(0)$ are readily available since they have already been computed for the sufficient decrease test (line 9 in Listing 4.5). The coefficients of the polynomial are then given by

$$a_2 = \frac{g(s_{\text{cur}}) - g'(0)s_{\text{cur}} - g(0)}{s_{\text{cur}}^2}, \qquad a_1 = g'(0), \qquad a_0 = g(0),$$

[17] When the linear system within the Newton-Raphson method is solved iteratively, one could also exploit the last defect vector of the iterative solver, say $\mathbf{d}_n$, to obtain the correct value of $\tilde{\mathbf{K}}\hat{\mathbf{u}}$, i. e., $\mathbf{d}_n = -\mathbf{r}(\mathbf{u}) - \tilde{\mathbf{K}}\hat{\mathbf{u}}_n \Rightarrow \tilde{\mathbf{K}}\hat{\mathbf{u}}_n = -\mathbf{r}(\mathbf{u}) - \mathbf{d}_n$. Alternatively, one could perform the matrix vector multiplication directly, which is, of course, comparatively expensive.

which in the special case of $g(s) = \|\mathbf{r}(\mathbf{u} + s\hat{\mathbf{u}})\|$ amounts to

$$a_2 = \frac{\|\mathbf{r}(\mathbf{u} + s_{\text{cur}}\hat{\mathbf{u}})\| - (1 - s_{\text{cur}})\|\mathbf{r}(\mathbf{u})\|}{s_{\text{cur}}^2}, \qquad a_1 = -\|\mathbf{r}(\mathbf{u})\|, \qquad a_0 = \|\mathbf{r}(\mathbf{u})\|.$$

Since g is nonnegative, it is justified to simply determine the global minimiser of p within the interval $[0, s_{\text{cur}}]$. Therefore, we determine the zero of p', i. e.,

$$p'(s_*) = 0 \Leftrightarrow s_* = -\frac{a_1}{2a_2} = \frac{s_{\text{cur}}^2 \|\mathbf{r}(\mathbf{u})\|}{2(\|\mathbf{r}(\mathbf{u} + s_{\text{cur}}\hat{\mathbf{u}})\| - (1 - s_{\text{cur}})\|\mathbf{r}(\mathbf{u})\|)}. \tag{4.25}$$

Due to the currently violated sufficient decrease condition (line 9 in Listing 4.5) we have $a_2 > 0$ and thus $s_* > 0$. We then apply the safeguards (4.24)

$$s_* \leftarrow \begin{cases} \sigma_l s_{\text{cur}} & \text{if } s_* < \sigma_l s_{\text{cur}} \\ \sigma_r s_{\text{cur}} & \text{if } s_* > \sigma_r s_{\text{cur}} \\ s_* & \text{else.} \end{cases} \tag{4.26}$$

This especially catches the case that the zero of p' lies outside the interval $(0, s_{\text{cur}})$.[18]

We apply a further heuristic precaution for determining the step size, based on the following observation: The damped Newton-Raphson scheme tends to take small step sizes when still far away from the solution. These regions should be traversed carefully in order not to take too large steps in a unsuitable direction, thus avoiding divergence of the entire scheme. This can be achieved by bounding the step size with respect to the step size s_{prev} taken in the previous Newton-Raphson iteration. In detail, we apply

$$s_* \leftarrow \min\{s_*, 2s_{\text{prev}}\}, \tag{4.27}$$

i. e., the new step size s_* must not be larger than twice the previous step size s_{prev}. Hence, when the step size is small (say, $s_{\text{prev}} = 0.05$), the subsequent step size is small, too ($s_* = 0.1$). For larger step sizes ($s_{\text{prev}} \geqslant 0.5$) the heuristic has no effect since the full step of $s_* = 1.0$ is the natural upper bound, anyway. Note, that this heuristic possibly overwrites the value computed with help of the quadratic polynomial. Of course, this procedure might unnecessarily slow down the progress of the Newton-Raphson scheme in some cases; in other cases, however, it prevents divergence. Hence, we deliberately sacrifice some efficiency to increase the robustness of the entire solution algorithm. Listing 4.6 summarises the complete line search method.

The presented strategy to determine an 'optimal' value for the step size can be varied. Instead of using the three data points $g(0)$, $g(s_{\text{cur}})$ and $g'(0)$ to determine the quadratic polynomial, one could also use $g(0)$, $g(s_{\text{cur}})$ and $g(s_{\text{prev}})$. Furthermore, taking all four data points one could

[18] Note, that the function g is nonnegative, though, but that this does not have to be true for the interpolating polynomial p, i. e., it can happen that $p(s_*) < 0$ for $s_* \in (0, s_{\text{cur}})$. Nevertheless, we will use this minimiser as an approximation of the minimiser of g.

```
1  subroutine lineSearchQuadr(u, û, θ, σ_l, σ_r, s_prev)
2    ! Input:  previous iterate u,
3    !         search direction û,
4    !         sufficient decrease parameter θ,
5    !         safeguards σ_l, σ_r,
6    !         step size of the previous Newton iteration s_prev
7    ! Output: new iterate u
8
9    s ← 1                                        ! set initial step size
10   u_tr ← u + s û                               ! compute trial point
11   do while (‖r(u_tr)‖ ⩾ (1 − θs)‖r(u)‖)
12     s_* ← s²‖r(u)‖ / (2(‖r(u_tr)‖ − (1−s)‖r(u)‖))   ! new trial step size (see eq. (4.25))
13     if (s_* < σ_l s) s = σ_l s                 ! apply safeguards (see eq. (4.26))
14     else if (s_* > σ_r s) s = σ_r s
15     else s = s_*
16     u_tr ← u + s û                             ! compute new trial point
17   enddo
18   if (s > 2 s_prev) s = 2 s_prev               ! apply additional heuristic
19   u ← u_tr                                     ! accept trial point as new iterate
```

Listing 4.6: Line search using quadratic polynomials.

calculate the interpolating *cubic* polynomial. As these are all only heuristics to approximate the optimal value for the step size, one cannot identify *the* best approach. We implemented and tested all three variants and found there are always examples where method A is better than method B and other examples where it is the other way around. So, it is advisable to provide different line search variants the user can select from. For the tests in this thesis, however, we confine ourselves to the described quadratic model.

Generally, the global Newton-Raphson scheme using the line search method and the heuristics described above is much more robust than the standard Newton-Raphson scheme described in Section 4.3.1 (see the numerical results at the end of this section). Nevertheless, there are still situations where the method can fail, e. g., in the presence of bifurcation points (see Section 3.1.5.5) or when the solution tends to a local minimum $\mathbf{u}$ with $\|\mathbf{r}(\mathbf{u})\| \neq \mathbf{0}$. In such cases, one has to apply more sophisticated methods as for example *arc-length continuation* [105].

We now perform a number of numerical tests to validate the correctness of the computed solutions, on the one hand, and to assess the globalisation technique, on the other hand. We use two benchmark configurations of Reese et al. [133], the rectangular block under compression we already considered before (see Figure 4.1 on page 200) and the well known *Cook's plane* example. For the latter we consider two different macro decompositions, a regular one and an irregular one (notation COOK-REG and COOK-IRREG; see Figure 4.19). We apply the same material parameters as in Reese et al. [133], i. e., $\mu = 80.194$ MPa and $\nu = 0.3$. The block is loaded by vertical surface forces varying between $f = -100$ MPa and $f = -600$ MPa (notation BLOCK-100, BLOCK-200 etc.), while Cook's plane is loaded by a vertical surface force of $f = 15.625$ MPa applied to the right side.

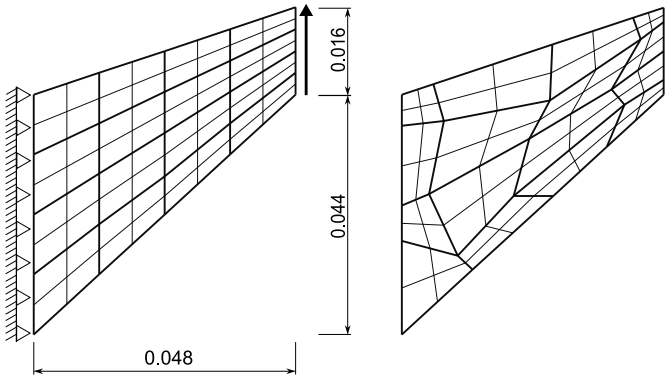


Figure 4.19: Cook's plane consisting of 16 regularly (left, COOK-REG) or irregularly (right, COOK-IRREG) shaped subdomains, level 1 depicted. The left side is fixed, the right side is loaded by a vertical force.

As a third validation test we perform the benchmark 'CSM1' defined in Turek and Hron [159]. It is the configuration already described in Section 3.2.2.1 (see Figure 3.1 on page 140 and Figure 3.4 on page 143), an elastic beam under load attached to a cylinder. The beam has dimensions 0.35 m $\times$ 0.02 m, it is loaded by a vertical volumetric gravity force of $g = -2\ \frac{\mathrm{m}}{\mathrm{s}^2}$, and the material parameters are Poisson ratio $\nu = 0.4$, shear modulus $\mu = 0.5$ MPa and density $\rho = 1000\frac{kg}{m^3}$. Only for this test, we use the St. Venant-Kirchhoff constitutive law, in all the other

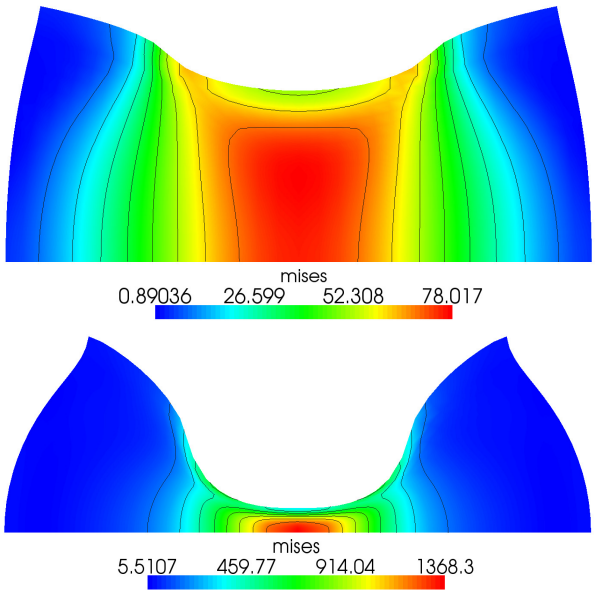


Figure 4.20: Deformed BLOCK-100 (top) and BLOCK-600 (bottom) configuration; von Mises stresses displayed.

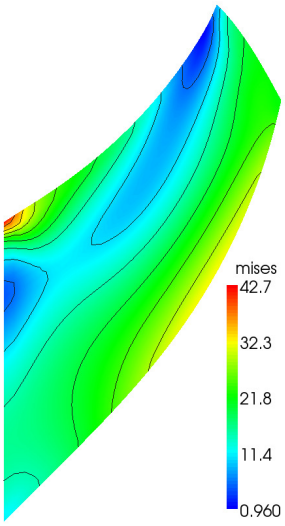


Figure 4.21: Deformed COOK configuration; von Mises stresses displayed.

tests we use Neo-Hooke material. Some of the deformed states can be seen in Figures 4.20, 4.21 and 4.22. Table 4.2 shows the vertical displacements of the point $\mathbf{A} = (0.1\ \mathrm{m}, 0.1\ \mathrm{m})$ in case of the BLOCK configuration (centre point on the upper side), those of the point $\mathbf{A} =$

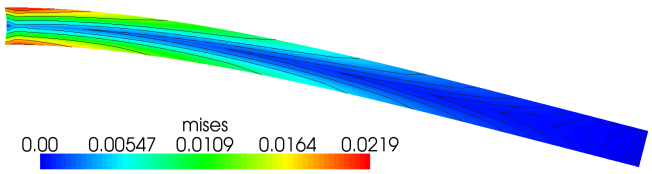


Figure 4.22: Deformed BEAM configuration; von Mises stresses displayed.

$(0.048\text{ m}, 0.06\text{ m})$ in case of the COOK configuration (upper right corner) and those of the point $\mathbf{A} = (0.35\text{ m}, 0.0\text{ m})$ in the case of the BEAM configuration (centre point on the right side). The results of the BLOCK configurations and the COOK-REG configuration correspond

# DOF	-100	-200	-300	-400	-500	-600
90	-3.3849743	-5.7094311	-7.1628177	-8.0441581	-8.5849390	-8.9300347
306	-3.3913185	-5.7189914	-7.1570128	-8.0150733	-8.5358723	-8.8662617
1.1K	-3.3914044	-5.7170012	-7.1471481	-7.9935835	-8.5021170	-8.8217846
4.3K	-3.3914167	-5.7163328	-7.1440057	-7.9868053	-8.4917313	-8.8086158
16.8K	-3.3914280	-5.7161650	-7.1431730	-7.9849962	-8.4889825	-8.8051980
66.3K	-3.3914331	-5.7161256	-7.1429624	-7.9845342	-8.4882917	-8.8043767

(a) BLOCK configuration.

# DOF	reg.	irreg.
162	1.5394614	1.5121809
578	1.5846614	1.5785159
2.2K	1.5972458	1.5959375
8.4K	1.6006008	1.6003027
33.K	1.6014680	1.6014024
132.1K	1.6016934	1.6017096

(b) COOK configuration.

# DOF	
2.3K	−6.49098
8.7K	−6.57872
33.9K	−6.60188
133.3K	−6.60801
528.6K	−6.60967
2105.9K	−6.61013

(c) BEAM configuration.

Table 4.2: Vertical displacements (in centimetres) of the BLOCK , the COOK and the BEAM configuration (in the points $\mathbf{A} = (0.1\text{ m}, 0.1\text{ m})$, $\mathbf{A} = (0.048\text{ m}, 0.06\text{ m})$, and $\mathbf{A} = (0.35\text{ m}, 0.0\text{ m})$, respectively) for different refinement levels.

to those of Reese et al. [133][19], and the results for the BEAM coincide with those obtained in Turek and Hron [159], who provide the reference value 0.0661 m for the vertical displacement. These examples validate the correctness of our computed solutions. The additional results for the COOK-IRREG configuration indicate that the solutions are qualitatively independent of the grid.

We would like to use the beam benchmark and the results found in Turek and Hron [159] to illustrate the shear locking problem described in Section 3.2.2.1. In Figure 4.23 we see the error (in percent) between the computed finite element solution and the given reference value on different refinement levels and for three different discretisations: the pure displacement formulation with the Q_1 element (six refinement levels from 2.3 k to 2.1 M DOF), the stabilised mixed $\boldsymbol{u}/p$ formulation with Q_1/Q_1 elements (six refinement levels from 3.5 k to 3.2 M DOF) and the

[19] Unfortunately, Reese et al. [133] present the results only in a graph such that the exact values are not known.

stable mixed formulation with Q_2/P_1 elements (four refinement levels from 6.5 k to 392.2 k DOF).[20] The results of the first two formulations were obtained by our program FEASTsolid,

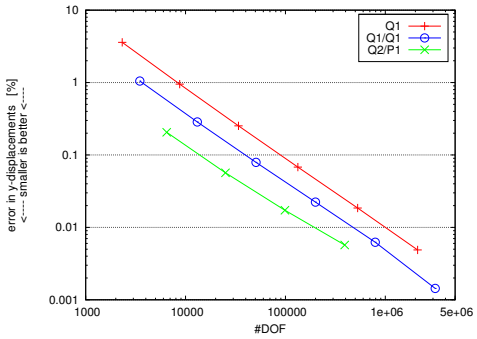


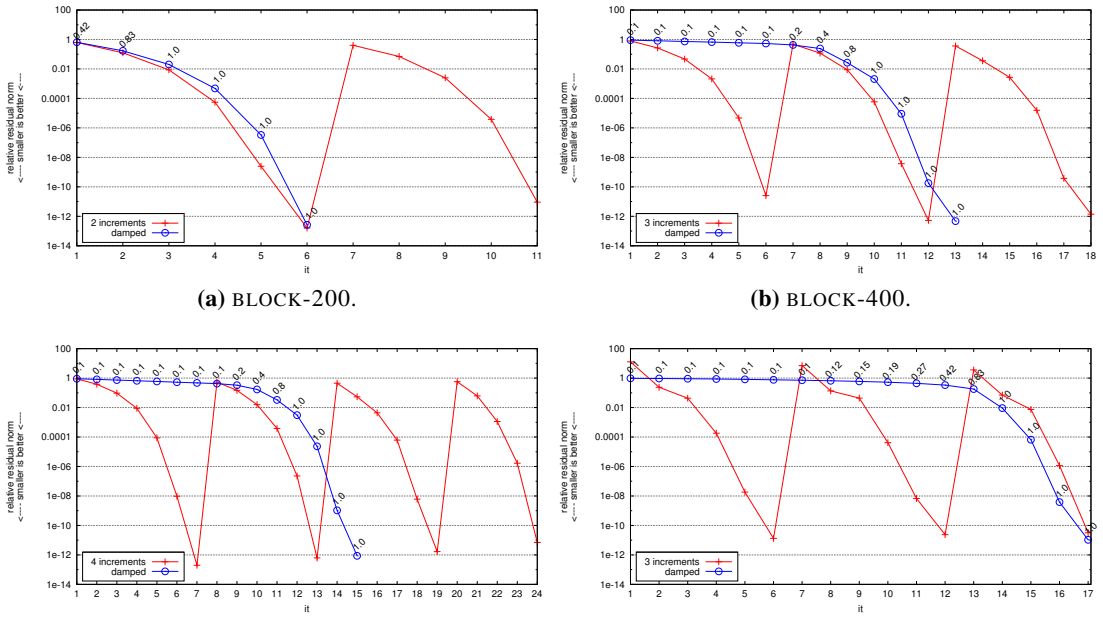
Figure 4.23: Error (in percent) of the vertical displacement in the reference point $\boldsymbol{A} = (0.35\text{ m}, 0.0\text{ m})$ for three different discretisations.

while the results of the latter formulation are taken from Turek and Hron [159].[21] We can see that about five times more DOF are needed in the case of the Q_1 discretisation to achieve the same error as Q_2/P_1, while the stabilised Q_1/Q_1 discretisation still needs roughly two times more DOF than Q_2/P_1. This confirms that the simple Q_1 formulation suffers most from the shear locking phenomenon. Applying a mixed formulations as well as using a higher order element greatly helps to alleviate the problem; simply spoken, both techniques provide 'additional DOF' to better adopt the bending deformation. Combining both techniques consequently yields the best results as the superiority of Q_2/P_1 illustrates. For the remainder of this chapter, we only consider the BLOCK and COOK configuration.

Now we compare the standard and the damped Newton-Raphson method with respect to the number of nonlinear iterations. We set $\varepsilon_{\text{rel}}^{\text{NL}} = 10^{-10}$ and $\varepsilon_{\text{abs}}^{\text{NL}} = 10^{-14}$ as stopping criterion and solve all linear systems with the direct solver UMFPACK. In case of the standard Newton-Raphson method we apply incremental loading if necessary. We present the results for the minimal number of loading steps for which the iteration converges reliably; for fewer steps the method fails or, at least, behaves very irregularly. Figure 4.24 shows the relative residual norm $\|\mathbf{r}(\mathbf{u}_i)\|/\|\mathbf{r}(\mathbf{u}_0)\|$ in the course of the iteration. The jumps in the graphs indicate the beginning of a new loading step. The graphs for the damped Newton-Raphson method are additionally labelled by the step size the line search procedure calculated. We only present the results on four of the eight configurations, i. e., the configurations BLOCK-200, BLOCK-400, BLOCK-600 and COOK-REG. The results for the three missing BLOCK configurations are in line with the three presented ones, and the results for the COOK-IRREG configuration are qualitatively the same as for the regular one. We can make the following observations in Figure 4.24:

[20] Results for the pure displacement formulation with Q_2 elements are unfortunately not available.

[21] Note that due to the compressibility of the material ($\nu = 0.4$), the mixed formulation is actually not necessary. However, Turek and Hron [159] also consider incompressible problems, so they use the mixed formulation for all tests.



(a) BLOCK-200. **(b)** BLOCK-400.

(c) BLOCK-600. **(d)** COOK-REG.

Figure 4.24: Relative residual norm of the standard and the damped Newton-Raphson method for four different configurations. The number of load increments for the standard method is given in the key.

- For the BLOCK configuration the increasing loads result in stronger nonlinearities. The problem is correspondingly harder to solve, which can be seen in the increasing number of Newton iterations and load increments.
- In all cases, the line search algorithm performs exactly one iteration, resulting in a step size less than 1.0, or no iteration, i. e., allowing the full step $s = 1.0$.
- The damping value $s = 0.1$ occurs very often. This indicates that the heuristic for finding an optimal step size actually returned a smaller value, which was then adapted due to the safeguard $\sigma_l = 0.1$ (see equation (4.26) on page 232). The subsequent values of 0.2, 0.4 and 0.8 indicate that the heuristic actually returned a larger value, which was then adapted due to the additional safeguard that bounds the increase of the step size (see equation (4.27)).
- The damped Newton-Raphson method shows typical behaviour: In the initial phase, it proceeds 'carefully' which is indicated by small step sizes. Then, when approaching the solution, it takes ever larger steps, ending with some full steps until the desired termination criterion is fulfilled.
- Both the standard and the damped Newton-Raphson method exhibit the typical quadratic convergence in the vicinity of the solution. For the damped variant this phase is accompanied by taking full steps.

- For the standard Newton-Raphson method the number of load increments has to be adapted for each configuration. The damped Newton-Raphson method, however, robustly solves all test configurations with the given set of parameters. Furthermore, it needs clearly less nonlinear iterations in most cases. Only on the COOK-REG configuration it proceeds very 'carefully' in the initial phase, such that it needs the same number of iterations as the standard method with three load increments.

- In case of the damped Newton-Raphson iteration the relative residual norm is always smaller than 1.0. For the standard method this is not the case as the COOK-REG configuration shows: The first iteration of each load increment actually increases the residual norm (in the first load increment by more than a factor of 10) resulting in a higher number of iterations. It needs even more load increments to avoid such unfavourable steps in the initial phase.

Summary

The results show that the damped Newton-Raphson method is clearly superior to the standard method, both in terms of robustness and efficiency. Especially the fact that the method works without being forced to adapt the number of load increments to the current configuration is very beneficial. On the other hand, some additional parameters have to be selected for the line search algorithm, but the employed set of established standard values [60, 92], seems to be well suited for the finite deformation elasticity problem, as well. Of course, for each configuration there is an optimal combination of parameters that yields better results than the standard values. But determining this combination *a priori* is usually not possible, and performing several parameter studies is usually more expensive than simply performing the simulation with the standard set of parameters. Only if the computation fails, one has to try different parameters in order to achieve convergence. In hard cases, it can be advantageous to combine incremental loading with line damping techniques.

4.3.3 Inexact Newton-Raphson Method

For large-scale problems the employment of direct solvers to treat the linear subproblems within the Newton-Raphson iteration becomes prohibitively expensive in terms of memory requirements and solving times. Hence, iterative methods have to be used which solve these subproblems only approximately. The nonlinear solution scheme is then referred to as **inexact Newton-Raphson method**. Hence, after the solution of the linear subproblem (see line 12 in Listing 4.3 on page 228) in the i-th nonlinear iteration we do not have

$$\hat{\mathbf{u}} = -\tilde{\mathbf{K}}(\mathbf{u}_i)^{-1}\mathbf{r}(\mathbf{u}_i),$$

but only

$$\|\tilde{\mathbf{K}}(\mathbf{u}_i)\hat{\mathbf{u}} + \mathbf{r}(\mathbf{u}_i)\| \leqslant \eta_i \|\mathbf{r}(\mathbf{u}_i)\|,$$

where η_i is the relative termination criterion of the iterative linear solver, which is also called **forcing term** in the context of inexact Newton-Raphson methods. In previous sections this value was simply denoted by ε_{rel}. The only condition the forcing term has to satisfy in order to guarantee linear convergence of the Newton-Raphson iteration (provided that the start vector is sufficiently close to the solution) is that it must be bounded away from 1. In order to achieve superlinear convergence it must fulfil $\eta_i \to 0$, and for quadratic convergence $\eta_i = O(\|\mathbf{r}(\mathbf{u}_i)\|)$ (see Kelley [92] for the definitions of the different types of convergence).

So, the question is how to choose the forcing term in practice. Setting it to a constant value $\eta_i = \eta < 1$ for the entire Newton-Raphson iteration is always feasible. However, if it chosen very small (say, $\eta = 10^{-8}$), it may lead to *oversolving* at the beginning of the Newton-Raphson iteration, which means that the linear systems are solved much more accurately than it is actually necessary to determine a sufficiently exact Newton-Raphson direction. On the other hand, if it is chosen very large (say, $\eta = 0.5$), the Newton-Raphson iteration will exhibit correspondingly slow convergence.

There are many heuristics to choose the forcing term adaptively in the course of the Newton-Raphson iteration (see, e. g., Brown and Saad [42], Dembo and Steihaug [57] and Eisenstat and Walker [60]). Their common goal is to minimise the total arithmetic work by providing a good balance between the number of inner (linear) iterations and the number of outer (nonlinear) iterations. We use the two heuristics of Eisenstat and Walker [60]. To describe them we assume that $\mathbf{u}_i = \mathbf{u}_{i-1} + s\hat{\mathbf{u}}$ is the current iterate, obtained with the help of the previous iterate and a (possibly damped) Newton-Raphson direction (see Listings 4.3 and 4.6). In the first heuristic, the forcing term is computed by

$$\eta_i = \frac{\big|\|\mathbf{r}(\mathbf{u}_i)\| - \|\mathbf{r}(\mathbf{u}_{i-1}) + s\tilde{\mathbf{K}}(\mathbf{u}_{i-1})\hat{\mathbf{u}}\|\big|}{\|\mathbf{r}(\mathbf{u}_{i-1})\|}. \tag{ADAP1}$$

It reflects how well the nonlinear function $\mathbf{r}(\mathbf{u}_{i-1} + s\hat{\mathbf{u}})$ and its linear approximation $\mathbf{r}(\mathbf{u}_{i-1}) + s\tilde{\mathbf{K}}(\mathbf{u}_{i-1})\hat{\mathbf{u}}$ agree. It guarantees two-step local quadratic convergence, i. e., in the vicinity of the solution the error norm e is reduced to e^2 within two iterations. In the second heuristic, the forcing term is computed by

$$\eta_i = \gamma\left(\frac{\|\mathbf{r}(\mathbf{u}_i)\|}{\|\mathbf{r}(\mathbf{u}_{i-1})\|}\right)^2, \tag{ADAP2}$$

with $\gamma \in [0,1]$. It guarantees local quadratic convergence and incorporates the current reduction rate of the nonlinear residual $\|\mathbf{r}(\mathbf{u})\|$. In accordance with Eisenstat and Walker [60] we choose $\gamma = 0.9$. For both heuristics the forcing term in the first nonlinear iteration is set to $\eta_1 = 0.5$.

The rationale of both heuristics is that the forcing terms will likely be large when still far away from solution, and will decrease ever faster when approaching the solution. However, also far away from the solution it might happen that the nonlinear residual function coincidentally agrees very well with its linear approximation (ADAP1) or that the nonlinear residual decreases

unexpectedly fast in one step (ADAP2). This would result in too small forcing terms and oversolving. To prevent this, Eisenstat and Walker [60] suggest to apply a safeguard of the form

$$\eta_i^{(1)} \leftarrow \begin{cases} \max\{\eta_i, s(\eta_i)\} & \text{if } s(\eta_i) > 0.1, \\ \eta_i & \text{else,} \end{cases}$$

where for the first variant the value s is given by

$$s(\eta_i) = \eta_{i-1}^{(1+\sqrt{5})/2} \tag{ADAP1}$$

and for the second variant by

$$s(\eta_i) = \gamma \eta_{i-1}^2. \tag{ADAP2}$$

The threshold 0.1 is responsible to heuristically decide whether we are already 'close enough to the solution'. In this case, we trust the computed forcing term and apply no safeguard.

It can also happen that the computed forcing term is greater than 1, so there has to be a second safeguard applying a general upper bound,

$$\eta_i^{(2)} \leftarrow \min\{\eta_{\max}, \eta_i^{(1)}\},$$

where we use $\eta_{\max} = 0.9$.

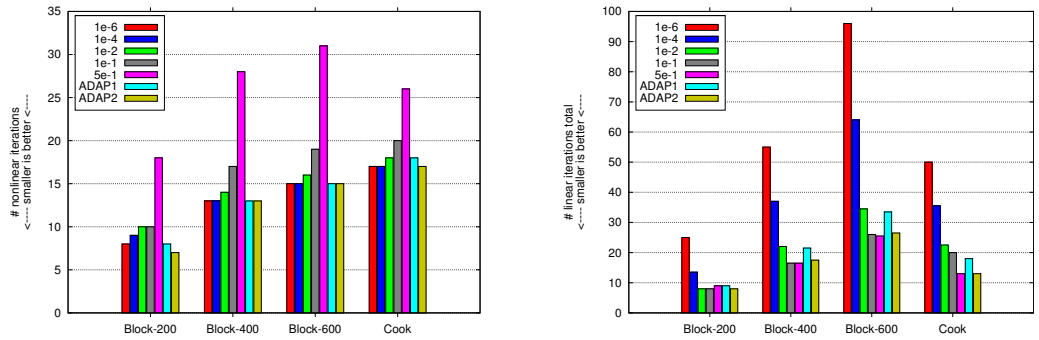
Small forcing terms in the vicinity of the solution facilitate the quadratic convergence of the inexact Newton-Raphson iteration, though, but setting them too small might also be harmful: It can be difficult for the linear solver to fulfil such small relative termination criterions, especially when the initial residual is already very small. Additionally, the current nonlinear residual norm might already be quite close to the desired tolerance, such that it does not make sense to solve the linear system too accurately. So, we apply a third safeguard to prevent too small forcing terms. We follow Kelley [92] and set

$$\eta_i \leftarrow \max\{\eta_i^{(2)}, 0.5\varepsilon / \|\mathbf{r}(\mathbf{u}_i)\|\},$$

i. e., the forcing term is bounded from below by a constant multiple (here 0.5) of the ratio of the termination threshold for the nonlinear iteration, $\varepsilon = \max\{\varepsilon_{\text{rel}}^{\text{NL}} \|\mathbf{r}(\mathbf{u}_0)\|, \varepsilon_{\text{abs}}^{\text{NL}}\}$ (see line 9 in Listing 4.3 on page 228), and the norm of the current nonlinear residual. The rationale is: When the current nonlinear residual norm is, say, $\|\mathbf{r}(\mathbf{u}_i)\| = 10^{-5}$ and for termination a norm of $\varepsilon = 10^{-6}$ is required, then it is most probably unnecessary to use a forcing term of magnitude $\eta_i = 10^{-5}$ to solve the linear system. This size would be necessary, though, to guarantee the quadratic convergence of the Newton-Raphson iteration, but to achieve the desired nonlinear residual norm of $\varepsilon = 10^{-6}$, a forcing term of only $\eta_i = 0.05$ is probably sufficient. This means, we deliberately sacrifice the quadratic convergence of the Newton-Raphson iteration in order to avoid oversolving of the linear system in the final phase and thus save a significant amount of arithmetic work.

4.3.3.1 Numerical Comparison of Different Forcing Terms

We now perform numerical tests on the four configurations we already used in Figure 4.24, applying constant forcing terms between $\eta = 0.5$ and $\eta = 10^{-6}$ on the one hand and the adaptively computed forcing term on the other hand. We use the linear solver `BiCG-MG-BSor` with the 1-layer-ScaRC solver `MG__ADI` to solve the scalar subsystems (cf. Section 4.2) and we apply line damping (see Section 4.3.2).[22] We compare the number of Newton-Raphson iterations (Figure 4.25a) and the total number of linear iterations (Figure 4.25b). Since the tangent matrix is reassembled in each nonlinear iteration, we also consider total computation times (Figure 4.26). For the BLOCK configuration we only consider refinement level 8 (1052K DOF) and for the COOK-REG configuration level 7 (526K DOF)



(a) Number of nonlinear iterations. **(b)** Total number of linear iterations.

Figure 4.25: Comparing constant and adaptive forcing terms on four different configurations.

As expected, the number of nonlinear iterations rises for larger (constant) forcing terms (Figure 4.25a). It is, however, interesting to observe that there is practically no difference between $\eta = 10^{-6}$ and $\eta = 10^{-4}$. Also the increase from $\eta = 10^{-4}$ to $\eta = 0.01$ is relatively small compared to the two subsequent jumps from $\eta = 0.01$ to $\eta = 0.1$ and from $\eta = 0.1$ to $\eta = 0.5$. Since smaller forcing terms result in more arithmetic work per nonlinear iteration, it is clear already now that values of $\eta = 10^{-4}$ or smaller render the overall solution method unnecessarily expensive. This is confirmed when considering Figure 4.25b: The accumulated number of linear iterations for the forcing terms $\eta = 10^{-6}$ and $\eta = 10^{-4}$ is 1.5 to 3 times higher than for $\eta = 0.01$. These large differences cannot be compensated by slightly shorter assembly times.

Hence, the question is now which of the remaining five forcing terms ($0.01, 0.1, 0.5$, ADAP1 or ADAP2) is most efficient. The choice $\eta = 0.5$ results in the smallest number of linear iterations,

[22] Note, that the use of approximate iterative linear solvers can negatively influence the robustness of the line search algorithms in some cases. Depending on the eigenvalues and the condition of the system matrix, the method can behave counterintuitively in the sense that a more accurate linear solution may degrade the robustness of the backtracking procedure, eventually leading to failure of the method. See, e. g., Cai and Keyes [46], Hwang [84], Tuminaro et al. [157] for details in this regard and for alternative approaches. However, for the numerical examples in this section such problems do not occur.

though, but also in the – by far – largest number of nonlinear iterations. The other four are closer together, such that we have to look at the total timings in Figure 4.26 to better assess the situation. The figure shows the total runtimes of serial computations (performed on an

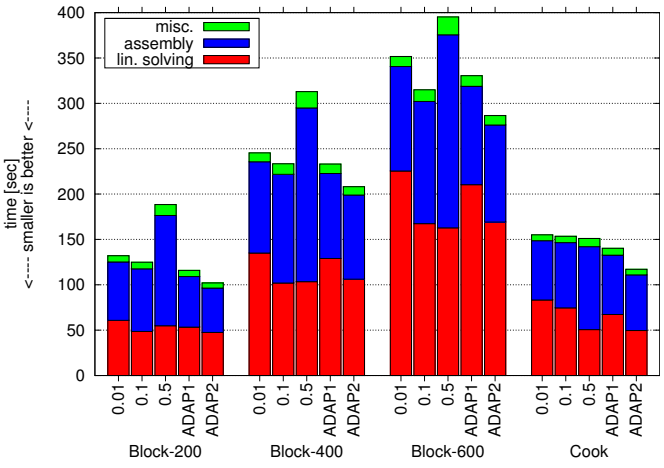


Figure 4.26: Runtimes on four configurations for five different forcing terms.

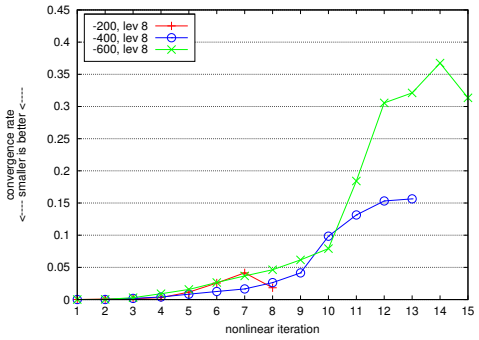
AMD Opteron DP 844 CPU with 1.8 GHz), consisting of linear solving times, assembly times for residual vectors and tangent matrices, and times for the remaining operations within the Newton-Raphson iteration (linear solver initialisations, norm calculations, adaptive tolerance heuristic, line damping, etc.). In most cases, the choice $\eta = 0.5$ indeed yields the shortest linear solving times, but also the longest assembly times, such that, in summary, it is clearly the worst choice for all three BLOCK configurations. For the COOK-REG configuration, however, the differences of nonlinear iteration numbers for the different forcing terms are not as significant as for the BLOCK configurations (see Figure 4.25a), such that the additional assembly times for the choice $\eta = 0.5$ do not have such an impact, and $\eta = 0.5$ turns out to be the third best choice. The forcing term $\eta = 0.01$ leads to higher total computation times than $\eta = 0.1$ in all cases which is due to the higher linear solving costs. Comparing $\eta = 0.1$ and the first adaptive forcing term ADAP1 shows that it depends on the configuration which of the two is superior: For the BLOCK-200 and the COOK-REG configuration the adaptive forcing term yields slightly better results, for BLOCK-400 the runtimes are equal and only for BLOCK-600 the constant choice $\eta = 0.1$ actually beats the adaptive forcing term ADAP1. For the second adaptive forcing term ADAP2, the results are clearer: It performs best in all four tests. Looking back at Figure 4.25, the reason becomes apparent: It yields the smallest number of nonlinear and linear iterations in nearly every case. It is indeed able to combine the advantage of small forcing terms (fast convergence of the outer Newton-Raphson iteration) with the advantage of large forcing terms (low arithmetic costs of the inner linear iterations).

We can summarise that heuristics to adaptively determine the forcing term can be beneficial, but for the kind of nonlinear problems we are dealing with they seem not to be mandatory. Simply choosing the constant forcing term $\eta = 0.1$ seems to be a reasonable choice and has the additional advantage that no extra parameters (safeguards etc.) have to be provided. So,

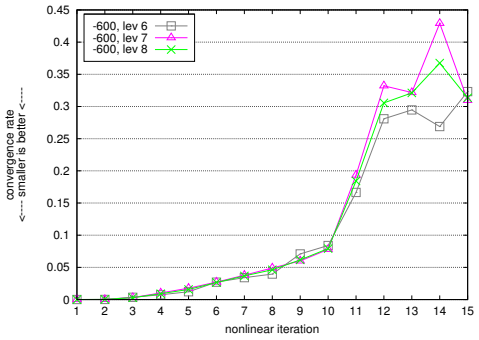
while line damping is a mandatory technique to significantly increase the robustness of the Newton-Raphson method, forcing term heuristics do usually not decide over success or failure of the method, but merely aim at increasing its efficiency. Whether they are successful in this regard, can depend on the special configuration as the results for ADAP1 illustrate. The second variant ADAP2 behaves clearly better in our tests, which, of course, does not mean that this must always be the case. A finite element software package should offer such heuristics for the solution of nonlinear equations, but the decision when to apply them certainly requires some user experience.

4.3.3.2 Linear Solver Efficiency

Finally, we want to briefly examine how the linear solver efficiency is affected by increasing nonlinearities. Therefore, we consider the BLOCK configuration and take a closer look at the linear solves within the damped Newton-Raphson iteration using the constant forcing term of $\eta = 10^{-6}$. This was the most expensive configuration in the tests above, though, but is best suited to assess the linear solver behaviour. Figure 4.27 shows the convergence rates of the inner linear solves for each outer nonlinear iteration. In Figure 4.27a we compare the level 8



(a) Level 8 (1052K DOF) computations with varying loads.



(b) BLOCK-600 with varying refinement levels.

Figure 4.27: Convergence rates of the linear solver in the course of the Newton-Raphson iteration for the BLOCK configuration.

(1052K DOF) results for the BLOCK under three different loadings. One can clearly see that the performance of the linear solver deteriorates – as expected – due to the increasing nonlinearities. Figure 4.27b, however, indicates that the solver behaviour is – aside from some irregularities – in principle level independent. We also performed these tests with other linear solvers from Section 4.2.6 (results not presented here), and found that the degradation due to increasing nonlinearities is qualitatively the same, sometimes even worse than for the solver used above. Hence, our choice of `BiCG-MG-BSor` as favourite solver (see Section 4.2.8) is also justified for the nonlinear elasticity problems considered here.

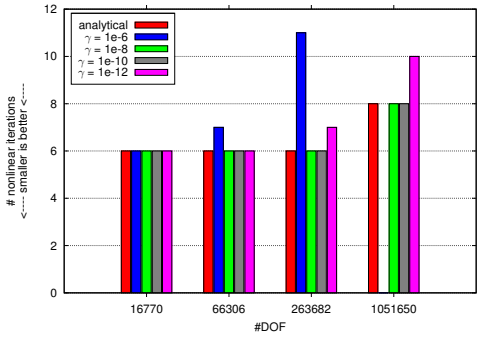
4.3.3.3 2-layer-ScaRC Solvers and Nonlinearities

We want to briefly comment on a further test we performed, but which we do not present here in detail. In Section 4.2.7.5 we compared 1-layer-ScaRC and 2-layer-ScaRC solvers in their role as preconditioners for elasticity solvers and found that for certain configurations exhibiting high element aspect ratios 2-layer-ScaRC can be superior to 1-layer-ScaRC. An interesting question is whether 2-layer-ScaRC can be advantageous with respect to nonlinearities. So, we repeated the tests above (BLOCK-200, BLOCK-400, BLOCK-600 and COOK-REG) using the 1-layer-ScaRC solver `MG(1e-1)-FGMRES4__JAC__D` and the 2-layer-ScaRC solver `MG(1e-1)-FGMRES4__MG-BICG-JAC-D__D` from Section 4.2.7.5 for the scalar solves within the block SOR preconditioner. The result was disappointing: For all four configurations, the 2-layer-ScaRC solver was about 10 times more expensive than the 1-layer-ScaRC solver in terms of arithmetic work. For the COOK-REG configuration, the 2-layer-ScaRC solver performed about 20% less iterations than 1-layer-ScaRC (which means a corresponding reduction of communication in a parallel computation), for the BLOCK-600 configuration about 15% less iterations, while on the two remaining BLOCK configurations the iteration numbers were equal. These reductions are clearly not sufficient to compensate for the significantly higher arithmetic costs, not even in a massively parallel computing environment. Of course, this does not mean that the 2-layer-ScaRC concept is useless for nonlinear simulations; as soon as mesh anisotropies come into play it will tap its potential again. However, there seems to be no indication that 2-layer-ScaRC can handle the nonlinearities themselves significantly better than 1-layer-ScaRC. A final assessment, however, requires more thorough studies, which has to be part of future work.

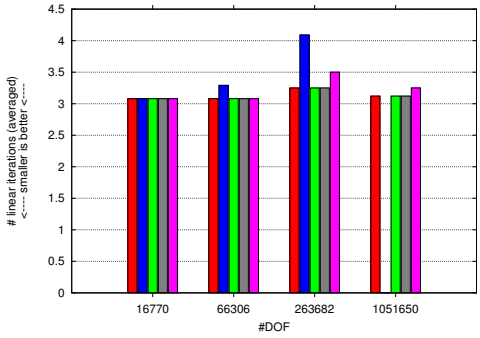
4.3.4 Numerical Tangents

In Section 3.2.4.3 we explained how the tangent matrix can be computed with the help of finite differences (see equation (3.98) on page 151). The question arises how to choose the perturbation parameter γ. Miehe [112] suggests to set it to the square root of the machine precision $\varepsilon_{\text{prec}}$ as to perturb 'half of the digits', i. e., $\gamma = \sqrt{\varepsilon_{\text{prec}}} = 10^{-8}$ in double precision arithmetic (also see Kelley [92]). We test this suggestion with the help of some numerical examples, where we vary the perturbation parameter between $\gamma = 10^{-2}$ and $\gamma = 10^{-12}$. We use the results obtained with the analytically computed tangent matrix as reference. We apply line damping, and the linear systems are solved iteratively gaining 6 digits.

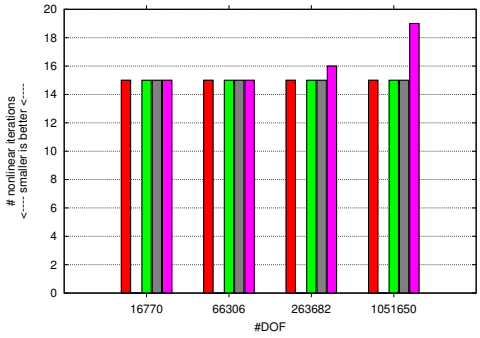
In Figure 4.28 we compare the number of nonlinear iterations and the average number of linear iterations per nonlinear step on three of the configurations used in the previous sections. Setting the perturbation parameter to $\gamma = 10^{-2}$ or $\gamma = 10^{-4}$ the Newton-Raphson method fails in all tests, so these values do not appear in the plots. Also a value of $\gamma = 10^{-6}$ is not very robust: For the BLOCK-600 configuration the solver fails on all levels (Figure 4.28c), while for the other two configuration it only converges on smaller grid levels (Figures 4.28a and 4.28e.) The perturbation parameters $\gamma = 10^{-8}$ and $\gamma = 10^{-10}$ yield equal nonlinear iteration counts, which exactly coincide with those of the analytical tangent. For $\gamma = 10^{-12}$ the Newton-Raphson



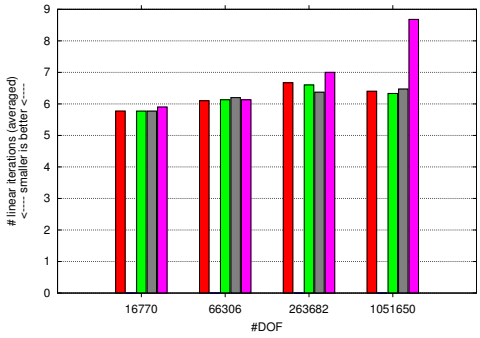
(a) BLOCK-200, nonlinear iterations.



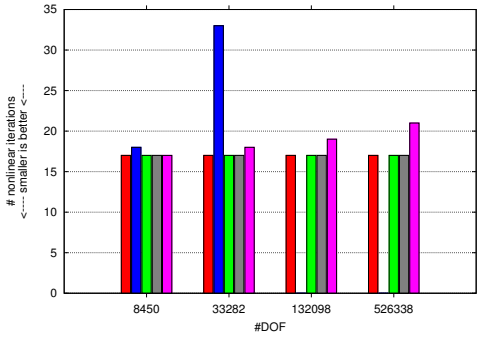
(b) BLOCK-200, linear iterations (averaged).



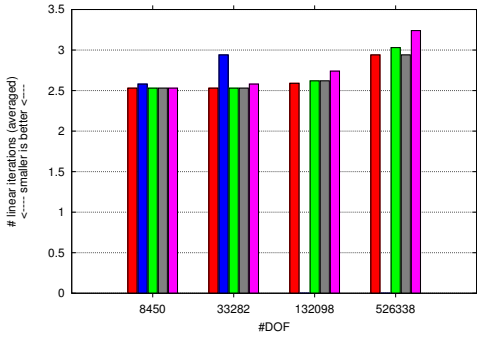
(c) BLOCK-600, nonlinear iterations.



(d) BLOCK-600, linear iterations (averaged).



(e) COOK-REG, nonlinear iterations.



(f) COOK-REG, linear iterations (averaged).

Figure 4.28: Number of nonlinear iterations (left) and average number of linear iterations per nonlinear step (right) for different perturbation parameters γ on three configurations; reference iteration counts for the analytically computed tangent matrix. All plots share the colour key of plot 4.28a.

convergence deteriorates on higher refinement levels, which indicates that this value is too small. Hence, we can confirm the suggestion of Miehe [112] to set the perturbation parameter to the square root of the machine precision; slightly smaller values ($\gamma \in [10^{-10}, 10^{-8}]$) seem to be feasible, as well.

Considering the right hand side plots 4.28b, 4.28d and 4.28f, these findings are confirmed: The average number of linear iterations per nonlinear step for the two perturbation parameters $\gamma = 10^{-8}$ and $\gamma = 10^{-10}$ agree very well with those for the analytical tangent, there are only slight deviations. This means that – at least for the examples considered here – the linear solver performance is not negatively affected when using numerically computed tangent matrices and when the perturbation parameter is chosen correctly. Unsuited values like $\gamma = 10^{-12}$, however, clearly impair the linear solver performance as the higher linear iteration counts show.

4.3.5 Parallel Efficiency

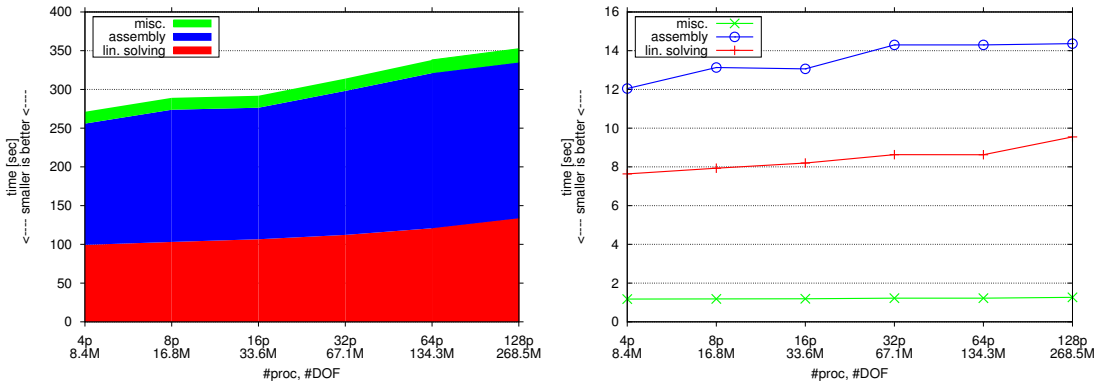
We now present one numerical example that demonstrates the good parallel efficiency of the overall nonlinear solver. We consider the six macro decompositions of the BLOCK configuration which we already used for the linear parallel computations in Section 4.2.7.6 (see Figure 4.17 on page 223). We refine the subdomains ten times and apply a surface load of $f = -400$. The nonlinear problem is solved with the damped Newton-Raphson method (see Section 4.3.2), using a stopping criterion of $\varepsilon_{\text{rel}} = 10^{-6}, \varepsilon_{\text{abs}} = 10^{-12}$. The linear problems are solved by `BiCG-MG-BSor` gaining one digit and using the 1-layer-ScaRC solver `MG__ADI` for the scalar subsystems (cf. Section 4.2). The computations are performed on the compute cluster LiDO (see Section 4.2.7.6 for details).

Note that the scalar multigrid solver applies a block wise working smoother, such that the varying number of subdomains of the six BLOCK configurations leads to slight variations in the linear solver behaviour. The nonlinear damped Newton-Raphson method is quite sensitive to such perturbations. Furthermore, the flat block (4, 16 and 64 subdomains) evolves different deformations and nonlinearities than the higher block (8, 32 and 128 subdomains). Hence, there are slight variations in the number of linear and nonlinear iteration counts. Table 4.3 lists the number of nonlinear iterations, the total number of linear iterations and the average number of linear iterations per nonlinear iteration. One can see that the iteration counts slightly

# proc	# DOF	nonlinear	linear	linear av.
4	8.4M	13	10	0.77
8	16.8M	13	10	0.77
16	33.6M	13	10	0.77
32	67.1M	13	10	0.77
64	134.3M	14	10.5	0.75
128	268.5M	14	11	0.79

Table 4.3: Number of nonlinear iterations and the total and average number of linear iterations for the solution of the BLOCK-400 configuration. (Note that an early exit in the first iteration of the linear BiCGstab solver is counted as 0.5 iterations.)

increase for the two largest configurations. This is why we present the timing results in two plots: Figure 4.29a shows the total runtime in stacked form, composed of linear solving time, assembly time and time for remaining operations. Figure 4.29b shows the distinct runtimes (not



(a) Total runtimes (stacked). **(b)** Runtimes per nonlinear iteration (not stacked).

Figure 4.29: Parallel runtimes for the solution of the BLOCK-400 configuration with different grid partitions.

stacked), scaled with the number of nonlinear iterations in order to better assess the scalability of the method. Our main focus are the linear solving times. One can see in Figure 4.29b that the increase of the linear solving time from 4 to 128 processors corresponds to the increase we observed for the linearised elasticity case in Figure 4.18c on page 225. Only the slope from 64 to 128 processors is a little bit steeper, which, however, can be explained by the fact that the linear solver performs a half iteration more on the 128 processor configuration, while the number of nonlinear iterations is equal (see Table 4.3). The slope would correspond to that of the remaining curve, if the linear solving times were scaled with the number of linear iterations. Hence, we can state that the good weak scalability of our linear elasticity solvers transfers to the nonlinear case.[23]

4.3.6 Summary and Further Possible Improvements

In this section we demonstrated that FEASTsolid is able to efficiently solve nonlinear finite elasticity problems in the pure displacement formulation. Our basic concept of reducing multivariate operators and operations to scalar ones, fully applies to nonlinear solution methods like Newton-Raphson, as well. We confirmed that two popular techniques for improving Newton-Raphson methods are also successful in the context of solid mechanical problems: Line damping significantly increases the robustness of the Newton-Raphson method, while heuristics to adaptively choose the forcing terms in inexact Newton-Raphson methods help to lower the

[23] The increase of the assembly times, however, is unexpected and can not be explained currently. While the linear solver has to communicate very often in the course of the iteration, this is not the case for the assembly routine: It works element-wise and thus performs purely local calculations; only after the assembly process is finished, there has to be one data exchange between neighbouring subdomains. Hence, the ratio between arithmetic work and amount of communication is much more favourable than for the case of linear solving, and we actually expect better scalability properties. However, since the focus of this thesis is on linear solvers and we have not thoroughly examined the assembly process for performance bottlenecks and possible improvements by now, we ignore this issue here.

arithmetic costs. The iterative solvers introduced in Section 4.2 are able to solve the linearised systems arising in the course of the nonlinear iteration, even for heavily deformed configurations. However, a dependence on the strength of the nonlinearities is observable. The numerical examples we considered do not indicate that 2-layer-ScaRC solvers are better qualified than 1-layer-ScaRC solvers to deal with the occurring nonlinearities. The good parallel efficiency of our linear elasticity solvers is confirmed in the nonlinear setting.

We want to mention two further important techniques to improve the robustness and the efficiency of the Newton-Raphson method, which are not covered in detail in this thesis. The first technique, the so-called *multilevel Newton-Raphson method*, is concerned with finding a suitable start solution which can be essential for the succes of the overall method. The idea is to first solve the problem on a coarser grid level and to use the obtained solution as start solution for the nonlinear problem on the finest grid level. Doing so recursively over all grid levels leads to the algorithm outlined in Listing 4.7. The goal of this multilevel approach is to save some

```
1  subroutine newtonRaphsonMultilevel(f^in, f^ex, u, ε^NL_rel, ε^NL_abs)
2    ! Input:  internal forces f^in (as a function of u),
3    !         external force vector f^ex,
4    !         start vector u,
5    !         rel. and abs. tolerance ε^NL_rel, ε^NL_abs,
6    ! Output: solution of the equation f^in(u) - f^ex = 0
7
8    ! loop over all levels
9    do l = 1,2,...,L
10     if (l .eq. 1) then
11       ! on coarsest level, use zero start solution
12       u^(l) = 0
13     else
14       ! on finer levels, use the prolongated coarser grid solution
15       ! as start solution
16       u^(l) ← P^(l) u^(l-1)
17     endif
18     ! call the Newton-Raphson algorithm to solve the level l problem
19     ! with the current u^(l) as start vector
20     u^(l) ← newtonRaphson((f^in)^(l), (f^ex)^(l), u^(l), ε^NL_rel, ε^NL_abs)
21   enddo
22 end subroutine newtonRaphsonMultilevel
```

Listing 4.7: Multilevel Newton-Raphson method.

expensive Newton-Raphson iterations on the finest level by performing much cheaper iterations on the coarser grid levels, such that the total arithmetic costs are finally lower than those of the standard one-level Newton-Raphson method. The costs can be further decreased since the auxiliary problems on the coarser levels usually do not have to be solved as accurately as the actual problem on the finest level. Furthermore, the multilevel technique can help to solve very hard nonlinear problems for which the damped inexact Newton-Raphson method fails: On the one hand, the linear problems on coarser grid levels can be solved with a direct solver which elimi-

nates the possibility of failure of iterative solvers. On the other hand, one can apply additional strategies on the coarser grid levels to ensure convergence that would be very expensive on the finest level, e. g., load incrementing or arc-length continuation methods. If the problem on the coarser level is solved successfully, its solution might be good enough a start solution for the finest level problem, so that the latter can be solved with standard techniques again.

The second technique for increasing the efficiency of Newton-Raphson is the so-called *Shamanskii method* (see, e. g., Kelley [92]). It aims at the reduction of matrix assembly times. Instead of recomputing the tangent matrix in each Newton-Raphson step, it is used for m steps. For $m = 1$ this approach coincides with the standard Newton-Raphson method; for '$m = \infty$', the algorithm is also called the *chord method*, in which the entire nonlinear iteration is performed with the help of the initially computed tangent matrix. The rationale of the Shamanskii method is that with a suitably chosen m the nonlinear convergence is only mildly affected, while assembly times can be reduced significantly. However, the optimal choice of the parameter m clearly depends on different factors, as for example the ratio between assembly and linear solving times. Hence, the optimal value is usually not known and is hard to estimate a priori (especially when an adaptive forcing term heuristic is applied) such that user experience or some adaptive mechanism is necessary to find appropriate values.

5 Multilevel Saddle Point Solvers for (Nearly) Incompressible Elasticity Problems

In this chapter we extend the solution technique described in Chapter 4 to the important class of elasticity problems with (nearly) incompressible material, discretised by the mixed displacement/pressure formulation.

In Section 5.1 we introduce different kinds of saddle point solvers and explain how FEAST's ScaRC solvers are incorporated. In Section 5.2 we thoroughly compare the presented solvers in terms of numerical examples. For these solver studies we confine ourselves to the case of linearised elasticity and only briefly comment on the nonlinear finite deformation case in Section 5.3, where we also summarise our results.

5.1 Solvers and Preconditioners for Saddle Point Problems - A Brief Overview

The need to solve linear systems in saddle point form arises in many technical and scientific applications, where the mixed finite element method in solid and, especially, fluid mechanics is probably one of the most prominent. There is a great variety of solution and preconditioning methods for saddle point systems and the amount of literature on the topic is vast. Hence, an exhaustive survey is out of the scope of this thesis. We refer to the article of Benzi et al. [21] for an overview of applications and solution methods and for pointers to the literature on this topic. We confine ourselves to a brief description of some of the most prominent methods to solve saddle point systems arising from the mixed finite element discretisation of solid or fluid mechanical equations. For more detailed presentations in this regard see, e. g., the monographs of Brezzi and Fortin [39] and Elman et al. [65].

We want to recall the linear saddle point system which is to be solved (cf. equation (3.80) on page 146 and equation (3.107) on page 156 and their derivations). With the notations

$$\mathcal{A} := \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{pmatrix}, \qquad \mathbf{x} := \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix}, \qquad \mathbf{b} := \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix},$$

the system can be simply denoted as

$$\mathcal{A}\mathbf{x} = \mathbf{b}. \tag{5.1}$$

The lower right matrix block $\mathbf{C}$ is the sum of two matrices,

$$\mathbf{C} = \mathbf{C}^{\mathrm{c}} + \mathbf{C}^{\mathrm{s}},$$

where $\mathbf{C}^{\mathrm{c}}$ contains the compressibility terms and $\mathbf{C}^{\mathrm{s}}$ is the stabilisation matrix. In the case of the incompressible limit, we have $\mathbf{C}^{\mathrm{c}} = \mathbf{0}$, and in the case stable finite element pairs are used, we have $\mathbf{C}^{\mathrm{s}} = \mathbf{0}$. When both hold, then we obtain the standard saddle point form with a zero block $\mathbf{C} = \mathbf{0}$. $\mathbf{C}^{\mathrm{c}}$ is a (scaled) negative pressure mass matrix and is thus symmetric negative definite. For the PPP-type stabilisations, $\mathbf{C}^{\mathrm{s}}$ represents an approximation of the (scaled) negative Laplacian matrix for pure Neumann boundary conditions and is thus also symmetric negative semidefinite. The latter property also holds in the case of the PROJ stabilisation [27]. Hence, the matrix $\mathbf{C}$ is symmetric negative semidefinite.[1] The matrix $\mathbf{A}$ is symmetric positive definite.

We want to describe some of the most popular solution methods for saddle point problems, where we distinguish between two main categories: *segregated* (or *decoupled*) methods (Section 5.1.1) and *coupled* methods (Section 5.1.3).

Beside the solvers and preconditioners we describe in the following sections, there are many further techniques for treating saddle point problems, e. g., pressure convection-diffusion (PCD) preconditioners [91, 144], least-squares commutator (LSC; also called BFBt) preconditioners [66, 67], an improved version of the BFBt preconditioner [119], preconditioning by regularisation [6], primal-based penalty preconditioners [58], augmented Lagrangian approaches [20], saddle point ILU-type preconditioners [163] and hierarchical matrix preconditioners [104]. It is out of the scope of this thesis to describe and compare all these approaches. For general surveys and/or comparisons, see, e. g., Benzi et al. [21], de Niet and Wubs [56], Elman et al. [65], ur Rehman et al. [163].

5.1.1 Segregated Methods

5.1.1.1 Uzawa and Pressure Schur Complement Methods

Segregated solution methods decouple the two unknown vectors $\mathbf{u}$ and $\mathbf{p}$ and reduce the solution of the overall system (5.1) to the solution of smaller subsystems. Formally, this strategy is based on the block LU decomposition

$$\mathcal{A} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{B}^{\mathsf{T}}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & -\mathbf{S} \end{pmatrix},$$

where the matrix

$$\mathbf{S} := \mathbf{B}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C} \tag{5.2}$$

[1] It is also possible, to scale the continuity equation by -1. Then, the whole saddle point system is positive semidefinite, but not symmetric anymore.

is the negative **(pressure) Schur complement** of the matrix $\mathcal{A}$. The saddle point system (5.1) can then be rewritten as

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{B}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{f} \end{pmatrix}. \tag{5.3}$$

A simple 'two-step algorithm' for solving system (5.3) would be to successively solve the two reduced systems

$$\mathbf{S}\mathbf{p} = \mathbf{B}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{f}, \tag{5.4a}$$

$$\mathbf{A}\mathbf{u} = \mathbf{f} - \mathbf{B}\mathbf{p}. \tag{5.4b}$$

This strategy, however, is not practicable since the inverse of the Schur complement matrix $\mathbf{S}$ is hard to compute: Usually, $\mathbf{S}$ is available only implicitly and is furthermore (due to the factor $\mathbf{A}^{-1}$) not a sparse matrix. Hence, to avoid the direct inversion of $\mathbf{S}$, the algorithm is performed in an iterative way,

$$\mathbf{u}_{k+1} = \mathbf{A}^{-1}(\mathbf{f} - \mathbf{B}\mathbf{p}_k), \tag{5.5a}$$

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \gamma(\mathbf{B}^{\mathsf{T}}\mathbf{u}_{k+1} + \mathbf{C}\mathbf{p}_k), \tag{5.5b}$$

where γ is a positive parameter. This iteration is the **classical Uzawa algorithm** [3] (also see Listing 5.1). One can formally remove the variable $\mathbf{u}_{k+1}$ from the iterative scheme,

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \gamma(\mathbf{B}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{f} - (\mathbf{B}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C})\mathbf{p}_k), \tag{5.6}$$

which shows that the Uzawa algorithm is a Richardson method for iteratively solving equation (5.4a). In this form, the algorithm is also called **pressure Schur complement (PSC) method** [158]; also see Listing 5.2, where the auxiliary vector $\mathbf{z}$ is merely introduced to emphasise that the iteration is indeed independent of the vector $\mathbf{u}$. Although the Uzawa and the PSC method are formally equivalent, their practical application slightly differs: The Uzawa algorithm uses the residual of the whole system (5.1) for determining the stopping criterion, while

```
1  subroutine uzawa(A, b, x, ε)
2    !Input:      saddle point matrix A, RHS vector b = (f,0)^T,
3    !            start vector x = x_0 = (u_0,p_0)^T, rel. stopping parameter ε
4    !Output:     solution vector x = (u,p)^T
5    δ = ‖r_0‖ = ‖b − A x_0‖          ! initial residual norm
6    do k = 0,1,...
7      u_{k+1} = A^{-1}(f − B p_k)
8      p_{k+1} = p_k + γ(B^T u_{k+1} + C p_k)
9      r_{k+1} = b − A x_{k+1}
10     if (‖r_{k+1}‖ ⩽ εδ)
11       exit
12     endif
13   enddo
14 end subroutine uzawa
```

Listing 5.1: Classical Uzawa algorithm for solving the saddle point system (5.1).

```
1  subroutine psc(𝒜, b, x, ε)
2    !Input:       saddle point matrix 𝒜, RHS vector b = (f, 0)^T,
3    !             start vector x = x_0 = (u_0, p_0)^T, rel. stopping parameter ε
4    !Output:      solution vector x = (u, p)^T
5    z = A^{-1}(f − B p_0)
6    δ = ‖r_0‖ = ‖B^T z + C p_0‖          !initial residual norm
7    do k = 0, 1, ...
8      p_{k+1} = p_k + γ r_k
9      z = A^{-1}(f − B p_{k+1})
10     r_{k+1} = B^T z + C p_{k+1}
11     if (‖r_{k+1}‖ ⩽ εδ)
12       exit
13     endif
14   enddo
15   u = z
16 end subroutine psc
```

Listing 5.2: Pressure Schur complement method for solving the saddle point system (5.1).

the PSC algorithm uses that of the (scalar) Schur complement system (5.4a). Hence, when the user applies a stopping criterion like 'gain six digits', the solutions of the two methods can actually differ.

The main disadvantage of the classical Uzawa algorithm is that it requires the exact (or, at least, very accurate) computation of the inverse of the matrix $\mathbf{A}$ in each iteration[2], which is prohibitively expensive for large-scale simulations. Fortunately, this is easy to repair by transforming system (5.5a) into a preconditioned defect correction, where the preconditioner for the matrix $\mathbf{A}$ is denoted by $\tilde{\mathbf{A}}$. Usually, $\tilde{\mathbf{A}}$ represents an approximate solve by some inner iterative method. Introducing at the same time a preconditioner $\tilde{\mathbf{S}}$ for the Schur complement matrix $\mathbf{S}$, leads to the well-known **inexact Uzawa algorithm** [37]:

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \tilde{\mathbf{A}}^{-1}(\mathbf{f} - \mathbf{A}\mathbf{u}_k - \mathbf{B}\mathbf{p}_k), \tag{5.7a}$$

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \tilde{\mathbf{S}}^{-1}(\mathbf{B}^\mathsf{T}\mathbf{u}_{k+1} + \mathbf{C}\mathbf{p}_k). \tag{5.7b}$$

Obviously, the algorithm coincides with the classical Uzawa algorithm if $\tilde{\mathbf{A}}^{-1} = \mathbf{A}^{-1}$ and $\tilde{\mathbf{S}}^{-1} = \gamma\mathbf{I}$ (cf. Listing 5.3).[3] Due to the exact defect computation with the matrix $\mathbf{A}$ in equation (5.7a), the algorithm is guaranteed to converge (if it converges at all) to the correct solution, regardless of how well $\tilde{\mathbf{A}}^{-1}$ approximates the inverse $\mathbf{A}^{-1}$. Of course, the choice of $\tilde{\mathbf{A}}^{-1}$ decides over the convergence speed and whether the algorithm converges at all. When the preconditioner $\tilde{\mathbf{A}}^{-1}$ is realised in terms of a *nonlinear* iterative solver (e. g., a Krylov method), then the overall scheme

[2] This can be seen very easily, when replacing $\mathbf{A}^{-1}$ by $\tilde{\mathbf{A}}^{-1}$, representing an approximate solution of the corresponding systems. In this case, the algorithm does simply not 'know' the actual matrix $\mathbf{A}$ and is, hence, not able to compute a correct solution. In other words, the matrix vector multiplication with the Schur complement matrix $\mathbf{S}$ can not be performed exactly.

[3] Sometimes, the version (5.7) is called the *preconditioned inexact Uzawa algorithm*, and only the version with $\tilde{\mathbf{S}}^{-1} = \gamma\mathbf{I}$ is referred to as *inexact Uzawa algorithm* [62].

```
1  subroutine uzawaInexact(A, b, x, ε)
2    !Input:       saddle point matrix A, RHS vector b = (f, 0)^T,
3    !             start vector x = x_0 = (u_0, p_0)^T, rel. stopping parameter ε
4    !Output:      solution vector x = (u, p)^T
5
6    δ = ||r_0|| = ||b − A x_0||            ! initial residual norm
7    do k = 0, 1, ...
8      u_{k+1} = u_k + Ã^{-1}(f − A u_k − B p_k)
9      p_{k+1} = p_k + S̃^{-1}(B^T u_{k+1} + C p_k)
10     r_{k+1} = b − A x_{k+1}
11     if (||r_{k+1}|| ⩽ εδ)
12       exit
13     endif
14   enddo
15 end subroutine uzawaInexact
```

Listing 5.3: Inexact Uzawa algorithm for solving the saddle point system (5.1).

is referred to as **nonlinear inexact Uzawa method**. Corresponding to algorithm (5.7), one can also define an inexact version of the pressure Schur complement method (see Listing 5.4). The

```
1  subroutine pscInexact(A, b, x, ε)
2    !Input:       saddle point matrix A, RHS vector b = (f, 0)^T,
3    !             start vector x = x_0 = (u_0, p_0)^T, rel. stopping parameter ε
4    !Output:      solution vector x = (u, p)^T
5    u_1 = u_0 + Ã^{-1}(f − A u_0 − B p_0)
6    δ = ||r_0|| = ||B^T u_1 + C p_0||      !initial residual norm
7    do k = 0, 1, ...
8      p_{k+1} = p_k + S̃^{-1} r_k
9      u_{k+2} = u_{k+1} + Ã^{-1}(f − A u_{k+1} − B p_{k+1})
10     r_{k+1} = B^T u_{k+2} + C p_{k+1}
11     if (||r_{k+1}|| ⩽ εδ)
12       exit
13     endif
14   enddo
15 end subroutine pscInexact
```

Listing 5.4: Inexact pressure Schur complement method for solving the saddle point system (5.1).

preconditioned version of the exact PSC iteration (5.6) is given by

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \tilde{\mathbf{S}}^{-1}(\mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{f} - (\mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C})\mathbf{p}_k), \tag{5.8}$$

where $\tilde{\mathbf{S}}$ again denotes the preconditioner for the Schur complement matrix $\mathbf{S}$. In Section 5.1.2 we describe practical choices for the preconditioners $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{S}}$.

The inexact Uzawa algorithm (5.7) can be equivalently formulated as follows,

$$\begin{pmatrix}\mathbf{u}_{k+1}\\ \mathbf{p}_{k+1}\end{pmatrix} = \begin{pmatrix}\mathbf{u}_k\\ \mathbf{p}_k\end{pmatrix} + \begin{pmatrix}\tilde{\mathbf{A}} & \mathbf{0}\\ \mathbf{B}^\mathsf{T} & -\tilde{\mathbf{S}}\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{f} - \mathbf{A}\mathbf{u}_k - \mathbf{B}\mathbf{p}_k\\ \mathbf{0} - \mathbf{B}^\mathsf{T}\mathbf{u}_k - \mathbf{C}\mathbf{p}_k\end{pmatrix}, \tag{5.9}$$

i. e., as a Richardson iteration applied to the whole system and preconditioned by the **lower block-triangular preconditioner**

$$\tilde{\mathcal{P}}_L := \begin{pmatrix} \tilde{\mathbf{A}} & \mathbf{0} \\ \mathbf{B}^\mathsf{T} & -\tilde{\mathbf{S}} \end{pmatrix}. \tag{5.10}$$

Denoting this Richardson iteration in compact form,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \tilde{\mathcal{P}}_L^{-1}(\mathbf{b} - \mathcal{A}\mathbf{x}_k),$$

one could also interpret it as *coupled* method since the outer method does not take the block structure of the saddle point system into account. In this sense, the lines between segregated and coupled methods become blurred. However, since the block triangular preconditioner, which is essential for the convergence of the outer iterative scheme, clearly segregates the **u**- and **p**-component, we still account the overall method decoupled.

5.1.1.2 Accelerating the Basic Iteration

The variants of the Uzawa method introduced above can all be regarded as (preconditioned) Richardson methods. Naturally, the next step is to accelerate these basic iterations in terms of Krylov space or multigrid methods (also compare Sections 2.6.3 and 4.2.3).

Acceleration by Krylov Space Methods

We always apply *right-preconditioned* Krylov methods to accelerate the inexact Uzawa method. In this case, one actually does not use the *lower*, but the **upper block-triangular preconditioner**

$$\tilde{\mathcal{P}}_U := \begin{pmatrix} \tilde{\mathbf{A}} & \mathbf{B} \\ \mathbf{0} & -\tilde{\mathbf{S}} \end{pmatrix}. \tag{5.11}$$

This reason for this is given in Section 5.1.2. We want to denote this preconditioner by **SCBTria**, where 'SC' indicates that the Schur complement matrix is involved. Its practical application is based on the factorisation

$$\begin{pmatrix} \tilde{\mathbf{A}} & \mathbf{B} \\ \mathbf{0} & -\tilde{\mathbf{S}} \end{pmatrix}^{-1} = \begin{pmatrix} \tilde{\mathbf{A}}^{-1} & \tilde{\mathbf{A}}^{-1}\mathbf{B}\tilde{\mathbf{S}}^{-1} \\ \mathbf{0} & -\tilde{\mathbf{S}}^{-1} \end{pmatrix} = \begin{pmatrix} \tilde{\mathbf{A}}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\tilde{\mathbf{S}}^{-1} \end{pmatrix}.$$

This means, to solve the system $\tilde{\mathcal{P}}_U \mathbf{c} = \mathbf{d}$ for some right hand side vector $\mathbf{d} = (\mathbf{d}^u, \mathbf{d}^p)^\mathsf{T}$, one successively computes the two components of the vector $\mathbf{c} = (\mathbf{c}^u, \mathbf{c}^p)^\mathsf{T}$ as

$$\begin{aligned} \mathbf{c}^p &= -\tilde{\mathbf{S}}^{-1}\mathbf{d}_k^p \\ \mathbf{c}^u &= \tilde{\mathbf{A}}^{-1}(\mathbf{d}_k^u - \mathbf{B}\mathbf{c}^p). \end{aligned}$$

Here, two clear differences between the Uzawa method and the PSC method (which are formally equivalent in the unaccelerated Richardson context) become evident:

- Since the Schur complement matrix $\mathbf{S} = \mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C}$ is symmetric positive definite (see, e. g., Verfürth [169]), the PSC iteration (5.6) for solving equation (5.4a) can be accelerated in terms of a CG method, denoted by **PSCCG**. This is also possible for the preconditioned version (5.8) when the preconditioner $\tilde{\mathbf{S}}$ is symmetric. For the (exact or inexact) Uzawa iteration, however, simple CG is not an option since the block triangular preconditioners (5.10) and (5.11) are not symmetric and the saddle point system (5.1) is indefinite. Hence, we have to use a Krylov method that is able to deal with unsymmetric, indefinite systems, e. g., BiCGstab or GMRes. Another possibility is the approach of Bank et al. [11] who symmetrise the block preconditioner which requires an additional solution of the $\tilde{\mathbf{A}}$ systems. This has to be seen as advantage of the PSC-Krylov type solvers over the Uzawa-Krylov type solvers.

  However, since we want to keep the flexibility to use unsymmetric smoothers/preconditioners for solving the inner $\mathbf{A}$-systems (cf. the discussion in Section 4.2.3), the CG method is not suited as accelerator anyway.

- The requirement of the PSC method to solve the $\mathbf{A}$-systems exactly could easily be relaxed leading to the inexact PSC method (see Listing 5.4). Unfortunately, this is not realisable so easily in terms of the PSCCG method. The reason is that additional matrix vector products with the Schur complement matrix $\mathbf{S}$ are necessary to determine the optimal step length for the updates of the residual and the solution vector; the simple trick of transforming iteration (5.5) into iteration (5.7) is not applicable. Hence, when applying an inexact PSCCG method, the error of the final solution depends on how accurately the inner $\mathbf{A}$-systems are solved. The inexact PSCCG method is analysed by Verfürth [169].

  When accelerating the inexact Uzawa algorithm by Krylov techniques, the error of the final solution does *not* depend on the inner solution of the $\mathbf{A}$-systems. The reason is that the approximations of the inverse of $\mathbf{A}$ are only needed in terms of preconditioning (see equation (5.10)), while the correctness of the solution is guaranteed by the outer BiCGstab or GMRes iteration. Hence, solving the $\mathbf{A}$-systems only approximately influences the efficiency of the preconditioner and, thus, the convergence speed, but the outer iterative process still converges to the correct solution. This has to be seen as clear advantage of the Uzawa-Krylov type solvers over the PSC-Krylov type solvers.

Exemplarily using BiCGstab as accelerator, we denote the accelerated inexact Uzawa and PSC method by **BiCGstab-SCBTria** and **PSCBiCGstab**, respectively. Correspondingly, we denote the unaccelerated versions by **Rich-SCBTria** and **PSC**, respectively.

We want to present two possibilities to circumvent the accuracy problems of the PSC-Krylov type solvers – relaxation strategies and using PSC-Krylov solvers as preconditioner:

Relaxation strategy: With help of a so called *relaxation strategy* [166], the correct solution can be obtained without computing the $\mathbf{A}$-systems exactly in each outer iteration step. The idea is to perform very accurate computations at the beginning of the outer iteration and then to successively relax the accuracy requirements. The resulting solver belongs to the class of *inexact Krylov space methods* [148], where the term 'inexact' reflects that

the matrix vector multiplication with the Schur complement matrix $\mathbf{S}$ is not performed exactly.

More precisely, the relaxation works as follows: Let us denote the Schur complement system (5.4a) to be solved by $\mathbf{Sp} = \mathbf{h}$, remembering that both the matrix and the right hand side contain the factor $\mathbf{A}^{-1}$. Furthermore, we assume that a relative stopping criterion ε_{rel} is used for the outer solver, i. e., the iteration is exited when $\|\mathbf{r}_k\| \leqslant \varepsilon_{\text{rel}}\|\mathbf{r}_0\|$, where $\mathbf{r}_k := \mathbf{h} - \mathbf{Sp}_k$ is the residual vector in the k-th iteration. Let η_k denote the accuracy of the matrix vector multiplication $\mathbf{Sp}_k$ in the k-th iteration. In our case, this means that the $\mathbf{A}$-systems are solved with an accuracy of η_k. Now, following van den Eshof et al. [166], this inner criterion is set to

$$\eta_k := \varepsilon_{\text{rel}} \frac{\|\mathbf{r}_0\|}{\|\mathbf{r}_{k-1}\|}, \tag{5.12}$$

i. e., it is based on the norm of the current residual of the outer iteration. Obviously, this results in very high accuracy requirements in the beginning ($\eta_1 = \varepsilon_{\text{rel}}$), which are relaxed as the residual norm decreases ($\eta_k \nearrow 1$ for $\|\mathbf{r}_{k-1}\| \searrow \varepsilon_{\text{rel}}\|\mathbf{r}_0\|$).

Simoncini and Szyld [148] analyse the technique and especially explain why the somehow counterintuitive strategy of relaxing the accuracy works: Simply spoken, the error between the computed and the actual residual is determined by a product of two factors, one of them being the error caused by the approximate matrix vector multiplication; since the other factor is guaranteed to decrease in the course of the iteration, it is feasible to relax the accuracy of the matrix vector multiplication. Simoncini and Szyld show that the convergence speed of the outer Krylov space method might suffer from the inexact matrix vector multiplication. Furthermore, they introduce an improved version of the relaxation strategy (5.12), which, however, requires additional knowledge of the system matrix and Krylov spaces which is not readily available. We confine ourselves to the simple strategy (5.12).

PSC-Krylov preconditioner: Instead of using the PSC-Krylov method as outer solver, one can apply it as preconditioner for another solution method. This outer method is applied to the whole saddle point system and calls the PSC-Krylov method as inner solver, e. g., performing a fixed number of iterations. Thus, the 'incorrectness' of the PSC-Krylov method merely affects its quality as preconditioner, while the outer method guarantees the correctness of the computed solution.

If the inner PSC preconditioner is not a Krylov method, but a simple Richardson iteration performing exactly one step, then it coincides with the inexact Uzawa algorithm in block triangular preconditioner formulation (5.10). Hence, the strategy of applying PSC methods as preconditioner can be interpreted as an extension of the inexact Uzawa algorithm: On the one hand, one can apply more than one preconditioning iteration, and on the other hand, one can replace the simple Richardson procedure by a Krylov scheme.

However, since the outer solution scheme is usually a Krylov method as well, it is not clear whether this additional (inner) Krylov method for preconditioning does really improve the overall efficiency of the method.

Exemplarily using BiCGstab as accelerator, we denote the relaxed PSC-Krylov methods by **PSCrelax-BiCGstab**, and the methods using PSC-Krylov preconditioners by **BiCGstab-PSC-BiCGstab**. Correspondingly, the unaccelerated version of the latter method is denoted by **Rich-PSC**.

Acceleration by Multigrid Methods

Multigrid accelerators are usually employed to achieve convergence rates that are independent of the grid size. However, since the methods described above already exhibit grid independent convergence when the preconditioner $\tilde{\mathbf{A}}$ is suitably chosen (see Section 5.1.2), it is questionable whether an outer multigrid acceleration is advantageous; we will examine this point in Section 5.2.

For the inexact Uzawa method the multigrid acceleration is straightforward: The basic iteration (5.9) is simply used as smoothing operation on each refinement level. However, as in the case of acceleration by Krylov solvers, we use the *upper* block triangular preconditioner (5.11) as smoother. We apply standard grid transfer operations to the three-component vectors, as usual performed in terms of three corresponding scalar operations. Also the coarse grid solver can simply be realised by a direct UMFPACK method applied to the whole 3×3 saddle point system. In Section 5.2 we will examine the efficiency of this multigrid solver.

The multigrid acceleration of PSC methods is more intricate. In principle, the procedure is the same as for the Uzawa methods: The preconditioned basic iteration (5.8) is used as smoothing operation on each refinement level and standard transfer operations are employed, which – in contrast to the multigrid-accelerated Uzawa methods – work on scalar components only. However, two difficulties occur: First, since the Schur complement matrix is often only implicitly available, UMFPACK can not be applied directly as coarse grid solver. Instead, one has to apply an iterative (one-level) PSC method (like PSCBiCGstab) to solve the coarse grid system. Second, the accuracy problem of the PSC approach is present again, i. e., the $\mathbf{A}$-systems have to be solved exactly to obtain the correct solution. We tried to apply the relaxation strategy (5.12), but were not successful in this regard. Hence, the method can only be efficient when it is used as preconditioner of an outer Krylov method, where the $\mathbf{A}$-systems can then be solved again with lower accuracy.

We denote the multigrid accelerated Uzawa and PSC methods by **MG-SCBTria** and **PSCMG**, respectively. When they are used as preconditioners, the resulting methods are denoted, for example, by **BiCGstab-MG-SCBTria** and **BiCGstab-PSCMG**.

In Section 5.2.1 we will compare the different variants of (accelerated) Uzawa and PSC methods in terms of convergence rates and total arithmetic efficiency.

5.1.1.3 Some Literature on Uzawa-type Methods

The classical Uzawa algorithm [3] is already more than fifty years old, but especially in the last two decades Uzawa-type methods attracted quite some attention. The reason is that they are relatively easy to implement, requiring only standard operations which are available in every solver toolkit. They have thus become a viable alternative for solving large-scale saddle point problems.

Two early articles providing analysis of inexact Uzawa methods are those by Queck [130] and Elman and Golub [62], where the latter treat stabilised saddle point systems. The results are improved by Bramble et al. [37] for the unstabilised case. Bank et al. [11] present and analyse a symmetrised version of the inexact Uzawa algorithm for unstabilised systems, and Zulehner [180] unifies and improves the results of Bramble et al. [37] and Bank et al. [11]. Cao [47] extends the results of Bramble et al. [37] to stabilised systems, while Cao et al. [48] apply the unified approach of Zulehner [180] to stabilised systems. Verfürth [169] analyses the inexact PSCCG method, and Cheng and Zou [50] present an extension of the method by Bank et al. [11] which exhibits slightly improved convergence properties but is not symmetric anymore. Elman [61] numerically compares the inexact Uzawa method with three other solvers for saddle point problems in terms of the Stokes equation.

Analyses of Krylov space methods using the block triangular preconditioner (5.10) can often be found in the literature (e. g., [63, 97, 106, 147]; also see the references in Benzi et al. [21]). Elman and Silvester [63] are concerned with the solution of the steady-state Navier-Stokes equation (where the matrix $\mathbf{A}$ is replaced by $\nu\mathbf{A}+\mathbf{N}$, ν being the fluid's viscosity and $\mathbf{N}$ representing the convection), considering stable finite elements (i. e., $\mathbf{C}=\mathbf{0}$). Klawonn [97] treats the mixed $\boldsymbol{u}/p$ formulation of linear elasticity with nearly incompressible material using stable elements (i. e., $\mathbf{C}^{\mathrm{c}}\neq\mathbf{0},\mathbf{C}^{\mathrm{s}}=\mathbf{0}$). Loghin and Wathen [106] and Simoncini [147] provide more general analyses for the cases $\mathbf{C}=\mathbf{0}$ and $\mathbf{C}\neq\mathbf{0}$, respectively. Simoncini [147] especially improves the results of Klawonn [97].

Multigrid PSC methods (PSCMG) are described and extensively tested by Turek [158] in the context of the (Navier-)Stokes equation. Multigrid-accelerated Uzawa-type methods, i. e., multigrid solvers applied directly to the saddle point system (5.1), have also been considered in the literature, e. g., the *SIMPLE* smoother [126, 127] or the *Braess-Sarazin* smoother [35]. They are related to the symmetrised preconditioner of Bank et al. [11]. In our tests, however, we only use the (unsymmetric) upper block triangular preconditioner (5.11) for smoothing (MG-SCBTria).

5.1.2 Preconditioners for Saddle Point Systems

The efficiency of the (accelerated) Uzawa-type methods decisively depends on the choice of the preconditioners $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{S}}$. These preconditioners have to approximate the original operators $\mathbf{A}$ and $\mathbf{S}$ in some sense and – remembering that they are applied in each outer iteration – they

must be inexpensive compared to the computation of the inverses $\mathbf{A}^{-1}$ and $\mathbf{S}^{-1}$. However, we briefly consider the special choices $\tilde{\mathbf{A}} = \mathbf{A}$ and $\tilde{\mathbf{S}} = \mathbf{S}$, i. e.,

$$\mathcal{P}_U := \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & -\mathbf{S} \end{pmatrix}. \tag{5.13}$$

Murphy et al. [115] analyse this *exact preconditioner* for the case $\mathbf{C} = \mathbf{0}$. The authors' basic observation also applies to the case $\mathbf{C} \neq \mathbf{0}$: For the right-preconditioned system matrix $\mathcal{N} := \mathcal{A}\mathcal{P}_U^{-1}$ we have

$$\begin{aligned}
\mathcal{N}^0 &= \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \\
\mathcal{N}^1 &= \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\mathsf{T} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{-1} \\ \mathbf{0} & -\mathbf{S}^{-1} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{B}^\mathsf{T}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix}, \\
\mathcal{N}^2 &= \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{B}^\mathsf{T}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix}^2 = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ 2\mathbf{B}^\mathsf{T}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix} = 2\mathcal{N}^1 - \mathcal{N}^0,
\end{aligned}$$

i. e., $\mathcal{N}^2$ is a linear combination of $\mathcal{N}^0$ and $\mathcal{N}^1$. Hence, the Krylov space

$$\operatorname{span}\{\mathcal{N}^0\mathbf{r}_0, \mathcal{N}^1\mathbf{r}_0, \mathcal{N}^2\mathbf{r}_0, \mathcal{N}^3\mathbf{r}_0, \ldots\}$$

actually is of dimension *two*, which means that a suitable Krylov method would only need *two* iterations to converge. This is, of course, only of theoretical relevance since applying the exact preconditioner $\mathcal{P}_U$ is much too expensive in large-scale computations or not possible at all. However, having a 'good' approximation $\tilde{\mathcal{P}}_U$ to the exact preconditioner $\mathcal{P}_U$, one can expect that the number of outer iterations will be small, too.

At first, we want to describe the preconditioner $\tilde{\mathbf{A}}$. A critical property of the outer saddle point solver is to exhibit a convergence behaviour that is independent of the problem size. One requirement to achieve this is that the preconditioner $\tilde{\mathbf{A}}$ is spectrally equivalent to the matrix $\mathbf{A}$. This means there have to be positive constants α_0, α_1 which are *independent of the refinement level* such that

$$\alpha_0 \mathbf{u}^\mathsf{T}\tilde{\mathbf{A}}\mathbf{u} \leqslant \mathbf{u}^\mathsf{T}\mathbf{A}\mathbf{u} \leqslant \alpha_1 \mathbf{u}^\mathsf{T}\tilde{\mathbf{A}}\mathbf{u}. \tag{5.14}$$

In this case the eigenvalues of the preconditioned matrix $\tilde{\mathbf{A}}^{-1}\mathbf{A}$ are contained in intervals that are independent of the grid size parameter h [147]. Hence, thus preconditioned solvers will exhibit iteration counts that are independent of the refinement level. An efficient preconditioning technique that fulfils condition (5.14) is using a multigrid solver, for which it is actually often sufficient to perform only one or few V-cycles [11, 37, 96, 97, 143, 147, 180].

5.1.2.1 The Schur Complement Preconditioner $\tilde{\mathbf{S}}$ in the Stationary Case

Similar to condition (5.14), we need to find a preconditioner $\tilde{\mathbf{S}}$ that is spectrally equivalent to the Schur complement matrix $\mathbf{S}$, i. e.,

$$\beta_0 \mathbf{p}^\mathsf{T}\tilde{\mathbf{S}}\mathbf{p} \leqslant \mathbf{p}^\mathsf{T}\mathbf{S}\mathbf{p} \leqslant \beta_1 \mathbf{p}^\mathsf{T}\tilde{\mathbf{S}}\mathbf{p} \tag{5.15}$$

with positive constants β_0, β_1 that are independent of the problem size. For the stationary case, where $\mathbf{S} = \mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C}$, the pressure mass matrix $\frac{1}{2\mu}\mathbf{M}_p$ fulfils condition (5.15) (see, e. g., Verfürth [169], Elman et al. [65] and Klawonn [97]) and is thus a suitable preconditioner. The scaling factor results from the factor 2μ contained in the matrix $\mathbf{A}$ (see equation (3.57) on page 134). To better account for the contribution of the matrix $\mathbf{C}^\mathrm{c} = -\frac{1}{\lambda}\mathbf{M}_p$ (see equation (3.58) on page 134), which increases with the material becoming more compressible, we subtract it from the preconditioner, $\frac{1}{2\mu}\mathbf{M}_p - \mathbf{C}^\mathrm{c}$. This simply leads to a corresponding scaling in the case of compressible material, such that the preconditioner is finally given by

$$\tilde{\mathbf{S}} = \begin{cases} (\frac{1}{2\mu} + \frac{1}{\lambda})\mathbf{M}_p & \text{if } \nu < 0.5 \\ \frac{1}{2\mu}\mathbf{M}_p & \text{if } \nu = 0.5. \end{cases} \tag{5.16}$$

The application of the preconditioner means to approximately solve the corresponding system with the pressure mass matrix.

Although the (approximate) inversion of the pressure mass matrix in each iteration is much cheaper than the (approximate) inversion of the Schur complement matrix $\mathbf{S}$, it is still relatively expensive since it requires the application of an inner iterative solution method. To decrease the costs, one can exploit the fact that $\mathbf{M}_p$ is spectrally equivalent to the lumped mass matrix $\mathbf{M}_p^l$ [65]. Using this diagonal matrix leads to the Schur complement preconditioner

$$\tilde{\mathbf{S}}^l = \begin{cases} (\frac{1}{2\mu} + \frac{1}{\lambda})\mathbf{M}_p^l & \text{if } \nu < 0.5 \\ \frac{1}{2\mu}\mathbf{M}_p^l & \text{if } \nu = 0.5. \end{cases} \tag{5.17}$$

Its application only requires one vector-vector operation, i. e., the scaling of the current right hand side vector with the inverse of the diagonal. On the other hand, one can expect an influence on the outer convergence. In Section 5.2.1.8 we compare the costs of the two preconditioners $\tilde{\mathbf{S}}$ and $\tilde{\mathbf{S}}^l$ with the help of some numerical examples.

One could also add the complete matrix $\mathbf{C}$ to the preconditioner, $\tilde{\mathbf{S}} = \frac{1}{2\mu}\mathbf{M}_p + \mathbf{C}$, including the stabilisation matrix $\mathbf{C}^\mathrm{s}$. However, since the entries of this matrix are of order $O(h^2)$, they can be neglected within the preconditioner. Furthermore, adding $\mathbf{C}^\mathrm{s}$ would prevent the lumping technique.

The efficiency of the described preconditioners $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{S}}$ fulfilling conditions (5.14) and (5.15), respectively, has been proven and numerically demonstrated by many authors in different contexts. For the type of saddle point system we are concerned with, i. e., having a nonzero negative semidefinite block $\mathbf{C}$, see, e. g., the analyses of Cao [47], Cao et al. [48], Elman [61], Klawonn [97], Silvester and Wathen [143], and Simoncini [147].

5.1.2.2 The Schur Complement Preconditioner $\tilde{\mathbf{S}}$ in the Transient Case

In the case of time-dependent computations, Schur complement preconditioning is more involved. We briefly recall the system matrix that has to be inverted (approximately) in each iteration of the time stepping scheme,

$$\begin{pmatrix} \frac{\rho}{\tau^2\beta}\mathbf{M}+\mathbf{A} & \mathbf{B} \\ \mathbf{B}^\mathsf{T} & \mathbf{C} \end{pmatrix},$$

with $\mathbf{M}$ being the displacement mass matrix, ρ the material's density, β an algorithmic parameter of the Newmark scheme (usually, $\beta = 0.25$), and τ the time step size (see the derivations in Sections 3.1.4 and 3.4.1). We use the short notation $\delta := \frac{\rho}{\tau^2\beta}$, where

$$\delta \to \begin{cases} 0 & \text{if } \tau \to \infty \\ \infty & \text{if } \tau \to 0 \end{cases}$$

and the limit $\delta = 0$ coincides with the stationary case.

Before discussing the Schur complement preconditioner $\tilde{\mathbf{S}}$, we briefly mention that the preconditioner $\tilde{\mathbf{A}}$ for the upper left block of the saddle point system does not have to be changed compared to the stationary case; the addition of the displacement mass matrix improves the spectral properties of the system.

The time step size τ is an algorithmic parameter that is either fixed by the user or that is varying in the course of the simulation due to an adaptive time stepping scheme [164, 165]. Hence, a preconditioner for the Schur complement has to be robust with respect to the size of the time step, i. e., it has to exhibit convergence rates that are independent of the size of δ. For large time step sizes the system tends to that of the stationary case, being spectrally equivalent to the (pressure) mass matrix, while for small step sizes it tends to a mixed formulation of the discrete (pressure) Laplace operator. It is exactly this fact which aggravates preconditioning of the Schur complement operator

$$\mathbf{S}(\delta) := \mathbf{B}^\mathsf{T}(\delta\mathbf{M}+\mathbf{A})^{-1}\mathbf{B} - \mathbf{C}; \tag{5.18}$$

the spectral properties of the matrix $\mathbf{S}(\delta)$ differ significantly for the two extreme cases $\delta \ll 1$ and $\delta \gg 1$. Cahouet and Chabard [45] take into account these two extreme cases to construct a suitable preconditioner. For its description we neglect the matrix $\mathbf{C}$ for the time being and will later comment on modifications that incorporate this matrix. For $\delta \ll 1$, i. e.,

$$\mathbf{S}(\delta) \approx \mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{B},$$

we already know that the scaled pressure mass matrix $\frac{1}{2\mu}\mathbf{M}_p$ (or, its lumped version) is a suitable preconditioner (see previous section). It is referred to as **diffusive preconditioner** [158]. For $\delta \gg 1$ we have

$$\mathbf{S}(\delta) \approx \mathbf{B}^\mathsf{T}(\delta\mathbf{M})^{-1}\mathbf{B},$$

i. e., the Schur complement matrix corresponds to a discrete mixed formulation of the Laplacian. Hence, the scaled pressure Laplace operator $\frac{1}{\delta}\mathbf{L}_p$ with natural (i. e., zero Neumann) boundary conditions is a suitable preconditioner, called the **reactive preconditioner** [158].[4] The desired preconditioner for the complete Schur complement operator (5.18) has to be efficient for the whole range of time step sizes. To achieve this it is constructed by linearly combining the (properly scaled) preconditioners for the two extreme cases described above. We define the resulting transient preconditioner $\tilde{\mathbf{T}}$ in terms of its action on a given (defect) vector $\mathbf{d}$:

$$\tilde{\mathbf{T}}^{-1}\mathbf{d} = 2\mu\mathbf{M}_p^{-1}\mathbf{d} + \delta\mathbf{L}_p^{-1}\mathbf{d}. \tag{5.19}$$

This preconditioning technique has been first proposed by Cahouet and Chabard [45], who demonstrate its efficiency by means of some numerical examples with the generalised Stokes problem. Bramble and Pasciak [36], Kobelkov and Olshanskii [98] and Mardal and Winther [110] analyse the preconditioner for the generalised Stokes equation. They show that the convergence rates of correspondingly preconditioned iterative schemes are independent of the grid size parameter h and of the time step size τ. Turek [158] examines the preconditioner in terms of the nonstationary Navier-Stokes equation.

Now, we describe how the matrix $\mathbf{C}$ can be taken into account. *To our knowledge, this setting has not been analysed in the literature yet, neither for stabilised systems, nor for elasticity systems with nearly incompressible material.* The matrix stemming from the compressibility terms, $\mathbf{C}^{\mathrm{c}}$, is included in the diffusive preconditioner by properly scaling the pressure mass matrix as it is done in the stationary case (see equation (5.16)). Precisely, the factor 2μ in equation (5.19) is replaced by the factor $\left(\frac{1}{2\mu}+\frac{1}{\lambda}\right)^{-1}$, which results in

$$\tilde{\mathbf{T}}^{-1}\mathbf{d} = \begin{cases} \frac{2\mu\lambda}{\lambda+2\mu}\mathbf{M}_p^{-1}\mathbf{d} + \delta\mathbf{L}_p^{-1}\mathbf{d} & \text{if } \nu < 0.5 \\ 2\mu\mathbf{M}_p^{-1}\mathbf{d} + \delta\mathbf{L}_p^{-1}\mathbf{d} & \text{if } \nu = 0.5. \end{cases} \tag{5.20}$$

The stabilisation matrix $\mathbf{C}^{\mathrm{s}}$ is simply neglected in the stationary case since its entries are of order $O(h^2)$. The question is if this is feasible for the transient case, as well. It certainly *is* feasible if the time step is large, i. e., when we are 'closer' to the stationary case. Hence, for the following considerations we assume the time step to be small and neglect the diffusive part of the preconditioner. Adding the stabilisation matrix to the reactive preconditioner then results in

$$\tilde{\mathbf{T}} = \tfrac{1}{\delta}\mathbf{L}_p - \mathbf{C}^{\mathrm{s}}.$$

Recalling that $\frac{1}{\delta}$ is of order $O(\tau^2)$ and the entries of $\mathbf{C}^{\mathrm{s}}$ of order $O(h^2)$, we can make the following observation: When $\tau \gg h$, then the stabilisation matrix can still be neglected without crucially degrading the convergence rate of the solver. When $\tau \ll h$, then the stabilisation part actually represents the main contribution of the complete Schur complement operator (assuming the material is nearly incompressible). The fact that these auxiliary, purely algorithmic and

[4] It is also possible to approximate the action of the operator $\mathbf{B}^{\mathsf{T}}(\delta\mathbf{M})^{-1}\mathbf{B}$ directly in terms of an iterative solution method. For certain finite elements the operator can also be constructed exactly. See Turek [158] for details in this regard.

'non-physical' stabilisation terms outweigh the actual 'physical' contributions, indicates that the problem can not be well-posed in this case. This observation is in line with the theoretical findings of Bochev et al. [28] about stabilisation in the small time step limit (see Section 3.4.2). The main result of this study is that the time step τ must be at least of size $O(h)$ in order to guarantee a stable formulation and reliable discrete solutions. Our observation about the usage of the stabilisation matrix in transient preconditioning underlines this result from a different point of view.

As a consequence, we suppose that the only range where the addition of the stabilisation matrix *would* make sense, is the range where the stabilised formulation is not reliable. Hence, when avoiding this range right from the beginning one can neglect the stabilisation terms in the transient preconditioner, as well. So, our final preconditioner is that defined in equation (5.20). Again, we can replace the pressure mass matrix by its lumped version $\mathbf{M}_p^l$, the resulting preconditioner is then denoted by $\tilde{\mathbf{T}}^l$.

Applying this preconditioner means to successively solve two scalar systems. For diffusive preconditioning, the (eventually lumped) pressure mass matrix has to be approximately inverted, which works exactly as in the stationary case. For reactive preconditioning, we have to solve a pressure Poisson problem which can be efficiently done with the help of our scalar ScaRC solvers (see Chapter 2). The difficulty here is the fact that a pure Neumann problem with a singular system matrix has to be solved. To obtain a unique solution we force the mean integral of the iteration vector to be zero by shifting its entries correspondingly. A clear disadvantage is that this is only possible in iterative solution methods, hence, we can not apply a direct solver for the coarse grid problem within the ScaRC multigrid scheme. Instead, we have to apply a Krylov method which is possibly sensitive to mesh anisotropies. In the case of complicated geometries that already require a large number of coarse grid elements, this might lead to a performance bottleneck. Furthermore, it is unclear how efficient the simple 'shifting'-technique is on irregular meshes. For a more sophisticated method to solve pure Neumann problems, see Bochev and Lehoucq [26].

The described Schur complement preconditioner (5.20) for transient saddle point problems works indeed well when the time step is large enough; for an example, see Section 5.3. However, we encounter severe convergence difficulties when the time step becomes too small. We suppose that this is due to the fundamental 'small time step problem' for stabilised formulations described in Section 3.4.2. Unfortunately, it is impossible for a given (realistic) configuration to determine this critical time step limit a priori. It depends on several factors like the geometry, boundary conditions or material parameters. We performed several tests and found that in many cases only slight changes of the configuration, of parts of the solver or of the time step size could lead to strongly varying convergence behaviour. Currently, we do not fully understand all these interdependencies. Hence, we refrain from presenting detailed numerical convergence studies in this thesis. Instead, we only present in Section 5.3 one transient simulation with an oscillating beam using finite deformation elasticity to basically validate that our approach can be successful. In this simulation, for example, the time steps $\tau = 0.02$ and $\tau = 0.01$ s were unproblematic, but for $\tau = 0.005$ s our solvers already showed very irregular

convergence behaviour or even diverged. Taking additionally into account, that the oscillation frequency of the beam is roughly 1 s, time steps of $\tau > 0.5$ s do not make much sense. Hence, the interval of applicable time steps is quite limited, on the one hand by the 'physics' of the given configuration, on the other hand by numerical instabilities. In future work, the described problem has to be examined and understood in more detail.

5.1.3 Coupled Multigrid with Vanka-type Smoothing

In this section, we want to present a class of multigrid smoothers which has been originally introduced by Vanka [168] for solving the Navier-Stokes equations discretised by Finite Differences. Basically, the technique belongs to the category of multiplicative domain decomposition (block Gauß-Seidel) methods, locally coupling all field variables occurring in the formulation. The smoother is sometimes denoted as *symmetrically coupled Gauß-Seidel* (SCGS) [168] or *box iteration/relaxation* [172]. The Vanka technique has especially been developed to deal with saddle point systems exhibiting a zero block appearing on the diagonal of the system matrix, where standard (point-wise) Jacobi or Gauß-Seidel smoothers fail. Stokes and linearised Navier-Stokes systems belong to this category which is the main reason for the strong influence the method had (and still has) in the field of computational fluid dynamics (CFD). Other reasons are that it can be implemented with the help of elementary techniques available in all finite element packages and that it is efficient and robust for a wide class of problem configurations.

Multigrid-Vanka solvers have to be seen in contrast to the class of segregated solvers that are described in Section 5.1.1. The main idea of the Vanka approach is to directly couple all field variables, i. e., the displacements and pressure in our case, on a *local* level, resulting in small coupled systems that have to be solved successively (see below). Segregated methods treat the system in a *global and decoupled* manner, leading to reduced systems for displacements on the one hand and for the pressure on the other hand. In Section 5.1.1 we mentioned solvers that also apply multigrid to the whole saddle point system [35, 126, 127]. They, however, perform a segregation of variables within the smoothing process, so they belong to the class of decoupled saddle point solvers.

5.1.3.1 Some Literature on Vanka-type Smoothers

While there seem to be only few papers dealing with theoretical aspects of Vanka-type smoothing [109, 114, 141], much literature can be found presenting numerical studies in the context of the discretised Navier-Stokes equations in CFD. John [86] and John and Tobiska [89] apply it to the nonconforming Crouzeix/Raviart element P_1/P_0, Turek [158] to the corresponding nonconforming rotated bilinear Rannacher/Turek element $\tilde{Q}_1/P_0$ and Becker [17] to the stabilised Q_1/Q_1 element. In all cases, the smoother is extensively tested on the benchmark configuration 'flow around a cylinder' [160] for the steady and unsteady state. Ouazzi and Turek [120] transfer the Vanka idea to edge-oriented storage- and stabilisation techniques for the Navier-Stokes equations. Zeng and Wesseling [177] compare Vanka-type smoothers to ILU methods

for the case of Navier-Stokes in general coordinates. To treat anisotropic grids more robustly, several extensions have been introduced. Thompson and Ferziger [155] define *symmetrically coupled alternating line (SCAL)* versions for Finite Difference discretisations, Becker [17] uses a *string-wise* version for the stabilised Q_1/Q_1 discretisation, and Schmachtel [139] develops an adaptive blocking strategy. John and Matthies [88] and John [87] successfully apply Vanka smoothers to higher order finite element methods. Comparative solver studies including Vanka smoothers can be found in the articles of John [86], John and Tobiska [89], Turek [158], Benzi and Olshanskii [20] and Larin and Reusken [103]. For further references, see the overview paper of Wesseling and Oosterlee [172].

There are only few papers describing the use of Vanka-type smoothers in the context of CSM. For many kinds of solid mechanical problems there is obviously no need to refrain from standard (point-wise) multigrid smoothers. But for the mixed $\boldsymbol{u}/p$ formulation, which is used to treat (nearly) incompressible material effects, similar equation systems as for Navier-Stokes arise (see Section 3.2.3) such that in this case the use of Vanka-type smoothers is reasonable. Suttmeier [153] applies the variant developed by Becker [17] to elastoplastic materials and compares it to standard Gauß-Seidel smoothing. Gaspar et al. [69] compare Vanka-type smoothers to decoupled smoothers in the context of incompressible poroelasticity equations. Hron and Turek [80] employ Vanka smoothers to solve coupled systems arising from the Q_2/P_1 discretisation of fluid structure interaction problems.

To further close this gap, we performed extensive numerical studies of Vanka-type smoothers applied to typical CSM problems (see Wobker and Turek [175]). Some of the results are presented in this thesis, for further details we refer to our paper.

5.1.3.2 Basic Strategy of Vanka-type Smoothers

The basic common idea of Vanka-type smoothers is to decompose the mesh into small subregions (usually, consisting of one or few elements/vertices) and treat these subregions separately. In more detail, one smoothing step consists of a loop over all these subregions where in each iteration the following steps are performed:

1. *Extract the entries of the global matrix corresponding to the degrees of freedom (DOF) of the current subregions and assemble them into a small local matrix.*

2. *Build the corresponding local residual.*
   This is done in a Gauß-Seidel manner, i. e., information, which has been updated in previously treated subregions, is immediately incorporated into the assembly process of the current local residual.

3. *Solve the resulting system with the local residual as right hand side.*
   Note that the resulting local matrices are always invertible: When the subregion lies in the interior of the mesh the local matrix contains the 'full' entries of the global matrix and can therefore be interpreted as arising from a mesh consisting of the subregion itself

enclosed by a further element layer with zero Dirichlet boundary conditions. When the subregion lies at the boundary, the local system 'inherits' the boundary information of the global matrix, i. e., unit rows/columns in case of Dirichlet boundary and 'half' entries in case of Neumann boundary, respectively. We employ a direct solver to invert the local systems.

4. *Update the corresponding parts of the global solution with this local correction.*

This is a general description of the Vanka process. In the case of the FEAST environment, however, it has to be interpreted in a special way. First, in order to avoid confusion, we emphasise that these 'subregions' do not coincide with the M tensor product subdomains $\Omega_i, i = 1, \ldots, M$, that result from FEAST's initial macro decomposition of the domain $\bar{\Omega}$ (see Section 2.3). Instead, the 4-step process described above is *independently applied to each subdomain* Ω_i, where the small subregions within a subdomain are traversed corresponding to the rowwise numbering from the 'lower left' to the 'upper right' corner of each subdomain. This means, what is called *global matrix* in the 4-step Vanka process, actually refers to each of FEAST's *local matrices*. Hence, in FEAST, Vanka smoothing is never applied to the (only virtually existing) global matrix corresponding to the whole domain. Instead, we are using a *block-smoothed* multigrid scheme (see Section 2.1.2), i. e., the contributions of the M local corrections are assembled after each smoothing step in an additive (block Jacobi) way (exactly, as it is done in the case of the scalar FEAST smoothers like ADITriGS). This means, data exchange across subdomain boundaries is performed only *after* all subdomains have been treated. This strategy of applying Vanka smoothers per subdomain is mandatory for enabling an efficient *parallel* execution. On the other hand, the smoothers suffer from the usual disadvantages that are inherent to the block Jacobi approach (see Chapter 2). The numerical results in Section 5.2.2 illustrate this.

We now present four Vanka variants that differ in the choice of the subregions and how the local systems are built.

1. The Cell-based Vanka Smoother

The first variant we describe is the *cell-based* Vanka smoother. Here, each subregion consists of exactly one element and the local system matrix contains the DOF of the four element nodes (see Figure 5.1a). The smoother can be seen as a multiplicative domain decomposition method with *minimal overlap*, i. e., the subregions (=elements) only intersect at their (element) boundaries which minimises computational overhead.

Let e be the current element in the smoothing procedure, let the restriction of a vector or a matrix to this element be denoted with the index e, and let ω be a relaxation parameter. Then the necessary calculations can be formulated as follows:

$$\begin{aligned} \mathbf{r}_{\text{loc}} &= \mathbf{f}_e - (\mathcal{A}\mathbf{x})_e \quad \text{(local residual)}, \\ \mathbf{x}_{\text{loc}} &= \mathcal{A}_e^{-1}\mathbf{r}_{\text{loc}} \quad \text{(local correction)}, \\ \mathbf{x}_e &= \mathbf{x}_e + \omega\mathbf{x}_e \quad \text{(update of global solution)}. \end{aligned} \tag{5.21}$$

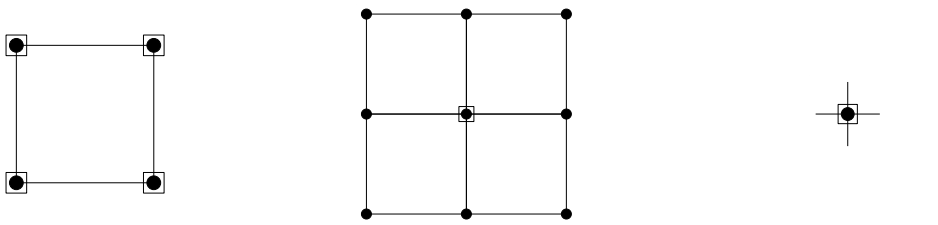
The advantage of this smoother variant is that it is not restricted to 3×3 saddle point systems arising from the mixed formulation, but it can also be applied to 2×2-systems stemming from the pure displacement formulation. In the case of the mixed formulation, displacements and pressure are treated equally (see Figure 5.1a). This makes the smoother very attractive from an implementational point of view as it can be applied to arbitrarily coupled equation systems without deeper knowledge about the underlying problem. It clearly distinguishes this Q_1/Q_1 variant of the Vanka smoother from the original version [168] where the pressure is represented in the local system with only one DOF.

The (local) relaxation with the parameter ω is different from (global) damping in the multigrid method: The local residuals corresponding to the subsequently treated elements are immediately affected by the relaxation, whereas the damping in the multigrid method scales the global correction vector *after* completion of the smoothing procedure. We emphasise this in view of the strategy to improve multigrid by enclosing the smoother by some Krylov space method (see Sections 2.6.3 and 4.2.3). The benefit of doing so is the 'automatic choice' of the correct damping parameter which can considerably increase the robustness of multigrid. The (locally acting) relaxation parameter, however, can *not* be 'adjusted' this way, it has to be set manually by the user.

The vector $(\mathcal{A}\mathbf{x})_e$ can be constructed in $O(1)$ time due to the sparse structure of $\mathcal{A}$. In the case of a 3×3 saddle point system for the mixed $\boldsymbol{u}/p$ formulation with Q_1/Q_1 in 2D on a tensor-product mesh, there are at most 27 non-zero elements per matrix row.

2. The Patch-based Vanka Smoother

Becker [17] adapts the standard Vanka smoother for the stabilised Q_1/Q_1-discretisation and applies it to solve the incompressible Navier-Stokes equations. Due to the similar structure of the linear equation systems arising in this context, the adapted smoother can be applied to the mixed formulation in CSM, as well. This is done, for instance, by Suttmeier [153] for the case of elastoplastic material.



(a) Cell-based. **(b)** Patch-based. **(c)** Vertex-based.

Figure 5.1: Subregions for the cell-, patch- and vertex-based Vanka smoother and the corresponding DOF (• displacement DOF, □ pressure DOF).

Instead of looping over all elements in the mesh we iterate over all pressure DOF, i. e., over all nodes of the mesh. For each of them the displacements coupling with the node are taken

into account such that in case of a tensor product mesh a patch consisting of the four adjacent elements has to be considered (see Figure 5.1b). The corresponding local systems to be solved have the following form:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & b_n \\ d_1 & \dots & d_n & c \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \\ f \end{pmatrix}.$$

In case of Q_1/Q_1 in 2D and the patch centre being an inner mesh node we have $n = 18$. For linearised elasticity, the matrix is symmetric, i. e., $d_i = b_i, i = 1, \dots, n$.

The motivation behind this approach is that in the case of the Q_1-discretisation the incompressibility constraint $\operatorname{div} \boldsymbol{u} = 0$ can, in general, not be met locally in *one* element. It *is* possible, however, to fulfil this condition locally by increasing the number of involved displacement DOF, e. g., by considering the whole patch around one node. Moreover, the ratio between number of displacement and pressure DOF better reflects the fact that in the underlying equation (3.56) on page 134, the order of displacement derivatives is higher than the order of pressure derivatives, i. e., displacements should be approximated 'more accurately'. So, the patch-based Vanka smoother is better suited for (nearly) incompressible material behaviour.

Compared to the cell-based Vanka smoother described in the previous section, the patch-based variant has some (minor) disadvantages:

- The resulting local systems are larger such that more time for the LU decomposition is needed.
- The patches overlap each other by one element layer which results in some computational overhead.
- With its special form (taking the third DOF only in the patch centre) the patch-based variant is only applicable to the mixed $\boldsymbol{u}/p$ formulation, but not to the pure displacement case. (The idea of overlapping patches, however, *could* be transferred to the pure displacement case, of course. But this is not done here.)
- Assembling the local systems is slightly more involved since a larger neighbourhood has to be considered and one has to distinguish between the cases whether the pressure node lies in the interior of a tensor product subdomain Ω_i, on its boundary, or in a corner. In the latter two cases, we only have twelve or eight displacement DOF, respectively, instead of eighteen.

3. The Patch-based Diagonal Vanka Smoother

Becker [17] and Suttmeier [153] apply a simplified variant of the patch-based Vanka smoother. The idea is to couple each displacement DOF only with itself such that the upper left $n \times n$-part of the local matrix is a diagonal matrix:

$$\begin{pmatrix} a_{11} & 0 & \dots & 0 & b_1 \\ 0 & a_{22} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \dots & 0 & a_{nn} & b_n \\ d_1 & \dots & \dots & d_n & c \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ \vdots \\ f_n \\ f \end{pmatrix}.$$

Denoting this system with

$$\begin{pmatrix} \mathbf{D} & \mathbf{b} \\ \mathbf{d}^\mathsf{T} & c \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ y \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ f \end{pmatrix},$$

it can be solved very easily by eliminating $\mathbf{x}$:

$$y = \frac{\mathbf{d}^\mathsf{T}\mathbf{D}^{-1}\mathbf{f} - f}{\mathbf{d}^\mathsf{T}\mathbf{D}^{-1}\mathbf{b} - c}, \qquad \mathbf{x} = \mathbf{D}^{-1}(\mathbf{f} - y\mathbf{b}).$$

Basically, this can be performed with two scalar products and two vector-vector multiplications/additions which is much cheaper than applying an LU decomposition to the whole $(n+1) \times (n+1)$-system as in the full patch-based approach. It has been observed that diagonal variants of the Vanka smoother are less robust with respect to mesh anisotropies in the context of incompressible Navier-Stokes equations (see, e. g., Schieweck [138]). In a previous publication [175], we confirm this for the elasticity case.

4. The Vertex-based Diagonal 'Vanka Smoother'

For sake of completeness we present an even simpler smoother variant where the 'subregion' consists of only one vertex (see Figure 5.1c). Now we let each DOF only couple with itself, such that the system to be 'solved' looks, in case of the 2D mixed formulation, as following:

$$\begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f \end{pmatrix}.$$

Actually, this describes the standard Gauß-Seidel smoother. But, according to the above constructions it could also be interpreted as *vertex-based diagonal Vanka smoother*.

Evaluation of the Four Vanka-type Smoothers

The presented Vanka smoothers build a hierarchy that is determined by the size of the subregions:

$$\text{vertex} \quad \rightarrow \quad \text{cell} \quad \rightarrow \quad \text{patch}.$$

They are numerically studied in detail in a previous publication [175], covering the pure displacement and the mixed formulation, compressible and (nearly) incompressible material. We only want to summarise the main conclusions here:

- All smoothers show a certain degree of sensitivity with respect to the relaxation parameter ω. A value that is optimal for one configuration may lead to divergence in a different configuration. Consequently, a proper choice of this parameter cannot be done automatically and requires some experience by the user. This has to be seen as general drawback of *all* considered smoothers.
- For compressible material and fairly isotropic meshes the standard Gauß-Seidel smoother provides good results and is often the most efficient one. But as soon as mesh anisotropies and/or incompressibility are involved it is advantageous or even necessary to switch to one of the more sophisticated Vanka smoothers.
- The observation already made by other authors that the diagonal patch-based Vanka smoother shows deficiencies on anisotropic meshes is clearly confirmed. *But:* When the multigrid scheme is enclosed by an outer Krylov method, this effect is significantly weakened. However, none of the presented smoothers is robust with respect to higher aspect ratios.
- For incompressible material it is mandatory to use one of the patch-based Vanka smoothers. The standard Gauß-Seidel and the cell-based Vanka smoother have to be strongly underrelaxed and solve only very simple, isotropic configurations in reasonable time.

In this thesis we are especially interested in the performance of Vanka smoothers applied to saddle point systems stemming from the mixed formulation in the case of nearly incompressible material. As demonstrated in a previous publication [175], only the two patch-based variants are efficient in this case, so we will confine our numerical tests in Section 5.2.2 to them.

5.1.4 Segregated vs. Coupled Saddle Point Solvers

We want to emphasise three major differences between segregated and coupled methods that are important in the context of this thesis.

The first difference refers to our basic strategy of reducing the solution of multivariate systems to the solution of a series of scalar systems (see Section 4.1). This strategy is perfectly pursued within segregated solvers. The $\mathbf{A}$ block of the saddle point system is treated similarly to the pure displacement case (see Section 4.2). For Schur complement preconditioning, additionally the pressure mass matrix and/or the pressure Laplace matrix have to be inverted in each outer iteration. Especially, the additional Laplace systems in case of transient computations require a significant part of the overall work. However, this is the key capability of FEAST's ScaRC solvers which can tap their full potential there. Hence, *the segregated saddle point solvers benefit greatly from the strategy* described in Section 4.1. On the other hand, *the strategy cannot be employed at all within coupled multigrid Vanka solvers* which reduce the solution process

to the successive treatment of many tiny linear systems. All the efficient solver structures of FEAST can not be exploited at all. The tensor product property helps to efficiently build the small systems, though, but line-wise processing of (global) matrix bands – which is decisive for FEAST's processor efficiency – is not possible. From this point of view, we regard multigrid Vanka solvers as *the* essential (and popular) alternative to our basic strategy. Hence, a comparison between the two approaches is particularly well suited to demonstrate the superiority of our strategy.

The second difference between segregated and coupled methods is the following: The efficiency of the segregated methods described in Section 5.1.1 decisively depends on the existence of suitable Schur complement preconditioners. To construct such preconditioners, it is mandatory to exploit special properties of the underlying equation like the spectral properties of the Schur complement operator (see Section 5.1.2). Coupled multigrid Vanka solvers, on the other hand, do *not* require such deeper knowledge. Simply spoken, they 'blindly' solve the local representations of the given global problem, without taking heed of the physical or numerical properties of the equation. In the case of the Stokes-like saddle point system we are concerned with, this is not necessarily an advantage since we actually *have* efficient Schur complement preconditioners. In more complicated situations, however, like the nonsymmetric Oseen equation (stemming from the linearisation of the Navier-Stokes equation) or finite deformation elasticity, efficient Schur complement preconditioners may *not* always be available. Then, the coupled multigrid Vanka smoothers can indeed be superior. For a detailed discussion of this topic in the context of the Navier-Stokes equation, see Turek [158].

The third difference between segregated and coupled methods has to be seen in the treatment of mesh anisotropies. In the case of segregated methods, these anisotropies are exclusively handled within the scalar ScaRC solvers that are used for preconditioning, usually by choosing ADITriGS as scalar multigrid smoother (see Section 2.5.3.1). Hence, the outer saddle point solver does not have to be modified. Coupled multigrid Vanka solvers, on the other hand, are known to be inefficient for anisotropic meshes. Here, special modifications and nontrivial enhancements are necessary to render the method more robust [17, 139, 155]. These extensions, however, are out of the scope of this thesis.

In Section 5.2.2 we compare the two solver approaches in terms of their numerical and parallel efficiency.

5.2 Numerical Tests

5.2.1 Comparison of Segregated Saddle Point Solvers

In this section we compare the various segregated saddle point solvers introduced in Sections 5.1.1 and 5.1.2. We are especially interested in the question whether one solution method is clearly superior to the others. First, we specify the solvers in more detail (Section 5.2.1.1). Then, we perform numerical tests with known analytical solutions on the simple unitsquare

(Section 5.2.1.2). The tests are then extended to more complex real configurations (Sections 5.2.1.3, 5.2.1.5 and 5.2.1.6). For all tests in this section we use PPPLC stabilisation (see Section 3.3.2). Furthermore, we employ the non-lumped mass matrix for Schur complement preconditioning in all tests; a comparison with the lumped version is performed in Section 5.2.1.8.

5.2.1.1 Solver Variants

To specify saddle point solvers, one has to account for different stages within the nested algorithm. We distinguish between the following building blocks:

1. the basic saddle point solver for the whole 3×3 block matrix $\mathcal{A}$,
2. the solver for the 2×2 block matrix $\mathbf{A}$,
3. the scalar ScaRC solver for the blocks $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$,
4. the scalar ScaRC solver for diffusive and reactive preconditioning,
5. the coarse grid solvers within multigrid schemes.

It depends on the specific saddle point solver, which of these building blocks are actually needed. For each of the five building blocks, we now list the individual solver variants that we use for our numerical tests. Some important solver parameters are appended in parentheses (cf. Sections 2.5.5.2 and 4.2.6):

1. We compare the following saddle point solvers:

   a) `PSCBiCG`:
   An exact pressure Schur complement solver (see equation (5.6) on page 253 and Listing 5.2 on page 254), accelerated by replacing the basic iteration by a BiCGstab method. When performing matrix vector multiplications with the Schur complement matrix $\mathbf{S}$, the $\mathbf{A}$-systems have to be solved exactly (i. e., with the same accuracy as the whole system) in order to obtain an accurate solution.

   b) `PSCRelaxBiCG`:
   The same solver as the previous one, using the relaxation technique (5.12) described in Section 5.1.1.2.

   c) `BiCG-PSC(1)`:
   The basic pressure Schur complement solver used as preconditioner for an outer BiCGstab solver which is applied to the whole saddle point system. The PSC preconditioner performs exactly one iteration, where it suffices to solve the $\mathbf{A}$-systems in matrix vector multiplications with the Schur complement matrix $\mathbf{S}$ only approximately.

d) `BiCG-MG(1,V11)-PSC(1)`:
An extension of the previous solver, where the outer BiCGstab solver uses one V-cycle multigrid iteration (applied to the whole saddle point system) as preconditioner which is then smoothed by one step of the basic pressure Schur complement solver.

e) `BiCG-PSCMG(1,V11)`:
Another extension of the solver under point 1c, where the basic pressure Schur complement solver for preconditioning is replaced by one V-cycle of a pressure Schur complement multigrid solver (see Section 5.1.1.2) performing one pre- and one postsmoothing step. The **A**-systems in the matrix vector multiplications with **S** have to be solved only approximately.

f) `BiCG-SCBTria`:
A BiCGstab solver applied to the whole saddle point system using the block triangular preconditioner SCBTria (see equation (5.10) on page 256).

g) `BiCG-MG(1,V11)-SCBTria`:
An extension of the previous solver, where the outer BiCGstab solver uses one multigrid iteration (applied to the whole saddle point system) as preconditioner which is then smoothed by the block preconditioner SCBTria.

To facilitate a fair comparison, we confine ourselves to solvers with an outer BiCGstab scheme. This is, of course, not always necessary, especially on 'simple' configurations. But from the tests in Section 4.2.7.3 we already know that in certain situations an outer Krylov scheme is mandatory. Hence, we will perform *all* tests with such solvers, accepting that for 'simple' configurations the costs could eventually be slightly reduced by using less complex solver schemes (e. g., `MG-SCBTria` instead of `BiCG-MG(1,V11)-SCBTria`).

We want to emphasise again that the strategy to perform exactly one (unaccelerated) PSC iteration for preconditioning/smoothing is actually very similar to the SCBTria approach, such that the boundary between these two concepts blurs. The PSC approach is more flexible than SCBTria in the following sense: Considering the (similar) solvers `BiCG-SCBTria` and `BiCG-PSC(1)`, the latter can be 'enhanced' by performing more preconditioning iterations, e. g., `BiCG-PSC(2)`, by using a PSC-Krlyov method as preconditioner, e. g., `BiCG-PSCBiCG(1)`, or by gaining digits instead of performing a fixed number of iterations, e. g., `BiCG-PSCBiCG(1e-1)`. In our experience, however, the total solver efficiency does usually not benefit from these 'enhancements'. On the contrary, nesting two Krylov methods (`BiCG-PSCBiCG`) sometimes even results in unstable solver behaviour (cf. Section 2.6.3). For these reasons, we consider the PSC variants listed above as the only reasonable ones.

2. All of the described saddle point solvers need to solve subsystems with the matrix **A**, either for performing the (exact or inexact) matrix vector multiplication with the Schur complement matrix (PSC), or for preconditioning/smoothing (SCBTria). Such solvers

are described and tested in detail in Chapter 4. In the current chapter, we mainly focus on the outer saddle point solver, hence we do not perform as detailed comparisons of the inner solvers for the **A**-systems as in Section 4.2.6. We confine ourselves to the following solvers:

a) `MG(1,V22)-Jac`:
One step of a standard V-cycle multigrid solver applied to the whole **A**-system with a *pointwise* Jacobi smoother, performing two pre- and postsmoothing steps. No scalar subsystems have to be solved, i. e., the definition of a scalar ScaRC solver is not necessary. The coarse grid problem is treated by a direct solver. Due to the pointwise Jacobi smoother, the solver is only suited for isotropic grids.

b) `BSor`:
One application of a block SOR preconditioner. This means, that actually only three blocks of the 2×2-block matrix **A** are involved, i. e., the two diagonal blocks $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ and the lower left block $\mathbf{A}_{21}$. For the solution of the scalar subsystems with the two diagonal matrix blocks a scalar ScaRC solver has to be defined.

c) `MG(1,V11)-BSor`:
One step of a V-cycle multigrid solver smoothed by the block preconditioner `BSor`. In contrast to the previous scheme, the entire matrix **A** is treated. In contrast to the multigrid solver under point 2a, we need scalar ScaRC solvers to treat the two diagonal blocks $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$. The coarse grid problem is solved by a direct method.

In Chapter 4 we identified the solver `BiCG-MG-BSor` as the most efficient one. This, however, does not mean that it is automatically best suited as subsolver within a saddle point solver. Actually, we found that using a BiCGstab scheme for the inner **A**-solvers (e. g., `BiCG-BJac/BSor` and `BiCG-MG(1)-BJac/BSor`) rendered the outer solver unstable in certain situations. For instance, it converged efficiently for the grid levels 3 and 5, but diverged on level 4. However, also for those test configurations, where the inner BiCGstab scheme did not lead to instabilities, it did not (or only very slightly) improve the efficiency of the overall solver. For these reasons, we do not use BiCGstab for the solution of the **A**-systems. We want to note that similar problems occurred when using FGMRes instead of BiCGstab, hence, this is not an option either.[5] Instead of the block preconditioner `BSor`, we also performed all tests with the block Jacobi preconditioner. The results were similar to those for the pure displacement formulation presented in Section 4.2.7: In the majority of all cases, `BSor` results in a better total arithmetic efficiency, and in the case of some more complicated configurations (see Section 5.2.1.6), `BJac` even leads to divergence while `BSor` is successful. Hence, we completely omit the results for `BJac` and once again suggest to always use `BSor` instead.

[5] The possibly negative effect of nested Krylov solvers we already observed in the case of the scalar ScaRC solvers in Section 2.6.3.

3. The solvers employing BSOR preconditioners need to solve scalar subsystems. Again, since we focus on the outer saddle point solvers here, we do not perform as detailed comparisons as in Chapter 2. We confine ourselves to the following 1-layer-ScaRC solvers:

   a) `MG(1,V22)__JAC`:
   In the case of isotropic grids, we use one V-cycle of a standard multigrid solver with Jacobi smoothing and damping parameter 0.7.

   b) `MG(1,V22)__ADI`:
   In the case of anisotropic grids, we use ADITriGS smoothing and the damping parameter 1.0.

4. For diffusive and reactive Schur complement preconditioning, we use the same ScaRC solvers as described under point 3.[6]

5. The solver algorithms described above employ multigrid schemes on different layers. We use the following coarse grid solvers:

   a) `PSCBiCG(1e-1)`:
   In the case of the saddle point solver `BiCG-PSCMG`, the coarse grid problem can not be solved by a direct solver since the Schur complement matrix is only implicitly available. Instead, we use `PSCBiCG(1e-1)` as iterative coarse grid solver, in which the $\mathbf{A}$-systems are solved by a direct solver. Gaining more than one digit did not influence the outer solver behaviour in the tests we performed.

   b) `CG(1e-1)-JAC`:
   In the case of transient problems, a scalar Poisson problem with pure Neumann boundary conditions has to be solved for reactive preconditioning. The resulting singular system matrix can not be treated by a direct coarse grid solver. Instead, we gain one digit with a Jacobi preconditioned CG solver, that uses the 'shifting technique' described in Section 5.1.2.2 to obtain a unique solution.

   c) `UMFPACK`:
   For all the other multigrid schemes, we use a direct UMFPACK solver for treating the coarse grid problems, either applied to the complete 3×3 system $\mathcal{A}$ (`BiCG-MG-SCBTria`, `BiCG-MG-PSC`), or to the 2×2 system $\mathbf{A}$ (`MG-Jac`, `MG-BSor`), or to scalar systems (all ScaRC solvers).

All multigrid solvers perform V-cycles with either one or two pre- and postsmoothing steps as described in the list. Since this setting is constant for all tests, it is not denoted anymore from now on. Furthermore, in most tests all the subsolvers perform exactly one iteration instead of gaining digits. Hence, to keep the notations short, we omit the parameter '`(1)`' in these cases. Instead, when we gain digits, this is emphasised by the notation of, e. g., '`(1e-1)`'.

[6] For the non-lumped mass matrix for diffusive preconditioning, a (single-grid) Krylov solver shows level-independent behaviour, as well, and would actually be sufficient. However, in our tests the multigrid solver was more efficient.

Subsolvers are denoted in square brackets. For instance, `BiCG-MG-SCBTria[MG-Jac]` is a BiCGstab solver, preconditioned by one iteration of a V-cycle multigrid solver that performs one pre- and postsmoothing step with the SCBTria smoother, which uses one iteration of a V-cycle multigrid solver with two Jacobi pre- and postsmoothing steps for treating the $\mathbf{A}$-systems. `PSCBiCG[MG(1e-8)-Jac]`, for instance, is a BiCGstab accelerated PSC solver for which the $\mathbf{A}$-systems are treated by a V-cycle multigrid solver with two Jacobi pre- and postsmoothing steps that gains eight digits.

We want to emphasise that the different solver combinations contain up to *three nested multigrid schemes*: For the two saddle point solvers `BiCG-MG-PSC` and `BiCG-MG-SCBTria` a multigrid scheme acts on the whole 3×3 block saddle point system, while for the solver `BiCG-PSCMG` it acts on the Schur complement system, which implicitly involves all nine blocks, too. In the next stage, the $\mathbf{A}$-solvers `MG-Jac` and `MG-BSor` contain a multigrid scheme acting on the 2×2 block system $\mathbf{A}$. Then, on the third stage, the block preconditioner `BSor` applies scalar ScaRC multigrid solvers to treat the two diagonal blocks. This means, that the solver `BiCG-SCBTria[BSor]`, for instance, applies only one multigrid scheme (to scalar subsystems), while the solver `BiCG-MG-PSC[MG-BSor]`, for instance, contains three nested multigrid schemes.

5.2.1.2 Tests on the Unitsquare with Analytical Solutions

We begin the numerical test series with a basic configuration, i. e., the unitsquare with a known analytical solution. The right side of the unitsquare is set to Neumann, the other three sides to Dirichlet boundary conditions. We use incompressible material with $\mu = \nu = 0.5$. The right hand side is set according to the analytical functions

$$\begin{aligned} \boldsymbol{u}(x_1,x_2) &= \begin{pmatrix} x_1^2(1-x_1)^2 2\pi \sin(\pi x_2)\cos(\pi x_2) \\ -(2x_1(1-x_1)^2 - 2x_1^2(1-x_1))\sin^2(\pi x_2) \end{pmatrix}, \\ p(x_1,x_2) &= \sin(x_1)\cos(x_2) + (\cos(1)-1)\sin(1), \end{aligned} \tag{5.22}$$

which are also used by Bochev et al. [28]. The displacement solution is divergence free and zero on the boundary of the unitsquare. For the solution of the $\mathbf{A}$-systems we use the standard `MG-Jac` solver (point 2a in Section 5.2.1.1), i. e., we do not have to employ scalar ScaRC solvers. The outer iterations are stopped when the initial residual norm is reduced by a factor of 10^{-8}.

At first, we want to examine the accuracy problems related to PSC-Krylov methods (cf. Section 5.1.1.2). Figure 5.2 shows the L^2 error between the exact and the computed displacement solution for four solvers variants on different refinement levels. One can observe that the accuracy of the BiCGstab-accelerated pressure Schur complement method indeed depends on the accuracy the $\mathbf{A}$-systems are solved with: When only one digit is gained (`PSCBiCG[MG(1e-1)-Jac]`), the L^2 error stagnates; when eight digits are gained (`PSCBiCG[MG(1e-8)-Jac]`), we see the expected error reduction. Furthermore, the plot demonstrates the success of the two strategies to overcome the accuracy problems (see Section 5.1.1.2): Both the relaxation technique (`PSCRelaxBiCG[MG(var)-Jac]`) and using PSC as preconditioner for an outer Krylov method

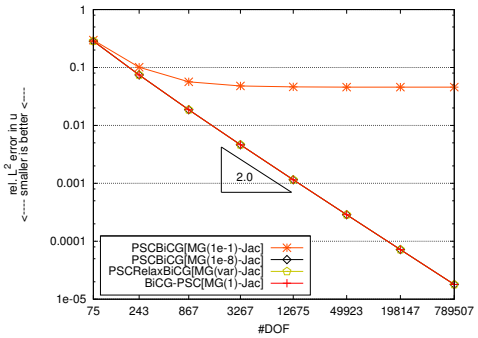
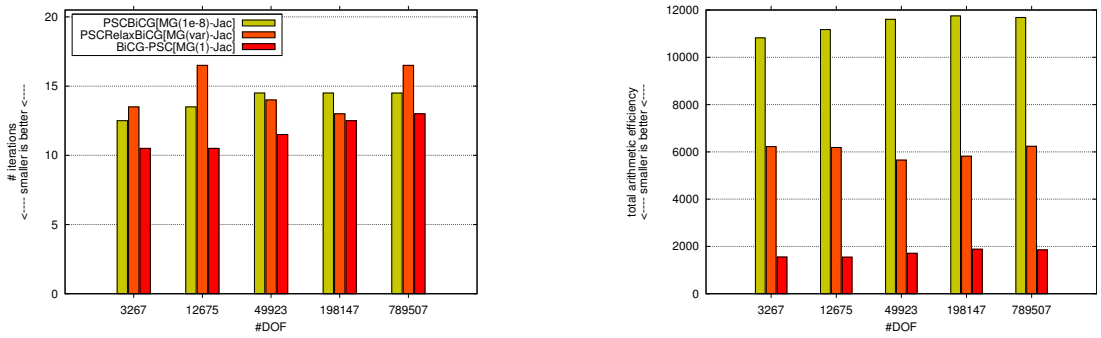


Figure 5.2: Displacement L^2 error for four different PSC solvers on the analytical test configuration.

(`BiCG-PSC[MG(1)-Jac]`[7]) lead to correct solutions, despite of the (adaptively) reduced accuracy of the inner solves. All the other solvers that are compared in the following, obtained the same accuracy as `PSCRelaxBiCG[MG(var)-Jac]`, so we omit these results in Figure 5.2 for the sake of a clearer representation. The solver `PSCBiCG[MG(1e-1)-Jac]` is not considered anymore in the subsequent tests due to its inaccuracy.

In Figure 5.3 we compare the three remaining solvers from Figure 5.2 with respect to iteration counts and total arithmetic efficiency (see definition (2.15) on page 58). Note that the two



(a) Number of iterations.

(b) Total arithmetic efficiency.

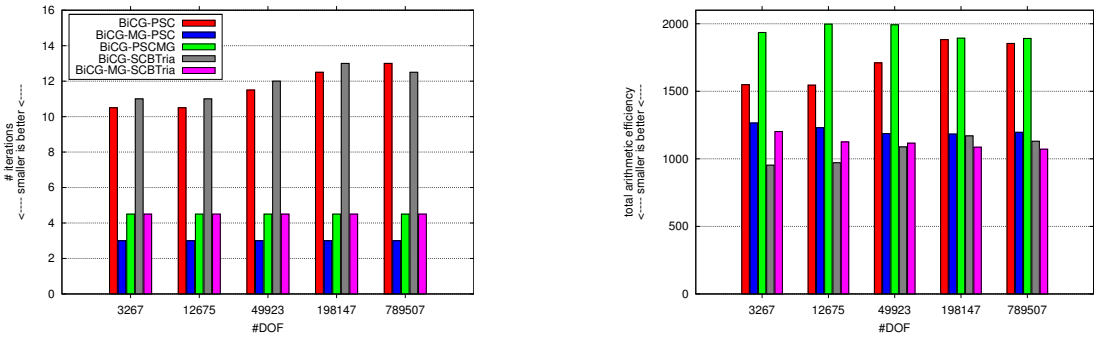
Figure 5.3: Results of the analytical tests on the unitsquare for three variants of PSC solvers. Both plots share the colour key of Figure 5.3a.

solvers `PSCBiCG[MG(1e-8)-Jac]` and `PSCRelaxBiCG[MG(var)-Jac]` use a relative stopping criterion for solving the $\mathbf{A}$-systems, while the third solver `BiCG-PSC[MG(1)-Jac]` performs exactly one iteration. One can observe in Figure 5.3a that the solvers' behaviour is, in principle, level independent: The iteration numbers of `PSCBiCG` and `BiCG-PSC` only slightly increase, while those of `PSCRelaxBiCG` exhibits slight oscillations. This is due to the fact that the relaxation strategy ensures the correctness of the result, though, but still can influence the convergence behaviour of the outer scheme [148]. More interesting is the comparison of the actual

[7] The number of inner multigrid iterations (`MG(1)`) is denoted here only for the sake of a clear distinction. In further tests, this notation is omitted again since it is the standard setting.

solver costs displayed in Figure 5.3b. As expected, the solver `PSCBiCG[MG(1e-8)-Jac]` is the most expensive one due to the accurate solutions of the inner **A**-systems for each matrix vector multiplication with the Schur complement matrix **S**. The relaxation strategy reduces the costs roughly by of factor of two, which is in line with the findings of van den Eshof et al. [166]. The second strategy, i. e., applying a PSC method as preconditioner for an outer Krylov method, however, achieves a cost reduction by a factor of six. We want to note that we confirmed these cost relations between the three solvers in further numerical tests, which we do not present here. Hence, we state that *the second strategy is clearly more efficient than the relaxation strategy*. For this reason we omit the solvers `PSCBiCG` and `PSCRelaxBiCG` in all subsequent tests and only use `BiCG-PSC` for further comparisons.

We now extend the tests with the analytical solutions to the other types of saddle point solvers listed in Section 5.2.1.1, which provides a first impression how the solvers perform in comparison. All solvers apply exactly one iteration of the standard multigrid scheme `MG-Jac` to solve the **A**-systems, so we omit its notation. Figure 5.4 shows the number of iterations and the total arithmetic efficiency. As we mentioned in the beginning of Section 5.2.1, we only



(a) Number of iterations.

(b) Total arithmetic efficiency.

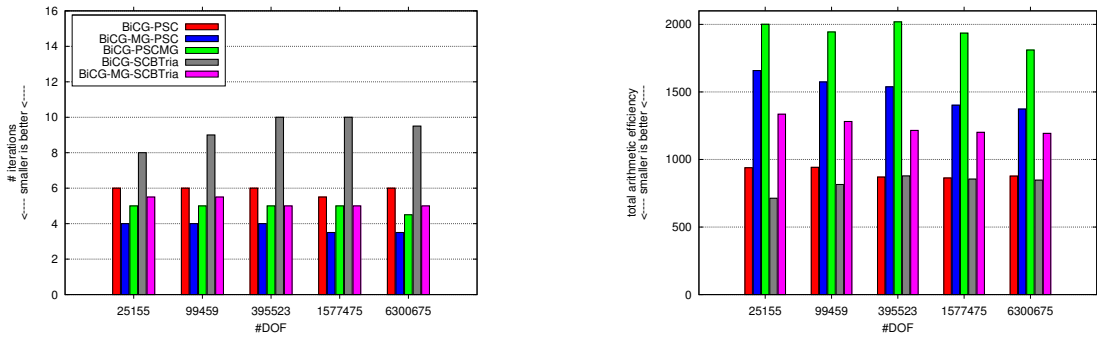
Figure 5.4: Results of the analytical tests on the unitsquare. Both plots share the colour key of Figure 5.4a.

consider the non-lumped mass matrix for Schur complement preconditioning. The corresponding scalar systems are treated by one iteration of the standard 1-layer-ScaRC solver `MG__JAC` described under point 3 in Section 5.2.1.1. Results for the lumped mass matrix are briefly discussed in Section 5.2.1.8. Figure 5.4a shows that the iteration numbers of the three solvers containing multigrid schemes applied to the whole system (`BiCG-MG-PSC`, `BiCG-PSCMG`, `BiCG-MG-SCBTria`) show perfectly level independent behaviour. The other two solvers that apply multigrid schemes merely for solving subsystems (`BiCG-PSC`, `BiCG-SCBTria`) perform more iterations and show a slight increase of iteration numbers. Nevertheless, they can still be regarded as generally level independent. Obviously, the amount of arithmetic work per outer iteration differs significantly for the various solvers, so we have to look at the actual solver costs in Figure 5.4b. The solvers `BiCG-PSC` and `BiCG-PSCMG` are clearly the most expensive ones. The three solvers `BiCG-MG-PSC`, `BiCG-SCBTria` and `BiCG-MG-SCBTria` perform best, where a clear favourite can not be determined due to slight variations over the different refinement lev-

els. It is interesting to observe that, when comparing `BiCG-SCBTria` and `BiCG-MG-SCBTria`, the 'inserted' multigrid scheme does stabilise the iteration counts, but it does not significantly decrease the arithmetic costs. The additional multigrid scheme within the solver `BiCG-MG-PSC`, however, clearly lowers the costs compared to the solver `BiCG-PSC`. A comparison of `BiCG-MG-PSC` and `BiCG-PSCMG` shows, that it seems to be more efficient to apply a multigrid solver to the whole 3×3 saddle point system instead of to the Schur complement system.

5.2.1.3 Tests for an Isotropic Real Configuration

We now consider the more realistic BLOCK-4X2-ISO configuration with an isotropic grid (see Figure 4.1 on page 200) using a nearly incompressible rubber material with shear modulus $\mu = 8.2$ MPa and Poisson ratio $\nu = 0.49999$. We apply a vertical line force of -10 MPa. We employ the same solvers as for the previous test in Figure 5.4. Again, we only examine the case of the non-lumped Schur complement preconditioner, the other case is briefly treated in Section 5.2.1.8. Comparing the iteration number for the BLOCK configuration (Figure 5.5a)



(a) Number of iterations.

(b) Total arithmetic efficiency.

Figure 5.5: Test results for the BLOCK-4X2-ISO configuration. Both plots share the colour key of Figure 5.5a.

with the analytical tests (Figure 5.4a), one can see that the results are qualitatively the same. The only remarkable difference is that the solver `BiCG-PSC` shows clearly lower and less varying iteration counts on the BLOCK configuration. Consequently, the solver `BiCG-PSC` now belongs to the most efficient ones (see Figure 5.5b), while the efficiency of the other solvers does not vary significantly.

5.2.1.4 Comparing Different A-Solvers

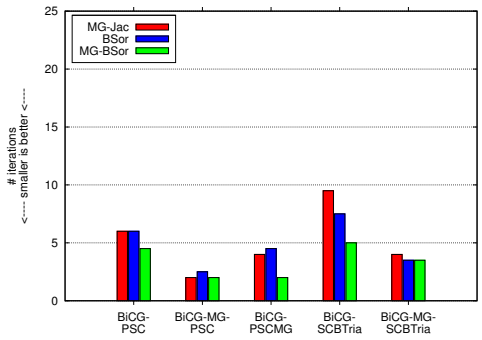
In this section we want to examine how the choice of the inner solver for the **A**-systems influences the behaviour of the outer schemes. We use three different configurations: the BLOCK-4X2-ISO configuration, already employed in the previous section, the BLOCKWITHHOLES configuration (see Figure 3.14a on page 172) with the same vertical line force of -10 MPa, and

the RUBBERBUSHING6 configuration (see Figure 3.14d on page 172), moving the inner circle by $\boldsymbol{u} = (-0.003\text{ m}, -0.003\text{ m})^{\mathsf{T}}$. For all three, we apply the same material parameters as in the previous section.

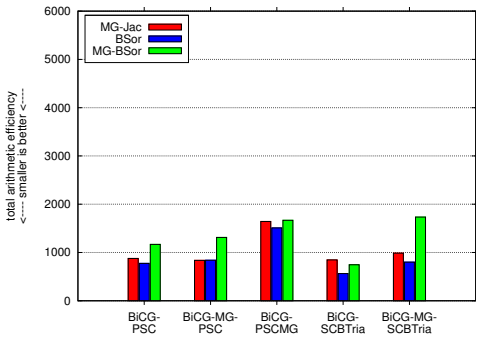
We consider the same five saddle point solvers as in the previous section, and vary between the three different inner **A**-solvers described under point 2 in Section 5.2.1.1. For the regular BLOCK-4X2-ISO grid we use the ScaRC solver `MG__JAC` to solve all scalar systems, on the two irregular grids we use `MG__ADI`. In order to keep the amount of data for the resulting 15 solver combinations manageable, we present in Figure 5.6 the iteration counts and total arithmetic efficiency for grid level 9 only, resulting in 6.30 M unknowns for BLOCK-4X2-ISO and RUBBERBUSHING6, and in 7.88 M unknowns for BLOCKWITHHOLES. On the two irregular grids BLOCKWITHHOLES and RUBBERBUSHING6 (Figures 5.6c–5.6f), the **A**-solver `MG-Jac`, i. e., multigrid with standard Jacobi smoothing, is not efficient: The standard damping parameter of 0.7 leads to divergence, such that it has to be reduced significantly, while, at the same time, the number of inner iterations has to be increased in order to achieve a sufficient preconditioning/-solving effect, in summary leading to a strong increase of arithmetic work. *Hence, we consider the inner solver* `MG-Jac` *unsuitable for these irregular grids and consequently disregard it.*

We can make the following observations in Figure 5.6:

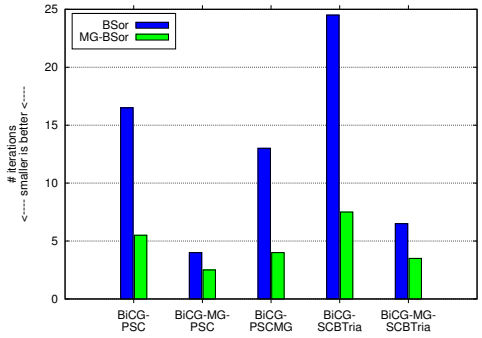
- On the BLOCK-4X2-ISO configuration (Figures 5.6a and 5.6b), there is no big difference whether to apply a multigrid solver to the whole 2×2 block matrix $\mathbf{A}$ (`MG-Jac`), or to the scalar subsystems $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ only (`BSor`) – both the number of iterations and the total arithmetic efficiency are very close to each other, with a slight advantage for `BSor`. Using both layers of multigrid (`MG-BSor`), decreases the number of iterations for most saddle point solvers, though, but it leads to higher arithmetic costs in all cases but one. Hence, we can state that the simple `BSor` block preconditioner is sufficient for treating the **A**-systems on this isotropic configuration.
- On the RUBBERBUSHING6 configuration (Figures 5.6e and 5.6f), the situation is similar: `MG-BSor` is able to decrease the iteration counts of all saddle point solvers, though, but the simple `BSor` block preconditioner is still more efficient in terms of arithmetic work.
- On the BLOCKWITHHOLES configuration the situation is not so clear: Here, the `MG-BSor` solver is able to lower the iteration counts even more. For the solver `BiCG-BSCTria`, for instance, the number of iterations is reduced by a factor of three, while it is reduced by not more than a factor of 1.5 on the other two grids. As a consequence, for all saddle point solvers but one, the **A**-solver `MG-BSor` is now superior in terms of the total arithmetic efficiency.
- From the tests in Figure 5.6, it is hardly possible to identify a favourite saddle point solver. There seem to be certain, not fully predictable interdependencies between outer solver, inner solver and shape/quality of the grid. For example, while the solver `BiCG-PSCMG[BSor]` is by far the most expensive one on the BLOCKWITHHOLES configuration, `BiCG-PSC[MG-BSor]` is the most expensive one on the RUBBERBUSHING6 configuration.



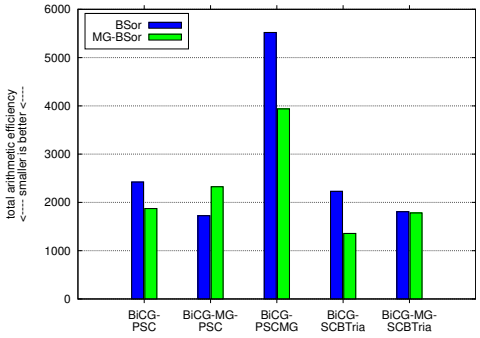
(a) BLOCK-4X2-ISO; number of iterations.



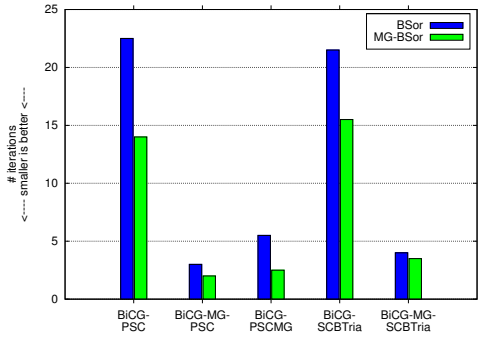
(b) BLOCK-4X2-ISO; total arithmetic eff.



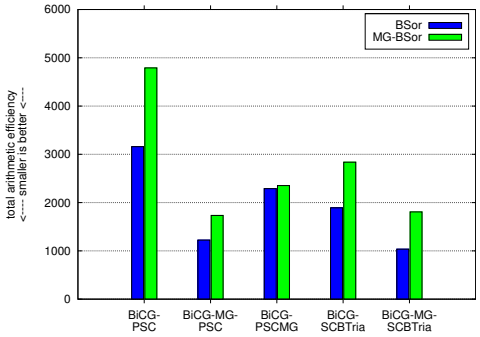
(c) BLOCKWITHHOLES; number of iterations.



(d) BLOCKWITHHOLES; total arithmetic eff.



(e) RUBBERBUSHING6; number of iterations.



(f) RUBBERBUSHING6; total arithmetic eff.

Figure 5.6: Comparison of three **A**-solvers, used in five saddle point solvers on three different grids; number of iterations (left) and total arithmetic efficiency (right).

- The two solvers `BiCG-MG-PSC` and `BiCG-MG-SCBTria` seem to exhibit the smallest oscillations in terms of iteration counts and total arithmetic efficiency, but they are not necessarily the cheapest ones. In order to obtain iteration numbers that are more or less independent of the domain shape and the grid quality, it seems to be necessary to apply multigrid either to the 2×2 block system **A**, or to the whole 3×3 block saddle point sys-

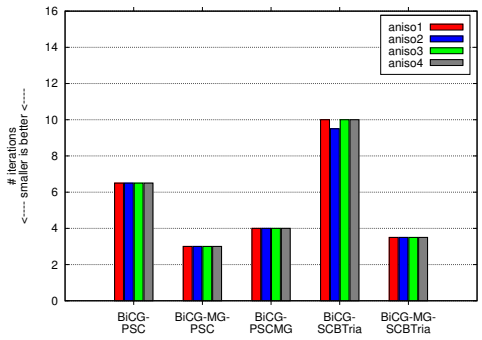
tem; the solvers `BiCG-PSC[BSor]`, `BiCG-SCBTria[BSor]` and even `BiCG-PSCMG[BSor]` show quite strong oscillations in terms of iteration counts and/or arithmetic costs.

5.2.1.5 Dependence on Mesh Anisotropies

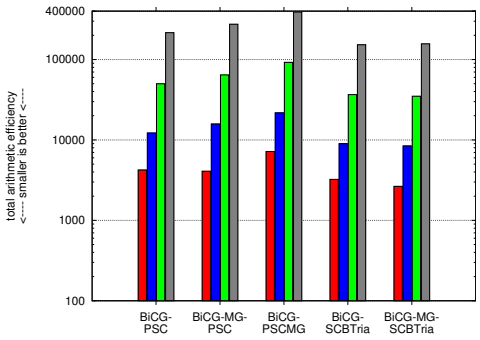
Continuing our search for a superior saddle point solver, we increase the degree of difficulty by introducing mesh anisotropies. We use the same material parameters as in the previous tests and the grids BLOCK-4X2-ANISO[1,2,3] from the corresponding pure displacement tests in Section 4.2.7.2 (see Figure 4.4 on page 206). Additionally, we use the grid BLOCK-4X2-ANISO4 which is created correspondingly, using the anisotropy factors $a_F = 0.0078125, a_L = a_I = 1.0$. The maximal element aspect ratios of the four grids are $\sigma = 3, 15, 63$ and 255, respectively. We consider the same solver combinations as before, neglecting the **A**-solver `MG-Jac` which is not applicable to anisotropic grids. All scalar systems are solved by the 1-layer-ScaRC solver `MG__ADI`; those within the block preconditioner `BSor` with varying accuracy (see below), those for Schur complement preconditioning always with exactly one iteration (`MG__ADI`). In the following figures the colour codes correspond to the four different grids in order to better focus on the dependency on mesh anisotropies. The five saddle point solvers are grouped together in one figure.

As a first test, we solve the scalar systems within the block preconditioner `BSor` exactly as it is done in Section 4.2.7.2, i. e., we gain eight digits (`MG(1e-8)__ADI`). The resulting number of iterations and the total arithmetic efficiency for the **A**-solver `BSor` are depicted in Figures 5.7a and 5.7b. On the one hand, one can see that the iteration numbers of all five saddle point solvers are independent of the degree of anisotropy (see Figure 5.7a). This means that the difficulties resulting from such anisotropies can be – as in the case of the pure displacement solvers in Section 4.2.7.2 – completely 'absorbed' by the scalar solvers. On the other hand, the total arithmetic efficiency in Figure 5.7b shows that this approach is much too expensive and thus not practicable. The plot also shows a dependency on the degree of anisotropy, which exclusively results from the additive Schwarz character of the scalar solvers (for an explanation, see the interpretation of Figure 4.12 on page 218 in Section 4.2.7.4). The decisive influence of the scalar ScaRC solvers is confirmed in the next test, where they only gain one instead of eight digits. One can see in Figure 5.7c, that now the numbers of iterations of the outer saddle point solvers slightly increase with the degree of anisotropy. This is due to the fact that the effects of the anisotropies are not completely caught by the scalar ScaRC solvers. Despite of this increase of iteration numbers, the arithmetic costs are by far smaller than in the previous test (compare Figure 5.7d with Figure 5.7b).

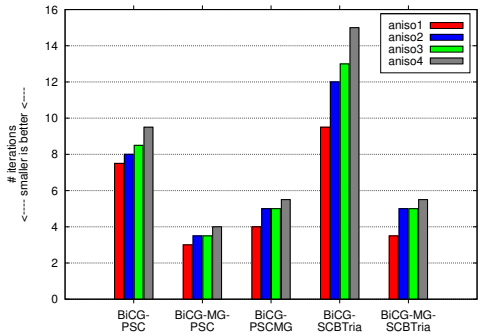
However, the arithmetic costs can be decreased even more when the scalar solver performs exactly one iteration instead of gaining digits. Figure 5.8a shows that the number of iterations in this case much stronger increases with the degree of anisotropy (note the logarithmic scale), though, but the arithmetic costs are – at least for the higher degrees of anisotropy – clearly smaller (compare Figure 5.8b with Figure 5.7d). We want to emphasise this difference between saddle point solvers and pure displacement solvers: In Section 4.2.7.4 we found out that the



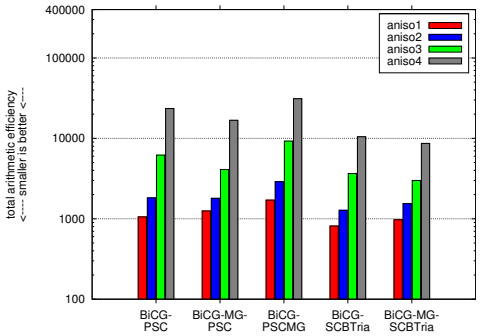
(a) Scalar solver `MG(1e-8)__ADI`; number of iterations.



(b) Scalar solver `MG(1e-8)__ADI`; total arithmetic efficiency.



(c) Scalar solver `MG(1e-1)__ADI`; number of iterations.



(d) Scalar solver `MG(1e-1)__ADI`; total arithmetic efficiency.

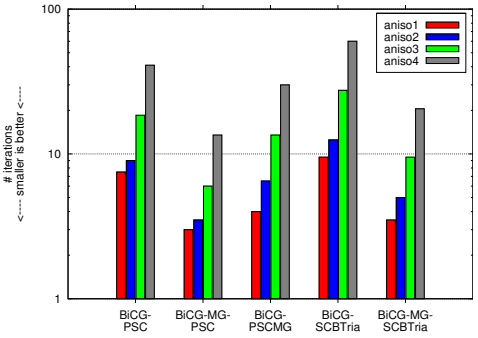
Figure 5.7: Dependence on mesh anisotropies; number of iterations (left) and total arithmetic efficiency (right) for five different saddle point solvers and **A**-solver `BSor`. Note the logarithmic y-scale of the right plots.

pure displacement solvers usually perform best when the scalar solvers gain one digit. In case of the saddle point solvers, however, we get the best results when the scalar solver performs exactly one iteration. This holds for *all* tests we performed.

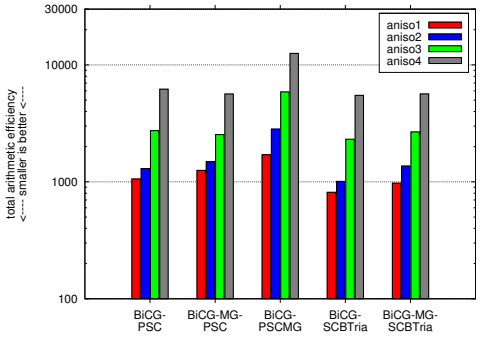
Comparing the five different saddle point solvers, one can see in Figure 5.8b that the solver `BiCG-PSCMG` is the most expensive one. The other four solvers are close to each other, with `BiCG-SCBTria` being slightly cheaper. Hence, for these configurations the additional multigrid scheme applied to the whole 3×3 system is not necessary (compare `BiCG-PSC` with `BiCG-MG-PSC` and `BiCG-SCBTria` with `BiCG-MG-SCBTria`).

Replacing the **A**-solver `BSor` by `MG-BSor` lowers the number of iterations, though, but the arithmetic costs are still higher in most cases (compare Figure 5.8c with Figure 5.8a and Figure 5.8d with Figure 5.8b). Hence, we can state that also for the **A**-solver the additional multigrid scheme is unnecessary.

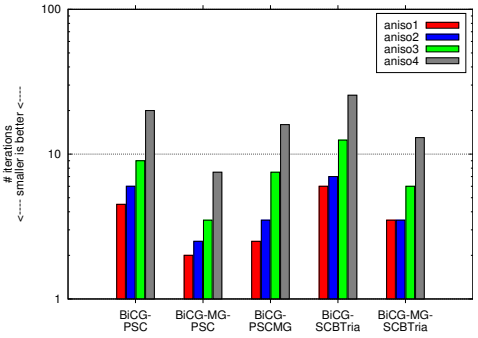
We finally want to confirm that the strong dependency of the arithmetic costs on the degree of anisotropy is indeed due to the anisotropic refinement towards the *inner* subdomain boundaries.



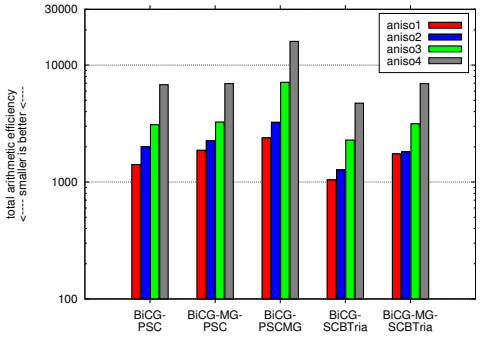
(a) **A**-solver `BSor`; number of iterations.



(b) **A**-solver `BSor`; total arithmetic eff.



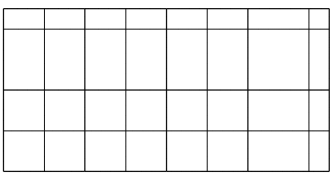
(c) **A**-solver `MG-BSor`; number of iterations.



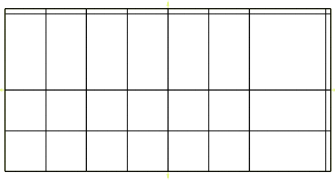
(d) **A**-solver `MG-BSor`; total arithmetic eff.

Figure 5.8: Dependence on mesh anisotropies; number of iterations (left), total arithmetic efficiency (right) for five different saddle point solvers and two different **A**-solvers; scalar solver `MG(1)__ADI`. Note the logarithmic y-scale.

To this end, we perform the same tests again, but the grid is now refined towards the top right corner of the domain, denoted with BLOCK-4X2-ANISO1-TR etc. Two of the four grids are depicted in Figure 5.9. We apply the same anisotropy factors as before, resulting in the same



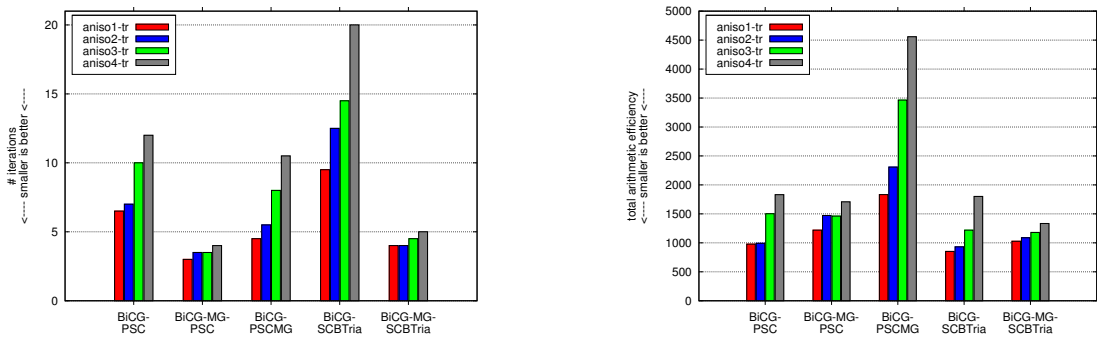
(a) BLOCK-4X2-ANISO1-TR.



(b) BLOCK-4X2-ANISO2-TR.

Figure 5.9: Two BLOCK configurations with anisotropic refinement towards the top right corner; level 1 depicted.

maximal element aspect ratios. Figure 5.10 shows the corresponding numbers of iterations and the total arithmetic efficiency. Again, the costs for the **A**-solver `MG-BSor` are higher in all cases, so we only present the results for the **A**-solver `BSor`. Comparing Figure 5.10a with Figure 5.8a,



(a) Number of iterations. **(b)** Total arithmetic efficiency.

Figure 5.10: Dependence on mesh anisotropies towards the top right corner; number of iterations (left) and total arithmetic efficiency (right) for five different saddle point solvers; **A**-solver `BSor`; scalar solver `MG(1)__ADI`.

one can see that the dependence on the degree of anisotropy is much weaker now: In the worst case the iteration number roughly doubles, while it increased by a factor of up to seven in case of the refinement towards the centre. The same is true for the arithmetic costs (compare Figure 5.10b with Figure 5.8b). In general, we can observe again that the solver `BiCG-PSCMG` is the most expensive one, and that the others are quite close to each other. However, we can make one further interesting observation now: While for the first two degrees of anisotropy the solver `BiCG-SCBTria` (`BiCG-PSC`) is slightly cheaper than the solver `BiCG-MG-SCBTria` (`BiCG-MG-PSC`), it is the other way around for the two configurations BLOCK-4X2-ANISO3-TR and BLOCK-4X2-ANISO4-TR. The reason is that the additional multigrid scheme sufficiently weakens the dependency on the degree of anisotropy (e. g., on BLOCK-4X2-ANISO4-TR an increase from 9.5 to 20 iterations for `BiCG-SCBTria`, but from 4 to only 5 iterations for `BiCG-MG-SCBTria`). One can expect that the solver `BiCG-MG-SCBTria` is even more superior for higher aspect ratios. In this sense, we can state a minor advantage for the solver using the additional multigrid scheme.

5.2.1.6 Dependence on Geometry

In this section we compare how the five saddle point solvers' convergence behaviour depends on the global domain anisotropy (cf. Sections 3.2.2.1 and 4.2.4). We use the same BEAM configurations as in the case of the pure displacement solvers in Section 4.2.7.3 (see Figure 4.6 on page 208). We increase the degree of difficulty by considering not only the three beams with global anisotropies of $\frac{L}{H} = 4, 16, 64$, but additionally the corresponding beams with global anisotropies of $\frac{L}{H} = 128, 256, 512$. We consider the isotropic refinement variants (BEAM-4X1-ANISO4, ..., BEAM-512X1-ANISO512) for which the number of (square-shaped) subdomains grows with the increasing global anisotropy, and the anisotropic refinement variants (BEAM-1X1-ANISO4, ..., BEAM-1X1-ANISO512), for which the grid always consists of exactly one subdomain with increasing element aspect ratios. We use the same material param-

eters as in the previous sections. The six different beams are loaded by vertical body forces of $-512, -16, -1, -0.1, -0.01$ and -0.001 $\frac{\mathrm{N}}{\mathrm{m}^3}$, respectively. We compare the same solver variants as in the previous sections, and the outer solver gains six digits. Again we consider only one multigrid level, i. e., level 9 for all anisotropically refined grids, resulting in 789.5 k DOF, and varying levels for the isotropically refined grids which are listed in Table 5.1.

configuration	level	# DOF
BEAM-4X1-ANISO4	8	790.3 k
BEAM-16X1-ANISO16	7	793.0 k
BEAM-64X1-ANISO64	6	798.9 k
BEAM-128X1-ANISO128	6	1597.6 k
BEAM-256X1-ANISO256	5	811.1 k
BEAM-512X1-ANISO512	5	1622.1 k

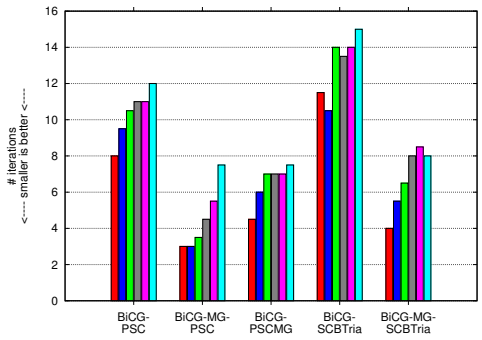
Table 5.1: Grid level and number of DOF for the six isotropically refined beams used in the numerical tests.

We first examine the easier case of the isotropically refined beams in Figure 5.11. Due to the isotropic elements, we can employ the simple ScaRC solver `MG__JAC` for all scalar systems, and in addition to the two **A**-solvers `BSor` and `MG-BSor` used in Section 5.2.1.5, we can also apply a standard multigrid solver with point Jacobi smoothing (`MG-Jac`). We can make the following observations:

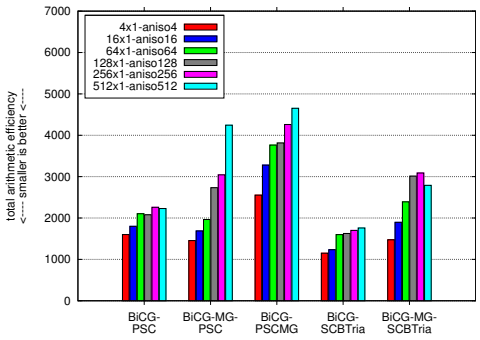
- Considering the **A**-solver `MG-Jac` first, we see that all saddle point solvers show a certain degree of dependency on the global anisotropy of the beam (see Figure 5.11a). The solver `BiCG-PSCMG` exhibits the slightest variations, but nevertheless, it is the most expensive one (see Figure 5.11b). Comparing the solver `BiCG-PSC` with `BiCG-MG-PSC` and `BiCG-SCBTria` with `BiCG-MG-SCBTria`, one can see that the additional multigrid scheme applied to the 3×3 system reduces the number of iterations, but increases the arithmetic costs at the same time. It is especially interesting that the additional multigrid scheme seems to render the overall solver costs more sensitive to the degree of anisotropy. While the costs for the solver `BiCG-PSC`, for instance, increase from 1600 to 2229 (factor 1.4), those for the solver `BiCG-MG-PSC` increase from 1455 to 4244 (factor 2.9). In summary, the solver `BiCG-SCBTria` is the cheapest one.

- Considering now the **A**-solver `MG-BSor` (see Figures 5.11c and 5.11d), the findings for the **A**-solver `MG-Jac` are in general confirmed. The iteration numbers of all five saddle point solvers are smaller than for the **A**-solver `MG-Jac`, but on the other hand the costs are clearly higher. We can state that the additional multigrid scheme on the scalar layer in terms of the block preconditioner `BSor` is not necessary for these configurations.

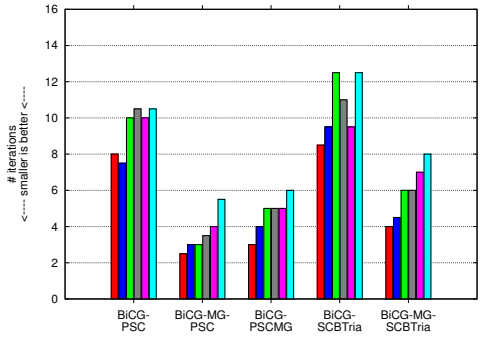
- The most interesting observations can be made for the **A**-solver `BSor` (see Figures 5.11e and 5.11f), i. e., the only one that does *not* use a multigrid scheme applied to the **A**-system. The missing bars mean that the corresponding solvers did not converge within 512 iterations or even diverged. We can see that the two solvers `BiCG-PSC` and `BiCG-SCBTria` are not able to solve the two longest beams, while the solver `BiCG-PSCMG` is



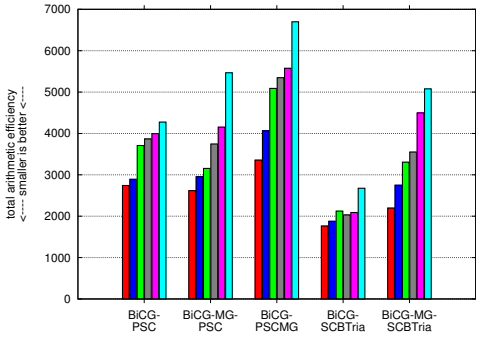
(a) **A**-solver `MG-Jac`; number of iterations.



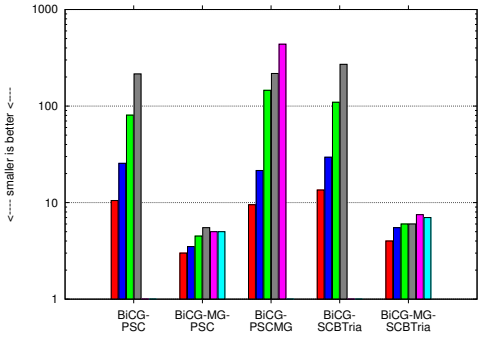
(b) **A**-solver `MG-Jac`; total arithmetic eff.



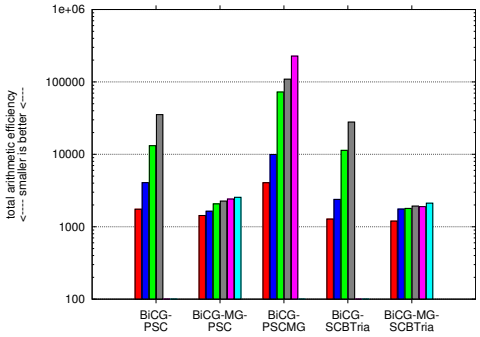
(c) **A**-solver `MG-BSor`; number of iterations.



(d) **A**-solver `MG-BSor`; total arithmetic eff.



(e) **A**-solver `BSor`; number of iterations.



(f) **A**-solver `BSor`; total arithmetic eff.

Figure 5.11: Dependence on domain geometry for the isotropically refined beams; number of iterations (left) and total arithmetic efficiency (right) for five different saddle point solvers and three different **A**-solvers; scalar solver `MG__JAC`. Note the logarithmic scale of the two lower plots. All plots share the colour key of Figure 5.11b.

not able to solve the longest beam. Furthermore, the costs of these three solvers for solving the other beam configurations are extremely high (note the logarithmic scale). However, the two solvers containing a multigrid scheme applied to the 3×3 system can quite robustly deal with all six beam configurations. This means, that it is mandatory to either apply multigrid to the whole 3×3 system, or, at least, to the **A**-system (see the

results for MG-BSor and MG-Jac). On the other hand, applying a multigrid scheme *only* to the scalar subsystems is clearly *not* sufficient. A surprising result is that also the solver BiCG-PSCMG[BSor], which applies a multigrid scheme to the Schur complement system, fails for the longer beams.

We want to compare the arithmetic costs of the five best solver combinations directly in one plot. Figure 5.12 shows that the solver BiCG-SCBTria[MG-Jac] is the cheapest for all six beam

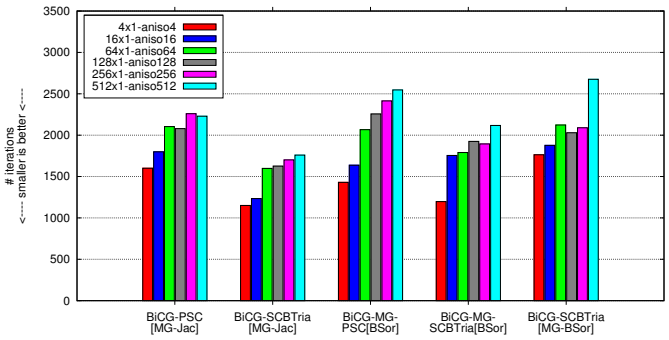
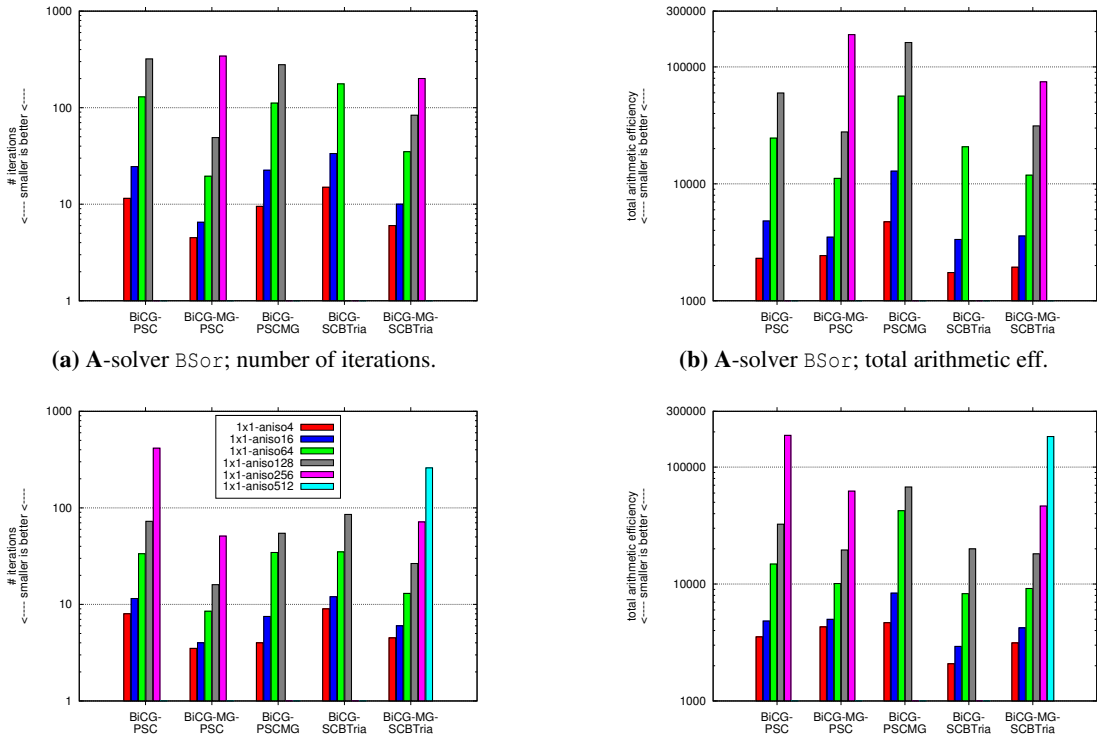


Figure 5.12: Total arithmetic efficiency of the five best solver combinations for the isotropically refined beams.

configurations. The performance of the first two solvers illustrates that the standard multigrid solver with point Jacobi smoothing (MG-Jac) is sufficient for solving the **A**-systems on these isotropically refined beam configurations. In summary, despite the slight performance differences that are visible in the figure, *all* five solver combinations can be regarded as efficient.

We now consider the six *anisotropically* refined beams, i. e., we additionally have to deal with high element aspect ratios. For these grids, the standard multigrid solver MG-Jac is not able to treat the **A**-systems, such that we only consider the **A**-solvers BSor and MG-BSor. For the solution of all scalar systems we use the ScaRC solver MG__ADI. Again, the missing bars in Figure 5.13 mean that the corresponding solver did not converge within 512 iterations or – as in most cases – even diverged.

The most important result is that the **A**-solver BSor is *not* sufficient: Figures 5.13a and 5.13b show that no saddle point solver is able to successfully treat the longest beam configuration, and only two solvers can solve the second longest beam. Switching from BSor to the **A**-solver MG-BSor, the number of outer iterations decreases for all cases, while the arithmetic costs increase in some cases and decrease in other cases (see Figures 5.13c and 5.13d). The solvers BiCG-PSCMG and BiCG-SCBTria still fail for the two longest beams, while the saddle point solver BiCG-PSC is now able to solve at least the BEAM-1X1-ANISO256 configuration. The solvers BiCG-MG-SCBTria is the only one that converges for all six beam configurations. Of course, the arithmetic costs are very high, but nevertheless this result is interesting: *It shows that for this kind of geometry and mesh refinement it is actually necessary to combine three layers of multigrid solvers.* For all the previous tests, it was sufficient to use only two or even only one multigrid solver. However, the failure of BiCG-MG-PSC for the longest beam shows that



(a) **A**-solver `BSor`; number of iterations.

(b) **A**-solver `BSor`; total arithmetic eff.

(c) **A**-solver `MG-BSor`; number of iterations.

(d) **A**-solver `MG-BSor`; total arithmetic eff.

Figure 5.13: Dependence on domain geometry for the anisotropically refined beams; number of iterations (left) and total arithmetic efficiency (right) for five different saddle point solvers and two different **A**-solvers. Note the logarithmic scale of the plots. All plots share the colour key of Figure 5.13c.

combining three multigrid solvers might be necessary, though, but still it does not guarantee convergence. [8]

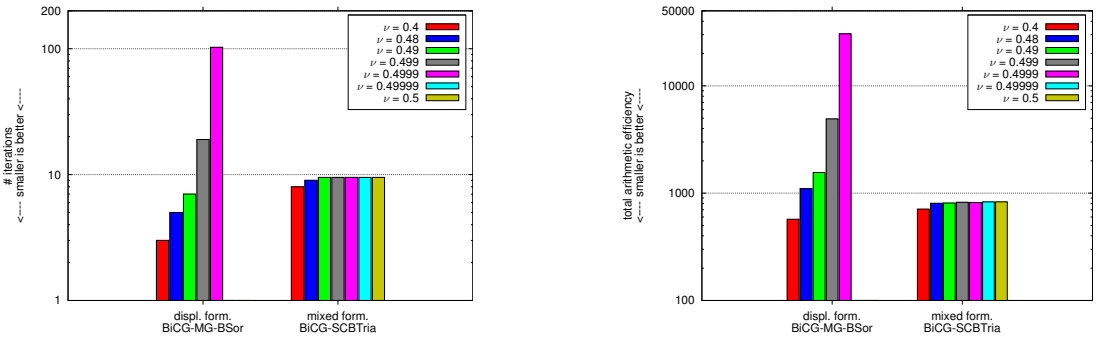
5.2.1.7 Dependence on Compressibility

In this section we briefly want to illustrate the robustness of the saddle point solvers with respect to the degree of compressibility, and their superiority over the pure displacement solvers. We use the BLOCK-4X2-ISO configuration on level 9 (i. e., 4.2 M DOF for the pure displacement and 6.3 M DOF for the mixed formulation) with standard loading (see Section 5.2.1.3), shear modulus of $\mu = 8.2$ MPa and Poisson ratio ν varying between 0.4 and 0.5. We exemplarily employ the pure displacement solver `BiCG-MG-BSor` and the saddle point solver `BiCG-SCBTria` which performed best within their respective solver class on this configuration. Both solvers

[8] Some further tests with the solver `BiCG-MG-PSC[MG-BSor]`, which are not presented here, showed that it *converges* for the longest beam configuration, when the outer multigrid solver performs two smoothing steps instead of only one. The costs, however, were nearly three times as high as for the solver `BiCG-MG-SCBTria[MG-BSor]`.

stop the iteration when the initial residual norm is reduced by the factor $\varepsilon = 10^{-8}$. The pure displacement solver uses the ScaRC solver `MG(e-1)__ADI`, while the saddle point solver uses `MG__JAC` for solving the occurring scalar systems. Remember that the ADITriGS smoother within the pure displacement solver is necessary due to the operator anisotropy induced by the incompressibility (cf. Table 4.1 on page 195).

Figure 5.14 shows that the pure displacement solver is superior to the saddle point solver only in the compressible case $\nu = 0.4$. For $\nu = 0.48$ and $\nu = 0.49$ it performs less iterations, though



(a) Number of iterations. **(b)** Total arithmetic efficiency.

Figure 5.14: Number of iterations (left) and total arithmetic efficiency (right) for varying values of ν; pure displacement formulation and mixed formulation. Note the logarithmic y-scale.

(see Figure 5.14a), but it is already more expensive in terms of total arithmetic efficiency (see Figure 5.14b). While the saddle point solver's behaviour is nearly independent of the degree of compressibility, the pure displacement solver's performance degrades dramatically with ν approaching the incompressible limit. The value of $\nu = 0.5$ is not valid for the pure displacement formulation ('$\lambda = \infty$'), but it already fails to converge for $\nu = 0.49999$. In order to further emphasise the strong difference, we note that for the case $\nu = 0.4999$ the pure displacement solver needs 823 seconds on a four processor machine, while the saddle point solver needs 33 seconds. Absolutely, it is 25 times faster although the linear system is 1.5 times larger.

5.2.1.8 Lumped vs. Non-Lumped Mass Matrix for Diffusive Preconditioning

In all the tests above we used the non-lumped mass matrix as diffusive preconditioner. In this section we want to compare it to the lumped mass matrix. On the one hand the lumped matrix is a diagonal matrix and thus much cheaper to invert, on the other the general expectation is that the preconditioning quality suffers and the number of outer iterations increases. With the help of some selected examples we want to show that the latter is surprisingly not always the case. Furthermore, we want to show that it depends on different factors whether the usage of the lumped mass matrix actually lowers the total arithmetic costs.

In Table 5.2 we present the iteration numbers and total arithmetic efficiency for selected configurations and solver combinations. Better, i. e., smaller values are printed in bold face. There

Configuration	Solver	#iter		TAE		row
		lump.	non-lump.	lump.	non-lump.	
BLOCK-4X2-ISO	BiCG-SCBTria[MG-Jac]	11	**9.5**	**777**	847	1
	BiCG-SCBTria[MG-BSor]	8	**5**	998	**746**	2
	BiCG-PSC[MG-Jac]	8	**6**	1077	**877**	3
BLOCKWITHHOLES	BiCG-SCBTria[BSor]	**22.5**	24.5	**1588**	2229	4
	BiCG-MG-SCBTria[BSor]	7	**6.5**	**1534**	1807	5
	BiCG-PSC[MG-BSor]	7	**5.5**	2218	**1872**	6
RUBBERBUSHING6	BiCG-SCBTria[BSor]	**19.5**	21.5	**1300**	1891	7
	BiCG-MG-SCBTria[BSor]	5.5	**4**	1160	**1035**	8
	BiCG-PSC[BSor]	**19**	22.5	**2511**	3159	9
	BiCG-PSC[MG-BSor]	14	14	**4366**	4790	10
BEAM-1X1-ANISO4	BiCG-SCBTria[BSor]	16.5	**15**	**1519**	1741	11
	BiCG-MG-PSC[BSor]	6	**4.5**	2887	**2432**	12

Table 5.2: Lumped vs. non-lumped diffusive preconditioner for selected configurations and solvers.

are basically three constellations:

1. The non-lumped mass matrix yields a smaller number of iterations *and* a better total arithmetic efficiency (rows 2, 3, 6, 8, 12).

2. The non-lumped mass matrix yields a smaller number of iterations, but the total arithmetic efficiency is worse (rows 1, 5, 11).

3. The lumped mass matrix yields a smaller (or equal) number of iterations *and* a better total arithmetic efficiency (rows 4, 7, 9, 10).

We consider the first two constellations first. Since the application of the non-lumped mass matrix is more expensive, it can only be advantageous when the iteration number is sufficiently decreased. The lower the costs of one iteration are, the more iterations must be saved. In row 1 of Table 5.2, for instance, the decrease by 1.5 iterations is not sufficient – the lumped matrix still results in a better total arithmetic efficiency. The decrease by 3 iterations in row 2 *is* sufficient. However, also choosing another outer solver can change the situation, as a comparison of row 1 and row 3 shows. Comparing row 5 to row 6 and row 11 to row 12 shows, that such situations also occur for other outer and inner solver combinations and other grids.

The third constellation, which occurs four times in the table, shows that one can *not* generally expect that using the non-lumped mass matrix leads to a better convergence of the outer solver.[9] Of course, when we obtain smaller iteration counts with the lumped mass matrix, then the arithmetic costs of the overall solving process are smaller, as well.

[9] We want to note that this is *not* due to the fact that we only perform one iteration to 'invert' the non-lumped mass matrix; inverting it more accurately (e. g., gaining eight digits) did not lower the outer iteration counts in our tests.

The different results in Table 5.2 show that it is hardly possible to predict whether the lumped or the non-lumped mass matrix is more efficient for a given configuration. It depends on the grid, on the choice of the outer and of the inner solver. An advantage of the lumped mass matrix clearly is that it needs less storage and that no (additional) scalar solver has to be set up. However, in the tests we performed, the convergence behaviour over several multigrid levels was often a little bit more consistent when using the non-lumped mass matrix (results are not presented here). Furthermore, due to our efficient ScaRC solvers, the usage of the non-lumped mass matrix does not have a large impact on the overall solver costs. When less robust scalar solvers were used, the lumped mass matrix might turn out to be the more efficient preconditioner for the majority of all cases.

5.2.2 Parallel Efficiency of Segregated and Coupled Methods

In this section we want to combine the comparison of segregated and coupled Vanka-type saddle point solvers with the examination of the methods' parallel efficiency. The Vanka methods described in Section 5.1.3 are only efficient for isotropic or mildly anisotropic meshes [175]. Hence, we confine ourselves to the BLOCK configurations which we already used to study the parallel efficiency of the pure displacement solvers (see Figure 4.17 on page 223). In order to examine the weak scalability of our saddle point solvers, we simultaneously increase the number of unknowns (by increasing the number of square-shaped coarse grid elements, respectively, subdomains) and the number of processors. All grids are refined ten times, such that the six configurations range from 12.6 M DOF (4 subdomains distributed to 4 processors) to 402.7 M DOF (128 subdomains distributed to 128 processors). We use the same settings as in the previous sections, i. e., a vertical surface load of -10 MPa and a nearly incompressible rubber material with shear modulus $\mu = 8.2$ MPa and Poisson ratio $\nu = 0.49999$. For the calculations we use up to 65 dual-processor compute nodes of the LiDO cluster of the TU Dortmund (for details, see Section 4.2.7.6).

As representatives for the *segregated* saddle point methods we consider four variants using the upper block triangular Schur complement preconditioner (see equation (5.11)). We use the non-lumped mass matrix, and all scalar systems are solved by the ScaRC solver `MG__JAC` (see point 3 in Section 5.2.1.1). On the other hand, as representatives for *coupled* methods we only consider the full patch-based and the diagonal patch-based Vanka smoothers (see Section 5.1.3.2) since they are the only ones being efficient for nearly incompressible material [175]. In detail, we compare the following solvers:

- `BiCG-SCBTria[BSor]`:
  A BiCGstab solver, preconditioned by the upper block triangular Schur complement preconditioner. The $\mathbf{A}$-systems are treated by one application of the block SOR preconditioner. Hence, there is one multigrid scheme involved, which is applied to scalar systems.

- `BiCG-SCBTria[MG-BSor]`:
  The same outer solver as the previous one, but the $\mathbf{A}$-systems are treated by one iteration

of a multigrid solver smoothed by block SOR. Hence, there are *two nested multigrid schemes*, one for the **A**-systems, and one for the scalar systems.

- `BiCG-MG-SCBTria[BSor]`:
  A BiCGstab solver, preconditioned by one multigrid iteration, which is smoothed by SCBTria. The **A**-systems are treated by one application of the block SOR preconditioner. Hence, there are *two nested multigrid schemes*, one applied to the whole 3×3-system, and one to the scalar systems.

- `BiCG-MG-SCBTria[MG-BSor]`:
  The same outer solver as the previous one, but the **A**-systems are treated by one iteration of a multigrid solver smoothed by block SOR. Hence, there are *three nested multigrid schemes*, one applied to the whole 3×3 system, one to the 2×2 **A**-systems one to the scalar systems.

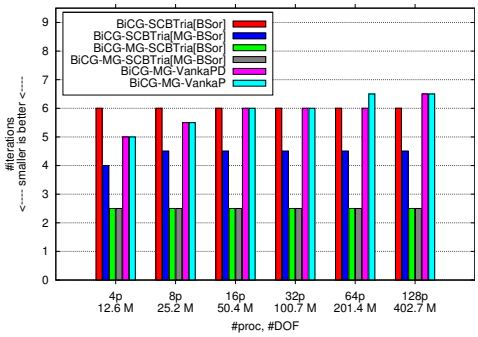
- `BiCG-MG-VankaPD`, `BiCG-MG-VankaP`:
  A BiCGstab solver preconditioned by one multigrid iteration smoothed by the patch-based diagonal (`VankaPD`) and the patch-based full Vanka-smoother (`VankaP`), respectively. The relaxation parameter of the diagonal Vanka smoother is set to $\omega = 0.6$ and that of the full Vanka smoother to $\omega = 0.5$. These values are optimal for the present configurations, which had to be determined manually by performing some test computations.

All multigrid solvers applied to 2×2 or 3×3 systems perform one V-cycle with one pre- and one postsmoothing step. The scalar ScaRC solvers apply one V-cycle with two pre- and two postsmoothing steps. All solvers stop the iteration when the initial residual norm is reduced by the factor $\varepsilon = 10^{-6}$.

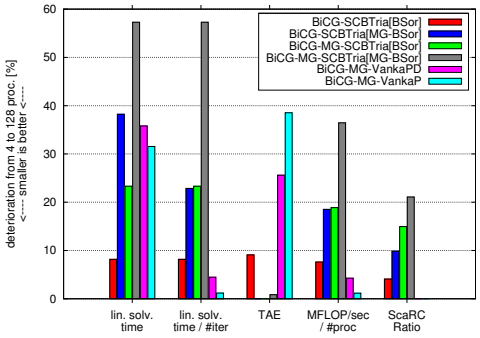
Figures 5.15 and 5.16 show various aspects of the parallel performance of the six solvers (cf. Sections 4.2.7.4 and 4.2.7.6). For some metrics the values of the segregated and the coupled methods differ so strongly that we have to separate the plots. The alternative, a logarithmic representation, would obscure some important details. Figure 5.15b displays the *deterioration of the solver performance from 4 to 128 processors* in some of these metrics, i. e., by how many percent the value for 128 processors increased or decreased compared to the value for 4 processors. For the metric 'MFLOP/s/#proc', deterioration corresponds to a *decrease* of the value, for the other metrics it corresponds to an *increase*. We can make the following observations:

- The iteration numbers of the four block-preconditioned variants (`SCBTria`) are independent of the number of sudomains, only the solver `BiCG-SCBTria[MG-BSor]` performs 4 instead of 4.5 iterations on the 4-processor grid (see Figure 5.15a). The total arithmetic efficiency only slightly varies (less than 10%) for the four solvers (see Figures 5.15c and 5.15b).[10]

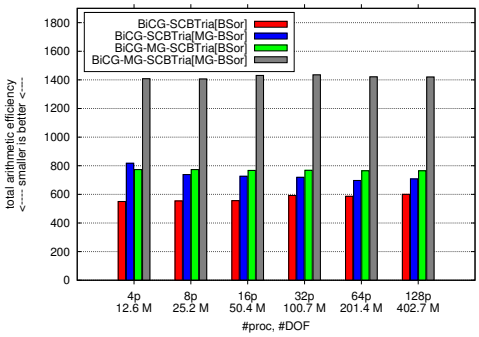
[10] The total arithmetic efficiency of the two solvers `BiCG-SCBTria[MG-BSor]` and `BiCG-MG-SCBTria[BSor]` even slightly improves which is not shown in Figure 5.15b.

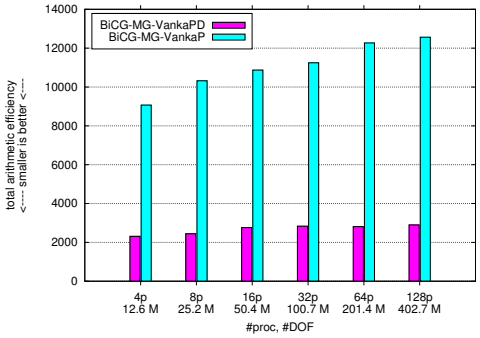


(a) Number of iterations.

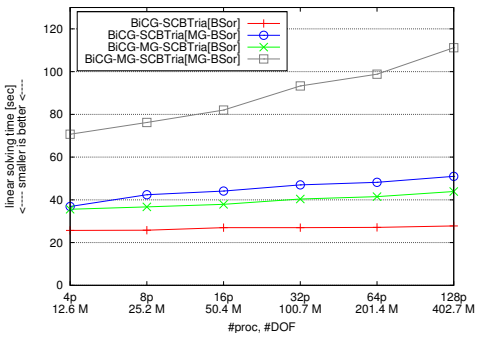


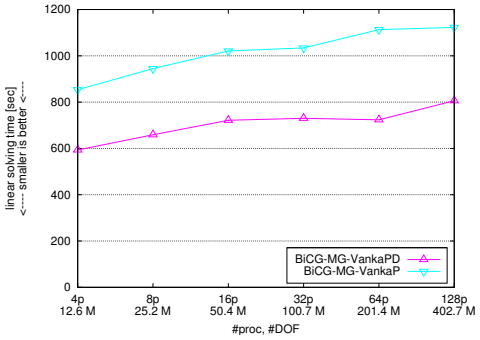
(b) Deterioration from 4 to 128 processors (%).





(c) Total arithmetic efficiency.

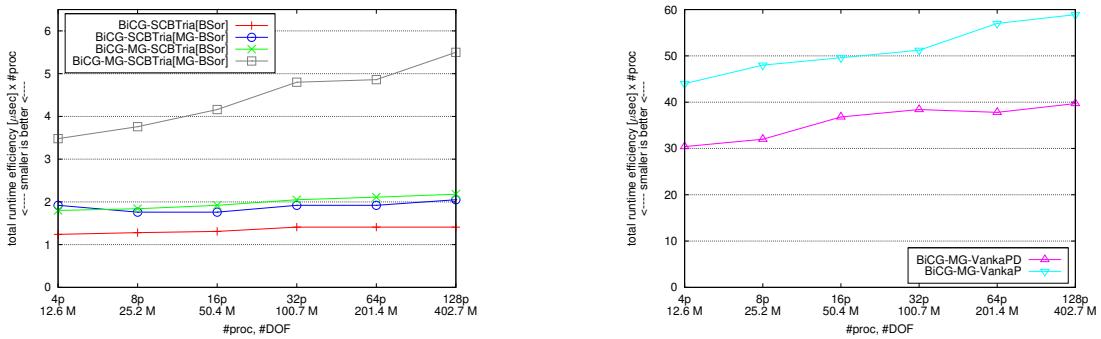




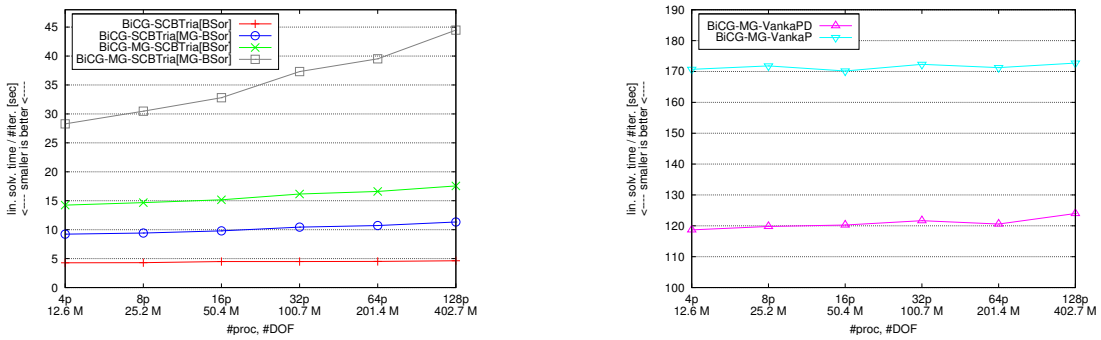
(d) Linear solving time.

Figure 5.15: Parallel performance of six different solvers on the six BLOCK configurations displayed in Figure 4.17 on page 223.

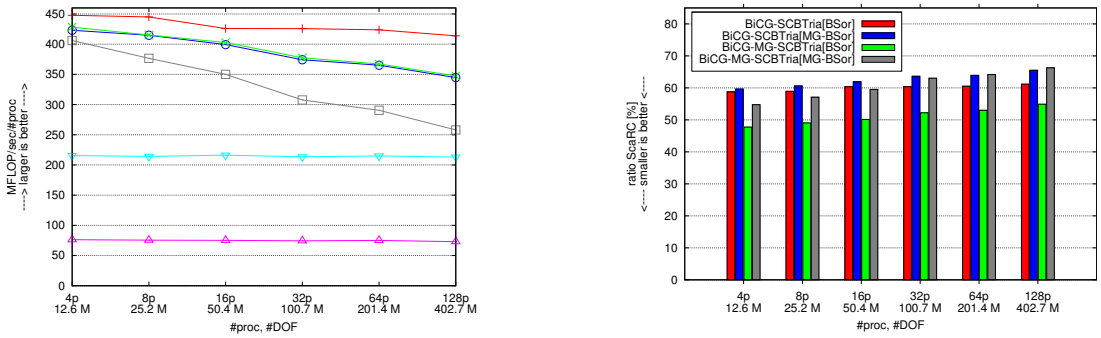
- The iteration numbers of the two Vanka solvers, however, increase from 5 to 6.5 (see Figure 5.15a), and the total arithmetic efficiency deteriorates by 25% and nearly 40% for `VankaPD` and `VankaP`, respectively (see Figures 5.15c and 5.15b). This dependence on the number of subdomains can be explained as follows: The strength of the Vanka smoothers comes from the *multiplicative* approach (block Gauß-Seidel) within each tensor product subdomain, i. e., the *successive* processing of the four-element patches and



(a) Total runtime efficiency multiplied with the number of processors.



(b) Linear solving time per iteration.



(c) MFLOP/s rate per processor.

(d) ScaRC ratio.

Figure 5.16: Parallel performance of six different solvers on the six BLOCK configurations displayed in Figure 4.17 on page 223.

the resulting immediate transport of information through the grid. This information stream, however, is 'interrupted' at the subdomain boundaries (see the explanation in Section 5.1.3.2). Consequently, the more subdomains the grid consists of, the more localised the information stream becomes.

- Among the segregated methods, the simplest one, `BiCG-SCBTria[BSor]`, is most efficient, followed by `BiCG-MG-SCBTria[BSor]` and `BiCG-SCBTria[MG-BSor]`, which are

quite close to each other. The most expensive one is `BiCG-MG-SCBTria[MG-BSor]`. This order was expected: In the previous sections we already found out that the additional multigrid schemes are only beneficial for more complicated configurations. The isotropic block configuration, however, is a very simple one.

- For the same reason, the full Vanka smoother is clearly more expensive than the diagonal one: The two solvers perform roughly the same number of iterations, while one iteration of the full Vanka smoother is of course much more expensive (see Section 5.1.3.2). Only on anisotropic meshes, for instance, the full Vanka smoother can be superior to the diagonal one [175].

  It is, however, interesting that the total arithmetic efficiencies of the two solvers differ by a factor of roughly 5, while the timings differ by a factor of 1.4 only (see Figures 5.15d and 5.16b). The reason for this can be found in Figure 5.16c: The MFLOP/s rate per processor of the diagonal Vanka solver is about three times worse than that of the full Vanka solver. The essential disadvantage of the Vanka smoothers is that the amount of actual arithmetic work, i. e., solving one small system per patch, is relatively small compared to the data organisation which is necessary to prepare this system, but which is not included in our FLOP statistics: The local matrix and the local residuum corresponding to the current patch have to be extracted from the global matrix and the global residuum, respectively. The contributions of the previously processed patches has to be included in the local residuum (Gauß-Seidel principle). After solving the system, the local correction vector has to be distributed to the corresponding entries of the global solution vector. The data organisation is the same for both Vanka variants. In case of the full Vanka smoother, however, we have to solve a dense 19×19 system with a direct solver, while for the diagonal Vanka smoother we only have to perform a few vector-vector operations (see the description in Section 5.1.3.2). Hence, the ratio of actual arithmetic work is larger for the full Vanka smoother which thus obtains a clearly higher MFLOP/s rate.

  Our implementation of the Vanka smoothers can be improved[11], but even when optimally implemented, one can not expect that they are able to obtain the MFLOP/s rates of the segregated methods: Figure 5.16c shows that the MFLOP/s rate of `BiCG-SCBTria[BSor]` is roughly two times as high as that of `BiCG-MG-VankaP` and nearly six times as high as that of `BiCG-MG-VankaPD`. While the segregated methods apply FEAST's cache-optimised techniques (see Section 2.1.1), the inherently *recursive* process of the Vanka smoothers, that leads to that disadvantageous ratio between data movement and data processing described above, prevents the achievement of a satisfying processor efficiency. The recursive process could be relaxed by colouring techniques, though, but even then we

[11] In our current Vanka implementation, the relation between global and local indices (i. e., those of the global matrix/residuum and those of the patch-wise created local matrices/residua) is computed on the fly for each subdomain and in each smoothing step. Although this process is already quite efficient due to the tensor product property of the subdomain meshes, it could be even more efficient, when the necessary local/global mappings were computed *once* in a preprocessing step and then stored in a corresponding data structure which could then be accessed more efficiently in *all* subsequent smoothing steps. This approach would, of course, significantly increase the storage requirements of the solver.

do not expect the Vanka techniques to achieve the processor efficiency of the segregated methods. Furthermore, the numerical efficiency usually suffers from such colouring techniques.

- In summary, when considering the arithmetic work (Figure 5.15c) and the absolute solving times (Figure 5.15d), the segregated methods are clearly superior to the coupled Vanka methods. The total arithmetic efficiency of the best segregated method `BiCG-SCBTria[BSor]` is up to 20 times better than that of `BiCG-MG-VankaP` and still up to five times better than that of `BiCG-MG-VankaPD`. The absolute timing differences are even more drastic: `BiCG-SCBTria[BSor]` is roughly 40 times faster than `BiCG-MG-VankaP` and still roughly 30 times faster than `BiCG-MG-VankaPD`.

- The most interesting result when comparing the four segregated methods to each other is that their parallel efficiencies differ significantly. The solving times per iteration in Figure 5.16b show this: While the solving time of `BiCG-SCBTria[BSor]` rises by only roughly 8% from 4.3 to 4.6 seconds (also see Figure 5.15b), that of `BiCG-MG-SCBTria[MG-BSor]` increases by 57% from 28.3 to 44.5 seconds. The other two solvers lie in between. These relations are confirmed by the total runtime efficiency (multiplied by the number of processors) displayed in Figure 5.16a.

  The reason for these differences is the number of nested multigrid schemes: While `BiCG-SCBTria[BSor]` applies only one multigrid scheme, `BiCG-SCBTria[MG-BSor]` and `BiCG-MG-SCBTria[BSor]` apply two, and `BiCG-MG-SCBTria[MG-BSor]` even three nested multigrid schemes. Nesting more multigrid schemes automatically means performing more work on the coarser grid levels and especially performing more direct coarse grid solves on the master process only. This results in a less favourable ratio between computation and communication (see Section 2.5.4 and also Section 4.2.7.6). As a consequence, the MFLOP/s rates drop correspondingly (see Figures 5.16c and 5.15b): While the MFLOP/s rate of `BiCG-SCBTria[BSor]` decreases by less than 8% (from 448 to 414), that of `BiCG-MG-SCBTria[MG-BSor]` loses 36% (from 406 to 258). The remaining two segregated solvers lie in between (decrease of 19%).

  This means, on more complicated configurations, where we *have* to combine two or even three multigrid schemes in order to obtain a satisfying numerical efficiency (see the tests in the previous sections), we have to accept a certain loss of parallel efficiency. For massively parallel computations, on the other hand, this could actually even mean that a numerically less efficient solver (e. g., involving only one multigrid scheme) is faster (in total runtime) than a numerically more efficient solver (e. g., involving two multigrid schemes).

- Compared to the segregated solvers, the Vanka solvers show a remarkably good ratio between computation and communication. Both the linear solving time per iteration and the MFLOP/s rate deteriorate by only 4 percent for `Vanka-PD` and by only 1 percent for `Vanka-P` (see Figures 5.16b, 5.16c and 5.15b). However, this good parallel efficiency is ‘paid’ with a loss of numerical efficiency as the increasing numbers of iterations in

Figure 5.15a show (for an explanation, see above). Furthermore, we suppose that on the processor numbers we tested the negative effect of increasing communication is simply superimposed by the overall bad performance of the Vanka solvers, and that the parallel efficiency *will* eventually deteriorate for a higher number of processors. The slight deterioration of the time per iteration (Figure 5.16b) and of the MFLOP/s rate (Figure 5.16c) from 64 to 128 processors might be an indicator for that. However, lacking a larger compute cluster we are not able to verify this assumption.

- Figure 5.16d shows that the ratio the solution schemes spent in scalar ScaRC solves ranges between 48% and 66% of the total solution time. For the relevance of this measure, see Section 4.2.7.4, page 216. The solver `BiCG-SCBTria[BSor]` shows a fairly constant ScaRC ratio of about 60% for all numbers of processors. For the other solvers the value increases more clearly (also see Figure 5.15b). This is due to the fact that these solvers call ScaRC more often and especially on coarser grid levels, such that ScaRC suffers disproportionately from the deteriorating ratio between communication and arithmetic work with increasing processor numbers.

  Comparing the ScaRC ratios for 4 processors with those obtained for the pure displacement solvers (see Figure 4.11 on page 216 and Figure 4.12 on page 218)[12], one can see that the ratios of the latter are clearly higher (over 80% for `BiCG-BSor` and still nearly 70% for `BiCG-MG-BSor`). This is mainly due to the fact that in the case of the pure displacement solvers we let the scalar ScaRC solvers gain one digit, while in the case of the saddle point solvers they perform exactly one iteration. Hence, we can conclude that saddle point solvers (as we specified them for our numerical tests) do not profit as much from enhancements of FEAST's scalar solver library as it is the case for the pure displacement solvers. If we, for instance, let the scalar ScaRC solvers within the saddle point methods also gain one digit instead of performing exactly one step, the ScaRC ratio will probably improve. However, in most cases, also the amount of total arithmetic work will increase, hence, it is questionable whether this strategy is advantageous. It is, especially, hardly possible to decide this *a priori*. In the case of transient computations, an additional scalar Laplace problem has to be solved for preconditioning the reactive part of the Schur complement (see Section 5.1.2.2). Hence, the ScaRC ratio will clearly improve in transient computations.

5.3 Summary and Future Work

In this chapter we described and compared various strategies to solve saddle point systems stemming from the mixed $\boldsymbol{u}/p$ discretisation of linearised elasticity problems with (nearly) incompressible materials. Saddle point solvers can be divided into segregated and coupled methods. The segregated methods can further be divided into pressure Schur complement solvers

[12] In the case of the pure displacements solvers, we performed *serial* computations. So, a comparison with the case of 4 processors is most appropriate.

(PSC) and methods applying block-triangular preconditioners (SCBTria). For the simple unaccelerated versions, the two classes in principle coincide, but there are important differences when the methods are accelerated by means of Krylov or multigrid schemes. As representatives of coupled saddle point solvers, we considered Vanka-type multigrid methods specially adapted for the stabilised Q_1/Q_1 element pair.

We emphasised three crucial differences between segregated and coupled methods: While only the former are able to exploit FEAST's ScaRC solving techniques and do not require special treatment of mesh anisotropies, the latter have the advantage that no Schur complement preconditioners are needed, but they require non-trivial enhancements to robustly handle mesh anisotropies.

Schur complement preconditioners for segregated methods were described for the stationary and the transient case, taking into account the compressibility and stabilisation terms leading to a nonzero block $\mathbf{C}$ in the saddle point system. We discussed the deficiency of the transient preconditioner for small time steps, which probably results from the Q_1/Q_1 stabilisation.

Extensive numerical studies were performed to evaluate and compare the described techniques to solve saddle point problems. The results can be summarised as follows:

- Of the two possibilities to overcome the accuracy problem of accelerated PSC methods, our strategy to use such methods as preconditioner within an outer solver is superior to the relaxation strategy.
- Among the segregated solvers, it is hardly possible to identify *the* best method. It depends on the given configuration, which solver performs best. As a general rule, one can say that for 'simple' configurations 'simple' solvers are sufficient, while more complicated configurations require more sophisticated solvers. For example, a saddle point solver using multigrid only on the layer of scalar subsolves is efficient on 'isotropic' domains like the BLOCK configuration, even for high element aspect ratios. But on strongly anisotropic domains like the BEAM configuration it is mandatory to apply multigrid schemes to the 2×2 system $\mathbf{A}$ or to the whole 3×3 saddle point system.
- Interestingly, the multigrid-accelerated PSC method (`BiCG-PSCMG`) is *not* efficient on the anisotropic BEAM domains, although it also treats the whole saddle point system – implicitly in terms of the Schur complement matrix – on all levels. Actually, the method performs worse than `BiCG-MG-PSC` in all our tests. This indicates that it seems to be more efficient to only let the smoother perform the implicit reduction to the scalar Schur complement system and let the other components (grid transfer, coarse grid solution) act on the 3×3 system (`BiCG-MG-PSC`), than to perform the whole multigrid algorithm on the basis of the scalar Schur complement system (`BiCG-PSCMG`).
- On the anisotropically refined beams (consisting of only one stretched subdomain) it even seems to be necessary to combine all three layers of multigrid solvers (scalar system, 2×2 system and 3×3 system). The solver `BiCG-MG-SCBTria[MG-BSor]` is the only one among the tested solvers that is able to solve all BEAM configurations. Of course,

it is not reasonable to apply such a complex, multiply nested multigrid solver to simple configurations. This is confirmed in the parallel tests performed on the isotropic BLOCK configuration. The solver not only needs much more total arithmetic work compared to simpler solver configurations, but due to the three nested multigrid schemes a relatively large part of this work is performed on coarser grid levels. Hence, the solver especially suffers from a bad ratio between computation and communication.

In summary, the numerical efficiency does not necessarily improve when nesting two or more multigrid schemes, but the parallel efficiency certainly deteriorates. Hence, one should try to use multiply nested multigrid schemes *only* if this is actually necessary from a numerical point of view (e. g., on the long BEAM configurations). Of course, for a given configuration it is usually not clear a priori which degree of solver complexity is actually necessary (especially, when nonlinearities come into play). So, it might be necessary to perform some initial trial tests before starting the actual simulation. If this is not possible, one should keep in mind that a certain loss of parallel efficiency is not as bad as a diverging solver.

- Comparing pressure Schur complement methods (PSC) and those using block-triangular preconditioners (SCBTria), our tests seem to indicate that SCBTria methods are numerically slightly superior. Additionally, this class of methods is much simpler to incorporate into existing solver implementations: It just has to be defined as a standalone preconditioning routine that receives the 3×3 saddle point system and a corresponding right hand side vector. Inside the routine, all necessary subsolves etc. are performed *independently of the calling solver method*. This means the outer solver does not have to take special care of the saddle point structure, i. e. one can employ exactly the same Krylov- or multigrid implementations as for solving the pure displacement problems described in Chapter 4. The outer solver simply acts on a 3×3 instead of a 2×2 system and calls the SCBTria preconditioner/smoother instead of, e. g., BJac or BSOR as in the pure displacement case.

  This is different for the PSC methods. They have to implicitly transform the 3×3 saddle point system into the scalar Schur complement system. This means that, e. g., all residual vectors, norms and matrix vector multiplications (with the only implicitly available Schur complement matrix) have to be calculated in a special way. Hence, all the accelerated versions PSCCG, PSCBiCGstab, PSCMG etc. *have to be fully reimplemented*; it is not possible to use the existing CG, BiCGstab, MG implementations and merely exchange the preconditioner/smoother as it is done in the case of the SCBTria approach. Roughly speaking, the amount of code doubles such that maintenance and future extensions of the linear solver library are correspondingly more complex, error-prone and time-consuming.

  Taking additionally the accuracy problem of the accelerated PSC methods into account which needs special care, we clearly favour the SCBTria approach.

- The question which is *the* best suited subsolver for the 2×2 $\mathbf{A}$ systems or for the scalar systems can not be answered in general. On isotropic meshes, for example, it suffices

to use a standard multigrid solver with point-Jacobi smoother (`MG-Jac`) to treat the $\mathbf{A}$ systems. This solver does not need to apply scalar ScaRC schemes. Alternatively, one can use one step of a BSOR preconditioner which applies the ScaRC solver `MG__JAC` to the two blocks $\mathbf{A_{11}}$ and $\mathbf{A_{22}}$. However, as soon as mesh anisotropies are present, we need to apply the ScaRC solver `MG__ADI` which has the ability to completely hide these irregularities from the outer solvers. The standard multigrid solver with point-Jacobi smoother can not be used to solve the $\mathbf{A}$ systems. Generally, we can state that – as in the case of the pure displacement formulation – the block preconditioner BSOR is more efficient than BJac.

For anisotropic domains like the BEAM configuration it is necessary to apply a multigrid scheme to the $\mathbf{A}$ systems if there is no multigrid scheme applied to the 3×3 saddle point system. For example, one can either use `BiCG-SCBTria[MG-BSor]` or `BiCG-MG-SCBTria[BSor]`. If additionally highly stretched elements are present, then even both layers of multigrid might be necessary. Applying multigrid *only* to the scalar layer (e. g., `BiCG-SCBTria[BSor]`) is *not* sufficient.

In general, we found that it is more efficient to let the scalar solvers only perform exactly one iteration. This is different from the pure displacement solvers in Chapter 4, where the scalar solvers gain one digit. As a consequence, the ratio of ScaRC solving times is smaller than in the case of pure displacement solvers. Furthermore, the solver that was identified as most efficient for the pure displacement case (`BiCG-MG-BSor`) is not well suited for solving the $\mathbf{A}$ systems within saddle point solvers: The total costs are in most cases higher than for the simple `MG-BSor` solver, and the additional inner BiCGstab scheme can even lead to instabilities of the outer solver in some situations.

- It depends on various factors whether the lumped or the non-lumped mass matrix for diffusive preconditioning performs better. One cannot say that one is generally superior to the other.

- Solvers basing on the mixed $\boldsymbol{u}/p$ formulation are significantly more robust than pure displacement solvers with respect to the incompressibility of the material. Already for values of $\nu \geqslant 0.48$ the mixed formulation is more efficient.

- Segregated solution methods are in most aspects clearly superior to coupled Vanka-type multigrid solvers. While the convergence rates of the latter deteriorate when the number of subdomains is increased, the numerical efficiency of the former is unaffected. The processor efficiency of the Vanka solvers is by far worse than that of the segregated methods: While the latter can exploit all the cache-aware implementations of the underlying SparseBandedBLAS library, the former suffers from the requirement to solve many small linear systems resulting in a large overhead of data organisation. Furthermore, the success of the Vanka-smoother highly depends on the correct choice of the relaxation parameter, which is very sensitive with respect to various aspects as the geometry of the

domain, mesh irregularities or operator anisotropies. The parameter has to be set manually by the user which has to be seen as a major disadvantage. The segregated solution methods, although being quite complex, do not contain such a critical parameter.

Only the parallel efficiency (i. e., the ratio of communication and computation) of the Vanka solvers seems to be better: While for the segregated methods the time per iteration slightly increases with increasing number of processors, that of the Vanka solvers stays nearly constant. However, we believe that the general bad performance of the Vanka solvers superimposes the negative effects of the increasing communication amount, and that with a higher number of processors these effects will be visible. Furthermore, this seemingly good parallel performance can not outweigh the bad numerical efficiency, i. e., the increasing number of iterations resulting from the increasing number of subdomains. While the total runtime of the segregated solver `BiCG-SCBTria` deteriorates by less than 10% from 4 to 128 processors, the total runtimes of the Vanka solvers deteriorate by more than 30%.

Future work should concentrate on the examination of the presented solving methods in the context of stationary nonlinear finite deformation problems on the one hand, and (linear and nonlinear) transient simulations on the other hand. For all these problem classes the main question is: *Are the presented Schur complement preconditioners efficient, or do they need substantial modifications?*

For the transient Schur complement preconditioner, we already identified severe difficulties for the case of small time steps, most probably a consequence of the Q_1/Q_1 stabilisation. It has to be examined if this is an unavoidable problem and one *has* to use sufficiently large time step sizes, or if there is a way to circumvent the occurring instabilities.

In the case of stationary linearised elasticity, it is known that the pressure mass matrix $\mathbf{M}_p$ is spectrally equivalent to the Schur complement $\mathbf{S} = \mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C}$ and is thus a good preconditioner. It is, however, unclear how it performs in the nonlinear case where the spectral equivalence is not guaranteed anymore. The choice of the constitutive law may have some influence on the efficiency of $\mathbf{M}_p$. Using the Neo-Hooke law, for example, results in a nonsymmetric saddle point system with a lower left block $\mathbf{D} \neq \mathbf{B}^\mathsf{T}$. Another typical example in the field of solid mechanics is to use non-isotropic materials, e. g., fibre materials which react differently across the fibres than along the fibres. For the 2D saddle point system, this means that, e. g., the two blocks $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ have completely different properties. In such a case, it is questionable if the use of the mass matrix as preconditioner is reasonable at all.

For the described situations it might be necessary to use completely different Schur complement preconditioning techniques. One can draw a parallel to the field of fluid dynamics: It is well known, that the simple diffusive Schur complement preconditioner is efficient for the (linear) Stokes system, but not necessarily for the (nonlinear and nonsymmetric) Navier-Stokes system, especially when the Reynolds number is large, i. e., when the convective terms dominate. This motivated many researchers to develop alternative Schur complement preconditioning strategies (see the brief overview at the beginning of Section 5.1). We want to mention one approach,

the so called BFBt preconditioner based on the commutation of operators, which might also be successful in the context of finite deformation elasticity. Elman et al. [66] suggest the following Schur complement preconditioner for the case of stable systems with a zero matrix block $\mathbf{C}$:

$$\mathbf{S}^{-1} = \mathbf{L}_p^{-1}\mathbf{B}^{\mathsf{T}}(\mathbf{M}^l)^{-1}\mathbf{A}(\mathbf{M}^l)^{-1}\mathbf{B}\mathbf{L}_p^{-1}.$$

The matrix $\mathbf{L}_p$ is the discrete pressure Laplacian and $\mathbf{M}^l$ is the displacement mass matrix in lumped form. The application of this operator on a given vector means two solves of scalar Laplace problems, two vector scalings with the inverse of the diagonal displacement mass matrix and three matrix vector multiplications with $\mathbf{B}$, $\mathbf{B}^{\mathsf{T}}$ and $\mathbf{A}$. Elman et al. [67] provide a version of this preconditioner for stabilised systems, which, however, is more intricate to realise. The BFBt preconditioner is discussed in detail by Olshanskii and Vassilevski [119]. In the same article, the following improved variant is introduced:

$$\mathbf{S}^{-1} = (\mathbf{M}_p^l)^{-1}\mathbf{B}^{\mathsf{T}}\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-1}\mathbf{B}(\mathbf{M}_p^l)^{-1}.$$

In contrast to the first version, one now has to invert two scalar lumped pressure mass matrices $\mathbf{M}_p^l$ and two displacement Laplacians $\mathbf{L}$. The two preconditioning variants are analysed and compared in detail by Olshanskii and Vassilevski [119]. For the second variant, a modification for unstable finite elements is provided, which is easier to realise than that of Elman et al. [67]. It is given by

$$\mathbf{S}^{-1} = (\mathbf{M}_p^l)^{-1}(\mathbf{B}^{\mathsf{T}}\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-1}\mathbf{B} - \mathbf{C})(\mathbf{M}_p^l)^{-1}.$$

One advantage of all three preconditioners is that they are composed of standard matrices that are readily available. Hence, they are relatively easy to incorporate into existing solver libraries. The decisive advantage, however, is that they contain the original matrix $\mathbf{A}$. Thus, all properties of this matrix are 'automatically incorporated' into the preconditioner, which makes this technique very attractive for nonlinear elasticity problems.

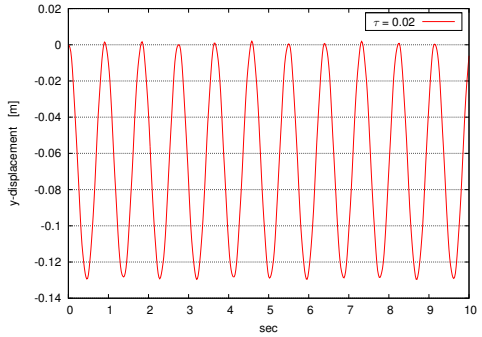
However, the preconditioners have been developed and tested with focus on the Navier-Stokes equation. Whether they are actually efficient for elasticity problems, as well, has to be examined. To our knowledge, this has not been done yet in the literature.

The possible difficulties to construct efficient Schur complement preconditioners for segregated solution methods in the context of nonlinear and transient elasticity problems let the coupled Vanka methods appear more attractive again. For these methods, no Schur complement preconditioners are required such that the described problems simply do not arise. Hence, despite the deficiencies of the Vanka solvers, they might be competitive again in certain situations. However, it it has to be examined how efficient they actually are for nonlinear and transient problems. First experiments in the context of finite deformation elasticity presented in a previous publication [175] are promising, but more detailed studies and comparisons to segregated solution methods are necessary.

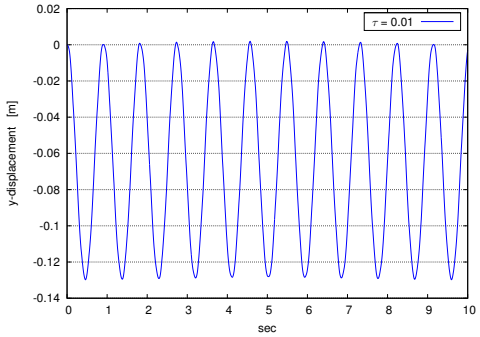
When performing such numerical tests, it is generally difficult to decide if certain (unexpected) effects are actually due to deficiencies of the solver/preconditioner, or if they are due to the Q_1/Q_1 stabilisation. In our opinion, it is advisable to first perform such tests with a stable

finite element pair in order to exclude these uncertainties. Then, when one knows what can be expected, one should perform the same tests with the stabilised Q_1/Q_1 pair. In future work, the solver tests we performed in this chapter also have to be repeated for a stable formulation.

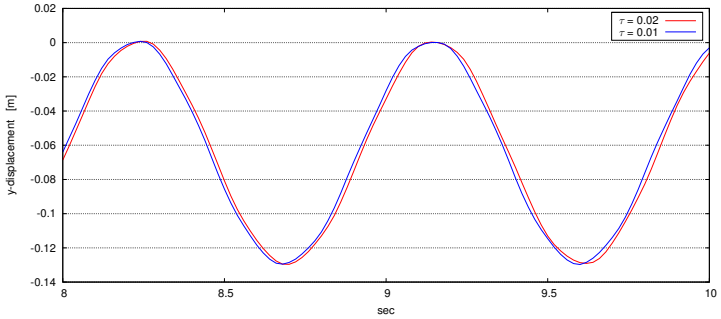
To validate that FEASTsolid is in principle able to successfully treat transient, non-linear saddle point problems, we present results for the benchmark test 'CSM3' from Turek and Hron [159]. It is the transient version of the static 'CSM1' benchmark described in Section 4.3.2 (see Figure 4.22 on page 235 and Table 4.2c). The geometry of the beam, the material ($\nu = 0.4, \mu = 0.5$ MPa, $\rho = 1000\ \frac{\text{kg}}{\text{m}^3}$, St. Venant-Kirchhoff constitutive law with finite deformation) and the loading situation (vertical gravity force of $g = -2\ \frac{\text{m}}{\text{s}^2}$) are the same. We apply the Newmark time-discretisation (see Section 3.4.1). In each time step, the resulting nonlinear equation system is solved by a Newton-Raphson method (without line damping) that reduces the initial residual norm by the factor 10^{-6}. The linear systems within the Newton iteration are solved by the block-preconditioned saddle point solver `BiCG-MG-SCBTria` gaining two digits. All computations are performed on grid refinement level 7 with 793 k DOF. The time interval is $I = [0.0, 10.0]$; all times are given in seconds. In Figure 5.17 we display the vertical displacement of the reference point $\boldsymbol{A} = (0.35\text{ m}, 0.0\text{ m})$ for two different time steps. Figure 5.17c



(a) $\tau = 0.02, t \in [0.0, 10.0]$.



(b) $\tau = 0.01, t \in [0.0, 10.0]$.



(c) $\tau \in \{0.01, 0.02\}, t \in [8.0, 10.0]$.

Figure 5.17: Displacements of the point $\boldsymbol{A} = (0.35\text{ m}, 0.0\text{ m})$ in the course of the transient simulation.

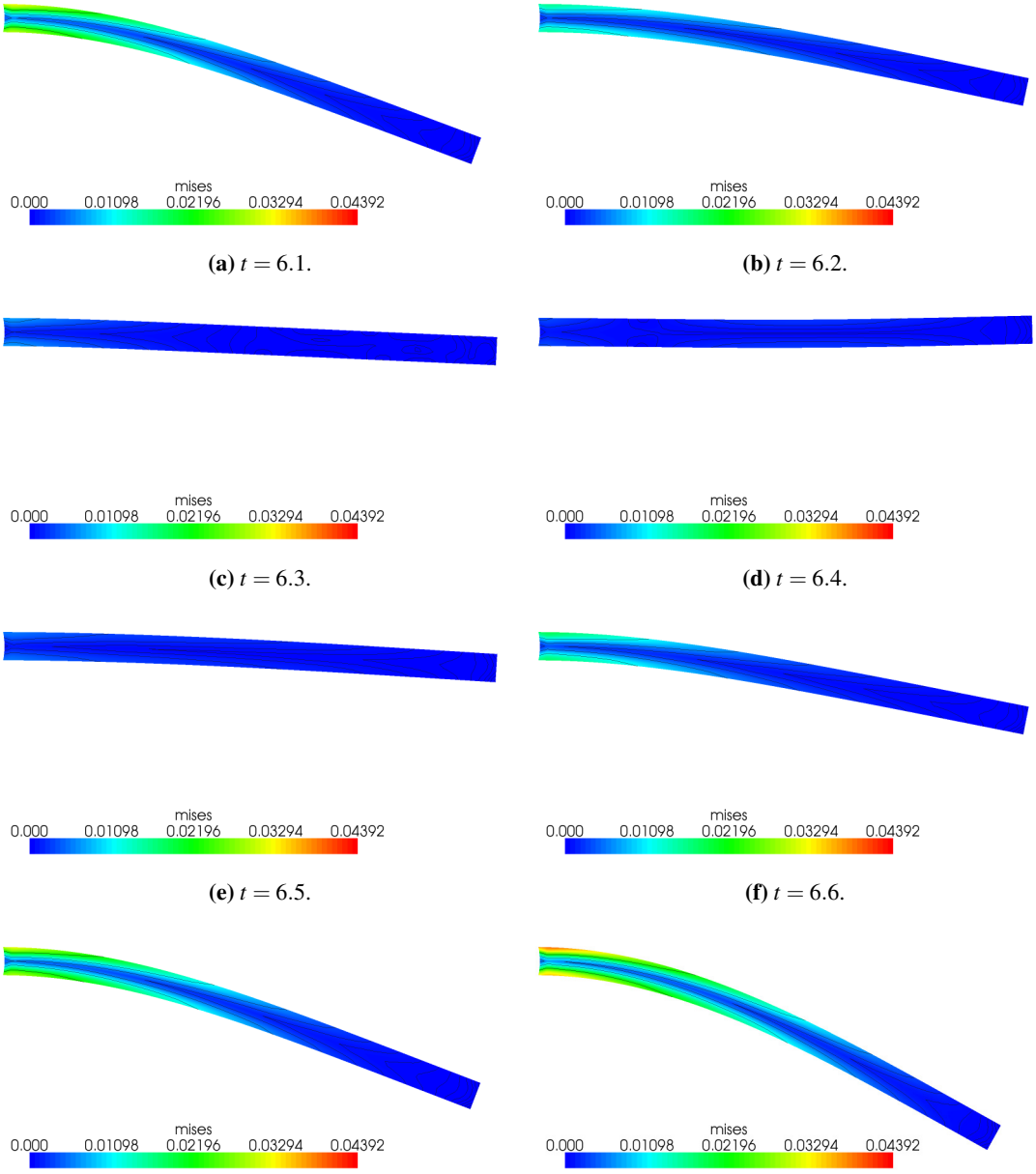
shows the subinterval $[8.0, 10.0]$ in more detail. The curves are qualitatively the same as those in Turek and Hron [159]. We obtain a mean value and an amplitude of

$$\begin{aligned} \tau = 0.02: &\qquad -0.063821 \pm 0.065788 \\ \tau = 0.01: &\qquad -0.063956 \pm 0.065809. \end{aligned}$$

These values are in the range of the values calculated with the stable Q_2/P_1 element and 98.8 k DOF [159]:

$$\begin{aligned} \tau = 0.02: &\qquad -0.064371 \pm 0.064695, \\ \tau = 0.01: &\qquad -0.064766 \pm 0.064948, \\ \tau = 0.005: &\qquad -0.063607 \pm 0.065160. \end{aligned}$$

The deformation states of the beam are exemplarily depicted for the time interval $[6.1, 6.8]$ in Figure 5.18.



(a) $t = 6.1$. **(b)** $t = 6.2$.

(c) $t = 6.3$. **(d)** $t = 6.4$.

(e) $t = 6.5$. **(f)** $t = 6.6$.

(g) $t = 6.7$. **(h)** $t = 6.8$.

Figure 5.18: Deformation of the BEAM configuration in the time interval $[6.1, 6.8]$; von Mises stresses displayed.

Bibliography

[1] Carlos Agelet de Saracibar, Michele Chiumenti, Quino Valverde, and Miguel Cervera. On the orthogonal subgrid scale pressure stabilization of finite deformation J2 plasticity. *Computer Methods in Applied Mechanics and Engineering*, 195(9–12):1224–1251, 2006. DOI: 10.1016/j.cma.2005.04.007.

[2] Mike Altieri. Robuste und effiziente Mehrgitter-Verfahren auf verallgemeinerten Tensorprodukt-Gittern. Diploma thesis, Universität Dortmund, Fachbereich Mathematik, 2001.

[3] Kenneth J. Arrow, Leonid Hurwicz, and Hirofumi Uzawa, editors. *Studies in Linear and Nonlinear Programming*. Stanford University Press, Stanford, 1958.

[4] Ferdinando Auricchio, Lourenco Beirao da Veiga, Carlo Lovadina, and Alessandro Reali. An analysis of some mixed-enhanced finite element for plane linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 194(27–29):2947–2968, 2005. DOI: 10.1016/j.cma.2004.07.028.

[5] Ferdinando Auricchio, Lourenco Beirao da Veiga, Carlo Lovadina, and Alessandro Reali. A stability study of some mixed finite elements for large deformation elasticity problems. *Computer Methods in Applied Mechanics and Engineering*, 194(9–11):1075–1092, 2005. DOI: 10.1016/j.cma.2004.06.014.

[6] Owe Axelsson. Preconditioning of indefinite problems by regularization. *SIAM Journal on Numerical Analysis*, 16(1):58–69, 1979. DOI: 10.1137/0716005.

[7] Owe Axelsson. On iterative solvers in structural mechanics; separate displacement orderings and mixed variable methods. *Mathematics and Computers in Simulations*, 50 (1–4):11–30, 1999. DOI: 10.1016/S0378-4754(99)00058-0.

[8] Owe Axelsson and Ivar Gustafsson. Iterative methods for the solution of the Navier equations of elasticity. *Computer Methods in Applied Mechanics and Engineering*, 15 (2):241–258, 1978. DOI: 10.1016/0045-7825(78)90026-9.

[9] Ivo Babuška and Manil Suri. Locking effects in the finite element approximation of elasticity problems. *Numerische Mathematik*, 62(1):439–463, 1992. DOI: 10.1007/BF01396238.

[10] Randolph E. Bank, Tony F. Chan, William M. Coughran Jr., and R. Kent Smith. The alternate-block-factorization procedure for systems of partial differential eequations. *BIT*, 29(4):938–954, 1989. DOI: 10.1007/BF01932753.

[11] Randolph E. Bank, Bruno D. Welfert, and Harry Yserentant. A class of iterative methods for solving saddle point problems. *Numerische Mathematik*, 56(7):645–666, 1990. DOI: 10.1007/BF01405194.

[12] Gabriel R. Barrenechea and Jordi Blasco. Pressure stabilization of finite element approximations of time-dependent incompressible flow problems. *Computer Methods in Applied Mechanics and Engineering*, 197(1–4):219–231, 2007. DOI: 10.1016/j.cma.2007.07.027.

[13] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk A. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994. `http://www.netlib.org/templates/templates.html`.

[14] Teri Barth, Pavel B. Bochev, Max D. Gunzburger, and John Shadid. A taxonomy of consistently stabilized finite element methods for the Stokes problem. *SIAM Journal on Scientific Computing*, 25(5):1585–1607, 2004. DOI: 10.1137/S1064827502407718.

[15] Klaus-Jürgen Bathe. *Finite-Elemente-Methoden*. Springer, Berlin, 2nd edition, 2002.

[16] Christian Becker. *Strategien und Methoden zur Ausnutzung der High-Performance-Computing-Ressourcen moderner Rechnerarchitekturen für Finite Element Simulationen und ihre Realisierung in FEAST (Finite Element Analysis & Solution Tools)*. PhD thesis, Universität Dortmund, Fachbereich Mathematik, 2007. `http://www.logos-verlag.de/cgi-bin/buch?isbn=1637`.

[17] Roland Becker. *An Adaptive Finite Element Method for the Incompressible Navier-Stokes Equations on Time-Dependent Domains*. PhD thesis, Universität Heidelberg, Fakultät für Mathematik, 1995.

[18] Roland Becker and Malte Braack. A finite element pressure gradient stabilization for the Stokes equations based on local projections. *Calcolo*, 38(4):173–199, 2001. DOI: 10.1007/s10092-001-8180-4.

[19] Roland Becker and Rolf Rannacher. Finite element solution of the incompressible Navier-Stokes equations on anisotropically refined meshes. In *Fast Solvers for Flow Problems (Proceedings 10th GAMM Seminar Kiel 1994)*, volume 49 of *Notes on Numerical Fluid Mechanics*, pages 52–62. Vieweg, Braunschweig, 1995.

[20] Michele Benzi and Maxim A. Olshanskii. An augmented Lagrangian-based approach to the Oseen-problem. *SIAM Journal on Scientific Computing*, 28(6):2095–2113, 2006. DOI: 10.1137/050646421.

[21] Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005. DOI: 10.1017/S0962492904000212.

[22] Radim Blaheta. Displacement decomposition – Incomplete factorization preconditioning techniques for linear elasticity. *Numerical Linear Algebra with Applications*, 1(2):107–128, 1994. DOI: 10.1002/nla.1680010203.

[23] Jordi Blasco. An anisotropic GLS–stabilized finite element method for incompressible flow problems. *Computer Methods in Applied Mechanics and Engineering*, 197(45–48): 3712–3723, 2008. DOI: 10.1016/j.cma.2008.02.031.

[24] Jordi Blasco. Personal communication, 2008.

[25] Pavel B. Bochev and Max D. Gunzburger. An absolutely stable pressure-Poisson stabilized finite element method for the Stokes equations. *SIAM Journal on Numerical Analysis*, 47(3):1189–1207, 2004. DOI: 10.1137/S0036142903416547.

[26] Pavel B. Bochev and Richard B. Lehoucq. On the finite element solution of the pure Neumann problem. *SIAM Review*, 47(1):50–66, 2005. DOI: 10.1137/S0036144503426074.

[27] Pavel B. Bochev, Clark R. Dohrmann, and Max D. Gunzburger. Stabilization of low-order mixed finite elements for the Stokes equations. *SIAM Journal on Numerical Analysis*, 44(1):82–101, 2006. DOI: 10.1137/S0036142905444482.

[28] Pavel B. Bochev, Max D. Gunzburger, and Richard B. Lehoucq. On stabilized finite element methods for the Stokes problem in the small time step limit. *International Journal for Numerical Methods in Fluids*, 53(4):573–597, 2007. DOI: 10.1002/fld.1295.

[29] Malte Braack, Erik Burman, Volker John, and Gert Lube. Stabilized finite element methods for the generalized Oseen problem. *Computer Methods in Applied Mechanics and Engineering*, 196(4–6):853–866, 2007. DOI: 10.1016/j.cma.2006.07.011.

[30] Dietrich Braess. A multigrid method for the membrane problem. *Computational Mechanics*, 3(5):321–329, 1988. DOI: 10.1007/BF00712146.

[31] Dietrich Braess. Enhanced assumed strain elements and nearly incompressible material. In *European Conference on Computational Mechanics, ECCM-2001, June 26-29, Cracow, Poland*, 2001.

[32] Dietrich Braess. *Finite Elemente*. Springer, Berlin, 3rd edition, 2003.

[33] Dietrich Braess and Carola Blömer. A multigrid method for a parameter dependent problem in solid mechanics. *Numerische Mathematik*, 57(1):747–761, 1990. DOI: 10.1007/BF01386441.

[34] Dietrich Braess and Pingbing Ming. A finite element method for nearly incompressible elasticity problems. *Mathematics of Computation*, 74(249):25–52, 2005. DOI: 10.1090/S0025-5718-04-01662-X.

[35] Dietrich Braess and Regina Sarazin. An efficient smoother for the Stokes problem. *Applied Numerical Mathematics*, 23(1):3–20, 1997. DOI: 10.1016/S0168-9274(96)00059-1.

[36] James H. Bramble and Joseph E. Pasciak. Iterative techniques for time dependent Stokes problems. *Computers and Mathematics with Applications*, 33(1–2):13–30, 1997. DOI: 10.1016/S0898-1221(96)00216-7.

[37] James H. Bramble, Joseph E. Pasciak, and Apostol T. Vassilev. Analysis of the inexact Uzawa algorithm for saddle point problems. *SIAM Journal on Numerical Analysis*, 34 (3):1072–1092, 1997. DOI: 10.1137/S0036142994273343.

[38] Franco Brezzi and Jim Douglas. Stabilized mixed methods for the Stokes problem. *Numerische Mathematik*, 53(1–2):225–235, 1988. DOI: 10.1007/BF01395886.

[39] Franco Brezzi and Michel Fortin. *Mixed and Hybrid Finite Element Methods*, volume 15 of *Springer Series in Computational Mathematics*. Springer, New York, 1991.

[40] Franco Brezzi and Juhani Pitkäranta. On the stabilisation of finite element approximations of the Stokes equations. In W. Hackbusch, editor, *Efficient Solutions of Elliptic Systems (Proceedings GAMM Seminar Kiel)*, volume 10 of *Notes on Numerical Fluid Mechanics*, pages 11–19. Vieweg, Braunschweig, 1984.

[41] Ulrich Brink and Erwin Stein. On some mixed finite element methods for incompressible and nearly incompressible finite elasticity. *Computational Mechanics*, 19(1):105–119, 1996. DOI: 10.1007/BF02824849.

[42] Peter N. Brown and Yousef Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing*, 11(3):450–481, 1990. DOI: 10.1137/0911026.

[43] Sven Buijssen, Hilmar Wobker, Dominik Göddeke, and Stefan Turek. FEASTSolid and feastflow: FEM applications exploiting feast's HPC technologies. In Wolfgang Nagel, , Dietmar Kröner, and Michael Resch, editors, *High Performance Computing in Science and Engineering '08*, Transactions of the High Performance Computing Center Stuttgart (HLRS) 2008, pages 425–440. Springer, Berlin, 2008. DOI: 10.1007/978-3-540-88303-6.

[44] Sven H.M. Buijssen. *Efficient Multilevel Solvers and High Performance Computing Techniques for the Finite Element Simulation of the Transient, Incompressible Navier–Stokes Equations*. PhD thesis, TU Dortmund, Fakultät für Mathematik, 2010. to be published.

[45] J. Cahouet and Jean-Paul Chabard. Some fast 3-d finite element solvers for the generalized Stokes problem. *International Journal for Numerical Methods in Fluids*, 8(8): 869–895, 1988. DOI: 10.1002/fld.1650080802.

[46] Xiao-Chuan Cai and David E. Keyes. Nonlinearly preconditioned inexact Newton algorithms. *SIAM Journal on Scientific Computing*, 24(1):183–200, 2002. DOI: 10.1137/S106482750037620X.

[47] Zhi-Hao Cao. Fast Uzawa algorithm for generalized saddle point problems. *Applied Numerical Mathematics*, 47(2):157–171, 2003. DOI: 10.1016/S0168-9274(03)00023-0.

[48] Zhi-Hao Cao, David J. Evans, and Mei Qin. Convergence of iterative methods for stabilized saddle-point problems. *International Journal of Computer Mathematics*, 82(8): 987–999, 2005. DOI: 10.1080/00207160412331336125.

[49] Miguel Cervera, Michele Chiumenti, Quino Valverde, and Carlos Agelet de Saracibar. Mixed linear/linear simplicial elements for incompressible elasticity and plasticity. *Computer Methods in Applied Mechanics and Engineering*, 192(49–50):5249–5263, 2003. DOI: 10.1016/j.cma.2003.07.007.

[50] Xiao-Liang Cheng and Jun Zou. An inexact Uzawa-type iterative method for solving saddle point problems. *International Journal of Computer Mathematics*, 80(1):55–64, 2003. DOI: 10.1080/00207160304658.

[51] Michele Chiumenti, Quino Valverde, Carlos Agelet de Saracibar, and Miguel Cervera. A stabilized formulation for incompressible elasticity using linear displacement and pressure interpolations. *Computer Methods in Applied Mechanics and Engineering*, 191(46): 5253–5264, 2002. DOI: 10.1016/S0045-7825(02)00443-7.

[52] Philippe G. Ciarlet. *Mathematical Elasticity. Volume I: Three-Dimensional Elasticity*, volume 20 of *Studies in Mathematics and its Applications*. North-Holland, Amsterdam, 1988.

[53] Phillip Colella, Thom H. Dunning, William D. Gropp, and David E. Keyes. A science–based case for large–scale simulation. Technical report, DOE Office of Science, 2003. `http://www.pnl.gov/scales`.

[54] Stéphane Commend, Andrzej Truty, and Thomas Zimmermann. Stabilized finite elements applied to elastoplasticity: I. Mixed displacement-pressure formulation. *Computer Methods in Applied Mechanics and Engineering*, 193(33–35):3559–3586, 2004. DOI: 10.1016/j.cma.2004.01.007.

[55] Timothy A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):165–195, 2004. DOI: 10.1145/992200.992205.

[56] Arie C. de Niet and Fred W. Wubs. Two preconditioners for saddle point problems in fluid flows. *International Journal for Numerical Methods in Fluids*, 54(4):355–377, 2007. DOI: 10.1002/fld.1401.

[57] Ron S. Dembo and Trond Steihaug. Truncated Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26(2):190–212, 1983. DOI: 10.1007/BF02592055.

[58] Clark R. Dohrmann and Richard B. Lehoucq. A primal-based penalty preconditioner for elliptic saddle point problems. *SIAM Journal on Numerical Analysis*, 44(1):270–282, 2006. DOI: 10.1137/040619016.

[59] Jean-Jaques Droux and Thomas J. R. Hughes. A boundary integral modification of the Galerkin least squares formulation for the Stokes problem. *Computer Methods in Applied Mechanics and Engineering*, 113(1–2):173–182, 1994. DOI: 10.1016/0045-7825(94)90217-8.

[60] Stanley C. Eisenstat and Homer F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, 1996. DOI: 10.1137/0917003.

[61] Howard C. Elman. Multigrid and Krylov subspace methods for the discrete Stokes equations. *International Journal for Numerical Methods in Fluids*, 22(8):755–770, 1996. DOI: 10.1002/(SICI)1097-0363(19960430)22:8<755::AID-FLD377>3.0.CO;2-1.

[62] Howard C. Elman and Gene H. Golub. Inexact and preconditioned Uzawa algorithms for saddle point problems. *SIAM Journal on Numerical Analysis*, 31(6):1645–1661, 1994. DOI: 10.1137/0731085.

[63] Howard C. Elman and David Silvester. Fast nonsymmetric iterations and preconditioning for Navier-Stokes equations. *SIAM Journal on Scientific Computing*, 17(1):33–46, 1996. DOI: 10.1137/0917004.

[64] Howard C. Elman, Oliver G. Ernst, and Dianne P. O'Leary. A multigrid method enhanced by Krylov subspace iteration for discrete Helmholtz equations. *SIAM Journal on Scientific Computing*, 23(4):1291–1315, 2001. DOI: 10.1137/S1064827501357190.

[65] Howard C. Elman, David Silvester, and Andrew J. Wathen. *Finite Elements and Fast Iterative Solvers*. Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, 2005.

[66] Howard C. Elman, Victoria E. Howle, John Shadid, Robert Shuttleworth, and Ray Tuminaro. Block preconditioners based on approximate commutators. *SIAM Journal on Scientific Computing*, 27(5):1651–1668, 2006. DOI: 10.1137/040608817.

[67] Howard C. Elman, Victoria E. Howle, John Shadid, David Silvester, and Ray Tuminaro. Least squares preconditioners for stabilized discretizations of the Navier-Stokes equations. *SIAM Journal on Scientific Computing*, 30(1):290–311, 2007. DOI: 10.1137/060655742.

[68] Leopoldo P. Franca and Rolf Stenberg. Error analysis of some Galerkin least squares methods for the elasticity equations. *SIAM Journal on Numerical Analysis*, 28(6):1680–1697, 1991. DOI: 10.1137/0728084.

[69] Francisco J. Gaspar, Francisco J. Lisbona, Cornelis W. Oosterlee, and Roman Wienands. A systematic comparison of coupled and distributive smoothing in multigrid for the poroelasticity system. *Numerical Linear Algebra with Applications*, 11(2–3):93–113, 2004. DOI: 10.1002/nla.372.

[70] Dominik Göddeke. *Fast and Accurate Finite-Element Multigrid Solvers for PDE Simulations on GPU Clusters*. PhD thesis, TU Dortmund, Fakultät für Mathematik, 2010. to be published.

[71] Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, Sven Buijssen, Matthias Grajewski, and Stefan Turek. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. *Parallel Computing*, 33(10–11):685–699, 2007. DOI: j.parco.2007.09.002.

[72] Dominik Göddeke, Hilmar Wobker, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, and Stefan Turek. Co-processor acceleration of an unmodified parallel structural mechanics code with FEAST-GPU, 2007. Supercomputing 2007 Posters.

[73] Dominik Göddeke, Robert Strzodka, Jamal Mohd-Yusof, Patrick McCormick, Hilmar Wobker, Christian Becker, and Stefan Turek. Using GPUs to improve multigrid solver performance on a cluster. *International Journal of Computational Science and Engineering*, 4(1):36–55, 2008. DOI: 10.1504/IJCSE.2008.021111.

[74] Dominik Göddeke, Sven H. M. Buijssen, Hilmar Wobker, and Stefan Turek. GPU acceleration of an unmodified parallel finite element Navier-Stokes solver. In Waleed W. Smari and John P. McIntire, editors, *High Performance Computing & Simulation 2009*, pages 12–21, 2009.

[75] Dominik Göddeke, Hilmar Wobker, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, and Stefan Turek. Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU. *International Journal of Computational Science and Engineering*, 4(4):254–269, 2009. DOI: 10.1504/IJCSE.2009.029162.

[76] Oscar Gonzalez. Exact energy and momentum conserving algorithms for general models in nonlinear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 190 (13–14):1763–1783, 2000. DOI: 10.1016/S0045-7825(00)00189-4.

[77] Susan L. Graham, Marc Snir, and Cynthia A. Patterson (eds.). *Getting up to Speed – The Future of Supercomputing*. The National Academies Press, Washington, D. C., 2004.

[78] Matthias Grajewski. *A new fast and accurate grid deformation method for r-adaptivity in the context of high performance computing*. PhD thesis, TU Dortmund, Fakultät für Mathematik, 2008. `http://www.logos-verlag.de/cgi-bin/buch?isbn=1903`.

[79] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[80] Jaroslav Hron and Stefan Turek. A monolithic FEM solver for an ALE formulation of fluid structure interaction with configurations for numerical benchmarking. In M. Papadrakakis, E. Onate, and B. Schrefler, editors, *Computational Methods for Coupled Problems in Science and Engineering*, Proceedings 'First International Conference on Computational Methods for Coupled Problems in Science and Engineering' (Santorini, May 25th–27th), 2005.

[81] Thomas J. R. Hughes and Leopoldo P. Franca. A new finite element formulation for computational fluid dynamics: VII. The Stokes problem with various well-posed boundary conditions: Symmetric formulations that converge for all velocity/pressure spaces. *Computer Methods in Applied Mechanics and Engineering*, 65(1):85–96, 1987. DOI: 10.1016/0045-7825(87)90184-8.

[82] Thomas J. R. Hughes, Leopoldo P. Franca, and Marc Balestra. A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška-Brezzi condition. A stable Petrov-Galerkin formulation of the Stokes problem accomodating equal-order interpolations. *Computer Methods in Applied Mechanics and Engineering*, 59(1):85–99, 1986. DOI: 10.1016/0045-7825(86)90025-3.

[83] Thomas J. R. Hughes, Leopoldo P. Franca, and Gregory M. Hulbert. A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective-diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73(2):173–189, 1989. DOI: 10.1016/0045-7825(89)90111-4.

[84] Feng-Nan Hwang. *Some Parallel Linear and Nonlinear Schwarz Methods with applications in CFD*. PhD thesis, University of Colorado, Department of Applied Mathematics, 2004.

[85] Kenneth E. Jansen, S. Scott Collis, Christian Whiting, and Farzin Shakib. A better consistency for low-order stabilized finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 174(1–2):153–170, 1999. DOI: 10.1016/S0045-7825(98)00284-9.

[86] Volker John. A comparison of parallel solvers for the incompressible Navier-Stokes equations. *Computing and Visualization in Science*, 1(4):193–200, 1999. DOI: 10.1007/s007910050018.

[87] Volker John. Higher order finite element methods and multigrid solvers in a benchmark problem for the 3D Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 40(6):775–798, 2002. DOI: 10.1002/fld.377.

[88] Volker John and Gunar Matthies. Higher order finite element discretizations in a benchmark problem for incompressible flows. *International Journal for Numerical Methods in Fluids*, 37(8):885–903, 2001. DOI: 10.1002/fld.195.

[89] Volker John and Lutz Tobiska. Numerical performance of smoothers in coupled multigrid methods for the parallel solution of the incompressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 33(4):453–473, 2000. DOI: 10.1002/1097-0363(20000630)33:4<453::AID-FLD15>3.0.CO;2-0.

[90] George Karpys and Vipin Kumar. METIS – A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`, 1998.

[91] David Kay, Daniel Loghin, and Andrew Wathen. A preconditioner for the steady-state Navier-Stokes equations. *SIAM Journal on Scientific Computing*, 24(1):237–256, 2002. DOI: 10.1137/S106482759935808X.

[92] Carl T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.

[93] David E. Keyes. Terascale implicit methods for partial differential equations. In Xiaobing Feng and Tim P. Schulze, editors, *Recent Advances in Numerical Methods for Partial Differential Equations and Applications*, volume 306 of *Contemporary Mathematics*, pages 29–84. AMS, 2002.

[94] Susanne Kilian. *ScaRC: Ein verallgemeinertes Gebietszerlegungs-Mehrgitterkonzept auf Parallelrechnern*. PhD thesis, Universität Dortmund, Fachbereich Mathematik, Logos Verlag, Berlin, 2001. `http://www.logos-verlag.de/cgi-bin/buch?isbn=0092`.

[95] Ottmar Klaas, Antoinette Maniatty, and Mark S. Shephard. A stabilized mixed finite element method for finite elasticity. Formulation for linear displacement and pressure interpolation. *Computer Methods in Applied Mechanics and Engineering*, 180(1–2): 65–79, 1999. DOI: 10.1016/S0045-7825(99)00059-6.

[96] Axel Klawonn. An optimal preconditioner for a class of saddle point problems with a penalty term. *SIAM Journal on Scientific Computing*, 19(2):540–552, 1998. DOI: 10.1137/S1064827595279575.

[97] Axel Klawonn. Block-triangular preconditioners for saddle point problems with a penalty term. *SIAM Journal on Scientific Computing*, 19(1):172–184, 1998. DOI: 10.1137/S1064827596303624.

[98] Georgij M. Kobelkov and Maxim A. Olshanskii. Effective preconditioning of Uzawa type schemes for a generalized Stokes problem. *Numerische Mathematik*, 86(3):443–470, 2000. DOI: 10.1007/s002110000160.

[99] Frank Koschnick. *Geometrische Locking-Effekte bei Finiten Elementen und ein allgemeines Konzept zu ihrer Vermeidung*. PhD thesis, TU München, Fakultät für Bauingenieur- und Vermessungswesen, 2004.

[100] Michael Köster. Robuste Mehrgitter-Krylowraum-Techniken für FEM-Verfahren. Diploma thesis, Universität Dortmund, Fachbereich Mathematik, 2004.

[101] Michael Köster and Stefan Turek. The influence of higher order FEM discretisations on multigrid convergence. *Computational Methods in Applied Mathematics*, 6(2):221–232, 2006.

[102] Michael Köster, Dominik Göddeke, Hilmar Wobker, and Stefan Turek. How to gain speedups of 1000 on single processors with fast FEM solvers – Benchmarking numerical and computational efficiency. Ergebnisberichte des Instituts für Angewandte Mathematik, Nr. 382, TU Dortmund, Fakultät für Mathematik, 2008.

[103] Maxim Larin and Arnold Reusken. A comparative study of efficient iterative solvers for generalized Stokes equations. *Numerical Linear Algebra with Applications*, 15(1): 13–34, 2008. DOI: 10.1002/nla.561.

[104] Sabine Le Borne. Hierarchical matrix preconditioners for the Oseen equations. *Computing and Visualization in Science*, 11(3):147–157, 2008. DOI: 10.1007/s00791-007-0065-x.

[105] Patrick Le Tallec. Numerical methods for nonlinear three-dimensional elasticity. In P. G. Ciarlet and J. L. Lions, editors, *Handbook of Numerical Analysis*, volume III, pages 465–662. North-Holland, 1994.

[106] Daniel Loghin and Andrew J. Wathen. Analysis of preconditioners for saddle-point problems. *SIAM Journal on Scientific Computing*, 25(6):2029–2049, 2004. DOI: 10.1137/S1064827502418203.

[107] David S. Malkus and Thomas J. R. Hughes. Mixed finite element methods – Reduced and selective integration techniques: A unification of concepts. *Computer Methods in Applied Mechanics and Engineering*, 15(1):63–81, 1978. DOI: 10.1016/0045-7825(78)90005-1.

[108] Antoinette Maniatty, Yong Liu, Ottmar Klaas, and Mark S. Shephard. Higher order stabilized finite element method for hyperelastic finite deformation. *Computer Methods in Applied Mechanics and Engineering*, 191(13–14):1491–1503, 2002. DOI: 10.1016/S0045-7825(01)00335-8.

[109] Sandro Manservisi. Numerical analysis of Vanka-type solvers for steady Stokes and Navier-Stokes flows. *SIAM Journal on Numerical Analysis*, 44(5):2025–2056, 2006. DOI: 10.1137/060655407.

[110] Kent-Andre Mardal and Ragnar Winther. Uniform preconditioners for the time dependent Stokes problem. *Numerische Mathematik*, 98(2):305–327, 2004. DOI: 10.1007/s00211-004-0529-6.

[111] Stefano Micheletti, Simona Perotto, and Marco Picasso. Stabilized finite elements on anisotropic meshes: A priori error estimates for the advection-diffusion and the Stokes problems. *SIAM Journal on Numerical Analysis*, 41(3):1131–1162, 2003. DOI: 10.1137/S0036142902403759.

[112] Christian Miehe. Numerical computation of algorithmic (consistent) tangent moduli in large-strain computational inelasticity. *Computer Methods in Applied Mechanics and Engineering*, 134(3–4):223–240, 1996. DOI: 10.1016/0045-7825(96)01019-5.

[113] Milan D. Mihajlović and Slobodan Mijalković. A component decomposition preconditioning for 3D stress analysis problems. *Numerical Linear Algebra with Applications*, 9 (6–7):567–583, 2002. DOI: 10.1002/nla.298.

[114] Johannes Molenaar. A two-grid analysis of the combination of mixed finite elements and Vanka-type relaxation. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods III*, Proc. 3rd Eur. Conf., Bonn/Germany, ISNM 98, pages 313–323. Birkhäuser Verlag, 1991.

[115] Malcolm F. Murphy, Gene H. Golub, and Andrew J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM Journal on Scientific Computing*, 21(6):1969–1972, 1999. DOI: 10.1137/S1064827599355153.

[116] Nathan M. Newmark. A method of computation for structural dynamics. *Journal of Engineering Mechanics Division, ASCE*, 85(3):67–94, 1959.

[117] Sean Norburn and David Silvester. Stabilised vs. stable mixed methods for incompressible flow. *Computer Methods in Applied Mechanics and Engineering*, 166(1–2):131–141, 1998. DOI: 10.1016/S0045-7825(98)00087-5.

[118] Yvan Notay. Flexible conjugate gradients. *SIAM Journal on Scientific Computing*, 22 (4):1444–1460, 2000. DOI: 10.1137/S1064827599362314.

[119] Maxim A. Olshanskii and Yuri V. Vassilevski. Pressure Schur complement preconditioners for the discrete Oseen problem. *SIAM Journal on Scientific Computing*, 29(6): 2686–2704, 2007. DOI: 10.1137/070679776.

[120] Abderrahim Ouazzi and Stefan Turek. Efficient multigrid and data structures for edge-oriented FEM stabilization. In *Numerical Mathematics and Advanced Applications Enumath 2005*, pages 520–527. Springer, Berlin, 2006. ISBN-10 3-540-34287-7.

[121] Evgueni E. Ovtchinnikov and Leonidas S. Xanthis. Iterative subspace correction methods for thin elastic structures and Korn's type inequality in subspaces. *Proceedings: Mathematical, Physical and Engineering Sciences*, 454(1976):2023–2039, 1998.

[122] Daniel Pantuso and Klaus-Jürgen Bathe. A four-node quadrilateral mixed-interpolated element for solids and fluids. *Mathematical Models and Methods in Applied Sciences*, 5 (8):1113–1128, 1995. DOI: 10.1142/S0218202595000589.

[123] Daniel Pantuso and Klaus-Jürgen Bathe. On the stability of mixed finite elements in large strain analysis of incompressible solids. *Finite Elements in Analysis and Design*, 28(2):83–104, 1997. DOI: 10.1016/S0168-874X(97)81953-1.

[124] Katerina-D. Papoulia. Mixed and selective reduced integration procedures in large strain hyperelastic analysis of nearly incompressible solids. *Computational Mechanics*, 23(1): 63–74, 1999. DOI: 10.1007/PL00009637.

[125] Manuel Pastor, Manuel Quecedob, and Olgierd C. Zienkiewicz. A mixed displacement-pressure formulation for numerical analysis of plastic failure. *Computers & Structures*, 62(1):13–23, 1997. DOI: 10.1016/S0045-7949(96)00208-8.

[126] Suhas V. Patankar. *Numerical Heat Transfer and Fluid Flow*. McGraw-Hill, New York, 1980.

[127] Suhas V. Patankar and D. Brian Spalding. A calculation procedure for heat and mass transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15(10):1787–1806, 1972. DOI: 10.1016/0017-9310(72)90054-3.

[128] Christoph Pflaum. A multigrid conjugate gradient method. *Applied Numerical Mathematics*, 58(12):1803–1817, 2008. DOI: 10.1016/j.apnum.2007.11.020.

[129] Roger Pierre. Simple C^0 approximations for the computation of incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 68(2):205–227, 1988. DOI: 10.1016/0045-7825(88)90116-8.

[130] Werner Queck. The convergence factor of preconditioned algorithms of the Arrow-Hurwicz type. *SIAM Journal on Numerical Analysis*, 26(4):1016–1030, 1989. DOI: 10.1137/0726057.

[131] Binoj Ramesh and Antoinette Maniatty. Stabilized finite element formulation for elastic-plastic finite deformations. *Computer Methods in Applied Mechanics and Engineering*, 194(6–8):775–800, 2005. DOI: 10.1016/j.cma.2004.06.025.

[132] Stefanie Reese and Peter Wriggers. A stabilization technique to avoid hourglassing in finite elasticity. *International Journal for Numerical Methods in Engineering*, 48(1):79–109, 2000. DOI: 10.1002/(SICI)1097-0207(20000510)48:1<79::AID-NME869>3.0.CO;2-D.

[133] Stefanie Reese, Martin Küssner, and Batmanathan Dayanand Reddy. A new stabilization technique for finite elements in finite elasticity. *International Journal for Numerical Methods in Engineering*, 44(11):1617–1652, 1999. DOI: 10.1002/(SICI)1097-0207(19990420)44:11<1617::AID-NME557>3.0.CO;2-X.

[134] Arnold Reusken. Steplength optimization and linear multigrid methods. *Numerische Mathematik*, 58(1):819–838, 1991. DOI: 10.1007/BF01385656.

[135] Ulrich Rüde. Technological trends and their impact on the future of supercomputers. In Hans-Joachim Bungartz, Franz Durst, and Christoph Zenger, editors, *High Performance Scientific and Engineering Computing*, volume 8 of *Lecture notes in Computational Science and Engineering*, pages 459–471, 1999.

[136] Yousef Saad. A flexible inner-outer preconditioned GMRES. *SIAM Journal on Scientific and Statistical Computing*, 14(2):461–469, 1993. DOI: 10.1137/0914028.

[137] Yousef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. DOI: 10.1137/0907058.

[138] Friedhelm Schieweck. *Parallele Lösung der stationären, inkompressiblen Navier-Stokes Gleichungen*. Habilitation, Otto-von-Guericke Universität Magdeburg, Fakultät Mathematik, 1997.

[139] Rainer Schmachtel. *Robuste lineare und nichtlineare Lösungsverfahren für die inkompressiblen Navier-Stokes-Gleichungen*. PhD thesis, Universität Dortmund, Fachbereich Mathematik, 2003.

[140] Joachim Schöberl. Multigrid methods for a parameter dependent problem in primal variables. *Numerische Mathematik*, 84(1):97–119, 1999. DOI: 10.1007/s002110050465.

[141] Joachim Schöberl and Walter Zulehner. On Schwarz-type smoothers for saddle point problems. *Numerische Mathematik*, 95(2):377–399, 2003. DOI: 10.1007/s00211-002-0448-3.

[142] Hermann A. Schwarz. *Gesammelte mathematische Abhandlungen*. Springer, Berlin, 1890.

[143] David Silvester and Andrew Wathen. Fast iterative solution of stabilised Stokes systems. Part II: Using general block preconditioners. *SIAM Journal on Numerical Analysis*, 31 (5):1352–1367, 1994. DOI: 10.1137/0731070.

[144] David Silvester, Howard Elman, David Kay, and Andrew Wathen. Efficient preconditioning of the linearized Navier-Stokes equations for incompressible flow. *Journal of Computational and Applied Mathematics*, 128(1–2):261–279, 2001. DOI: 10.1016/S0377-0427(00)00515-X.

[145] Juan-Carlos Simo and Francisco Armero. Geometrically non-linear enhanced strain mixed methods and the method of incompatible modes. *International Journal for Numerical Methods in Engineering*, 33(7):1413–1449, 1992. DOI: 10.1002/nme.1620330705.

[146] Juan-Carlos Simo and M. S. Rifai. A class of mixed assumed strain methods and the method of incompatible modes. *International Journal for Numerical Methods in Engineering*, 29(8):1595–1638, 1990. DOI: 10.1002/nme.1620290802.

[147] Valeria Simoncini. Block triangular preconditioners for symmetric saddle-point problems. *Applied Numerical Mathematics*, 49(1):63–80, 2004. DOI: 10.1016/j.apnum.2003.11.012.

[148] Valeria Simoncini and Daniel B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM Journal on Scientific Computing*, 25(2): 454–477, 2003. DOI: 10.1137/S1064827502406415.

[149] Valeria Simoncini and Daniel B. Szyld. Recent computational developments in Krylov subspace methods for linear systems. *Numerical Linear Algebra with Applications*, 14 (1):1–59, 2007. DOI: 10.1002/nla.499.

[150] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition – Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[151] Erwin Stein and Franz-Joseph Barthold. Elastizitätstheorie. In *Der Ingenieurbau, Grundwissen: Werkstoffe, Elastizitätstheorie*, pages 165–428. Ernst & Sohn, Berlin, 1996.

[152] Theodore Sussmann and Klaus-Jürgen Bathe. A finite element formulation for nonlinear incompressible elastic and inelastic analysis. *Computers & Structures*, 26(1–2):357–409, 1987. DOI: 10.1016/0045-7949(87)90265-3.

[153] Franz-Theo Suttmeier. An adaptive displacement/pressure finite element scheme for treating incompressibility effects in elasto-plastic materials. *Numerical Methods for Partial Differential Equations*, 17(4):369–382, 2001. DOI: 10.1002/num.1017.

[154] Robert L. Taylor, Peter J. Beresford, and Edward L. Wilson. A non-conforming element for stress analysis. *International Journal for Numerical Methods in Engineering*, 10(6): 1211–1219, 1976. DOI: 10.1002/nme.1620100602.

[155] M. C. Thompson and Joel H. Ferziger. An adaptive multigrid technique for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 82(1):94–121, 1989. DOI: 10.1016/0021-9991(89)90037-5.

[156] Andrea Toselli and Olof B. Widlund. *Domain Decomposition Methods – Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer, Berlin, 2005.

[157] Raymond S. Tuminaro, Homer F. Walker, and John N. Shadid. On backtracking failure in Newton-GMRES methods with a demonstration for the Navier-Stokes equations. *Journal of Computational Physics*, 180(2):549–558, 2002. DOI: 10.1006/jcph.2002.7102.

[158] Stefan Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999.

[159] Stefan Turek and Jaroslav Hron. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. Ergebnisberichte des Instituts für Angewandte Mathematik, Nr. 312, TU Dortmund, Fakultät für Mathematik, 2006.

[160] Stefan Turek and Michael Schäfer. Benchmark computations of laminar flow around cylinder. In E. H. Hirschel, editor, *Flow Simulation with High-Performance Computers II*, volume 52 of *Notes on Numerical Fluid Mechanics*, pages 547–566. Vieweg, 1996.

[161] Stefan Turek, Alexander Runge, and Christian Becker. The FEAST indices – Realistic evaluation of modern software components and processor technologies. *Computers and Mathematics with Applications*, 41(10–11):1431–1464, 2001. DOI: 10.1016/S0898-1221(01)00108-0.

[162] Stefan Turek, Christian Becker, and Susanne Kilian. Hardware-oriented numerics and concepts for PDE software. *Future Generation Computer Systems*, 22(1–2):217–238, 2003. DOI: 10.1016/j.future.2003.09.007.

[163] Mehfooz ur Rehman, Cornelis Vuik, and Guus Segal. A comparison of preconditioners for incompressible Navier-Stokes solvers. *International Journal for Numerical Methods in Fluids*, 57(12):1731–1751, 2008. DOI: 10.1002/fld.1684.

[164] Andréa M. P. Valli, Graham F. Carey, and Alvaro L. G. A. Coutinho. Finite element simulation and control of nonlinear flow and reactive transport. In *Proc. 10th Int. Conf. Finite Element in Fluids*, pages 450–455, Tucson, Arizona, 1998.

[165] Andréa M. P. Valli, Renato N. Elias, Graham F. Carey, and Alvaro L. G. A. Coutinho. PID adaptive control of incremental and arclength continuation in nonlinear applications. *International Journal for Numerical Methods in Fluids*, Online publication in advance of print, 2009. DOI: 10.1002/fld.1998.

[166] Jasper van den Eshof, Gerard L. G. Sleijpen, and Martin B. van Gijzen. Relaxation strategies for nested Krylov methods. *Journal of Computational and Applied Mathematics*, 177(2):347–365, 2005. DOI: 10.1016/j.cam.2004.09.024.

[167] Henk A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992. DOI: 10.1137/0913035.

[168] S. Pratap Vanka. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *Journal of Computational Physics*, 65(1):138–158, 1986. DOI: 10.1016/0021-9991(86)90008-2.

[169] Rüdiger Verfürth. A combined conjugate gradient-multigrid algorithm for the numerical solution of the Stokes problem. *IMA Journal of Numerical Analysis*, 4(4):441–455, 1984. DOI: 10.1093/imanum/4.4.441.

[170] Judith A. Vogel. Flexible BiCG and flexible Bi-CGSTAB for nonsymmetric linear systems. *Applied Mathematics and Computation*, 188(1):226–233, 2007. DOI: 10.1016/j.amc.2006.09.116.

[171] Wolfgang A. Wall. *Fluid-Struktur-Interaktion mit stabilisierten Finiten Elementen*. PhD thesis, Universität Stuttgart, Fakultät Bauingenieur- und Vermessungswesen, 1999.

[172] Pieter Wesseling and Cornelis W. Oosterlee. Geometric multigrid with applications to computational dynamics. *Journal of Computational and Applied Mathematics*, 128(1–2):311–334, 2001. DOI: 10.1016/S0377-0427(00)00517-3.

[173] Christian Wieners. Robust multigrid methods for nearly incompressible elasticity. *Computing*, 64(4):289–306, 2000. DOI: 10.1007/s006070070026.

[174] Maurice Wilkes. The memory gap (keynote). In *Solving the Memory Wall Problem Workshop*, 2000. `http://www.ece.neu.edu/conf/wall2k/wilkes1.pdf`.

[175] Hilmar Wobker and Stefan Turek. Numerical studies of Vanka-type smoothers in computational solid mechanics. *Advances in Applied Mathematics and Mechanics*, 1(1):29–55, 2009. `http://www.global-sci.org/aamm/volumes/v1n1/pdf/11-29.pdf`.

[176] Peter Wriggers. *Nichtlineare Finite-Element-Methoden*. Springer, Berlin, 2001.

[177] Shi Zeng and Pieter Wesseling. Numerical study of a multigrid method with four smoothing methods for the incompressible Navier-Stokes equations in general coordinates. In N. D. Melson, T. A. Manteuffel, and S. F. McCormick, editors, *Sixth Copper Mountain Conference on Multigrid Methods*, pages 691–708. NASA Langley Research Center, 1993.

[178] Olgierd C. Zienkiewicz and Robert L. Taylor. *The Finite Element Method: The Basis*, volume 1. Butterworth-Heinemann, Oxford, 5th edition, 2000.

[179] Olgierd C. Zienkiewicz and Robert L. Taylor. *The Finite Element Method: Solid Mechanics*, volume 2. Butterworth-Heinemann, Oxford, 5th edition, 2000.

[180] Walter Zulehner. Analysis of iterative methods for saddle point problems: A unified approach. *Mathematics of Computation*, 71(238):479–505, 2002. DOI: 10.1090/S0025-5718-01-01324-2.