iceFEM:
Open Source Package for Hydro-elasticity Problems

Balaje Kalyanaraman, Michael Meylan, Luke Bennetts, Bishnu Lamichhane

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.

```
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 description of the functions available can be found in Section 2.1. 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++**. Newer versions of **FreeFem++** contains a routine to perform visualizations using ParaView.

A more convenient way to use the **FreeFem++** module 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 by the **FreeFem++** code. The list of functions available are listed below.

```
colscheme.m #For complex plot.

dispersion_elastic_surface.m #For computing the roots of an elastic-plate
    dispersion relation.

dispersion_free_surface.m #For computing the roots of a free-surface
    dispersion relation.

eigenFreqSW.m #To find the resonance frequencies of the shallow water
    problem using Newton Raphson.

export_fig.m #EXPORT_FIG package

findHInterpolated.m #Interpolating the H matrix at a specified frequency

findResonanceCplx.m #[depreciated]
getMatrices.m #[depreciated]
```

```
iceFEM
├─ include
│  ├─ macros.idp
│  └─ functions.idp
├─ 1_SIMPLE5
├─ modules
├─ Meshes
│  └─ BEDMAP2
├─ CreatePaths.m
├─ RealRefIce.m
├─ bedMapProfile.m
├─ thinPlate.m
├─ eigenSolve.edp
├─ iceshelf2d.edp
├─ iceshelf3d.edp
├─ iceberg.edp
├─ solveBEDMAP2.edp
├─ timeDomain.edp
├─ simple.edp
└─ simpleEB.edp
```

```
1_SIMPLE5
├─ 2_Deformation
├─ 2_Modes
├─ 2_ModesMatrix
│  ├─ Interpolated_L
│  ├─ Interpolated_H
│  └─ Interpolated_F
├─ 2_Potential
├─ 2_RefCoeff
│  ├─ Interpolated_R
│  ├─ RefCoeff_Dif
│  └─ RefCoeff_Rad
└─ 2_Stresses
```

Figure 1: Directory structures of the main package (left).The `modules` folder contains a list of **`MATLAB`** scripts for interpolation and visualization. The `meshes` folder contains a sample geometry data from the BEDMAP2 dataset. A list of example FreeFem scripts using the iceFEM package (`*.edp`) and `MATLAB` scripts (`*.m`) available in the directory is also shown. A sample working directory 1_SIMPLE5 (right), generated using the directory generation script **`./genDir.sh 1_SIMPLE5`**.

```

getProperties.m #Functions to get the properties of the ice

importfiledata.m #For FreeFem visualization in MATLAB to import the function values on the mesh points

importfilemesh.m #For FreeFem visualization in MATLAB to import the mesh points

interpolateFreq.m #Interpolate the system of equations on the new frequency space (real valued) and store the system of equations in 2_ModesMatrix/Interpolated_*

interpolateFreqComplex.m #Interpolate the system of equations on the new frequency space (complex valued) and store the system of equations in 2_ModesMatrix/Interpolated_*

interpolateRefCoeff.m #Interpolate the diffraction and radiation reflection coefficients and store in 2_RefCoeff/Interpolated_R

movingplate.m #Solve the thin-plate potential flow moving plate problem for uniform geometries.

pltphase.m #For complex plot

resonSW.m #To find the complex resonance frequencies of the shallow water problem. Uses eigenFreqSW.m.

shallowmovingplate.m #To solve the thin-plate shallow water problem for uniform geometries.

zdomain.m #For complex plot.
```

1.2 Modal Expansion Methods

We consider the fluid–structure interaction problem of modelling ocean–wave induced ice–shelf vibrations whose schematic and the governing equations is shown in Figure 2. The details of the governing equations together with the boundary conditions and the solution method is discussed by [30]. In this section, we provide a brief overview of the governing equations and the solution method for the purpose of demonstrating the code. The fluid flow is modelled using potential flow theory and the ice shelf is modelled as an elastic solid. The incident wave from the open–ocean region is modelled using a non–local boundary condition at the ocean–cavity interface, formulated as a general robin boundary condition [26, 30]. The vibration response of the ice shelf is expressed as a linear combination of its in–vacuo modes, i.e.,

$$\mathbf{u}(x, z) = \sum_{j=1}^{N} \lambda_j \boldsymbol{\eta}_j(x, z). \tag{1}$$

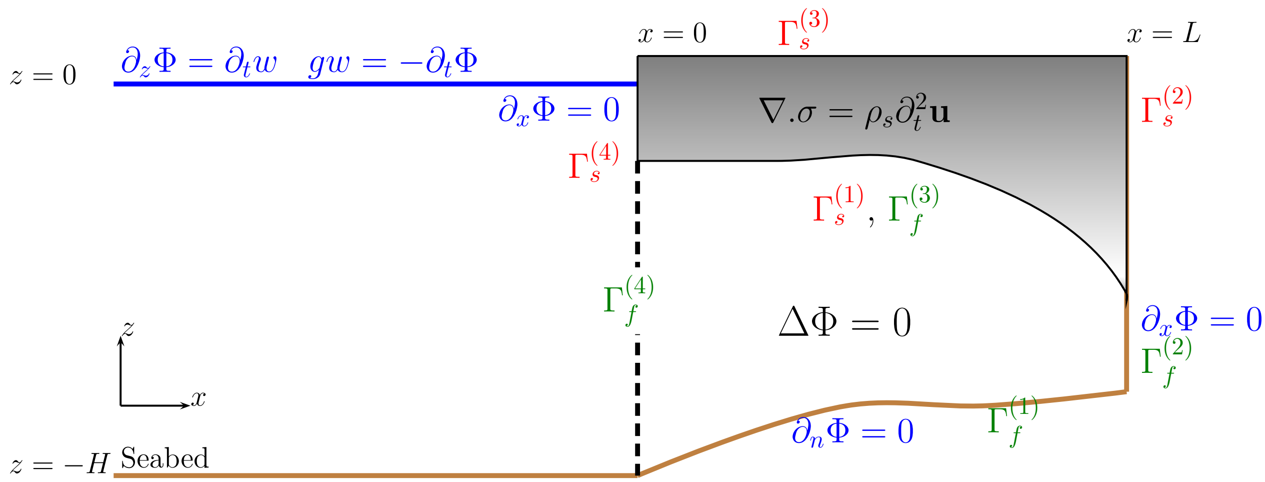


Figure 2: Geometry and governing equations

Similarly, the velocity potential is expressed as a sum of the diffraction potential, $\phi_0(x, z)$ which corresponds to the incident wave and the radiation potentials, $\phi_j(x, z)$ which are coupled with the in–vacuo modes of the ice–shelf. Each of the diffraction and radiation potentials are obtained by the finite element method. The expression for the velocity potential $\phi(x, z)$ is given by

$$\phi(x, z) = \phi_0(x, z) + \sum_{j=1}^{N} \lambda_j \phi_j(x, z). \tag{2}$$

This sets up a reduced system

$$\mathbf{H}\boldsymbol{\lambda} = \mathbf{f}, \tag{3}$$

which can be solved to obtain modal coefficients $\boldsymbol{\lambda}$. The scattering matrix has the form,

$$\mathbf{H} = \mathbf{K} - \omega^2 \mathbf{M} + i\omega \mathbf{B} \tag{4}$$

where $\mathbf{K}$ denotes the modal stiffness matrix, $\mathbf{M}$ denotes the modal mass matrix and $\mathbf{B}$ is a matrix that arises due to the fluid coupling. The method is powerful in the sense that the dimension of the linear system (4) is extremely small compared to the finite element degrees of freedom. Moreover, the entries of the reduced system are analytic functions of frequency. These two properties enable us to construct a large number of solutions without solving the finite element problem for different frequencies. The only knowledge required to obtain the displacements are the in-vacuo mode shapes which are independent of the fluid.

1.3 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

```
>> ./genDir.sh 1_TEST
>> mpirun -np 2 FreeFem++-mpi -v 0 simple.edp -Tr 4000 -hsize 0.08
```

in the command line. The first command generates the required directory structure used by the `simple.edp` script. This solves the ice-shelf problem using linear elasticity for the ice combined with potential flow for the fluid. The code produces the following output:

```
Dimension : 2
Dimension : 2
Imported Cavity Mesh (proc 1)...
Refining Cavity Mesh (proc 1) ...
Cavity : Before Refinement, NBV = 1828
Imported Ice Mesh (proc 0)...
Refining Ice Mesh (proc 0) ...
Ice : Before Refinement, NBV = 1746
Ice : After Refinement, NBV = 2423
Cavity : After Refinement, NBV = 3591


Reflection Coefficient = (0.631992,0.775192)
Absolute Value = 1.00017
```

For the ice-shelf problems, $|R| \approx 1$ due to energy conservation and it can be used to check the solution. When the macro **setProblem** is invoked, optional parameters can be specified to modify the problem.

```
 FreeFem++ -ne -v 0 [FILENAME].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]
```

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.

NOTE: For running the default FreeFem scripts, some default directories need to be generated. Simply run the command

```
>> for m in 1_BEDMAP2 1_SIMPLE5 2_ICEBERG 1_SIMPLE3D; do echo ./genDir.sh
   $m; done
```

1.4 MATLAB Interface

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

1.4.1 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.

1.4.2 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).

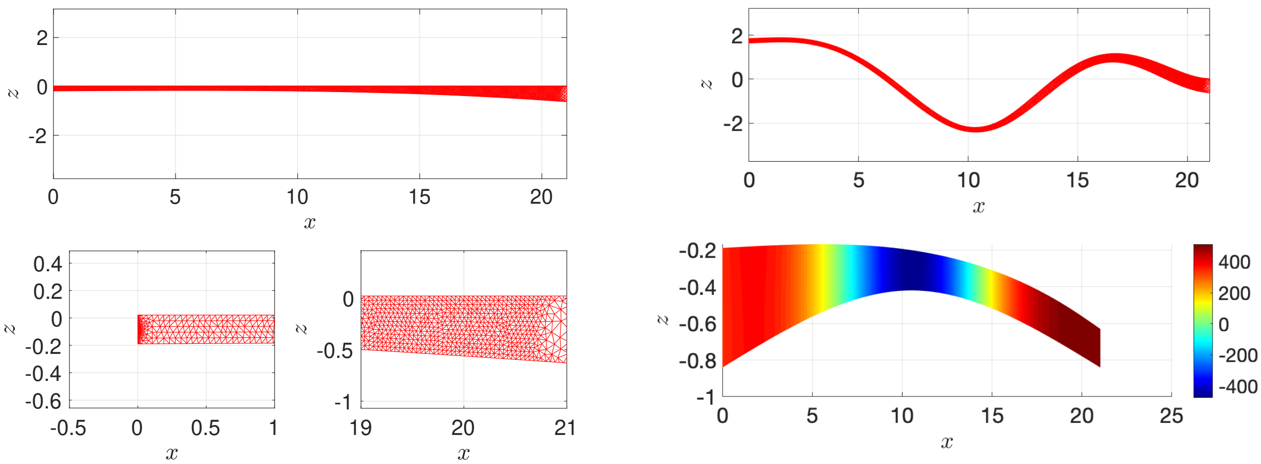
```
%% Run the FreeFem++ code;
global ff
file = 'simple1.edp';
ffpp=[ff,' -nw -ne ', file];
cmd=[ffpp,' -Tr ',num2str(real(T)),' -Ti ',num2str(imag(T)),' -H ',num2str(
   H), ' -L ',num2str(L),' -h ' ,num2str(th),' -N ',num2str(3), ' -isUniIce
    ',num2str(0), ' -isUniCav ',num2str(0)];
[aa,bb1]=system(cmd);
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.

1.4.3 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, solname);
```



(a) Finite element mesh

(b) Displacement and Velocity Potential.

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

where **uh** denotes the finite element solution, **Th** denotes the finite element mesh and **solname** denotes string variables which contains the name of the solution files. The macro writes two files **solname.msh** and **solname** which contains the mesh data and the solution name 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 **[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
```

Sample results are shown in Figure 3.

NOTE: 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**. The **MATLAB** routines have been tested only for 2D meshes. For 3D meshes, it is recommended to use ParaView.

Visualization can also be done using ParaView, for example,

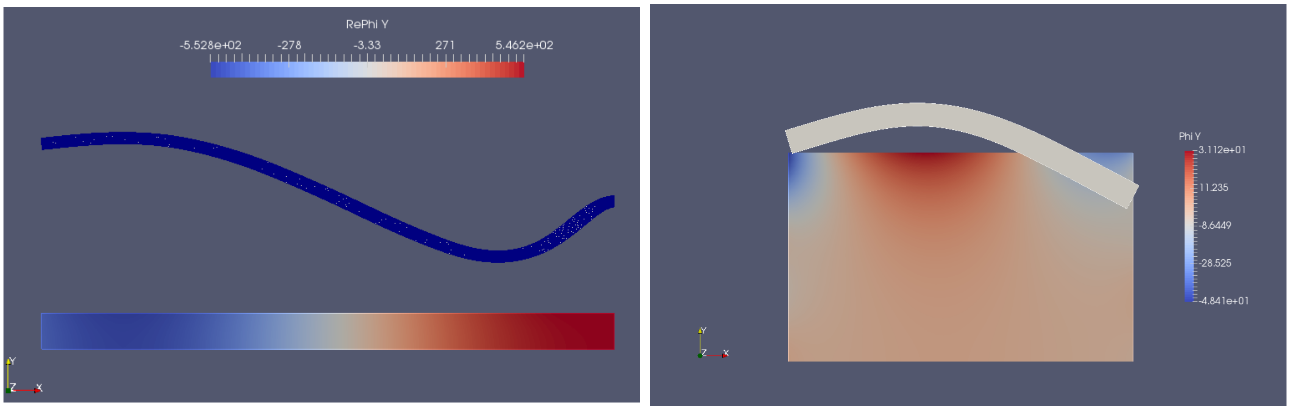


Figure 4: Figure showing the plots generated by ParaView.

```
int[int] Order=[1];
savevtk(FileName,MeshName,FuncName,dataname="VecFun ScalFun",Order=Order);
```

saves a `.vtk`/`.vtu` file for ParaView visualization. FreeFem offers extensive support for ParaView and examples of the figures generated using ParaView can be found in the `README.md` file located in the main repository. For more details on the ParaView visualization, refer to the FreeFem manual.

NOTE: In later versions of iceFEM, ParaView would become the default method of visualization.

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. If not using non-dimensionalization, leave the values of L_c and t_c to be equal to 1. Automatically set if `setProblem` is called.

- `omega`. Data Type: `complex`. The incident frequency computed from the wave period. $\omega = 2\pi/T$.

- `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.

- `LL, HH, dd, tth`. Data Type: `real`. The non-dimensional values length of the ice-shelf, cavity-depth, submergence, shelf-thickness. Can be set manually, but consistency needs to be ensured, if meshes are imported from an external source.

  `lambdahat, muhat`. The ratio λ/L_c^2 and μ/L_c^2 where λ, μ are the Lamé parameters.

  `gammahat, deltahat`. The ratio ρ_w/L_c and $\rho_w g/L_c$. `tt`. Data Type: `complex`. Value of the incident wave period, `Tr + 1i*Ti`. Computed from the user-input values of `-Tr, -Ti`. Set automatically.

- `Ap`. Data Type: `complex`. The amplitude of the incident velocity potential. Computed from the incident wave period. Set automatically.

- `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`. 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.`

- `WhBdy, VhBdy`. Data Type: `fespace`. Boundary Finite element spaces for the cavity (`WhBdy`) and the ice–shelf (`VhBdy`). Used to speed–up the construction of the reduced system by the `macro buildReducedSystemOptim`.

- `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 before obtaining the matrices using `getLaplaceMat` macro.

- `STIMA, BMASSMA, LHS`. 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()` and is generally not changed. Once computed the user can set the `LHS` matrix, for example, `LHS=STIMA+(BMASSMA)`.

- `RHS`. Data Type: `Wh<complex>`. Contains the RHS function corresponding to the function `fh`. The finite element solution, say, `uh` is computed using

```
uh[]=LHS^-1*RHS[];
```

- `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*()`.
- `xi`. Data Type: `complex[int]`. Solution of the reduced system set by the `macro solveReducedSystem` Do not modify.
- `mu`. Data Type: `real[int]`. Variable to store the eigenvalues of the in-vacuo Euler Bernoulli problem. Generated by `macro solveEigenEB()`. Do not Modify.

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()`:

  The `macro setProblem` receives the length of the ice-shelf L, thickness h and submergence d of the ice-shelf, depth of the ocean H and the incident wave frequency ω. The macro also computes the value of the non-dimensional constants along with `lambdahat, muhat, gammahat, deltahat`.

- `macro solveDispersion()`:

  Macro to obtain the roots of the dispersion equations

  $$\begin{aligned} -k\tan kH &= \alpha, \quad \text{and} \\ -k_d \tan k_d(H-d) &= \alpha \end{aligned}$$

  which updates the arrays `k,kd`.

- `macro setMeshIce(bottom_right_y, middle_x, middle_y)`:

  Macro to set simple non-uniform meshes using 3–point cubic spline technique. Updates `ThIce`

- `macro setMeshCav(middle_x, middle_y, bottom_right_y)`:

  Macro to set simple non-uniform meshes using 3–point cubic spline technique. `ThCavity`. NOTE: The boundary labels are as follows and the same convention is followed in 3D problems as well.

For the ice–shelf meshes:

- `label=1`: Free-boundary where $\boldsymbol{\sigma} \cdot \mathbf{n} = 0$.
- `label=2`: Fixed-boundary where $\mathbf{u} = 0$.
- `label=3`: Shelf–cavity interface boundary where $\boldsymbol{\sigma} \cdot \mathbf{n} = \rho_w g u_2 - \rho_w \partial_t \Phi$.

For the cavity meshes:

- `label=1`: No–normal flow boundary where $\nabla\phi \cdot \mathbf{n} = 0$.
- `label=2`: Free–Surface boundary condition where $\partial_z \phi = \alpha\phi$.
- `label=3`: Shelf–cavity interface boundary where $\partial_t \mathbf{u} \cdot \mathbf{n} = \nabla\phi \cdot \mathbf{n}$.
- `label=4`: Non–Local boundary **inlet**, where $\partial_n \phi = \mathbf{Q}\phi + \chi$. The corresponding routine to compute the reflection coefficient is `macro getRefCoeff(4, phi,Ref).`
- `label=5`: Non–Local boundary **outlet**, where $\partial_n \phi = \mathbf{Q}\phi$. The corresponding routine to compute the reflection coefficient is `macro getRefModes(5, phi,Ref).`

The default boundary labels used by `setMeshIce` and `setMeshCav` are shown in Figure 5.

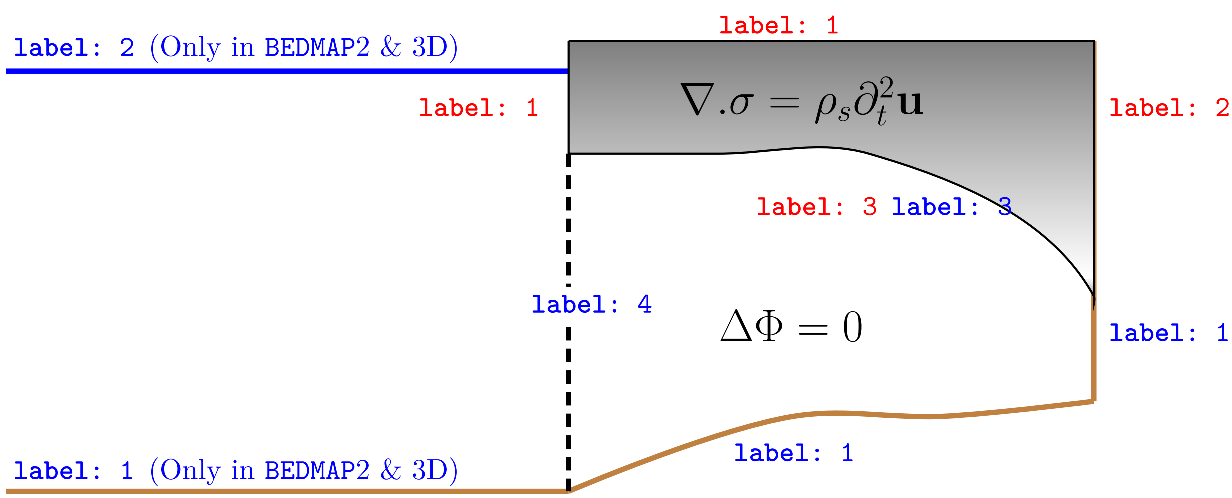


Figure 5: Geometry and labels for the cavity domain (blue) and the ice–shelf domain (red).

- `macro solveEigen()`:

  Macro to solve the Eigenvalue problem to obtain the in–vacuo modes. Always preceded by:

```
    //-------------------
    Xh[int][VX,VY](nev);
    real[int] ev(nev);
    //-------------------
    solveEigen;
```

  and updates the variables `VX,VY,ev`.

- `macro writeEigen()`:

  Macro to write the eigenmodes to the disk. Saved in `iceFEM/[WORKING_DIR]/2_Modes`.

- `macro readEigen()`:

  Macro to read the Eigenvalue problem to from `iceFEM/[WORKING_DIR]/2_Modes`. Always preceded by:

```
    //------------------
    Xh[int][VX,VY](nev);
    real[int] ev(nev);
    //------------------
    readEigen;
```

  and updates the variables `VX,VY,ev`.

- `macro getQphi(int bInd,matrix<complex> MQ)`:

  Macro to compute the non–local boundary condition on the boundary `bInd` and store it in the variable `MQ`. Refer to the boxed text above for more details on the boundary labelling.

- `macro getChi(Wh<complex> chi1)`:

  Macro to compute the forcing function on the boundary `label=4` and store the function in `chi1`.

- `macro getLaplaceMat(a,b,c)`: Computes the matrices `LHS, STIMA, BMASSMA`. The `STIMA` is the stiffness matrix, i.e.,

$$[\texttt{LHS}]_{jk} = \int_{\Omega} \nabla u_h \cdot \nabla v_h \, dx,$$

  the boundary mass matrix `BMASSMA` corresponding to the boundary with the `integer label 2` i.e., the inner–product (for free-surfaces)

$$[\texttt{BMASSMA}]_{jk} = \int_{\Gamma_2} u_h \, v_h \, ds.$$

  Also computes the `RHS` vector corresponding to the boundary with the `integer label 4` and `3` (for non-local **inlet** and wetted surfaces, respectively) i.e., the inner-product

$$[\texttt{RHS}]_k = \int_{\Gamma_4} (-\,\texttt{fh}\,)\, v_h \, ds + \int_{\Gamma_3} -i * \texttt{omega} * \texttt{Lc} * \big(\texttt{a} \cdot \texttt{N.x} + \texttt{b} \cdot \texttt{N.y} + \texttt{c} \cdot \texttt{N.z}\big)\, ds$$

  //IF 3D.

  and

$$[\texttt{RHS}]_k = \int_{\Gamma_4} (-\,\texttt{fh}\,)\, v_h \, ds + \int_{\Gamma_3} -i * \texttt{omega} * \texttt{Lc} * \big(\texttt{a} \cdot \texttt{N.x} + \texttt{b} \cdot \texttt{N.y}\big)\, ds$$

  //IF 2D. (`c` can be set to 0).

- `macro getLaplaceMatEB(m,rad)`

  Same as `getLaplaceMat(a,b,c)` but to compute the solution of the Laplace's equation using the roots of the Euler–Bernoulli equation `mu`. Argument `m` denotes the mth radiation potential and `rad`=0 for rigid–ice ($\int_{\Gamma_3} = 0$) and `rad`=1 for moving-ice ($\int_{\Gamma_3} \neq 0$).

- `macro getLaplaceMatDBC(m,rad)`:

  Same as `getLaplaceMatEB(m,rad)` but for Dirichlet boundary condition

```
on(4, phih=fh)
```

  on Γ_4, instead of the Neumann boundary condition

```
\int1d(ThCavity, 4)(fh*vh)
```

  on the inlet.

- `macro buildReducedSystem(Xh[int,int] [VX, VY] ,Wh<complex> phi0,Wh<complex>[int] phij (nev))`:

  Build the reduced system using the modal functions `VX, VY, phi0, phij`. Refer to the tutorial on how to assign the functions for a sample ice–shelf problem.

- `macro buildReducedSystemEB(real[int] mu ,Wh<complex> phi0, Wh<complex>[int]phij(nev ), complex alpha,real beta, real gamma)`:

  Same as `buildReducedSystem`, but for the thin–plate problem. The in–vacuo modes are characterized by the eigenvalues `mu`, but contains extra parameters `alpha, beta, gamma`. Refer to the tutorial on how to assign the functions for a sample ice–shelf problem.

- `macro buildReducedSystemOptim()`:

  Optimized version of the `buildReducedSystem` macro where the construction of the reduced system is done using the boundary value of the solution only. The user needs to define a new function `phi00`:

```
WhBdy<complex> phi00=phi0;
```

  and an array of boundary functions

```
WhBdy<complex>[int] phijj(nev);
for(int m=0; m<nev; m++)
{  : //Compute phij[m]
   phijj[m]=phij[m];
}
```

  Once `phi00, phijj, VX, VY` set, then simply call

```
buildReducedSystem;
```

  For more advanced example using `MPI`, see the examples `iceshelf2d.edp, iceshelf3d.edp` etc.

- `macro solveReducedSystem()`:

Solve the reduced system built using the previous macros and store the solution in the variable `xi`.

- `macro writeToMATLAB(uh, Th, solfilename, meshfilename)`
- `macro writeReducedSystem()`:

  Macro to write the relevant files for analysis. These files are used by the relevant Python/**MATLAB** Scripts to generate useful plots. See Section 3 for more details.

- `macro setupWorkingDir(DirName)system("./genDir.sh "+DirName);`

  This is a macro to generate the working directory from the **FreeFem++** code is needed.

3 Quantities of Interest

The main feature of the modal expansion method is that the entries in the final reduced system are analytic functions of the frequency. Thus interpolation can be performed on the reduced system to obtain more solutions in the frequency space without solving the finite element problem repeatedly. This is an inexpensive operation since the dimension of the reduced system is much less than that of the original system. Using the **FreeFem++** part of the package, we first obtain the reduced system and then using an external package like **MATLAB** or Python, the reduced system can be obtained through interpolation on the frequency space. Once the reduced system

$$\mathbf{H}(\omega)\,\lambda = \mathbf{f}(\omega)$$

is obtained, we can perform interpolation on the LHS matrix and the RHS vector. The associated matrix and vector generated by **FreeFem++** is written using the macro `writeReducedSystem`. This generates the following files:

- `SolutionDir+"2_ModesMatrix/ReH"+iter+".dat"`: Stores the **Real** part of the **H** matrix in the folder `2_ModesMatrix` folder. The variable `iter` (user specified) is appended to the filename.
- `SolutionDir+"2_ModesMatrix/ImH"+iter+".dat"`: Stores the **Imaginary** part of the **H** matrix in the folder `2_ModesMatrix` folder. The variable `iter` (user specified) is appended to the filename.
- `SolutionDir+"2_ModesMatrix/ReF"+iter+".dat"`: Stores the **Real** part of the **F** vector in the folder `2_ModesMatrix` folder. The variable `iter` (user specified) is appended to the filename.
- `SolutionDir+"2_ModesMatrix/ImF"+iter+".dat"`: Stores the **Imaginary** part of the **F** vector in the folder `2_ModesMatrix` folder. The variable `iter` (user specified) is appended to the filename.

Once the diffraction R_0 and radiation R_i reflection coefficients are computed using the macro `getRefCoeff` and `getRefModes`, respectively, we could construct the reflection coefficients back by setting

$$R = R_0 + \sum_{j=1}^{N} \lambda_j R_j.$$

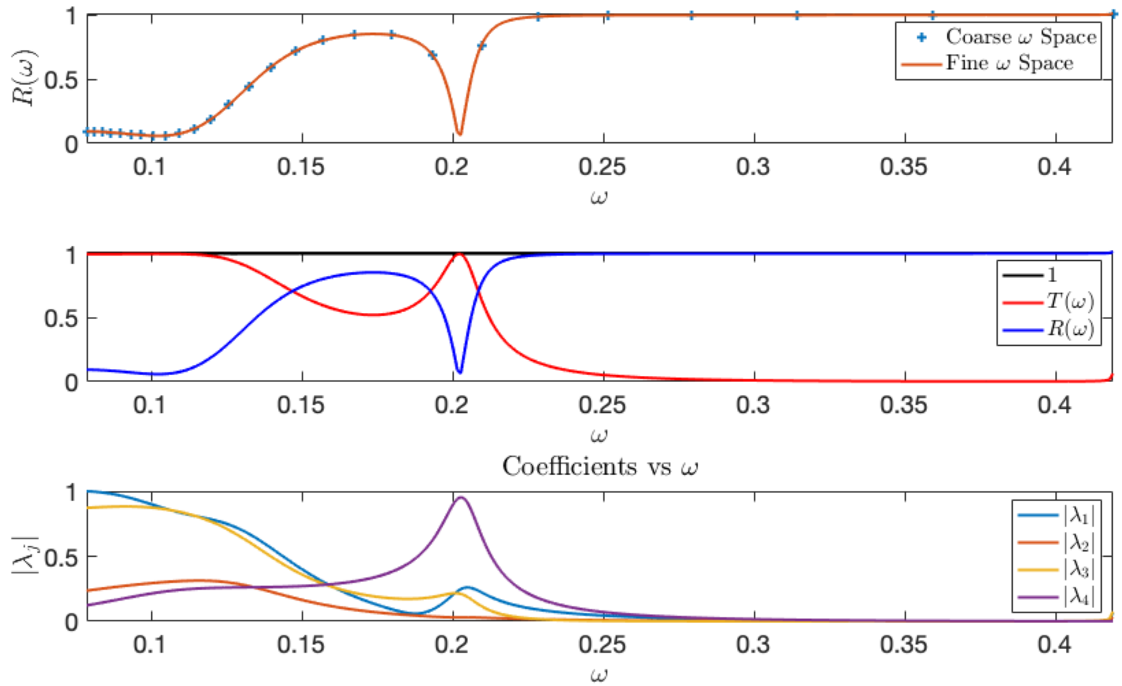


Figure 6: Figure showing (Top) the value of the reflection coefficients on a coarse -space (blue,+) and on a fine -space (red,solid) obtained after solving the interpolated system. (Middle) The value of the reflection and transmission coefficients as a function of the incident frequency. The energy conservation result $|T|^2 + |R|^2 = 1$ is also verified. (Bottom) Modal contribution, $|\lambda_j|$ of the various in-vacuo modes as a function of frequency.

The reflection coefficients can then be written to the folders, `[ROOT_DIR]/2_RefCeoff/RefCoeff_Dif/` for the diffraction part and `[ROOT_DIR]/2_RefCeoff/RefCoeff_Rad/` for the radiation part. The diffraction coefficient is stored in the variable `complex RefC` (an `iceFEM` keyword). The radiation reflection coefficients are stored in the array `complex[int] Refc[nev]` (an `iceFEM` keyword):

- `SolutionDir+"2_RefCoeff/RefCoeff_Rad/refC"+iter+".dat"`: Stores the diffraction reflection coefficient.
- `SolutionDir+"2_RefCoeff/RefCoeff_Rad/refC"+iter+".dat"`: Stores the radiation reflection coefficients.

The first column in the files contains the real part of the reflection coefficient and the second column, the imaginary part.

As mentioned earlier, several **MATLAB** routines are available in the **modules** folder to help the user start off with generating more solutions. For more details on the list of **MATLAB** routines available, refer Section 1. Here we describe the two important routines that implements the interpolation. The main **MATLAB** functions:

```
[omegaNew,detH,condH]=interpolateFreq(a,b,omega,Nev,filePath,npts,isSolve):
    % Input:
```

```
    % 1) a, b: Endpoints of the space frequency space.
    % 2) omega: The original frequency space. Must be the same as the
       finite element frequency domain.
    % 3) Nev: Number of in-vacuo modes.
    % 4) filePath: Full Path to the '2_ModesMatrix' file in the working
       directory. For eg. the string '[EXAMPLE_DIR]/2_ModesMatrix'.
    % 5) npts: Number of points in the new frequency space.
    % 6) isSolve: Toggle Option to solve the new reduced system and write
       to disk at the locations:
    %   [EXAMPLE_DIR]/2_ModesMatrix/Interpolated_H/
    %   [EXAMPLE_DIR]/2_ModesMatrix/Interpolated_F/
    %   [EXAMPLE_DIR]/2_ModesMatrix/Interpolated_L/

    % Output:
    % omegaNew: The new frequency space.
    % detH: Determinant of the scattering matrix H.
    % condH: Condition number of the scattering matrix H.

interpolateFreqComplex(omega,omegaNew,Nev,filePath):
    % Routine to perform interpolation, but for a complex frequency grid.
    % Input:
    % 1) omega: The original frequency space (grid). Must be the same as
       the finite element frequency domain.
    % 2) omegaNew: The new frequency space (grid).
    % 3) Nev: Number of in-vacuo modes.
    % 4) filePath: Full Path to the '2_ModesMatrix' file in the working
       directory. For eg. [EXAMPLE_DIR]/2_ModesMatrix.

    % Output:
    % Reduced system and solutions at the locations:
    %   [EXAMPLE_DIR]/2_ModesMatrix/Interpolated_H/
    %   [EXAMPLE_DIR]/2_ModesMatrix/Interpolated_F/
    %   [EXAMPLE_DIR]/2_ModesMatrix/Interpolated_L/


interpolateRefCoeff(omega,omegaNew,Nev,filePath,TorC):
    % Input:
    % 1) omega: The original frequency space (grid). Must be the same as
       the finite element frequency domain.
    % 2) omegaNew: The new frequency space (grid).
    % 3) Nev: Number of in-vacuo modes.
    % 4) filePath: Full Path to the '2_ModesMatrix' file in the working
       directory. For eg. [EXAMPLE_DIR]/2_ModesMatrix.
    % 5) TorC is a variable to indicate reflection/transmission coeffcients
    %   If TorC = 'C' (Interpolated Reflection Coefficients)
    %   If TorC = 'T' (Interpolated Transmission Coefficients)

    % Output:
    % Files containing the Diffraction and Radiation Reflection
       coefficients found in [EXAMPLE_DIR]/2_RefCoeff/Interpolated_R/
```

Figure 6 shows an example output of the interpolation. An example on how to access the interpolated quantities is shown in the repository example **MATLAB** script **here**.

> NOTE 1: In the current version (v1.0.0), the finite element problem is handled by **FreeFem++**, while the interpolation is done using **MATLAB** or Python, if using the **PyIceFem** branch.
>
> NOTE 2: Switching the branch over to **PyIceFem**, is the Python Module located in the repository. This code is largely under development, with some examples available to solve simple problems. The main module that needs to be added is the BEDMAP2 integration which needs to implemented in the Python version.

4 Tutorial

In this section, we describe how to solve two ice-shelf problems using the iceFEM package.

4.1 Euler-Bernoulli beam

In this subsection, we solve the example in [26]. The complete example is provided in the package as **simpleEB.edp**. We illustrate the use of the various routines available in the package. First we run the **genDir.sh** command to generate the necessary directories for the package. For example run

```
>> ./genDir.sh 1_THIN
```

Next, we invoke the necessary modules by importing **macros.idp** and set dimension and the finite element space of the problem.

```
verbosity=0.; //Sets the level of output.
macro dimension 2//EOM" Sets the dimension of the problem.
macro fspace 1//EOM" Set the FE Space order.
include "macros.idp" //Import the package.

SolutionDir="1_THIN" //Should be an existing directory structure.
```

Next we set the problem and solve the Dispersion equation to obtain the wave numbers. The macro **setProblem** sets up the problem by obtaining the dimensional parameters from the command line. The macro also computes the characteristic length and time.

```
//Set the problem and solve the dispersion equation.
setProblem;
solveDispersion; //The roots are stored in complex[int] k(NModes+1), kd(
   NModes+1);
```

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

refineMesh; //Perform Uniform Mesh refinement.
```

If **isUniformIce/isUniformCav=true**, any option given as arguments will be overridden to default values. The default numbering for the ice–shelf and cavity labels are shown in Figure 5. 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-vacuo EB
    modes.
```

The eigenvalues of the cantilever modes will be stored in the variable **mu**. 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 macros are available in the **macros.idp** file to calculate the necessary matrix and function. The following code block is used to obtain the boundary condition. For more details on the derivation of the non–local boundary condition, see [26].

```
Wh<complex> chi1;
matrix<complex> MQ;
getQphi(4,MQ); //Get the matrix on the boundary 4.
getChi(chi1); //Get the forcing function chi1.
```

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;
```

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.

```
//Parameters for the system.
complex ndOmega=2*pi/tt; //tt is the non-dimensional wave--period. Set using [setProblem]
alpha = HH*ndOmega^2;//alpha is a keyword in the package
real beta = 1;
real gamma = densRat*tth; //densRat is the ratio of density.
                          //tth is the non-dimensional thickness.
                          //Both set by [setProblem]

//Build and solve the reduced system
buildReducedSystemEB(mu, phi0, phij, alpha, beta, gamma);
solveReducedSystem; //The solution is stored in xi
```

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

```
Wh<complex> phi=phi0; //Define a function phi of type Wh<complex>
for(int m=0; m<nev; m++)
    phi=phi+xi[m]*phij[m];
```

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 macro **getRefCoeff** is available in the module **macros.idp**. The following code block computes the reflection coefficient for the problem.

```
complex Ref;
getRefCoeff(4,phi,Ref);

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

To run the sample code **simpleEB.edp**, run

```
mpirun -np 2 FreeFem++-mpi -v 0 simpleEB.edp -Tr 200  -L 10000 -h 200 -H
    800 -nev 8 -hsize 0.04
```

which produces the following output

```
Dimension : 2
Dimension : 2
Imported Cavity Mesh (proc 1)...
Refining Cavity Mesh (proc 1) ...
Cavity : Before Refinement, NBV = 1495
Imported Ice Mesh (proc 0)...
Refining Ice Mesh (proc 0) ...
Ice : Before Refinement, NBV = 871
Ice : After Refinement, NBV = 4010
Cavity : After Refinement, NBV = 11837
Reflection Coefficient = (0.530218,0.848019)     Absolute Value = 1.00013
```

Computing the reflection coefficient is a good check for the accuracy of the solution. The thin–plate model (`thinPlate.m`) for the same dimensional parameters output the following reflection coefficient.

```
RefTP =

    0.4936 + 0.8697i
```

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.

4.2 2D Linear Elasticity

The second example is when the ice is modelled using the 2D elasticity equations under plane strain conditions. 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. Before running the code, run

```
>> ./genDir.sh 1_TEST
```

to generate the required working directory. The full code can be found below.

```
verbosity=0;

macro dimension 2//EOM"
macro fspace 1//EOM"

include "macros.idp"

```

```
SolutionDir="1_TEST";

setProblem;

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);

//Refine Mesh
refineMesh;


//Solve the Eigenvalue problem;
Xh[int][VX,VY](nev); //Define
real[int] ev(nev); //Define
solveEigen;

//Get non-local boundary condition
Wh<complex> chi1;
matrix<complex> MQ;
getQphi(4,MQ);
getChi(chi1);


Wh<complex> phi0;
func fh=chi1; //Set fh
getLaplaceMat(0,0,0);
LHS=STIMA+(MQ);
set(LHS,solver=UMFPACK);
phi0[]=LHS^-1*RHS[];


Wh<complex>[int] phij(nev);
for(int m=0; m<nev; m++)
 {
     func fh=0;
     getLaplaceMat(VX[m],VY[m],0);
     LHS=STIMA+(MQ);
     set(LHS,solver=UMFPACK);
     phij[m][]=LHS^-1*RHS[];
 }

buildReducedSystem(VX,VY,phi0,phij);

solveReducedSystem;
```

```

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];
 }

complex Ref;
getRefCoeff(4,phi,Ref);
if(mpirank==0){
    int[int] Order=[1,1];
    savevtk(SolutionDir+"/sol1_"+iter+".vtk",ThIce,[real(etax),real(etay)],
       dataname="ReDisp",order=Order);
    savevtk(SolutionDir+"/sol2_"+iter+".vtk",ThCavity,[real(phi),imag(phi)
       ],dataname="ReDisp",order=Order);
    cout<<"\n\nReflection Coefficient = "<<Ref<<endl;
    cout<<"Absolute Value = "<<abs(Ref)<<endl;
 }
```

Running the code, for example, for the following parameters

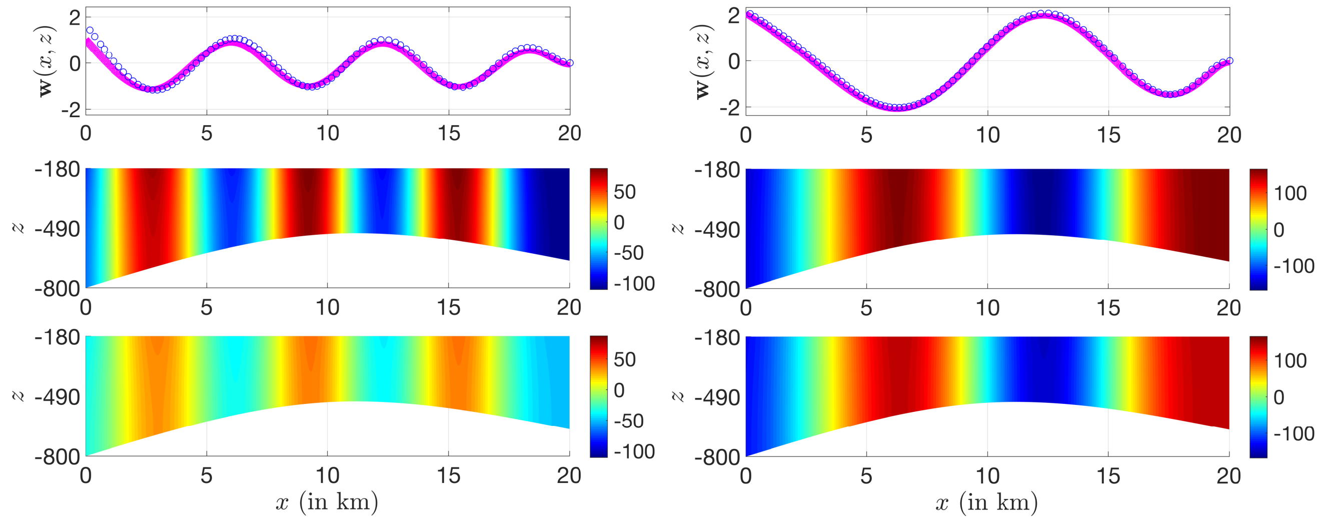
```
>> mpirun -np 2 FreeFem++-mpi -v 0 simple.edp -Tr 200 -L 10000 -h 200 -H
   800 -nev 8 -hsize 0.04
```

produces the following output,

```
Dimension : 2
Dimension : 2
Imported Cavity Mesh (proc 1)...
Refining Cavity Mesh (proc 1) ...
Cavity : Before Refinement, NBV = 1495
Imported Ice Mesh (proc 0)...
Refining Ice Mesh (proc 0) ...
Ice : Before Refinement, NBV = 871
Ice : After Refinement, NBV = 4010
Cavity : After Refinement, NBV = 11837


Reflection Coefficient = (0.504379,0.86364)
Absolute Value = 1.00014
```

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 7 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 7: 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 deviation can be observed for $T = 80\,\mathrm{s}$ case.

4.3 An example using MPI

In this section an example using MPI and the macro **buildReducedSystemOptim** is illustrated. The code snippet inside the macro is written assuming that the eigenfunctions corresponding to the in-vacuo modes and the velocity potentials are restricted to the boundary of the domain. The the construction of the reduced system is parallelized, if the user enables splitting of the meshes. The complete program along with the comments is shown below.

```
//Simple example to demonstrate the ice-shelf toolbox;
//NOTES:
//1)   The functions/macros/keywords from the toolbox is denoted by [iceFEM
   ] next to it.
//2)   For a list of KEYWORDS and FUNCTIONS, refer to the manual located in
    the Repository.

verbosity=0.;
real cpu=mpiWtime();
bool debug=true;

macro dimension 2//EOM" (Sets up the dimension of the problem);
macro fspace 2//EOM" (Sets up the FESpace);

include "macros.idp"

```

```
real timeTaken;

SolutionDir="1_SIMPLE5/";

macro Sigma(u,v)[2*muhat*dx(u)+lambdahat*(dx(u)+dy(v)),
                 2*muhat*dy(v)+lambdahat*(dx(u)+dy(v)),
                 muhat*(dy(u)+dx(v))]//EOM (Macro to define the stress tensor)"

//Sets up an example problem. Can control input using CMD line args.
//For a detailed list of default args. Refer the manual.
setProblem;

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

real starttime=mpiWtime();
//Build the meshes.
real botRight=-3.*tth, midPX=3.7*LL/4, midPY=-2.5*tth;

setMeshIce(botRight, midPX, midPY);//[iceFEM] For a list of ARGS, refer to the manual

real midx=LL/2., midy=-0.5*HH, endy=-HH;

setMeshCav(midx, midy, endy);//[iceFEM] For a list of ARGS, refer to the manual

refineMesh;//[iceFEM] Refine Mesh.
splitMesh(isSplit);//[iceFEM] Split Mesh for domain decomposition (isSplit is defaulted to 0).

real endtime=mpiWtime();
real difTime=endtime-starttime;//Time the code.
mpiReduce(difTime,timeTaken,processor(0),mpiMAX);
if(mpirank==0)
    cout<<"Time taken for Meshing = "<<timeTaken<<" s"<<endl;


//The respective finite element spaces are
//Vh, Vh<complex> -> P1/P2 on ICE.
//Wh, Wh<complex> -> P1/P2 on CAVITY.
//Xh, Xh<complex> -> [P1,P1]/[P2,P2] on ICE2d
//                -> [P1,P1,P1]/[P2,P2,P2] on ICE3d

Xh[int][VX,VY](nev); //[iceFEM] Define an array of fe-function to store in-vacuo modes. (Should be named: [VX,VY])
real[int] ev(nev); //[iceFEM] Define a real array for the eigenvalues. (Should be named: ev[])
```

```
starttime=mpiWtime();

solveEigen; //[iceFEM] Solve the Eigenvalue problem;


endtime=mpiWtime();
difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiMAX);
if(mpirank==0)
    cout<<"Time to solve Eigenvalue = "<<timeTaken<<" s"<<endl;
//readEigen;


//  2) Get the Non-local boundary condition
Wh<complex> chi1; //[iceFEM] Define a function chi1 on the cavity domain for the incident wave.
matrix<complex> MQ;//[iceFEM] Define a matrix MQ for the Q-operator.

getQphi(4,MQ);//[iceFEM] Build Q-Operator on boundary 4 of cavity.
getChi(chi1);//[iceFEM] Build Function Chi.

//  3) Solve for the diffraction potential.
Wh<complex> phi0;
func fh=chi1;//[iceFEM] Define and store in keyword fh, the right-hand side function on the ocean-cavity interface.
getLaplaceMat(0,0,0);//[iceFEM]
LHS=STIMA+(MQ);
set(LHS,solver=UMFPACK,eps=1e-20);
phih[]=LHS^-1*RHS[];
phi0=phih;//Store in phi0;
//Interpolate to boundary.
WhBdy<complex> phi00=phi0;


//Solve for radiation potential. [PARALLEL RUN is OPTIONAL]
Wh<complex>[int] phij(nev);
WhBdy<complex>[int] phijj(nev);//Define array of boundary functions
buildParti(nev); //[iceFEM] Build partition depending on the number of CPUs.
complex[int,int] PHIJ(WhBdy.ndof,nev),PHIJProc(WhBdy.ndof,partisize);
starttime=mpiWtime();
for(int m=start; m<=stop; m++)
 {
     func fh=0;
     getLaplaceMat(VX[m],VY[m],0);//[iceFEM]
     LHS=STIMA+(MQ);
     set(LHS,solver=UMFPACK,eps=1e-20);
     phih[]=LHS^-1*RHS[];
     phij[m]=phih; //The full solution is used to visualize.
     phijj[m]=phih; //Interpolate on the BOUNDARY alone and store in matrix
```

```
          .
     PHIJProc(:,m-start)=phijj[m][];
 }
int[int] rcounts1=rcounts*WhBdy.ndof, dspls1=dspls*WhBdy.ndof;
mpiAllgatherv(PHIJProc,PHIJ,rcounts1,dspls1); //Gather the solutions.
endtime=mpiWtime();
difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiSUM);
if(mpirank==0)
    cout<<"Time taken to solve potentials = "<<timeTaken/mpisize<<" s"<<endl;

//Unpack and set to phij locally in all procs (Only boundary)
for(int m=0; m<nev; m++)
    phijj[m][]=PHIJ(:,m);


//Build Reduced System using the modes. [H]{\lambda}={f}
starttime=mpiWtime();
buildReducedSystemOptim;//[iceFEM]

endtime=mpiWtime();
difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiMAX);
if(mpirank==0){
    complex[int] Refm(nev), Reft(nev);

    writeReducedSystem;//[iceFEM] Write the reduced system to a file.
    cout<<"Time taken to build reduced system = "<<timeTaken<<" s"<<endl;

 }

//Solve the reduced system;
solveReducedSystem; //[iceFEM]

//Compute the solution.
Vh<complex> etax, etay, etaxProc, etayProc;
Wh<complex> phi, phiProc;
for(int m=start; m<=stop; m++)
 {
     phiProc = phiProc + xi[m]*phij[m];
     etaxProc = etaxProc + xi[m]*VX[m];
     etayProc = etayProc + xi[m]*VY[m];
 }

mpiReduce(phiProc[],phi[],processor(0),mpiSUM);
mpiReduce(etaxProc[],etax[],processor(0),mpiSUM);
mpiReduce(etayProc[],etay[],processor(0),mpiSUM);

if(mpirank==0){
```

```
    //Compute the reflection coefficient.
    cout<<"\n\n\n";
    cout<<"Non-Dim parameter Lc,Tc = "<<Lc<<","<<tc<<endl;
    phi=phi+phi0;
    complex Ref;

    getRefCoeff(4,phi,Ref);//[iceFEM] Compute the Reflection Coefficient

    cout.precision(16);
    cout<<"Reflection Coefficient = "<<Ref<<endl<<"|R| = "<<abs(Ref)<<endl;

    //Write data to Paraview
    //iter is used to index the solution.
    int[int] Order=[1,1];
    savevtk(SolutionDir+"/sol1_"+iter+".vtk",ThIce,[real(etax),real(etay)
      ],[imag(etax),imag(etay)], dataname="ReDisp ImDisp",order=Order);
    savevtk(SolutionDir+"/sol2_"+iter+".vtk",ThIce,[real(Sigma(etax,etay)
      [0]), real(Sigma(etax,etay)[1]), real(Sigma(etax,etay)[2])],[imag(
      Sigma(etax,etay)[0]), imag(Sigma(etax,etay)[1]), imag(Sigma(etax,
      etay)[2])],dataname="ReSigma ImSigma",order=Order);
    savevtk(SolutionDir+"/solCavity"+iter+".vtk",ThCavity,[real(phi),imag(
      phi)],dataname="RePhi ImPhi",order=Order);
 }

starttime=cpu;
endtime=mpiWtime();
difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiMAX);
if(mpirank==0)
    cout<<"Total time = "<<timeTaken<<" s"<<endl;
```

4.4 Real shelf profiles using BEDMAP2

The last example is solving the linear elasticity problem with data obtained from the BEDMAP2 dataset. In this example the cavity region is assumed to extend into the open–ocean. This module of the software requires **MATLAB** and the Antaractic Mapping Tools (AMT) along with the BEDMAP2 dataset to run. Once the requisite packages are installed, launch **MATLAB** and run

```
antmap
load coast
patchm(lat,long, [0.5,0.5,0.5]);
bedmap2 patchshelves
[hice,hbed,hwater]=bedmap2_profile();
```

This lauches a **MATLAB** figure window showing the map of Antarctica. The user needs to click two points on the map to define a path as shown. Undo points by hitting **Backspace**. When you're satisfied with a path you've drawn, hit **Enter** to create a profile. To quit the user interface without creating a profile, hit **Esc**. The structures **hice,hbed,hwater** contain the coordinates of the ice–

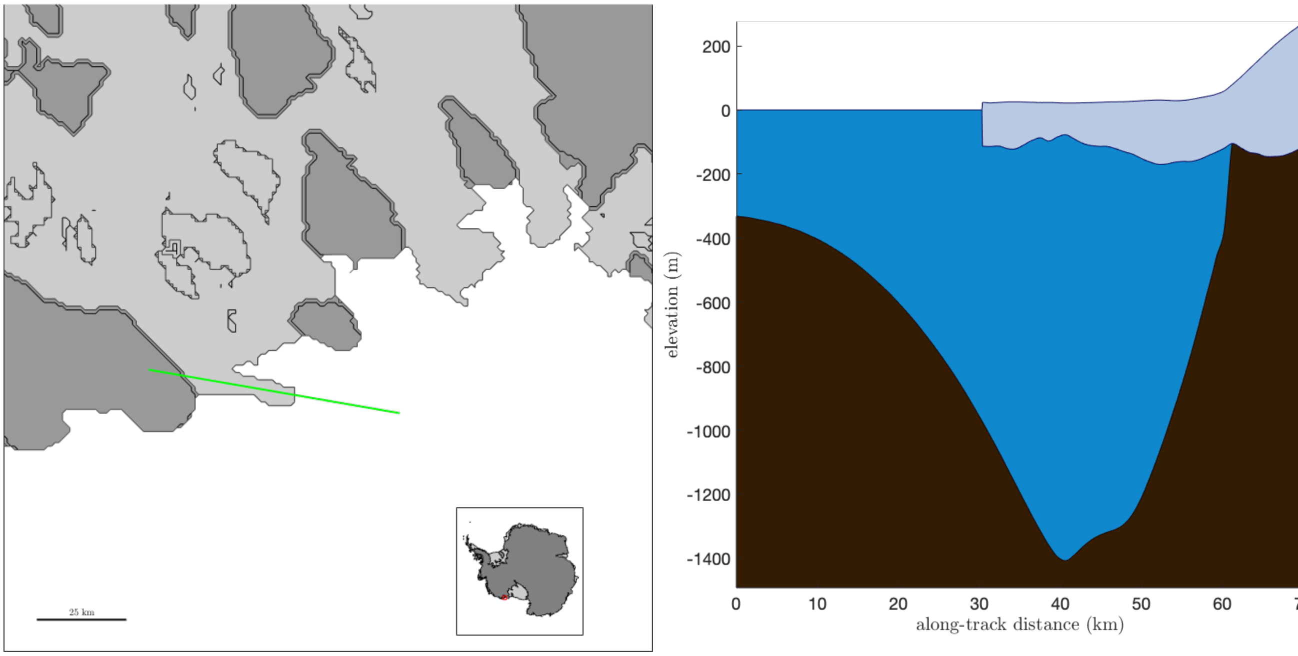


Figure 8: Sample output of **bedmap2_profile** command and the user-defined path (Left), shown in Green. The marked path shows the cross section of the Brunt ice-shelf (Right).

shelf, sea–floor and the open–ocean profiles, respectively inside the variable **Vertices**. The spline data can be extracted from the coordinates using **MATLAB** and this will be used by **FreeFem++** to reconstruct the cubic spline. The **MATLAB** script **bedMapProfile2** obtains the profile from the user and the script **solveBEDMAP2.edp** solves the problem. The results are shown in Figure 9. The code for the same is found below and is titled **solveBEDMAP2.edp** in the package. Sample output data can be found in **BEDMAP_Samples** folder.

```
/*
  WARNING:
  Should be run after splitting the meshes
  ff-mpirun -np 4 splitMesh.edp -hsize 0.02 -isBEDMAP 1
```

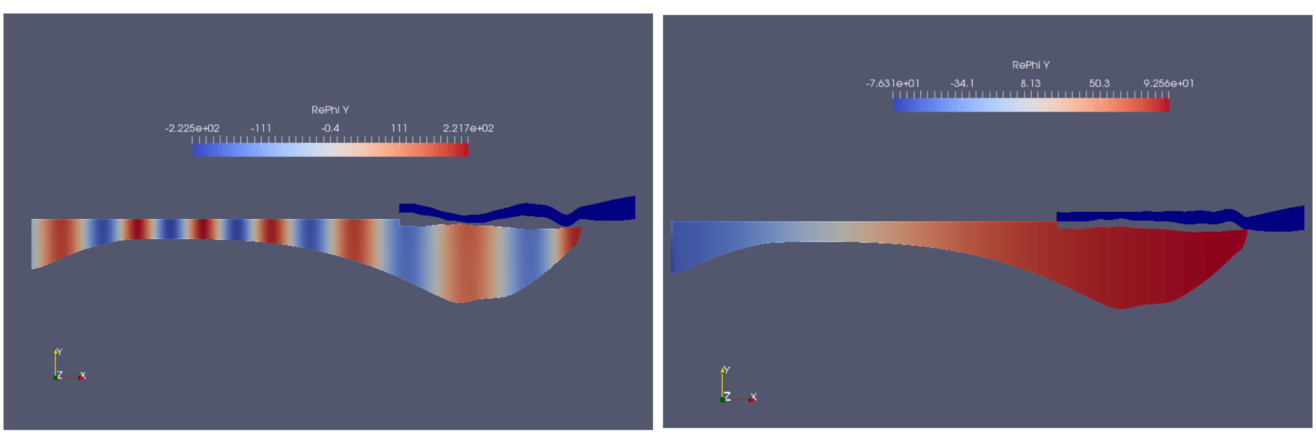


Figure 9: Figure showing sample Frequency domain solutions for an incident wave forcing of $T = 200\,\mathrm{s}$ and $T = 4000\,\mathrm{s}$.

```
*/

verbosity=0;

macro dimension 2//EOM"
macro fspace 2//EOM"

include "macros.idp"

macro extractFS(fefunc, febdymesh, filename){
  ofstream file(filename);
  for(int m=0; m<febdymesh.nv; m++){
    if(febdymesh(m).y==0)
      file<<real(fefunc(febdymesh(m).x,0))<<"\t"<<imag(fefunc(febdymesh(m).x,0))<<"\n";
  }
}//EOM" End of macro to extract the Free Surface of the function .

macro Sigma(u,v)[2*muhat*dx(u)+lambdahat*(dx(u)+dy(v)),
                 2*muhat*dy(v)+lambdahat*(dx(u)+dy(v)),
                 muhat*(dy(u)+dx(v))]//EOM (Macro to define the stress tensor)"

real timeTaken;

NModes=5;
SolutionDir="1_BEDMAP2/";
real cpu=mpiWtime();
int isMesh= getARGV("-isMesh",1);
int nborders= getARGV("-nborders",4);

macro writeSolution(var, filename){
  ofstream file(filename);
  file<<var<<endl;
}//Tiny macro to write files

//Construct/Load the meshes.
real starttime=mpiWtime();
iceBEDMAP2(6,isMesh);
if(isMesh){
  refineMesh; //refine the meshes and overwrite the original meshes.
 }
splitMesh(isSplit);
real endtime=mpiWtime();
real difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiMAX);
if(nborders!=4)
  dd=0.;

//Solve the dispersion equation.
```

```
solveDispersion;
if(mpirank==0)
  cout<<"Solved Dispersion Equation ... "<<endl;

matrix<complex> MQ;
Wh<complex> chi1;
getQphi(4,MQ);//[iceFEM] Build Q-Operator on boundary 4 of cavity.
getChi;//[iceFEM] Build Function Chi.
for(int m=0; m<NModes+1; m++)
  chi1=chi1+ctilde[m]*cos(kd[m]*(y+HH))/cos(kd[m]*(HH-dd));

if(mpirank==0)
  cout<<"Obtained the nonlocal boundary condition"<<endl;

//Solve EigenValue problem
Xh[int][VX,VY](nev);
real[int] ev(nev);
starttime=mpiWtime();
solveEigen;
endtime=mpiWtime();
difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiMAX);
if(mpirank==0)
  cout<<"Time to solve Eigenvalue = "<<timeTaken<<" s"<<endl;

//Solve for the Diffraction potential
Wh<complex> phi0;
func fh=chi1;
getLaplaceMat(0,0,0);
BMASSMA=alpha*BMASSMA;
LHS=STIMA+(MQ)+(-BMASSMA);
set(LHS,solver=UMFPACK,eps=1e-20);
phih[]=LHS^-1*RHS[];
phi0=phih;
WhBdy<complex> phi00=phi0;

Wh<complex>[int] phij(nev);
WhBdy<complex>[int] phijj(nev);
buildParti(nev);
complex[int,int] PHIJ(WhBdy.ndof,nev),PHIJProc(WhBdy.ndof,partisize);
starttime=mpiWtime();
for(int m=mpirank*parti; m<mpirank*parti+parti; m++)
  {
    func fh=0;
    getLaplaceMat(VX[m],VY[m],0);
    BMASSMA=alpha*BMASSMA;
    LHS=STIMA+(MQ)+(-BMASSMA);
    set(LHS,solver=UMFPACK,eps=1e-20);
    phih[]=LHS^-1*RHS[];
    phij[m]=phih;
```

```
    phijj[m]=phih;
    PHIJProc(:,m-start)=phijj[m][];
  }
int[int] rcounts1=rcounts*WhBdy.ndof, dspls1=dspls*WhBdy.ndof;
mpiAllgatherv(PHIJProc,PHIJ,rcounts1,dspls1);
endtime=mpiWtime();
difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiSUM);
if(mpirank==0)
  cout<<"Time taken to solve potentials = "<<timeTaken/mpisize<<" s"<<endl;

//Unpack and set to phij locally in all procs
for(int m=0; m<nev; m++)
  phijj[m][]=PHIJ(:,m);

F.resize(nev);
B.resize(nev,nev);
K.resize(nev,nev);
AB.resize(nev,nev);
starttime=mpiWtime();
buildReducedSystemOptim;
endtime=mpiWtime();
difTime=endtime-starttime;
mpiReduce(difTime,timeTaken,processor(0),mpiMAX);
if(mpirank==0){
    complex[int] Refm(nev), Reft(nev);
    cout<<"Time taken to build reduced system = "<<timeTaken<<" s"<<endl;
    writeReducedSystem;
 }

//Solve the reduced system.
solveReducedSystem;

//Compute the solution.
Vh <complex> etax, etay, etaxProc, etayProc;
Wh<complex> phi, phiProc;
for(int m=start; m<=stop; m++)
  {
    phiProc = phiProc + xi[m]*phij[m];
    etaxProc = etaxProc + xi[m]*VX[m];
    etayProc = etayProc + xi[m]*VY[m];
  }

mpiReduce(phiProc[],phi[],processor(0),mpiSUM);
mpiReduce(etaxProc[],etax[],processor(0),mpiSUM);
mpiReduce(etayProc[],etay[],processor(0),mpiSUM);

if(mpirank==0){
  //Compute the reflection coefficient.
  cout<<"\n\n\n";
```

```
  cout<<"Non-Dim parameter Lc,Tc = "<<Lc<<","<<tc<<endl;
  phi=phi0+phi;
  complex Ref;
  getRefCoeff(4,phi,Ref);
  cout.precision(16);
  cout<<"Reflection Coefficient = "<<Ref<<endl<<"|R| = "<<abs(Ref)<<endl;

  //Write data to MATLAB
  //iter could be used to index the solution.

  int[int] Order=[1,1];
  savevtk(SolutionDir+"sol1_"+iter+".vtk",ThIce,[real(etax),real(etay)],[imag(etax),imag(etay)], dataname="ReDisp ImDisp",order=Order);
  savevtk(SolutionDir+"sol2_"+iter+".vtk",ThIce,[real(Sigma(etax,etay)[0]), real(Sigma(etax,etay)[1]), real(Sigma(etax,etay)[2])],[imag(Sigma(etax,etay)[0]), imag(Sigma(etax,etay)[1]), imag(Sigma(etax,etay)[2])], dataname="ReSigma ImSigma",order=Order);
  savevtk(SolutionDir+"solCavity"+iter+".vtk",ThCavity,[real(phi),imag(phi)],dataname="RePhi ImPhi",order=Order);
 }

if(mpirank==0)
  cout<<"\nP: "<<mpirank<<" T = "<<mpiWtime()-cpu<<"\t s"<<endl;
```

4.5 Vibration of Iceberg

For solving the problem of iceberg vibration, we solve the problem using meshes with different labels. This is a good example to show how the mesh variables can be altered by the user. The following code snippet produces a different numbering as shown in Figure 10.

```
int N1=getARGV("-N1",4);
int[int] lbl=[3,1,1,1];
ThIce=square(LL/tth*N1,N1,[LL*x,-dd+tth*y],label=lbl);

int N2=getARGV("-N2",4);
lbl=[1,5,3,4];
ThCavity=square(LL/(HH-dd)*N2,N2,[LL*x,-HH+(HH-dd)*y],label=lbl);
```

The free boundary condition produces three eigenvalues close to zero (which can be verified!) corresponding to the rigid body modes. To obtain the boundary condition on the outlet of the cavity region, we employ

```
matrix<complex> MQ2,MQ1;
getQphi(5,MQ2); //Outlet boundary
getQphi(4,MQ1); //Inlet Boundary
```

and can be added to stiffness matrix as shown

```
getLaplaceMat(0,0,0);
LHS=STIMA+(MQ1)+(MQ2); //Add the Q-Matrices
```

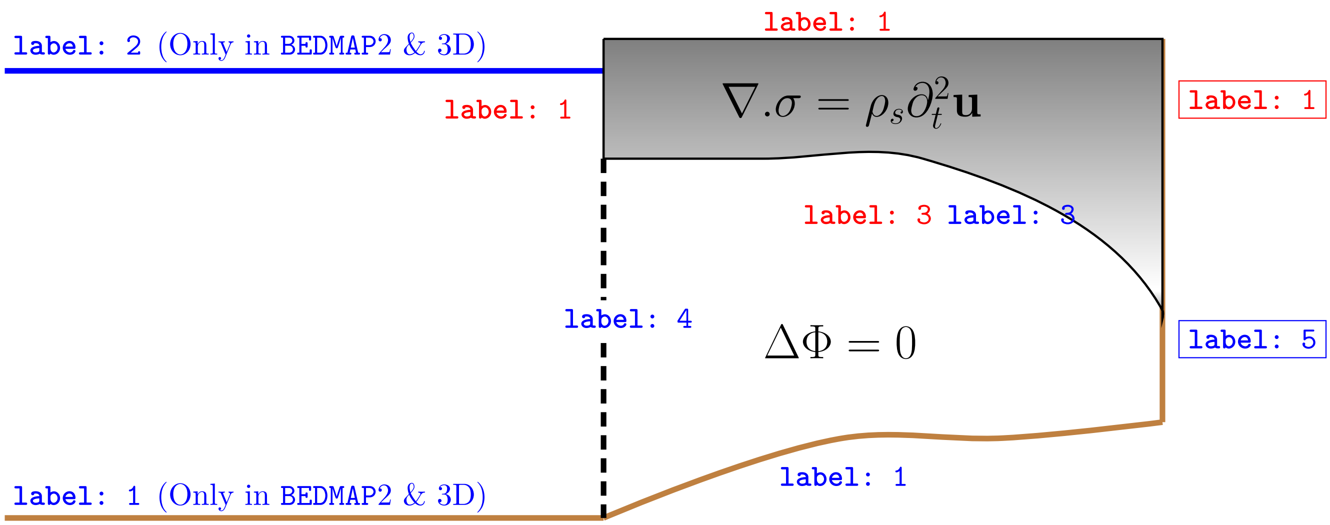


Figure 10: Geometry and labels for the cavity domain (blue) and the ice–shelf domain (red). The new labels are shown in square denoted free boundary condition for the ice–shelf and the outlet boundary for the cavity.

The full program to solve the iceberg vibration problem is given in **iceberg.edp**. To get sample results, run

```
>> ./genDir.sh 2_ICEBERG
>> mpirun -np 2 FreeFem++-mpi -v 0 iceberg.edp -N1 20 -N2 30 -Tr 40 -L 3000
   -H 2000 -h 200 -nev 8
```

which produces the following output

```
Dimension : 2
Dimension : 2
Splitting off ...
Done Radiation Potential 0
Done Radiation Potential 1
Done Radiation Potential 2
Done Radiation Potential 3
(0.475792,-0.647576)    (0.479653,0.352414)
Reflection Coefficient = 0.99999584
```

The first complex number is the reflection coefficient `R=(0.475792,-0.647576)` and the second is the transmission coefficient `T=(0.479653,0.352414)`. The last line prints the value `|R|^2+|T|^2` which is equal to `1` due to energy conservation. Sample solution for the iceberg vibration is shown in Figure 4.

5 Future Work

The first version of `iceFEM` is mostly **`FreeFem++`** and **`MATLAB`** stitched together. The idea was to use **`FreeFem++`** for easy implementation of the finite element aspect of the modelling and to use **`MATLAB`** or Python to perform interpolation and other operations. If you have any ideas/feature requests/bugs to report, please use the issue feature in the official repository located in GitHub at `https://github.com/Balaje/iceFem`. We hope to make the software better and more user friendly over time. Cheers!

References

[1] N J Balmforth and R V Craster. Ocean waves and ice sheets. *J. Fluid Mech.*, 395:89–124, 1999.

[2] L G Bennetts. *Wave scattering by ice sheets of varying thickness.* PhD thesis, University of Reading, UK, 2007.

[3] J. L. Black. Wave forces on vertical axisymmetric bodies. *J. Fluid Mech.*, 67:369–376, 1975.

[4] P. Brocklehurst, A. A. Korobkin, and E. I. Părău. Interaction of hydro-elastic waves with a vertical wall. *J. Eng. Math.*, 68(3):215–231, 2010.

[5] P. D. Bromirski, Z. Chen, R. A. Stephen, P. Gerstoft, D Arcas, A Diez, R. C. Aster, D. A. Wiens, and A. Nyblade. Tsunami and infragravity waves impacting Antarctic ice shelves. *J. Geophys. Res.*, pages 2612–2621, 2017.

[6] P. D Bromirski, A. Diez, P. Gerstoft, R. A. Stephen, T. Bolmer, D. A. Wiens, R. C Aster, and A. Nyblade. Ross ice shelf vibrations. *Geophys. Res. Lett.*, 42(18):7589–7597, 2015.

[7] P. D. Bromirski, O. V. Sergienko, and D. R. MacAyeal. Transoceanic infragravity waves impacting Antarctic ice shelves. *Geophys. Res. Lett.*, 37(2), 2010.

[8] P. D. Bromirski and R. A. Stephen. Response of the Ross Ice Shelf, Antarctica, to ocean gravity-wave forcing. *Ann. Glaciol.*, 53(60):163–172, 2012.

[9] Peter D Bromirski, Zhao Chen, Ralph A Stephen, Peter Gerstoft, D Arcas, Anja Diez, Richard C Aster, Douglas A Wiens, and Andrew Nyblade. Tsunami and infragravity waves impacting Antarctic ice shelves. *J. Geophys. Res. Oceans*, 2017.

[10] Peter D. Bromirski, O. V. Sergienko, and D. R. MacAyeal. Transoceanic infragravity waves impacting Antarctic ice shelves. *Geophys. Res. Lett.*, 37(2):1–6, 2010.

[11] K. M. Brunt, E. A. Okal, and D. R. MacAyeal. Antarctic ice-shelf calving triggered by the Honshu (Japan) earthquake and tsunami, March 2011. *J. Glaciol.*, 57(205):785–788, 2011.

[12] R.L. Burden and J.D. Faires. *Numerical Analysis.* Cengage Learning, 2010.

[13] L. M. Cathles, E. A. Okal, and D. R. MacAyeal. Seismic observations of sea swell on the floating Ross Ice Shelf, Antarctica. *J. Geophys. Res.*, 114(F2), 2009.

[14] Z Chen, PD Bromirski, P Gerstoft, RA Stephen, WS Lee, S Yun, SD Olinger, RC Aster, DA Wiens, and Andrew Arnold Nyblade. Ross Ice Shelf Icequakes Associated With Ocean Gravity Wave Activity. *Geophys. Res. Lett.*, 46:8893–8902, 2019.

[15] Zhao Chen, Peter D Bromirski, Peter Gerstoft, Ralph A Stephen, Douglas A Wiens, Richard C Aster, and Andrew A Nyblade. Ocean-excited plate waves in the Ross and Pine Island Glacier ice shelves. *J. Glaciol.*, 64:730–744, 2018.

[16] H. Chung and C. Fox. Calculation of wave–ice interaction using the Wiener–Hopf technique. *New Zealand J. Math.*, 31:1–18, 2002.

[17] D.V. Evans and T.V. Davies. *Wave-ice Interaction*. Report. Stevens Institute of Technology, 1968.

[18] C. Fox and V. A. Squire. On the Oblique Reflection and Transmission of Ocean Waves at Shore Fast Sea Ice. *Phil. Trans. R. Soc. Lond. A.*, 347:185–218, 1994.

[19] Colin Fox. A scaling law for the flexural motion of floating ice. In *IUTAM Symposium on Scaling Laws in Ice Mechanics and Ice Dynamics*, pages 135–148. Springer, 2001.

[20] Colin Fox and Vernon A Squire. Coupling between the ocean and an ice shelf. *Ann. Glaciol.*, 15:101–108, 1991.

[21] P. Fretwell, H. D. Pritchard, D. G. Vaughan, J. L. Bamber, N. E. Barrand, R. Bell, C. Bianchi, R. G. Bingham, D. D. Blankenship, G. Casassa, G. Catania, D. Callens, H. Conway, A. J. Cook, H. F. J. Corr, D. Damaske, V. Damm, F. Ferraccioli, R. Forsberg, S. Fujita, Y. Gim, P. Gogineni, J. A. Griggs, R. C. A. Hindmarsh, P. Holmlund, J. W. Holt, R. W. Jacobel, A. Jenkins, W. Jokat, T. Jordan, E. C. King, J. Kohler, W. Krabill, M. Riger-Kusk, K. A. Langley, G. Leitchenkov, C. Leuschen, B. P. Luyendyk, K. Matsuoka, J. Mouginot, F. O. Nitsche, Y. Nogi, O. A. Nost, S. V. Popov, E. Rignot, D. M. Rippin, A. Rivera, J. Roberts, N. Ross, M. J. Siegert, A. M. Smith, D. Steinhage, M. Studinger, B. Sun, B. K. Tinto, B. C. Welch, D. Wilson, D. A. Young, C. Xiangbin, and A. Zirizzotti. Bedmap2: improved ice bed, surface and thickness datasets for antarctica. *The Cryosphere*, 7(1):375–393, 2013.

[22] J.A. Griggs and J.L. Bamber. Antarctic ice-shelf thickness from satellite radar altimetry. *J. Glaciol.*, 57(203):485–498, 2011.

[23] F. Hecht. New development in FreeFem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[24] G Holdsworth and J Glynn. Iceberg calving from floating glaciers by a vibrating mechanism. *Nature*, 1978.

[25] Gerald Holdsworth and JE Glynn. A mechanism for the formation of large icebergs. *J. Geophys. Res. Oceans*, 86(C4):3210–3222, 1981.

[26] 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.

[27] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Dover Books on Mathematics Series. Dover Publications, Incorporated, 2012.

[28] D.S. Jones. *The Theory of Electromagnetism*. Pergamon Publications, 1964.

[29] B Kalyanaraman, LG Bennetts, Bishnu P Lamichhane, and MH Meylan. On the shallow-water limit for modelling ocean-wave induced ice-shelf vibrations. *Wave Motion*, 90:1 – 16, 2019.

[30] Balaje Kalyanaraman, Michael H Meylan, Luke G Bennetts, and Bishnu P Lamichhane. A coupled fluid-elasticity model for the wave forcing of an ice-shelf. *Journal of Fluids and Structures*, 97, 2020.

[31] A. E. Karperaki, K. A. Belibassakis, and T. K. Papathanasiou. Time-domain, shallow-water hydroelastic analysis of VLFS elastically connected to the seabed. *Marine Struct.*, 48:33–51, 2016.

[32] AL Kohout, MH Meylan, S Sakai, K Hanai, P Leman, and D Brossard. Linear water wave propagation through multiple floating elastic plates of variable properties. *J. Fluids Struct.*, 23(4):649–663, 2007.

[33] B. P. Lipovsky. Ice shelf rift propagation: stability, three-dimensional effects, and the role of marginal weakening. *The Cryosphere*, 14(5):1673–1683, 2020.

[34] Bradley Paul Lipovsky. Ice shelf rift propagation and the mechanics of wave-induced fracture. *Journal of Geophysical Research: Oceans*, 123(6):4014–4033, 2018.

[35] D. R. MacAyeal, E. A. Okal, R. C. Aster, J. N. Bassis, K. M. Brunt, L. M. Cathles, R. Drucker, H. A. Fricker, Y.-J. Kim, S. Martin, M. H. Okal, O. V. Sergienko, M. P. Sponsler, and J. E. Thom. Transoceanic wave propagation links iceberg calving margins of Antarctica with storms in tropics and Northern Hemisphere. *Geophys. Res. Lett.*, 33(17), 2006.

[36] R. A. Massom, T. A. Scambos, L. G. Bennetts, P. Reid, V. A. Squire, and S. E. Stammerjohn. Antarctic ice shelf disintegration triggered by sea ice loss and ocean swell. *Nature*, 558(7710):383–389, 2018.

[37] M. H. Meylan, L. G. Bennetts, R. J. Hosking, and E. Catt. On the calculation of normal modes of a coupled ice-shelf/sub-ice-shelf cavity system. *J. Glaciol.*, 63(240):751–754, 2017.

[38] Michael H. Meylan. A variational equation for the wave forcing of floating thin plates. *Appl. Ocean Res.*, 23(4):195–206, 2001.

[39] Michael H Meylan. Spectral solution of time-dependent shallow water hydroelasticity. *J. Fluid Mech.*, 454:387–402, 2002.

[40] John Miles and Freeman Gilbert. Scattering of gravity waves by a circular dock. *J. Fluid Mech.*, 34(4):783–793, 1968.

[41] F Montiel, LG Bennetts, and VA Squire. The transient response of floating elastic plates to wavemaker forcing in two dimensions. *J. Fluids Struct.*, 28:416–433, 2012.

[42] Y Namba, M Ohkusu, et al. Hydroelastic behavior of floating artificial islands in waves. *nt. J. Offshore Polar Eng.*, 9(01), 1999.

[43] T. K. Papathanasiou, A. Karperaki, E. E. Theotokoglou, and K. A. Belibassakis. A higher order FEM for time-domain hydroelastic analysis of large floating bodies in an inhomogeneous shallow water environment. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2173), 2015.

[44] T. K. Papathanasiou, A. E. Karperaki, E. E. Theotokoglou, and K. A. Belibassakis. Hydroelastic analysis of ice shelves under long wave excitation. *Nat. Hazards Earth Syst. Sci.*, 15(8):1851–1857, 2015.

[45] Theodosios K Papathanasiou, Angeliki E Karperaki, and Kostas A Belibassakis. On the resonant hydroelastic behaviour of ice shelves. *Ocean Modell.*, 133:11–26, 2019.

[46] T.K. Papathanasiou and K.A. Belibassakis. Resonances of enclosed shallow water basins with slender floating elastic bodies. *J. Fluids Struct.*, 82:538 – 558, 2018.

[47] M. A. Peter, M. H. Meylan, and H. Chung. Wave scattering by a circular elastic plate in water of finite depth: a closed form solution. *Int. J. Offshore Polar*, 14(2):81–85, 2004.

[48] Alexander B. Rabinovich. Spectral analysis of tsunami waves: Separation of source and topography effects. *Journal of Geophysical Research: Oceans*, 102(C6):12663–12676, 1997.

[49] Niels Reeh. On the calving of ice from floating glaciers and ice shelves. *J. Glaciol.*, 7(50):215–232, 1968.

[50] M.H. Sadd. *Elasticity: Theory, Applications, and Numerics*. Elsevier Science, 2010.

[51] O. V. Sergienko. Elastic response of floating glacier ice to impact of long-period ocean waves. *J. Geophys. Res.*, 115(4):1–16, 2010.

[52] O. V. Sergienko. Normal modes of a coupled ice-shelf / sub-ice-shelf cavity system. *J. Glaciol.*, 59(213):76–80, 2013.

[53] O. V. Sergienko. Behavior of flexural gravity waves on ice shelves: Application to the Ross Ice Shelf. *J. Geophys. Res.*, 122(8):6147–6164, 2017.

[54] O. V. Sergienko. Behavior of flexural gravity waves on ice shelves: Application to the Ross Ice Shelf. *J. Geophys. Res.*, pages 1–18, 2017.

[55] OV Sergienko. Behavior of flexural gravity waves on ice shelves: Application to the Ross Ice Shelf. *J. Geophys. Res. Oceans*, 2017.

[56] VA Squire, P Rottier, C Fox, Y Zhouwen, CL Tang, RH Preller, and W Huiding. A first look at some wave-ice interaction data from McMurdo Sound, Antarctica. In *Sea Ice Observations and Modelling—Ptroceedings of the 93's International Symposium on Sea Ice. Y. Zhouwen, CL Tang, RH Preller, W. Huiding eds. China Ocean Press, Beijing, PRC (1994)*, pages 19–33, 1994.

[57] J.J. Stoker. *Water waves: the mathematical theory with applications*. Pure Appl. Math. Interscience Publishers, 1957.

[58] Izolda V. Sturova. Time-dependent response of a heterogeneous elastic plate floating on shallow water of variable depth. *J. Fluid Mech.*, 637:305–325, 2009.

[59] L. A. Tkacheva. The diffraction of surface waves by a floating elastic plate at oblique incidence. *J. Appl. Math. Mech.*, 68(3):425–436, 2004.

[60] David G Vaughan. Tidal flexure at ice shelf margins. *J. Geophys. Res. Solid Earth*, 100(B4):6213–6224, 1995.

[61] OG Vinogradov and G Holdsworth. Oscillation of a floating glacier tongue. *Cold Reg. Sci. Technol.*, 10(3):263–271, 1985.

[62] CC Walker, JN Bassis, HA Fricker, and RJ Czerwinski. Structural and environmental controls on antarctic ice shelf rift propagation inferred from satellite monitoring. *Journal of Geophysical Research: Earth Surface*, 118(4):2354–2364, 2013.

[63] Elias Wegert. *Visual complex functions: an introduction with phase portraits.* Springer Science & Business Media, 2012.

[64] T.D.C. Williams. *Reflections on ice : scattering of flexural gravity waves by irregularities in Arctic and Antarctic ice sheets.* PhD thesis, University of Otago, 2006.

[65] Chong Wu, Eiichi Watanabe, and Tomoaki Utsunomiya. An eigenfunction expansion-matching method for analyzing the wave-induced responses of an elastic floating plate. *Appl. Ocean Res.*, 17(5):301–310, 1995.