This is a short guid to using our implementation.

as examples, we will first consider the application IBVP1_Results_I for real simulation and then IBVP3_Results_I for analytical simulation

IBVP1_Results_I

This code was created by modifying the code 'cc2d', which is found in Featflow2/applications, and saved in featflow2/area51.

to see how the modifications were done, please use the linux command 'takediff' to compare the files in 'cc2d/src' with those in 'IBVP1_Results_I/src'

The code contains many lines from cc2d code, which are unneeded and related to special issues concerning Navier-Stokes equation.

However, to save time, these lines left untouched but their work was cancelled from the input dat files (see ./data)

In this short guide, we shall show how to use this code to solve problem of section 4.1.1 in my thesis.

To solve the problem of figure 4.1 using, for example, the anisotropic mesh, do the following steps

- Create the domain and generate the coarse mesh using DeViSoR.Grid 3D. you can Load this Software, in our institute by opening a linux Terminal and typing:

```
> module Load devisor/grid/3.0.25 ↵
> grid3 ↵
```

this software generates two files with extensions 'prm' and 'tri'

save these files in 'IBVP1_Results_I/pre'

The required files for our problem have been already created and saved in the folder 'pre'

- go to 'IBVP1_Results_I/data and open the file 'master.dat' and do the following:

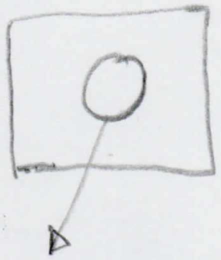
$\boxed{1}$ at Line 219 and Line 220 set the name of 'prm' and 'tri' files, which you created by 'DeViSoR.Grid' and saved in 'IBVP1_Results_I/pre'. that is,

chang this!

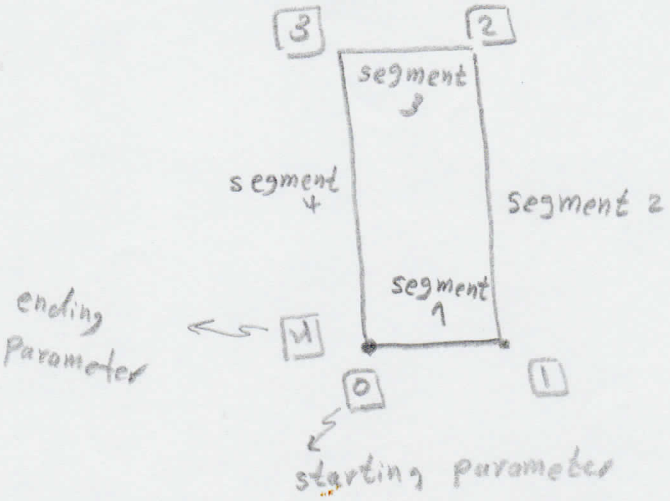
```
sParametrisation = '%{spredirectory}/filename.prm'
sMesh = '%{spredirectory}/filename.tri'
```

$\boxed{2}$ apply the BCs. 1st, the domain (we generated with DeViSor.Grid 3D) is illustrated by the following Figure, in which the boundary Segments and the boundary point numbers are shown..

an example for 2 components of bCs :



the hole is 2nd boundary component



The application BC's in 'master.dat' is explained below :

title

boundary component

number of Lines below

```
##############
[BDCONDITIONS]
##############
bdComponent1(4)=
   1.0  3   0   1   0   1   0   1   '  '   'D0'   '  '   'D0'   '  '   'D0'
   2.0  3   1   0   1   0   1   0   'D0'   '  '   'D0'   '  '   'D0'   '  '
   3.0  3   0   0   0   0   0   0   '  '   '  '   '  '   '  '   '  '   '  '
   4.0  3   1   0   1   0   1   0   'D0'   '  '   'D0'   '  '   'D0'   '  '
```

U_{s1} U_{s2} V_{s1} V_{s2} V_{f1} V_{f2}

1: the DoF is given Dirichlet bc.
0: " " " " Neumann bc.

Neumann bc's left blank and imposed in ccgeneraldiscretisation.f90 Line 1856-1860, 2095-2100

see 'bd Expressions' in the data file 'bdconditions.dat'.

This works only for Dirichlet. However in the current cc2d code you can also use it for Neumann bc

0: start and end point don't belong to the interval
1: only start point belong
2: only end point belong
3: start and end point belong.

if a DoF is given a Dirichlet & Neumann bc then the Dirichlet condition is applied

end parameter, where the start parameter is automatically set to $\boxed{2}$ (the end parameter of the previous line)

The input control data in 'master.dat' are already found in the other data files and their tasks are well explained there.

However, pay attention that the explanation is based on cc2d code which assumes that we have V_1, V_2 for velocities and p for pressure while in our applicatin we have 7 components: us_1, us_2, vs_1, vs_2 v_{F_1}, v_{F_2} (or w_1, w_2 in IBVP2_* and IBVP_3*) and p. and there are many control parameters inhirited from cc2d code, which we do not need, such as some viscosity models, ... etc. and their effect is cancelled by setting the suitable control values

Furthermore, the input parameter 'SMesh' is found in 'paramtriang.dat' and if you want to use it in 'master.dat', do not forget to copy '[PARAMTRIANG]' which is name of the commads group. the same applies to othe control parameters in other data file to run the code open Linux Terminal, then type

```
... IBVP1_Results_I > module purge
                    > module load gcc/4.9.0 openblas/0.2.9

           > ./configure --id=pc64-sandybridge-linux-
             gcc-openblas --opt

           > make -j 8

          > cc2d_elastBPM
```

<u>or</u>

```
> cc2d_elastBPM    ./data/master.dat
```

However if you change the name of master file to newName.dat then, you must type

```
> cc2d_elastBPM   ./data/newName.dat
```

the output solutions for the selected points
in the domain are found in folder 'ns'
and the log file is found in folder 'log'
while the contour plots and the shapes
of configurations with time are stored
as gmv files in folder 'gmv' and
and viewed by gmv (general mesh viewer software)
assume the gmv files are u.gmv.0000, ... u.gmv.7938

```
> module gmv
```

open the preferred gmv file with gmv software

```
> gmv file.gmv
```

then do some settings (for example color bars
, title, —) and save as attribute file
, say for example) 'aaa.attr'. then type
in Linux terminal

```
> gmvmpeg4 -a aaa.attr -i u.gmv.%%%%% -fls 0,7938 -o nF -x 800 -y 600 --videoformat x264 -I -o
tmp -r 25.0 -j 2
> mplayer tmp.x264.avi
```

in the previous discussion, we have shown how to perfurm real FE Simulation and we used the application 'IBVP1_Results_I' as an example for this purpose.

Now, we will show the missing information for analytical simulation.

and we rfer in our discussion to Section 4.2.1 of my Thesis and the second problem. That is, Figure 4.16

and
$$f_u = \begin{pmatrix} y^3 - 1 - \frac{1}{4} t \cdot y \\ 0 \end{pmatrix}$$

$$f_w = \begin{pmatrix} 10 \cdot y^3 \cdot t + 2 y_2^3 - 1 \\ 0 \end{pmatrix}$$

with
$$u_{s1} = \frac{1}{24} y^3 \cdot t$$

$$u_{s2} = 0$$

$$w_1 = y^3 \cdot t$$

$$w_2 = w_2$$

$$p = \frac{1}{2} - x_1$$

The steps go as follow :

- in file 'master.dat' set UL = 0
- in 'cccallback.f90' define the rhs source terms $f_u=\begin{bmatrix}f_{u1}\\f_{u2}\end{bmatrix}$, $f_w=\begin{bmatrix}f_{w1}\\f_{w2}\end{bmatrix}$ the true solutions and the boundary values, where X= DPoints(1,:,:) , y=Dpoints(2,:,:) and time=dt.
  - for $f_{u1} = y^3 - \frac{1}{4}ty - 1$;

```
subroutine   coeff_RHS_uSx (-----)

      :
      :
      :

   Dcoefficient(1,:,:) = DPoints(2,:,:)**3
   - dt*Dpoints(2,:,:)/4.0_DP -1.0_DP

   end subroutine
```

and do the same step for

Source term	subroutine
f_{u1}	coeff_RHS_usx
f_{u2}	coeff_RHS_usy
f_{w1}	coeff_RHS_VFx
f_{w2}	coeff_RHS_VFy

10

- in 'cccallback.f90', define the true solutions u_{s1}, u_{s2}, v_{s1}, v_{s2}, v_{F1}, v_{F2} and p. For example,

to define $\underline{\underline{u_{s1} = \frac{1}{24} y^3 \cdot t}}$, do the following:

where $x \equiv$ Dpoints(1,:,:), $y \equiv$ Dpoints(2,:,:) and $time \equiv dt$

```
subroutine ffunction_TargetuSx (....)

.
.
.
.
.

    select case (cderivative)
    case (DER_FUNC);     Dvalues(:,:) =  Dpoints(2,:,:)**3*dt/24.0_DP → Us1
    case (DER_DERIV_X);  Dvalues(:,:) =  0.0_DP                       → ∂Us1/∂x
    case (DER_DERIV_Y);  Dvalues(:,:) =  Dpoints(2,:,:)**2*dt/8.0_DP
    case (DER_DERIV_XX); Dvalues(:,:) =  0.0_DP
    case (DER_DERIV_XY); Dvalues(:,:) =  0.0_DP
    case (DER_DERIV_YY); Dvalues(:,:) =  Dpoints(2,:,:)*dt/4.0_DP → ∂²Us1/∂y²
    end select

end subroutine
```

true solution Term	subroutine
u_{s1}	ffunction_TargetuSx
u_{s2}	ffunction_TargetuSy
v_{s1}	ffunction_TargetvSx
v_{s2}	ffunction_TargetvSy
v_{F1}	ffunction_TargetvFx
v_{F2}	ffunction_TargetvFy

The dirichlet bc's for our analytical Simulation are defined in 'cccallback.f90' in subroutine 'getBoundaryValues'. However, we need 1st to tell the code to read from this subroutine. Therefore, we use 'A' in 'master.dat'

```
##############
[BDCONDITIONS]
##############

bdComponent1(4)=
   1.0    3    1   1   1   1   1   1  'A' 'A' 'A' 'A' 'A' 'A'
   2.0    3    1   1   1   1   0   0  'A' 'A' 'A' 'A' ' ' ' '
   3.0    3    1   1   1   1   1   1  'A' 'A' 'A' 'A' 'A' 'A'
   4.0    3    1   1   1   1   1   1  'A' 'A' 'A' 'A' 'A' 'A'
```

Neumann bc's are left blank and always read in ccgeneraldiscretisation.f90 and cccallback.f90 as we will show in the next page

next, we use dx, dy and dt. for x-c, y-c and time and apply Dirich. BC's as shown below. 'd' in 'dx' to indicate double precision

```
subroutine getBoundaryValues (sexpressionName,icomponent,rdiscretisation,&
                              rboundaryRegion,dwhere, dvalue, rcollection)

                    :
                    :

    if (icomponent .EQ. 1) then
      dvalue =   dy**3*dt/24.0_DP
    elseif (icomponent .EQ. 2) then
      dvalue   =   0.0_DP
    elseif (icomponent .EQ. 3) then
     dvalue = dy**3/24.0_DP
    elseif (icomponent .EQ. 4) then
      dvalue = 0.0_DP
    elseif (icomponent .EQ. 5) then
      dvalue = dy**3*dt
    elseif (icomponent .EQ. 6) then
       dvalue = 0.0_DP
    else
  end if
end subroutine
```

→ U_{s1}

→ U_{s2}

→ V_{s1}

→ V_{s2}

→ W_1

→ W_2

The zero Neumann bc's are automatically aunderstood by the code. We need only to specify the non zero Nuemann bc's. To do so, we open the source file 'ccgeneraldiscretisation.f90' in the folder src
and specify the Neumann regions in the 'subroutine cc_generateBasicRHS' as shown below:

```
! -------------------------------------------------------------------------
  subroutine cc_generateBasicRHS (rproblem,rasmTemplates,rrhsassembly,rrhs)
! -------------------------------------------------------------------------


                :
                :
                :

    ! variable for selecting a specifig boundary region
    type(t_boundaryRegion) :: rboundaryRegion
    call boundary_createRegion(rproblem%rboundary, 1, 2, rboundaryRegion)
! 1: boundary componet (if, e.g, the domain has a hole, then the hole is 2nd component)
! 2: boundary segment (see 'master.dat' to figure out the neumann segment)

                :
                :
                :

! see 'master.dat' to figure out the neumann segments
      rboundaryRegion%dminParam = 1.0_DP ! you may need to inspect 'prm' and 'tri' files
      rboundaryRegion%dmaxParam = 2.0_DP ! in 'folder pre' to see the parameter values
      iblock = 5 ! 1 for T_S_1, 2 for T_S_2, .. 5 for T_F1
      call linf_buildVectorScalarBdr2d(rlinform, CUB_G2_1D, .false., &
          rrhs%RvectorBlock(iblock), RHS_2D_surf, rboundaryRegion, rproblem%rcollection)
! go to RHS_2D_surf in cccallback.f90 and specify the neumann bc's
          :
          :
  end subroutine
! -------------------------------------------------------------------------
```

Finally, we open cccallback.f90 and apply the neumann bcs using subroutine 'RHS_2D_surf' as shown below

```
! -------------------------------------------------------------------------
  subroutine RHS_2D_surf (rdiscretisation, ...)
! -------------------------------------------------------------------------
:
:
    Dcoefficients(1,:,:) = 0.5_DP

  end subroutine RHS_2D_surf
! -------------------------------------------------------------------------
```