iceFEM:
Open Source Package for Hydro-elasticity Problems

Balaje Kalyanaraman

Contents

1 Introduction

The package is intended for researchers aiming to solve Hydroelasticity problems using the finite element method. The principal idea behind the package is to use `FreeFem++` to solve the finite element problem and use `MATLAB` for visualization and other operations such as interpolation and cubic-spline constructions. It is necessary to have a basic `FreeFem++` installation to use this package and can be downloaded from the official website.

1.1 Installation

The package can be downloaded from `https://github.com/Balaje/iceFem`. The root directory contains the structure shown in Figure 1. The `include` folder contains a collection of `.idp` files which are `FreeFem++` scripts that contains pre-written functions and macros. To begin using the programs in the package, open the terminal and type the following.

```
iceFEM
├─ include
├─ modules
├─ Images
├─ DOCS
├─ examples
│  ├─ 1_Forced
│  ├─ 1_DirectFEM
│  ├─ 1_ThinPlate
│  └─ 1_EigenModes
└─ Meshes
   └─ BEDMAP2
```

Figure 1: Main package

```
export FF_INCLUDEPATH="$PWD/include"
```

This tells the `FreeFem++` compiler to add the `include` folder inside the package to the include path. Any new script could be added in the root directory. Then when writing scripts, the required `.idp` file can be imported by adding

```
include "macros.idp"
```

for example, to include the `macros.idp` file. In most cases, when using the predefined macros to solve the problem, adding `macros.idp` includes all the other `.idp` files in the package. When solving custom problems, individual `.idp` files can be included in the main program. Detailed decription of the functions available can be found in Section 4.

The `FF_INCLUDEPATH` variable must be set each time a new terminal session is started. One way to override this problem is to set the variable permanently by adding the line

```
export FF_INCLUDEPATH="/path/to/iceFem/include"
```

in `$HOME/.bashrc` or `$HOME/.bash_profile`. This ensures that the `FreeFem++` compiler locates the file each time a new terminal window is opened. Visualization can be done using `gnuplot` or the native plotter of `FreeFem++`.

A more convenient way to use the `FreeFem++` code is in conjunction with `MATLAB`. The `modules` folder consists of a set of `MATLAB` scripts that are used for visualization and validation of the `FreeFem++` code. This folder also contains routines that perform interpolation on certain quantities generated

(a) Displacement of the ice

(b) Velocity potential in the cavity

Figure 2: Output produced by the FreeFem++ code. The results are plotted using `FreeFem++`'s native plotting tool, `ffglut`. While useful for quick visualization, better results can be obtained by using `MATLAB` or `gnuplot`.

by the `FreeFem++` code. The `MATLAB` scripts present in the `examples` folder illustrate the use of the scripts.

1.2 A Quick Example

In this subsection, we describe the use of the `FreeFem++` code to solve a simple example. A set of reserved keywords used in the package are listed in Section 2. Once the `FF_INCLUDEPATH` is set, type

```
FreeFem++ -v 0 simple1.edp
```

in the command line. This solves the ice-shelf problem using linear elasticity for the ice combined with potential flow for the fluid and writes the solution as `eps` files and the outputs are shown in Figure 2. The program also displays the reflection coefficient R and its absolute value:

```
>> FreeFem++ -v 0 simple1.edp
 Reflection Coefficient = (0.4596963202350497,0.8880761753152261)
 |R| = 1.000000000000083
```

For the ice-shelf problems, $|R| = 1$ due to energy conservation and it can be used to check the solution. Optional parameters can be specified to modify the problem.

```
FreeFem++ -ne -v 0 simple1.edp -L [LENGTH]
                               -H [DEPTH OF OPEN OCEAN]
                               -h [THICKNESS OF ICE]
                               -N [MESH PARAM]
                               -Tr [REAL(period)]
                               -Ti [IMAG(period)]
                               -iter [SOL. INDEX]
                               -isUniIce [ON/OFF UNIFORM/NON UNIFORM ICE]
                               -isUniCav [ON/OFF UNIFORM/NON UNIFORM CAVITY]
                               -isForced [ON/OFF SHELF-FRONT FORCES]
```

where the `[.]` indicates the corresponding numerical value of the optional parameters. The length, thickness of the ice and the depth of the ocean is specified in meters (m). The wave-period is specified in seconds (s). The `ON/OFF` values are specified in binary, i.e., 0 or 1. The **`iter`** variable is used to number the solution which aids in interpolation and other batch manipulations. For example, the following commands:

```
>> FreeFem++ -ne -v 0 simple1.edp -L 10000 -H 800 -h 200 -N 4 -Tr 100 -Ti 0 -
   iter 0 -isUniIce 1 -isUniCav 1 -isForced 0

>> FreeFem++ -ne -v 0 simple1.edp -L 15000 -H 800 -h 200 -N 4 -Tr 200 -Ti 0 -
   iter 0 -isUniIce 1 -isUniCav 0 -isForced 0
```

produce the following outputs for the reflection coefficients.

```
Reflection Coefficient = (0.8507259058288464,0.525609582438974)
|R| = 0.9999999999999919

Reflection Coefficient = (-0.3166231272563836,0.9485514194213012)
|R| = 0.9999999999998886
```

1.3 MATLAB Interface

As mentioned earlier, **`MATLAB`** can be used to produce high quality graphics. To use **`MATLAB`** seamlessly with **`FreeFem++`**, one needs to follow the instructions below carefully:

1.3.1 Setting Up

The user must find the location of the FreeFem++ compiler. This can be done by running

```
which FreeFem++
```

in the command line. This produces an output like

```
/usr/local/ff++/openmpi-2.1/3.61-1/bin/FreeFem++
```

By default, the package comes with an initialization script called `CreatePaths.m` that tells the MATLAB compiler, the installation location of `FreeFem++` in the computer and also sets the `FF_INCLUDEPATH` variable for the current `MATLAB` session. The file also defines a set of variables that will be used to generate the plots.

```
%% Filename: CreatePaths.m

function CreatePaths
clc
close all

fprintf('Run:\n\nwhich FreeFem++\n\nin your command line to get the path
    for FreeFem++.\n Set the full path in the variable `ff` in CreatePaths.
    m\n');
addpath([pwd,'/modules/']);
set(0,'defaultLegendInterpreter','latex');
set(0,'defaulttextInterpreter','latex');
set(0,'defaultaxesfontsize',20);
envvar = [pwd,'/include'];
setenv('FF_INCLUDEPATH',envvar);

%% Should be set manually by the user.
global ff
ff='/usr/local/ff++/openmpi-2.1/3.61-1/bin/FreeFem++';
```

Once the compiler location is obtained, the user must add the **full path** in the `global ff` variable as shown in the code block above. Then the script `CreatePaths.m` should be run to set the global variable for the session.

1.3.2 Running FreeFem++ in MATLAB

First the function script `getProperties.m` can be used to get the default physical properties of ice and water. The usage is as follows

```
[L, H, th, d, E, nu, rhow, rhoi, g, Ad] = getProperties();
```

where

```
L: Length of the ice.           H: Depth of the open ocean.
th: Thickness of the shelf.     d: Submergence of the ice.
E: Young`s modulus.             nu: Poisson`s ratio.
rhow: Density of Water.         rhoi: Density of Ice.
g: Gravity Acceleration.        Ad: Amplitude of the wave.
```

To override certain parameters, use ~ at the desired entry. For example:

```
%% Get the Parameters of the ice.
[~,~,~,~,E,nu,rhow,rhoi,g,~] = getProperties();
H = 800;
L = 20000;
omega = 2*pi/(300); % Wave Period can be complex.
T = 2*pi/omega;
th = 200;
d = (rhoi/rhow)*th;
```

The most intuitive way to run the **FreeFem++** code is to call the script as an external program from **MATLAB**. The best way to do it is to use the following code block (with appropriate modifications).

```
%% Run the FreeFem++ code;
global ff
file = 'simple1.edp';
ffpp=[ff,' -nw -ne ', file];
cmd=[ffpp,' -Tr ',num2str(real(T)),' -Ti ',num2str(imag(T)),' -H ',num2str(H),
    ' -L ',num2str(L),' -h ' ,num2str(th),' -N ',num2str(3), ' -isUniIce ',
    num2str(0), ' -isUniCav ',num2str(0)];
[aa,bb1]=system(cmd);
if(aa)
    error('Cannot run program. Check path of FF++ or install it');
end
```

The global variable `ff` contains the full path to the **FreeFem++** compiler. If an error occurs in running the **FreeFem++** code, the variable `bb1` can be printed out to check the error. If the code runs successfully, then `aa=0` and the error message is not printed. The `num2str` function converts the numerical values to string and appends them to the full command, which is then passed to the `system` function.

1.3.3 Visualization

We use `pdeplot` command to plot the mesh. The macro `writeToMATLAB` in `include/macros.idp` contains a code snippet to write the **FreeFem++** data. In the **FreeFem++** code, call

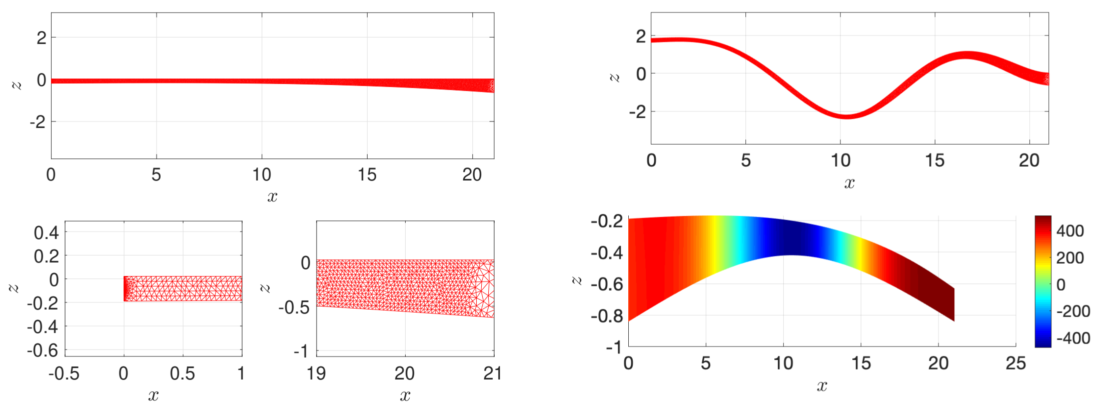
```
writeToMATLAB(uh, Th, solfilename, meshfilename);
```

where `uh` denotes the finite element solution, `Th` denotes the finite element mesh and `solfilename, meshfilename` denotes string variables which contains the name of the solution file and the mesh file respectively.

The **MATLAB**functions that will be used here are

```
[pts,seg,tri] = importfilemesh(filename);
uh = importfiledata(filename);
```

The variable `pts` is a $2 \times N$ array containing the $x-$ and $y-$ coordinate of the points. The variables



(a) Finite element mesh

(b) Displacement and Velocity Potential.

Figure 3: Figure showing the plots generated by the **MATLAB** script.

[seg,tri] stores the mesh connectivity information which will be used by **pdeplot**. To plot the mesh in **MATLAB**, we write

```
figure;
pdeplot(pts,seg,tri);
axis equal
grid on
```

and to plot a finite element function in **MATLAB**, we write

```
figure;
pdeplot(pts,seg,tri,'XYData',real(uh)','colormap','jet');
axis equal
grid on
```

An example **MATLAB** script demonstrating the interface has been added in the package named **eg1.m** and the results are shown in Figure 3. As we observe, the visualization is precise using **MATLAB** than the native plotter, whose functionalities are limited in comparison. For generating high-quality PDF plots, it is recommended to use the **export_fig** package which can be found in `https://github.com/altmany/export_fig`.

2 Macros and Keywords

2.1 Keywords

The iceFEM package consists of a few reserved keywords that can be changed within any program. The package also consists of a list of macros that can be used to solve certain Hydroelasticity problems. They can be found in the script file **macros.idp**. A list of currenlty available reserved keywords are discussed below.

- **`isUniIce, isUniCav`**
  Data Type: `bool`
  Variable to switch between of uniform/non-uniform profiles. `Can be changed. Set to default as true.`

- **`NModes`**
  Data Type: `int`
  Sets the number of modes in the modal expansion in the open-ocean solution. This is used in the construction of the non-local boundary condition at the ocean/cavity interface. `Set to default as 3.`

- **`nev`**
  Data Type: `int`
  Sets the number of in-vacuo modes of vibration of the ice-shelf. This is also the dimension of the reduced system obtained in the final step. `Set to default as 20.`

- **`iter`**
  Data Type: `int`
  A variable used to index the solution for batch operations in **`MATLAB`** such as interpolation. `Default set to 0.`

- **`Lc, tc`**
  Data Type: `real`
  The characteristic length and time computed for non-dimensionalization. `Do not modify.`

- **`omega`**
  Data Type: `complex`
  The incident frequency computed from the wave period.

- **`rhoi, rhow, ag, densRat`**
  Data Type: `real`
  The denisities of ice and water, the acceleration due to gravity g and the ratio of densities ρ_i/ρ_w, respectively. Obtained from the **`getProperties`** function. `Do not modify.`

- **`LL, HH, dd, tth`**
  `lambdahat, muhat`
  `gammahat, deltahat`
  Data Type: `real`
  The non-dimensional values length of the ice-shelf, cavity-depth, submergence, shelf-thickness.
  The ratio λ/L_c^2 and μ/L_c^2 where λ, μ are the Lamé parameters.
  The ratio ρ_w/L_c and $\rho_w g/L_c$.
  `Do not modify.`

  `tt`
  Data Type: `complex`
  Value of the incident wave period, **`Tr + 1i*Ti`**. Computed from the user-input values of **`-L, -H, -h, -Tr, -Ti`**.
  `Do not modify.`

- **Ap**
  Data Type: `complex`
  The amplitude of the incident velocity potential. Computed from the incident wave period. `Do not modify.`

- **ThIce, ThCavity**
  Data Type: `mesh`
  The variables containing the mesh data for the ice-shelf and the sub-shelf cavity, respectively. `Can be modified to any valid mesh file. See the FreeFem++ manual for more details.`

- **Wh, Vh, Xh**
  Data Type: `fespace`
  `P1` finite element spaces for the cavity (W_h) and the ice-shelf (V_h, X_h). The space X_h is a vectorial finite element space. `Depends on the finite element mesh. Modified if the mesh is modified.`

- **k, kd**
  Data Type: `complex[int]`
  Complex 1D arrays containing the wave-numbers obtained after solving the free-surface dispersion equation with depths H and $H - d$, respectively.
  The length of the arrays is **NModes+1** and is computed by the **dispersionfreesurface** function.

- **fh**
  Data Type: `func`
  An external function that is used to specify the non-homogeneous part of the Dirichelt/Neumann boundary condition. Should be specified in the program.

- **STIMA, BMASSMA**
  Data Type: `matrix<complex>`
  Contains the stiffness matrix on the cavity mesh, boundary mass matrix on the ocean-cavity interface. The quantities are computed by `macro` **getLaplaceMat()**.

- **B, K, AB, Hmat**
  Data Type: `complex[int,int]`
  Complex 2D arrays that are the components of the final reduced system.
  F
  Data Type: `complex[int]`
  Right hand side of the reduced system.

  The quantities are computed by `macro` **buildReducedSystem()**. `Do not modify.`

- **mu**
  Data Type: `real[int]`
  Variable to store the eigenvalues of the in-vacuo Euler Bernoulli problem. Generated by `macro` **solveEigenEB()**.

2.2 Macros

In this subsection, we list a set of predefined macros that can be used when writing a program. The usage of the macros will be discussed in the Tutorial section.

- `macro setProblem()`
- `macro solveDispersion()`
- `macro setMeshIce()`
- `macro setMeshCav()`
- `macro solveEigen()`
- `macro writeEigen()`
- `macro getLaplaceMat(a,b)`
- `macro getLaplaceMatEB(m,rad)`
- `macro getLaplaceMatDBC(m,rad)`
- `macro buildReducedSystem(VX,VY,phi0,phij,c0,cc,isForcedFront)`
- `macro buildReducedSystemEB(mu,phi0,phij,alpha,beta,gamma)`
- `macro solveReducedSystem()`
- `macro constructEBdisp()`
- `macro writeToMATLAB(uh, Th, solfilename, meshfilename)`

3 Tutorial

In this section, we describe how to solve two ice-shelf problems using the iceFEM package. The first example is the ice-shelf modelled using the Euler Bernoulli beam theory. The second example is when the ice is modelled using the 2D elasticity equations under plane strain conditions.

3.1 Euler-Bernoulli beam

In this subsection, we solve the example in [1]. The complete example is provided in the package as **`simple3.edp`**. First, we invoke the necessary modules by importing **`macros.idp`**.

```
verbosity=0.; //Sets the level of output.
include "macros.idp"
```

Next we set the problem by specifying the number of modes in the series expansions and solve the Dispersion equation to obtain the wave numbers.

```
nev=20; //Number of in-vacuo modes
NModes=5; //Number of open-ocean modes

//Set the problem.
setProblem;

//Solve the Dispersion equation to obtain the wave number arrays k, kd
solveDispersion;
```

The next step is to build the mesh for the cavity. To specify the shelf/cavity interface, we first build a uniform mesh for the ice-shelf. However, this mesh will not be used to compute the solution.

```
isUniformIce=true;//Force uniform mesh for the ice to obtain the shelf-cavity
    interface.
setMeshIce(0,0,0);

isUniformCav=false;
//  A three point cubic spline is used: Args. (midX, midY, endY)
setMeshCav(LL/2., -0.5*HH, -HH);
```

If `isUniformIce/isUniformCav=true`, any option given as arguments will be overridden to default values. The next step is to solve the eigenvalue problem to obtain the in-vacuo modes of the ice-shelf. This is done by simply calling,

```
solveEigenEB; //Solves the Eigenvalue problem to obtain the in-vaco EB modes.
```

The next step is to obtain the non-local boundary condition which is of the form:

$$\partial_x \phi = \underbrace{\mathrm{Q}\phi}_{:\text{Matrix}} + \underbrace{\chi}_{:\text{Vector}}.$$

Two functions are available in the `nonLocal.idp` file to calculate the necessary matrix and vector. The following code block is used to obtain the boundary condition. For more details, see [1].

```
//Matrix MQ stores the Q-operator
matrix<complex> MQ;
//ctilde stores the vector corresponding to the incident wave
complex[int] ctilde(NModes+1);

//Obtain the matrix and vector.
MQ=getQphi(ThCavity,NModes,k,kd,HH,dd,Ap,4);
ctilde=getChi(ThCavity,NModes,k,kd,HH,dd,Ap);

//Obtain the function by combining the modes.
Wh<complex> chi1;
for(int m=0; m<NModes+1; m++)
  chi1 = chi1+ctilde[m]*cos(kd[m]*(y+HH))/cos(kd[m]*(HH-dd));
```

The next step is to obtain the diffraction potential in the sub-shelf cavity.

```
// Solve for the diffraction potential
Wh<complex> phi0; //Declare a complex FE function the cavity region.
//Set the external function.
func fh=chi1;

//Call the routine to compute the FE matrices for the problem.
getLaplaceMatEB(0,0); //Indicates the routine to solve for the Diffraction
    potential.

//Set the LHS matrix and solve the problem.
LHS=STIMA+(MQ); //Add the Q-matrix.
set(LHS,solver=sparsesolver);
phih[]=LHS^-1*RHS[];
phi0=phih;

//Plot the result.
plot(phi0,wait=1,fill=1,value=1);
```

Similarly, the radiation potentials can be obtained by

```
//4) Solve for the radiation potential
Wh<complex>[int] phij(nev);
for(int m=0; m<nev; m++)
  {
    fh=0; //0 for radiation potential.
    getLaplaceMatEB(m,1); //Indicates the routine to input the mth mode of
        vibration.
    LHS=STIMA+(MQ);
    set(LHS,solver=sparsesolver);
    phih[]=LHS^-1*RHS[];
    phij[m]=phih;
  }
```

The next step is to build the reduced system which will be solved to obtain the final solution. The mathematics can be found in the Appendix.

Build the Appendix.

```
//Parameters for the system.
complex ndOmega=2*pi/tt;
complex alpha = HH*ndOmega^2;
real beta = 1;
real gamma = densRat*tth;

//Call the routine to construct the reduced system
buildReducedSystemEB(mu, phi0, phij, alpha, beta, gamma);
```

Finally we solve the reduced system to obtain the modal contributions. The final solution is the linear combinatinon of these coefficients with the corresponding bases for the displacement and potential.

```
//Solve the reduced system.
complex[int] xi(nev);
solveReducedSystem; //The solution is stored in xi
```

The solution can be visualized using any of the means discussed in Section 1. Further, if the user wants to compute the reflection coefficients, a function **getRefCoeff** is available in the module **refCoeff.idp**. The following code block computes the reflection coefficient for the problem.

```
complex[int] phiVec(phi.n), c(NModes+1);
phiVec=phi[]; //Convert the FE solution to an array.

//Call the function to compute the reflection coefficient.
complex Ref = getRefCoeff(ThCavity, NModes, kd, k, phiVec, HH, dd, Ap, c);

//Print the value.
cout.precision(16);
cout<<"Reflection Coefficient = "<<Ref<<endl<<"|R| = "<<abs(Ref)<<endl;
```

This concludes the first tutorial. In the next tutorial, we will discuss the second model, where the ice-shelf is modelled using 2D linear elasticity equations under plane strain assumptions.

3.2 2D Linear Elasticity

In this subsection, the same problem can be solved using 2D linear elasticity for the ice-shelf. The code follows along the same line except for slight modifications. The full code can be found below.

```
verbosity=0.;
include "macros.idp"

//Sets up an example problem. Can control input using CMD line args
nev=20;
setProblem;

//Solve the dispersion equation -k tan(k h) = \alpha. -k tan(k (h-d)) = \alpha
solveDispersion;

//Build the meshes.
real botRight=-3.*tth, midPX=3.7*LL/4, midPY=-2.5*tth;
setMeshIce(botRight, midPX, midPY);
real midx=LL/2., midy=-0.5*HH, endy=-HH;
setMeshCav(midx, midy, endy);


//  1) Solve the in-vacuo eigenvalue problem.
Xh[int][VX,VY](nev); //Define an array of fe-function to store in-vacuo modes.
real[int] ev(nev); //Define a real array for the eigenvalues.
solveEigen;

```

```

//  2) Get the Non-local boundary condition
matrix<complex> MQ;
complex[int] ctilde(NModes+1);
MQ=getQphi(ThCavity,NModes,k,kd,HH,dd,Ap,4);
ctilde=getChi(ThCavity,NModes,k,kd,HH,dd,Ap);
Wh<complex> chi1;
for(int m=0; m<NModes+1; m++)
  chi1 = chi1+ctilde[m]*cos(kd[m]*(y+HH))/cos(kd[m]*(HH-dd));

//  3) Solve for the diffraction potential.
Wh<complex> phi0;
func fh=chi1;//Store in fh, the right-hand side function on the ocean-cavity
    interface.
getLaplaceMat(0,0);
LHS=STIMA+(MQ);
set(LHS,solver=sparsesolver);
phih[]=LHS^-1*RHS[];
phi0=phih;//Store in phi0;


//  4) Solve for radiation potential.
Wh<complex>[int] phij(nev);
for(int m=0; m<nev; m++)
  {
    func fh=0;
    getLaplaceMat(VX[m],VY[m]);
    LHS=STIMA+(MQ);
    set(LHS,solver=sparsesolver);
    phih[]=LHS^-1*RHS[];
    phij[m]=phih;
  }

//Build the reduced system and solve it.
complex[int] c0(NModes+1);
complex[int,int] cc(NModes+1,nev);
c0=0.; cc=0.;
buildReducedSystem(VX,VY,phi0,phij,c0,cc,0.);
complex[int] xi(nev);
solveReducedSystem;

//Compute the solution.
Vh <complex> etax, etay;
Wh<complex> phi;
phi = phi0;
for(int m=0; m<nev; m++)
  {
    phi = phi + xi[m]*phij[m];
    etax = etax + xi[m]*VX[m];
    etay = etay + xi[m]*VY[m];
```

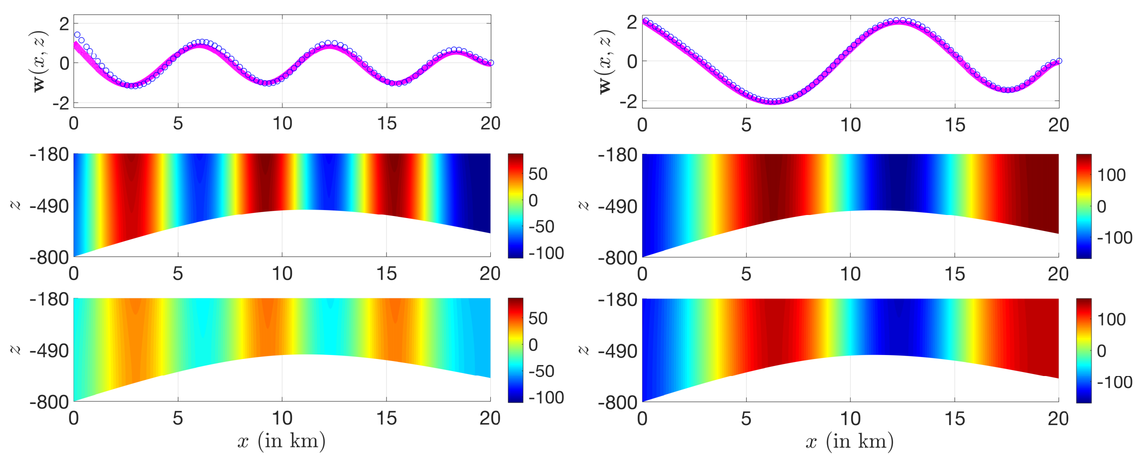
```
  }

//Compute the reflection coefficient.
complex[int] phiVec(phi.n), c(NModes+1);
phiVec = phi[]; //Get the vector form of the finite element function.
complex Ref = getRefCoeff(ThCavity, NModes, kd, k, phiVec, HH, dd, Ap, c); // From "refCoeff.idp"
cout.precision(16);
cout<<"Reflection Coefficient = "<<Ref<<endl<<"|R| = "<<abs(Ref)<<endl;

mesh ThNewIce=movemesh(ThIce,[x+real(etax),y+real(etay)]);
plot(ThNewIce,wait=1,ps="deformedMesh.eps");
Wh rphi=real(phi);
plot(rphi,wait=1,fill=1,value=1,ps="velocity.eps");

//Write data to MATLAB
//iter could be used to index the solution.
Vh ux=real(etax), uy=real(etay);
writeToMATLAB(ux,ThIce,"xDisp"+iter+".bb","meshIce"+iter+".msh")
writeToMATLAB(uy,ThIce,"yDisp"+iter+".bb","meshIce"+iter+".msh");
writeToMATLAB(rphi,ThCavity,"potentialCav"+iter+".bb","meshCav"+iter+".msh");
```

As guessed, the linear elasticity solution coincides with the Euler Bernoulli solution for thin ice-shelves. The thinness is determined with respect to the incident wavelength that the ice-shelf is subject to. Figure 4 shows the two solutions for a uniform ice-shelf of length 20 km subject to two different incident wave-forcing.



(a) $T = 80\,\mathrm{s}$

(b) $T = 200\,\mathrm{s}$

Figure 4: Comparison results for the ice-shelf vibration for two different wave-periods. The thinness of the ice-shelf is determined with respect to the incident wavelengths. The longer the incident wave (higher T), the better the agreement is, since the front thickness is negligibly small compared to long wavelengths. Hence more discrepancy can be observed for $T = 80\,\mathrm{s}$ case.

4 The `include` folder

References

[1] M. Ilyas, M. H. Meylan, B. Lamichhane, and L. G. Bennetts. Time-domain and modal response of ice shelves to wave forcing using the finite element method. *J. Fluids Struct.*, 80:113–131, 2018.