Documentation

December 6, 2011

1 Introduction

The objective of this code is to provide a tool for quick linear stability studies in two dimension. It used existing codes for the discretization of the flow equations and for the stability calculations. FreeFem++ (`www.freefem.org/ff++`), a finite elements software, is used for the discretization of the non-linear and linearized equations. Most linear stability computations could be done from within FreeFem++, however the approach used here is to export sparse discretization matrices which can be used for stability analyses in python or Matlab.

2 Requirements

In addition to FreeFem++, using this tool requires either Matlab or python and the following packages.

numpy, scipy and matplotlib Numpy (>1.5, `www.numpy.scipy.org/`) and scipy (>0.9, `www.scipy.org/`) are required for sparse matrix computations. These provide iterative linear solvers and interfaces to direct linear solvers (SuperLU) and eigenvalue solvers (ARPACK). Matplotlib (>1.0,`http://matplotlib.sourceforge.net/`) is a plotting library which allows contour plot on unstructured meshes. On Linux, these libraries can easily be installed using pip. On Windows and Mac, this libraries can be found in the Enthought EPD package (`http://www.enthought.com/products/epd.php`).

HDF5, h5py h5py (`http://alfven.org/wp/hdf5-for-python/`) can be used for IO. It allows much faster IO that using text files, and allows to easily import the data in Matlab.

PETSc, SLEPc, mpi4py, petsc4py and slepc4py In order to have a choice between more solvers, petsc4py (>1.2,`http://code.google.com/p/petsc4py/`) and slepc4py (>1.2 `http://code.google.com/p/slepc4py/`) can be used[1]. In particular, significant gain in computation time have been observed using MUMPS as a linear solver and SLEPc's Krylov-Schur solver for eigenvalue computations. Using these packages, computations can be run in parallel provided mpi4py (`http://code.google.com/p/mpi4py/`) is installed.

[1]PETSc can for example be compiled using the following options:
```
--with-mpi=yes --with-shared-libraries --with-scalar-type=complex --with-fortran-interfaces=1 --CC=mpicc
--FC=mpif90 --FFLAGS=-I/usr/include --with-fortran --with-fortran-kernels=1 --with-clanguage=c COPTFLAGS=-O3
FOPTFLAGS=-O3 --download-mumps --download-scalapack --download-blacs --download-parmetis
```

3 Examples

Directory	Description
`step`	incompressible flow over a backwards facing step. Case considered e.g. by Marquet & Sipp (7th IUTAM symposium on Transition, 2009) for an optimal forcing study and by Barkley et al. (Journal of fluid mechanics, 2002) for a modal analysis
`sphere`	flow around a sphere. Case considered e.g. by Meliga et al. (Journal of fluids and strutures, 2009)

4 Discretization

4.1 Steady state computations

Steady states can be computed using standard P2-P1 discretization of the Navier–Stokes equations and a Newton method. In some cases, a simple initial guess is sufficient for the algorithm to converge (EXAMPLE). If this fails, or if finding an initial guess that satisfies the boundary conditions, the Reynolds number can be gradually increased from zero or a finite value. Several variational formulations can be found in `src/freefem/varf`, as described in table 1. After the steady state is computed, several files need to be saved in `dir`:

- The mesh should be saved using the `savemesh` function, can it can be loaded to generate the linear operator.
- The flow fields should also be saved using e.g.

  ```
  { ofstream file(outdir+"/ux0.dat");
    file << ux0[] << endl;
  }
  ```

- Some information about the mesh should also be exported so that data can be plotted from python / Matlab:

  ```
  { ofstream file(dir+"/base/coordinates.dat");
    for (int j=0;j<Th.nv; j++) {
      file << Th(j).x << " " << Th(j).y<< endl;}
  }
  { ofstream file(dir+"/base/connectivity.dat");
    int nbtriangle = Th.nt;
    for (int i=0;i<Th.nt; i++){
      file << Th[i][0]+1 << " " << Th[i][1]+1 << " " << Th[i][2]+1 << endl;}
  }
  ```

  In order to plot fields discretized using P2 elements, a mesh containing the midpoints is also needed: mesh Th2 = splitmesh(Th,2):

  ```
  mesh Th2 = splitmesh(Th,2);
  { ofstream file(dir+"/base/coordinates-2.dat");
    for (int j=0;j<Th2.nv; j++) {
      file << Th2(j).x << " " << Th2(j).y<< endl;}
  }
  ```

File	Description
`rect/varf.edp`	2D incompressible Navier–Stokes (cartesian coordinates)
`rect/varf3d.edp`	2.5D incompressible Navier–Stokes (cartesian coordinates). For non-linear computation, the flow is assumed to be parallel in the third dimension. For linear computation, a spanwize wave-number can be specified.
`axi/varf.edp`	incompressible Navier–Stokes (cylindrical coordinates). For non-linear computation, the flow is assumed to in independent on the azimuthal direction (NO SWIRL). For linear computation, an azimuthal wave-number can be specified.

Table 1: Variational formulations

```
{ ofstream file(dir+"/base/connectivity-2.dat");
  int nbtriangle = Th2.nt;
  for (int i=0;i<Th2.nt; i++){
    file << Th2[i][0]+1 << " " << Th2[i][1]+1 << " " << Th2[i][2]+1 << endl;}
}
```

4.2 Linear operators

The first step is to load the mesh and the results of the steady state computation using the same procedure used to save them. Information about the state space is also required, mostly for plotting purposes. Assuming there are 3 velocity components, the finite element space is defined as

```
fespace Vh(Th,[P2,P2,P2,P1]);
```

The field corresponding to each component of the state vector as well as the coordinate of the elements can be recovered using

```
Vh [iu1,iu2,iu3,iu4] = [0,1,2,3];
Vh [u1x,u2x,u3x,u4x] = [x,x,x,x];
Vh [u1y,u2y,u3y,u4y] = [y,y,y,y];
```

This data should be saved using

```
{ofstream file(dir+"/lin/dofs.dat");
  for (int j=0;j<iu1[].n; j++)
    file << iu1[][j] << " "<< u1x[][j] << " " << u1y[][j] << endl;
}
```

The base flow should also be exported using

```
Vh [u1b,u2b,u3b,u4b] = [ux0,ur0,0,p0];
{ofstream file(dir+"/lin/base.dat");
  for (int j=0;j<iu1[].n; j++)
    file << u1b[][j] << endl;
}
```

Matrices should be formed from the variational formulations (some are available, see table 1), using e.g. for the mass matrix

```
matrix<complex> B     = Mass(Vh,Vh,tgv=tgv);
{ofstream file(dir+"/lin/B.dat");
  file << B <<endl;
}
```

Matrices can be saved as real or complex. It is important that the linearized operator matrix is saved as `LNS.dat` and the mass matrix as `B.dat`. Boundary conditions should not be included in the matrices but provided separately on the form

```
varf bcvar([ux,ur,ut,p],[vx,vr,vt,q]) =
  on(1,ur=1,ut=1) + on(3,4,ut=1,ur=1,ux=1);
real[int] bcdir = bcvar(0,Vh);
}
{ofstream file(dir+"/lin/BC.dat");
  for (int j=0;j<bcdir.n; j++)
    file << bcdir[j] << endl;
}
```

5 Linear stability analyses

5.1 `FreeFEMDisc` object

After the base flow has been computed and the linear operators have been generated, all the data can be loaded into a python `FreeFEMDisc` object, defined in file `src/python/freefem.py` [2]:

```
import freefem as ff
ffdisc = ff.FreeFEMdisc(dir+'/lin/')
```

After matrices are read, components of the state vector that are not degrees of freedom (i.e. where Dirichlet boundary conditions are applied) are removed (the corresponding rows and columns are removed from the matrices). This allows in particular to avoid problems when dealing with the adjoint equations. The `FreeFEMDisc` object also contains information about the mesh, the fields etc. Detailed information can be found using in the documentation strings. Objects have 4 functions:

- `Save` saves the object using the pickle module. It can then be loaded back using `pickle.load`
- `SaveHDF5` save the object using h5py. This is more efficient than the previous solution both in terms if time and disk space. The `.h5` file can then be used to initialize an object.
- `PlotVar` creates contour plot for variables (real or complex, with only DOFs or all elements).
- `GetValue` performs interpolation to extract the value of a field at given points.

ALL THESE FUNCTIONS SHOULD BE RUN ON ONE SINGLE PROCESS

5.2 Use of PETSc - SLEPc

By default, sparse matrices are stored using scipy's CSR format. Functions to convert scipy sparse matrices (sequential) to PETSc matrices (distributed over all process), as well as numpy arrays to and from PETSC vectors, are available in `src/python/parfemstab.py`.

[2]Routines that read matrices from the disc are compiled in order to save time. This compilation with `f2py` (included in numpy) should be done automatically the first time the module is loaded

To do : For now the matrix is distributed naively. It should be interesting to use PARMETIS to do it better.

5.3 Time stepping

Time stepping of the linearized equations can easily be done using a backwards Euler method:

$$(B - \delta t L)\boldsymbol{q}^{n+1} = B\boldsymbol{q}^n$$

The linear system is solved using a direct LU solver: the decomposition of $B - \delta t L$ is computed once and used at each iteration. The `TimeStepping` object in `src/python/parfemstab.py` can be used to build a PETSc shell matrix that performs time stepping over a given time, using either Crank-Nicholson[3] or Euler as a temporal discretization scheme. The adjoint time stepping is also implemented.

To do : Do it with Matlab / python without PETSc

5.4 Optimal perturbations

The purpose of this type of studies is to determine the perturbations that are most amplified over a finite time interval T. Let $T = N\delta t$.

$$\boldsymbol{q}^N = \left\{(B - \delta t L)^{-1} B\right\}^N \boldsymbol{q}^0$$

Formally, the optimization reads

$$\begin{aligned}
\boldsymbol{q}_{opt} &= \arg\max_{\boldsymbol{q}_0} \frac{\|\boldsymbol{q}^N\|^2}{\|\boldsymbol{q}^0\|^2} \\
&= \arg\max_{\boldsymbol{q}_0} \frac{\langle \left\{(B - \delta t L)^{-1} B\right\}^N \boldsymbol{q}^0 | \left\{(B - \delta t L)^{-1} B\right\}^N \boldsymbol{q}^0 \rangle}{\langle \boldsymbol{q}^0 | \boldsymbol{q}^0 \rangle} \\
&= \arg\max_{\boldsymbol{q}_0} \frac{\boldsymbol{q}^{0\dagger} \left[\left\{(B - \delta t L)^{-1} B\right\}^N\right]^\dagger Q \left\{(B - \delta t L)^{-1} B\right\}^N \boldsymbol{q}^0}{\boldsymbol{q}^{0\dagger} Q \boldsymbol{q}^{0\dagger}}
\end{aligned}$$

For the incompressible Navier–Stokes equation, the initial pressure field is unimportant. Let $\widetilde{\boldsymbol{q}}$ be a state vector containing only velocity degrees of freedom, and P be the operator that turns $\widetilde{\boldsymbol{q}}$ into a state vector containing velocity and pressure DOFs (pressure being zero). Then

$$\boldsymbol{q}_{opt} = P \arg\max_{\boldsymbol{q}_0} \frac{\widetilde{\boldsymbol{q}^0}^\dagger P^\dagger \left[\left\{(B - \delta t L)^{-1} B\right\}^N\right]^\dagger Q \left\{(B - \delta t L)^{-1} B\right\}^N P\widetilde{\boldsymbol{q}^0}}{\widetilde{\boldsymbol{q}^0}^\dagger P^\dagger Q P \widetilde{\boldsymbol{q}^0}}$$

and $\widetilde{Q} = P^\dagger Q P$ is Hermitian positive definite. Let $\widetilde{Q} = M^\dagger M$ be its Cholesky decomposition.

$$\begin{aligned}
\boldsymbol{q}_{opt} &= PM^{-1} \arg\max_{\boldsymbol{r}_0} \frac{\widetilde{\boldsymbol{r}^0}^\dagger M^{-\dagger} P^\dagger \left[\left\{(B - \delta t L)^{-1} B\right\}^N\right]^\dagger Q \left\{(B - \delta t L)^{-1} B\right\}^N PM^{-1}\widetilde{\boldsymbol{r}^0}}{\widetilde{\boldsymbol{r}^0}^\dagger \widetilde{\boldsymbol{r}^0}} \\
&= PM^{-1}\widetilde{\boldsymbol{r}_{opt}}
\end{aligned}$$

[3] In order to ensure mass conservation, the first time step is performed using Euler's method. In this case, the first time step is half the next ones so that the same LU factorization can be used at each iteration.

In the definition of $\widetilde{\boldsymbol{r}_{opt}}$, one can see a Rayleigh quotient. $\widetilde{\boldsymbol{r}_{opt}}$ is therefore the leading eigenvector of

$$M^{-\dagger} P^{\dagger} \left[\left\{ (B - \delta t L)^{-1} B \right\}^{N} \right]^{\dagger} Q \left\{ (B - \delta t L)^{-1} B \right\}^{N} P M^{-1} \widetilde{\boldsymbol{r}_{opt}} = \lambda \widetilde{\boldsymbol{r}_{opt}}$$

and therefore

$$\begin{aligned} M^{-1} M^{-\dagger} P^{\dagger} \left[\left\{ (B - \delta t L)^{-1} B \right\}^{N} \right]^{\dagger} Q \left\{ (B - \delta t L)^{-1} B \right\}^{N} P M^{-1} \widetilde{\boldsymbol{r}_{opt}} &= \lambda M^{-1} \widetilde{\boldsymbol{r}_{opt}} \\ M^{-1} M^{-\dagger} P^{\dagger} \left[\left\{ (B - \delta t L)^{-1} B \right\}^{N} \right]^{\dagger} Q \left\{ (B - \delta t L)^{-1} B \right\}^{N} P \widetilde{\boldsymbol{q}_{opt}} &= \lambda \widetilde{\boldsymbol{q}_{opt}} \\ \underbrace{\widetilde{Q}^{-1} P^{\dagger} \left[\left\{ (B - \delta t L)^{-1} B \right\}^{N} \right]^{\dagger} Q \left\{ (B - \delta t L)^{-1} B \right\}^{N} P}_{\mathcal{L}_{o.p.}} \widetilde{\boldsymbol{q}_{opt}} &= \lambda \widetilde{\boldsymbol{q}_{opt}} \end{aligned}$$

In order to find the leading eigenvector of $\mathcal{L}_{o.p.}$ using and iterative solver, one only need to apply it to vectors. This only requires direct and adjoint time stepper, and a linear solver to apply $\widetilde{Q}^{-1}$. The latter can be done using a Cholesky decomposition. The `OptimalPerturbations` object in `src/python/parfemstab.py` can be used to build a PETSc shell matrix corresponding to $\mathcal{L}_{o.p.}$. Its leading eigenvalue can then be computed using function `OptimalPerturbationsSLEPc` in the same file. An example of the whole process can be found in `src/python/tgpar.py`.

To do : Do it with Matlab / python without PETSc

5.5 Eigenmodes

Eigenmodes are solutions of the evolution equation of the form $\boldsymbol{q}(x, y; t) = \widetilde{\boldsymbol{q}}(x, y) \exp(-\mathrm{i}\omega t)$. They are therefore solution of

$$-\mathrm{i}\omega B \widetilde{\boldsymbol{q}} = L \widetilde{\boldsymbol{q}}$$

i.e. they correspond to generalized eigenvectors of the pair (L, B). Computation of all eigenvectors is not feasible. The standard approach is to compute only a few eigenmodes, for example those associated with eigenvalues that lie closest to a shift parameter σ in the complex plane. For this, the simplest method is to use the shift-invert spectral transformation together with an iterative eigenvalue solver. Several possibilities are implemented:

- `DirectMode` in `src/python/femstab.py` uses ARPACK and superLU through scipy.
- `DirectModeSLEPc` in `src/python/femstab.py` uses on of SLEPc's iterative eigenvalue solvers (Krylov-Schur by default, possibly ARPACK) and one of the linear solvers installed in PETSc (direct or iterative, selected as a command-line argument). This approach is more effective, in particular concerning the convergence of the Krylov-Schur method compared to the IRAM.
- `DirectModeSLEPc` in `src/python/parfemstab,py` does the same thing in *parallel*. Note that PETSc's default linear solver is sequential.

These computations can be run from the scripts `src/python/modes.py` in sequential or `src/python/modespar.py` in parallel.

5.6 Optimal forcing

The purpose of this type of studies is to determine the harmonic forcing at a frequency ω that results in the largest time harmonic response. Let $\widetilde{\boldsymbol{f}}$ be a forcing vector such that the evolution equation reads

$$B\dot{\boldsymbol{q}} = L\boldsymbol{q} + B_2 P \widetilde{\boldsymbol{f}} \exp(-\mathrm{i}\omega t)$$

then the time harmonic response reads

$$\boldsymbol{q} = -(\mathrm{i}\omega B + \delta t L)^{-1} B_2 P \widetilde{\boldsymbol{f}}$$

The same analysis as in the case of optimal perturbations gives

$$\underbrace{\widetilde{Q}^{-1} P^{\dagger} B_2^{\dagger} (\mathrm{i}\omega B + \delta t L)^{-\dagger} Q (\mathrm{i}\omega B + \delta t L)^{-1} B_2 P}_{\mathcal{L}_{o.f.}} \widetilde{\boldsymbol{f}_{opt}} = \mu \widetilde{\boldsymbol{f}_{opt}}$$

This analysis is implemented in

- `FR` in `src/python/femstab.py` uses ARPACK and superLU through scipy.

These computations can be run from the scripts `src/python/fr.py` in sequential.

To do : DO it with PETSc in //