Feat_FloWer Manual

Raphael Münster, Robert Jendrny

May 24, 2018

Table Of Contents

Chapter 1

Installation

1.1 Prerequisites

The Feat_FloWer software package is accessible through the version control system *Git*. The main repository of the software is hosted on the servers of the TU Dortmund. In order to checkout a copy of the code an account that provides access to the LS3 servers is required, so it is recommended to get such an account. Furthermore, the code repository does not include the meshes used in the example applications. The meshes are stored in a different repository. So, at first we will get this repository and set up an environment variable so that the configuration mechanism of Feat_FloWer can find the mesh repository. The mesh repo can be checked out by:

```
> git clone ssh://username@lannister/home/user/git/mesh_repo.git
```

If the above command does not complete successfully it is very likely that you do not yet have access rights to the git mesh repository. In this case please ask your administrator to grant you the required access rights. After you have checked out the mesh repository, it is time to set up the environment variable for Feat_FloWer. For the `BASH` or `ZSH` shells this can be done by adding the following line to your *.bashrc* or respectively *.zsh*:

```
> export Q2P1_MESH_DIR=/path/to/meshrepo/mesh_repo
```

1.2 Checking out the Code from the Repository

When you have acquired an appropriate account you can clone the Feat_FloWer repository by opening a terminal and entering the following sequence of commands:

```
> mkdir FeatFlower && cd FeatFlower
> git clone --recursive ssh://username@lannister/home/user/git/Feat_FloWer.git
```

This will create a parent folder called FeatFlower and within this folder the source code will be contained. This is a preparation for building the code from source. The Feat_Flower code supports only out-of-source builds, meaning the binary files are not built in the same directory as the source code. This is done in order to prevent that binary files or object files litter the source directory or that these files show up in the version control system.

1.3 Building the Code from Source

The Feat_FloWer code can be build on Linux and Windows operating systems (and probably on MacOS, but this is not tested yet). On Linux systems an installation of the GNU compiler is needed, including the gfortran compiler and the matching OpenMPI libraries. The minimum requirement for the GCC is version 4.9.2. On Linux systems the Intel compiler can be used as an alternative compiler. On Windows systems the Intel compiler is a necessary requirement.

1.3.1 Linux Systems

After you have cloned the repository, navigate to the *FeatFlower* folder that you have created in the previous step and create another folder that will contain the binaries:

```
> mkdir bin
```

Verify with the command **pwd** that your FeatFlower folder now contains two subfolders *bin* and *Feat_FloWer*. At the core of the Feat_FloWer build system is the multi-platform build files generator CMake. On the servers of the TU Dortmund CMake is available as a loadable module. In order to successfully compile the code with the GNU compiler the following components are needed:

- GCC the GNU Compiler Collection including the gfortran compiler
- A matching OpenMPI installation
- CMake with a minimum version of 2.8

On the servers of the TU Dortmund load for example the following modules or newer versions of them:

- > module load gcc/6.1.0
- > module load openmpi/gcc6.1.x/1.10.2/non-threaded/no-cuda/ethernet
- > module load cmake/3.5.2-ssl

In order for CMake to use the compiler that has been added by the module system it is recommended to set the environment variables `CC`, `CXX`, `FC` accordingly. In detail this this means executing the commands (for `BASH` or `ZSH` shells):

- > export CC=$(which mpicc)
- > export CXX=$(which mpicxx)
- > export FC=$(which mpif90)

for the `TCSH` or `CSH` shells the syntax:

- > setenv CC ‘which mpicc‘
- > setenv CXX ‘which mpicxx‘
- > setenv FC ‘which mpif90‘

After you have typed these commands, verify that these module are loaded by checking the output of the command **module list**. The build of the basic software package is initiated by navigating into the *bin* folder that you have created before and evoking CMake from there:
> cd bin
> cmake -DBUILD_METIS=True ../Feat_FloWer
The build process is then started by the command:
> make -j 5
The option <-j 5> starts a parallel build using 5 processes.

1.3.2 Windows Systems

Under construction

Chapter 2

Running a FeatFlower Application

2.1 Introduction to the Application Framework

A FeatFlower application is a particular simulation case that is based on the underlying Finite-Element-Method framework. The possibility of a FeatFloWer application range from very simple CFD-Setups to well-known benchmark configurations to complex realistic configurations that have an industrial or scientific background. The FeatFlower code base comes with a number of preconfigured applications that demonstrate the capabilities of the software. The user can then change and adapt these preconfigured applications to his needs or add completely new applications based on the Featflower framework. A FeatFloWer applications roughly consists of a preprocessing step, a solving procedure and an output of the solution. At first we will introduce the tools used in the preprocessing step.

2.1.1 Preprocessing and Mesh Generation

The main task in the preprocessing step is the creation of the mesh used for the FeatFloWer application. A mesh can be acquired by an external mesh generator, by manual construction or by taking a mesh from the FeatFloWer mesh repository.

2.1.1.1 Using Meshes from the Mesh Repository

Taking a mesh from the mesh repository is the easiest scenario for a new user. Usually this means that a benchmark configuration or a preconfigured application is run by the user. All the user has to do in this case is to partition the mesh according to the

number of MPI processes that the user has to his disposal. The preferred partitioning tool for meshes from the mesh repository is the Python command line program *PyPartitioner.py*. In order to demonstrate the use of the python partitioning tool, we will use the application *q2p1_fc_ext*. The application can be found in the *Feat_FloWer/applications* directory. The python partitioner tool can be found in the *Feat_FloWer/tools* directory. For our first contact with the tool, we will just copy the python files from the *Feat_FloWer/tools* directory to the *bin/applications/q2p1_fc_ext*. So start by navigating to the *bin/applications/q2p1_fc_ext* and copying the python files by:

```
> cp ../../../Feat_FloWer/tools/*.py .
```

Additionally, we need to copy the metis library so that the PyPartitioner can interface with that library:

```
> cp ../../../bin/extern/libraries/metis-4.0.3/Lib/libmetis.so .
```

We can now invoke the PyPartitioner by typing:

```
> python PyPartitioner.py 4 1 1 NEWFAC _adc/2D_FAC/2Dbench.prj
```

which will take the mesh referenced in the project file *_adc/2D_FAC/2Dbench.prj* and decompose the domain into 4 partitions which are then stored in the *NEWFAC* directory. The additional 1 1 parameters denote the type of the partitioning method. As the simulation requires a master node to control some processes the simulation should be launched with 4+1 processes by:

```
> mpirun -np 5 ./q2p1_fc_ext
```

2.2 q2p1_cc (*by R. Jendrny*)

2.2.1 Systems that can be solved

Using this application you can solve the following systems in a coupled way:

- Stationary (Navier-) Stokes equations

$$\begin{pmatrix} L + K(u) & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix},$$
$$\text{with } K(v)w \sim v \cdot \nabla w.$$

  In a defect-correction procedure it is written as:

$$\begin{bmatrix} u^n \\ p^n \end{bmatrix} = \begin{bmatrix} u^{n-1} \\ p^{n-1} \end{bmatrix} + \begin{bmatrix} L + R(u^{n-1}) & B \\ B^T & 0 \end{bmatrix}^{-1} \left(\begin{bmatrix} g \\ 0 \end{bmatrix} - \begin{bmatrix} L + K(u^{n-1}) & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} u^{n-1} \\ p^{n-1} \end{bmatrix} \right)$$
$$\text{with } R(u^{n-1}) = K(u^{n-1}) + \alpha \bar{M}(u^{n-1}).$$

Changing the value α you can switch between Fixpoint and Newton method. If $\alpha = 0$ the pure Fixpoint method will be used. If $\alpha \neq 0$ the code uses an adaptive technique to switch between both methods.

- unsteady (Navier-) Stokes equations
  - general θ-scheme

$$\begin{pmatrix} M + \theta\Delta t\{L + K(u^{n+1})\} & \Delta t B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u^{n+1} \\ p^{n+1} \end{pmatrix} = \begin{pmatrix} Mu^n - (1-\theta)\Delta t\{L + K(u^n)\}u^n \\ 0 \end{pmatrix}$$

  - Backward Difference Formula (BDF)
    - BDF(1): Backward Euler method is the same.
    - BDF(2):

$$\begin{pmatrix} M + \frac{2}{3}\Delta t\{L + K(u^{n+1})\} & \frac{2}{3}\Delta t B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u^{n+1} \\ p^{n+1} \end{pmatrix} = \begin{pmatrix} \frac{4}{3}Mu^n - \frac{1}{3}Mu^{n-1} \\ 0 \end{pmatrix}$$

    - BDF(3):

$$\begin{pmatrix} M + \frac{6}{11}\Delta t\{L + K(u^{n+1})\} & \frac{6}{11}\Delta t B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u^{n+1} \\ p^{n+1} \end{pmatrix} = \begin{pmatrix} \frac{18}{11}Mu^n - \frac{9}{11}Mu^{n-1} + \frac{2}{11}Mu^{n-2} \\ 0 \end{pmatrix}$$

2.2.2 q2p1_param.dat

In this parameter file you have to set up all simulation parameters. Since this is a special case of the general program only the differences are listed:

- SimPar@TimeScheme: Choose between BE, CN (or FE); for stationary simulations this has to be BE.
- SimPar@TimeStep: any; for stationary this has to be 1d0.
- SimPar@MaxNumStep: Number of maximum time iterations; for stationary this has to be 1.
- SimPar@MatrixRenewal: Only M, K and S are active in this version; for Stokes K0.
- SimPar@FlowType: nonNewtonian is the only case that is working.
- SimPar@SteadyState: Choose between Yes or No; for stationary this has to be Yes and in SimPar@MatrixRenewal set M to 1.

- CCuvwp@NLmin: One possible choice is 1.
- CCuvwp@NLmax: Maximum of non-linear iteration in each time step; for Stokes this has to be 1.
- CCuvwp@Alpha: In general this is a value between 0d0 (Fixpoint) and 1d0 (Newton); for Stokes it is useful to take a big value (e.g. 11 which results in 1E-12 as the stopping)
- CCuvw@ValAdap: Insert two values for the adaptivity curve (1st greater than 1, 2nd lower than 0.9): Scaling factors for the adaptivity bewtween Fixpoint and Newton
- CCuvwp@Stopping: Relative stopping criterion for each time-step.
- CCuvwp@MGMinLev or MGMedLev: Set this as the value of SimPar@MaxMeshLevel and the linear equations are solved by MUMPS.
- CCuvwp@Vanka: Only 0 is working correctly.
- CCuvwp@BDF: Choose between 0 (for BE/CN or stationary), 2 (for BDF(2)) or 3 (for BDF(3)).
- Prop@PowerLawExp: Together with S in SimPar@MatrixRenewal the value 1.00d0 results in a Newtonian flow

2.2.3 Source files

q2p1_cc.f90

This is the main routine in which the application is configured: The application gets initialized, reads all parameters, runs the simulation and does some post-processing. You can also find the time loop in this file. The initialization for all time-schemes is added compared to the first initial version.

app_init.f90

It is called by the main routine, initializes the parallel structures, prepares the mesh (via decomposing the mesh to subdomains) and reads the simulation parameters SimPar. Here old solutions from older simulations can be read in, too.

assemblies_cc.f

Two important files are included in this file: On the one hand there is the subroutine CC_GetDefect_sub and on the other hand the subroutine CC_Extraction_sub. Both are called in the file q2p1_def_cc.f90 and will be explained in this file.
There is a subroutine which calculates the acting forces, too. But this is an old version because it is not of iso-parametric style.

iso_assemblies.f

In this file all the iso-parametric stuff is included: You can find iso-parametric assemblies of the matrices M, barM, K, S (and its further components), D (which is useless at the moment) and B. Additionally, there is the iso-parametric calculation of the forces (GetForceCyl_cc_iso). Here the acting forces are saved as follows: The first three entries are the components of the drag, the next three entries are the components of the lift and the last entry is the force in z-direction. The components of the drag and lift are sorted. First you have the complete force, then you see the velocity component and finally there is the pressure component of the acting force. The sum of the velocity and the pressure part is the complete force. If you compare the components of each force with a reference one the sum of the component errors can be greater thant the error of the complete one since the triangular inequality holds.

lin_transport_cc.f90

???

postprocessing.f90

It provides some helpful subroutines. Firstly, it writes the soultion to a vtk or gmv file. Secondly, the solver statistics are output at the end of the simulation. Finally, there are routines that are important for the dump files: Some files are necessary to read old dump files and some needs to be called to write dump-files. At the moment you can choose between prf and dmp files: prf needs a lot of memory and dmp outputs the solution of all ids to one file for velocity or pressure. The code is adjusted to use dmp files at the moment.

q2p1_cc_Umfpacksolver.f90

Some files are needed for factorization of the coarse and element matrices. Since MUMPS is set as the coarse grid solver the solver routins of UMFPACK can be useless.

q2p1_def_cc.f90

Creating the system matrices and working with these is one of the main tasks in this file. At the strat the system matrices are created: You have a big block matrix A for the non-linear equations and to compute the defect and a block matrix AA to perform the Preconditioner, i.e. $x^n = x^{n-1} + AA^{-1}(b - Ax^{n-1}$. The block matrices will only work with S (D is not working; but with S you can simulate Newtonian fluids as well.).
In the bottom of this file all matriy assemblies are called to create these using the iso-parametric concept.
In this file you can find some more basic routines that are important for the CC code: One of them calculates the defect (using CC_GetDefect_sub) and the next calculates the norm of this (non-linear) defect.
Of course, there are routines which are special for the CC version. There is a subroutine which creates special CC structures. These structures are used by CC_Extraction and Special_CC_Coarse. Without these two subroutines the code uses wrong matrices in the solver and you will get wrong results.
Ultimately the MG solver is initilaized and called in CC_mgSolve.

q2p1_mg_cc.f90

??

q2p1_transport_cc.f90

Before the main working subroutine is implemented some structures for the CC solver are initialized.
The main working/solving routine is Transport_q2p1_UxyzP_cc. The following sketch will show how this routins is working:

- Generate the adaptive function to have a background how to switch between Fixpoint and Newton method.
- Assemble the right hand side.
- Regarding time schemes set the correct scaling factor.

- Assemble the system matrices
- Factorize the element matrices.
- Compute the initial defect.
- Call the MG solver.
- Perform a full update of the solution
- Depending on the change of the defects the code decides if it can endure more Newton influence or less. The scaling is influenced by the adaptive function.
- Depending on the change of the defects and the use of Newton or Fixpoint the stopping criterion for MG is changed.
- Calculate the acting forces.
- Does the solution fulfill the non-linear stopping criterion?

In this file you can also find two subroutines which calculate the acting forces: On the one hand FAC_GetForces_CC_iso that is calling GetForceCyl_cc_iso and is the default decision; on the other hand there is a subroutine that calculates the forces using $Su + Bp$ (myFAC_GetForces).
The last important subroutine reads the CCuvwp parameters which are set in the param file.

q2p1_var_newton.f90

Additionally to QuadSc_var.f90 you can find a few new variables:

- Variables for the Newton-Preconditioner: barMXYmat, mg_barMXYmat, AAXY-mat and mg_AAXYmat,
- variables for the force calculation using myFAC_GetForces: BXMat_new, mg_-BXMat_new (Since the general B matrices are influenced by the Dirichlet BC there was a need to have B matrices without these BC to have correct forces.),
- a new type for the CC parameters used in the param.dat: TYPE tParamCC and
- additional variables for the BDF schemes

2.2.4 How to compile and run the code

The following comments compile the code and run it:

```
> cmake -DQ2P1_BUILD_ID=xeon-linux-intel-release -DBUILD_APPLICATIONS=True -DUSE_MUMPS=True ../Feat_FloWer/
> make -j4
> PyPartitioner.py 8 1 1 NEWFAC 'FILE.prj'
> mpirun -np 9 ./q2p1_cc
```

2.3 Appendix

TODO