A (kind of) practical review on global stability approches

D. Fabre

IMFT, groupe InterfaceS

Informal Workshop "D&D", 10 juillet 2018

Linear global stability : basic principles, and a few useful tricks

Qu'es aquò ? ?

Three ideas to really speed up your linear stability computations !

StabFem : a software that may save your life !

Qu'es aquò ? ? ?

Demonstration for the wake of a cylinder

List of test-cases currently available

Nonlinear global stability approaches : status and future

Weakly nonlinear approach

Self-consistent approach

Harmonic-balance approach

Qu'es aquò ? ?

Instability problems are ubiquous in fluid mechanics

Numerical resolution of such problems resorts to a specific class of numerical methods, which replace time-stepping of the full equations by assumptions about temporal dependance (modal expansion, amplitude equations, etc...)

These methods are complementary to direct numerical resolution methods (i.e. time-stepping).

We refer to *global stability* when the geometry requires resolution in 2 (or 3) spatial dimensions.

(as opposed to *local stability* which takes advantage of invariance directions (parallel flow, etc...) to bring the problem to spatially 1D).

Base flow

We look for a steady base-flow $(\mathbf{u}_b; p_b)$ satisfying the steady Navier-Stokes equations, i.e. $NS(\mathbf{u}_b, p_b) = 0$.
Suppose that we have a 'guess' for the base flow $[\mathbf{u}_b^g, p_b^g]$ which almost satisfies the equations. We look for a better approximation under the form

$$[\mathbf{u}_b, p_b] = [\mathbf{u}_b^g, p_b^g] + [\delta \mathbf{u}_b, \delta p_b] = 0. \quad (1)$$

Injecting into the Navier-Stokes equation lead to

$$NS(\mathbf{u}_b^g, p_b^g) + NSL_{\mathbf{u}_b^g}(\delta \mathbf{u}_b, \delta p_b)$$

Where NSL is the linearised Navier-Stokes operator.
=> matricial problem with the form $A \cdot \delta X = Y$. The procedure of Newton iteration is to solve iteratively this set of equations up to convergence.

Linear stability

$$\mathbf{u} = \mathbf{u}_b + \epsilon \hat{\mathbf{u}} e^{\lambda t} \tag{2}$$

The eigenmodes is governed by the linear problem

$$\lambda \hat{\mathbf{u}} = NSL_{\mathbf{u}_b}(\hat{\mathbf{u}}, \hat{p})$$

,

After discretization we end up with an eigenvalue problem with the matricial form

$$\lambda B \hat{X} = A \hat{X} \tag{3}$$

Iterative method : single-mode shift-invert iteration

$$X^n = (A - \lambda_{shift} B)^{-1} B X^{n-1}$$

Generalization : Arnoldi

Adjoint problem

Define a scalar product :

$$\langle \phi_1, \phi_2 \rangle = \int_\Omega \overline{\phi_1} \cdot \phi_2 \; \mathrm{d}\Omega$$

We can first define the *adjoint linearised Navier-Stokes operator NSL*† defined by the property :

$$\begin{aligned} \forall(\mathbf{u}, p; \mathbf{u}^\dagger, p^\dagger), \quad & \left\langle NSL^\dagger_{\mathbf{U}}(\mathbf{u}^\dagger, p^\dagger), \mathbf{u} \right\rangle + \left\langle \nabla \cdot \mathbf{u}^\dagger, p \right\rangle \\ = \quad & \left\langle \mathbf{u}^\dagger, NSL_{\mathbf{U}}(\mathbf{u}, p) \right\rangle + \left\langle p^\dagger, \nabla \cdot \mathbf{u} \right\rangle . \end{aligned} \tag{4}$$

We can then define the adjoint eigenmodes as the solutions to the eigenvalue problem

$$\forall(\mathbf{u}, p), \quad \lambda^\dagger \left\langle \hat{\mathbf{u}^\dagger}, \mathbf{u} \right\rangle = \left\langle NSL^\dagger_{\mathbf{U}}(\hat{\mathbf{v}}, \hat{p^\dagger}), \mathbf{u} \right\rangle + \left\langle \nabla \cdot \hat{\mathbf{u}}^\dagger, p \right\rangle \tag{5}$$

Matricial form :

$$\overline{\lambda}^\dagger B \hat{X}\dagger = A^T \hat{X}\dagger. \tag{6}$$

Adjoint mode and structural sensitivity

Significance of the adjoint mode :
(optimal perturbation)

The adjoint eigenmode also allows us to introduce the so-called *structural sensitivity tensor* that is defined as

$$\mathbf{S}(\mathbf{x}) = \frac{||\hat{\mathbf{u}}^\dagger|| \; ||\hat{\mathbf{u}}||}{\langle \hat{\mathbf{u}}^\dagger, \hat{\mathbf{u}} \rangle}, \tag{7}$$

which has became popular in the recent years.

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !
- Adapt your mesh !

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !
- Adapt your mesh !
  - to base flow,

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !
- Adapt your mesh !
  - to base flow,
  - to base flow + eigenmode,

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !
- Adapt your mesh !
  - to base flow,
  - to base flow + eigenmode,
  - to base flow + structural sensitivity !

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !
- Adapt your mesh !
  - to base flow,
  - to base flow + eigenmode,
  - to base flow + structural sensitivity !
- Escape the real world : go into complex plane !

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !
- Adapt your mesh !
  - to base flow,
  - to base flow + eigenmode,
  - to base flow + structural sensitivity !
- Escape the real world : go into complex plane !
  - Incompressible case : to handle strongly convective instabilities (example : whisling jet, with Vincenzo Citro & Raffaele Longobardi)

Three ideas to really speed up your stability computations !

- Use simple shift-invert for computing a single mode !
- Adapt your mesh !
  - to base flow,
  - to base flow + eigenmode,
  - to base flow + structural sensitivity !
- Escape the real world : go into complex plane !
  - Incompressible case : to handle strongly convective instabilities (example : whisling jet, with Vincenzo Citro & Raffaele Longobardi)
  - Compressible case : to handle non-reflective boundary conditions
    (example : cylinder wake ; with Javier Sierra)

- Linear global stability : basic principles, and a few useful tricks
  - Qu'es aquò ? ?
  - Three ideas to really speed up your linear stability computations !
- StabFem : a software that may save your life !
  - Qu'es aquò ? ? ?
  - Demonstration for the wake of a cylinder
  - List of test-cases currently available
- Nonlinear global stability approaches : status and future
  - Weakly nonlinear approach
  - Self-consistent approach
  - Harmonic-balance approach

FreeFem++

Finite element methods are well suited to global stability problerms. The *FreeFem++* software is gaining popularity in the hydrodynamic stabillity community.

- ☺ Simple and intuitive syntax directly based on weak formulation.

FreeFem++

Finite element methods are well suited to global stability problerms. The *FreeFem++* software is gaining popularity in the hydrodynamic stabillity community.

- ☺ Simple and intuitive syntax directly based on weak formulation.
- ☺ Powerful and adaptative mesh (movemesh, adaptmesh, etc...)

FreeFem++

Finite element methods are well suited to global stability problerms. The *FreeFem++* software is gaining popularity in the hydrodynamic stabillity community.

- ☺ Simple and intuitive syntax directly based on weak formulation.
- ☺ Powerful and adaptative mesh (movemesh, adaptmesh, etc...)
- ☺ 2D and 3D.

FreeFem++

Finite element methods are well suited to global stability problerms. The *FreeFem++* software is gaining popularity in the hydrodynamic stabillity community.

- ☺ Simple and intuitive syntax directly based on weak formulation.
- ☺ Powerful and adaptative mesh (movemesh, adaptmesh, etc...)
- ☺ 2D and 3D.
- ☺ Large choice of sequencial and parallel solvers (Mumps, Petsc/Slepc, ..)

FreeFem++

Finite element methods are well suited to global stability problerms. The *FreeFem++* software is gaining popularity in the hydrodynamic stabillity community.

- ☺ Simple and intuitive syntax directly based on weak formulation.
- ☺ Powerful and adaptative mesh (movemesh, adaptmesh, etc...)
- ☺ 2D and 3D.
- ☺ Large choice of sequencial and parallel solvers (Mumps, Petsc/Slepc, ..)
- ☹ Limited graphical interface (ffglut)

FreeFem++

Finite element methods are well suited to global stability problerms. The *FreeFem++* software is gaining popularity in the hydrodynamic stabillity community.

- ☺ Simple and intuitive syntax directly based on weak formulation.
- ☺ Powerful and adaptative mesh (movemesh, adaptmesh, etc...)
- ☺ 2D and 3D.
- ☺ Large choice of sequencial and parallel solvers (Mumps, Petsc/Slepc, ..)
- ☹ Limited graphical interface (ffglut)
- ☹ Interpreted language : not suited to functional programming (but powerful macros).

FreeFem++

Finite element methods are well suited to global stability problerms. The *FreeFem++* software is gaining popularity in the hydrodynamic stabillity community.

- ☺ Simple and intuitive syntax directly based on weak formulation.
- ☺ Powerful and adaptative mesh (movemesh, adaptmesh, etc...)
- ☺ 2D and 3D.
- ☺ Large choice of sequencial and parallel solvers (Mumps, Petsc/Slepc, ..)
- ☹ Limited graphical interface (ffglut)
- ☹ Interpreted language : not suited to functional programming (but powerful macros).
- ☹ Syntax may be touchy and debugging sometimes awkward...

Why interface FreeFem++ with another software ?

- Previous strategy :

  Computation chain : freefem solvers / shell scripts / postprocessing with tecplot/gnuplot...

  => 50 almost identical programs + 10 ways to handle post-processing => getting crazy !

Why interface FreeFem++ with another software ?

- Previous strategy :

  Computation chain : freefem solvers / shell scripts / postprocessing with tecplot/gnuplot...

  => 50 almost identical programs + 10 ways to handle post-processing => getting crazy !

- => Necessity of a set of "drivers" in a high-level language to monitor computations and draw the results in "command-line" or "script" mode.

Why interface FreeFem++ with another software ?

- Previous strategy :

  Computation chain : freefem solvers / shell scripts / postprocessing with tecplot/gnuplot...

  => 50 almost identical programs + 10 ways to handle post-processing => getting crazy !

- => Necessity of a set of "drivers" in a high-level language to monitor computations and draw the results in "command-line" or "script" mode.

- Philosophy (objective) : one work (one paper) = 1 unique program to generate all results and produce all figures. (cf. Basilisk...)

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.
- Initially oriented towards Global Stability Approches (Linear & Nonlinear) but actually allowing a larger number of computations (DNS, linear acoustics, etc...)

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.
- Initially oriented towards Global Stability Approches (Linear & Nonlinear) but actually allowing a larger number of computations (DNS, linear acoustics, etc...)
- Multi-platform (Unix, MacOs, Windows) and designed to run on "light" computers (Laptops...)

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.
- Initially oriented towards Global Stability Approches (Linear & Nonlinear) but actually allowing a larger number of computations (DNS, linear acoustics, etc...)
- Multi-platform (Unix, MacOs, Windows) and designed to run on "light" computers (Laptops...)
- Easy to use/install,

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.
- Initially oriented towards Global Stability Approches (Linear & Nonlinear) but actually allowing a larger number of computations (DNS, linear acoustics, etc...)
- Multi-platform (Unix, MacOs, Windows) and designed to run on "light" computers (Laptops...)
- Easy to use/install,
- Easy to customize to a variety of cases (incompressible, compressible, fixed/free objects, free surfaces,...)

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.
- Initially oriented towards Global Stability Approches (Linear & Nonlinear) but actually allowing a larger number of computations (DNS, linear acoustics, etc...)
- Multi-platform (Unix, MacOs, Windows) and designed to run on "light" computers (Laptops...)
- Easy to use/install,
- Easy to customize to a variety of cases (incompressible, compressible, fixed/free objects, free surfaces,...)
- Freeware, based on two softwares : FreeFem++ and ~~Matlab~~/Octave

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.
- Initially oriented towards Global Stability Approches (Linear & Nonlinear) but actually allowing a larger number of computations (DNS, linear acoustics, etc...)
- Multi-platform (Unix, MacOs, Windows) and designed to run on "light" computers (Laptops...)
- Easy to use/install,
- Easy to customize to a variety of cases (incompressible, compressible, fixed/free objects, free surfaces,...)
- Freeware, based on two softwares : FreeFem++ and ~~Matlab~~/Octave
- Developed as a collaborative project (IMFT, Università di Salerno, ONERA, UPFL, ...)

StabFem : Cahier des charges

- StabFem : an open-source and easy-to-use software allowing a large range of computations in fluid mechanics.
- Designed for both research & education.
- Initially oriented towards Global Stability Approches (Linear & Nonlinear) but actually allowing a larger number of computations (DNS, linear acoustics, etc...)
- Multi-platform (Unix, MacOs, Windows) and designed to run on "light" computers (Laptops...)
- Easy to use/install,
- Easy to customize to a variety of cases (incompressible, compressible, fixed/free objects, free surfaces,...)
- Freeware, based on two softwares : FreeFem++ and ~~Matlab~~/Octave
- Developed as a collaborative project (IMFT, Università di Salerno, ONERA, UPFL, ...)
- Maintained on Github

https://github.com/erbafdavid/StabFem

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".
- Les solveurs et les drivers génériques sont dans des répertoires communs **SOURCES_MATLAB**/ et **SOURCES_FREEFEM**/,

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".
- Les solveurs et les drivers génériques sont dans des répertoires communs **SOURCES_MATLAB**/ et **SOURCES_FREEFEM**/,
- Les programmes spécifiques à chaque cas d'étude sont dans un répertoire spécifique.
  Exemple : le répertoire "CYLINDRE" contient les programmes suivants :

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".
- Les solveurs et les drivers génériques sont dans des répertoires communs **SOURCES_MATLAB**/ et **SOURCES_FREEFEM**/,
- Les programmes spécifiques à chaque cas d'étude sont dans un répertoire spécifique.
  Exemple : le répertoire "CYLINDRE" contient les programmes suivants :
  1. Mesh_Cylinder.edp -> Génération du maillage

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".
- Les solveurs et les drivers génériques sont dans des répertoires communs **SOURCES_MATLAB**/ et **SOURCES_FREEFEM**/,
- Les programmes spécifiques à chaque cas d'étude sont dans un répertoire spécifique.
  Exemple : le répertoire "CYLINDRE" contient les programmes suivants :
  1. Mesh_Cylinder.edp -> Génération du maillage
  2. Macros_StabFem.edp -> Macros case-dependant (conditions limites et post-traitement)

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".
- Les solveurs et les drivers génériques sont dans des répertoires communs **SOURCES_MATLAB**/ et **SOURCES_FREEFEM**/,
- Les programmes spécifiques à chaque cas d'étude sont dans un répertoire spécifique.
  Exemple : le répertoire "CYLINDRE" contient les programmes suivants :
  1. Mesh_Cylinder.edp -> Génération du maillage
  2. Macros_StabFem.edp -> Macros case-dependant (conditions limites et post-traitement)
  3. SCRIPT_CYLINDER.m -> Script "Maitre".

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".
- Les solveurs et les drivers génériques sont dans des répertoires communs **SOURCES_MATLAB**/ et **SOURCES_FREEFEM**/,
- Les programmes spécifiques à chaque cas d'étude sont dans un répertoire spécifique.
  Exemple : le répertoire "CYLINDRE" contient les programmes suivants :
  1. Mesh_Cylinder.edp -> Génération du maillage
  2. Macros_StabFem.edp -> Macros case-dependant (conditions limites et post-traitement)
  3. SCRIPT_CYLINDER.m -> Script "Maitre".

Articulation FreeFem / Matlab

- Etage 1 : Solveurs FreeFem++ "briques de base".
  Un solveur par "classe de problèmes" (2D incompressible, 2D compressible, Axisymétrique incompressible...) et par "type de calcul" (calcul d'un champ de base, stabilité linéaire, ...)
- Etage 2 : drivers Matlab "génériques"
  Un unique driver pour chaque "type de calcul" , le choix du bon solveur est fait en fonction des paramètres optionnels fournis au driver.
- Etage 3 : Les boucles sur les paramètres et la génération des figures sont faites dans un "script principal".
- Les solveurs et les drivers génériques sont dans des répertoires communs **SOURCES_MATLAB**/ et **SOURCES_FREEFEM**/,
- Les programmes spécifiques à chaque cas d'étude sont dans un répertoire spécifique.
  Exemple : le répertoire "CYLINDRE" contient les programmes suivants :
  1. Mesh_Cylinder.edp -> Génération du maillage
  2. Macros_StabFem.edp -> Macros case-dependant (conditions limites et post-traitement)
  3. SCRIPT_CYLINDER.m -> Script "Maitre".

Remarques : les programmes FreeFem doivent pouvoir être utilisés directement en dehors du driver StabFem, notamment pour faciliter le développement/débuggage...)

Les contributeurs "utilisateurs" (ex. étudiant M1/M2) ne travaillent qu'à l'étage 3 et ne devraient travailler que sur ces 3 fichiers.

Les contributeurs "développeurs" travaillent aux étages inférieurs.

Format d'échange des données

- Format de fichier d'échange ".ff2m", généré par FreeFem++ et relu par Matlab

Format d'échange des données

- Format de fichier d'échange ".ff2m", généré par FreeFem++ et relu par Matlab
- Example d'en-tête d'un fichier .ff2m :

```
1 ### Data generated by Freefem++ ;
2 Temperature
3 Format :
4 P1 T
```

Ligne 4 : "TypeField1 NameField1 TypeField2 NameField2... " TypeField can be "real" (scalar data), "real.N" (vectorial data), "P1" (data associated to mesh), ...

Format d'échange des données

- Format de fichier d'échange ".ff2m", généré par FreeFem++ et relu par Matlab
- Example d'en-tête d'un fichier .ff2m :

```
1 ### Data generated by Freefem++ ;
2 Temperature
3 Format :
4 P1 T
```

  Ligne 4 : "TypeField1 NameField1 TypeField2 NameField2... "
  TypeField can be "real" (scalar data), "real.N" (vectorial data), "P1" (data associated to mesh), ...

- Le fichier est lu et importé sous forme d'une *structure matlab*

Format d'échange des données

- Format de fichier d'échange ".ff2m", généré par FreeFem++ et relu par Matlab
- Example d'en-tête d'un fichier .ff2m :

```
1 ### Data generated by Freefem++ ;
2 Temperature
3 Format :
4 P1 T
```

  Ligne 4 : "TypeField1 NameField1 TypeField2 NameField2... " TypeField can be "real" (scalar data), "real.N" (vectorial data), "P1" (data associated to mesh), ...
- Le fichier est lu et importé sous forme d'une *structure matlab*
- Illustration : cas "EXAMPLE_Lshape"

First step : Generation of a mesh and "guess" base flow

`bf=SF_Init('Mesh_Cylinder.edp', [-40 80 40]);`

What the `SF_Init` driver does :

- Runs the relevant FreeFem++ program `Mesh_Cylinder.edp` with the corresponding parameters (here size of the domain),

  This program generates the following output files : `mesh.msh` (mesh data), `mesh.ff2m` (mesh information), `SF_Init.ff2m` (auxiliary information), `BaseFlow_init.txt` and `BaseFlow_init.ff2` ("guess" base flow).

First step : Generation of a mesh and "guess" base flow

`bf=SF_Init('Mesh_Cylinder.edp', [-40 80 40]);`

What the `SF_Init` driver does :

- Runs the relevant FreeFem++ program `Mesh_Cylinder.edp` with the corresponding parameters (here size of the domain),

  This program generates the following output files : `mesh.msh` (mesh data), `mesh.ff2m` (mesh information), `SF_Init.ff2m` (auxiliary information), `BaseFlow_init.txt` and `BaseFlow_init.ff2` ("guess" base flow).

- Reads all the output files,

First step : Generation of a mesh and "guess" base flow

`bf=SF_Init('Mesh_Cylinder.edp', [-40 80 40]);`

What the `SF_Init` driver does :

- Runs the relevant FreeFem++ program `Mesh_Cylinder.edp` with the corresponding parameters (here size of the domain),

  This program generates the following output files : `mesh.msh` (mesh data), `mesh.ff2m` (mesh information), `SF_Init.ff2m` (auxiliary information), `BaseFlow_init.txt` and `BaseFlow_init.ff2` ("guess" base flow).

- Reads all the output files,
- Returns a matlab "structure" object containing all the data needed for post-processing and subsequent usage.

Computation of a Base flow : principle

We look for a steady base-flow $(\mathbf{u}_b; p_b)$ satisfying the steady Navier-Stokes equations, i.e. $NS(\mathbf{u}_b, p_b) = 0$.
Suppose that we have a 'guess' for the base flow $[\mathbf{u}_b^g, p_b^g]$ which almost satisfies the equations. We look for a better approximation under the form

$$[\mathbf{u}_b, p_b] = [\mathbf{u}_b^g, p_b^g] + [\delta\mathbf{u}_b, \delta p_b] = 0. \qquad (8)$$

Injecting into the Navier-Stokes equation lead to

$$NS(\mathbf{u}_b^g, p_b^g) + NSL_{\mathbf{u}_b^g}(\delta\mathbf{u}_b, \delta p_b)$$

Where NSL is the linearised Navier-Stokes operator.
=> matricial problem with the form $A \cdot \delta X = Y$. The procedure of Newton iteration is to solve iteratively this set of equations up to convergence.

Computation of a Base flow : implementation

`bf=SF_BaseFlow(bf,'Re',10);`

What the `SF_Init` driver does :

- Copies the previous base flow into file `BaseFlow_guess.txt` which will be read by Freefem++,

Computation of a Base flow : implementation

`bf=SF_BaseFlow(bf,'Re',10);`

What the `SF_Init` driver does :

- Copies the previous base flow into file `BaseFlow_guess.txt` which will be read by Freefem++,
- Runs the relevant FreeFem++ solver (here `Newton_2D.edp`) with the corresponding parameters (here the value of *Re*),

Computation of a Base flow : implementation

`bf=SF_BaseFlow(bf,'Re',10);`

What the `SF_Init` driver does :

- Copies the previous base flow into file `BaseFlow_guess.txt` which will be read by Freefem++,
- Runs the relevant FreeFem++ solver (here `Newton_2D.edp`) with the corresponding parameters (here the value of *Re*),
- Reads all the generated output files (here `BaseFlow.ff2m`),

Computation of a Base flow : implementation

`bf=SF_BaseFlow(bf,'Re',10);`

What the `SF_Init` driver does :

- Copies the previous base flow into file `BaseFlow_guess.txt` which will be read by Freefem++,
- Runs the relevant FreeFem++ solver (here `Newton_2D.edp`) with the corresponding parameters (here the value of *Re*),
- Reads all the generated output files (here `BaseFlow.ff2m`),
- Returns a matlab "structure" object containing all the data needed for post-processing and subsequent usage.

Mesh adaptation

Linear stability

$$\mathbf{u} = \mathbf{u}_b + \epsilon \hat{\mathbf{u}} e^{\lambda t} \tag{9}$$

The eigenmodes is governed by the linear problem

$$\lambda \hat{\mathbf{u}} = NSL_{\mathbf{u}_b}(\hat{\mathbf{u}}, \hat{p})$$

,
After discretization we end up with an eigenvalue problem with the matricial form

$$\lambda B \hat{X} = A \hat{X} \tag{10}$$

Iterative method : single-mode shift-invert iteration

$$X^n = (A - \lambda_{shift} B)^{-1} B X^{n-1}$$

Generalization : Arnoldi

Eigenvalue computation : implementation

`SF_Stability(bf,'shift',0.04` + `0.74i,'nev',1,' type ' , 'D' ) ;`

What the `SF_Stability` driver does :

- Copies the base flow into file `BaseFlow.txt` which will be needed by Freefem++,

Eigenvalue computation : implementation

`SF_Stability(bf,'shift',0.04` + `0.74i,'nev',1,' type ' , 'D' ) ;`

What the `SF_Stability` driver does :

- Copies the base flow into file `BaseFlow.txt` which will be needed by Freefem++,
- Runs the FreeFem++ solver (here `Stab_2D.edp`) with the corresponding parameters (shift, number of eigenvalues, direct eigenmode),

Eigenvalue computation : implementation

`SF_Stability(bf,'shift',0.04` + `0.74i,'nev',1,' type ' , 'D' ) ;`

What the `SF_Stability` driver does :

- Copies the base flow into file `BaseFlow.txt` which will be needed by Freefem++,
- Runs the FreeFem++ solver (here `Stab_2D.edp`) with the corresponding parameters (shift, number of eigenvalues, direct eigenmode),
- Reads all the generated output files (here `Eigenmode.ff2m`),

Eigenvalue computation : implementation

`SF_Stability(bf,'shift',0.04` + `0.74i,'nev',1,' type ' , 'D' ) ;`

What the `SF_Stability` driver does :

- Copies the base flow into file `BaseFlow.txt` which will be needed by Freefem++,
- Runs the FreeFem++ solver (here `Stab_2D.edp`) with the corresponding parameters (shift, number of eigenvalues, direct eigenmode),
- Reads all the generated output files (here `Eigenmode.ff2m`),
- Does a number of post-processing (sort the eigenvalues, update the "shift" in continuation mode...)

Eigenvalue computation : implementation

`SF_Stability(bf,'shift',0.04` + `0.74i,'nev',1,' type ' , 'D' ) ;`

What the `SF_Stability` driver does :

- Copies the base flow into file `BaseFlow.txt` which will be needed by Freefem++,
- Runs the FreeFem++ solver (here `Stab_2D.edp`) with the corresponding parameters (shift, number of eigenvalues, direct eigenmode),
- Reads all the generated output files (here `Eigenmode.ff2m`),
- Does a number of post-processing (sort the eigenvalues, update the "shift" in continuation mode...)
- Returns a matlab "structure" object containing all the data needed for post-processing and subsequent usage.

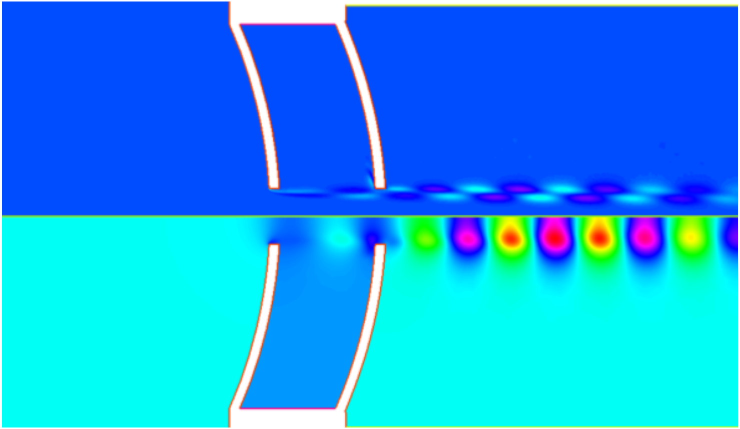
StabFem : list of test-cases currently available (or under development...)

- CYLINDER -> 2D incompressible, objet fixe
- CYLINDER_VIV -> 2D incompressible, objet mobile
  (Stage Diogo Ferrera-Sabino)
- IMPACTINGJET -> 2D incompressible, 3D stability.
  (with David LoJacono)
- CYLINDER_Compressible -> 2D compressible
  (Javier Serra, Vincenzo Citro...)
- BIRDCALL -> 2D-axisymmetric, incompressible or "augmented incompressible"
  (with R. Longobardi, V. Citro....)
- POROUS_DISK -> 2D-axisymmetric, with porous object
  (stage Adrien Rouvière)
- LiquidBridges -> 2D axi, with deformable free surface
  (stage Nabil Achour)
- ROTATING_POLYGONS
  (with Jérôme Mougel...)
- ....

=> Illustration dans le cas CYLINDER

Whistling jets : axisymmetric flow through a two-hole configuration

(with Raffaele Longobari, Vincenzo Citro & others...)



2D around a spring-mounted cylinder...

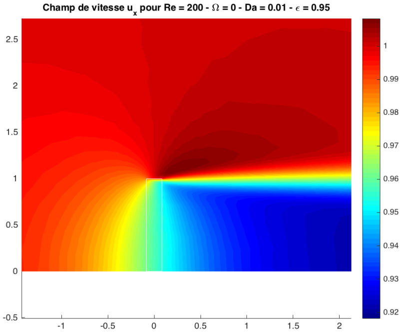
(with Diogo Ferreira Sabino & Olivier Marquet)

2D flow around a compressible cylinder...

(in fast progress with Javier Sierra...)

Flow around (and through) a porous (& rotating) disk...

(with Adrien Rouvière & D. Lo Jacono)



Liquid bridges...

Reference : Chireux et al., Phys. Fluids, 2015.

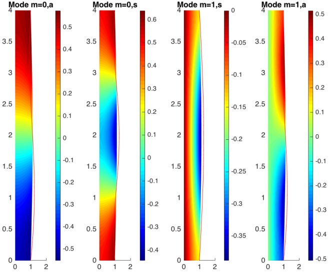
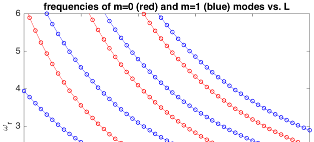
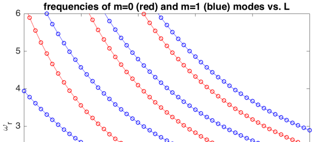


Figure – Oscillation modes of a liquid bridge of aspect ratio $L/R = 4$ and reduced volume $V^{=}$...





rotating polygons...

Reference : Mougel et al., JFM 2018

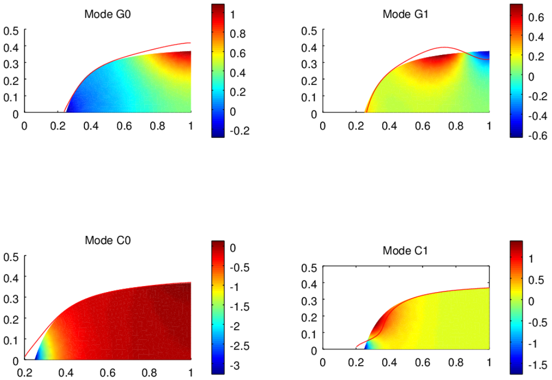
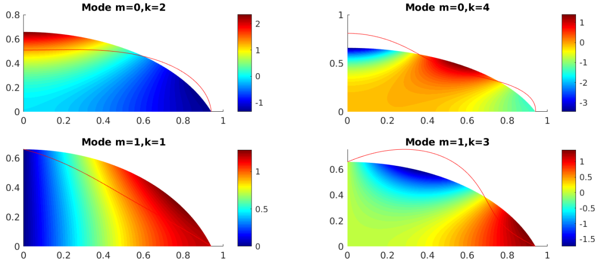


Figure – Oscillation modes of a potential vortex for $a = H/R = 0.3$ and $m = 3$ (figure 5, 6 of Mougel et al.).

Sessile drops...

With Nabil Achour (& Paul Bonnefis)



- Linear oscillations modes ("pined" or "fixed angle" conditions)
Next steps : investigate contact-line dynamics with WNL methods
(cf. Viola, Brun & Gallaire, 2018)

Linear global stability : basic principles, and a few useful tricks

- Qu'es aquò ? ?
- Three ideas to really speed up your linear stability computations !

StabFem : a software that may save your life !

- Qu'es aquò ? ? ?
- Demonstration for the wake of a cylinder
- List of test-cases currently available

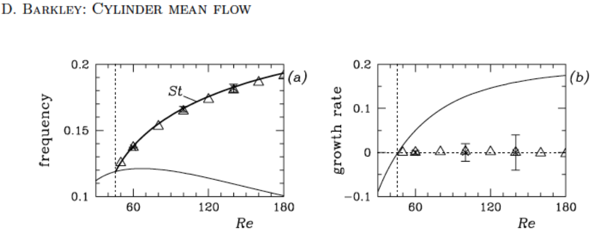
Nonlinear global stability approaches : status and future

- Weakly nonlinear approach
- Self-consistent approach
- Harmonic-balance approach

Nonlinear global stability approaches : review

The linear stability approach approach is the right tool to predict instability threshold (Re_c) and the shedding frequency at threshold ($St_c = \omega_c/2\pi$).

But for $Re > Re_c$ it badly predicts the frequency of the limit cycle.



It has been remarked that stability analysis of the *mean flow* obtained by time-averaging the limit cycle gives better predictions (Barkley, Leontini,...).

=> Objective of nonlinear stability approaches : provide rational approach to describe the nonlinear oscillation cycle, and provide amplitude equations to describe the transients.

Weakly nonlinear approach (Sipp & Lebedev, 2007)

Starting point : weakly non-linear expansion, with multiple scale method.

$$\epsilon = \frac{1}{Re_c} - \frac{1}{Re}; \quad \tau = \epsilon^2 t$$

$$\begin{aligned} \mathbf{u} &= \mathbf{u}_{bc} + \epsilon \left[A_{wnl}(\tau)\hat{\mathbf{u}} e^{i\omega_c t} + c.c. \right] && (11) \\ &+ \epsilon^2 \left[\mathbf{u}_\epsilon + |A_{wnl}|^2 \mathbf{u}_{2,0} + \left(A_{wnl}^2 \mathbf{u}_{2,2} e^{2i\omega_c t} + c.c. \right) \right] + \mathcal{O}(\epsilon^3) \end{aligned}$$

Resolution at order 2 :

$$\begin{aligned} \mathcal{LNS}_{\mathbf{u}_{bc}}(\mathbf{u}_\epsilon) - 2\nabla \cdot \mathrm{D}(\mathbf{u}_{bc}) &= 0, && (12) \\ \mathcal{LNS}_{\mathbf{u}_{bc}}(\mathbf{u}_{2,0}) &= \mathcal{C}(\hat{\mathbf{u}}, \overline{\hat{\mathbf{u}}}), && (13) \\ \mathcal{LNS}_{\mathbf{u}_{bc}}(\mathbf{u}_{2,2}) - 2i\omega_c \mathbf{u}_{2,2} &= \frac{1}{2}\mathcal{C}(\hat{\mathbf{u}}, \hat{\mathbf{u}}). && (14) \end{aligned}$$

Compatibility conditions at order 3 :

$$\frac{\partial A_{wnl}}{\partial \tau} = \Lambda A_{wnl} - (\nu_0 + \nu_2)|A_{wnl}|^2 A_{wnl}, \qquad (15)$$

$$\Lambda = -\frac{\left\langle \hat{\mathbf{u}}^\dagger, (\mathcal{C}(\mathbf{u}_\epsilon, \hat{\mathbf{u}}) + 2\nabla \cdot \mathrm{D}(\hat{\mathbf{u}})) \right\rangle}{\langle \hat{\mathbf{u}}^\dagger, \hat{\mathbf{u}} \rangle}, \qquad (16)$$

$$\nu_0 = \frac{\left\langle \hat{\mathbf{u}}^\dagger, \mathcal{C}(\mathbf{u}_{20}, \hat{\mathbf{u}}) \right\rangle}{\langle \hat{\mathbf{u}}^\dagger, \hat{\mathbf{u}} \rangle}, \quad \nu_2 = \frac{\left\langle \hat{\mathbf{u}}^\dagger, \mathcal{C}(\mathbf{u}_{22}, \overline{\hat{\mathbf{u}}}) \right\rangle}{\langle \hat{\mathbf{u}}^\dagger, \hat{\mathbf{u}} \rangle}., \qquad (17)$$

Self-Consistent approach (Mantic-Lugo, Arratia & Gallaire, 2014)

Starting point : Pseudo-eigenmode decomposition

$$\mathbf{u} = \mathbf{u}_m + A_{sc}\left[\tilde{\mathbf{u}}_1 e^{\sigma_{sc}t + i\omega_{sc}t} + \overline{\tilde{\mathbf{u}}_1} e^{\sigma_{sc}t - i\omega_{sc}t}\right], \quad \left(|\tilde{\mathbf{u}}_1|| = 1/\sqrt{2}\right) \tag{18}$$

where A_{sc} is an amplitude parameter, and $\lambda_{sc} = \sigma_{sc} + i\omega_{sc}$ is a pseudo-eigenvalue which depends upon the parameter A_{sc}.

Self-Consistent approach (Mantic-Lugo, Arratia & Gallaire, 2014)

Starting point : Pseudo-eigenmode decomposition

$$\mathbf{u} = \mathbf{u}_m + A_{sc}\left[\tilde{\mathbf{u}}_1 e^{\sigma_{sc}t + i\omega_{sc}t} + \overline{\tilde{\mathbf{u}}_1} e^{\sigma_{sc}t - i\omega_{sc}t}\right], \quad \left(|\tilde{\mathbf{u}}_1|| = 1/\sqrt{2}\right) \tag{18}$$

where A_{sc} is an amplitude parameter, and $\lambda_{sc} = \sigma_{sc} + i\omega_{sc}$ is a pseudo-eigenvalue which depends upon the parameter A_{sc}.
=> SC-model equations

$$\mathcal{NS}(\mathbf{u}_m) - A_{sc}^2 \mathcal{C}(\tilde{\mathbf{u}}_1, \overline{\tilde{\mathbf{u}}_1}) = 0, \tag{19a}$$

$$(\sigma_{sc} + i\omega_{sc})\tilde{\mathbf{u}}_1 = \mathcal{LNS}_{\mathbf{u}_m}(\tilde{\mathbf{u}}_1). \tag{19b}$$

Self-Consistent approach (Mantic-Lugo, Arratia & Gallaire, 2014)

Starting point : Pseudo-eigenmode decomposition

$$\mathbf{u} = \mathbf{u}_m + A_{sc}\left[\tilde{\mathbf{u}}_1 e^{\sigma_{sc}t + i\omega_{sc}t} + \overline{\tilde{\mathbf{u}}_1} e^{\sigma_{sc}t - i\omega_{sc}t}\right], \quad \left(|\tilde{\mathbf{u}}_1|| = 1/\sqrt{2}\right) \tag{18}$$

where A_{sc} is an amplitude parameter, and $\lambda_{sc} = \sigma_{sc} + i\omega_{sc}$ is a pseudo-eigenvalue which depends upon the parameter A_{sc}.

=> SC-model equations

$$\mathcal{NS}(\mathbf{u}_m) - A_{sc}^2 \mathcal{C}(\tilde{\mathbf{u}}_1, \overline{\tilde{\mathbf{u}}_1}) = 0, \tag{19a}$$

$$(\sigma_{sc} + i\omega_{sc})\tilde{\mathbf{u}}_1 = \mathcal{LNS}_{\mathbf{u}_m}(\tilde{\mathbf{u}}_1). \tag{19b}$$

=> Amplitude equation :

$$\frac{\partial A_{sc}}{\partial t} = \sigma_{sc}(A_{sc})A_{sc}$$

Self-Consistent approach (Mantic-Lugo, Arratia & Gallaire, 2014)

Starting point : Pseudo-eigenmode decomposition

$$\mathbf{u} = \mathbf{u}_m + A_{sc}\left[\tilde{\mathbf{u}}_1 e^{\sigma_{sc}t + i\omega_{sc}t} + \overline{\tilde{\mathbf{u}}_1} e^{\sigma_{sc}t - i\omega_{sc}t}\right], \quad \left(|\tilde{\mathbf{u}}_1|| = 1/\sqrt{2}\right) \tag{18}$$

where A_{sc} is an amplitude parameter, and $\lambda_{sc} = \sigma_{sc} + i\omega_{sc}$ is a pseudo-eigenvalue which depends upon the parameter A_{sc}.

=> SC-model equations

$$\mathcal{NS}(\mathbf{u}_m) - A_{sc}^2 \mathcal{C}(\tilde{\mathbf{u}}_1, \overline{\tilde{\mathbf{u}}_1}) = 0, \tag{19a}$$

$$(\sigma_{sc} + i\omega_{sc})\tilde{\mathbf{u}}_1 = \mathcal{LNS}_{\mathbf{u}_m}(\tilde{\mathbf{u}}_1). \tag{19b}$$

=> Amplitude equation :

$$\frac{\partial A_{sc}}{\partial t} = \sigma_{sc}(A_{sc})A_{sc}$$

Resolution method of (Mantic-Lugo 2014) : double iterative loop

- Inner loop :

Fix A_{sc}, iteratively solve (eigenvalue problem + calculation of mean flow) up to convergence for $(\sigma_{sc} + i\omega_{sc})$ as function of A_{sc}.

- Outer loop :

iterate over A_{sc} to reach $\sigma_{sc} = 0$

Self-Consistent approach : a direct resolution method

Let's forget about transients, and look directly for a description of the saturated cycle :

$$\mathbf{u} = \mathbf{u}_m + \mathbf{u}_{1,c} \cos(\omega t) + \mathbf{u}_{1,s} \sin(\omega t), \tag{20}$$

where $\mathbf{u}_{1,c}$ and $\mathbf{u}_{1,s}$ are two *real* fields
and ω is the (real) oscillation frequency of the limit cycle.

=> Equations :

$$\mathcal{N}\mathcal{S}(\mathbf{u}_m) = \frac{\mathcal{C}(\mathbf{u}_{1,c}, \mathbf{u}_{1,c}) + \mathcal{C}(\mathbf{u}_{1,s}, \mathbf{u}_{1,s})}{4}, \tag{21a}$$

$$\omega \mathbf{u}_{1,s} = \mathcal{L}\mathcal{N}\mathcal{S}_{\mathbf{u}_m}(\mathbf{u}_{1,c}), \tag{21b}$$

$$-\omega \mathbf{u}_{1,c} = \mathcal{L}\mathcal{N}\mathcal{S}_{\mathbf{u}_m}(\mathbf{u}_{1,s}). \tag{21c}$$

We need an extra scalar equation to fix the phase of the cycle, e.g :

$$\Im\{F_y(\mathbf{u}_1)\} = 0. \tag{22}$$

=> Direct resolution with Newton method !

Remarks :

- We need a good guess : use the WNL to produce it !
- Computation can be optimized using preconditioning (Ask Olivier...)

Harmonic-Balance

We start with the following expansion :

$$\mathbf{u} = \mathbf{u}_m + \mathbf{u}_{1,c}\cos(\omega t) + \mathbf{u}_{1,s}\sin(\omega t) + \mathbf{u}_{2,c}\cos(2\omega t) + \mathbf{u}_{2,s}\sin(2\omega t), \quad (23)$$

arriving to a system of equations :

$$\mathcal{NS}(\mathbf{u}_m) = \frac{\mathcal{C}(\mathbf{u}_{1,c},\mathbf{u}_{1,c}) + \mathcal{C}(\mathbf{u}_{1,s},\mathbf{u}_{1,s}) + \mathcal{C}(\mathbf{u}_{2,c},\mathbf{u}_{2,c}) + \mathcal{C}(\mathbf{u}_{2,s},\mathbf{u}_{2,s})}{4}, \quad (24a)$$

$$\omega\mathbf{u}_{1,s} = \mathcal{L}_{\mathbf{u}_m}(\mathbf{u}_{1,c}) - \frac{1}{2}\Big(\mathcal{C}(\mathbf{u}_{1,c},\mathbf{u}_{2,c}) + \mathcal{C}(\mathbf{u}_{1,s},\mathbf{u}_{2,s})\Big), \quad (24b)$$

$$-\omega\mathbf{u}_{1,c} = \mathcal{L}_{\mathbf{u}_m}(\mathbf{u}_{1,s}) - \frac{1}{2}\Big(\mathcal{C}(\mathbf{u}_{1,c},\mathbf{u}_{2,s}) - \mathcal{C}(\mathbf{u}_{1,s},\mathbf{u}_{2,c})\Big), \quad (24c)$$

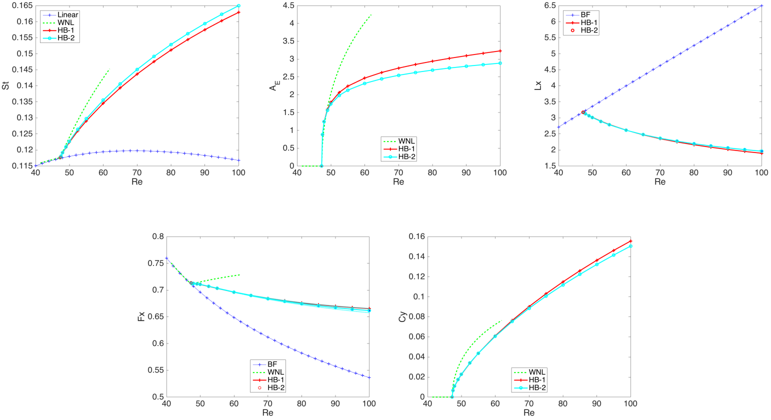
$$2\omega\mathbf{u}_{2,s} = \mathcal{L}_{\mathbf{u}_m}(\mathbf{u}_{2,c}) - \frac{1}{4}\Big(\mathcal{C}(\mathbf{u}_{1,c},\mathbf{u}_{1,c}) - \mathcal{C}(\mathbf{u}_{1,s},\mathbf{u}_{1,s})\Big), \quad (24d)$$

$$-2\omega\mathbf{u}_{2,s} = \mathcal{L}_{\mathbf{u}_m}(\mathbf{u}_{2,s}) - \frac{1}{2}\mathcal{C}(\mathbf{u}_{1,s},\mathbf{u}_{1,c}), \quad (24e)$$

$$\Im\{F_y(\mathbf{u}_1)\} = 0. \quad (25)$$

=> Direct Newton resolution again

Harmonic-Balance : results for the cylinder !



Conclusions

The future of StabFem

Recent progress

- Multi-platform objective : MacOs OK ; Unix OK ; Windows 10 currently 50 % compatible.
  main issues with windows : cp = copy,...
- Plotting options : recent intergration of "pdeplot2dff" from Markus "chloros" in place of pdeplot/pdetools .
  other solutions for plotting : tecplot converter, vtk converter, ...
- Compatibility with Octave : currently 50 % compatible.
  Main issues with octave : importdata, plotting (now solved), inputParser (now solved).
- Translation in Python ? ?

Besoins

- Maintaining a fully opensource (Matlab-Octave or Python ?) and fully multiplatform version (windows).
- Managing a list of test cases (non-regression tests, etc...)
- Help simplifying/rationalizing the programation style.
- Gestion of errors / debugging / "verbosity" ...
- Upgrading to 3D / parallel computation ? (currently not priority)
- Support with github (/ gitlab ?)
- Documentation
  Automatic generation from comments in programs ?
  Doxygen ? ?