No. 604 **June 2019**

PP-Löser für die Navier-Stokes Gleichung

T. Helmich, E. Tiemann

ISSN: 2190-1767

technische universität dortmund

fakultät für mathematik

Master Studienprojekt Modellbildung / Simulation-Technomathematik

PP-Löser für die Navier-Stokes Gleichung

Thomas Helmich und Enno Tiemann

Betreuer
Mirco Arndt
Prof. Dr. Stefan Turek

9. Oktober 2018

Inhaltsverzeichnis

Einleitung

Die Navier-Stokes Gleichung ist ein Gleichungssystem von Differentialgleichungen, welche durch die Geschwindigkeit und den Druck des beschriebenen Fluids gekoppelt sind. Um eine numerische Simulation des Fluids zu berechnen, wird ein Löser benötigt, welcher das Problem in Raum und Zeit diskretisiert und ebenfalls mit der Nichtlinearität der Navier-Stokes Gleichung lösen kann.
Die Ortsdiskretisierung wird durch die finite Elemente Methode erzeugt und die Zeitdiskretisierung wird durch die θ-Zeitschrittmethode erhalten.
Der PP-Löser wurde von Stefan Turek entwickelt und seine Veröffentlichung [2] diente als Hauptquelle dieser Arbeit.

wichtige Variablen

Variable	Bedeutung
Ω	Gebiet
u	Strömungsfeld/Geschwindigkeitsfeld
u^{inflow}	Strömungsfeld am Einflussrand
p	Druck
ν	Viskosität
f	Quellfunktion
Δt	Zeitschrittweite
T	Endzeit
θ	Parameter der Zeitschrittmethode
$level$	Gitterlevel
$h_x = h_y = h$	Gitterweite (konstant in x- und y-Richtung)
$N_x = N_y$	Anzahl der Elemente (konstant in x- und y-Richtung)
L	Laplace Matrix
$K(u)$	Transport Matrix
M_u	Masse Matrix von u
M_u^{lump}	gelumpte Masse Matrix von u
M_p	Masse Matrix von p
M_p^{lump}	gelumpte Masse Matrix von p
B	Gradient Matrix
B^T	Divergenz Matrix
$S(u)$	System Matrix
P	Pressure-Poisson Matrix
F	Rechte-Seite Vektor des diskretisierten Problems
g	Rechte-Seite Vektor des zu lösenden Systems
f_P	Rechte-Seite Vektor des Pressure-Poison Problems
f_{FP}	Druckfehler
$defect$	Geschwindigkeitsdefekt
$correction$	Geschwindigkeitskorrektur
q	Druckkorrektur I
$\tilde{q}$	Druckkorrektur II
α	Skalierung der Masse Matrix
α_R	Skalierung der Druckkorrektur I
α_D	Skalierung der Druckkorrektur II

1 Die Navier-Stokes Gleichung

Das Strömungsfeld u von inkompressiblen Fluiden in einem Raum $\Omega = [0,1]^2 \in \mathbb{R}^2$ wird durch die Navier-Stokes Gleichungen beschrieben. Der Ursprung dieser Gleichungen stammt aus der Mechanik und sie sind die Erhaltungsgleichungen von Impuls (1.1) und Masse (1.2):

$$u_t - \nu\Delta u + u \cdot \nabla u + \nabla p = f, \tag{1.1}$$

$$-\nabla \cdot u = 0. \tag{1.2}$$

Durch das Multiplizieren einer beliebigen Funktion $v \in V$ mit der Gleichung (1.1) beziehungsweise $q \in Q$ mit der Gleichung (1.2) und mit anschließendem Integrieren über das Gebiet Ω, wird die schwache Formulierung der Navier-Stokes Gleichungen formuliert:

$$\int_\Omega \partial_t u \; v \; dx + \int_\Omega (u \cdot \nabla u) \; v \; dx - \nu \int_\Omega \Delta u \; v \; dx + \int_\Omega \nabla p \; v \; dx = \int_\Omega f \; v \; dx, \tag{1.3}$$

$$\int_\Omega (\nabla \cdot u) \; q \; dx = 0. \tag{1.4}$$

Mittels partieller Integration folgt

$$-\nu \int_\Omega \Delta u \cdot v \; dx = \nu \int_\Omega \nabla u : \nabla v \; dx - \nu \int_{\partial\Omega} (\nabla u \cdot n) \cdot v \; dS \tag{1.5}$$

sowie

$$\int_\Omega \nabla p \cdot v \; dx = -\int_\Omega p \nabla \cdot v \; dx + \int_{\partial\Omega} pv \cdot n \; dS. \tag{1.6}$$

Die Randintegrale verschwinden durch die Wahl der Randbedingungen. Eine finite Elemente Methode wird für die Ortsdiskretisierung von Ω genutzt, so dass das diskrete schwache Problem mithilfe des L_2-Skalarprodukts die Form

$$(\partial_t u_h, v_h) + (u_h \cdot \nabla u_h, v_h) + \nu(\nabla u_h, \nabla v_h) - (p_h, \nabla \cdot v_h) = (f, v_h) \; \forall v_h \in V_h, \tag{1.7}$$

$$-(q_h, \nabla \cdot u_h) = 0 \qquad \forall q_h \in Q_h. \tag{1.8}$$

hat.
Jeder Additionsterm von (1.7) und (1.8) stellt eine Matrix dar. Diese Matrizen werden bezeichnet als:

Masse Matrix:	$M_{ij} = (v_h^j, v_h^i)_h,$
Transport Matrix:	$K(u)_{ij} = (u_h \cdot \nabla v_h^j, v_h^i)_h,$
Laplace Matrix:	$L_{ij} = (\nabla v_h^j, \nabla v_h^i)_h,$
Gradient Matrix:	$B_{ij} = -(q_h^j, \nabla \cdot v_h^i)_h,$
Divergenz Matrix:	$B^T = -(\nabla q_h^j, v_h^i)_h,$
Rechte-Seite Vektor:	$F_i = (f, v_h^i)_h.$

Mithilfe dieser Matrizen lassen sich die Gleichungen (1.7) und (1.8) in ein nichtlineares Gleichungssystem überführen:

$$\begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \partial_t u \\ 0 \end{bmatrix} + \begin{bmatrix} K(u) + \nu L & B \\ -B^T & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}. \tag{1.9}$$

Für die zeitliche Diskretisierung wird das θ-Verfahren benutzt, wodurch die implizite Euler Methode oder das Crank-Nicolson Verfahren genutzt werden kann ($\theta = 1$ bzw. $\theta = 0.5$). Diese Zeitdiskretisierung führt zu dem Gleichungssystem

$$\begin{bmatrix} S(u) & B \\ -B^T & 0 \end{bmatrix} \begin{bmatrix} u^{n+1} \\ p^{n+1} \end{bmatrix} = \begin{bmatrix} g \\ 0 \end{bmatrix} \tag{1.10}$$

mit

$$S(u) = \alpha M + \theta \Delta t (K(u) + \nu L) \tag{1.11}$$

und

$$g = [M - (1-\theta)\Delta t(K(u) + \nu L)] \cdot u^n + \theta \Delta t F^{n+1} + (1-\theta)\Delta t F^n. \tag{1.12}$$

Damit eine eindeutige Lösung des Problems gefunden werden kann, werden Randbedingungen benötigt. Dabei wird unterschieden zwischen dem Einfluss, dem Ausfluss und der Wand. Die Geschwindigkeit u^{inflow} beim Einfluss ist bekannt und fest angegeben, während die Geschwindigkeit beim Ausfluss hingegen unbekannt ist. An der Wand wird die 'No Slip'-Bedingung angenommen, sodass dort gilt, dass

$$u^{Wand} = 0. \tag{1.13}$$

Zusätzlich wird eine Anfangsbedingung von u und p benötigt. Dies bedeutet, dass u und p zum Startpunkt $t = 0$ vorgegeben sind. In allen betrachteten Problemfällen wird

$$u(x, t = 0) = u_0(x) = 0 \tag{1.14}$$

und

$$p(x, t = 0) = p_0(x) = 0 \tag{1.15}$$

gesetzt, um aus einer ruhigen Lage zu starten. Falls eine stationäre Lösung gesucht wird, werden ebenfalls diese Anfangsbedingungen verwendet.

2 Implementierung des PP-Lösers

Bei dem PP-Algorithmus müssen einige Parameter geeignet gewählt werden. Darunter fällt die Wahl von der internen Lösungsmethode sowie einiger konstanter Werte. Des Weiteren lässt sich das Zeitschrittverfahren durch θ festlegen. Der Funktionsraum V_h wird fest durch Q^2-Elemente und der Funktionsraum Q_h fest durch P^1-Elemente beschrieben, damit die inf-sup-Bedingung erfüllt wird und damit Stabilität gewährleistet werden kann. In diesem Kapitel werden sowohl diese Einstellungen, als auch die Codestruktur erläutert, damit der Leser versteht, wie der PP-Löser funktioniert und an welcher Stelle diese Parameter im Code festgelegt werden.

2.1 Einstellungen

Alle einzustellende Parameter stehen in den Dateien `Navier_Stokes.m`, `getSolverParameters.m` und `getProblemParameters.m`.
In den ersten Zeilen von `Navier_Stokes.m` werden die verschiedene Testfälle festgelegt. In der Datei `getSolverParameters.m` werden alle für die Lösungsmethode relevanten Werte festgelegt und in der Datei `getProblemParameters.m` wird das zu lösende Problem durch Randdaten und einer Quellfunktion f definiert.

- `Navier_Stokes.m`
  - Method → *VanKan* oder *Chorin*. Unterschiede werden im Abschnitt 2.3 erörtert,
  - Problem → Durch 'a' und 'b' festgelegt. Alle betrachteten Probleme sind Kanalströmungen mit einem Ein- und einem Ausfluss. Der einzige Unterschied zwischen den Problemen sind die unterschiedlichen Einflussfunktionen. Die zugehörigen Dirichlet-Randbedingungen werden in `getProblemParamter.m` definiert,
  - Gitterlevel → Berechnet die Anzahl der Elemente in x- und y-Richtung,
  - Zeitschrittweite Δt → Muss hinreichend klein gewählt werden,
  - Ausgabeeinstellungen → Anzahl der zu erstellenden Bilder für die graphische Darstellung sowie Anzahl der Textausgaben.
- `getSolverParameters.m`
  - Ordnung der Gaußquadratur → Standardeinstellung: 4,
  - Endzeit T,
  - Toleranzen für das PP-, Fixpunkt- und PCG-Verfahren,
  - Maximale Iterationszahl für das PCG-Verfahren,

- Dämpfungsparameter ω_{FP} für die Richardson Iteration,
- Zeitschrittparameter $\theta \rightarrow$ Legt fest, ob impliziter Euler oder CN-Schema verwendet werden soll,
- Viskosität ν,
- Skalierungsfaktor beim Update des Druckes durch den Parameter α_R,
- Skalierung der Massematrix durch den Parameter α,
- Lineare Extrapolation der Geschwindigkeit u (on/off).

- `getProblemParameters.m`
  - Gebiet $\Omega = I_x \times I_y = [0,1]^2$
  - Funktion der rechten Seite $f(x,y,t) = 0$
  - Dirichlet Randfunktion:
    - **Einfluss a:**
      - für $t \leq 1$

$$u^{inflow} := \begin{bmatrix} (1+t) \cdot \sin(y\pi) \\ 0 \end{bmatrix}$$

      - für $t > 1$

$$u^{inflow} := \begin{bmatrix} 2 \cdot \sin(y\pi) \\ 0 \end{bmatrix}$$

    - **Einfluss b:**
      - für $t \leq 1$

$$u^{inflow} := \begin{bmatrix} (1+\cos((1-t)\pi) \cdot \sin(y\pi) \\ 0 \end{bmatrix}$$

      - für $t > 1$

$$u^{inflow} := \begin{bmatrix} 2 \cdot \sin(y\pi) \\ 0 \end{bmatrix}$$

2.2 Allgemeine Codestruktur

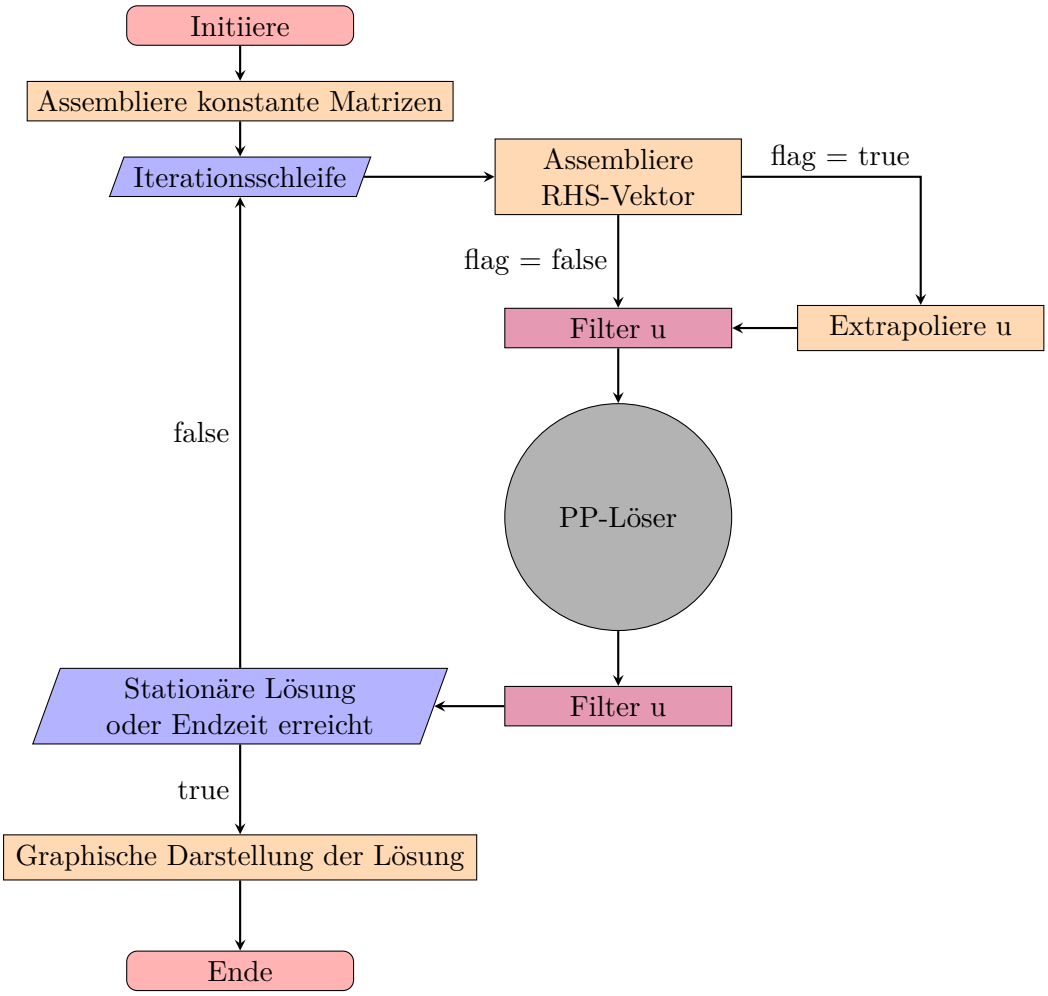

Abbildung 2.1: Flussdiagramm der Datei `Navier_Stokes.m`.

- **Initiiere**
  Ein zu lösendes Problem und die dazu genutzte Methode wird ausgesucht. Abhängig vom Diskretisierungslevel wird mit der Funktion *gittergenerator* die Ortsdiskretisierung und damit auch die Anzahl der Freiheitsgrade festgelegt. Weiterhin werden mit den bereits vorhandenen Informationen die Lösungsmatrizen u_{sol} und p_{sol} festgelegt. Diese Matrizen werden zum Postprocessing beötigt und speichern die Lösungsvektoren von jedem Zeitschritt.

- **Assembliere konstante Matrizen**

  Mithilfe der Information vom Gitter aus dem Schritt 'Initiiere' werden nun die konstanten Matrizen assembliert. Die Matrizen bleiben deshalb konstant, da

keine Gitterdeformationen während der Berechnung entstehen. Diese Matrizen sind die Laplacematrix L, Gradientenmatrix B bzw. Divergenzmatrix B^T, die Massematrix M_u von u und die Massematrix M_p von p. Berechnet werden diese Matrizen durch den Befehl *AssemblyMLB*.
Nun werden auf den Massematrizen M_u und M_p das sogenannte 'Lumping' durch den gleichnamigen Befehl ausgeführt. Im Anschluss wird die Pressure-Poisson Matrix P berechnet.

- **Iterationsschleife**

  Um die Lösung von u und p in jedem Zeitschritt zu berechnen, wird eine Schleife über die Zeit benötigt, welche jeden Zeitschritt durchläuft. Die maximale Iterationszahl dieser Schleife wird vorher mithilfe von Δt und dem gewähltem T berechnet.
  Ist ein stationäres Problem vorhanden, so wird trotzdem über „die Zeit“ iteriert. Sobald eine Lösung gefunden wurde, die sich mit der Zeit nicht mehr ändert, wird die Schleife beendet.

- **Assembliere RHS-Vektor**
  Für den PP-Löser wird der Rechte-Seite Vektor F vom jetzigen Zeitschritt benötigt. Dieser wird mit dem Befehl *AssemblyRHS* berechnet.

- **Extrapoliere** u
  Falls erwünscht, wird eine Extrapolation von u durchgeführt. In diesem Fall wird die Fixpunkt-Schleife nur einmal durchlaufen.

- **Filter** u
  Durch den Befehl *filter_ u_ bc* werden die bekannten Dirichlet-Randwerte von u in den Vektor eingesetzt. Dieser Befehl kommt auch in anderen Programmstücken vor und erfüllt dort den selben Zweck.

- **PP-Löser**
  Der eigentliche PP-Löser wird aufgerufen, um u und p zu berechnen. Der Befehl für den PP-Löser lautet *PP_solver* und wird im nächsten Unterkapitel weiter ausgeführt.

- **Erhöhe Iterationsschritt**
  Zunächst wird die vorher berechnete Lösung in u_{sol} und p_{sol} abgespeichert und anschließend wird geprüft, ob die maximale Iterationszahl erreicht ist oder nicht. Falls ein stationäres Problem vorliegt, werden weitere Abbruchkriterien für den PP-Löser festgelegt. Im Programm wurde als Abbruchkriterium gewählt, dass die Divergenz von u kleiner als eine gewählte Toleranz sein soll.

- **Graphische Darstellung der Lösung**
  Nach erfolgreichem Berechnen aller Zeitschritte wird ein Video von u und p, sowie die Kontur von u und p im letzten Zeitpunkt abgespeichert. Um ein Video mit MatLab aufzunehmen, werden zunächst alle gewünschten Bilder in der richtigen Reihenfolge erstellt, um diese als Frames im Video festzulegen.

2.3 PP-Löser

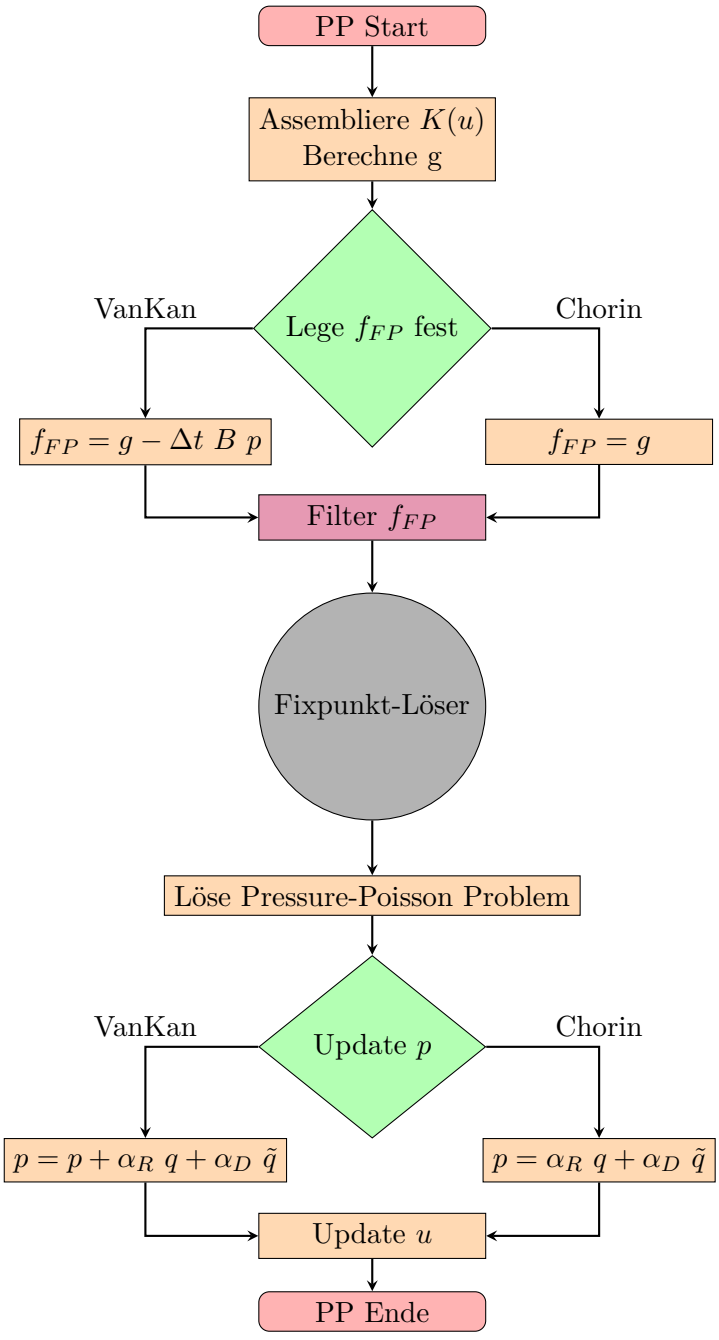

Abbildung 2.2: Flussdiagramm der Datei `PP_solver.m`.

- **Assembliere** $K(u)$
  Da die Transportmatrix K von der Geschwindigkeit u abhängt, muss die Matrix pro Iteration neu assembliert werden. Assembliert wird die Transportmatrix durch den Befehl *AssemblyTransport*.

- **Berechne** g
  Für den Fixpunkt-Löser wird der RHS-Vektor f_{FP} definiert und als Hilfe wird zunächst der Vektor g berechnet. Wegen der Zeitschrittmethode müssen auch einige Wert aus dem vorherigen Zeitschritt verarbeitet werden. Die Gleichung für g lautet

$$g = M_u^{lump} u^n - (1-\theta)\Delta t(\nu L + K)u^n + \theta \Delta t F^{n+1} + (1-\theta)\Delta t F^n.$$

- **Lege** f_{FP} **fest**
  Mithilfe von g lässt sich der Druckdefekt f_{FP} berechnen. Allerdings existieren mehrere Varianten dafür. Die im Code implementierten Methoden lauten *VanKan* und *Chorin*.
  - **Chorin** $f_{FP} = g$
  - **VanKan** $f_{FP} = g - \Delta t B p$

- **Filter** f_{FP}
  Analog zum Filtern von u wird auch f_{FP} mit dem Befehl `filter_u_bc.m` gefiltert.

- **Fixpunkt-Löser**
  Nun wird der Fixpunkt-Löser aufgerufen, um das Problem

$$S(\tilde{u})\tilde{u} = f_{FP}$$

  zu lösen. Dazu wird das Programm *Fixpunkt_solver* aufgerufen, welches die Lösung $\tilde{u}$ liefert. Eine genaue Beschreibung des Fixpunktlösers befindet sich im nächsten Abschnitt.

- **Löse Pressure-Poison Problem**
  Mithilfe des neu berechneten u wird die rechte Seite des Pressure-Poisson Problems aufgestellt:

$$f_P = \frac{1}{\Delta t} B^T u.$$

  Zusammen mit der Pressure-Poisson Matrix P ergibt sich ein linearen Gleichungssystem, welches die Lösung q besitzt:

$$Pq = f_P.$$

  Gelöst wird dieses Gleichungssystem mit dem PCG-Verfahren.
  Zusätzlich wird auch das Problem

$$M_p^{lump} \tilde{q} = f_P$$

  mit dem matlabinternen \-Operator gelöst.

- **Update** p
  Die Lösung q des Pressure-Poisson Problems und die Lösung $\tilde{q}$ wird für das Update von p benutzt. Je nach Methode ergibt sich:
  - **Chorin**
    $$p_{new} := \alpha_R q + \alpha_D \tilde{q},$$
  - **VanKan**
    $$p_{new} := p_{old} + \alpha_R q + \alpha_D \tilde{q}.$$

  Hierbei sind α_R und α_D Dämpfungsparamter. Im Allgemeinen werden folgende Einstellungen gewählt:

  $$\alpha_R := 1 \quad \text{und} \quad \alpha_D := \theta \nu \Delta t.$$

  In der Datei `getSolverParameter.m` kann α_R und in `Navier_Stokes.m` kann α_D verändert werden.

- **Update** u
  Ähnlich zum Update vom Druck p wird q genutzt, um die Geschwindigkeit u vom neuen Zeitschritt zu berechnen:

  $$u_{new} := \tilde{u} - \Delta t (M_u^{lump})^{-1} B q.$$

- **PP Ende**
  Der neue Druck p_{new} und die neue, gefilterte Geschwindigkeit u_{new} werden zurückgegeben. Für die Konsolenausgabe wird ebenfalls die Norm des Druckdefekt f_P übergeben.

2.4 Fixpunkt-Löser

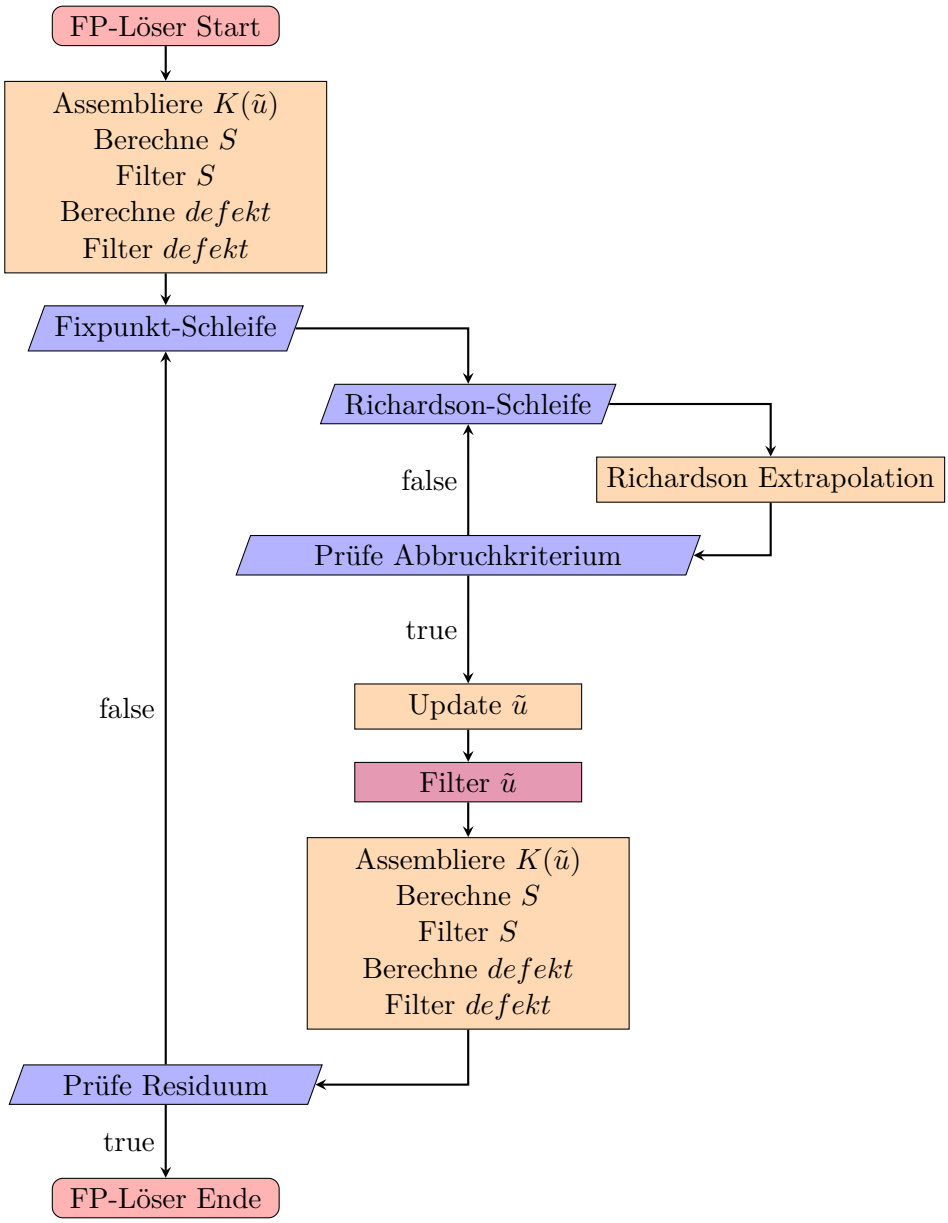

Abbildung 2.3: Flussdiagramm der Datei `Fixpunkt_solver.m`.

- **FP-Löser Start**
  Zum Start wird der Anfangsdefekt von $\tilde{u}$ berechnet.

- **Berechne S**
  Nachdem die Matrix $K(\tilde{u})$ assembliert wurde, kann die Matrix

  $$S = \alpha M_u + \theta \Delta t (K + \nu L)$$

  berechnet werden.

- **Filter S**
  Da die Matrix S von u abhängt, muss die Matrix S ebenfalls gefiltert werden. Dies bedeutet in diesem Fall, dass die Zeilen, die analog zu u zu den Rand-Freiheitsgraden gehören, zu einer entsprechenden Einheitszeile gesetzt werden.

- **Berechne defekt**
  Der Defekt wird nach der Vorschrift

  $$defekt = f_{FP} - S \cdot \tilde{u}$$

  berechnet.

- **Filter defekt**
  Da der Wert von u an den Rändern durch die Randwerte vorgegeben ist, ist folglich der Defekt an den selben Stellen null. Dies wird durch die Funktion `filter_defect_bc.m` erreicht.

- **Fixpunkt-Schleife**
  Die Fixpunkt-Schleife wird durch das Erfüllen des Abbruchkriteriums abgebrochen.

- **Richardson-Schleife**
  Anders als die Fixpunkt-Schleife wird die Richardson-Iteration immer spätestens nach gegebenen Parameter abbrechen.

- **Richardson Extrapolation**
  Der eigentliche Löser ist das Richardson-Verfahren. Dazu wird im ersten Schleifendurchlauf die Korrektur gleich dem negativen Defekt gesetzt. In jeder Iteration wird die Korrektur ausgebessert nach der Vorschrift

  $$correction_{new} = (I - S) \cdot correction_{old} + defekt.$$

- **Prüfe Abbruchkriterium**

  Das Richardson-Verfahren bricht ab, sobald der relative Fehler kleiner als $SolverParameters.TOL\_richardson$ wird.

- **Update $\tilde{u}$**
  Mit der berechneten Korrektur kann nun die Approximation der Geschwindigkeit $\tilde{u}$ ausgebessert werden:

  $$\tilde{u}_{new} = \tilde{u}_{old} + correction.$$

- **Filter $\tilde{u}$**
  Analog zu u wird $\tilde{u}$ mit der Funktion `filter_u_bc.m` gefiltert.

- **Prüfe Residuum**
  Nachdem der Defekt des ausgebesserten $\tilde{u}$ berechnet wurde, gibt nun der relative Fehler eine Auskunft darüber, wie gut ausgebessert wurde. Falls der relative Fehler kleiner als *SolverParameters.TOL_FP* ist, dann wird die Fixpunkt-Schleife abgebrochen.

- **FP-Löser Ende**
  Zum Schluss wird die Lösung des Fixpunkt-Problems $\tilde{u}$, sowie auch die Norm des Defektes am Anfang und am Ende zurück gegeben. Letzteres wird nur für die Text-Ausgabe gebraucht.

3 Ergebnisse

Bei allen Test werden die selben Gitter- und Zeitschrittweiten genutzt, nämlich

$$h = \frac{1}{16}$$

und

$$\Delta t = 10^{-3}.$$

Weiterhin wird auch der Parameter

$$\alpha_R = 1$$

konstant gewählt. Das θ wird in Abhängigkeit der genutzten Methode gewählt. So wird bei der Chorin-Methode das θ auf 1 gesetzt und bei der VanKan-Methode auf 0.5. Die Unterschiede beschränken sich daher auf die Methode (Chorin oder VanKan) und die gewählte Einflussbedingungen (Problem a oder b). Um die berechneten Lösungen von u und p zu überprüfen, wird der Ein- und Ausfluss von u gleichzeitig abgebildet. Wenn die Divergenz von u null ist, dann muss der Ein- und Ausfluss identisch sein. Für beide Probleme gilt, dass der Einfluss ab $t > 1$ konstant ist, sodass sich mit genug Zeit ein stationärer Fluss ergibt. Falls eine stationäre Lösung erreicht wird, so ähnelt die Lösung einer Poiseuille-Strömung (auch bekannt als Kanalströmung). Dafür typisch ist ebenfalls ein identischer Ein- und Ausfluss, weshalb der Vergleich zwischen den Flüssen als eine Art Benchmark genutzt werden kann. Des Weiteren nimmt der Druck bei einer Poiseuille-Strömung linear in Flussrichtung ab [1].
Bei beiden Problemen werden die selben Abbruchkriterien für den PP-Löser genutzt. Diese lauten:

$$T = 10,$$
$$SolverParameters.TOL\_PP = -1.$$

Da die Toleranz negativ gewählt ist, bricht der Algorithmus definitiv erst zur gewählten Endzeit ab.

3.1 Problem a)

Der Geschwindigkeitsverlauf für Ein- und Ausfluss können in der Abbildung 3.1 und der Druckverlauf kann in Abbildung 3.2 eingesehen werden. Die Geschwindigkeitsverläufe von Ein- und Ausfluss verhalten sich identisch, sodass die Strömung die Massenerhaltung erfüllt. Der Druckverlauf verhält sich nur annähernd linear, was daran liegt, dass das Gebiet keinen Kanal darstellt. Dafür müsste das Gebiet in Richtung der x-Achse wesentlich größer als in y-Richtung sein, was bei den betrachtenden

Problemen nicht der Fall ist. In Abbildung 3.3 und 3.5 wird die Geschwindigkeit und der Druck gezeigt.
Wird das Problem mit der VanKan Methode gelöst, so ändert sich an der berechneten Lösung nichts. Die Abbildungen 3.4 und 3.6 verifizieren dies.

3.2 Problem b)

Die Lösung zum letzten Zeitpunkt ist aufgrund des Einflusses die selbe Lösung wie bei Problem a) und beide Methode (Chorin und VanKan) führen zu den selben Ergebnissen wie bei Problem a).

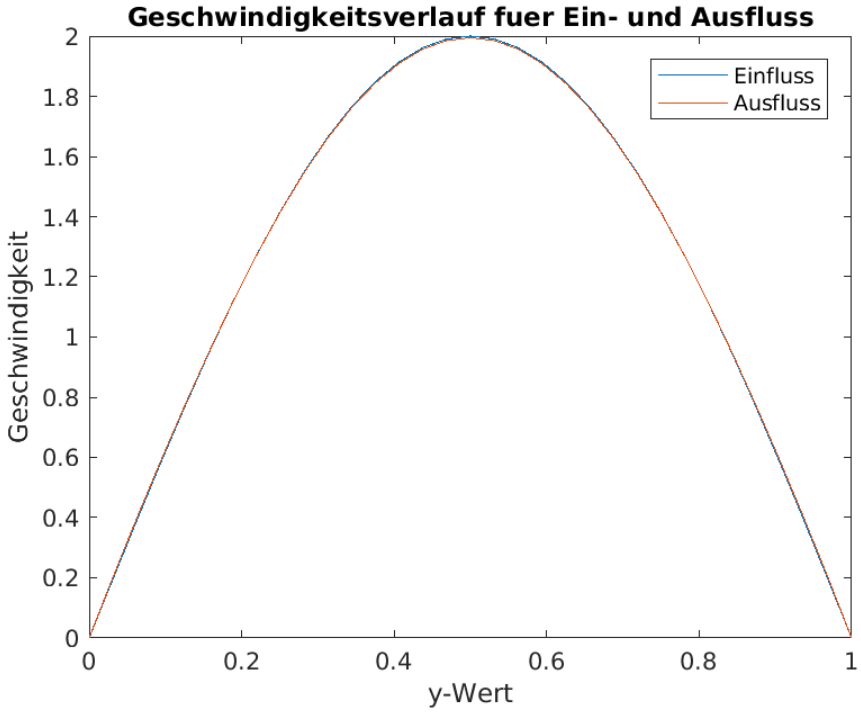

Abbildung 3.1: Ein- und Ausflussgeschwindigkeit zum letzten Zeitpunkt mit der Chorin Methode berechnet.

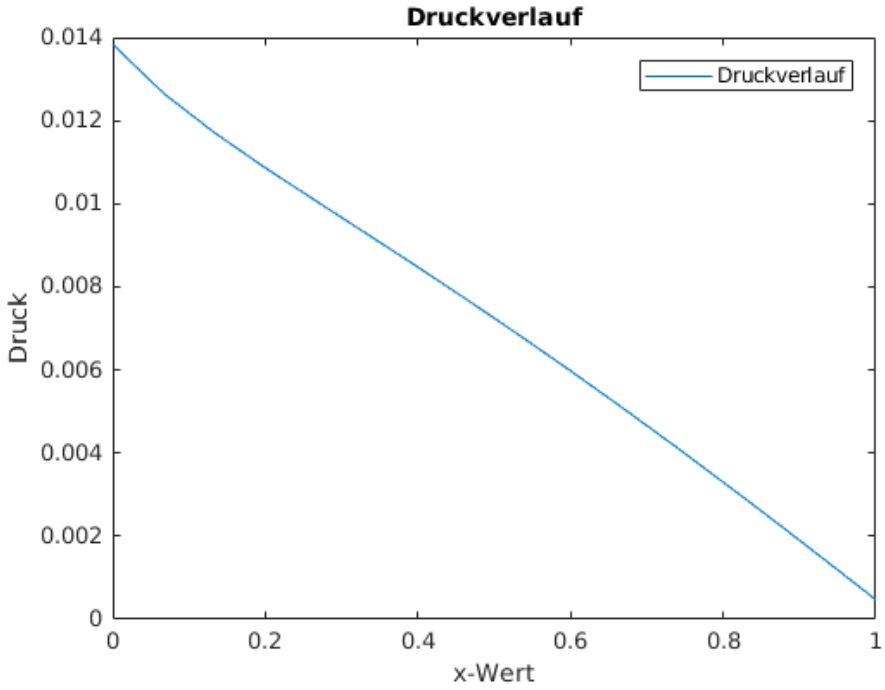

Abbildung 3.2: Druckverlauf zum letzten Zeitpunkt mit $y = 0.3$ und mit der Chorin Methode berechnet.

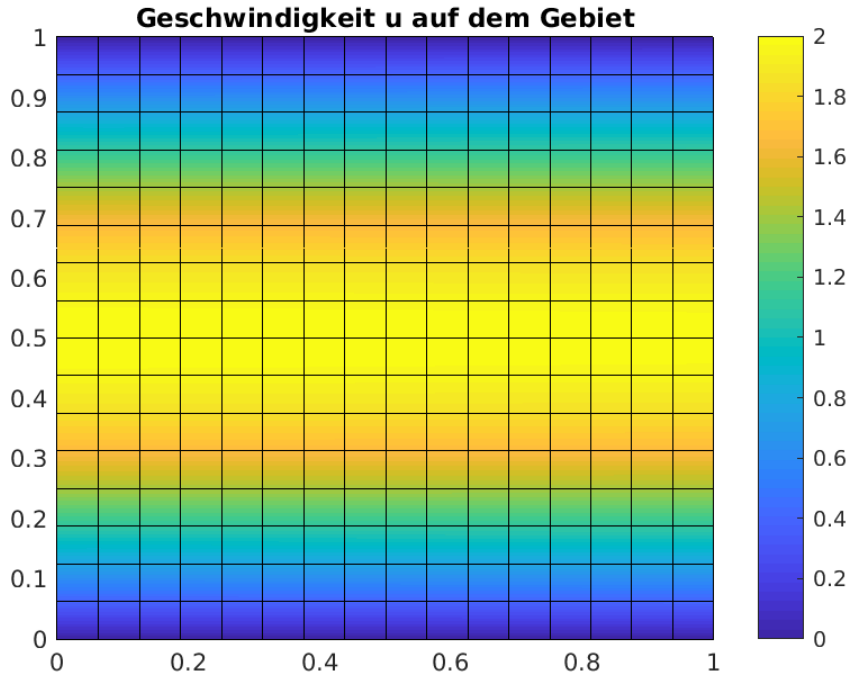

Abbildung 3.3: Geschwindigkeit zum letzten Zeitpunkt mit der Chorin Methode berechnet.

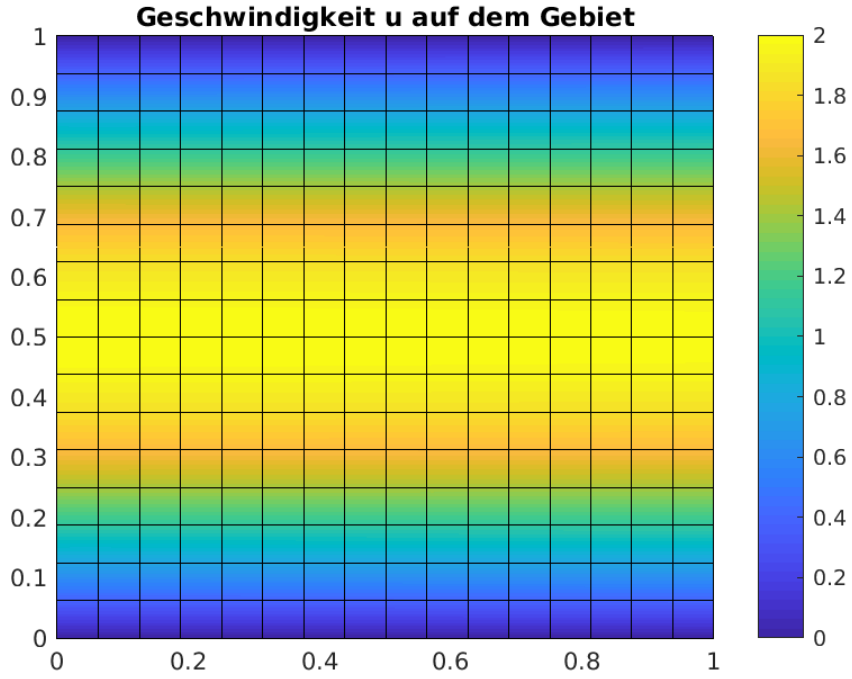

Abbildung 3.4: Geschwindigkeit zum letzten Zeitpunkt mit der VanKan Methode berechnet.

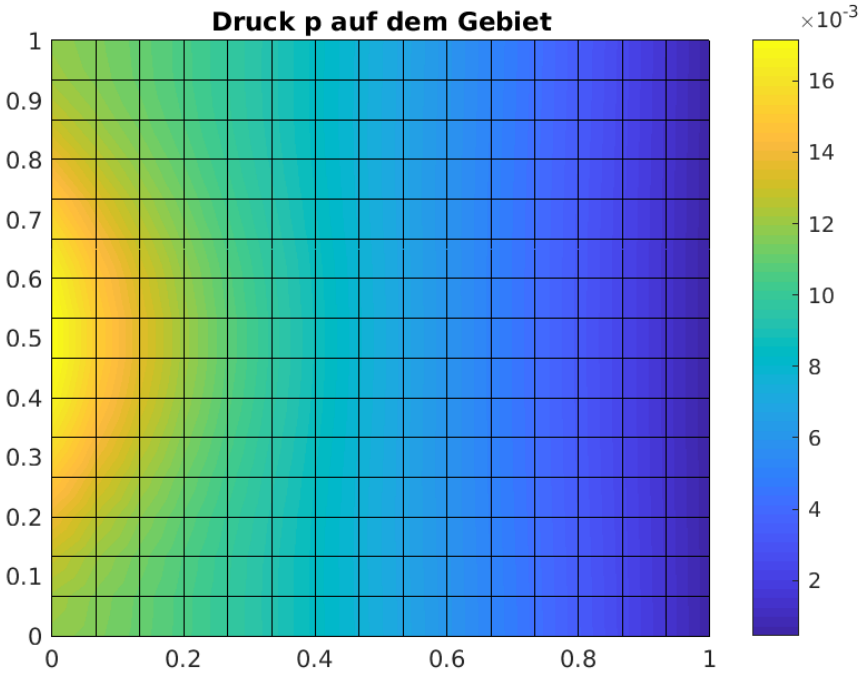

Abbildung 3.5: Druck zum letzten Zeitpunkt mit der Chorin Methode berechnet.

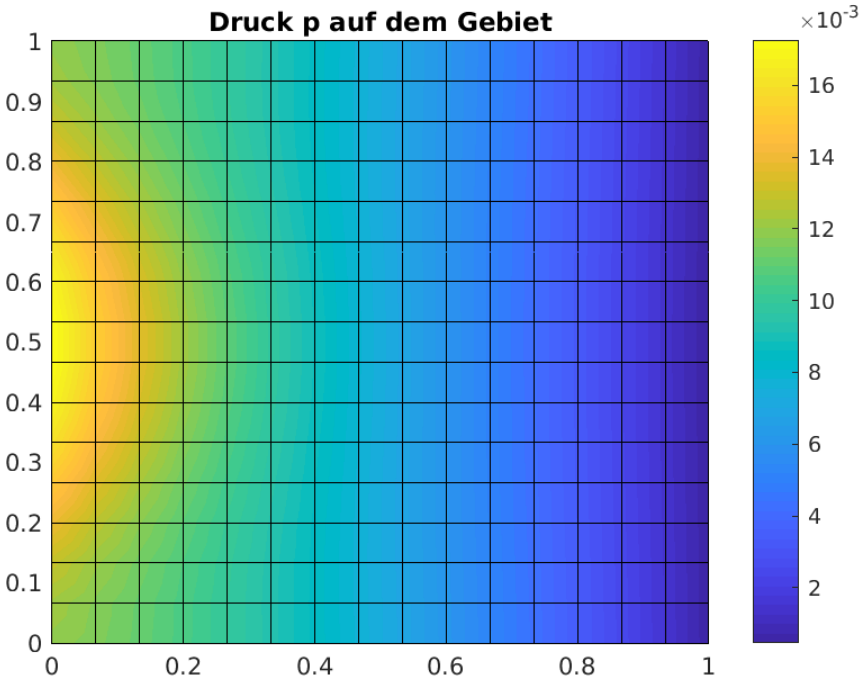

Abbildung 3.6: Druck zum letzten Zeitpunkt mit der VanKan Methode berechnet.

4 Fazit

Der PP-Löser berechnet Ergebnisse, welche physikalisch sinnvoll sind. So wird bei beiden Problemen die selbe stationäre Lösung berechnet. Diese Lösung ist ebenfalls näherungsweise eine Kanalströmung, was den Code weiter verifiziert. Eine Alternative zu dem Fixpunkt-Verfahren bildet die Newton-Methode. Newton hat theoretisch eine höhere Konvergenzordnung von 2 statt nur 1, allerdings ist das Fixpunkt-Verfahren robuster gegenüber fehlerhafte Randdaten. Diese Eigenschaft müsste mithilfe eines anderen Codes verifiziert werden. In beiden Fällen ist der PP-Algorithmus ein Verfahren, welches die Nichtlinearität der Navier-Stokes Gleichung lösen kann.

Literaturverzeichnis

[1] Stefan Kopecz. *Ein gekoppeltes Finite-Elemente/Discountinuous-Galerkin-Verfahren zur Simulation von Strömungs-Transport-Problemen.* 2012.

[2] Stefan Turek. *Effcient solvers for incompressible flow problems: An algorithmic approach in view of computational aspects.* 1999.