A brief tutorial of FELIX-S

All files are included in the folder “FELIXS”. The “$dir” below refers to the directory of your “FELIXS” folder.

Meshing

Install Ann

Ann is a component required by the 2D mesh generation program. It can be installed as follows:

```
cd $dir/Mesh/ann/ann_1.1.2
make linux-g++
```

Install the 2D mesh generation program

Then the 2D mesh generation program, writtern by Lili Ju, can be easily install as follows:

```
cd $dir/Mesh/MESHGEN
make
```

You may need to install the libx11-dev package to get this done.

Install the 3D mesh generation program

This program was written by Wei Leng, aiming to build a 3D tetrahedral mesh from the 2D triangle mesh files. To compile this program, we must have an installed PHG ready. We first go to the “meshinit” folder and “make”‘.

```
cd $dir/Mesh/meshinit
make
```

This will compile the binary file “tri2prism”. Note that you may change the makefile with your own correct installation directory of the PHG and MPI program (for example, Mpich2). Another binary, “read_mixed_mesh”, is also needed. Put the file “read_mixed_mesh.c” into the directory $PHG/examples and compile it.

```
cp read_mixed_mesh.c $PHG/examples
cd $PHG/examples
make read_mixed_mesh
cp read_mixed_mesh $dir/Mesh/meshinit
```

Now we are good to make our 3D meshes!

Generate a 3D mesh

For now, PHG supports tetraheral mesh only. In order to get a 3D tetraheral mesh, we first need to get a 2D triangle mesh. The 2D mesh generation program is based on Triangle that requires a “.poly” file as the input. The “.poly” file contains 2D information of nodes and lines a mesh requires. Here I provide two MATLAB/Octave scripts that automatically generate the “mesh.poly” file for the meshes of rectangular regions. See the following example

```
tong@think:~/MESHGEN$ octave
octave:1> get_geo_progress_poly
octave:2> exit
tong@think:~/MESHGEN$ ./mesh_init -f "mesh.poly"
tong@think:~/MESHGEN$ ./process.sh
```

Note that you can decide the vertical layer numbers you want to use in the file “meshinit/layers.dat”. You can also set your own output folder where the 3D mesh files would go. Make sure you have Numpy ready in your system as there is a simple Python script handling some meshing corrections. Keep in mind that the 3D mesh generated here is uniform in z, i.e., $z\in[0,1]$. A real 3D geometry is set in the FELIX-S code.

Install FELIX-S

The FELIX-S code relies heavily on Petsc and SuperLU. Make sure they are installed properly before installing PHG. An easy way of including SuperLU is to add the “-download-superlu_dist” option when configuring Petsc. After setting environmental variables PETSC_DIR and PETSC_ARCH, we can install PHG by typing the following commands in the PHG home folder (phg-0.8.5):

```
./configure
```

Normally PHG will just find out the Petsc and SuperLU you just installed. Otherwise you may want to try other versions of Petsc. On Hopper it is easy to include Petsc and Superlu. Since the Petsc on Hopper is compiled with the Mumps and Superlu components, we can just type:

```
module load petsc
```

Note that we can just use the PGI compile on hopper. Make sure PHG find the Petsc and Superlu in the configuration step. After we successfully installed PHG, we are ready to complete the FELIX-S installation. Change directory to FELIX-S, change the “PHG_HOME” at the beginning of the makefile to the dir you install it, and type the command:

```
make
```

The binary file “ice-flow” we get in the current folder is the exec file we want!

An overview of FELIX-S usage

Using FELIX-S is actually quite straightforward. For example, you can just type the following commands in the FELIX-S folder if you want to run the code parallelly with 4 processes on your laptop:

```
mpirun -np 4 ./ice-flow
```

On Hopper it is

```
aprun -n 4 ./ice-flow
```

Before FELIX-S runs, it will first check if there is any option file with the same name as the binary file, i.e., “ice-flow.options” in this case. If so, FELIX-S will read the options we set in this file and then change its behaviour accordingly. Here is an example of “ice-flow.options”:

```
-verbosity 0
    # "-" means we are setting this option, "+" means we are not.

+log_file log/ice.log
    # log output

-partitioner user
    # partition method

-periodicity 2
    # set periodicity along y

-mesh_file geo_files/poly25/prism.neu
-tria_file geo_files/poly25/trigs.dat
-vert_file geo_files/poly25/nodes.dat
-layer_file geo_files/poly25/layers.dat
-nodeZ_file geo_files/poly25/node.Z
    # mesh files

-x_txt_file geo_files/poly25/xgrid.txt
-y_txt_file geo_files/poly25/ygrid.txt
    # x y grid coord files (optional)

-thk_txt_file geo_files/H_stnd_new_contact8_dH_00202.txt
-sur_txt_file geo_files/s_stnd_new_contact8_dH_00202.txt
    # 1D geometry files

-utype P2
-ptype P1
-T_type P1
    # elment types

+solve_temp
    # temperature model

-init_temp_type 1
    # temperature field initialization method

-solve_height
    # surface updating model

-height_scheme 1
    # the way we update surface

-start_const_vis
    # start the model from a constant viscosity

-options_file options/solver-ASM-mumps.options
    # solver options

-Stokes_opts "-solver petsc -solver\_rtol 1e-10 -solver_maxit 1000"
    # solver options

-max_non_step 100
```

```
    # max iterations for the velocity calculation

-min_non_step 5
    # min iterations for the velocity calculation

-newton_start 10
    # switch to the Newton solver at step 10

-u_tol0 1e-3
    # firt tolerance for velocity convergence

-u_tol 5e-1
    # second tolerance for velocity convergence

-s_tol 1e-5
    # tolerance for checking steady state

-dt 1
    # time step

-time_end 1000
    # max time period

-max_time_step 10000
    # max time step.
    # For example, time_end = 100, dt = 0.1, max_time_step = 1000

-time_start 0
    # starting time

-step_span 2
    # save results every 2 steps

+resume
    # resume from a previous step

-step_span_resume 20
    # save the results for resuming every 20 steps

-record
    # record the data for resuming
```

There are three folders here for keeping model outputs, OUTPUT, dH_OUTPUT and MASK_OUTPUT. The OUTPUT folder is for previous model geometries and velocity/pressure results in case we want to resume FELIX-S from some time step before instead of starting over from the very beginning. The dH_OUTPUT folder is for keeping the free surface elevation updates. The MASK_OUTPUT is for saving the results of element masks, water pressure and nodal loads, etc. You can certainly do different ways as you like.

Example 1: A simple slab with frozen bed

The slab geometry is constructed through the function “func_ice_slab” in the file “TEST.c”. We then need to set the boundaries that use the Dirichlet BC in the function “set_boundary_mask” in the file “TEST.c”. What we do is setting an array called “DB_masks[3]”. For example, if we set

```
DB_MASKS[3] = {BC_BOTTOM, BC_BOTTOM, BC_BOTTOM},
```

all the 3 components of ice velocities {u, v, w} are set to 0 on the boundaries that have the mask of “BC_BOTTOM”.

Finally, we must change the variable “USE_SLIDING_BC” to 0 in the header file “ins.h” and recompile the code.

```
#define USE_SLIDING_BC 0
```

Example 2: A simple slab with slip bed

Similar to the frozen bed case, we need to use a proper “DB_MASKS”. A slip bed requires that the ice flow doesn’t penetrate into the bedrock, i.e., zero/free ice flow in the direction perpendicular/tangent to the bedrock. A coordinate rotation at the ice bottom is operated in the code. The first component of the rotated coordinate, x’, is treated like the perpendicular direction, and the other two components, y’ and z’, denote the tangent directions. Thus, we set

```
DB_MASKS[3] = {BC_BOTTOM, 0, 0}.
```

Again, the “USE_SLIDING_BC” in the header file “ins.h” should be set to 1.

```
#define USE_SLIDING_BC 1
```

There are currently 2 sliding laws implemented in FELIX-S, a simple linear friction law and the non-linear Weertman type sliding law, which could be freely switched in the “TEST.options” file by changing the value of the variable “slip_condition” (0 and 1 for linear and non-linear law, respectively).

```
-slip_condition 0 or 1
```

The sliding parameters are defined in the function "func_beta" in the file "TEST.c" (linear) and in the file "ins-system.c" (non-linear) (This still needs some improvements for easy use).

Example 3: Temperature model coupling

The temperature model could be easily coupled by setting the variable "-solve_temp" (use +solve_temp to disable).

```
-solve_temp
```

or

```
+solve_temp
```

To use the temperature model, we may set up the surface temperature distribution in the function "func_T" in the file "ins-main.c". In addition, the geothermal heat flux value is given in the header file "parameters.h".

```
#define GEOTHE_FLUX 4.2e-2
```

Note that this module was first created by Wei Leng. I then modified it at some places in another seperate model last year. I'll test/implement it at some point.

Example 4: Ice surface updating

The current code is using a 2D finite element method to update the surface elevations in prognostic simulations.
To enbale/disable this funcion, we can simply set "-/+solve_height" in the "TEST.options" file.

```
-solve_height
```

or

```
+solve_height
```

Example 5: Ice sheet/shelf system modeling

To simulate the ice sheet-shelf system (marine ice sheet), we just need to properly set up the boundary masks at the ice bottom.
The code will then automatically apply the BC at the ice shelf bottom and compute the results.

The ice shelf masks could be assigned in the function "func_ice_shelf_mask" in the "TEST.c". For example,

```
void
func_ice_shelf_mask(FLOAT x, FLOAT y, FLOAT z, FLOAT *ice_shelf_mask)
{
    x *= 1000;
    y *= 1000;

    if (x>519200){
        *ice_shelf_mask = 1;
    }
    else
        *ice_shelf_mask = 0;
}
```

The region with x > 519.2 km will be masked as floating.