Ice Sheet System Model 2017 (4.12)
User Guide

Authors:

Mathieu Morlighem[1]
Hélène Seroussi[2]
Éric Larour[2]
Nicole Schlegel[2]
Chris Borstad[2]
Basile de Fleurian[1]
Surendra Adhikari[2]
Johannes Bondzio[3]

[1]University of California, Irvine, Department of Earth System Science, Croul Hall, Irvine, CA 92697-3100, USA

[2] Jet Propulsion Laboratory - Caltech, 4800 Oak Grove Drive, Pasadena, CA 91109 USA

[3] Alfred Wegener Institute, Bremerhaven, Germany

May 19, 2017

Contents

Chapter 1

Download

1.1 Introduction

1.1.1 Binaries

The easiest way to install ISSM is to download the pre-compiled binaries. No need to compile the code, just open the compressed file.

1.1.2 Source Code

If you would like to install ISSM from source, you will need to download the source code first. The source code of ISSM (see License below) is available from an SVN repository. In order to fetch a version of the code, users will need to install SVN on their machine (It is usually installed by default on most platforms). Once SVN has been installed, ISSM can be downloaded by the following command:

```
$ svn --username anon --password anon checkout http://issm.ess.uci.edu/svn/issm/issm/trunk
```

This command will download the lastest version of ISSM from the repository, onto the current local directory. Users are free to choose whichever location they want.

If you downloaded the source code, you need to compile and install ISSM. Compilation of the ISSM source code is theoretically possible on any platform. It has been succesfully carried out on Linux (RedHat and ubuntu), Windows (9 and 10) and macOS (snow-leopard, Lion, Mountain Lion, Mavericks, Yosemite and El Capitan). Here are some instructions to compile and install ISSM from the source code:

- Linux/Mac
- Windows (under developement)
- Installation with AD capability (under developement)

Compilation is a more involved process, which is not recommended for beginners or casual users.

1.1.3 Become an ISSM developer !

`anon` users have read-only access. Users willing to actively participate in the development of ISSM can contact us.

1.1.4 License

1.2 Binaries

The binaries can be downloaded on the ISSM website.

This is the easiest way to install ISSM. No need to compile the code, just open the compressed file and ISSM is installed!

You will then need to open MATLAB and add ISSM tools to MATLAB's path.

1.3 Source installation of ISSM on UN*X systems

1.3.1 Environment variables

The compilation of ISSM requires several environment variables. Add the following lines in your shell environment script:

- `~/.bashrc` (or `~/.bash_profile` on Mac)

```
#ISSM
export ISSM_DIR=ISSMPATH
source $ISSM_DIR/etc/environment.sh
```

- `~/.cshrc`

```
#ISSM
setenv ISSM_DIR ISSMPATH
source $ISSM_DIR/etc/environment.csh
```

Where ISSMPATH is the path of ISSM main directory (ex: /home/user1/svn/issm/trunk).

1.3.2 macOS

In order to install ISSM on macOS, your system must have Xcode, which can be installed from the Mac App Store. You also need the Command Line Tools, which can be installed using

```
xcode-select --install
```

Unfortunately Xcode doesn't contain a Fortran compiler, which is required for some packages of PETSc. Therefore you will have to install one. GFortran Binaries are available on the GCC Wiki at the following address: http://gcc.gnu.org/wiki/GFortranBinaries

1.3.3 External packages installation

All ISSM external packages are located in the directory `externalpackages` of the trunk. Several packages may be installed depending on what users want to do. At least the following packages must be installed:

- autotools
- mpich (to be installed first, recommended version: 3.0)
- cmake
- petsc (after mpich and cmake, recommended version: 3.6)
- m1qn3
- triangle

To run ISSM with a Python interface, the following packages must also be installed:

- python (recommended version: 2.7)
- nose
- blas
- lapack
- git
- numpy

- cython
- scipy
- hdf5
- netcdf
- netcdf-python

For each library, different installation scripts may exist depending on the version to be installed and the machine operating system. Users should use the installation script that is the closest to their environment. For example:

```
$ cd $ISSM_DIR/externalpackages/mpich
$ ./install-3.0-macosx64.sh
```

There is no guarantee the compilation will work on all systems. Some tweaking of the installation script may be involved. Especially, the configuration part of the install. (See compilation troubleshooting)

Note: after the installation of each package, one should source the environment:

```
$ source $ISSM_DIR/etc/environment.sh
```

1.3.4 ISSM compilation

ISSM relies on autotools to make source-code packages portable to many Unix-like systems. The last step consists in generating the Makefiles needed to compile ISSM. First, ISSM must be reconfigured:

```
$ cd $ISSM_DIR
$ autoreconf -ivf
```

ISSM can then be configured. Here is an example of configuration script for a macOS machine with the matlab interface (you will need to change the path to MATLAB. We are using `/Applications/MATLAB_R2015b.app/` as an example).

```
./configure \
--prefix=$ISSM_DIR \
--with-matlab-dir="/Applications/MATLAB_R2015b.app/" \
--with-triangle-dir="$ISSM_DIR/externalpackages/triangle/install" \
--with-mpi-include="$ISSM_DIR/externalpackages/mpich/install/include"  \
--with-mpi-libflags="-L$ISSM_DIR/externalpackages/mpich/install/lib/ -lmpich" \
--with-petsc-dir="$ISSM_DIR/externalpackages/petsc/install" \
--with-metis-dir="$ISSM_DIR/externalpackages/petsc/install" \
--with-blas-lapack-dir="$ISSM_DIR/externalpackages/petsc/install" \
--with-scalapack-dir="$ISSM_DIR/externalpackages/petsc/install/" \
--with-mumps-dir="$ISSM_DIR/externalpackages/petsc/install/" \
--with-m1qn3-dir="$ISSM_DIR/externalpackages/m1qn3/install" \
--with-numthreads=2
```

If you get the following error: `ld: library not found for -lflapack`, remove the line `--with-blas-lapack-dir` (generally on macOS).

For the python interface, it should look like:

```
./configure \
--prefix="$ISSM_DIR" \
--with-triangle-dir="$ISSM_DIR/externalpackages/triangle/install" \
--with-python-dir="$ISSM_DIR/externalpackages/python/install" \
--with-python-numpy-dir="$ISSM_DIR/externalpackages/python/install/lib/python2.7/site-packages/numpy/
--with-mpi-include="$ISSM_DIR/externalpackages/mpich/install/include"  \
--with-mpi-libflags="-L$ISSM_DIR/externalpackages/mpich/install/lib -lpmpich -lmpich -lmpl" \
--with-petsc-dir="$ISSM_DIR/externalpackages/petsc/install" \
--with-scalapack-dir="$ISSM_DIR/externalpackages/petsc/install/" \
--with-mumps-dir="$ISSM_DIR/externalpackages/petsc/install/" \
--with-metis-dir="$ISSM_DIR/externalpackages/petsc/install/" \
--with-m1qn3-dir="$ISSM_DIR/externalpackages/m1qn3/install" \
--with-numthreads=2
```

The configuration file should be called `configure.sh` and placed in `$ISSM_DIR`. For most platforms, you might need to make some adjustments to the configuration options mentioned above.

If the configuration went without any error, ISSM can now be compiled:

```
$ cd $ISSM_DIR
$ make
$ make install
```

ISSM installation is done!

1.4 Source installation of ISSM on Windows (under developement)

1.4.1 Win10

Be sure to use an account name that does not have spaces in it, as this could be an issue with Cygwin.

1.4.2 Development Environment

Before you can begin to build ISSM you will need to configure your development environment. This will require you install the following:

1. MATLAB
2. Cygwin
3. Visual Studio

1.4.2.1 Installing MATLAB

Installing MATLAB is fairly straightforward. However, it can be beneficial to install in a directory chain that contains no spaces.

Keep in mind that you can use older versions of MATLAB, but we have only tested using R2015a and R2016a.

1.4.2.2 Cygwin

You will need Cygwin installed on your Windows platform to manage the compilation. Cygwin emulates unix behaviour on windows machines. The compilation will still be carried out by the windows SDK compiler, but the environment driving the compilation will be the unix-like Cygwin. When you download Cygwin, make sure that you install it in the `C:\Cygwin` directory.

You will find the Cygwin setup executable on the www.cygwin.com webpage. Here is a link to it: http://cygwin.com/setup. Download this file to `C:\Cygwin` and be sure to reuse it when you want to update your current Cygwin installation. Do not download setup.exe twice!

You will need the following packages downloaded to carry out the compilation of ISSM successfully, so be sure to include them in your Cygwin install:

- subversion
- vim (or any other editor)
- patch
- make: The GNU version of the 'make' utility
- python: Python language interpreter
- unzip

Don't worry if you forget any packages, as you can always close Cygwin and run the installer again if you find that you need to install other packages.

1.4.3 Visual Studio

Technically, you should be able to compile ISSM with any compiler set. However, MATLAB only officially supports Microsoft's compilers. As such, you will need to install Microsoft's Visual Studio in order to continue through this guide.

Most versions of Visual Studio will probably work, but we recommend installing the newest community version, which can be found here: `https://www.visualstudio.com/`. As of June 2016, the version is 14.0 a.k.a Community 2015.

Keep in mind that Visual Studio is a integrated development environment (IDE) that is used to develop for various environments and in different languages. As such, the default installation will not suffice.

You need to make sure that you select C++ as a language and the Windows 10 Software Development Kit (SDK).

Follow Figure 1 if you need help selecting the Visual Studio optional packages:

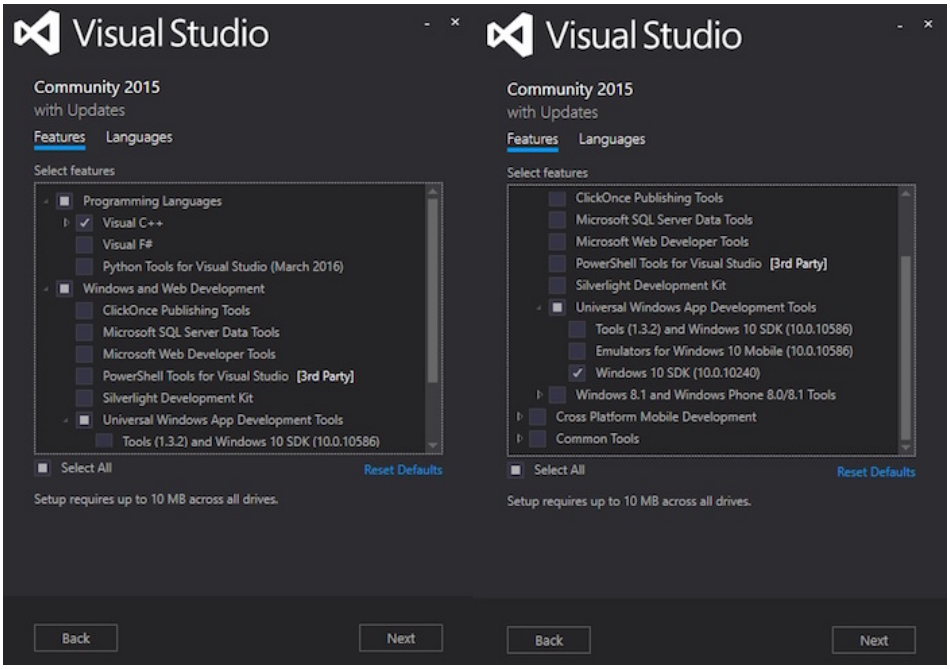


Figure 1.1: Figure 1

1.4.4 Building ISSM

1.4.4.1 Downloading ISSM

Download issm into your Cygwin home directory (note: this is different from your Windows home directory).

In Cygwin, run the following commands:

```
mkdir issm && cd issm
svn co --username anon --password anon http://issm.ess.uci.edu/svn/issm/issm/trunk
```

Once finished, it is now time to set important environment variables. You can do this by setting and exporting the environment variables at the Cygwin command line, or by modifying your '.bashrc', or whatever appropriate shell you are using. (Note: if you export from the command line, then you will have to do this every time you start a new session). In either case, the following lines should be executed or added to your specific rc file:

```
export ISSM_DIR="$HOME/issm/trunk"
export MATLAB_DIR='/cygdrive/c/Program\ Files\ \(x86\)/MATLAB/R2016a'
```

The above lines assume that you have installed and downloaded both MATLAB and ISSM into the default locations detailed earlier. If you have deviated from these instructions then simply modify the paths to their appropriate locations.

Finally, you need to source two scripts. This can be accomplished by executing the following commands:

```
source $ISSM_DIR/etc/environment.sh
source $ISSM_DIR/externalpackages/windows/configs/sdk10.0-win64.sh
```

Again, this assumes that you have installed everything in the default locations. If you did not, then you will have to modify 'sdk10.0-win64.sh' to reflect the different directories.

1.4.4.2 Checking Your Build Environment

Before trying to compile the externalpackages, you need to be sure that the Microsoft Visual Studio compiler 'cl.exe' is working. To make sure, create a simple hello world example:

`hello.cpp`

```
#include <iostream>
int main(){
std::cout << "Hello World!\n";
return 0;
}
```

Try to compile it by doing: `cl hello.cpp`

If this does not work, do not go any further! Unfortunately, this is a complicated build as compared to other platforms, so any number of things can go wrong. Make sure the following things are correct:

1. Your environment variables are set correctly.
2. You sourced the two scripts in the right order.
3. Cygwin, MATLAB, Visual Studio are installed in the default directories of you modified the appropriate variables/scripts to reflect these differences
4. Visual Studio was installed with C++ support and with SDK 10.0.10240

If everything is correct, but you sill cannot get the hello world example to work, then please contact us on our forum or by email and we'll be happy to help!

1.4.4.3 Installing External Packages

Now you can start installing the following external packages. This is done by changing directory into the appropriate directory within 'externalpackages' and running the appropriate scripts.

- For autotools, use install-win.sh
- For Petsc, use install-3.6-win10.sh
- For metis, use install-4.0-win10.sh
- For triangle, use install-win10.sh

1.4.4.4 Building ISSM

Once these externalpackages have been compiled, you can prepare the build system:

```
$ cd $ISSM_DIR
$ autoreconf -ivf
```

ISSM can then be configured. Here is an example of configuration script for windows 10:

```
 ./configure --prefix=$ISSM_DIR \
--with-vendor=MSVC-Win64  \
--with-cxxoptflags='' \
   --disable-static \
--enable-standalone-libraries \
--with-fortran=no  \
--without-Gia \
--without-kriging \
--without-kml \
--with-matlab-dir=$MATLAB_DIR \
--with-triangle-dir="$ISSM_DIR/externalpackages/triangle/install" \
--with-petsc-dir="$ISSM_DIR/externalpackages/petsc/install" \
--with-metis-dir=$ISSM_DIR/externalpackages/metis/install \
--with-blas-lapack-dir=$ISSM_DIR/externalpackages/petsc/install/lib/  \
--with-mpi-libdir="$ISSM_DIR/externalpackages/petsc/install/lib" \
--with-mpi-libflags="-Wl,libpetsc.lib" \
--with-mpi-include="$ISSM_DIR/externalpackages/petsc/install/include/petsc/mpiuni"
```

The configuration file must be placed in `$ISSM_DIR` and named `configure.sh`.

Finally, you can configure, make and make install:

```
./configure.sh
make
make install
```

That should complete the installation!

1.4.5 How to Setup a Cron Job

Download exim and cron from cygwin setup. Run `exim-config` (don't forget to launch your cygwin terminal as Administrator). If asked for a daemon name, type ntsec. Also, if cron complains about sendmail, symlink `/usr/lib/sendmail` to `/usr/sbin/sendmail`. Then link `/usr/sbin/sendmail` to `/usr/bin/exim`. Alternatively, if you have an smtp server, you can download and configure ssmtp. Run `ssmtp-config` as Administrator and link `/usr/sbin/sendmail` to `/usr/bin/ssmtp`. After the email has been configured, run `cron-config` as Administrator.

1.5 Source installation of ISSM with AD capability (under developement)

Automatic Differentiation is only supported for Linux and Mac. Please follow the steps of a regular installation first and make sure it is working before adding AD.

1.5.1 External packages installation

The following additional external packages need to be installed:

- adjoinablempi

- adolc

Note: after the installation of each package, one should source the environment:

```
$ source $ISSM_DIR/etc/environment.sh
```

1.5.2 ISSM compilation

The configuration script of ISSM needs to include the following additional options:

```
--with-adolc-dir=$ISSM_DIR/externalpackages/adolc/install \
--with-ampi-dir=$ISSM_DIR/externalpackages/adjoinablempi/install \
```

As of today, AD is not supported with PETSc, but you will need some of PETSc's packages to solve linear systems so you should take out the `--with-petsc-dir` line of the configuration script but keep the other PETSc related lines such as MUMPS or scalapack. You will also need to deactivate the kriging capability of ISSM with the `--without-kriging` option. Here is an example of configuration script:

```
./configure \
--disable-static \
--without-kriging \
--without-kml \
--without-Gia \
--prefix=$ISSM_DIR \
--with-matlab-dir="/Applications/MATLAB_R2015b.app/" \
--with-triangle-dir="$ISSM_DIR/externalpackages/triangle/install" \
--with-mpi-include="$ISSM_DIR/externalpackages/mpich/install/include"  \
--with-mpi-libflags="-L$ISSM_DIR/externalpackages/mpich/install/lib/ -lmpich" \
--with-metis-dir="$ISSM_DIR/externalpackages/petsc/install" \
--with-blas-lapack-dir="$ISSM_DIR/externalpackages/petsc/install" \
--with-scalapack-dir="$ISSM_DIR/externalpackages/petsc/install/" \
--with-mumps-dir="$ISSM_DIR/externalpackages/petsc/install/" \
--with-adolc-dir=$ISSM_DIR/externalpackages/adolc/install \
--with-ampi-dir=$ISSM_DIR/externalpackages/adjoinablempi/install \
--with-numthreads=2
```

You can now reconfigure and recompile ISSM, it is now fully adjoinable.

Chapter 2

Getting started

2.1 Loading ISSM tools

By default MATLAB and Python cannot locate ISSM functions. Therefore you must execute the `addpath` command within MATLAB to change the so-called matlabpath appropriately (and `sys.path.append` in Python).

2.1.1 MATLAB

Assuming that ISSM is installed in `/usr/local/issm/trunk`, you should do:

```
addpath /usr/local/issm/trunk/bin/ /usr/local/issm/trunk/lib/
```

You can verify that ISSM works by executing

```
issmversion
```

You should get a message similar to this:

```
Ice Sheet System Model (ISSM) Version 4.4
(website: http://issm.jpl.nasa.gov contact:issm@jpl.nasa.gov)

Build date: Wed Sep 18 14:00:06 PDT 2013
Copyright (c) 2009-2013 California Institute of Technology

    to get started type: issmdoc
```

Normally, you will have to enter the `addpath` command every time MATLAB is started. This can be avoided if the `addpath` command is added to an alias:

```
alias matlab='matlab -r "addpath $ISSM_DIR/bin $ISSM_DIR/lib"'
```

2.1.2 Python

In Python, you can add ISSM's function to the current path with the following command:

```
import sys
sys.path.append('/usr/local/issm/trunk/bin/')
sys.path.append('/usr/local/issm/trunk/lib/')
```

You can verify that ISSM works by executing

```
from issmversion import issmversion
```

You should get a message similar to this:

```
Ice Sheet System Model (ISSM) Version 4.4
(website: http://issm.jpl.nasa.gov contact:issm@jpl.nasa.gov)

Build date: Wed Sep 18 14:00:06 PDT 2013
Copyright (c) 2009-2013 California Institute of Technology
```

2.1.3 Developers

Developers use the MATLAB and Python scripts in `$ISSM_DIR/src/m` instead of `$ISSM_DIR/bin`. The path needs to be different.

In MATLAB:

```
addpath /usr/local/issm/trunk/src/m/dev
devpath;
```

or,

```
matlab -nodesktop -nosplash -r "addpath $ISSM_DIR/src/m/dev; devpath;"
```

In Python:

```
export PYTHONSTARTUP=$ISSM_DIR/src/m/dev/devpath.py
```

or for IPython users:

```
ipython -i $ISSM_DIR/src/m/dev/devpath.py
```

In the configuration script, the following option should be added to prevent MATLAB and Python scripts from being added to `$ISSM_DIR/bin`

```
--enable-development
```

2.2 Model class

2.2.1 MATLAB's model object

All the data belonging to a model (geometry, node coordinates, results, etc.) is held in the same MATLAB/Python object `model`. To create a new model, one can type the following command in MATLAB's Command window:

```
>> md=model;
```

This will create a new model named "md" whose class is "model". The information contained in the model "md" are grouped by class, that contain fields related to a particular aspect of the model (e.g. mesh, material properties, friction, stressbalance solution, results of the runs, etc.) When one creates a new model, all these fields are empty or `NaN` (not a number), but "md" is ready to be used as a model. The list of these classes is displayed when typing:

```
>> md
 md =

               mesh : [1x1 mesh]             -- mesh properties
               mask : [1x1 mask]             -- defines grounded and floating elements
           geometry : [1x1 geometry]         -- surface elevation, bedrock topography, ice thicknes.
          constants : [1x1 constants]        -- physical constants
    surfaceforcings : [1x1 surfaceforcings]  -- surface forcings
      basalforcings : [1x1 basalforcings]    -- bed forcings
          materials : [1x1 materials]        -- material properties
           friction : [1x1 friction]         -- basal friction/drag properties
       flowequation : [1x1 flowequation]     -- flow equations
       timestepping : [1x1 timestepping]     -- time stepping for transient models
     initialization : [1x1 initialization]   -- initial guess/state
              rifts : [1x1 rifts]            -- rifts properties
              debug : [1x1 debug]            -- debugging tools (valgrind, gprof)
            verbose : [1x1 verbose]          -- verbosity level in solve
           settings : [1x1 settings]         -- settings properties
             solver : [1x1 solver]           -- PETSc options for each solution
            cluster : [1x1 none]             -- cluster parameters (number of cpus...)
   balancethickness : [1x1 balancethickness] -- parameters for balancethickness solution
      stressbalance : [1x1 stressbalance]    -- parameters for stressbalance solution
      groundingline : [1x1 groundingline]    -- parameters for groundingline solution
          hydrology : [1x1 hydrology]        -- parameters for hydrology solution
      masstransport : [1x1 masstransport]    -- parameters for masstransport solution
            thermal : [1x1 thermal]          -- parameters for thermal solution
        steadystate : [1x1 steadystate]      -- parameters for steadystate solution
          transient : [1x1 transient]        -- parameters for transient solution
           autodiff : [1x1 autodiff]         -- automatic differenciation parameters
              flaim : [1x1 flaim]            -- flaim parameters
          inversion : [1x1 inversion]        -- parameters for inverse methods
                qmu : [1x1 qmu]              -- dakota properties
            results : [1x1 struct]           -- model results
       radaroverlay : [1x1 radaroverlay]     -- radar image for plot overlay
      miscellaneous : [1x1 miscellaneous]    -- miscellaneous fields
```

2.2.2 Saving/loading a model

One can save the model with all its fields so that the saved file contains all the information in the model, type the following command:

```
>> save squaremodel md
```

This will create a file `squaremodel.mat` made from the model md. To load this file, type:

```
>> loadmodel squaremodel
```

the loaded model will be named `md`.

2.3 Square ice shelf tutorial

This is an example of velocity computation in steady state for a square ice shelf. First thing, go to the `trunk/` and launch MATLAB. Then go to `examples/SquareIceshelf/`:

```
$ cd $ISSM_DIR
$ matlab
>> cd examples/SquareIceShelf/
```

Then, at the MATLAB prompt, you can create an empty model structure by typing:

```
>> md=model;
```

Create a mesh of the domain outline with a resolution of 50,000 meters:

```
>> md=triangle(md,'DomainOutline.exp',50000);
```

Define the glacier system as an ice shelf (no island):

```
>> md=setmask(md,'all','');
```

Parameterize the model with the file `Square.par`:

```
>> md=parameterize(md,'Square.par');
```

Define all elements as SSA:

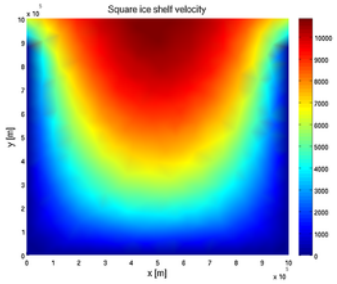
```
>> md=setflowequation(md,'SSA','all');
```

Compute the velocity field of the ice shelf:

```
>> md=solve(md,StressbalanceSolutionEnum);
```

Visualize the velocity:

```
>> plotmodel(md,'data',md.results.StressbalanceSolution.Vel);
```



Chapter 3

Tutorials

3.1 Dataset download

To run the tutorials, you will need to download the following datasets and put them in `trunk/examples/Data`:

- Square ice shelf dataset
- SeaRISE Antarctica v0.75
- SeaRISE Greenland dev1.2
- MEaSUREs Antarctic velocities
- Pine Island ice thickness cross overs (dakota)
- Jason Box's SMB data
- Jakobshavn Isbrae bed map (we only need Jakobshavn_2008_2011_Composite_XYZGrid.txt)

3.2 Mesh adaptation

3.2.1 Goals

In this tutorial, we show how to use the different meshers of ISSM:

- Learn how to use the different meshers of ISSM:
  - `squaremesh` for square domains (ISMIP)
  - `roundmesh` for round domain (EISMINT)
  - `triangle` (from J. Shewchuk)
  - `bamg` (adapted from F. Hecht)
- Use anisotropic mesh adaptation to optimize the mesh resolution spatially

Go to `trunk/examples/Mesh/` to do this tutorial.

3.2.2 Squaremesh

`squaremesh` generates structured uniform meshes for rectangular domains.

3.2.2.1 Usage

```
>> md=model;
>> md=squaremesh(md,100,200,15,25);
```

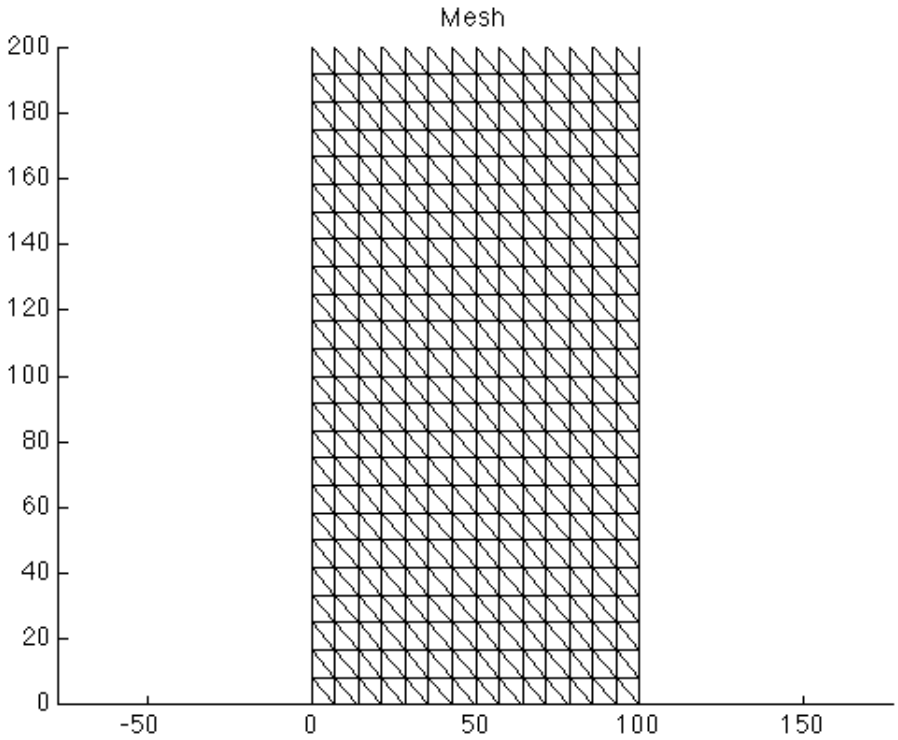
`squaremesh` takes the following arguments:

1. model
2. x-length (meters)
3. y-length (meters)
4. number of nodes along the x axis
5. number of nodes along the y axis

3.2.2.2 Example

The previous command creates the mesh shown below:

```
>> plotmodel(md,'data','mesh');
```



3.2.3 Roundmesh

`roundmesh` generates unstructured uniform meshes for circular domains.

3.2.3.1 Usage

```
>> md=roundmesh(model,100,10);
```

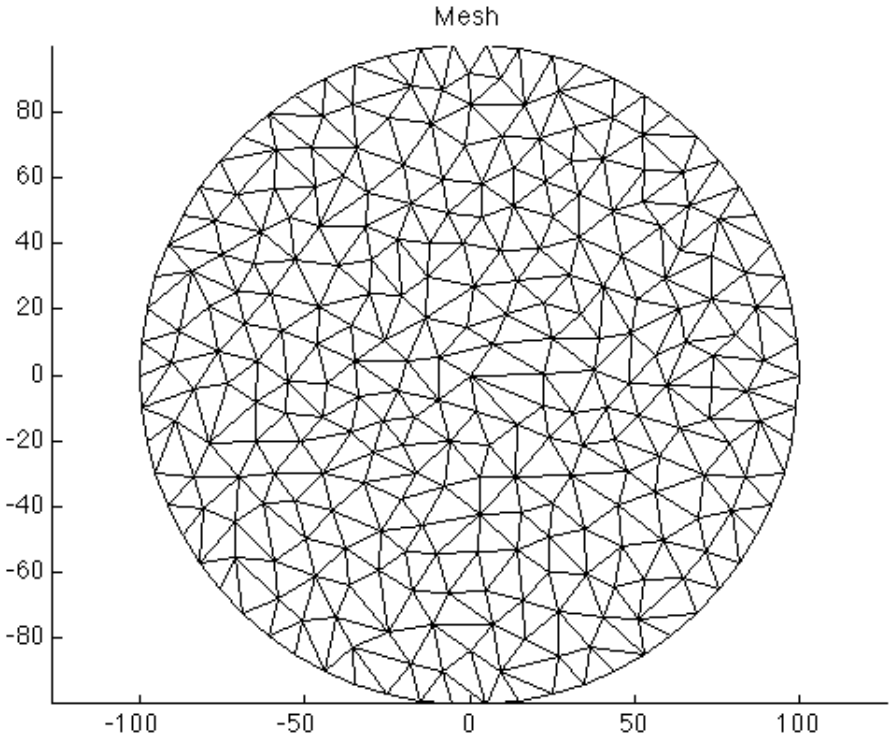
`roundmesh` takes the following arguments:

1. model
2. radius (meters)
3. element size (meters)

3.2.3.2 Example

The previous command creates the mesh shown below:

```
>> plotmodel(md,'data','mesh');
```



3.2.4 Triangle

`triangle` is a very fast algorithm for mesh generation. Developed by J Shewchuk, it generates unstructured triangular meshes.

3.2.4.1 Usage

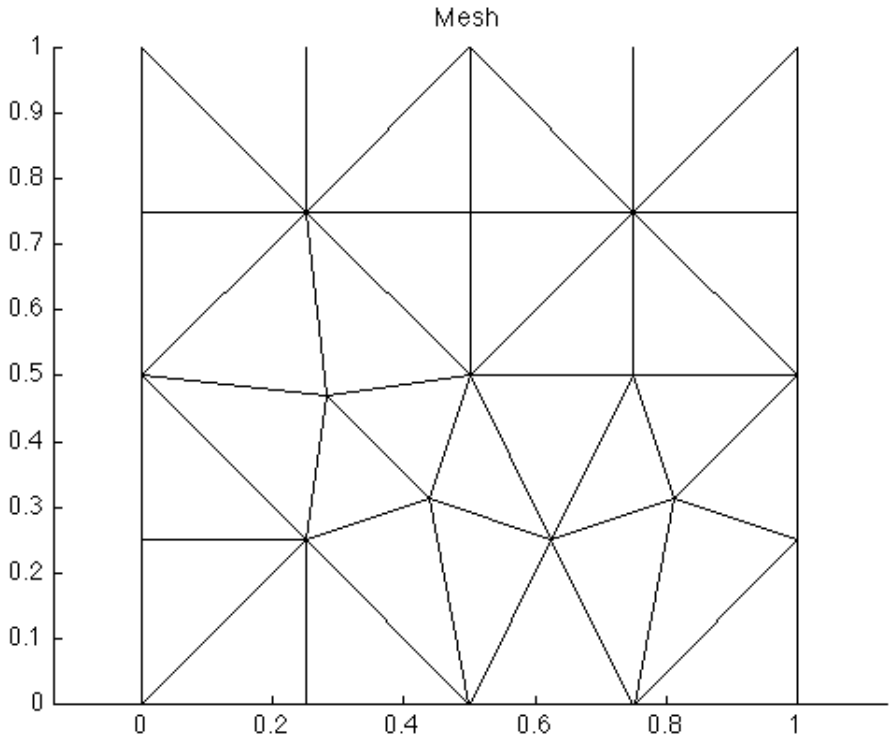
```
>> md=triangle(model,'Square.exp',.2);
```

`triangle` takes the following arguments:

1. model
2. ARGUS file of the domain outline (`.exp` extension, see here for more details)
3. average element size (meters)

The previous command creates the following mesh:

```
>> plotmodel(md,'data','mesh');
```



You can change the resolution from `0.2` to `0.05` to get a higher resolution.

3.2.5 Bamg

BAMG stands for Bidimensional Anisotropic Mesh Generator. It has been developed by Frederic Hecht, and was released in 2006 after more than 10 years of development. It is now part of FreeFEM++. The algorithm that is available on ISSM is inspired from this original software but has been entirely rewritten.

3.2.5.1 Usage

```
>> md=bamg(model,...);
```

`bamg` takes as it's first argument a model, and then pairs of options

1. model
2. pairs of options (type `help bamg` to get a full list of options)

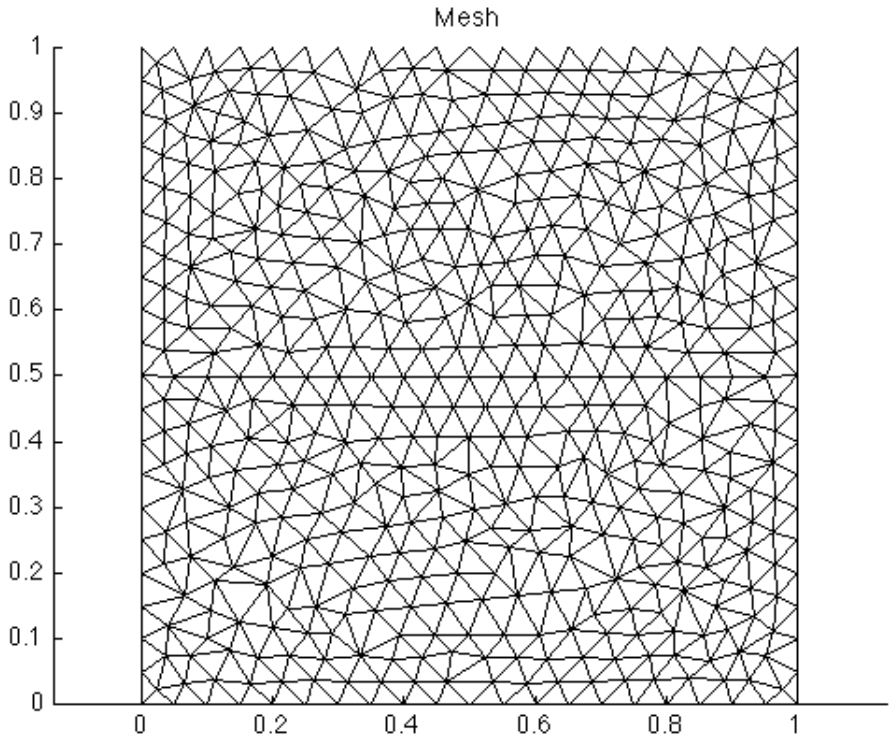
3.2.5.2 Uniform mesh

To create a non-uniform mesh, use the following options:

1. `'domain'` followed by the domain name
2. `'hmax'` followed by the size (meters) of each triangle

```
>> md=bamg(model,'domain','Square.exp','hmax',.05);
```

The previous command will create the following mesh (use `plotmodel(md,'data','mesh')` to visualize the mesh):



Note that the nodes are not as randomly distributed as `triangle`. The strength of BAMG is not for uniform meshes but for automatic mesh adaptation based on a metric.

3.2.5.3 Non-Uniform mesh

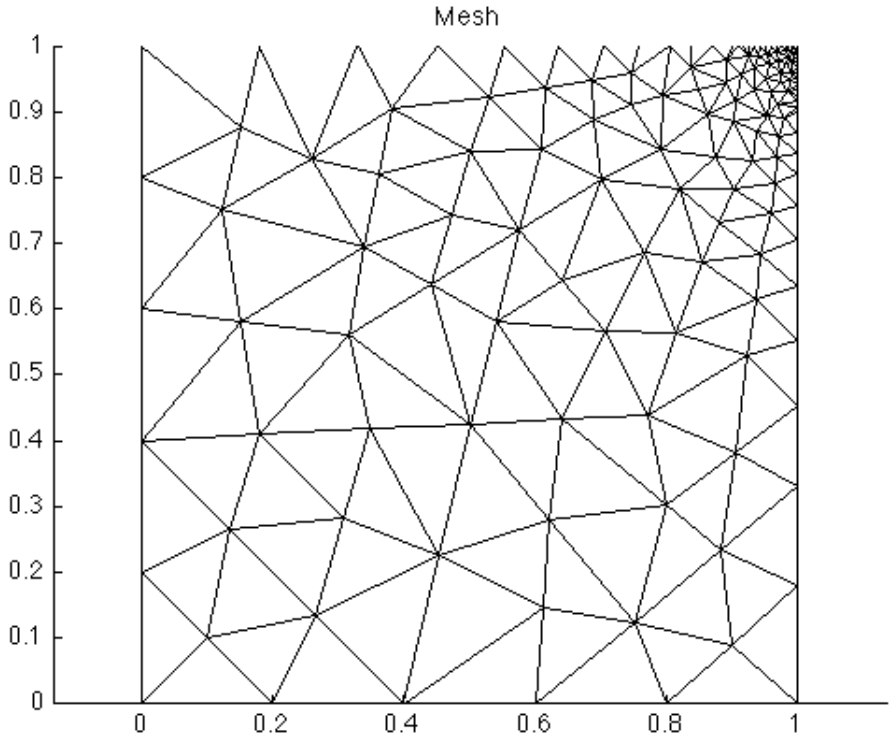
To create a non-uniform mesh, use the following options:

1. `'domain'` followed by the domain name
2. `'hvertices'` followed by the element size for each vertex of the domain outline

In our example, `Square.exp` has 4 vertices. If we want a resolution of 0.2, except in the vicinity of the third node, we use the following commands:

```
>> md=model;
>> hvertices=[0.2;0.2;0.005;0.2];
>> md=bamg(md,'domain','Square.exp','hvertices',hvertices);
```

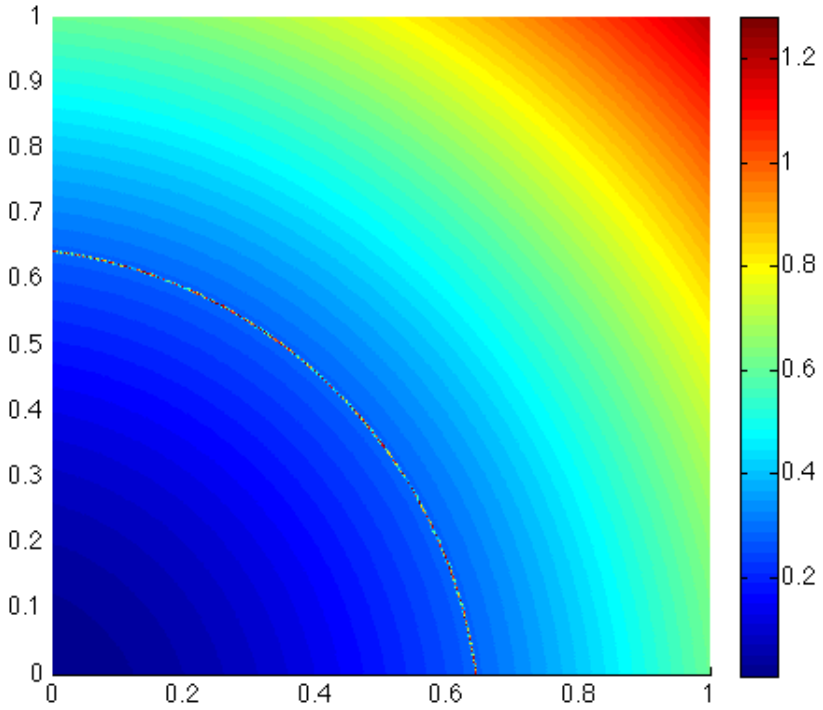
Use the `plotmodel(md,'data','mesh')` command to visualize the newly defined mesh:



3.2.5.4 Mesh adaptation

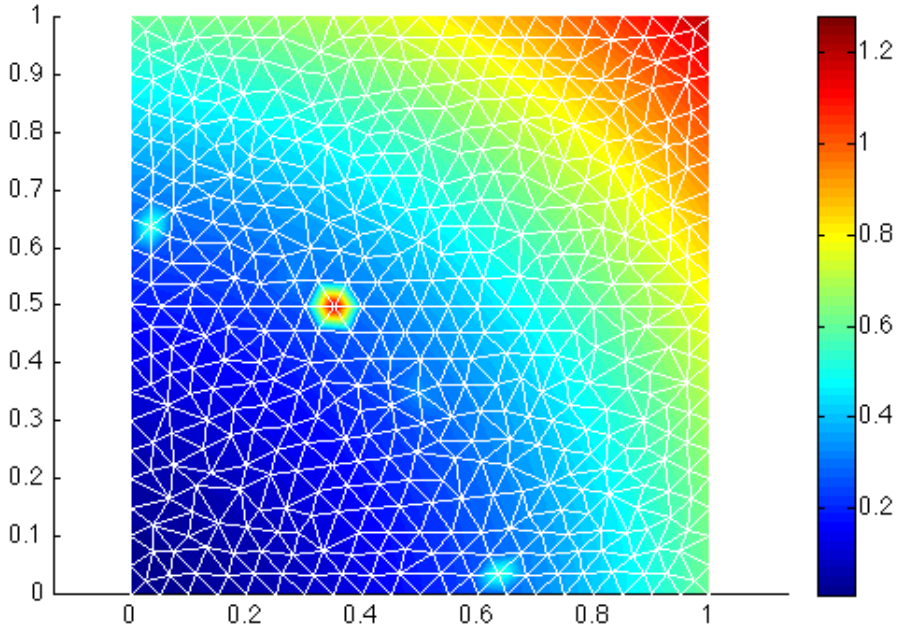
We can use observations to generate a mesh that is adapted to the solution we are trying to model. Given a solution field, `bamg` will calculate a metric based on the field's Hessian matrix (second derivative) to generate an anisotropic mesh that minimize the interpolation error (assuming that linear finite elements are used).

For a first example, we are going to use the observations given by the function `shock.m`. It generates a discontinuity that requires the mesh to be highly refined along a circle.



First, we generate a simple uniform mesh. We interpolate the observations on the vertices of this mesh:

```
>> md=bamg(model,'domain','Square.exp','hmax',.05);
>> vel=shock(md.mesh.x,md.mesh.y);
>> plotmodel(md,'data',vel,'edgecolor','w');
```



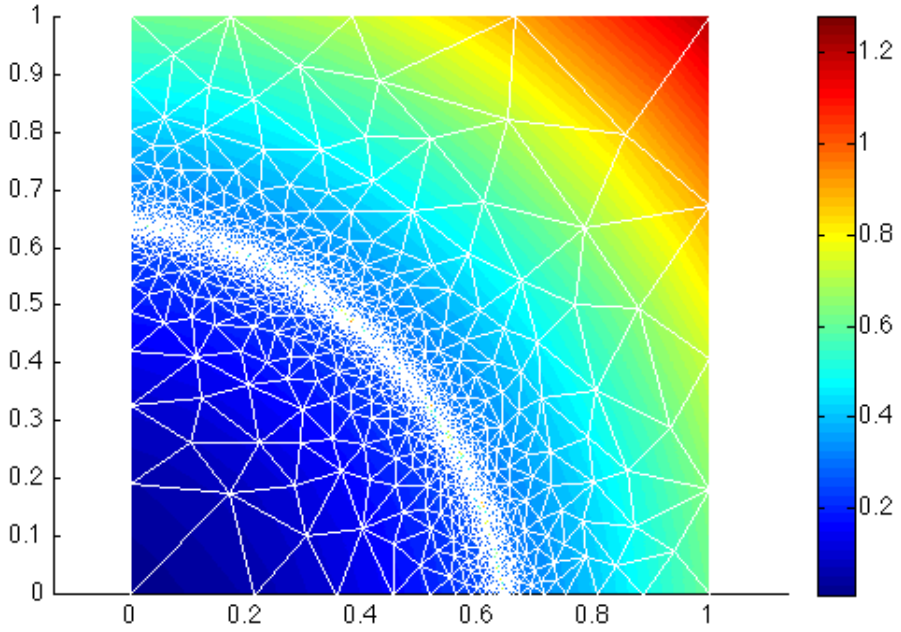
With a simple uniform mesh, the discontinuity is not captured. It is best to start with a finer mesh, which captures the discontinuity rather well, and interpolate the observations on this finer mesh to adapt the mesh anisotropically.

```
>> md=bamg(model,'domain','Square.exp','hmax',.005);
>> vel=shock(md.mesh.x,md.mesh.y);
```

Now, we call `bamg` a second time to adapt the mesh according the `vel`. We do not reinitialize `md` and call `bamg` again without specifying the `'domain'`, as a first mesh already exists in the model. We provide the following options:

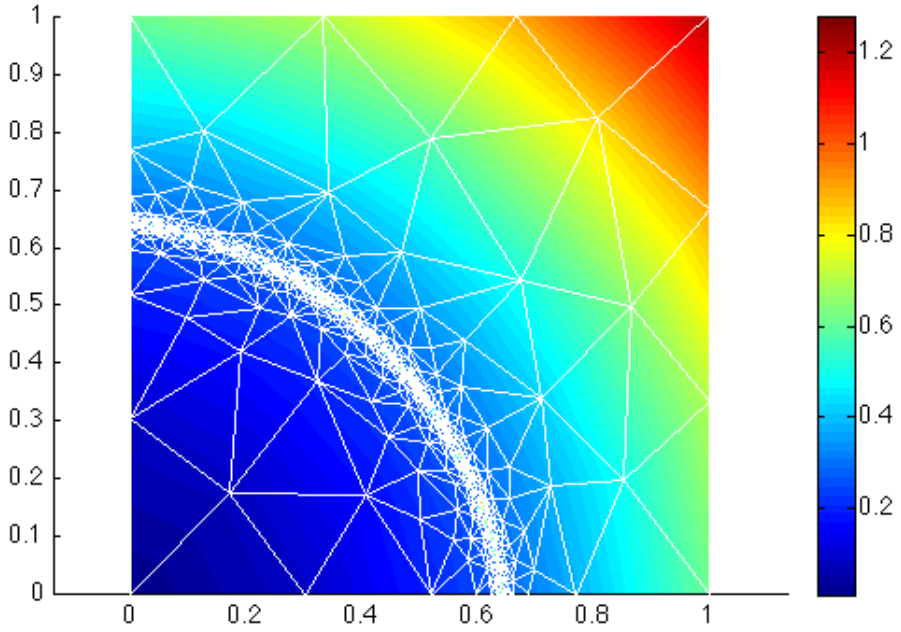
1. `'field'` followed by `vel`, the field we want to adapt the mesh to
2. `'err'` the allowed interpolation error (Here, the field must be captured within 0.05)
3. `'hmin'` minimum edge length
4. `'hmax'` maximum edge length

```
>> md=bamg(md,'field',vel,'err',0.05,'hmin',0.005,'hmax',0.3);
>> vel=shock(md.mesh.x,md.mesh.y);
>> plotmodel(md,'data',vel,'edgecolor','w');
```



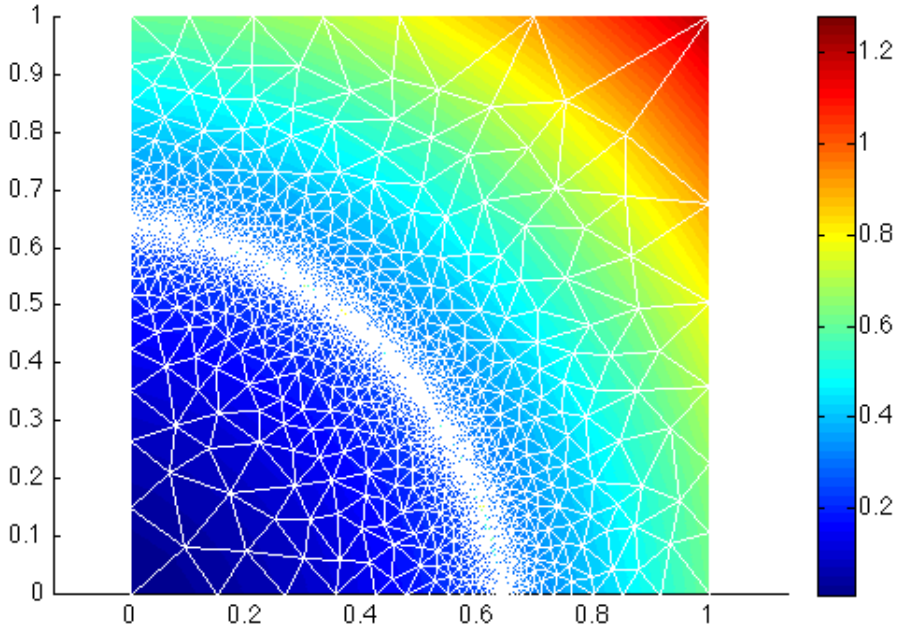
You can change the option `'err'` to 0.03, to see the effect of `'err'`. The ratio between two consecutive edges can be controled by the option `'gradation'`.

```
>> md=bamg(model,'domain','Square.exp','hmax',.005);
>> vel=shock(md.mesh.x,md.mesh.y);
>> md=bamg(md,'field',vel,'err',0.03,'hmin',0.005,'hmax',0.3,'gradation',3);
>> vel=shock(md.mesh.x,md.mesh.y);
>> plotmodel(md,'data',vel,'edgecolor','w');
```



We can also force the triangles to be equilateral by using the `'anisomax'` option, which specifies the maximum level of anisotry (between 0 and 1, 1 being fully isotropic).

```
>> md=bamg(model,'domain','Square.exp','hmax',.005);
>> vel=shock(md.mesh.x,md.mesh.y);
>> md=bamg(md,'field',vel,'err',0.03,'hmin',0.005,'hmax',0.3,'gradation',1.3,'anisomax',1);
>> vel=shock(md.mesh.x,md.mesh.y);
>> plotmodel(md,'data',vel,'edgecolor','w');
```



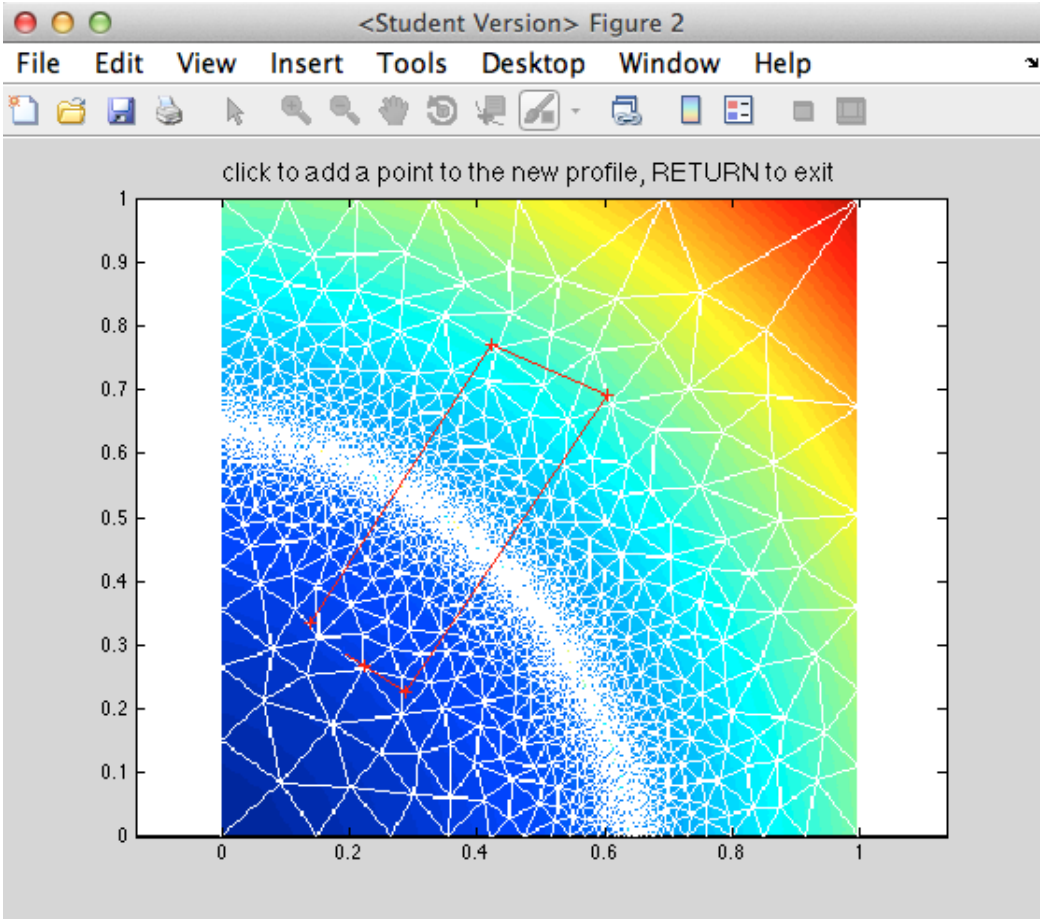
You can also try to refine a mesh using the function `circles.m`, which is provided in the same directory.

3.2.5.5 Mesh refinement in a specific region

It is sometimes necessary to specify a mesh resolution for an area of interest. We will use the same example as before. The first step consists of creating an ARGUS file that defines the region where we want to refine the mesh.

We first plot `vel` and we call the function `exptool` to create a file `refinement.exp` that defines this region. Select `add a contour (closed)`. Draw a contour over a given region, hit enter when you are done, and then select quit. You should now see the `refinement.exp` file in the current directory.

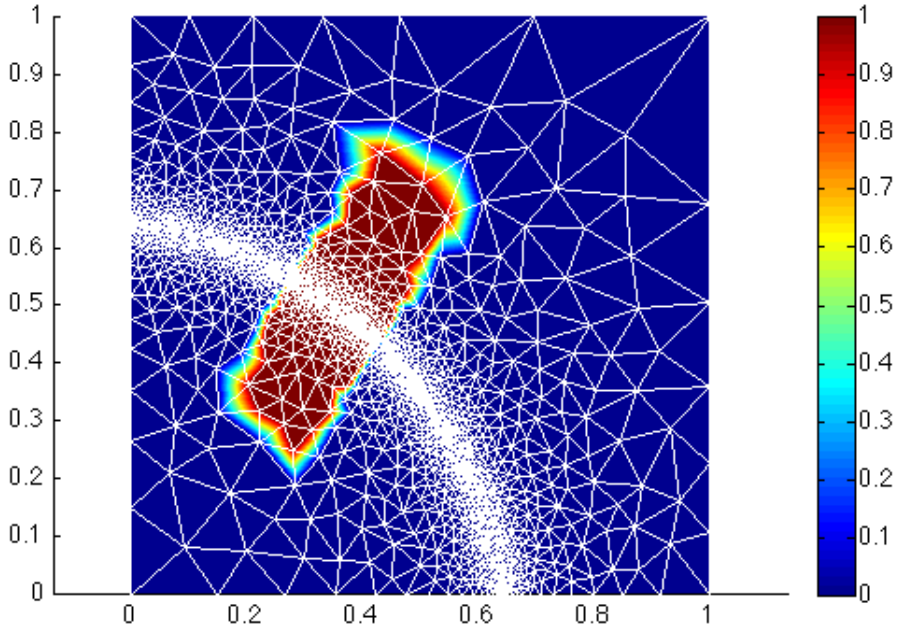
```
>> plotmodel(md,'data',vel,'edgecolor','w');
>> exptool('refinement.exp')
```



Now, we are going to create a vector that specifies, for each vertex of the existing mesh, the resolution of the adapted mesh. We use `NaN` for the vertices we do not want to change. So in this example, this will be a vector of `NaN`, except for the vertices in `refinement.exp`, where we want a resolution of 0.02:

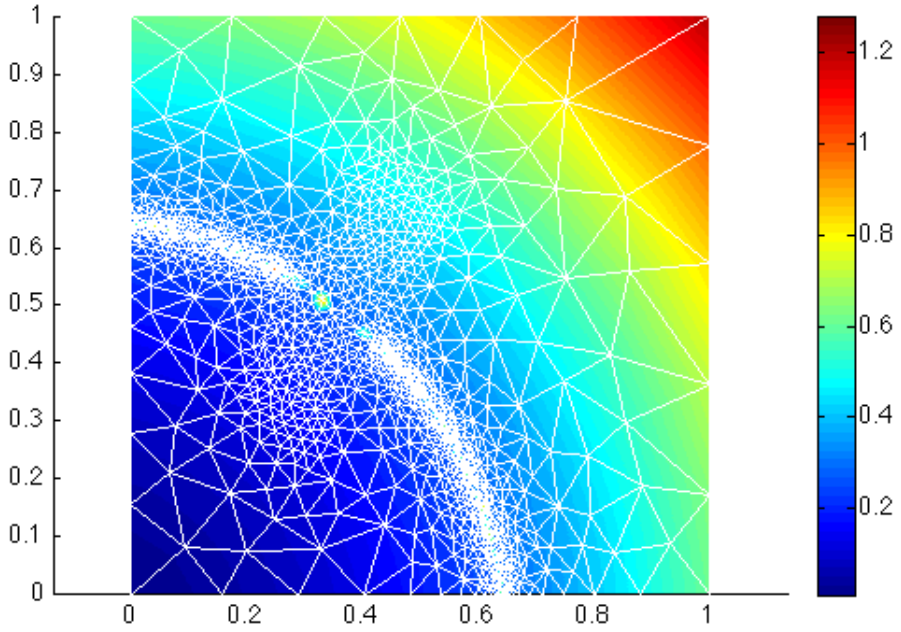
```
>> h=NaN*ones(md.mesh.numberofvertices,1);
>> in=ContourToNodes(md.mesh.x,md.mesh.y,'refinement.exp',1);
>> h(find(in))=0.02;
>> plotmodel(md,'data',in,'edgecolor','w');
```

You will see that all the vertices that are in `refinement.exp` have a value of 1 (they are inside the contour), and the others are 0.



Now, we call `bamg` a third time, with the specified resolution for the vertices that are in `refinement.exp`:

```
>> vel=shock(md.mesh.x,md.mesh.y);
>> md=bamg(md,'field',vel,'err',0.03,'hmin',0.005,'hmax',0.3,'hVertices',h);
>> vel=shock(md.mesh.x,md.mesh.y);
>> plotmodel(md,'data',vel,'edgecolor','w');
```



3.2.5.6 Another example

If you would like to try another example, you can use the function `cricles.m` instead of `shock.m`. It is also a 1x1 square but with a pattern that inclused five circles.

3.3 Inverse method

3.3.1 Goals

- Learn how to use the model to invert for ice rigidity (B) and basal friction from surface velocities
- Being able to choose the right cost functions, with the right weights
- Understand the limitations of inversions

3.3.2 Introduction

Several model input parameters, such as the ice rigidity B (`md.materials.rheology_B`) and basal fricion α (`md.friction.coefficient`), are difficult to measure remotely and are critical controls on ice dynamics.

To get a good guess of what these parameters are, we use *inversions*. Inversions consist in inferring unknown parameters using additional observations. Here, we use surface velocities to infer our unknown input parameters, by minimizing the misfit between the observed and modeled velocities. For example, our cost function could be:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2}\left(\left(v_x - v_x^{\text{obs}}\right)^2 + \left(v_y - v_y^{\text{obs}}\right)^2\right) dS \tag{3.1}$$

And so we would optimize our unknown model input to minimize the cost function $\mathcal{J}$.

Inversions where first introduced to glaciology by *MacAyeal* [1993] for an SSA model, and extended since to 3d models for other model parameters.

To illustrate this method, we are going to perform a twin experiment. We give ourselves a rigidity field (B) and use the modeled velocities as synthetic observation in a second run, where we start from another initial rigidity field, and see if we can recover the rigidity field that was used to generate the observations.

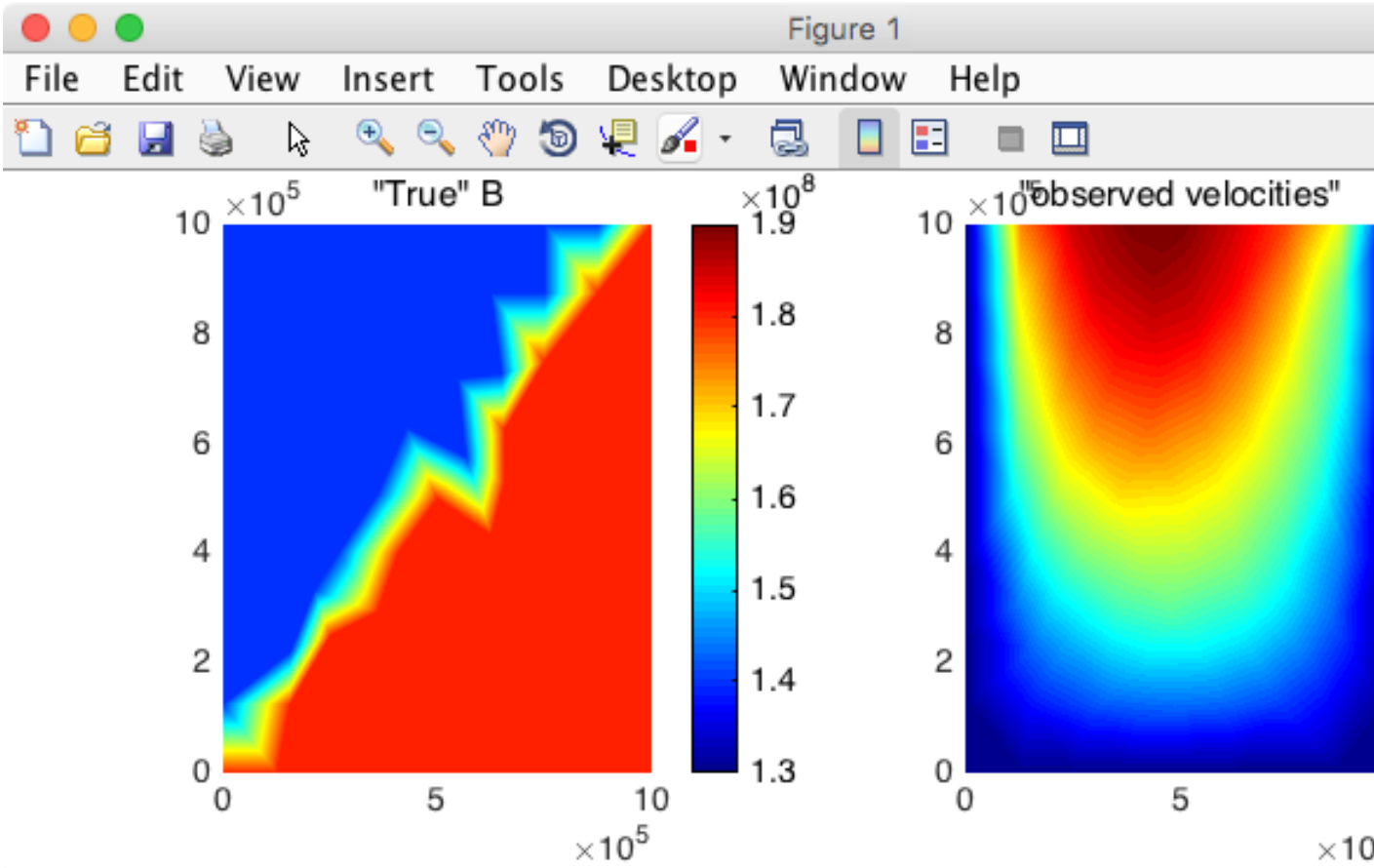
3.3.3 Hands on 1 (ice rigidity, B)

3.3.3.1 Setp 1: Generating Observations

First, go to `trunk/examples/Inversion/` and start MATLAB. We will start by creating a new model and generate our synthetic observations. Open the `runme.m` and ensure that `step=1` at the top of the file. Execute this first step:

```
>> runme
```

You will see on the left our prescribed rigidity, B, and to the right the calculated velocities. We choose a pattern with 2 distinct values for B for the upper left region, and stiffer ice for the lower right, with a sharp transition.

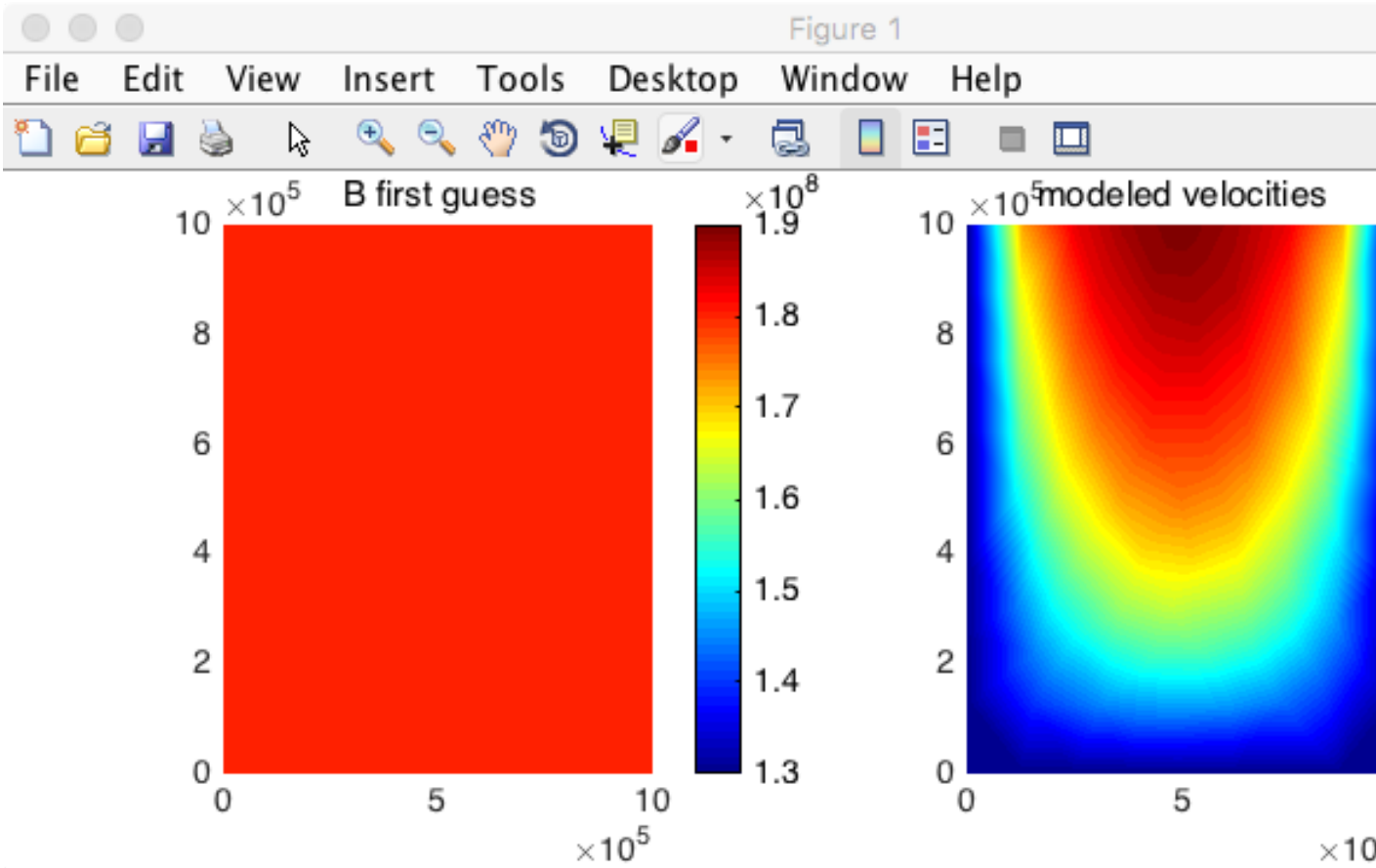


In the next step, we our going to change the rigidity to something uniform, use our previously calculated

velocities (from step 1) as observations, and see if we can recover that initial pattern that was used to generate the observations.

3.3.3.2 Step 2: Initial guess and initial velocity

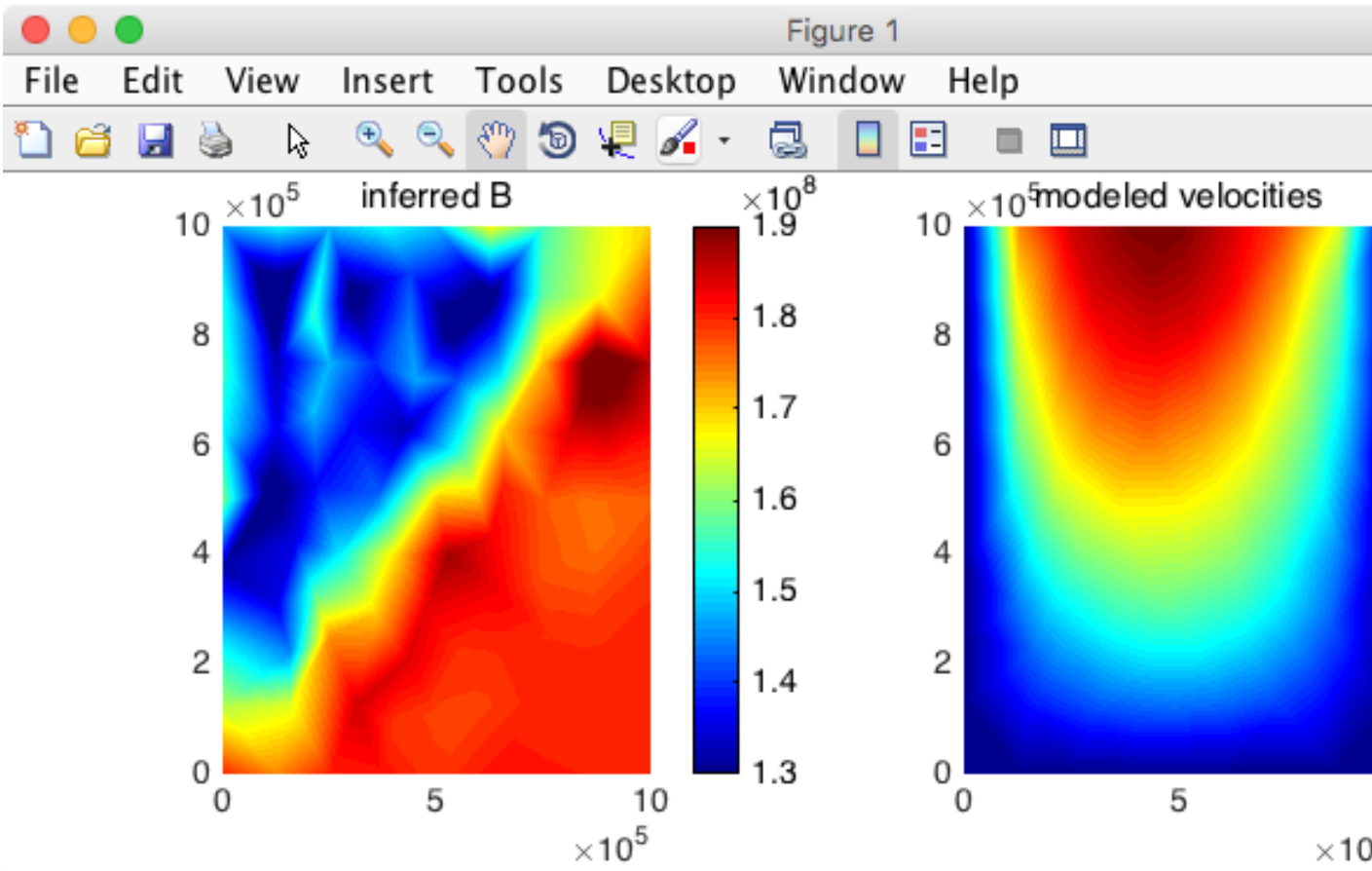
We now change the rigidity, B, and make it uniform. The results of the previous step are taken as observations (but we will only use them in step 3). Open `runme.m` and set `step=2`. Save the file and execute step 2 in MATLAB as above.



We now see that the left panel is constant, and the velocity is symetrical. This is our initial guess for B and our initial modeled velocity. In the next step, we are going to tune B, so that the modeled velocity is as close as possible to the velocity of step 1.

3.3.3.3 Step 3: inverting for B

We perform here the inversion of B. Open `runme.m` and set the step as `step=3`.



The general pattern is right (stiffer ice in the lower right), but it is noisy. Inverse problems are ill-posed: a solution might not exist, might not be unique, and might not depend continuously on input data. One of the consequences is that the inferred pattern for B is not smooth, and these wiggles or *not* physical. Adding regularization that penalizes wiggles in the control parameter stabilizes the inversion.

3.3.3.4 Step 4: Adding regularization

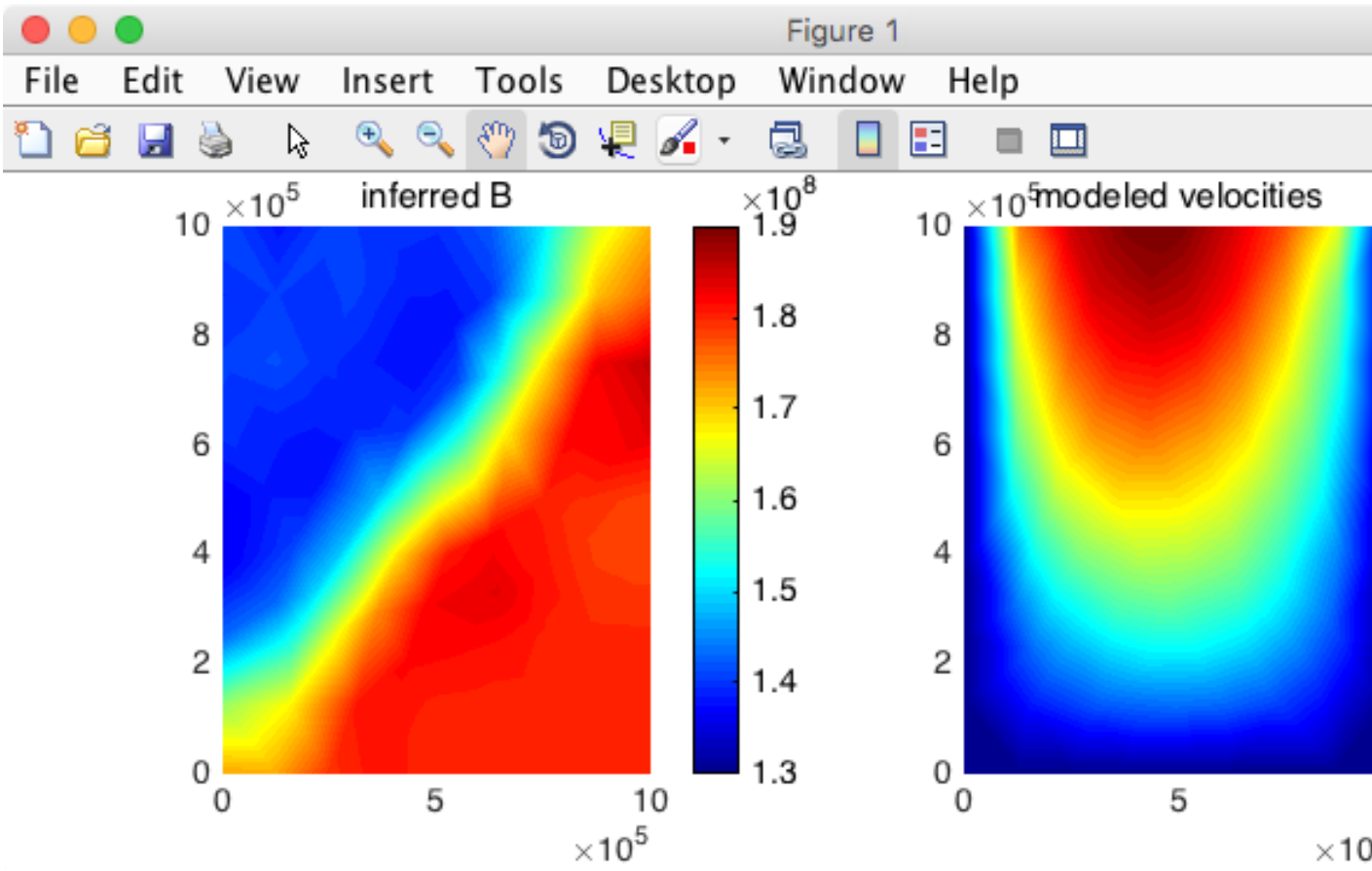
Here, we would like to add a term of regularzation to our cost function:

$$\mathcal{J}(B) = \int_S w_1 \frac{1}{2}\left(\left(v_x - v_x^{\text{obs}}\right)^2 + \left(v_y - v_y^{\text{obs}}\right)^2\right) dS + \int_b w_2 \frac{1}{2}\|\nabla B\|^2 db \tag{3.2}$$

The second term, known as Tikhonov regularization, penalizes strong gradients in B. Since the inversion tries to minimize our cost function $\mathcal{J}$, the optimization algorithm will try to also reduce the second term.

w_1 and w_2 are the weights associated to each component of the cost function. To have more regularization, one should increase w_2 (or decrease w_1), and vice versa.

Set `step=4` in the `runme.m` file and execute it. Your results should now look like this:



We successfully reconstructed the pattern of ice rigidity, but we could not capture the sharp transition between high and low rigidity because of the regularization that we had to introduce to stabilize the inversion.

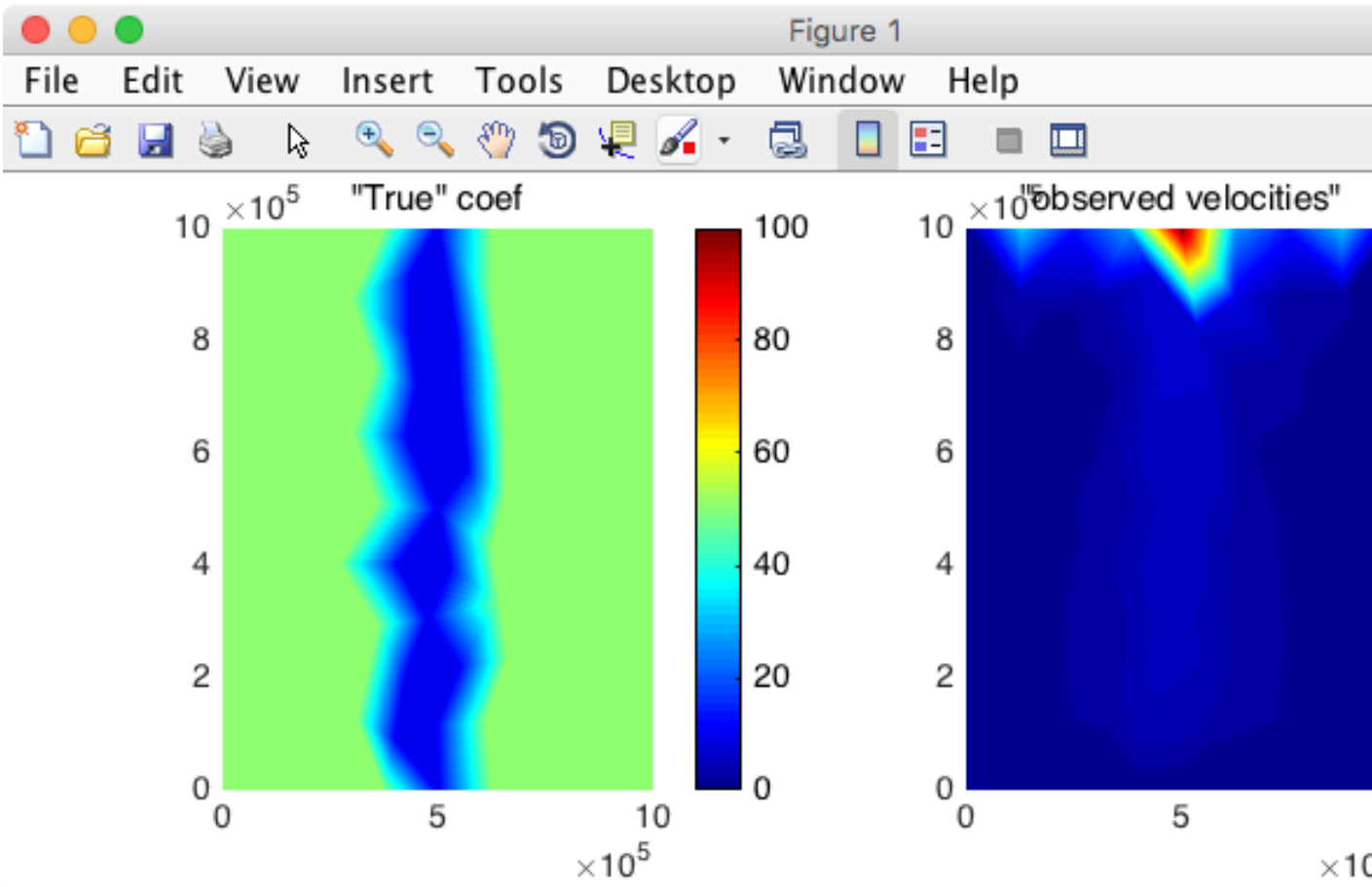
3.3.4 Hands on 2 (friction)

We would like to do the same twin experiment here, but invert for basal friction of a grounded glacier. Here, you are going to make modifications to the `runme.m` script.

3.3.4.1 Changes to step 1

1. The mask is now all grounded
2. Increase bed (`md.geometry.base`) and surface elevation (`md.geometry.surface`) by 100 meters
3. B (`md.materials.rheology_B`) is now uniform = 1.8x10^8
4. Friction coefficient: 50, and 10 for 600,000<x<400,000
5. change the `plotmodel` command and plot `md.friction.coefficient` instead, between 0 and 100.

After running step 1 again, you should get the following figure.



If you don't, then double check your changes before looking at the solutions below. We are modeling here a glacier flowing over a region where there is a lot of sliding. We want to see if the inversion can reconstruct this region of low friction.

3.3.4.2 Solutions to step 1

```
%Generate observations
md = model;
md = triangle(md,'DomainOutline.exp',100000);

%CHANGES START
md = setmask(md,'','');
%CHANGES END

md = parameterize(md,'Square.par');

%CHANGES START
md.geometry.base=md.geometry.base+100;
md.geometry.surface=md.geometry.surface+100;
md.materials.rheology_B(:)=1.8e8;
md.friction.coefficient(:)=50;
pos=find(md.mesh.x>400e3 & md.mesh.x<600e3);
md.friction.coefficient(pos)=10;
%CHANGES END
```

```
md = setflowequation(md,'SSA','all');
md.cluster = generic('np',2);
md = solve(md,StressbalanceSolutionEnum());

%CHANGES START
plotmodel(md,'axis#all','tight','data',md.friction.coefficient,'caxis',[0 100],'title','"True" coef',
   'data',md.results.StressbalanceSolution.Vel,'title','"observed velocities"')
%CHANGES END

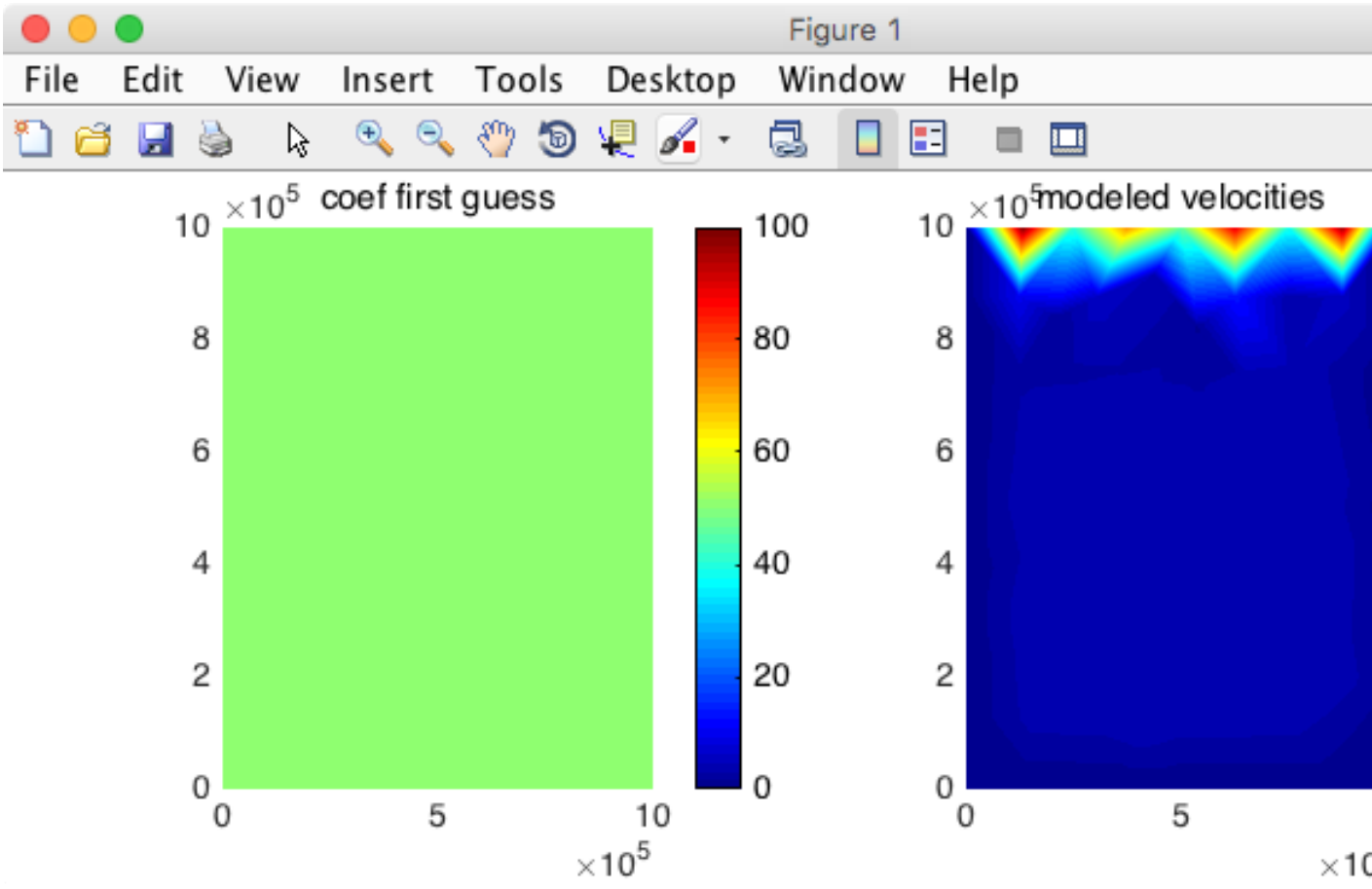
save model1 md
```

3.3.4.3 Changes to step 2

For step 2, we now want to set our new first guess for the basal friction to a uniform value.

1. set the friction (`md.friction.coefficient`) to a uniform value of 50
2. change the `plotmodel` command and plot `md.friction.coefficient` instead, between 0 and 100.

After running step 2, you should get the following figure:



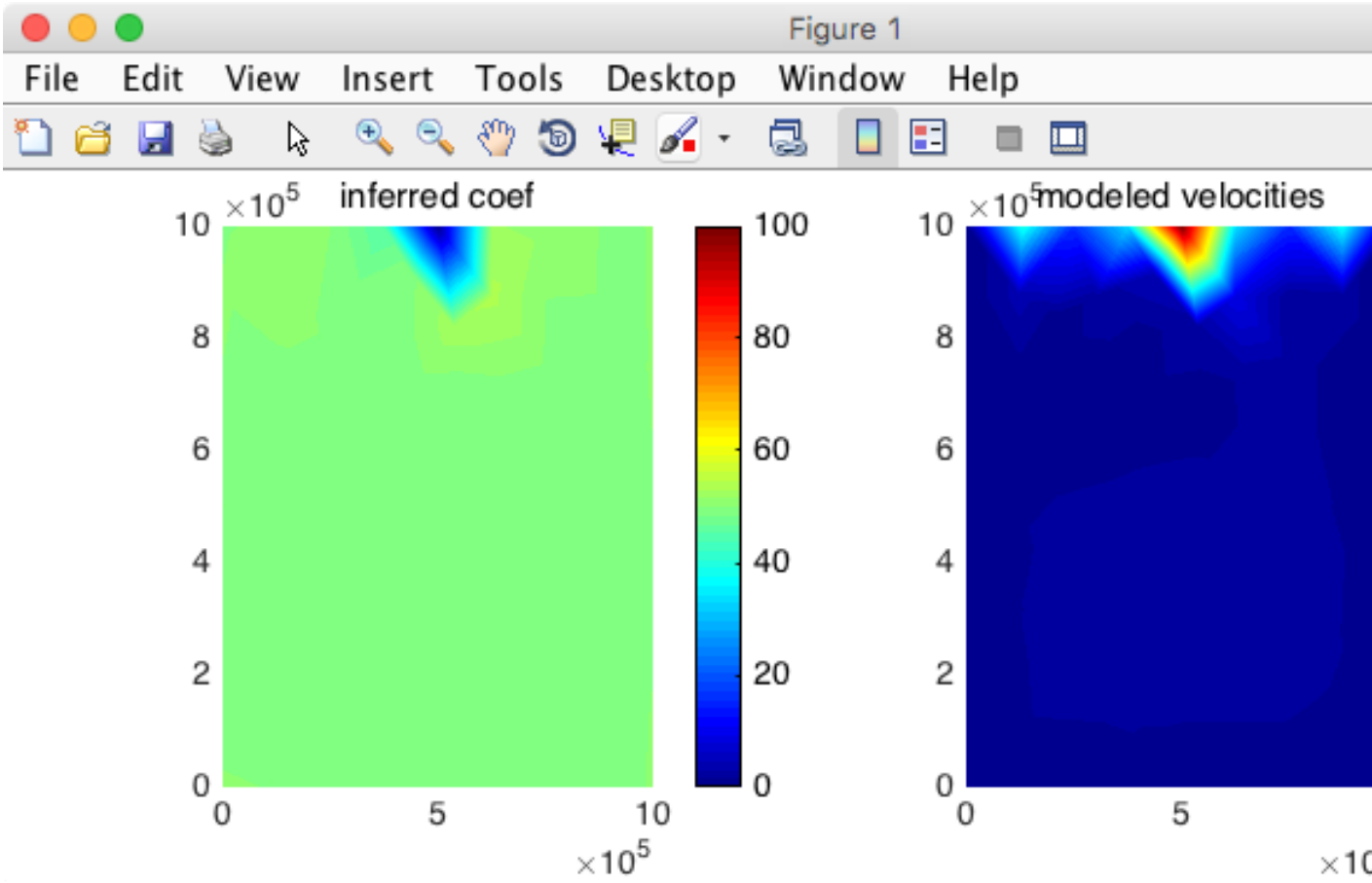
if you don't... double check your changes. As you can see, the velocity does not show any fast flowing ice stream in the center of the domain, as expected since the friction is uniform.

3.3.4.4 Changes to step 3

We now want to invert for basal friction and see if we can reconstruct the zone of sliding. We need to change what we are inverting for, and change the optimization parameters:

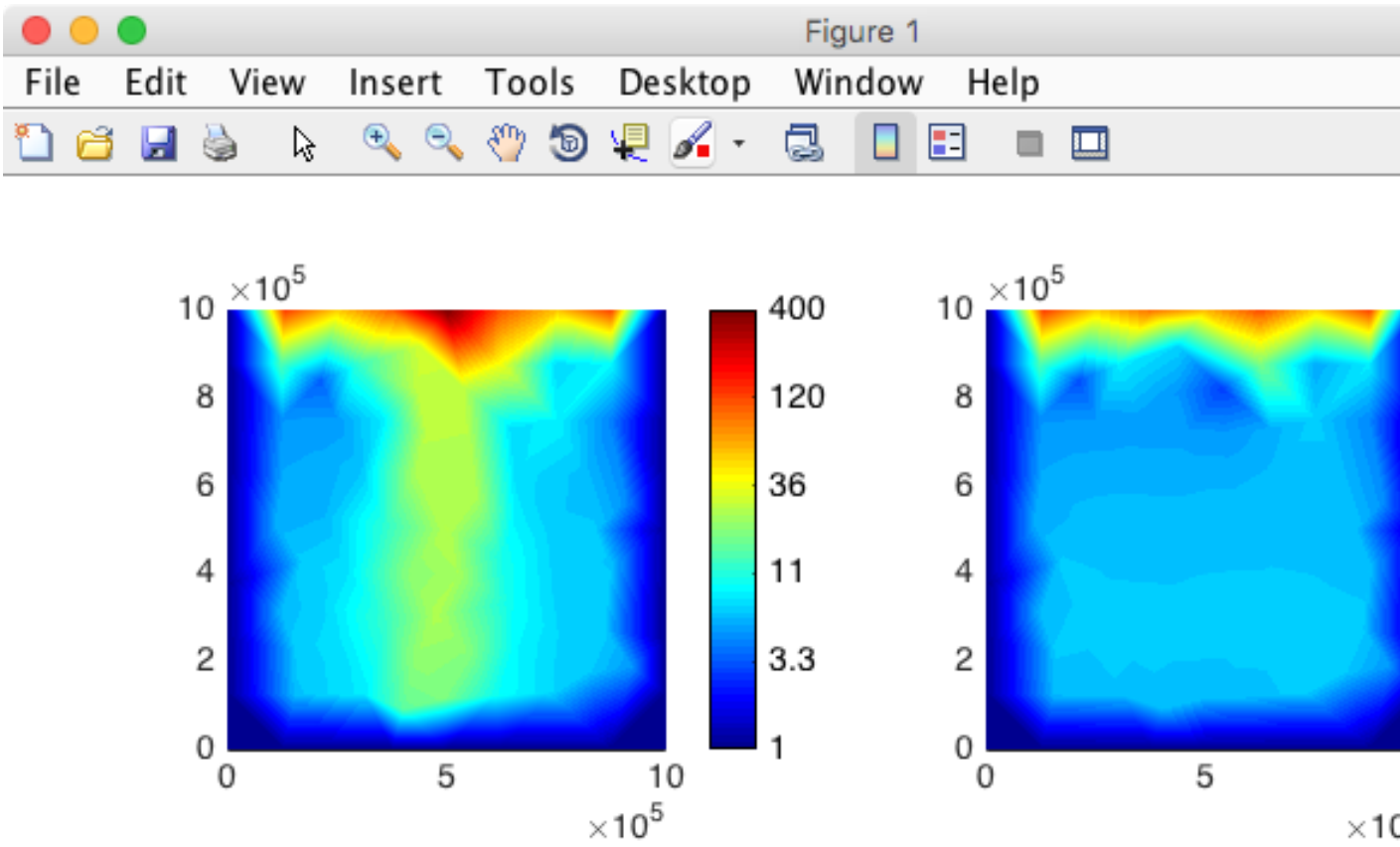
- We now invert for `'FrictionCoefficient'`
- Do we keep the same cost function? yes for now...
- We want the parameter to be between 1 and 100

After running step 3, you should get the following figure:



if you don't, the solutions are below. As you can see, we get more sliding close to the front, but the rest of the domain is unchanged. And that's because when we look at the velocity (right), it does capture the fast spot close to the front, so in terms of cost function, the inversion did a great job in matching the observation. But if we look at the log of the velocity (we are adding one to avoid `log(0)`):

```
plotmodel(md,'data',md.inversion.vel_obs+1,'data',md.results.StressbalanceSolution.Vel+1,'log#all',10
400])
```



we clearly see the zone of fast sliding in the observations but not in the results from the inversion. So we need to change the cost function to add this information, we not only want the square of the difference between modeled and observed velocities to be minimized, we also want their logs to be minimized.

3.3.4.5 Solutions to step 3

```
...
md.inversion.control_parameters = {'FrictionCoefficient'};
...
md.inversion.min_parameters = 1*ones(md.mesh.numberofvertices,1);
md.inversion.max_parameters = 100*ones(md.mesh.numberofvertices,1);
...
```

3.3.4.6 Changing the cost function

We want the cost function to include an additional term:

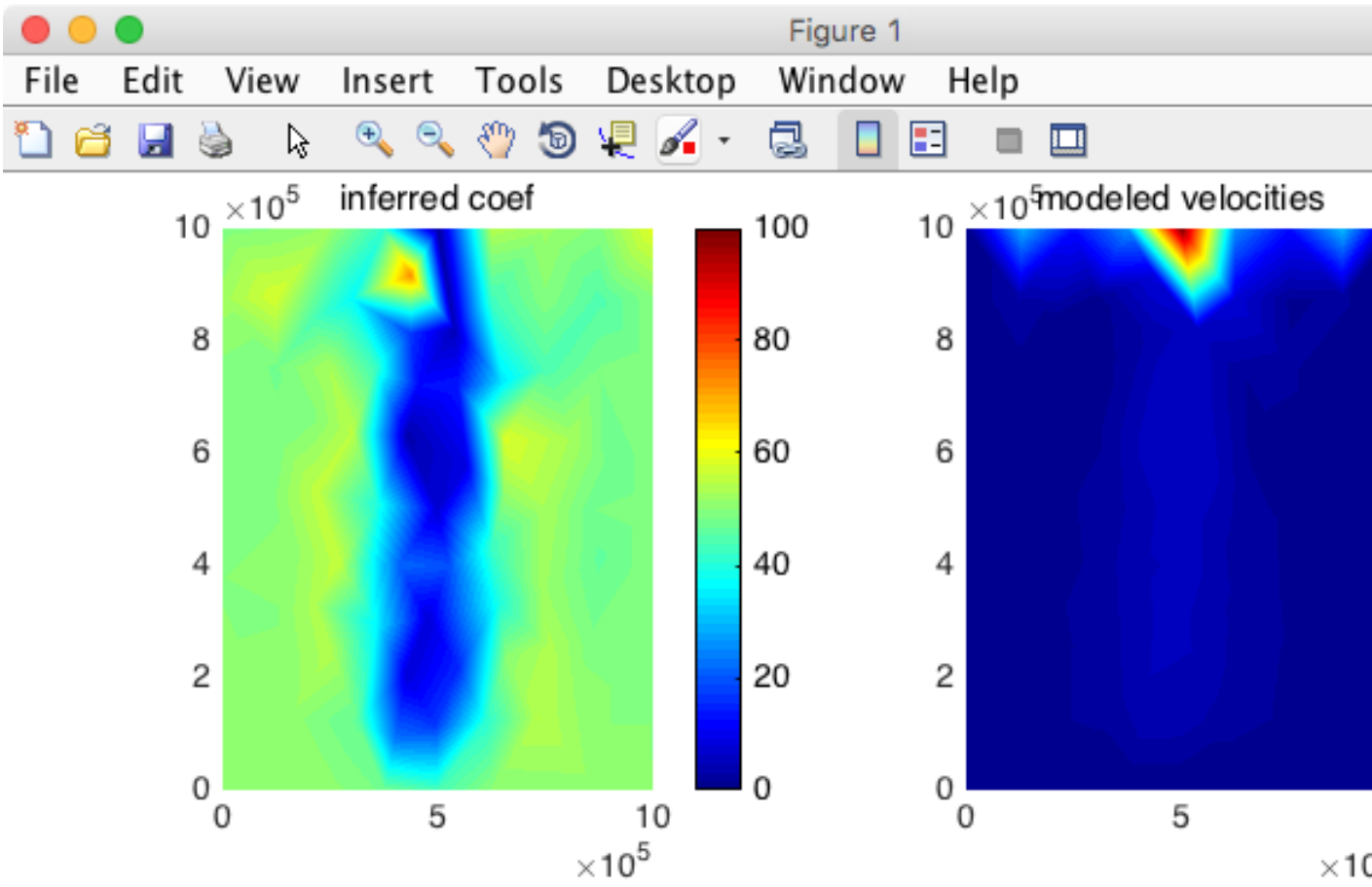
$$\mathcal{J}(\mathbf{v}) = \int_S w_1 \frac{1}{2}\left(\left(v_x - v_x^{\text{obs}}\right)^2 + \left(v_y - v_y^{\text{obs}}\right)^2\right) dS + \int_S w_2 \left(\log\left(\frac{\|\mathbf{v}\| + \varepsilon}{\|\mathbf{v}^{\text{obs}}\| + \varepsilon}\right)\right)^2 dS \tag{3.3}$$

This page lists all the cost function available. We want here the cost function to include the absolute and relative misfits. Typing in MATLAB `md.inversion` will give you the numbers associated to these cost function: `[101,103]`. We also need to determine the weights associated to each cost function: w_1

and w_2. As a rule of thumb, it is generally preferable if the two components have the same order of magnitude at the end of the optimization. You can try with $w_1 = w_2 = 1$ and run the inversion, look at their contribution at the end of the inversion and increase (or decrease) w_1. You need to change the following in step 3:

1. We now want the cost functions 101 and 103
2. the coefficients applied to each component of the cost functions has 2 columns (since there are 2 components)
3. We want to increase w_1 to 3000

You should get the following results:



The solutions are below if you don't have the same figure. We now successfully reconstructed the zone of sliding! But again, the pattern is a little bit noisy, and we are going to add regularization.

3.3.4.7 Solutions to step 3b

```
md.inversion.cost_functions = [101 103];
md.inversion.cost_functions_coefficients = ones(md.mesh.numberofvertices,2);
md.inversion.cost_functions_coefficients(:,1)=3000;
md.inversion.cost_functions_coefficients(:,2)=1;
```

3.3.4.8 Adding regularization

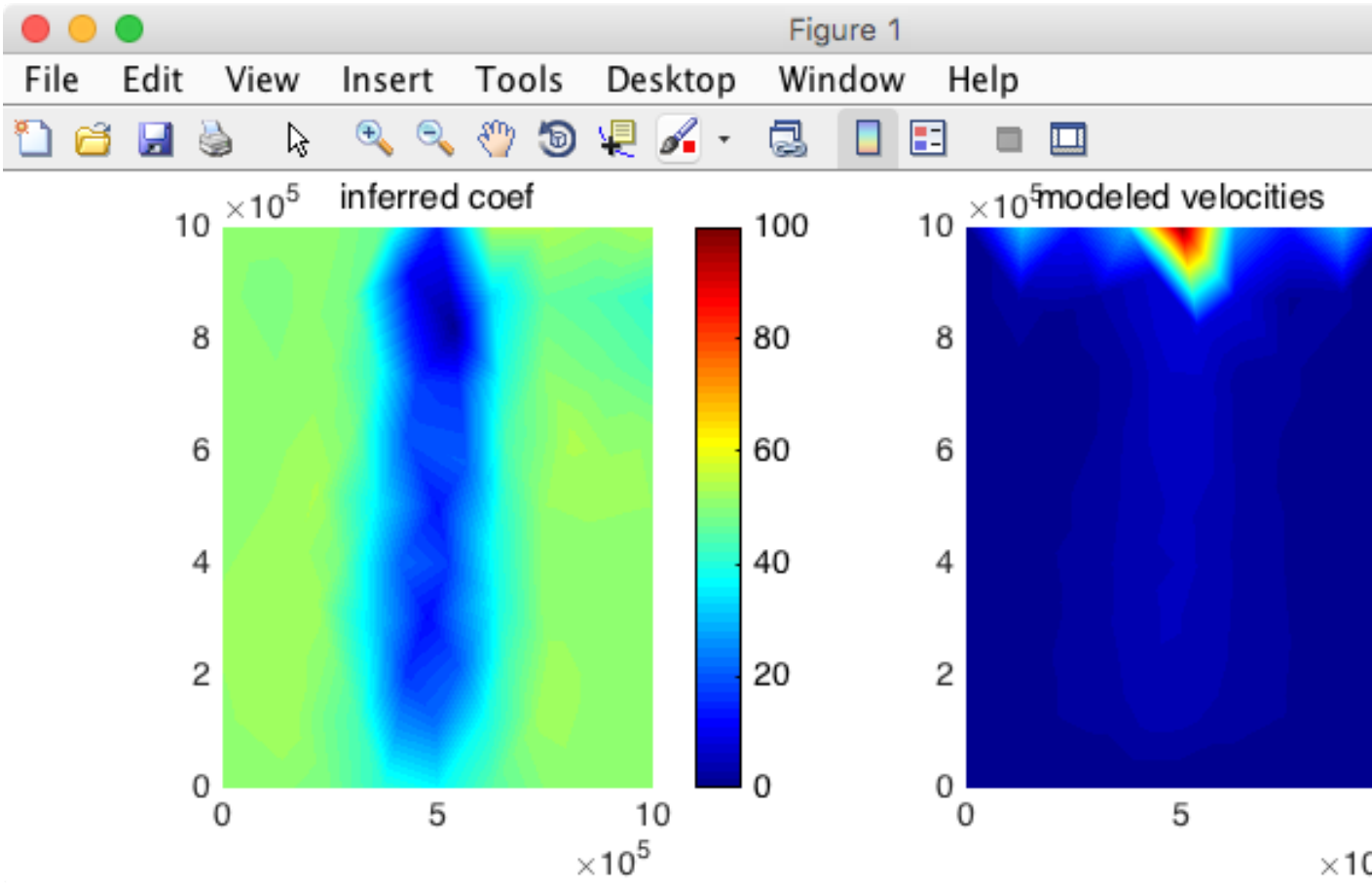
We want the cost function to include a regularization term:

$$\mathcal{J} = \int_S w_1 \frac{1}{2}\left(\left(v_x - v_x^{\text{obs}}\right)^2 + \left(v_y - v_y^{\text{obs}}\right)^2\right) dS + \int_S w_2 \left(\log\left(\frac{\|\mathbf{v}\| + \varepsilon}{\|\mathbf{v}^{\text{obs}}\| + \varepsilon}\right)\right)^2 dS + \int_B w_3 \frac{1}{2}\|\nabla\alpha\|^2 dB \tag{3.4}$$

You need to change the following in step 3:

1. We now want the cost functions 101, 103 and 501
2. the coefficients applied to each component of the cost functions has 3 columns (since there are 3 components)
3. We want to set w_3 to 0.01

You should get the following results:



The zone of sliding is captured and the inferred friction is smooth!

3.3.4.9 Solutions to step 3c

```
md.inversion.cost_functions = [101 103 501];
md.inversion.cost_functions_coefficients = ones(md.mesh.numberofvertices,3);
```

```
md.inversion.cost_functions_coefficients(:,1)=3000;
md.inversion.cost_functions_coefficients(:,2)=1;
md.inversion.cost_functions_coefficients(:,3)=0.01;
```

3.4 ISMIP test

3.4.1 Goals

- Test the ISSM skills that you have gained so far
- Create ISSM models by Following the given keyword instructions
- Run tests from the Ice Sheet Model Intercomparison Project (ISMIP - Tests A and F)

Go to `trunk/examples/ISMIP/` to do this tutorial.

3.4.2 Introduction / How To

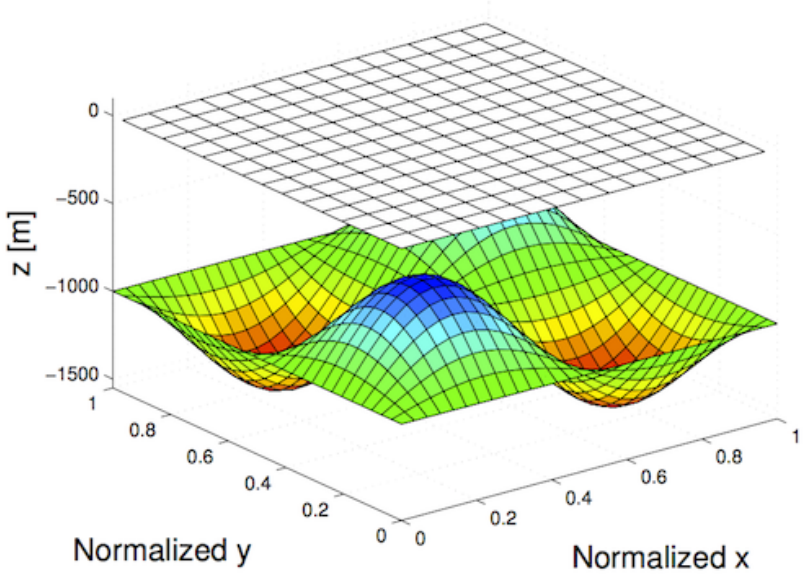
The `runme.m` file and `*par` files give a layout of the simulation that has to be modified.

- Each code line that has to be typed in is preceded by `\%->`. Type the appropriate code below this symbol.
- Keywords introduced by `#` should be typed in MATLAB to get more information, if necessary
- The `runme.m` and `*.par` files each have a corresponding `Cheaty*` file that should be referenced if stuck.

3.4.3 Test A

In Test A, we will generate a Square ice sheet flowing over a bumpy bed:

- Sinusoidal bedrock
- Ice frozen on the bed
- Periodic boundary conditions



3.4.4 Simulation File Layout and Organization

The simulation file `runme.m` is organized into different steps, each with the same structure:

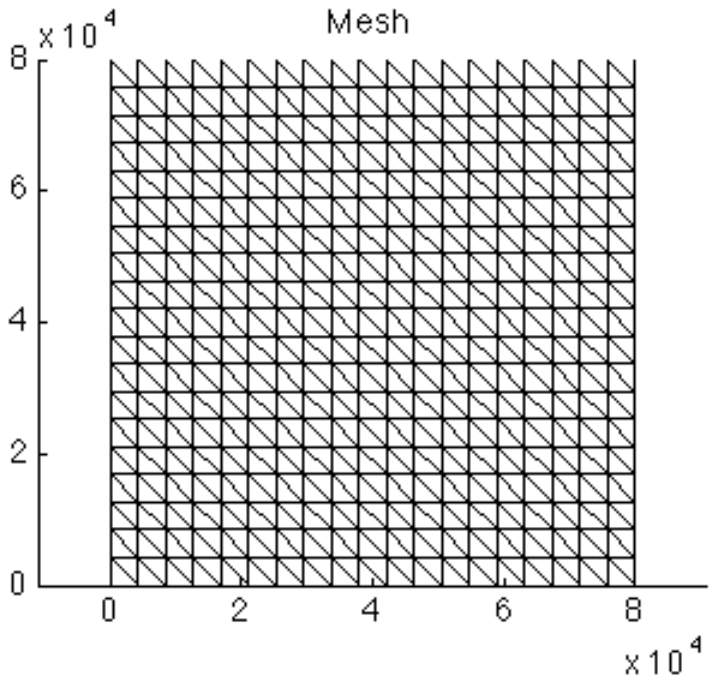
- Model loading
- Performing an action
- Model saving

The step specifier `steps` is defined at the top of the `runme.m` file.

3.4.5 Mesh

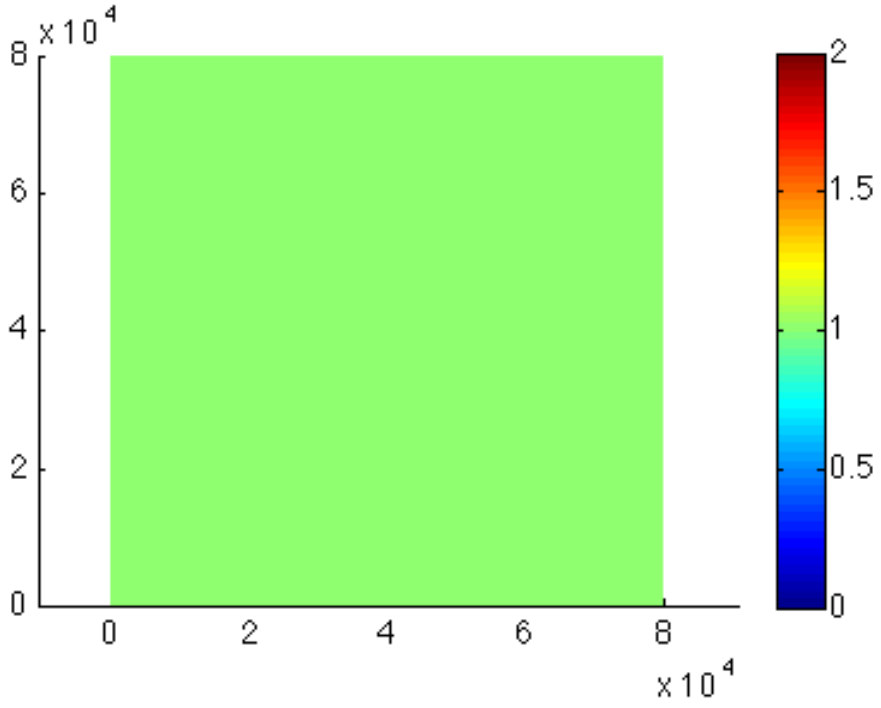
In place of loading a preceding model we initialize one. The action here is the generation of a mesh. To do this initialize `md` as a new model `(#help model)` and generate a `squaremesh (#help squaremesh)` with the following parameters. Afterward, plot the mesh and save the model.

- Mesh size: 80,000 meters
- Nodes in each direction: 20



Load the preceding step (`#help loadmodel`). Path is given by the organizer with the name of the given step. Set the mask (`#help setmask`). Note that all MISMIP nodes are grounded. Plot the given mask (`md.mask`) to locate the field. Save the model.

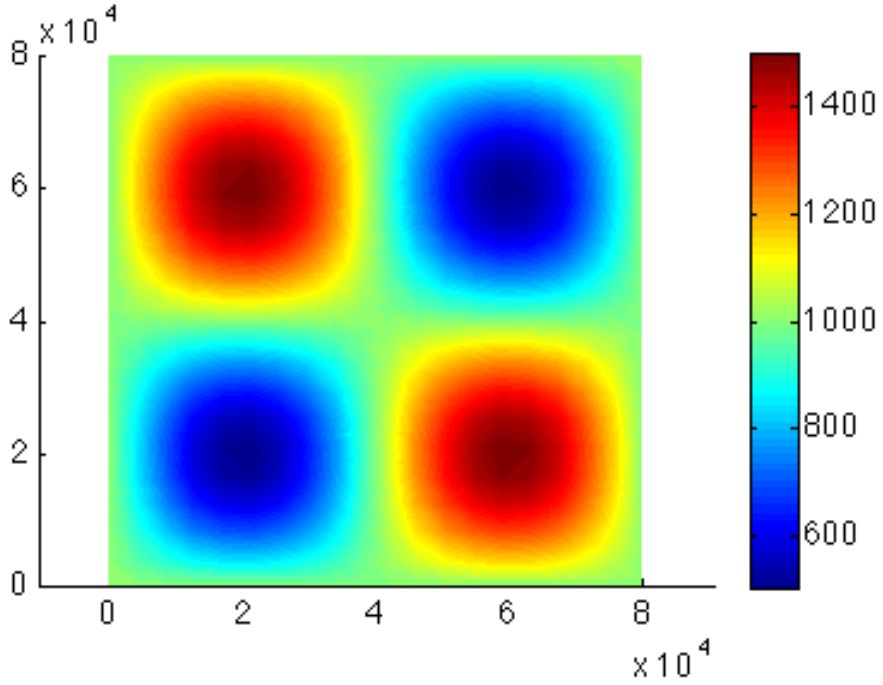
- Mesh size: 80,000 meters
- Nodes in each direction: 20
- All grounded: default



3.4.6 Parameterization

Load the preceding step. Next parameterize the model (`#help parameterize`). You will need to fill-up the parameter file (given by the name ParamFile variable). Save the given model. It is important to note that the values are not important as we are dealing with a no-sliding flux. The values will be overridden by the basal boundary conditions. Take care of the size of the parameters.

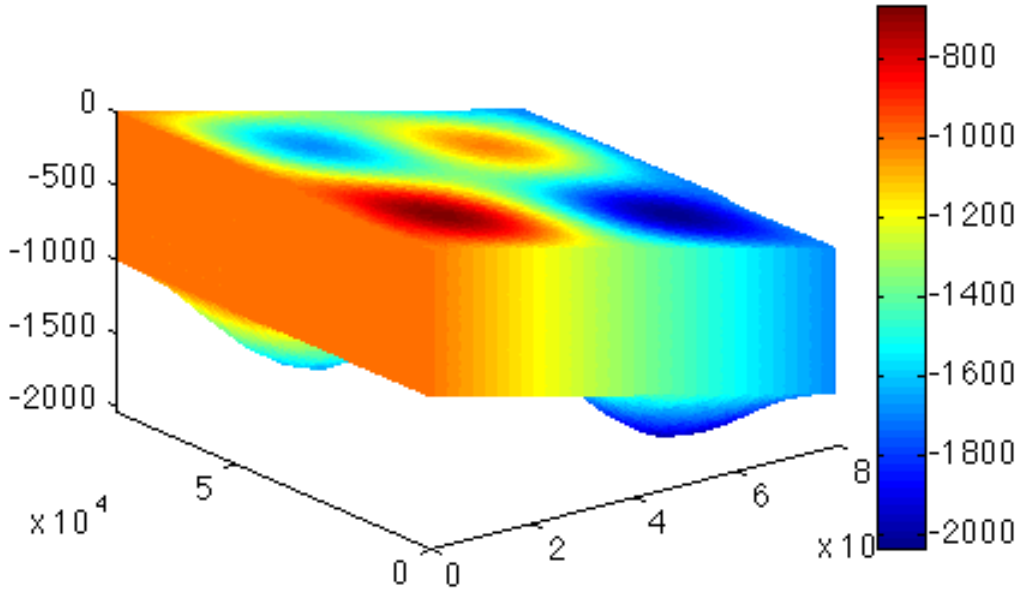
- Mesh size: 80,000 meters
- Nodes in each direction:20
- All grounded: default
- Ice-flow parameter: `B=6.8067 x 10^7 Pa s^1/n`
- Glen's exponent: n=3



3.4.7 Extrusion

Load `Parameterization` model. The action here is to extrude the preceding mesh. Next vertically extrude the preceding mesh `(#help extrude)` with only 5 layers exponent 1. Plot the 3D geometry and save the model.

- Mesh size: 80,000 meters
- Nodes in each direction: 20
- All grounded: default
- Ice-flow paramter: `B=6.8067 x 10^7 Pa s^1/n`
- Glen's exponent: n=3
- 5 layer extrusion



3.4.8 Flow Equation

Load the `Extrusion` model and set the approximation for the flow computation `(#help setflowequation)`. We will be using the Higher Order Model (HO). Save the model.

- Mesh size: 80,000 meters
- Nodes in each direction: 20
- All grounded: default
- Ice-flow parameter: `B=6.8067 x 10^7 Pa s^1/n`
- Glen's exponent: n=3
- 5 layers extrusion
- Flow model: HO

3.4.9 Boundary Conditions

Load the `SetFlow` model. Dirichlet boundary condition are known as SPC's, where ice is frozen to the base with no velocity. SPC's are initialized at NaN one value per vertex. Extract the nodenumbers at the base `(#md.mesh.vertexonbase)` and set the sliding to zero on the bed (Vx and Vy). Periodic boundaries have to be fixed on the sides. Create tabs with the side of the domain for x, and create maxX `(#help find)`. This command give subsets of matrices based on boolean operations. Now create minX. For y, max X and min X should be excluded. Now create min Y. Set the node that should be paired together `(#md.stressbalance.vertex_pairing)`. If we are dealing with IsmipF the solution is in masstransport. Save the given model. `(#md.masstransport.vertex_pairing=md.stressbalance.vertex_pairing)`.

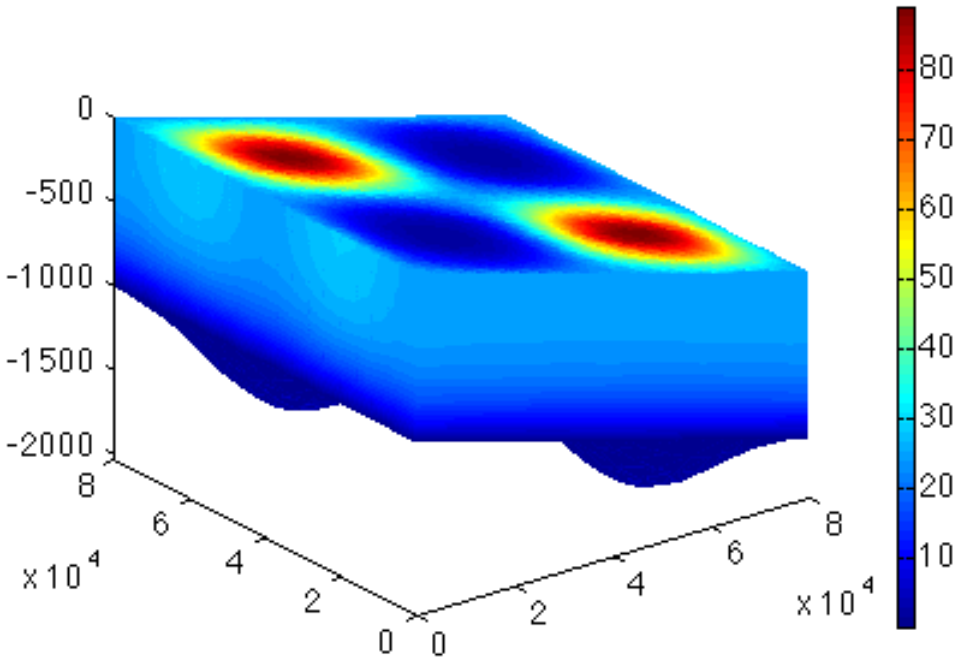
- Mesh size: 80,000 meters
- Nodes in each direction: 20

- All grounded: default
- Ice-flow parameter: `B=6.8067 x 10^6 Pa s^1/n`
- Glen's exponent: n=3
- 5 layer extrusion
- Flow model: HO

3.4.10 Solve Model

Load the `BoundaryConditions` model. Set the cluster `(#md.cluster)` with generic parameters `(#help generic)`. Set only the name and number of processes. Set which control message you want to see `(#help verbose.)` Solve `(#help solve)`. We are solving a StressBalance. Save the model, and plot the surface velocities.

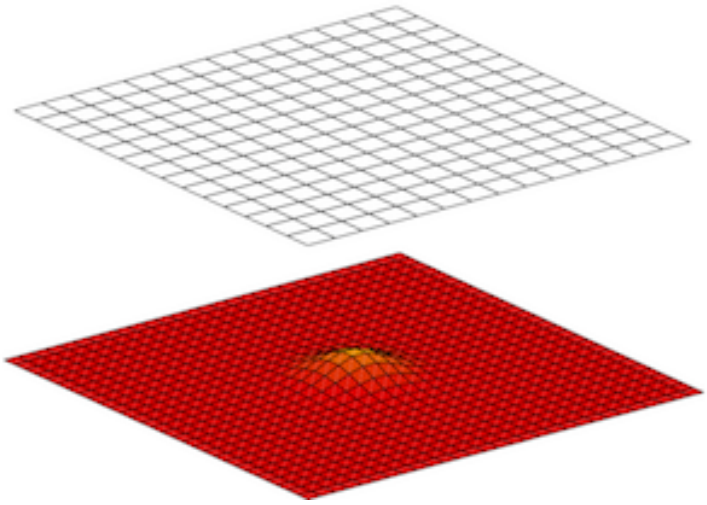
- Mesh size: 80,000 meters
- Nodes in each direction: 20
- All grounded: default
- Ice-flow parameter: `B=6.8067 x 10^7 Pa s^1/n`
- Glen's exponent: n=3
- 5 layers extrusion
- Flow model: HO



3.4.11 Test F

Square ice sheet flowing over a bump.

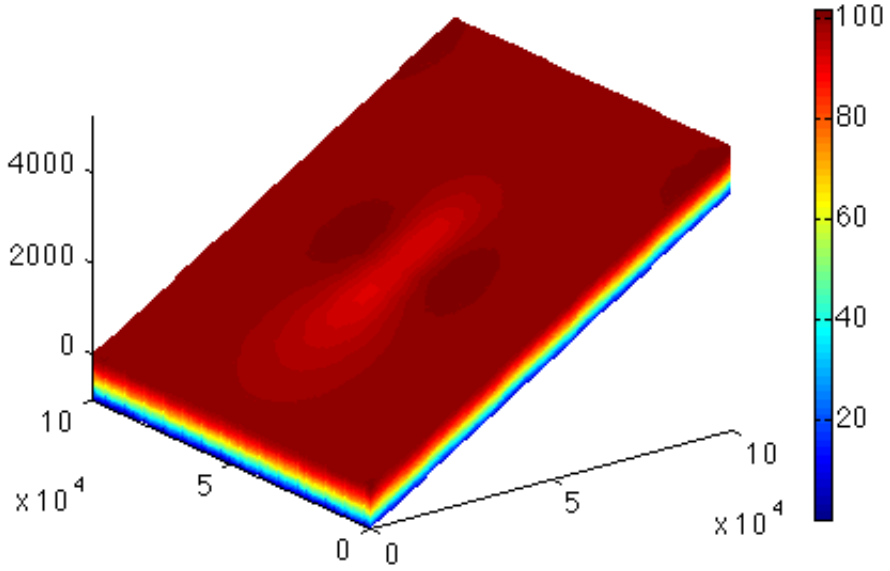
- Gaussian bumped bedrock
- Ice frozen or sliding on the bed
- Periodic boundary conditions
- Transient model until steady-state



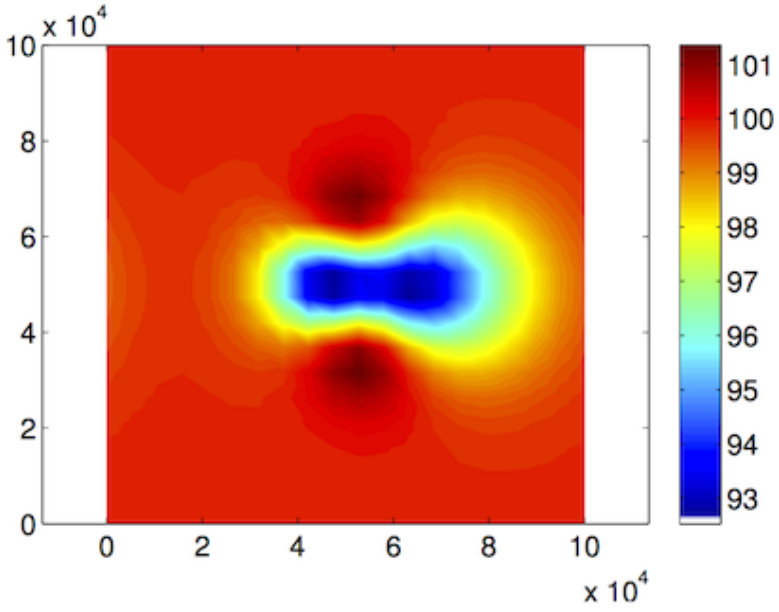
3.4.12 Actual Work and Results

Load the preceding model under the path given by the organizer with the name of the given step. Set the cluster with generic parameters. Set only the name and number of the process. Set which control message you want to see. Set the transient model to ignore the thermal model `(#md.transient)`. Define the timestepping scheme. Everything here should be provided in years `(#md.timestepping)`. Give the length of the `time step` (4 years). Give the `final_time (20*4 years time_steps)`. Now solve, we are solving for TransientSolution. Lastly plot the surface velocities. *Note: if using the cheatsheet file make sure you change line 8 to say "CheatyIsmipF.par". Here is the upper surface velocity:

Side view:



Top view:



3.5 Modeling Pine Island Glacier

3.5.1 Goals

- Model Pine Island Glacier
- Follow an example of how to create a mesh and set up the floating ice shelf of a real-world glacier
- Use observational data to parameterize the model
- Learn how to use inversions to infer basal friction and plot the results

3.5.2 Introduction

In this example, the main goal is to parameterize and model a real glacier. In order to build an operational simulation of Pine Island Glacier, we will follow these steps:

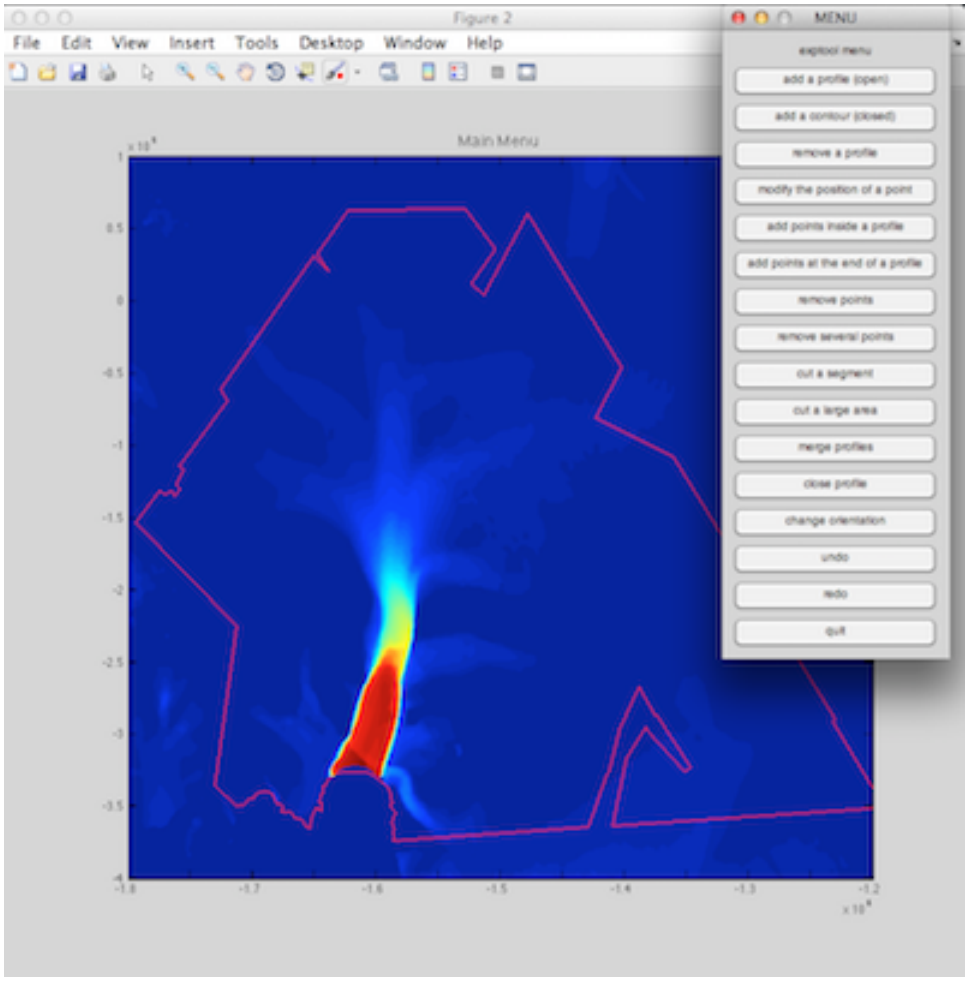
- Define the model region
- Create a mesh
- Apply masks
- Parameterize the model
- Invert friction coefficient
- Plot results
- Run higher-order simulation

Files needed for this tutorial can be found in `trunk/examples/Pig/`. The `runme.m` file contains the structure of the simulation, while the `.par` file includes most parameters needed for the model set-up. The `.exp` files are shape files that define geometric boundaries of the simulation.

Observed datasets needed for the parameterization also need to be downloaded.

3.5.3 Setting-up domain outline

We first draw the domain outline of Pine Island Glacier based on observed velocity map. First, run `PigRegion.m` in MATLAB. It produces a figure with the observed velocities:



You can then use the `exptool` to draw the model domain:

```
>> exptool('PigDomain.exp')
```

Note: if you have not downloaded the datasets, you will get the following error:

```
Could not open ../Data/Antarctica_ice_velocity.nc."
```

If this occurs, go into the `Data` directory and run the script to download the datasets. You will not be able to proceed until you do so.

This example shows you how to create your own model boundary, but for the rest of the tutorial, we will be using the provided domain outline, which is `ModelDomain.bkp`. Change this file into an `.exp` file to will erase your contour:

```
>>!mv ModelDomain.bkp ModelDomain.exp
```

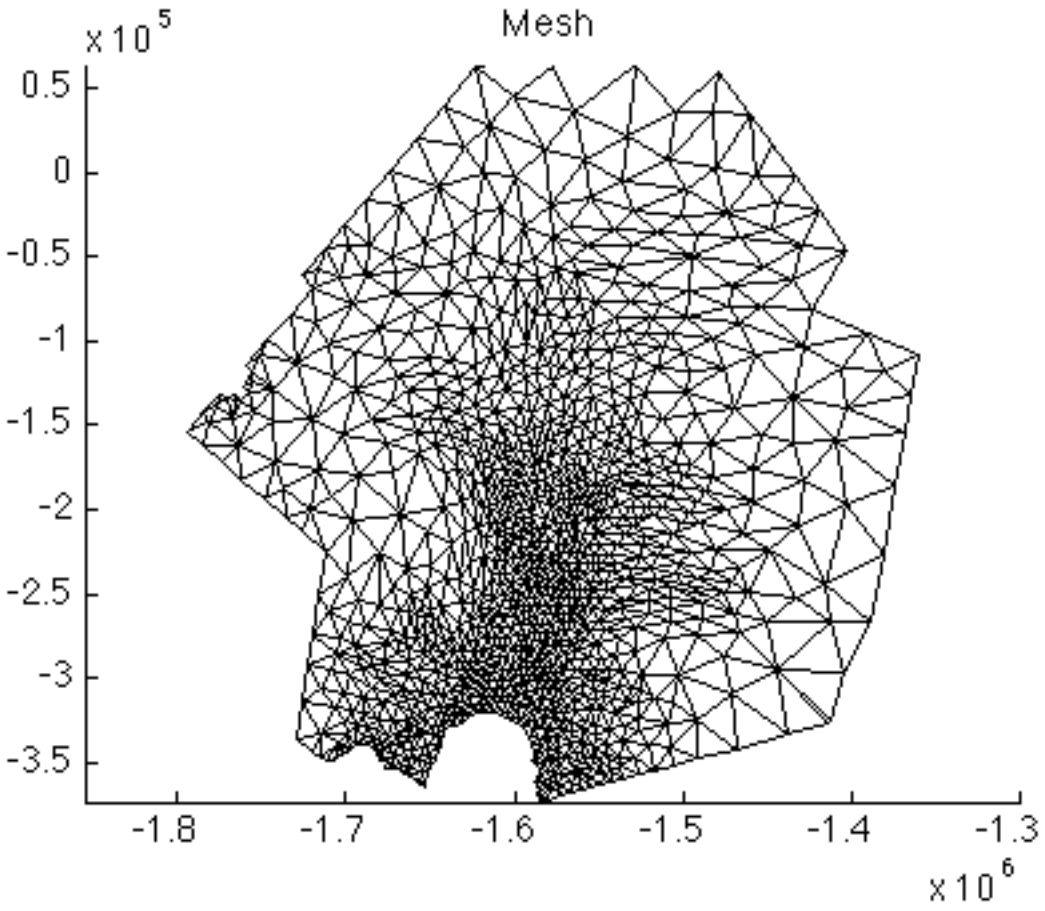
3.5.4 Mesh

The first step is to create the mesh of the model domain.

In the `runme.m` file, the mesh is generated in a multi-step process. Open the `runme.m` file and make sure that the variable `step`, at the top of the file, is set to `step=1`. In the code, you will see that in step 1 the following actions are implemented:

- a uniform mesh is created
- the mesh is then refined using anisotropic mesh refinement. We use the surface velocity as a metric
- Set the mesh parameters
- Plot the model and load the velocities from `http://nsidc.org/data/nsidc-0484.html`
- Get the necessary data to build up the velocity grid
- Get velocities (note: You can use `ncdisp('file')` to see an `ncdump`)
- Interpolate the velocities onto a coarse mesh. Adapt the mesh to minimize error in velocity interpolation
- Plot the mesh
- Save the model

Execute the `runme.m` file to perform step 1. You should see the following figure:



3.5.5 Mask

The second step of the `runme.m` creates the masks required to specify where there is ice in the domain, and where the ice is grounded.

First, we specify where the ice is grounded and floating in the domain:

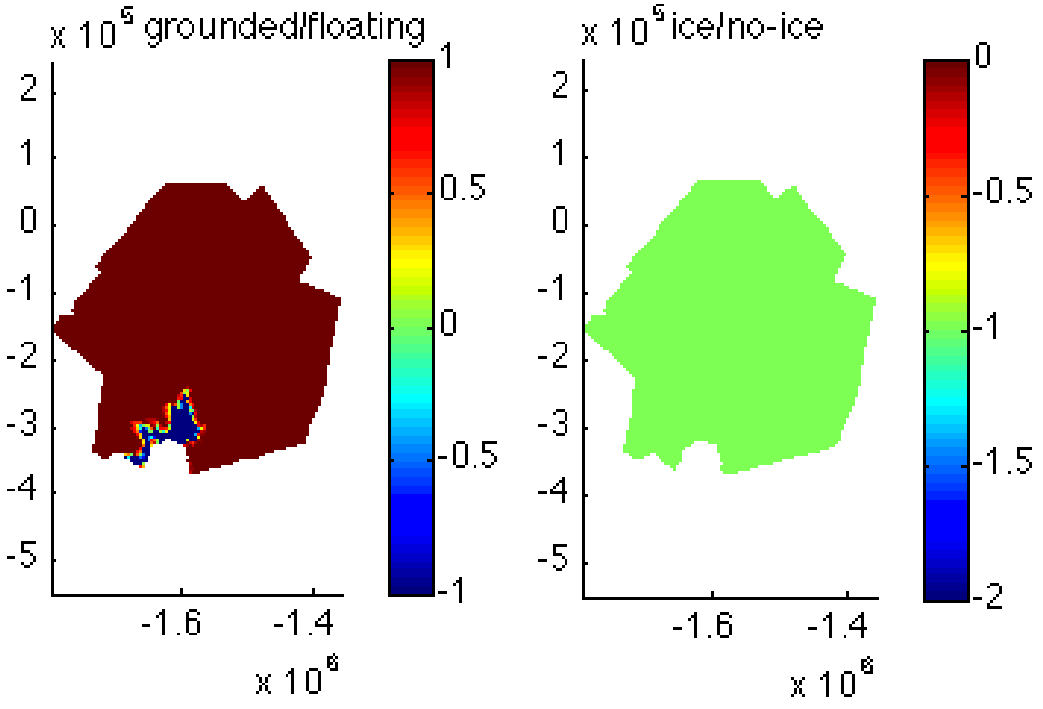
- The field `md.mask.groundedice_levelset` contains this information
  - Ice is grounded if `md.mask.groundedice_levelset` is positive
  - Ice is floating if `md.mask.groundedice_levelset` is negative
  - The grounding line lies where `md.mask.groundedice_levelset` equals zero

Then we specify where ice is present:

- The field `md.mask.ice_levelset` contains this information
  - Ice is present if `md.mask.ice_levelset` is negative
  - There is no ice if `md.mask.ice_levelset` is positiive
  - The ice front lies where `md.mask.ice_levelset` equals zero

Open `runme.m` and set `step=2`. Now, execute the `runme.m` file to run step 2.

After executing step 2, you should see the following figure that represents the mask:



3.5.6 Parameterization

Parameterization of models is usually done through a different file (`Pig.par`). Parameters which are unlikely to change for a given set of experiments are set there to lighten the `runme.m` file. In this example we use SeaRISE data to parameterize the following model fields:

- Geometry
- Initialization parameters
- Material parameters
- Forcings
- Friction coefficient
- Boundary conditions

Some parameters are adjusted in `runme.m`, as they are likely to be changed during the simulation. This is the case for the stress balance equation that is set-up using `setflowequation`

Now, change the `runme.m` file as before, and run step 3 to perform the Parameterization.

3.5.7 Inversion of basal friction

The friction coefficient is inferred from the surface velocity using the following friction law:

$$\tau_b = -\beta^2 N^r \|\mathbf{v_b}\|^{s-1}\mathbf{v_b} \tag{3.5}$$

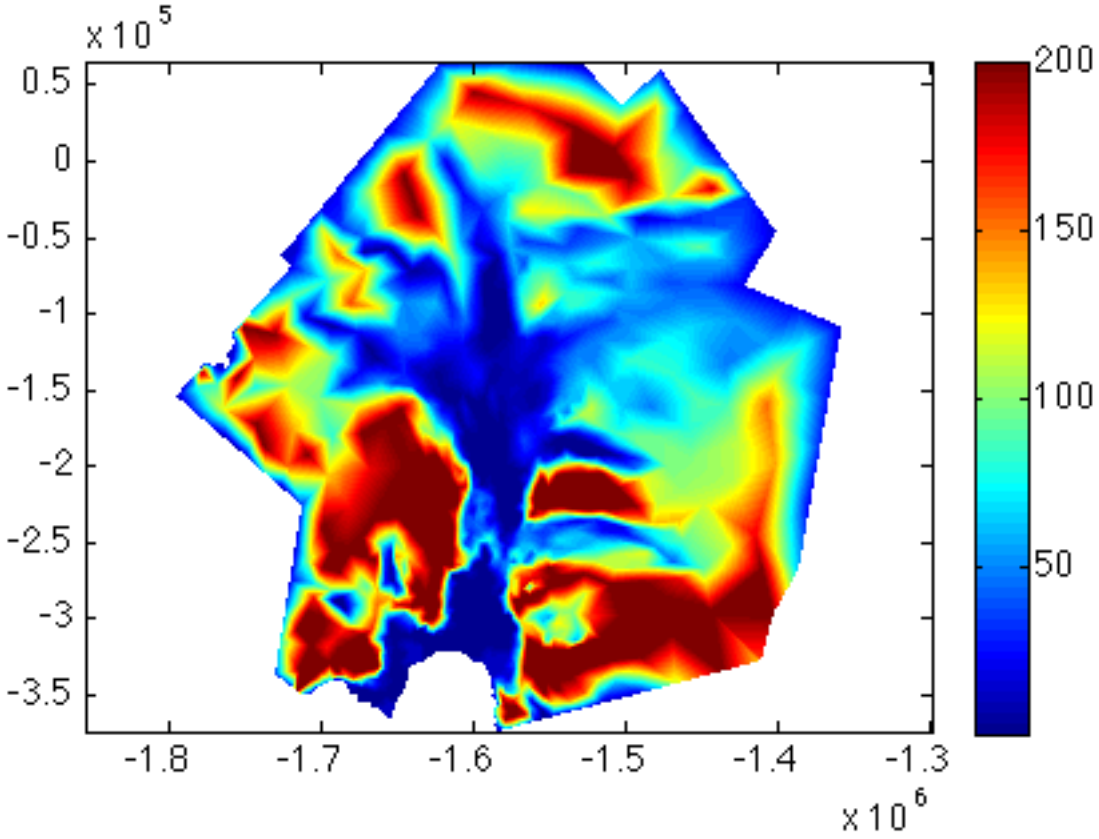
- τ_b : Basal drag
- N: Effective pressure
- v_b: Basal velocity (equal surface in SSA approximation)
- r: Exponent (equals q/p of the parameter file)
- s: Exponent (equals $1/p$ of the parameter file)

The procedure for the inversion is as follows:

- Velocity is computed from the SSA approximation
- Misfit of the cost function is computed
- Friction coefficient is modified following the gradient of the cost function

All the parameters that can be adjusted for the inversion are in `md.inversion`.

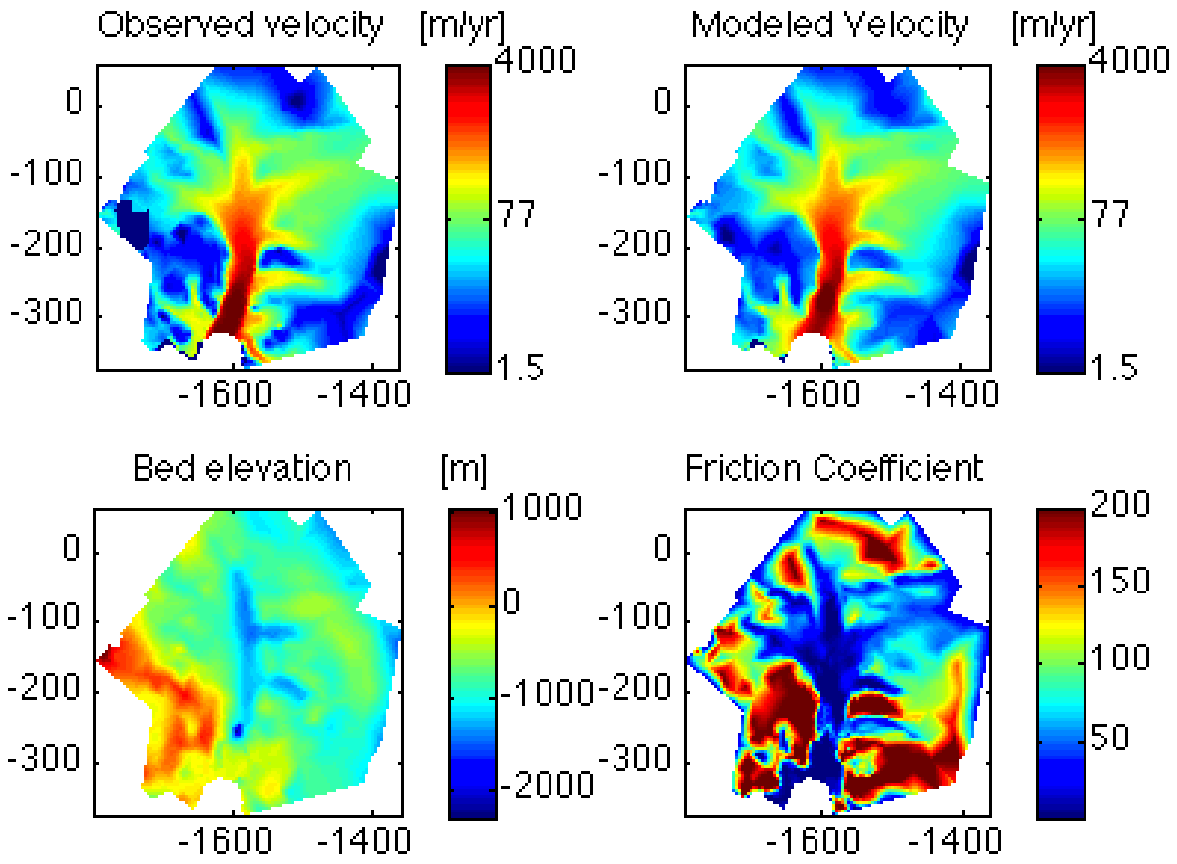
Run step 4 and look at the results, they should be similat to the figure below:



3.5.8 Plot results

Plotting ability are mainly based on `plotmodel` for simple graphs. However, you can also use or create your own routines.

Change the step to 5 and run the simulation; it should create the following figure:



3.5.9 Higher Order (HO) Ice Flow Model

The last step of this tutorial is to run a forward model of Pine Island Glacier with the Higher-Order stress balance approximation.

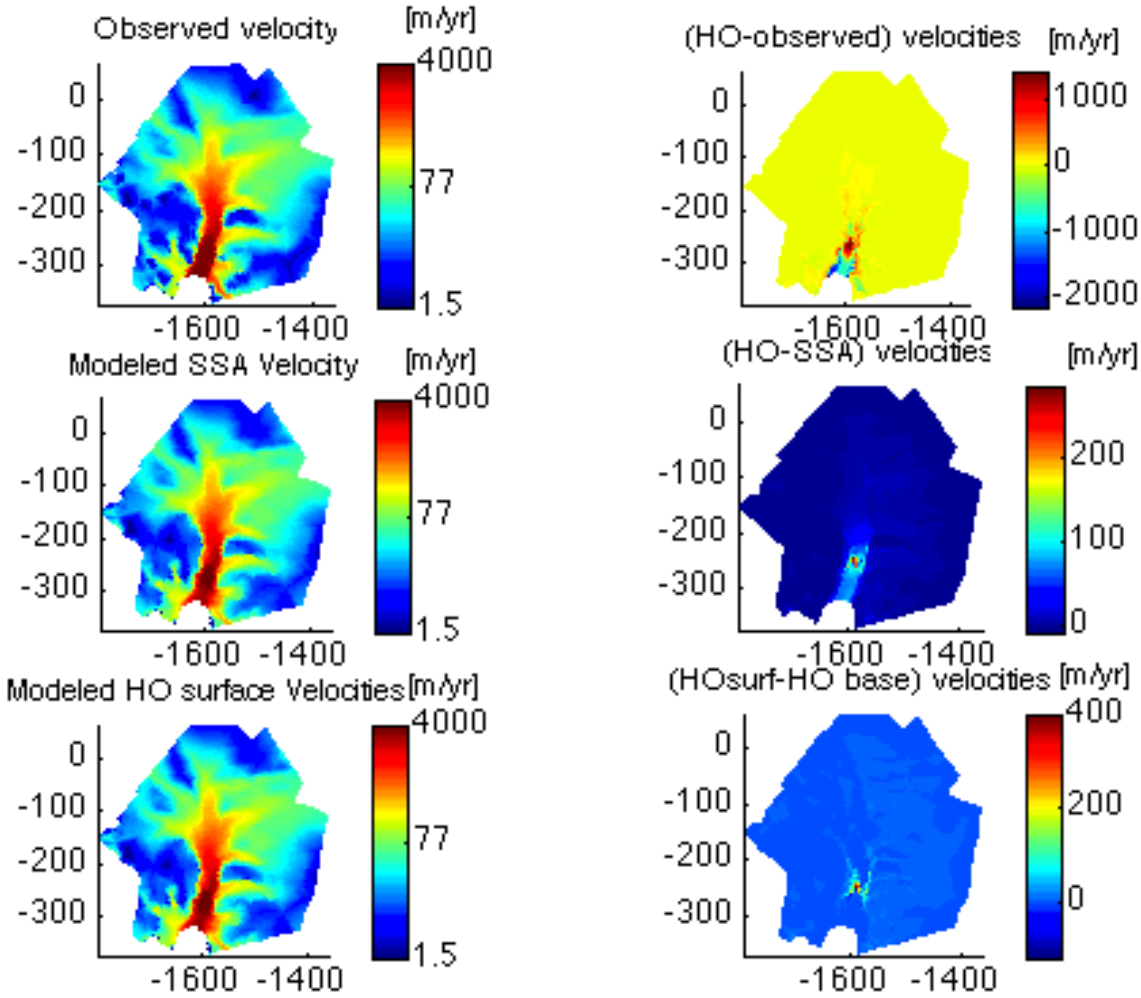
The following steps need to be performed in `step 7` of the `runme.m` file:

- Load the previous step
  - Model to load is `Control_drag`
- Disable the inversion process
  - Change `iscontrol` to zero the inversion flag (`md.inversion`)
- Extrude the mesh
  - `help extrude`
  - Keep the number of layers low to avoid long computational time
- Change the stress balance approximation
  - Use the function `setflowequation`
- Solve
  - We are still solving for a `StressBalanceSolution`

- Save the model as in the preceding steps

If you need help, the solution is provided below.

Step 7 provides a comparison of the shelfy-stream and higher-order approximations. The following figure should be created if you run step 7:



3.5.10 Solutions for step 6

```
if step==6
md = loadmodel('./Models/PIG_Control_drag');
md.inversion.iscontrol=0;

disp('   Extruding mesh')
number_of_layers=3;
md=extrude(md,number_of_layers,1);

disp('   Using HO Ice Flow Model')
md=setflowequation(md, 'HO', 'all');
```

```
md=solve(md,StressbalanceSolutionEnum);

save ./Models/PIG_ModelHO md;
end
```

3.6 Pine Island Glacier, melting experiment

3.6.1 Goals

This example is adapted from the results presented in *Seroussi et al.* [2014b]. We model the impact of different external forcings on the dynamic evolution of Pine Island Glacier. The main objectives are to:

- Run transient simulations (10 years) of a real glacier
- Change external forcings
- Compare the impact of changes on glacier dynamics and volume

Files needed to run this tutorial are located in `trunk/examples/PigSensitivity/`. This tutorial relies on the Pine Island tutorial, so make sure to complete it first.

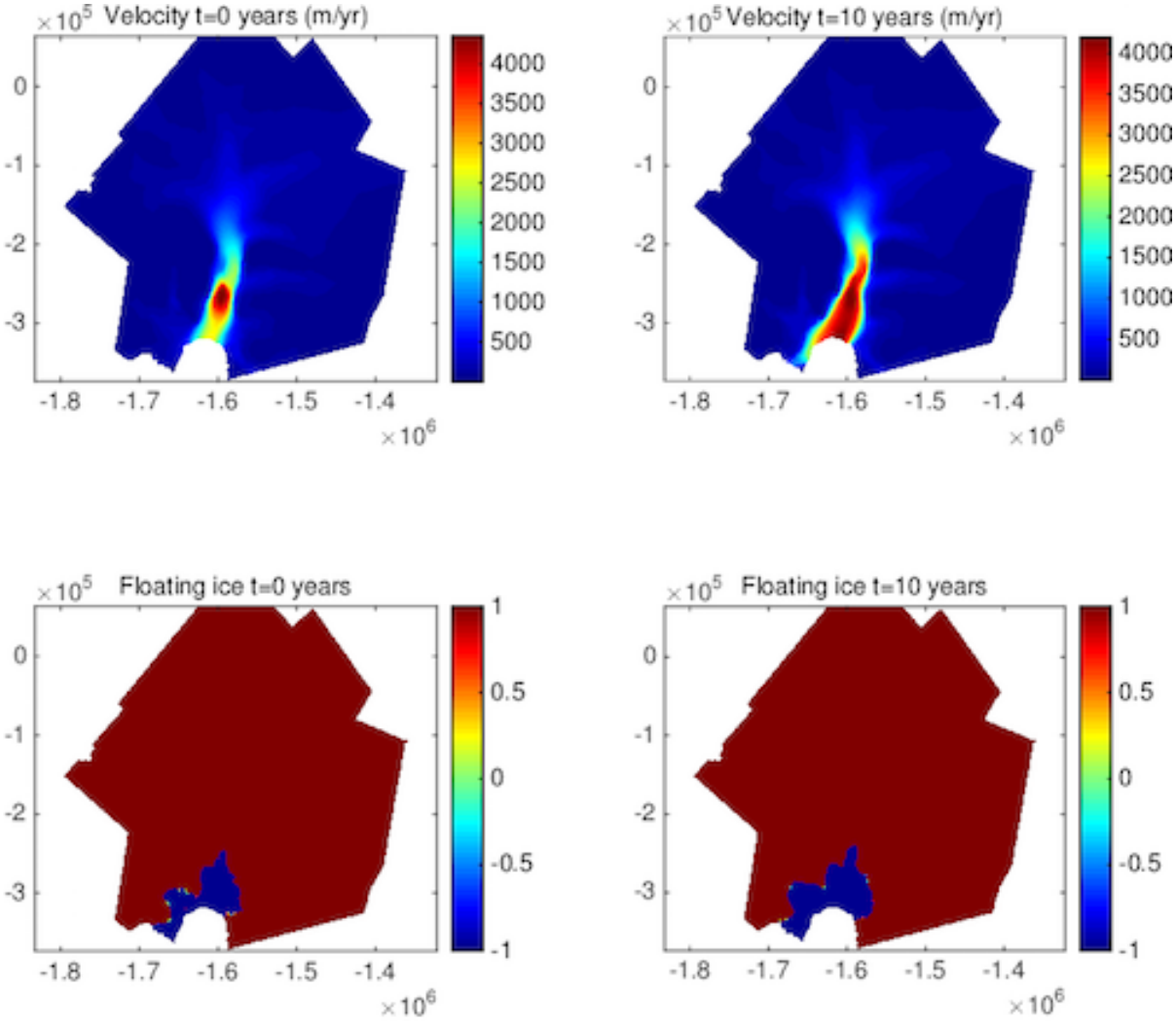
3.6.2 Evolution over 10 years

We first run a simulation of Pine Island Glacier over a 10 year period, starting from the `Pig` tutorial.

In the `runme.m` file, several parameters are adjusted before running the transient model. Open `runme.m` and make sure that the variable `step`, at the top of the file, is set to `step=1`. In the code, you will see that in step 1 the following actions are implemented:

- Load model from the `Pig` tutorial
- Apply some basal melting rate
  - On grounded ice: `md.basalforcings.groundedice_melting_rate`
  - On floating ice: `md.basalforcings.floatingice_melting_rate`
- Specify time step length and run duration in `md.timestepping`
- Disable inverse method in `md.inversion.iscontrol = 0`
- Indicate what components of the transient to activate
  - `md.transient.ismasstransport`
  - `md.transient.isstressbalance`
  - `md.transient.isthermal`
  - `md.transient.isgroundingline`
  - `md.transient.ismovingfront`
- Request additional outputs
- Solve transient solution

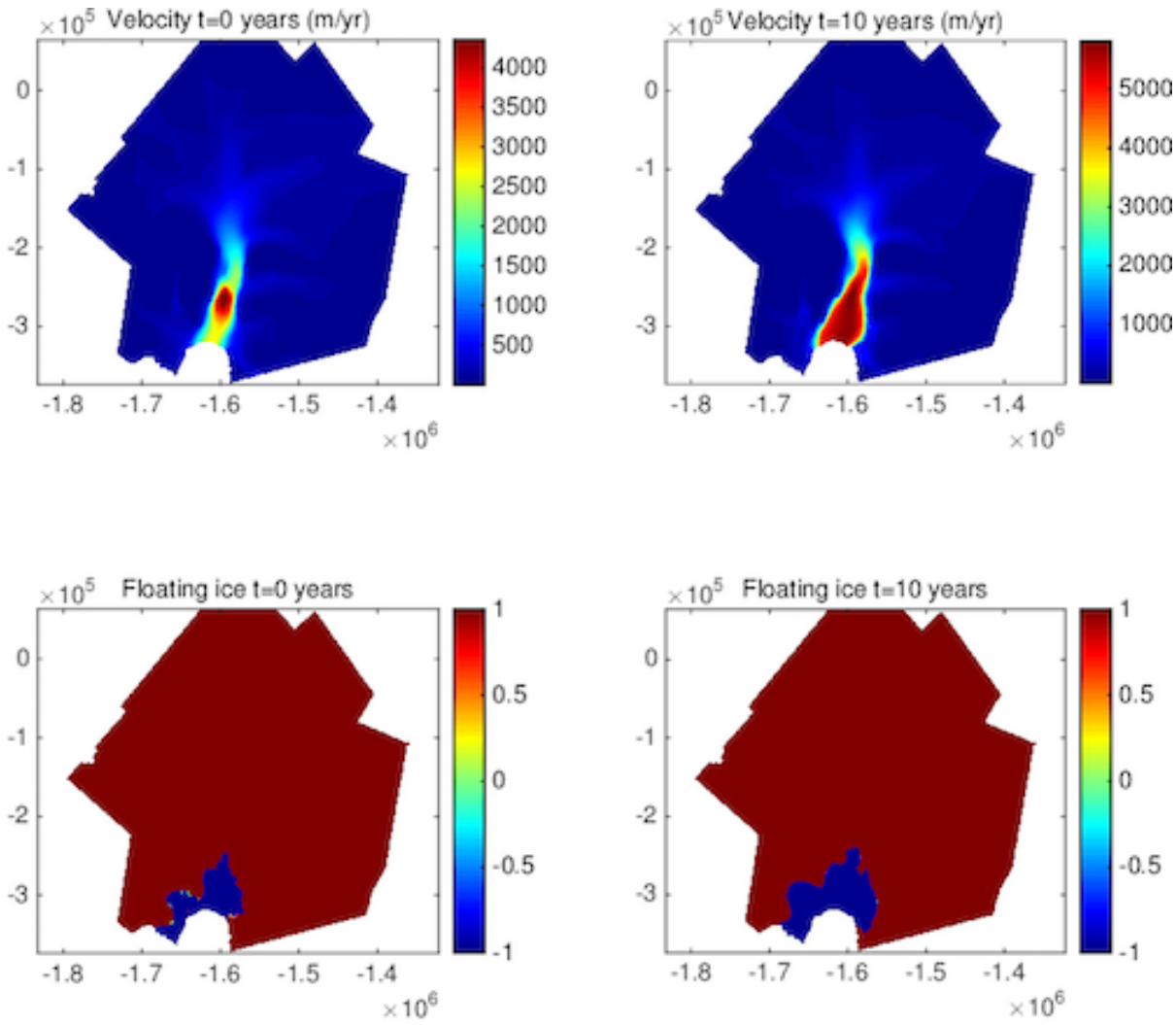
Execute `runme` to perform step 1. The following figure shows the evolution of the ice velocity and grounding line positions at the beginning and at the end of the simulation.



3.6.3 Increased basal melting rate

In this second step, we increase the basal melting rate under the floating portion of the domain from 25 to 60 m/yr. The other parameters remain the same as in the previous step.

Open `runme.m` and change the step at the top of the file to `step=2`, then run the simulation. The following figure shows the evolution of ice velocity and grounding line evolution for the increased melting scenario:



3.6.4 Retreat of ice front position

In this third step, we would like to test the sensitivity of Pig to calving events and retreat the position of the ice front. We first need to create a new contour of the region to be removed from the domain. Use `exptool` to create a new `RetreatFront.exp` contour that include the portion of floating ice that should calve off.

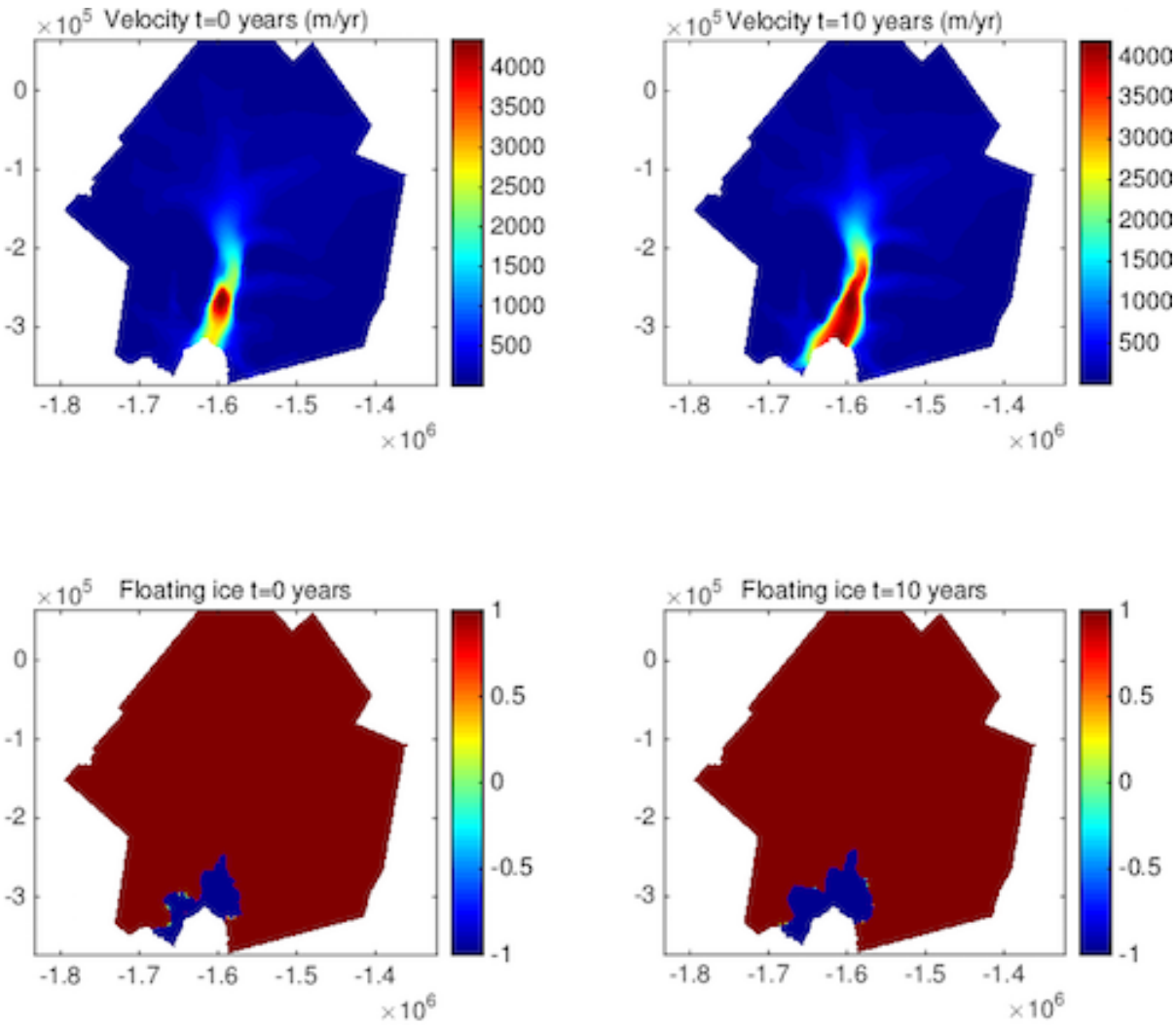
Then extract the domain from the initial model, exluding the `RetreatFront.exp` area using the `extrude` routine.

```
>> md2=modelextract(md,~RetreatFront.exp)md2=modelextract(md,~RetreatFront.exp)
```

As this operation changes the model domain, some parameters and boundary conditions have the be adjusted or redefined.

The boundary conditions are reset with `SetMarineIceSheetBC` and the model can then be solved.

Open `runme.m` and change the step at the top of the file to `step=3`, then run the simulation. The following figure shows the evolution of ice velocity and grounding line evolution with the new ice front:



3.6.5 Change in surface mass balance

In this last step, we change the surface mass balance, while the other parameters remain similar to the previous simulations.

Open `runme.m` and implement the changes needed to investigate the impact of the surface mass balance, similar to what was done with the other external forcings in the previous steps. These changes are:

- Load model from the `Pig` tutorial
- Change the surface mass balance
- Verify the ocean-induced melting rate
  - On grounded ice: `md.basalforcings.groundedice_melting_rate`
  - On floating ice: `md.basalforcings.floatingice_melting_rate`
- Specify time step length and run duration in `md.timestepping`
- Disable inverse method in `md.inversion.iscontrol`
- Indicate what components of the transient to activate

  - `md.transient.ismasstransport`
  - `md.transient.isstressbalance`
  - `md.transient.isthermal`
  - `md.transient.isgroundingline`
  - `md.transient.ismovingfront`

- Request additional outputs
- Solve transient solution

Don't forget to change `step` at the top of the `runme.m`.

Below is the solution to make this change:

```
if step==4
%Load model
md = loadmodel('./Models/PIG_Transient');

%Change external forcing basal melting rate and surface mass balance)
md.basalforcings.groundedice_melting_rate=zeros(md.mesh.numberofvertices,1);
md.basalforcings.floatingice_melting_rate=25*ones(md.mesh.numberofvertices,1);
md.smb.mass_balance=2*md.smb.mass_balance;

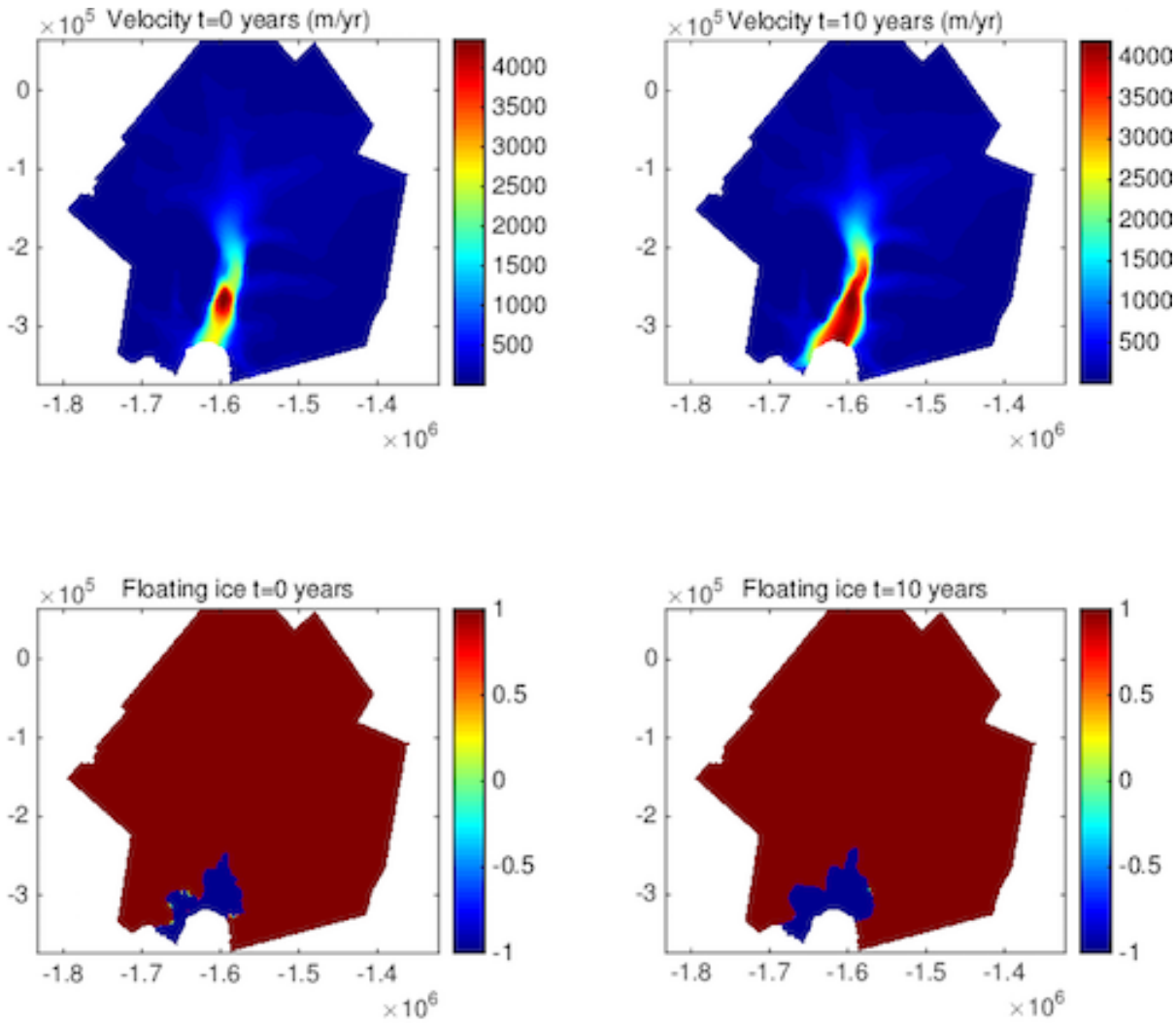
%Define time steps and time span of the simulation
md.timestepping.time_step=0.1;
md.timestepping.final_time=10;

%Request additional outputs
md.transient.requested_outputs={'default','IceVolume','IceVolumeAboveFloatation'};

%Solve
md=solve(md,TransientSolutionEnum);

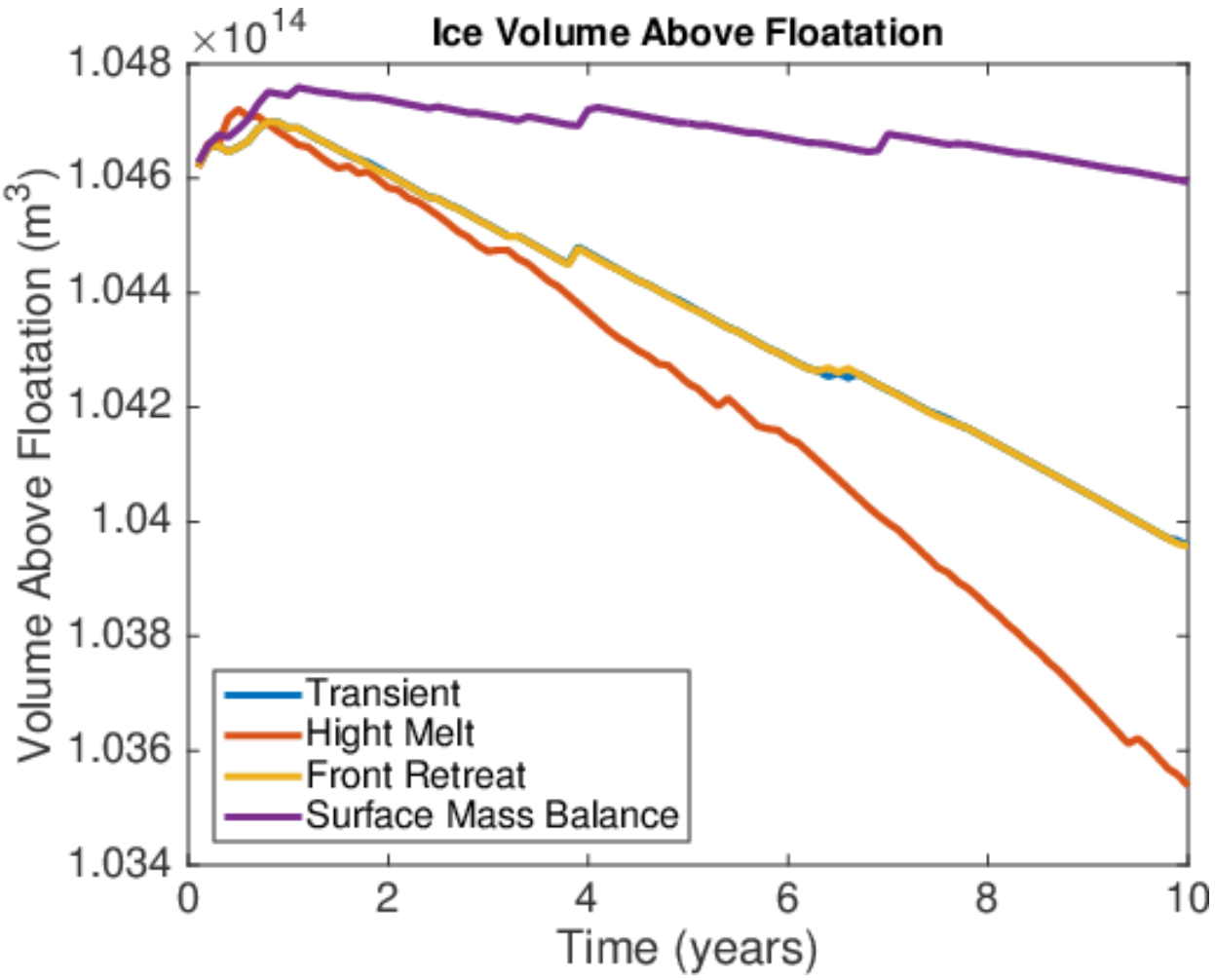
%Save model
save ./Models/PIG_SMB md;
end
```

Here is an example of velocity change and grounding line evolution when the surface mass balance is doubled:



3.6.6 Evolution of the ice volume above floatation

In the previous steps, we investigated the impact of changes in external forcings on ice flow dynamics (grounding line evolution and glacier acceleration). We can also see how these changes impact the glacier volume and its contribution to sea level rise. To do so, we use the additional output `IceVolumeAboveFloatation` requested in the transient simulation. The following figure shows the evolution of the volume (in Gt/yr) above floatation for the four scenarios performed previously.



3.7 Uncertainty quantification (requires Dakota)

3.7.1 Goals

- Use ISSM to assess how errors in model inputs propagate through a 2D SSA steady state ice flow model
- Use ISSM to assess how ice flow model diagnostics (e.g. velocity, mass flux, volume) can be affected by perturbations to input in other parts of the model domain
- Become familiar with the uncertainty quantification (DAKOTA-based) tools available in ISSM

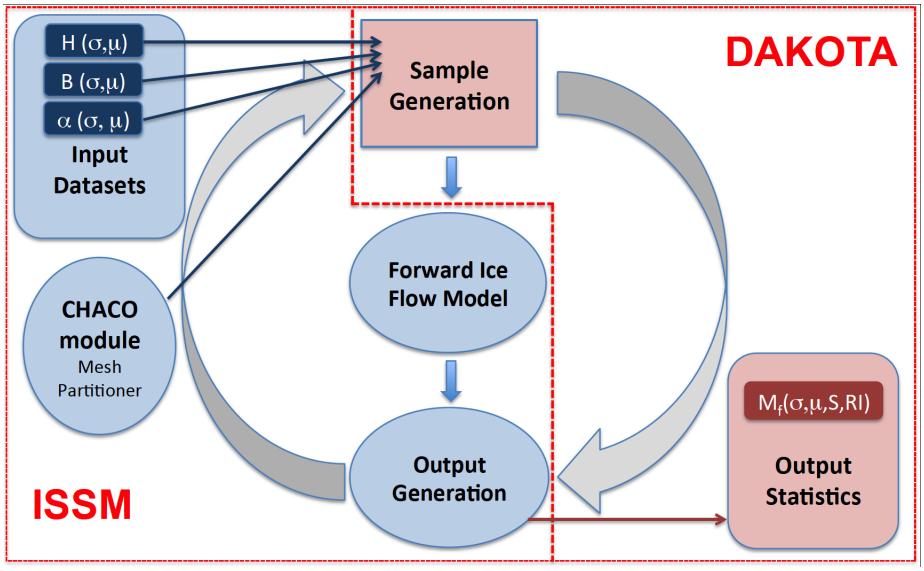
Go to `trunk/examples/UncertaintyQuantification/` to do this tutorial.

3.7.2 Introduction

This experiment will use the model of Pine Island Glacier that was saved in the previous PIG tutorial. It aims to use the ISSM-DAKOTA integrated model system to (1) quantify the uncertainties of model output in response to errors in model input and (2) quantify sensitivities of model output to spatial perturbations in model input.

- Our model inputs: ice thickness, ice rigidity, and basal friction.
- Our model outputs: mass flux at 13 flux gates across PIG.

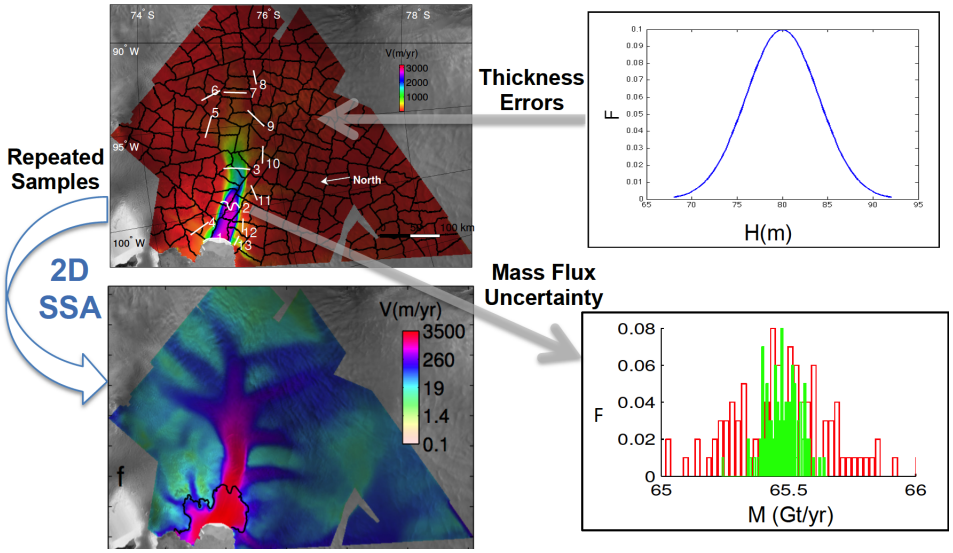
Our Uncertainty Quantification (UQ) methods are based on the Design Analysis Kit for Optimization and Terascale Applications (DAKOTA) software [*Eldred et al.*, 2008], which is embedded in ISSM. The following diagram illustrates the relationship between ISSM and DAKOTA. The ISSM mesh must be partitioned (i.e. vertices can be grouped together so that DAKOTA varies them together - this is helpful when you want to vary equal areas over the unstructured mesh). To partition the mesh, you can do so linearly (one partition per vertex), or you can use an external package software like Chaco to weight vertices and create the partitions you desire. DAKOTA is resposible for varying the provided inputs in the user-defined way (uniform, normal, etc.) for each mesh partition and then launching an ISSM run with the perturbed forcing. DAKOTA is also responsible for creating statistics for output, which are also user defined. Output diagnostics include ice mass flux through defined gates and scalar output (e.g. Ice Volume, Total SMB, etc.).



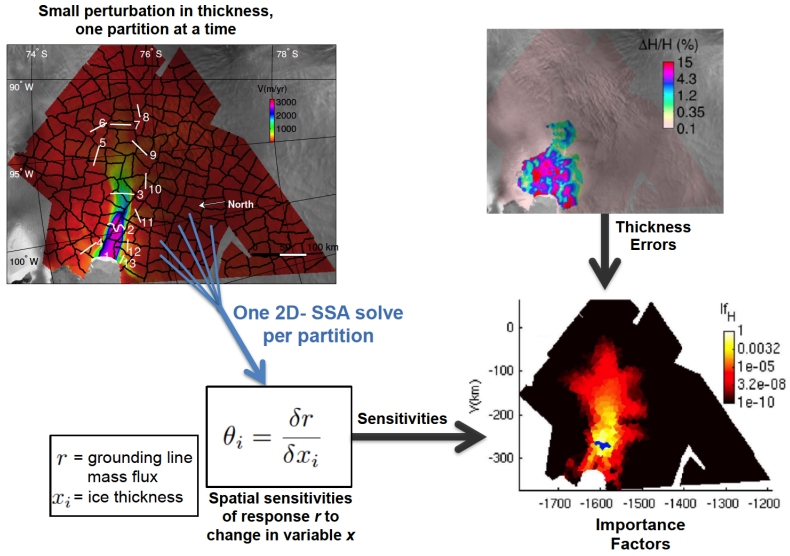
Tutorial steps to be taken:

- Begin by loading results from the `examples/Pig` tutorial (the end of basal friction inversion)
- Load ice thickness cross-over errors from IceBridge 2009 WAIS campaign
- Run sampling analysis using ice thickness cross-over and mass flux diagnostics
- Run sensitivity analysis using ice thickness, ice rigidity, and basal friction as inputs and mass flux diagnostics
- Plot results: partition, sampling, and sensitivities

Samping Analysis: Quantify the uncertainties of model output (diagnostics like mass flux, Ice Volume, Max Velocity) in response to errors in model input. The figure below illustrates an example of Sampling errors in ice thickness. The result for each gate, is a histogram of Mass Flux (one value per each model run, or sample). Below is the resulting histogram for mass flux gate 2.



Sensitivity Analysis: Quantify sensitivities of model output to small spatial perturbations in model input. The figure below illustrates how this is accomplished. One by one, partition input is changed by a small percentage, and a model run is launched. For this specific run, changes in model diagnostics (output) are assessed by DAKOTA. This is done for each partition, such that the number of model runs is equal to the number of mesh partitions. In the end, every diagnostic is associated with a sensitivity value at every partition. In this way, we can make a map of sensitivities for each diagnostic. Sensitivities can also be ranked, for each diangistic, in importance. One such example of DAKOTA output is the 'importance factor', or sensitivities scaled by error margins [*Larour et al.*, 2012b, a], illustrated below as UQ sensitivity analysis output for mass flux gate 2.



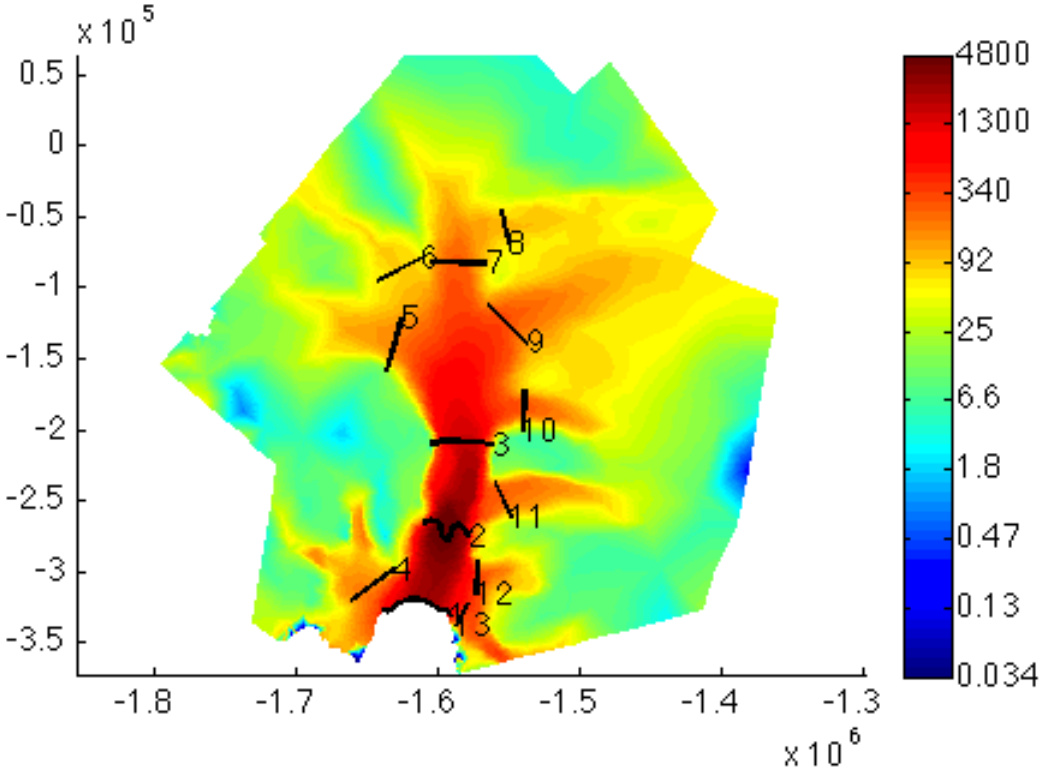
For maniscript examples of these studies, see *Larour et al.* [2012b, a]; *Schlegel et al.* [2013, 2015].

3.7.3 Flux Gates

Flux gates are ARGUS (`*.exp`) files found in `./MassFluxes`. The gates are positioned across PIG at the inset of tributary glaciers.

Mass fluxes will be computed in (Gt/yr) for all of these gates (using the depth-average ice velocity, ice thickness, and ice density)

Run step 1 of the `runme.m` to plot the gates overlayed over the PIG surface velocities.



3.7.4 Loading Cross-Over Errors

For ice thickness errors we will use McCords cross-over errors from CReSIS. First you will load errors. Some of these errors are too large, too small, or need to be interpolated onto a larger domain (you will filter these out). Load cross overs `'../Data/CrossOvers2009.mat'`. Interpolate cross over errors over our mesh vertices. Avoid `NaN` values. Filter out unrealistic error ranges. Avoid large unrealistic values. Transform into absolute errors and setup a minimum error everywhere.

Run Step 2 in the `runme.m` to load the crossover errors.

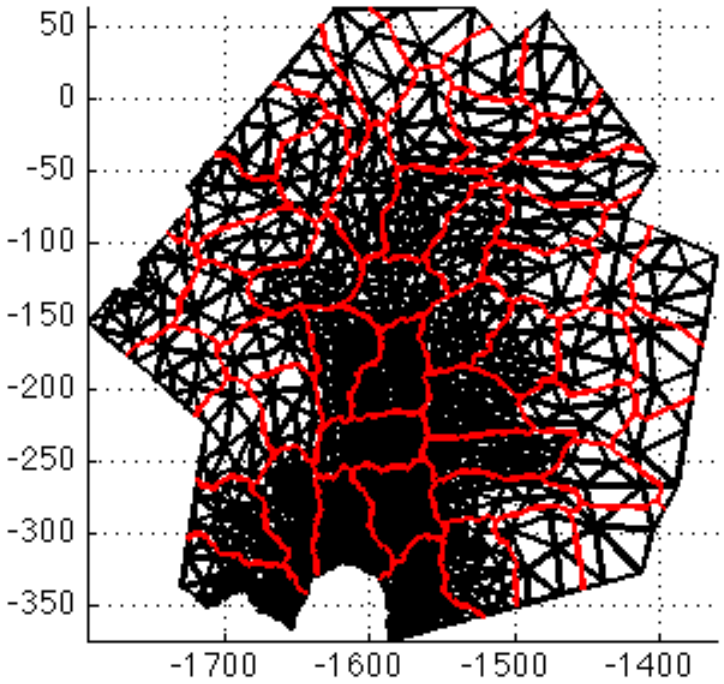
3.7.5 Sampling Analysis

In order to accomplish the sampling step, we must first partition the mesh into equal area partitions. We'll start with 50. You can try and play with the package for partitioning ('chaco' or 'linear'), the number of partitions, and weighting ('on' or 'off').

- See lines 69-73 in the `runme.m` file
- Run step 3

To plot the corresponding partition over a plot of the mesh:

- See lines 243-253, Step 5.



Note that after using Chaco, your partitions may look different from those illustrated here, because there is a randomness to the Chaco algorithm, and results differ on different computer systems.

Second, we must define our UQ input. Here, we will sample ice thickness (H), so we must define errors on each partition for H with a corresponding PDF (Probability Density Function). Here we calculate the crossover errors on each partition. In this example, we will sample a normal error distribution around every partition. To do so, we need to specify to DAKOTA that we want a normal sampling, and we must provide the standard deviation of error at every partition. Because crossover errors represent the full range of thickness errors, we assume this represents a 6-sigma normally distributed spread. Therefore, we set the standard deviation equal to the crossover error at a particular location, divided by 6:

- See lines 75-81

Third, we must set up the desired diagnostics, or output responses. In this case, we choose ice mass flux at 13 flux gates around the domain:

- See lines 83-109

For all responses, we specify a string identifier and the desired output confidence intervals. We also need to specify an `*.exp` file to define each flux gate, and directory where to find the latter.

- See lines 111-125

Finally, we need to designate a sampling strategy. Options include `'nond_samp'` for sampling or `'nond_l'` for local reliability method/sensitivity analysis, following DAKOTA guidelines. Because

this step is a sampling exercise, we choose `'nond_samp'`. We set the number of samples (30 for now) and also choose which sampling algorithm (e.g. `'lhs'` or `'random'`) DAKOTA will use.

In addition, we setup persistent parameters, this includes parallel concurrency, verbosity, and data backup.

- See lines 134-139

We also have to tighten the solver tolerance (in order to avoid spurious sensitivities to develop) before solving.

- See lines 141

Because the ISSM-DAKATA framework now runs in parallel, our implementation requires that DAKOTA runs with a master/slave configuration. This means that at least 2 cpu's are needed to run the UQ, such that:

```
md.cluster.np=md.qmu.params.processors_per_evaluation*N
```

where `N` is an integer which represents the number of parallel DAKOTA threads that will run at once. In this example, we run with 4 processors. One DAKOTA thread will run on 3 processors (slave), while 1 processor (always) serves as the master.

- See lines 152-153

Don't forget to deactivate inversion (`iscontrol=0`), and to activate UQ run (`isdakota=1`).

- See lines 156

Note that results will be in `md.results.dakota` and `md.qmu.results`.

To initiate the UQ sampling, run step 3 in the `runme.m` file.

3.7.6 Sensitivity Analysis

Next we quantify importance factors (sensitivities scaled by error margins) for model inputs: ice thickness (H), basal friction (α), and ice rigidity (B). We specify a 5% error margin on all inputs. For partitions, we choose 10 partitions, and setup for model diagnostics is the same as for sampling analysis.

- To add model inputs, and specify a 5% purterbation range:
  - See lines 175-179
- To specify new sensitivity method, tell DAKOTA to use local reliability or `'nond_l'`:
  - See lines 213-215

We specify the same parallel cpu configuration, and we solve the same way as in step 3. Note this time, we turn DAKOTA verbosity on as an example.

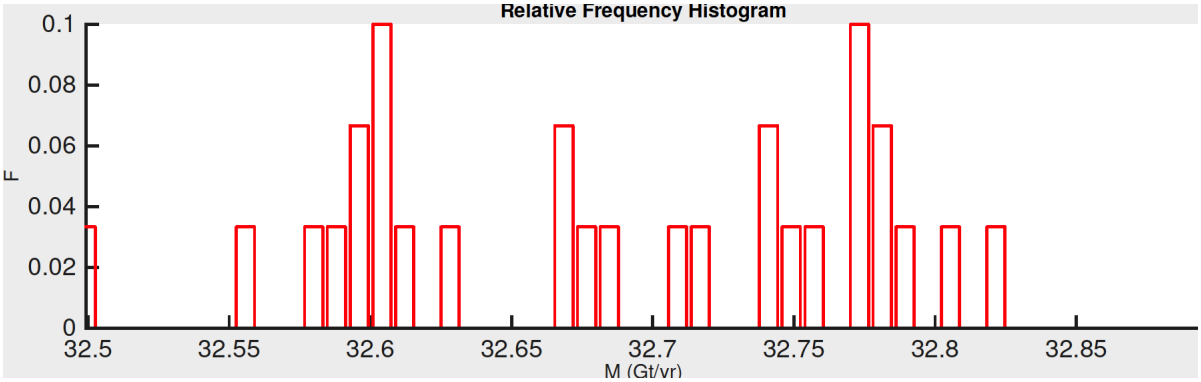
- See lines 228-238

Run step 4 to launch the sensitivity runs.

3.7.7 Plot Results

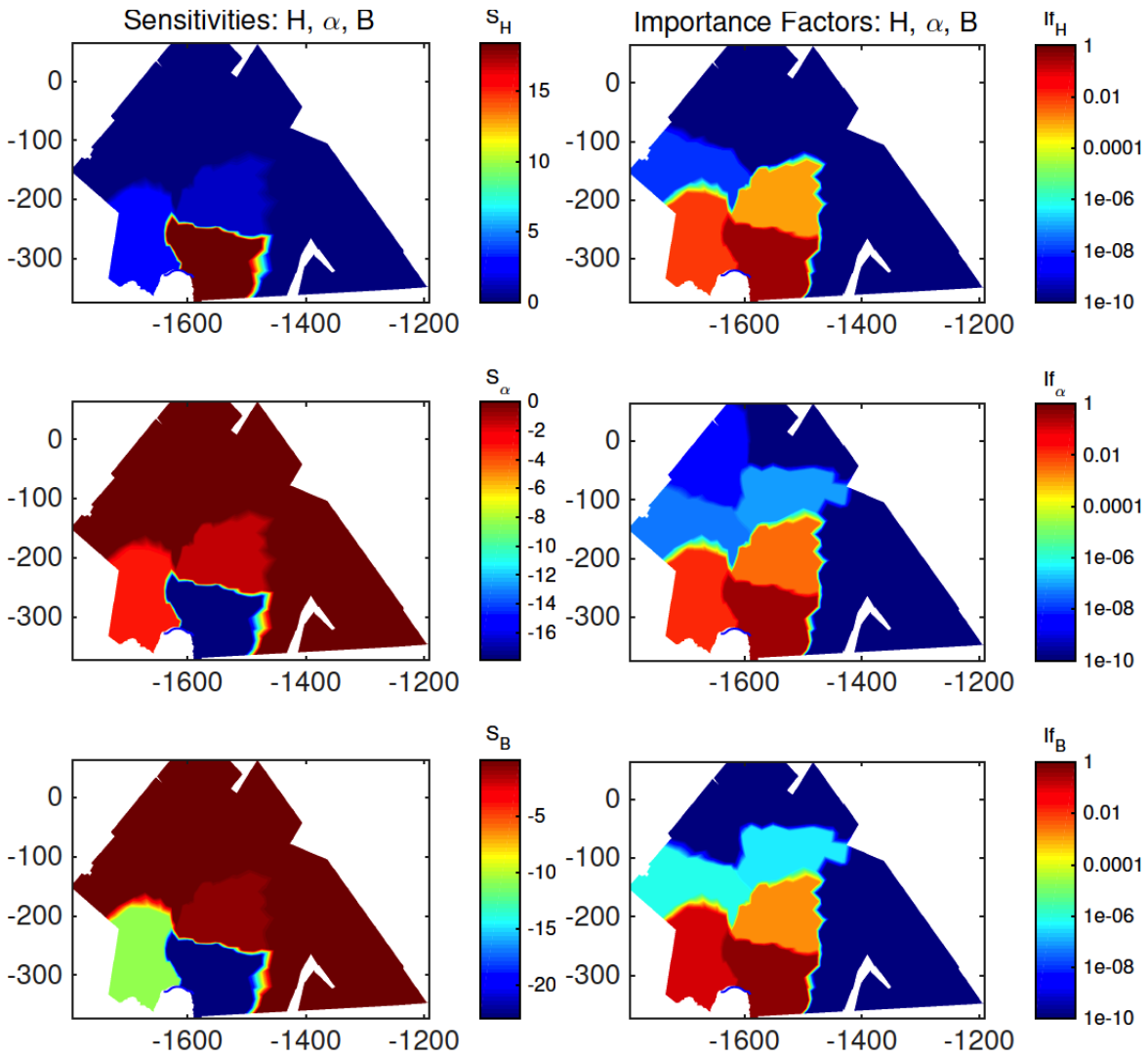
Plot Sampling Results: In order to plot the results, we extract the results for one of the mass flux gates, and display a histogram of the sampling results for that particular gate. ISSM has a plotting function for this, `'plot_hist_norm'`. Note that ISSM mass flux results are in mass flux in m^3 water equiv/s. Here we convert to Gt/yr before we plot. Remember that your results may look different because of the randomness that is introduced into the partitions and algorithms; results may be different on different computer systems.

- `runme.m` step 6 will plot the relative frequency histogram for mass flux gate 1.
- See lines 254-272



Plot Sensitivity Results:

- To retrieve sensitivities for each model input:
  - See lines 285-287
- To plot sensitivities:
  - See lines 289-296
- To retrieve importance factors for each model input:
  - See lines 299-301
- To plot the importance factors:
  - See lines 303-310
- Run step 7, this step will result in two images. The first is the sensitivities (S), and the second in the importance factors (If, sensitivities scaled by input errors).



3.7.8 Additional Exercises

- Add diagnostic IceVolume or MaxVelocity
- Sample with a uniform distribution (See `help uniform_uncertain`)
- Sample additional variables (i.e. friction coefficient, ice rheology)
- Try qmu on a different solution type
- Change number of partitions. Note: for sensitivity this could take a while!

3.8 Jakobshavn Isbræ

3.8.1 Goals

- Construct a 2-dimensional model of Jakobshavn-Isbrae, West Greenland
- Follow a simple tutorial exercise: create and parametrize an ISSM model

- Use ISSM to invert for a basal friction parameter on a real-world domain

Change into `trunk/examples/Jakobshavn/` to do this tutorial.

3.8.2 Introduction

In this tutorial, we construct a 2-dimensional model of Jakobshavn-Isbrae, West Greenland, and use it to invert for the basal friction parameter.

3.8.2.1 Download

For this tutorial, we will use a dataset from the SeaRISE Initiative: `Greenland_5km_v1.2.nc`. This data should be saved in the `examples/Data` directory (see dataset download).

3.8.3 runme file

The `runme.m` file in `trunk/examples/Jakobshavn/` is a list of commands to be run in sequence at the MATLAB command prompt. The tutorial is decomposed into 4 steps:

1. Mesh generation (anisotropic adaptation)
2. Model parameterization (using the SeaRISE dataset)
3. Launch of the inversion for basal friction
4. Plotting of the results

We will follow these steps one by one by changing the selected step at the top in `runme.m`.

3.8.4 Step 1: Mesh generation

Open `runme.m` and make sure that the first step is activated:

```
steps = [1];
```

In the first step, we create a triangle mesh with 2,000 meter resolution using the domain outline file `Domain.exp`. We then interpolate the observed velocity data onto the newly-created mesh. We use these observations to refine the mesh accordingly using `bamg`. In regions of fast flow we apply 1,200 m resolution, and in slow flowing areas we increase the resolution to up to 15 km:

```
md=bamg(md,'hmin',1200,'hmax',15000,'field',vel,'err',5);
```

Go to `trunk/` and launch MATLAB and then go to `examples/Jakobshavn/`.

```
$ cd $ISSM_DIR
$ matlab
>> cd examples/Jakobshavn/
```

Then execute the first step:

```
>> runme
Step 1: Mesh creation
  Anisotropic mesh adaptation
  WARNING: mesh present but no geometry found. Reconstructing...
     new number of triangles = 3017
```

3.8.5 Step 2: Model parameterization

In this step parameterize the model. We set for example the geometry and ice material parameters. We use the `setmask` command to define grounded and floating areas. All ice is considered grounded for now. Type `help setmask` to display documentation on how to use this command. The model is then parameterized using the `Jks.par` file. We soften the glacier's shear margins by reducing the model's ice hardness, B, in the area outlined by `WeakB.exp` to a factor 0.3.

Open `runme.m` and make sure that the second step is activated: `steps = [2];`

```
>> runme
Step 2: Parameterization
   Loading SeaRISE data from NetCDF
   Interpolating thicknesses
   Interpolating bedrock topography
   Constructing surface elevation
   Interpolating velocities
   Interpolating temperatures
   Interpolating surface mass balance
   Construct basal friction parameters
   Construct ice rheological properties
   Set other boundary conditions
      boundary conditions for stressbalance model: spc set as observed velocities
      no surfaceforcings.precipitation specified: values set as zero
      no basalforcings.melting_rate specified: values set as zero
      no balancethickness.thickening_rate specified: values set as zero
```

3.8.6 Step 3: Control method

In the parameterization step, we applied a uniform friction coefficient of 30. Here, we use the basal friction coefficient as a control so that the modelled surface velocities match the observed ones. The mismatch between observation and modelled surface velocities is quantified by the value of a cost function. The type of cost function determines to a large degree the result of the inversion process. Different cost functions are available, type md.inversion to see a list of available cost functions:

```
Available cost functions:
101: SurfaceAbsVelMisfit
102: SurfaceRelVelMisfit
103: SurfaceLogVelMisfit
104: SurfaceLogVxVyMisfit
105: SurfaceAverageVelMisfit
201: ThicknessAbsMisfit
501: DragCoefficientAbsGradient
502: RheologyBbarAbsGradient
503: ThicknessAbsGradient
```

Inverting for basal drag, we can use the cost functions that start with a 1. The cost functions can be combined and weighted individually:

```
%Cost functions
md.inversion.cost_functions=[101 103];
md.inversion.cost_functions_coefficients=ones(md.mesh.numberofvertices,2);
md.inversion.cost_functions_coefficients(:,1)=40;
md.inversion.cost_functions_coefficients(:,2)=1;
```

Our cost function is thus the sum of "SurfaceAbsVelMisfit", the absolute of the velocity misfit, and "SurfaceLogVelMisfit", the logarithm of the velocity misfit. We weigh the first cost function 40 times more than the latter one.

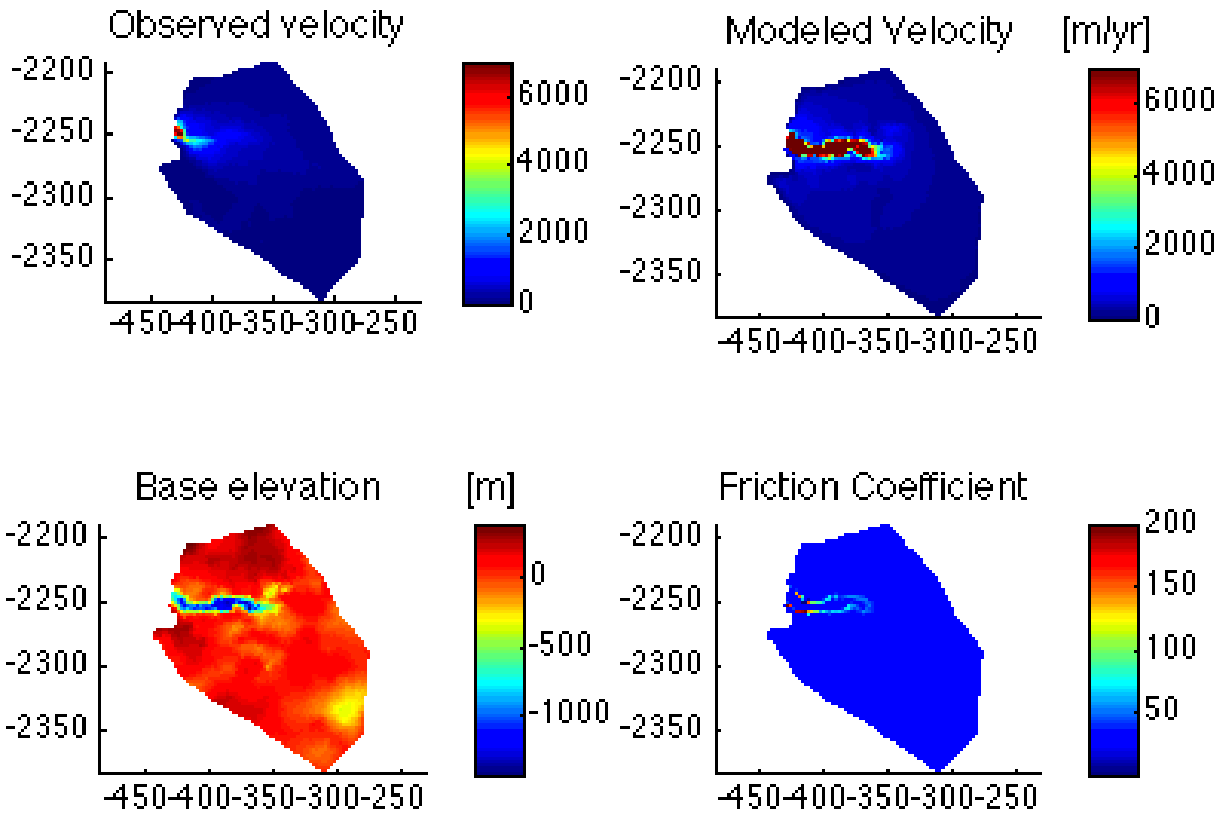
Open `runme.m` and make sure that the third step is activated: `steps = [3];`

```
>> runme
Step 3: Control method friction
  checking model consistency
  marshalling file Jakobshavn.bin
  uploading input file and queueing script
  launching solution sequence on remote cluster
  Launching solution sequence
  call computational core:
     preparing initial solution

     control method step 1/20
  ....
```

3.8.7 Step 4: Display results

Here, we display the results. Open `runme.m` and make sure that step number 4 is activated. Your results should look like this:



3.9 Modeling the Greenland ice sheet

3.9.1 Goals

- Learn how to set up a coarse continental-scale Greenland model
- Follow an example to initialize a continental domain, with a given ARGUS (`*.exp`) file and to parameterize with the SeaRISE netcdf dataset
- Become familiar with how to set up and force transient input in ISSM
- Plot results of forward simulation experiments

Go to `trunk/examples/Greenland/` to do this tutorial.

3.9.2 Introduction

In this tutorial, you will learn how to set up a continental Greenland model using the SeaRISE ice sheet model input dataset [*Nowicki et al.*, 2013]. In addition, you will gain experience in interpolation of datasets on to your continental ice sheet mesh and in setting up a transient forcing in ISSM. Finally, you will run a transient solution, resulting in a forward historical simulation of the Greenland Ice Sheet. Note that the model we set up here is coarse and is not recommended for use in a publication. A good

use for this example it is use it as a starting point to learn how to use ISSM. You may wish to improve the model provided here by increasing the resolution of the ice sheet domain outline, increasing the mesh resolution, and choosing your own/improved datasets for model parameterization.

Tutorial steps to be taken:

- Mesh Greenland with given `*.exp` file
- Adapt mesh using SeaRISE velocity data
- Parameterize (similar to the PIG model), except that all domain boundaries are on the ice front and do not have to be constrained
- Stress Balance: run inverse method to control drag
- Transient: Run a 20-year forward run
  - Use an appropriate time step for your resolution
  - Force SeaRISE surface mass balance for 10 years
  - For the next 10 years, simulate a warming scenario: decrease the surface mass balance linearly, reaching a decrease of 1.0 m/y by year 20
- Plot transient results
- Run an example exercise, forcing your Greenland model with historical SMB through time

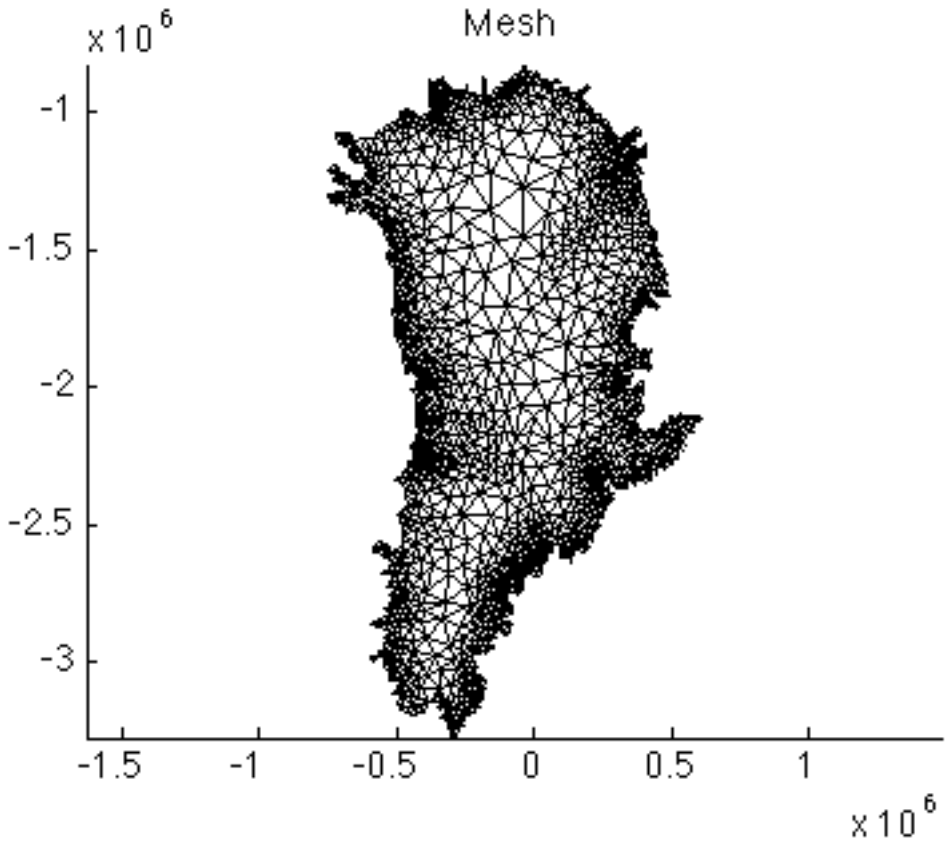
3.9.3 Mesh

In Step 1, we create a mesh using the `triangle` method (lines 10-11). This creates a new model named `md` and meshes the model domain, defined by an outline file `'DomainOutline.exp'`, at a resolution of 20,000 meters. Next, we adapt the mesh based on SeaRISE velocities, where the minimum resolution will be 5 km in locations where the velocity gradient is large and 40 km where the velocity gradient is small. The velicity data we will use resides in `'../Data/Greenland_5km_dev1.2.nc'` (line 5). Step 1 consists of the following steps:

- Fill the variable `vel` with the interpolated velocities (Hint: you need x and y velocities plus ncfile x and y coordinates)
- Mesh adapt your Greenland model (`bamg`)
  - Use variable `vel`
  - Set `hmax=400000` and `hmin=5000`
- Convert x, y coordinates to lat/long and then save your model to a file

Review the code used to create a continental Greenland mesh (lines 8-30) in the `readme.m` file. After creating the mesh and saving the model, the code uses `plotmodel` to plot a mesh visualization.

Execute step 1 in the `runme.m` file. After doing so, you should see the figure below:



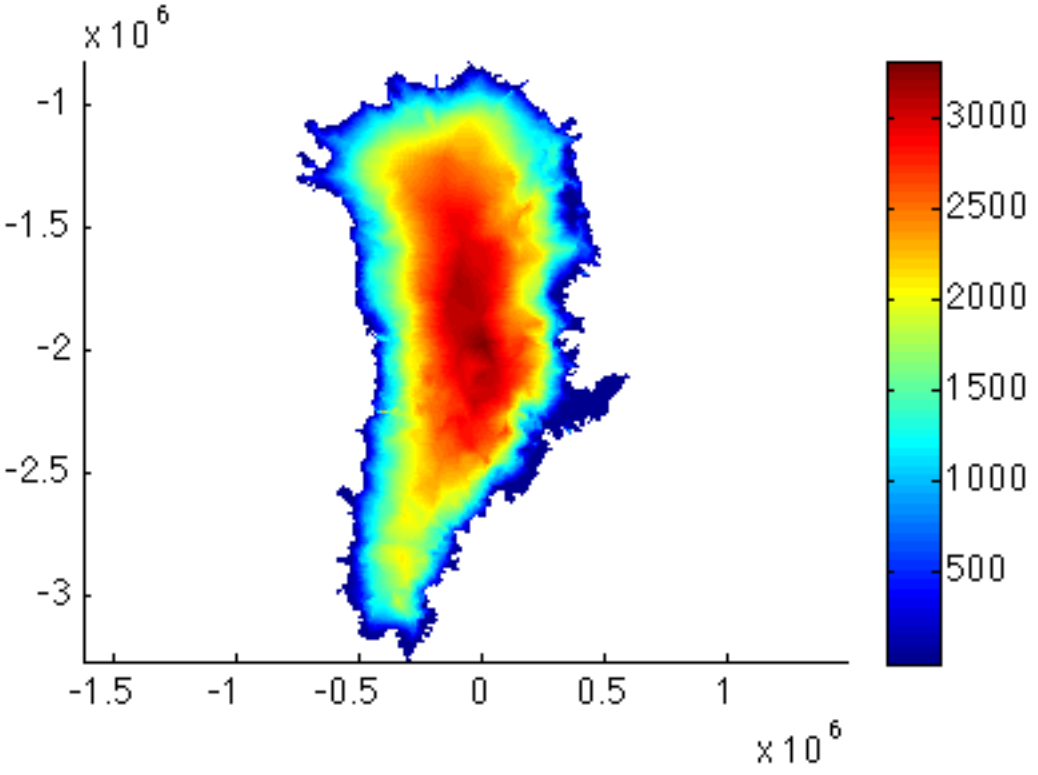
3.9.4 Parameterization

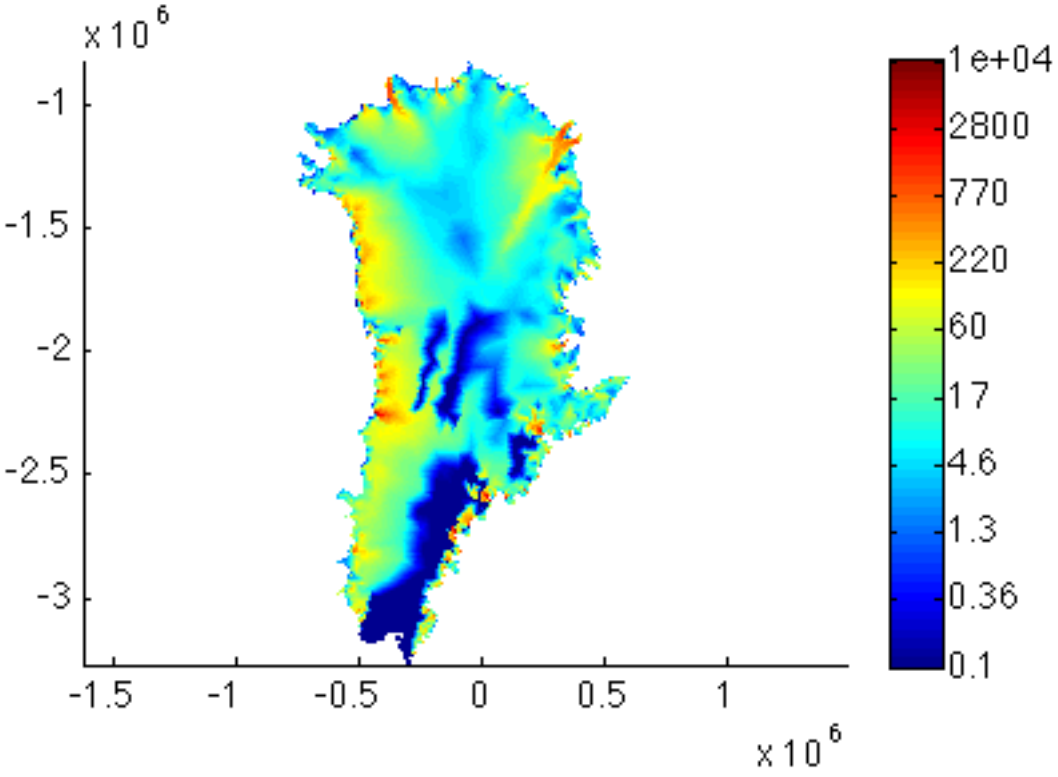
Call the `setmask` function with empty arguments, to denote that all ice is grounded. Then parameterize your mesh with file `Greenland.par`. Next, set your flow equation to SSA for all. Read through the parameter file `./Greenland.par`, which is similar to your PIG .par file, but for Greenland. Here, we are parameterizing a full continental domain, so all points along the domain boundary will be considered ice front. As a result, these boundaries do not need to be constrained, therefore the single point constraints variables will all be set to NaN.

Run step 2. This will save your parameterized model. Now, plot the new model thickness and velocity. For example:

```
>> plotmodel(md,'data',md.geometry.thickness)

>> plotmodel(md,'data',md.initialization.vel,'caxis',[1e-1 1e4],'log',10)
```


x 10^6
-1
-1.5
-2
-2.5
-3
-1.5
-1
-0.5
0
0.5
1
x 10^6
3000
2500
2000
1500
1000
500


x 10^6
-1
-1.5
-2
-2.5
-3
-1.5
-1
-0.5
0
0.5
1
x 10^6
1e+04
2800
770
220
60
17
4.6
1.3
0.36
0.1

3.9.5 Stress Balance

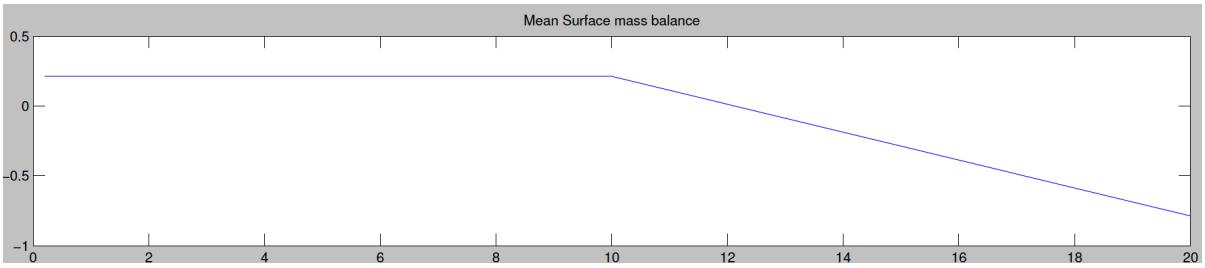
Use control methods to inversely solve for Greenland FrictionCoefficient (Step 3, lines 44-81). Note: Remember that `md.inversion` can be called for help!

- Set three different cost functions
  - Absolute value of surface velocity
  - Log of surface velocity
  - Drag coefficient gradient
- Set cost functions coefficients to 350, 0.6, and `2*10^-6`
- Set gradient scaling to 50
- Specify max inversion parameter = 200, min inversion parameter = 1
- Solve a 30-step Stress Balance model in 2D, SSA
- Copy result Friction Coefficient to model (`md`) value
- Save your model

Review step 3 in the `runme.m` to verify that the parameters have been set properly. Run step 3 in the `runme.m` to perform the steps above.

3.9.6 Transient

You are now ready to run a transient! In Step 4, we will simulate a simple constant warming trend over Greeland by forcing a temporal decrease in `md.smb.mass_balance`.



Specify a transient forcing by adding a time value to the end (in the end+1 position) of the column of forcing variable values. For example, let SMB be the initial values of surface mass balance. To impose the forcing such that before time 10, surface mass balance is set to the column vector `smb`, and after time 20, it is set to `smb-1` we use the following code:

- ```
  >> md.smb.mass_balance = [ smb smb-1]

  >> md.smb.mass_balance = [ md.smb.mass_balance; ...
   [10 20]]
  ```
  ```

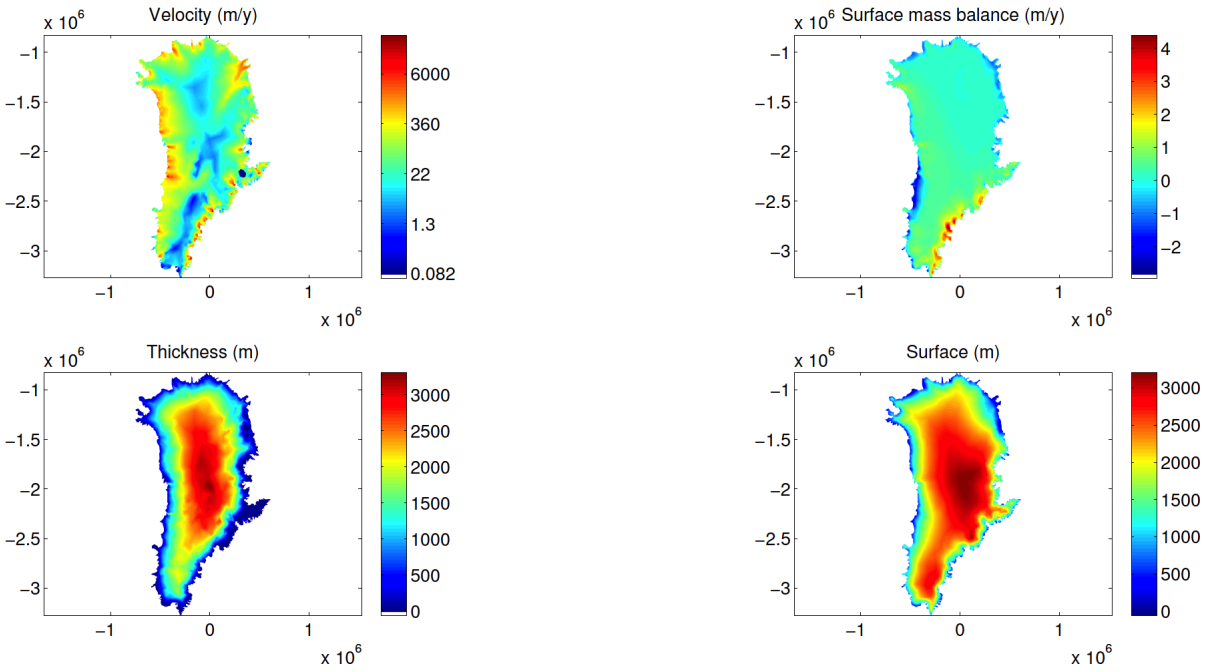
By default, ISSM will linearly interpolate surface mass balance between time 10 and time 20 in this example. Prior to first and after last imposed time, forcing values remain constant. In order to turn interpolation off (i.e. use a step function), you would set `md.timestepping.interp_forcings=0`. If this values is set to 0, then your surface mass balance will change at the specified time, and will remain constant until a new value (column vector with time in the last row) is specified.

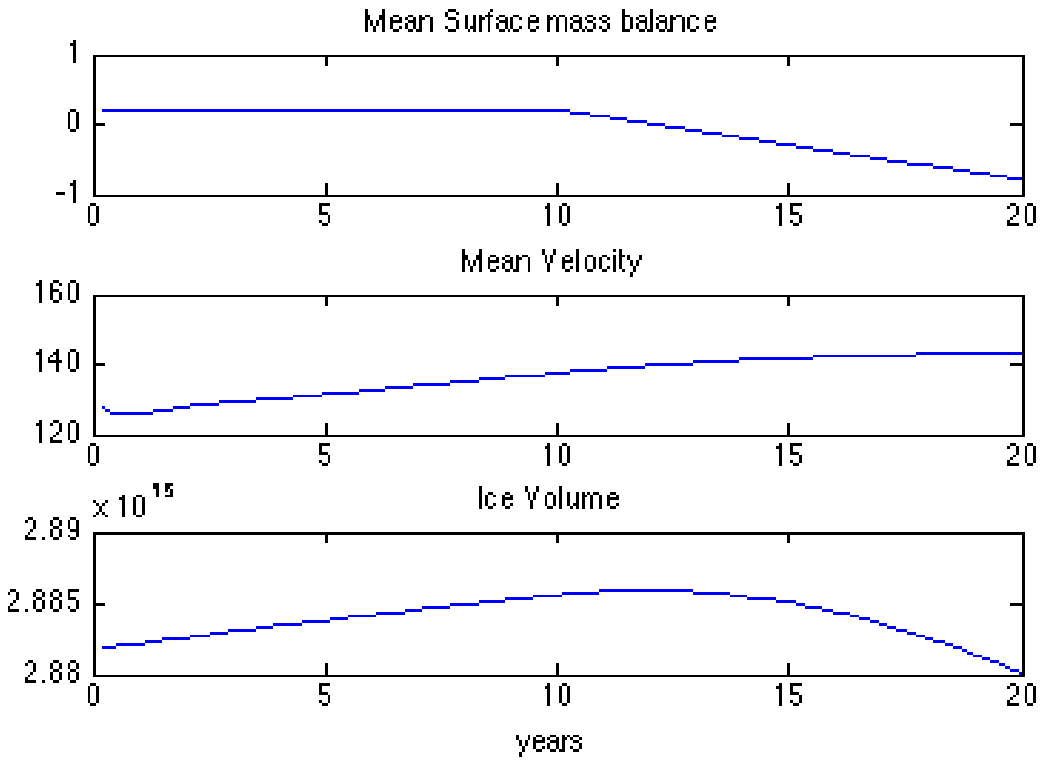
Steps to set up your transient:

- Set control `md.inversion.iscontrol` back to 0
- Interpolate surface mass balance from SeaRISE dataset, converting from water to ice equivalent
- Impose SeaRISE surface mass balance for 10 years then linearly decrease to 1 m/yr by year 20
- Set time step to 0.2 and output frequency to 1 (every time step will be output in results)
- Ask your model to save IceVolume, TotalSmb, and SmbMassBalance transient output
- Solve a 20 year Transient in 2D, SSA
- Save your model
- Review lines 83-112 in `runme.m`

In Step 5, we give you an example of how to plot the transient results (lines 114-145). To see how the transient results are stored in your model, type `md.results.TransientSolution`.

Now, run steps 4 and 5 to launch your transient and plot results.

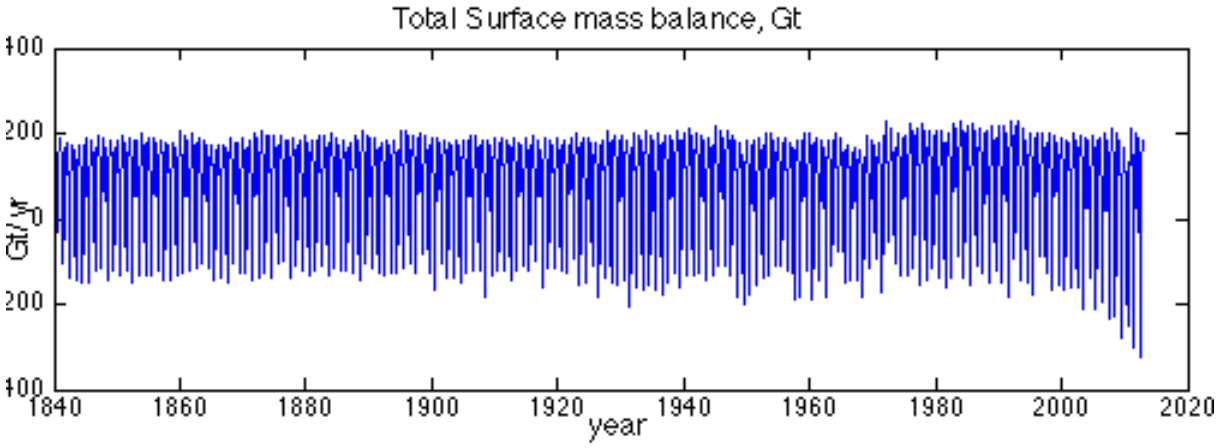




3.9.7 Exercise

Now, let's run our transient with historical mass balance! Use Jason Box's surface mass balance (SMB) time series as forcing [*Box et al.*, 2013; *Box*, 2013; *Box and Colgan*, 2013].[1]

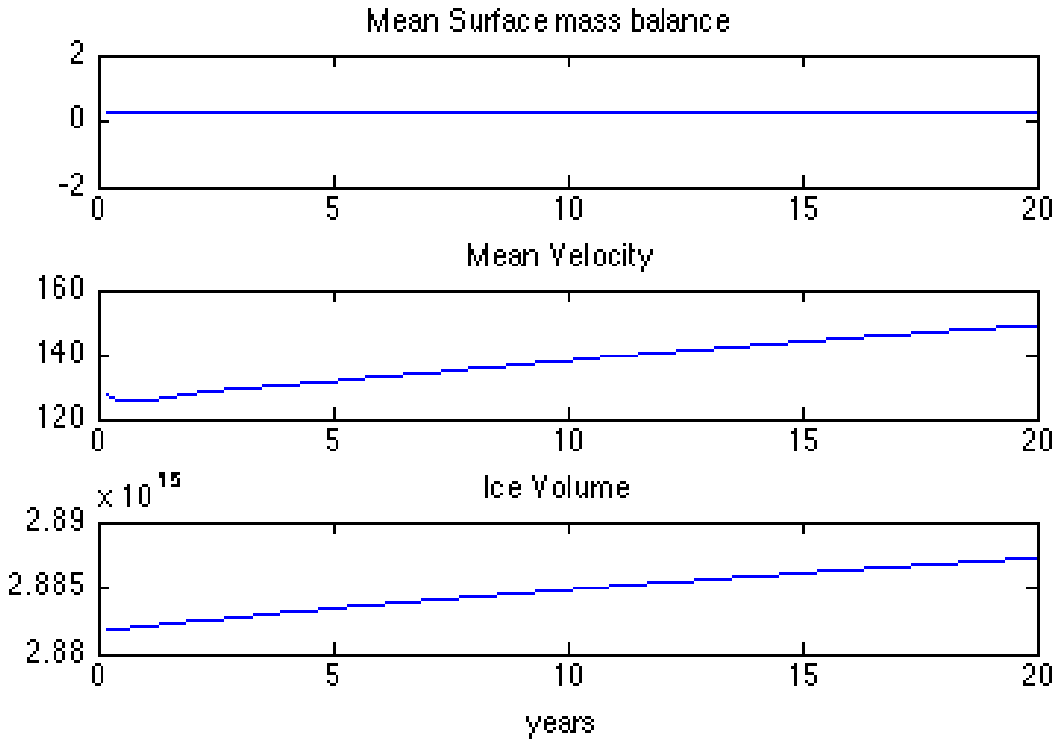
First, format the SMB provided. In Step 6 of the `runme.m` file, we extract the SMB timeseries from the netcdf file, and create a timeseries plot (lines 147-175). Execute step 6. This will result in the figure below:



In Step 7, we will relax the model towards equilibrium with the mean SMB forcing. An example of a 20 year relaxation to the time series mean is shown in `runme.m`, step 7. Run step 7, which

will assign the mean SMB to `md.smb.mass_balance` and run a transient model for 20 years, with a timestep of 0.2 years, saving the results every timestep. Step 7 will save the results in the Model `"Greenland.HistoricTransient."`

To plot the relaxed version of the model that you just created, change step 5 to load the model `"Greenland.HistoricTransient"` rather than `"Greenland.Transient,"` and run step 5 again.



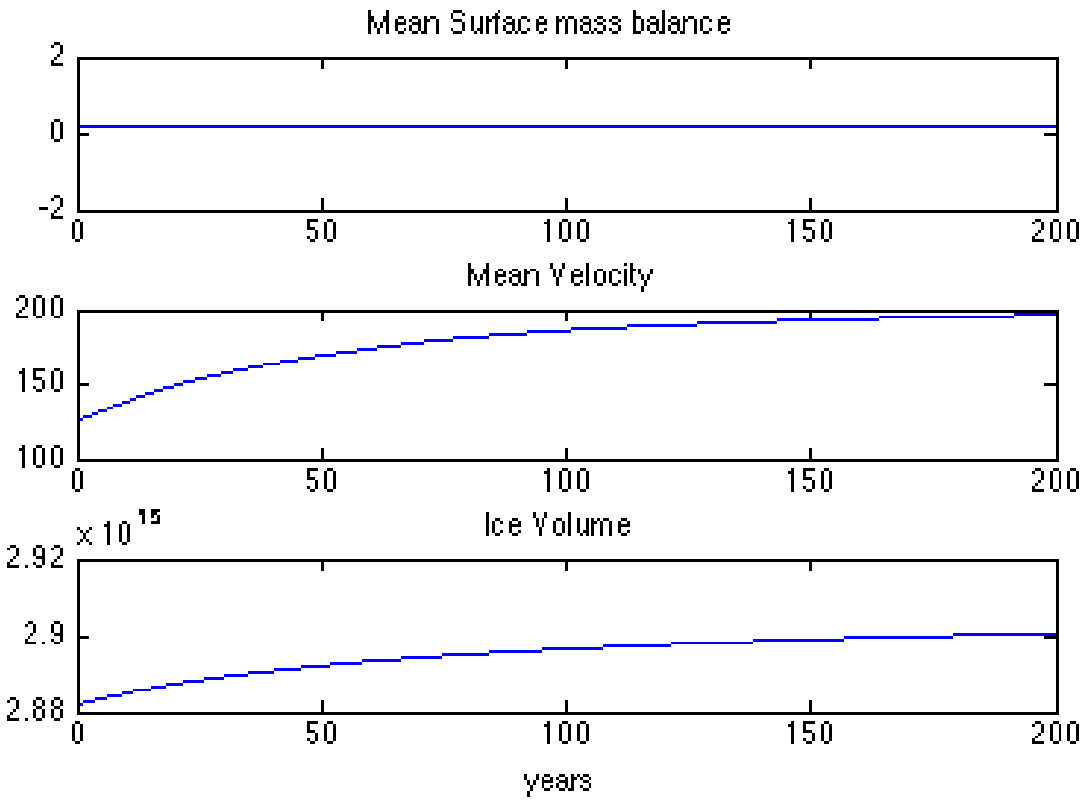
To reach equilibrium, the model should run on the order of 1000 years. Since, 1000 years might take quite a long time to run on a personal computer, you may want to try running for 200 years instead.

To accomplish this extended relaxation, alter step 7 to run for the extended time period (200 years instead of 20 years). In the last line of this step, save your model as `./Models/Greenland.HistoricTransient_200yr` instead of `./Models/Greenland.HistoricTransient`, to avoid overwriting the old model. Then, run step 7 again. This run of 200 years will take longer than your orginal 20 year run.

When you are done with step 7, complete step 8 on your own as an exercise. Fill in the required code to plot the results in step 8. Follow the comments, write the code to load the historic transient model, and create line plots of relaxation run (use Step 5 as a reference). Then, save surface mass balance by looping through 200 years (i.e. 1000 steps). Plot the surface mass balance time series in the first subplot. Title this plot "Mean Surface Mass Balance".

Next, save velocity by looping through 1000 steps. Plot velocity time series in a second subplot. Title this plot "Mean Velocity".

Lastly, save Ice Volume by looping through 1000 steps. Plot volume time series in a third subplot. Title this plot "Ice Volume" and add an x label of "years". The resulting plot should look like this:



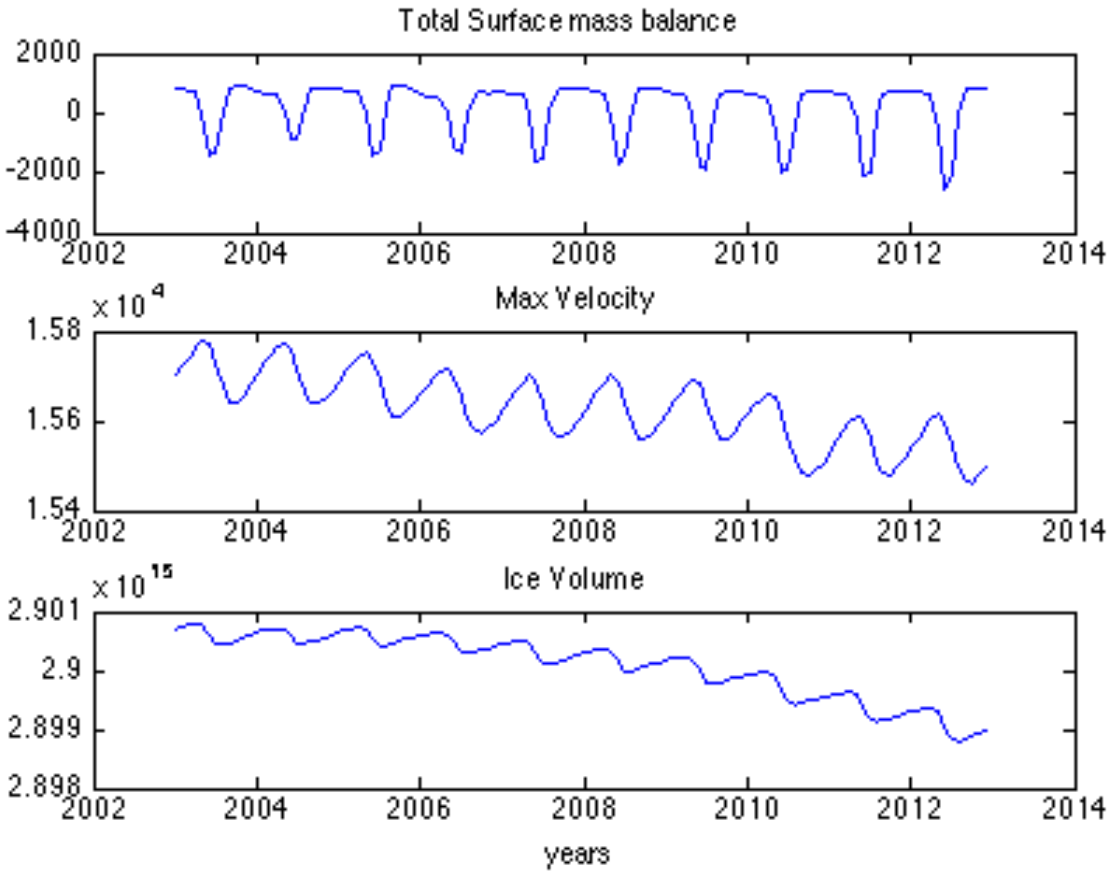
In Step 9, we will use the 200 year relaxed ice sheet as a starting condition for a historic transient run. To do so we need to save the 200 year resulting geometry and velocities into the model state. To load your past results see lines 254-259.

Next, we load the Box time series saved earlier in mat file, `smbbox.mat`, and then (lines 261-300):

- Interpolate every month of Box SMB onto the ISSM grid: insert a column for each month
- Add a final row indicating that the value should be set in the middle of each month
- Solve at a monthly time step and save monthly results

Run step 9, which will excute your historical transient forward simulation, monthly from 2003-2012.

Then, run step 10 to plot a time series of total surface mass balance, max velocity, and ice volume. See Lines 305-329.



[1] The year 1840-2012 Greenland near surface air temperature (T) and land ice SMB reconstruction after Box [2013] is calibrated to RACMO2 output [*van Meijgaard et al.*, 2008; *Ettema et al.*, 2009; *van den Broeke et al.*, 2009; *van Angelen et al.*, 2011]. The calibration for T and SMB components is based on the 53 year overlap period 1960-2012. The calibration for snow accumulation rate is shorter because ice core data availability drops after 1999. Calibration is made using linear regression coefficients for 5 km grid cells that match the average of the reconstruction to RACMO2.

The RACMO2 data are resampled and reprojected from the native 0.1 deg (∼10 km) grid to a 5 km grid better resolving areas where sharp gradients occur, especially near the ice margin where mass fluxes are largest. Several refinements are made to the Box [2013] temperature (T) and SMB reconstruction. Multiple station records now contribute to the near surface air temperature for each given year, month and grid cell in the domain while in Box [2013], data from the single highest correlating station yielded the reconstructed value. The estimation of values is made for a domain that includes land, sea, and ice. Box [2013] reconstructed T over only ice. A physically-based meltwater retention scheme [*Pfeffer et al.*, 1990, 1991] replaces the simpler approach used by Box [2013]. The RACMO2 data have a higher native resolution of 11 km as compared to the 24 km Polar MM5 data used by Box [2013] for air temperatures. The revised surface mass balance data end two years later in year 2012. The annual accumulation rates from ice cores are dispersed into a monthly temporal resolution by weighting the monthly fraction of the annual total for each grid cell in the domain evaluated using a 1960-2012 RACMO2 data.

3.9.8 Additional Exercises

- Increase SMB instead of decrease over time
- Create an instantaneous step in SMB forcing at 10 years instead of a steady change over time
- Create a more advanced SMB forcing, like cyclic steps or a curve

- Force SMB to change only in certain areas of the ice sheet
- Add more melt in the ablation zone, but more snow in the upper elevations
- Force another field transiently (e.g. friction coefficient)
- Run the Box time series yearly or for a longer subset of time. This could take a while!

3.10 Modeling the Greenland ice sheet using IceBridge data

3.10.1 Goals

- Follow an example of how to improve a coarse Greenland model by adding higher resolution Operation Icebridge (OIB) data
- Learn how to use the ISSM meshing tools to refine the Jakobshavn Isbræ (JI) basin
- Learn how to insert higher resolution bedrock and surface elevation data from the OIB campaign into the model within the JI basin

Go to `trunk/examples/IceBridge/` to do this tutorial.

3.10.2 Introduction

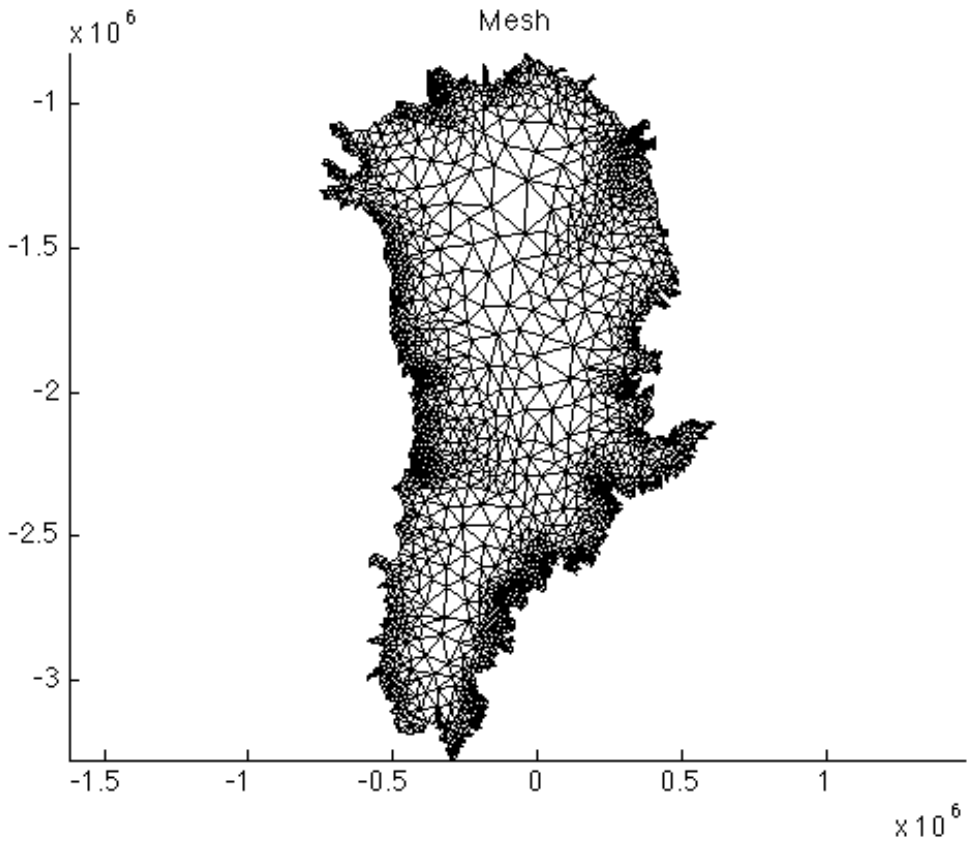
Tutorial steps to be taken:

- Refine the Greenland mesh using given JI outline.
- Parameterize the model, and include the high-resolution OIB bedrock and surface data.
- Plot the ice base and surface data.
- Stress Balance: run 2 inverse method runs to solve for control drag (20 steps recommended).
- Transient: launch 20 year runs, with coarse and refined bedrock and surface elevation data.
- Plot the transient results.

3.10.3 Mesh

We modify the experiment from the Greenland SeaRISE tutorial, and improve from there. Run the first step in `runme.m` file to mesh the Greenland domain (similar to the previous tutorial), and plot the model. Note that the code in step 1 is interrupted after making the default mesh. Plot the model:

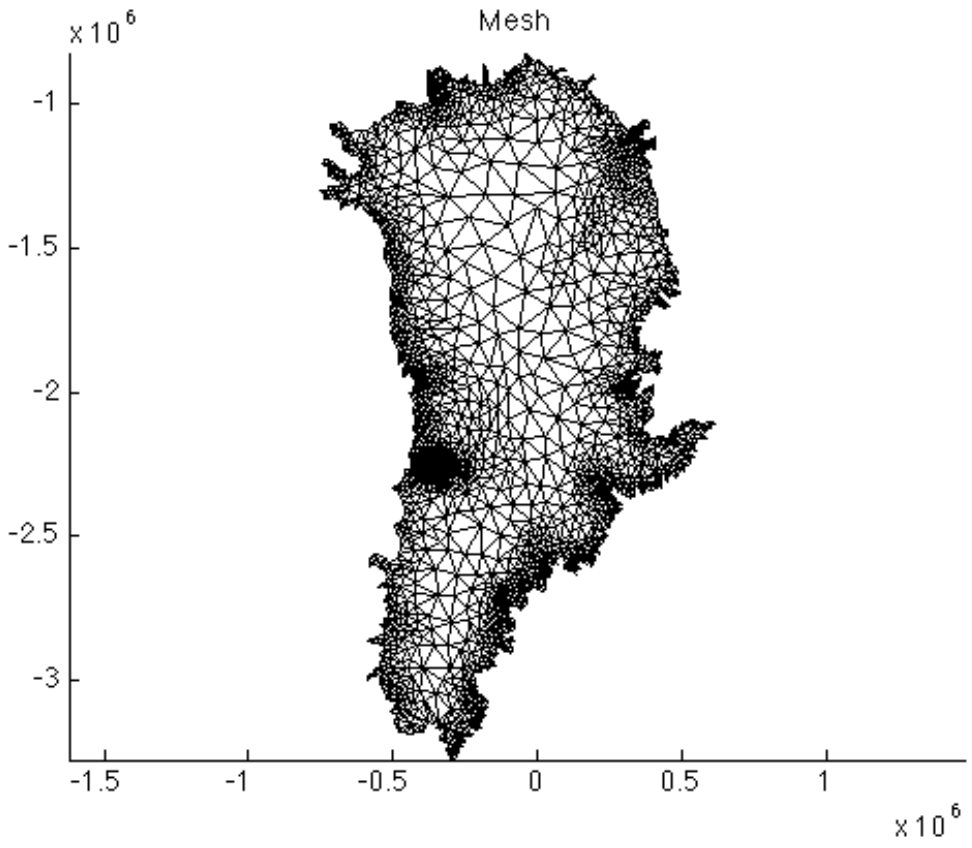
```
>> plotmodel (md,'data','mesh');
```



Now, we want to refine the mesh in JI area. An outline of this area `Jak_outline.exp` can be found in the current directory. Use the exptool command to view this outline:

```
>> exptool('Jak_outline.exp');
```

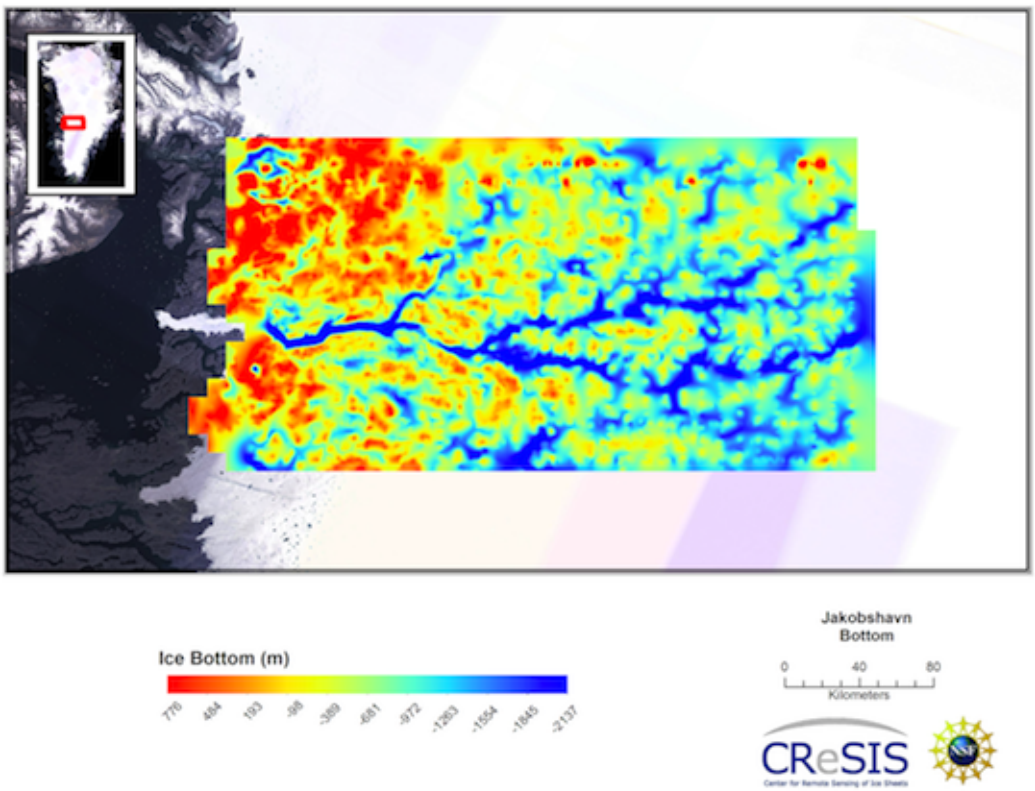
Next, we modify the `bamg` command by imposing a 3 km resolution within the JI area using `hmaxVertices`. Note that, to implement the changes noted above you must deactivate the first occurance of the `bamg` command in step 1, as well as the `return` command. Do this by commenting out these lines, and running step 1 again. Plot the results.



Use Matlab's zoom tool in the figure to make a close-up of the JI domain.

3.10.4 Parameterization

We want to include high-resolution bedrock and surface elevation data acquired in the OIB mission. The data is accessible at: `http://data.cresis.ku.edu/data/grids/Jakobshavn_2008_2011_Composite_XYZGrid.txt` Save the file in the `../Data/` directory.



To do this, the bedrock data is read, transformed into a usable grid, and interpolated to the mesh in the parameter file `Greenland.par`:

```
%Reading IceBridge data for Jakobshavn
disp('      reading IceBridge Jakobshavn bedrock');
fid  = fopen('../Data/Jakobshavn_2008_2011_Composite_XYZGrid.txt');
titles = fgets(fid);
data = fscanf(fid,'%g,%g,%g,%g,%g',[5 266400])';
fclose(fid);

[xi,yi]= ll2xy(md.mesh.lat,md.mesh.long,+1,45,70);
bed  = flipud(reshape(data(:,5),[360 740])); bed(find(bed&=& -9999))=NaN;
bedy = flipud(reshape(data(:,1),[360 740]));
bedx = flipud(reshape(data(:,2),[360 740]));

%Insert Icebridge bed and recalculate thickness
bed_jks=InterpFromGridToMesh(bedx(1,:)',bedy(:,1),bed,xi,yi,NaN);
in=ContourToMesh(md.mesh.elements,md.mesh.x,md.mesh.y,\ldots
      'Jak_grounded.exp','node',1);
bed_jks(~in)=NaN;
pos=find(~isnan(bed_jks));
md.geometry.base(pos)=bed_jks(pos);
```

Modify the `Greenland.par` file such that the surface elevation data is also included for the JI area.

Solution:

```
%Reading IceBridge data for Jakobshavn
disp('      reading IceBridge Jakobshavn bedrock');
fid  = fopen('../Data/Jakobshavn_2008_2011_Composite_XYZGrid.txt');
titles = fgets(fid);
data = fscanf(fid,'%g,%g,%g,%g,%g',[5 266400])';
fclose(fid);

[xi,yi]= ll2xy(md.mesh.lat,md.mesh.long,+1,45,70);
bed  = flipud(reshape(data(:,5),[360 740])); bed(find(bed&=& -9999))=NaN;
surf  = flipud(reshape(data(:,4),[360 740])); surf(find(surf&=& -9999))=NaN;
bedy = flipud(reshape(data(:,1),[360 740]));
bedx = flipud(reshape(data(:,2),[360 740]));

%Insert Icebridge bed and recalculate thickness
bed_jks=InterpFromGridToMesh(bedx(1,:)',bedy(:,1),bed,xi,yi,NaN);
surf_jks=InterpFromGridToMesh(bedx(1,:)',bedy(:,1),surf,xi,yi,NaN);
in=ContourToMesh(md.mesh.elements,md.mesh.x,md.mesh.y,\ldots
      'Jak_grounded.exp','node',1);
bed_jks(~in)=NaN;
surf_jks(~in)=NaN;
pos=find(~isnan(bed_jks));
md.geometry.base(pos)=bed_jks(pos);
md.geometry.surface(pos)=surf_jks(pos);
md.geometry.thickness=md.geometry.surface-md.geometry.base;
```

Next, let's plot the surface elevation, the ice thickness, and base:

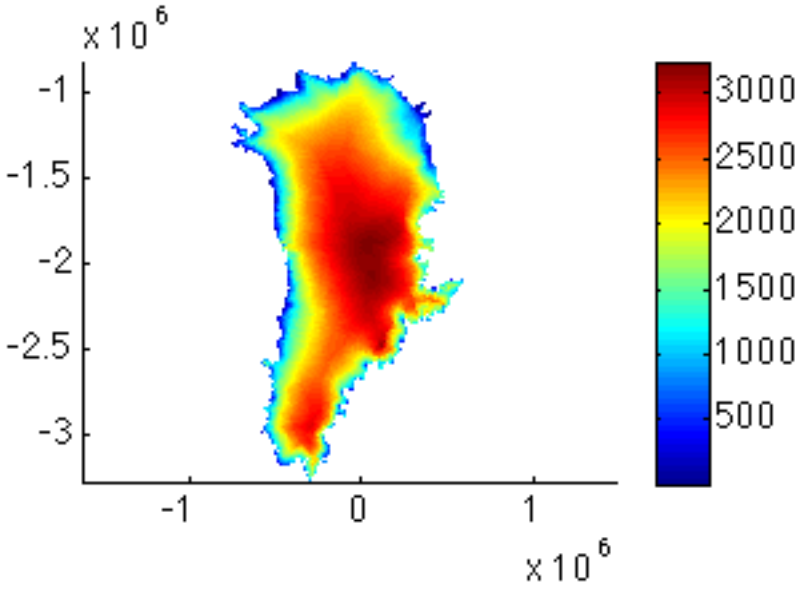


Figure 3.1: plotmodel(md,'data',md.geometry.surface)

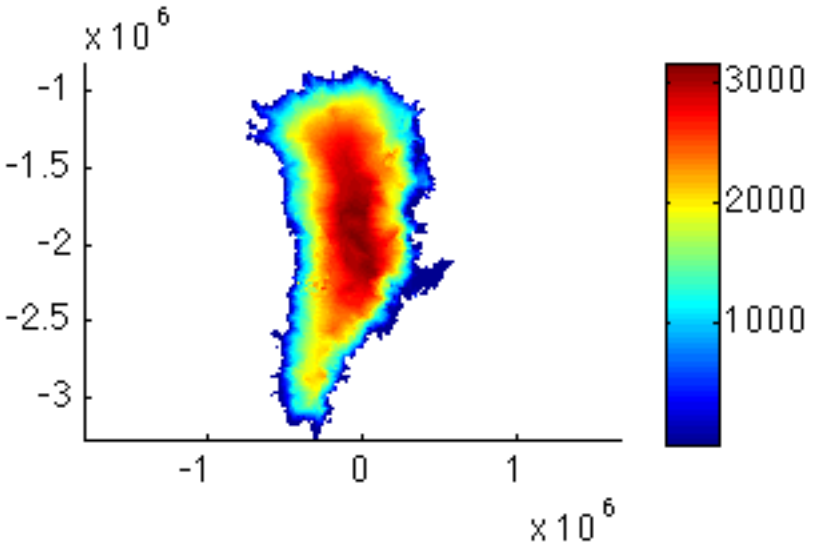


Figure 3.2: plotmodel(md,'data',md.geometry.thickness)

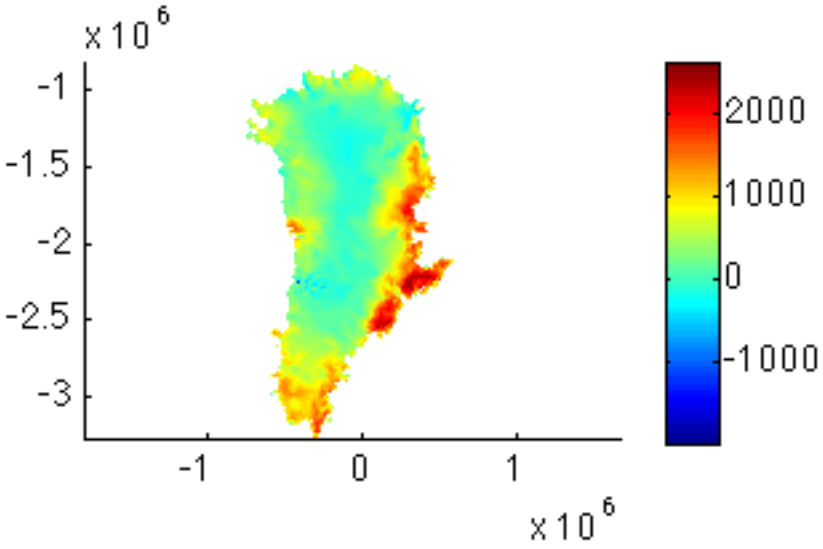


Figure 3.3: plotmodel(md,'data',md.geometry.base)

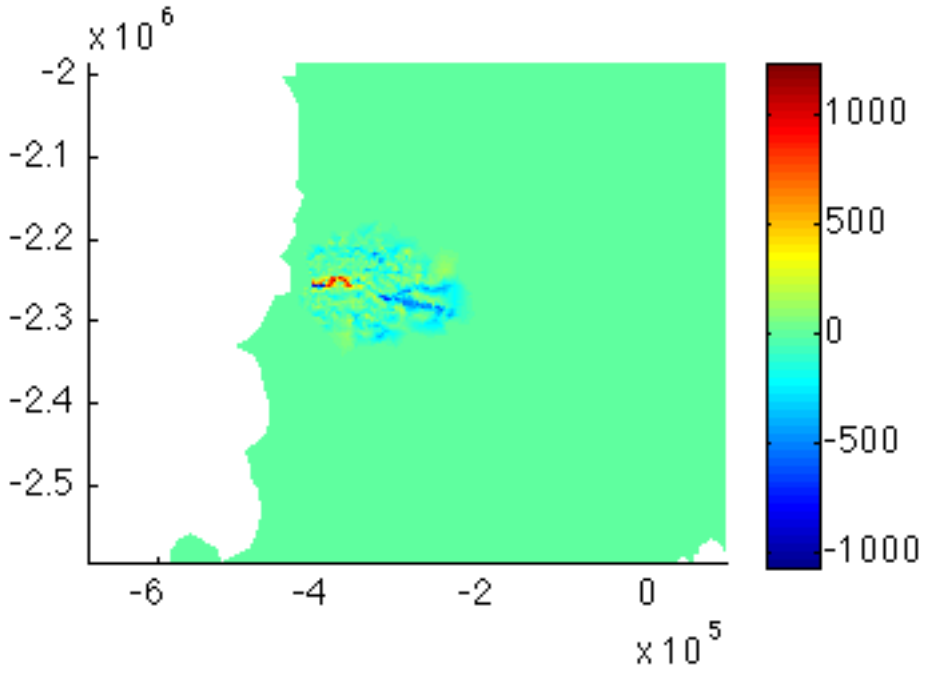
To plot the difference in the ice base topography between SeaRISE and OIB datasets do (1) modify the parameterization step in your `runme.m` file by commenting out all the above lines which insert the OIB data, and change the name the model is saved under from `Greenland.Parameterization2` to `Greenland.Parameterization` and run step 2 again. A difference in the fields can be plotted using:

```
>> md2=loadmodel('Models/Greenland.Parameterization2')

>> md=loadmodel('Models/Greenland.Parameterization')

>> plotmodel(md,'data',md2.geometry.base-md.geometry.base)
```

Zoom to the JI basin for better visibility.



3.10.5 Stress Balance

We now use inverse control methods to solve for Greenland friction coefficient. The velocity map below contains some gaps. Exclude the gaps from the inversion by creating a new `*.exp` file that outlines all the gaps in velocity data using the exptool:

```
>> exptool('data_gaps.exp')
```

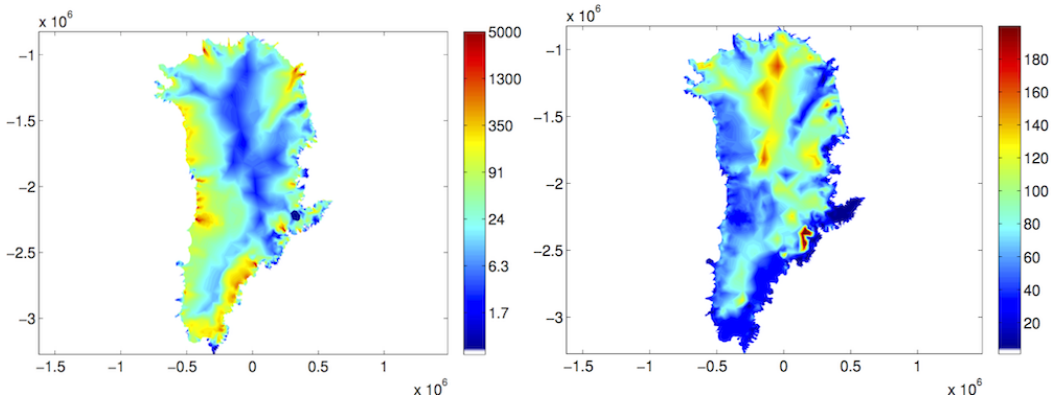
Exclude these data gaps in the inversion by giving them zero weight during the inversion process:

```
in=ContourToMesh(md.mesh.elements,md.mesh.x,md.mesh.y, 'data_gaps.exp','node',1);
md.inversion.cost_functions_coefficients(find(in),1)=0.0;
md.inversion.cost_functions_coefficients(find(in),2)=0.0;
```

Launch the stressbalance simulation, and plot velocity and basal friction coefficient. A logarithmic plot scale reveals more highlights of the velocity field structure:

```
>> plotmodel(md,'data',md.results.StressbalanceSolution.Vel,'log',10,'caxis',[0.5 5000]);
>> plotmodel(md,'data',md.results.StressbalanceSolution.FrictionCoefficient);
```

They should look like this:



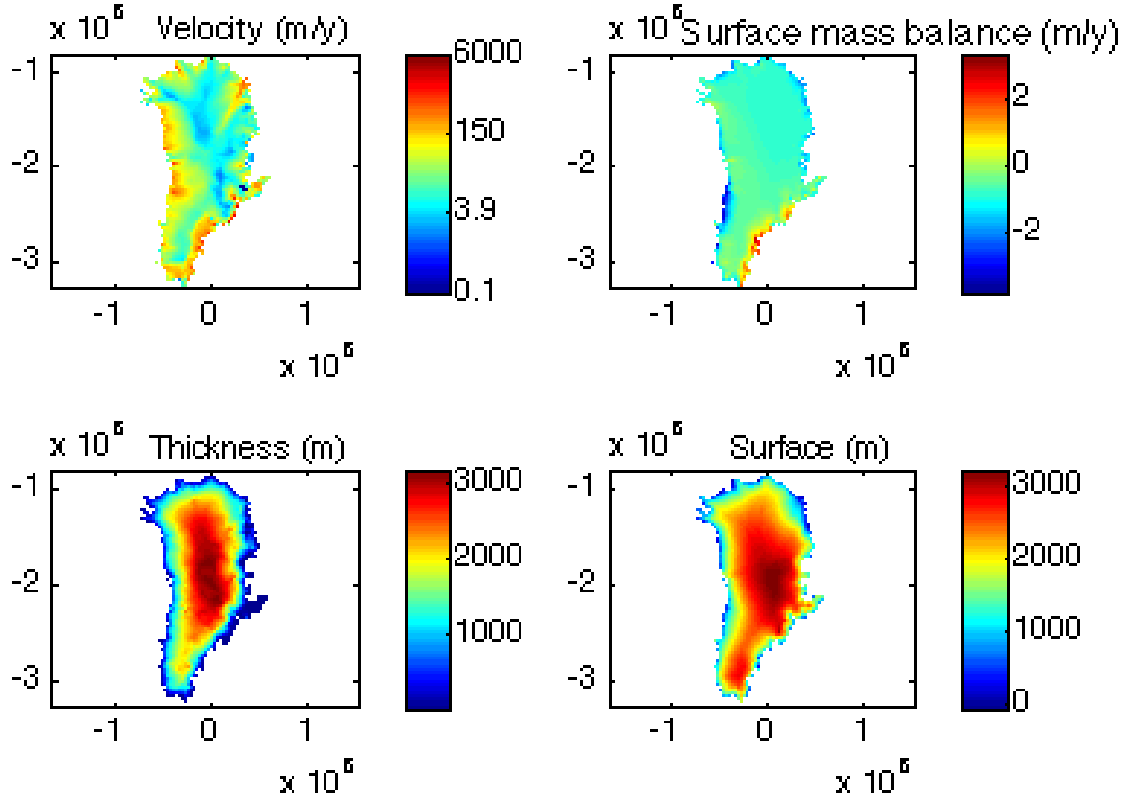
Even at this coarse resolution we can identify the high friction values inland and lower values towards the coast, which may be related to the basal thermal regime of the ice sheet.

3.10.6 Transient

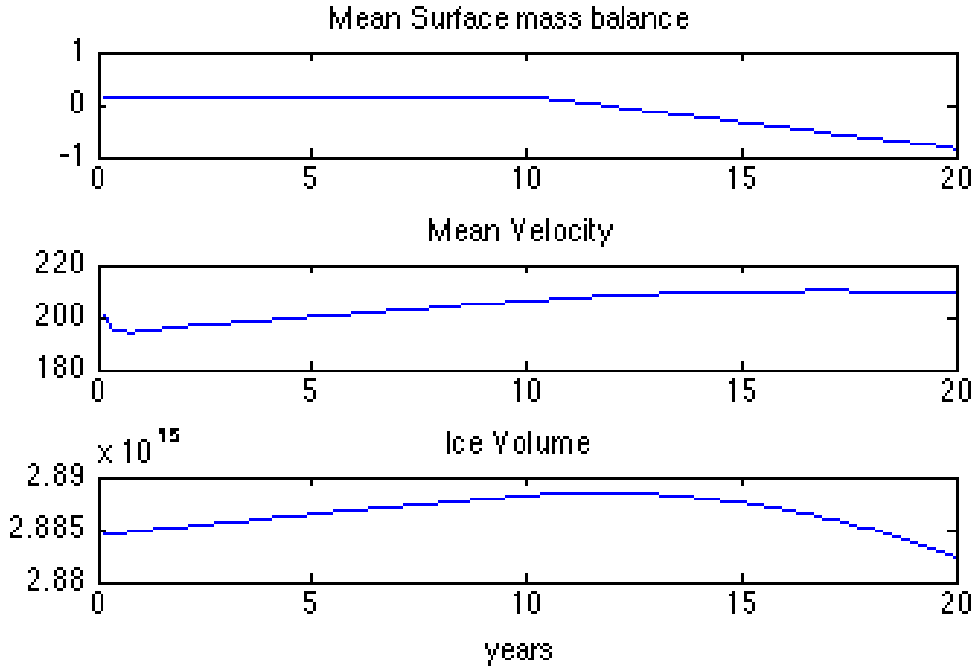
Finally, do a transient run (step 4) for 20 years, and decrease the surface mass balance linearly by 1 m w.e./yr over the last 10 years (`ncdata='../Data/Greenland_5km_dev1.2.nc';`).

```
%Set surface mass balance
x1  = ncread(ncdata,'x1');
y1  = ncread(ncdata,'y1');
smb = ncread(ncdata,'smb');
smb = InterpFromGridToMesh(x1,y1,smb',md.mesh.x,md.mesh.y,0)*1000/md.materials.rho_ice;
smb = [smb smb smb-1.0];
md.smb.mass_balance = [smb;1 10 20];
```

Your results will be located in `md.results.TransientSolution`. Plot your results using step 5. First, plot the initial plan view of velocity, surface mass balance, thickness, and surface. They should look like this:



You can plot time series of surface mass balance, mean velocity and ice volume:



3.10.7 Results

Well done! Here are some suggestions on what to explore further:

- How would you make a plot of time series of results from the SeaRISE and IceBridge experiments?
- How would you make a plot of the difference between final and initial ice thickness?

Chapter 4

Capabilities

4.1 Mesh generation

4.1.1 ARGUS file format

To mesh the domain, one needs a file containing all the coordinates of the domain outline in an ARGUS format. These files have a `*.exp` extension. Here is an example of such a file for a square glacier:

```
## Name:DomainOutline
## Icon:0
# Points Count  Value
5 1.000000
# X pos Y pos
0 0
1000000 0
1000000 1000000
0 1000000
0 0
```

The ARGUS format is used extensively by ISSM. One can use `exptool` to generate and manage ARGUS files.

4.1.2 triangle

`triangle` is a wrapper of triangle developed by Jonathan Shewchuk [*Shewchuk*, 1996]. It generates unstructured isotropic meshes:

```
>> md=triangle(md,'DomainOutline.exp',5000);
```

The first argument is the model you are working on, the second argument is the file from ARGUS containing the domain outline, and the last argument is the density of the mesh (the mean distance between two nodes). To see what the mesh looks like, one can type:

```
>> plotmodel(md,'data','mesh');
```

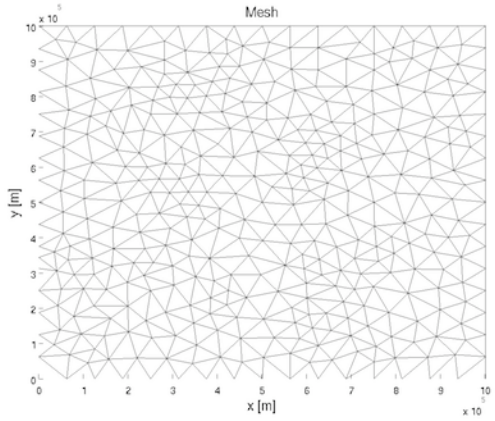


Figure 4.1: Mesh

ISSM includes a mesh adaptation capability embedded in the code, inspired by BAMG developed by Frederic Hecht [*Hecht*, 2006], and YAMS developped by Pascal Frey [*Frey*, 2001].

4.1.3 Bamg

4.1.3.1 Domain

To mesh the domain, you need a file containing all the coordinates of the domain outline in an ARGUS format. Assuming that this file is `DomainOutline.exp`

```
>> md=bamg(md,'DomainOutline.exp');
```

4.1.3.2 hmin/hmax

The minimum and maximum edge lengths can be specified by `'hmin'` and `'hmax'` options:

```
>> md=bamg(md,'DomainOutline.exp','hmax',1000);
```

4.1.3.3 hVertices

One can specified the edge length of domain outline vertices. `NaN` is used if not required.

```
>> h=[1000 100 100 100];
>> md=bamg(md,'DomainOutline.exp','hmax',1000,'hVertices',h);
```

4.1.3.4 field/err

The option `'field'` can be used with the option `'err'` to generate a mesh adapted to the field given as input for the error given as input:

```
>> md=bamg(md,'field',md.inversion.vel_obs,'err',1.5);
```

Several fields can also be used:

```
>> md=bamg(md,'field',[md.inversion.vel_obs md.geometry.thickness],'err',[1.5 20]);
```

4.1.3.5 gradation

The ratio of the lengths of two adjacent edges is controlled by the option `'gradation'`:

```
>> md=bamg(md,'field',md.inversion.vel_obs,'err',1.5,'gradation',3);
```

4.1.3.6 anisomax

The factor of anisotropy (ratio between the lengths of two edges belonging to the same triangle) can be changed by the option `'aniso'`. A factor of anisotropy equal to 1 will result in an isotropic mesh generation.

```
>> md=bamg(md,'field',md.vel_obs,'err',1.5,'anisomax',1);
```

4.1.4 Extrusion (3D)

One can extrude the mesh, in order to use a 3 dimensional model (Pattyn's higher order model and Full Stokes model). This step is not mandatory. If the user wants to keep a 2D model, skip this section. To extrude the mesh, type the following command:

```
>> md=extrude(md,8,3);
```

The first argument is the model, as usual. The second argument is the number of horizontal layers. A high number of layers gives a better precision for the simulations but creates more elements, which requires a longer computational time. Usually a number between 7 and 10 is a good balance. The third argument is called the extrusion exponent. Interesting things are usually happening near the bedrock and therefore users might want to refine the lower layers more than the upper ones. An extrusion exponent of 1 will create a mesh with layers equally distributed vertically. The higher the extrusion exponent, the more refined the base. An extrusion exponent of 3 or 4 is generally enough.

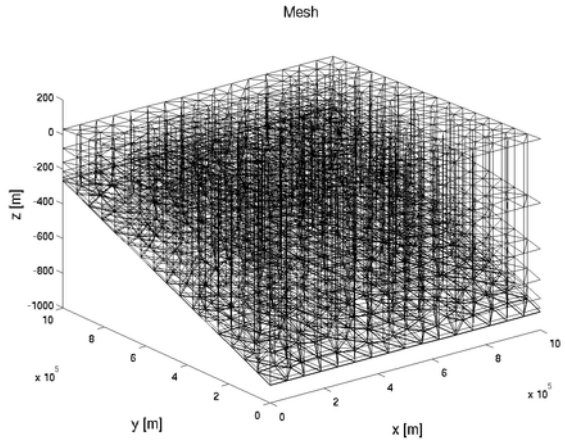


Figure 4.2: Extruded mesh

4.2 Stress balance

4.2.1 Physical basis

4.2.1.1 Conservation of linear momentum

The conservation of momentum reads:

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \tag{4.1}$$

where:

- ρ is the ice density
- $\mathbf{v}$ is the velocity vector
- $\boldsymbol{\sigma}$ is the Cauchy stress tensor
- $\mathbf{b}$ is a body force

Now if we assume that:

- The ice motion is a Stokes flow (acceleration negligible)
- The only body force is due to gravity (Coriolis effect negligible)

The equation of momentum conservation is reduced to:

$$\nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} = \mathbf{0} \tag{4.2}$$

4.2.1.2 Conservation of angular momentum

For a non-polar material body, the balance of angular momentum imposes the stress tensor to be symmetrical:

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^T \tag{4.3}$$

4.2.1.3 Ice constitutive equations

Ice is treated as a purely viscous incompressible material *Cuffey and Paterson* [2010]. Its constitutive equation therefore only involves the deviatoric stress and the strain rate tensor:

$$\boldsymbol{\sigma}' = 2\,\mu \dot{\boldsymbol{\varepsilon}} \tag{4.4}$$

where:

- $\boldsymbol{\sigma}'$ is the deviatoric stress tensor ($\boldsymbol{\sigma}' = \boldsymbol{\sigma} + p\mathbf{I}$)
- μ is the ice effective viscosity
- $\dot{\boldsymbol{\varepsilon}}$ is the strain rate tensor

Ice is a non-Newtonian fluid, its viscosity follows the generalized Glen's flow law *Glen* [1955]:

$$\mu = \frac{B}{2\,\dot{\varepsilon}_e^{\frac{n-1}{n}}} \tag{4.5}$$

where:

- B is the ice hardness or rigidity
- n is Glen's flow law exponent, generally taken as equal to 3
- $\dot{\varepsilon}_e$ is the effective strain rate

The effective strain rate is defined as:

$$\dot{\varepsilon}_e = \sqrt{\frac{1}{2}\sum_{i,j} \dot{\varepsilon}_{ij}^2} = \frac{1}{\sqrt{2}} \|\dot{\boldsymbol{\varepsilon}}\|_F \tag{4.6}$$

where $\|\cdot\|_F$ is the Frobenius norm.

4.2.1.4 Full-Stokes (FS) field equations

Without any further approximation, the previous system of equations are called the *Full-Stokes* model.

4.2.1.5 Higher-Order (HO) field equations

We make two assumptions:

1. Bridging effects are neglected
2. Horizontal gradient of vertical velocities are neglected compared to vertical gradients of horizontal velocities

With these two assumptions, the Full-Stokes equations are reduced to a system of 2 equations with 2 unknowns *Blatter* [1995]; *Pattyn* [2003]:

$$\begin{aligned} \nabla \cdot (2\mu \dot{\boldsymbol{\varepsilon}}_{HO1}) &= \rho g \frac{\partial s}{\partial x} \\ \nabla \cdot (2\mu \dot{\boldsymbol{\varepsilon}}_{HO2}) &= \rho g \frac{\partial s}{\partial y} \end{aligned} \tag{4.7}$$

With

$$\dot{\boldsymbol{\varepsilon}}_{HO1} = \begin{bmatrix} 2\dfrac{\partial v_x}{\partial x} + \dfrac{\partial v_y}{\partial y} \\ \dfrac{1}{2}\left(\dfrac{\partial v_x}{\partial y} + \dfrac{\partial v_y}{\partial x}\right) \\ \dfrac{1}{2}\dfrac{\partial v_x}{\partial z} \end{bmatrix} \quad \dot{\boldsymbol{\varepsilon}}_{HO2} = \begin{bmatrix} \dfrac{1}{2}\left(\dfrac{\partial v_x}{\partial y} + \dfrac{\partial v_y}{\partial x}\right) \\ \dfrac{\partial v_x}{\partial x} + 2\dfrac{\partial v_y}{\partial y} \\ \dfrac{1}{2}\dfrac{\partial v_y}{\partial z} \end{bmatrix} \tag{4.8}$$

4.2.1.6 Shelfy-Stream Approximation (SSA) field equations

We make the following assumption:

1. Vertical shear is negligible

With this assumption, we have a system of 2 equations with 2 unknowns in the horizontal plane [*Morland*, 1987; *MacAyeal*, 1989]:

$$\begin{aligned} \nabla \cdot (2\bar{\mu} H \dot{\boldsymbol{\varepsilon}}_{SSA1}) - \alpha^2 v_x &= \rho g H \frac{\partial s}{\partial x} \\ \nabla \cdot (2\bar{\mu} H \dot{\boldsymbol{\varepsilon}}_{SSA2}) - \alpha^2 v_y &= \rho g H \frac{\partial s}{\partial y} \end{aligned} \tag{4.9}$$

With

$$\dot{\boldsymbol{\varepsilon}}_{SSA1} = \begin{bmatrix} 2\dfrac{\partial v_x}{\partial x} + \dfrac{\partial v_y}{\partial y} \\ \dfrac{1}{2}\left(\dfrac{\partial v_x}{\partial y} + \dfrac{\partial v_y}{\partial x}\right) \end{bmatrix} \quad \dot{\boldsymbol{\varepsilon}}_{SSA2} = \begin{bmatrix} \dfrac{1}{2}\left(\dfrac{\partial v_x}{\partial y} + \dfrac{\partial v_y}{\partial x}\right) \\ \dfrac{\partial v_x}{\partial x} + 2\dfrac{\partial v_y}{\partial y} \end{bmatrix} \tag{4.10}$$

where:

- $\bar{\mu}$ is the depth-averaged viscosity
- H is the ice thickness
- α is the basal friction coefficient

4.2.1.7 Boundary conditions

At the surface of the ice sheet, Γ_s, we assume a stress-free boundary condition. A viscous friction law is applied at the base of the ice sheet, Γ_b, and water pressure is applied at the ice/water interface Γ_w. For FS, these boundary conditions are:

$$\begin{aligned} \boldsymbol{\sigma}\cdot\mathbf{n} &= \mathbf{0} && \text{on } \Gamma_s \\ \left(\boldsymbol{\sigma}\cdot\mathbf{n}\cdot\mathbf{n}+\alpha^2\mathbf{v}\right)_{\parallel} &= \mathbf{0} && \text{on } \Gamma_b \\ \mathbf{v}\cdot\mathbf{n} &= 0 && \text{on } \Gamma_b \\ \boldsymbol{\sigma}\cdot\mathbf{n} &= \rho_w g z\mathbf{n} && \text{on } \Gamma_w \end{aligned} \tag{4.11}$$

where

- $\mathbf{n}$ is the outward-pointing unit normal vector
- ρ_w is the water density
- z is the vertical coordinate with respect to sea level

For HO, these boundary conditions become:

$$\begin{aligned} \dot{\boldsymbol{\varepsilon}}_{HO1}\cdot\mathbf{n} &= 0 & \dot{\boldsymbol{\varepsilon}}_{HO2}\cdot\mathbf{n} &= 0 && \text{on } \Gamma_s \\ 2\mu\,\dot{\boldsymbol{\varepsilon}}_{HO1}\cdot\mathbf{n} &= -\alpha^2 v_x & 2\mu\,\dot{\boldsymbol{\varepsilon}}_{HO2}\cdot\mathbf{n} &= -\alpha^2 v_y && \text{on } \Gamma_b \\ 2\mu\,\dot{\boldsymbol{\varepsilon}}_{HO1}\cdot\mathbf{n} &= f_w n_x & 2\mu\,\dot{\boldsymbol{\varepsilon}}_{HO2}\cdot\mathbf{n} &= f_w n_y && \text{on } \Gamma_w \end{aligned} \tag{4.12}$$

where $f_w = \rho g\,(s-z) + \rho_w g \min(z,0)$.

For SSA, these boundary conditions are:

$$\dot{\boldsymbol{\varepsilon}}_{SSA1}\cdot\mathbf{n} = 0 \quad \dot{\boldsymbol{\varepsilon}}_{SSA2}\cdot\mathbf{n} = 0 \quad \text{on } \Gamma_s \tag{4.13}$$

$$\begin{aligned} 2\mu\,\dot{\boldsymbol{\varepsilon}}_{SSA1}\cdot\mathbf{n} &= \left(\tfrac{1}{2}\rho g H^2 - \tfrac{1}{2}\rho_w g b^2\right) n_x \\ 2\mu\,\dot{\boldsymbol{\varepsilon}}_{SSA2}\cdot\mathbf{n} &= \left(\tfrac{1}{2}\rho g H^2 - \tfrac{1}{2}\rho_w g b^2\right) n_y \end{aligned} \quad \text{on } \Gamma_w \tag{4.14}$$

4.2.2 Model parameters

The parameters relevant to the stress balance solution can be displayed by typing:

```
>> md.stressbalance
```

- `md.stressbalance.restol`: mechanical equilibrium residue convergence criterion
- `md.stressbalance.reltol`: velocity relative convergence criterion, (`NaN` if not applied)
- `md.stressbalance.abstol`: velocity absolute convergence criterion, (`NaN` if not applied)
- `md.stressbalance.maxiter`: maximum number of nonlinear iterations (default is 100)
- `md.stressbalance.viscosity_overshoot`: over-shooting constant defined as:

$$\mu^{\text{new}} = \mu^{\text{new}} + \alpha\left(\mu^{\text{new}} - \mu^{\text{old}}\right) \tag{4.15}$$

- `md.stressbalance.spcvx`: x-axis velocity constraint (`NaN` means no constraint)
- `md.stressbalance.spcvy`: y-axis velocity constraint (`NaN` means no constraint)
- `md.stressbalance.spcvz`: z-axis velocity constraint (`NaN` means no constraint)
- `md.stressbalance.rift_penalty_threshold`: threshold for instability of mechanical constraints
- `md.stressbalance.rift_penalty_lock`: number of iterations before rift penalties are locked

- `md.stressbalance.penalty_factor`: offset used by penalties:

$$\kappa = 10^{\text{penalty\_factor}} \max_{i,j} |K_{ij}| \tag{4.16}$$

- `md.stressbalance.vertex_pairing`: pairs of vertices that are penalized
- `md.stressbalance.shelf_dampening`: use dampening for floating ice ? Only for Stokes model
- `md.stressbalance.referential`: local referential
- `md.stressbalance.requested_outputs`: additional outputs requested

The solution will also use the following model fields:

- `md.flowequations`: FS, HO or SSA
- `md.materials`: material parameters
- `md.initialization.vx`: x component of velocity (used as an initial guess)
- `md.initialization.vy`: y component of velocity (used as an initial guess)
- `md.initialization.vz`: y component of velocity (used as an initial guess)

4.2.3 Running a simulation

To run a simulation, use the following command:

```
>> md=solve(md,StressbalanceSolutionEnum);
```

The first argument is the model, the second is the nature of the simulation one wants to run.

4.3 Mass transport / Free surface

4.3.1 Physical basis

4.3.1.1 Conservation of mass

The mass transport equation is derived from the depth-integrated form of the mass conservation equation and reads:

$$\frac{\partial H}{\partial t} = -\nabla \cdot (H\bar{\mathbf{v}}) + \dot{M}_s - \dot{M}_b \tag{4.17}$$

where

- $\bar{\mathbf{v}}$ is the depth-averaged velocity vector
- H is the ice thickness
- $\dot{M}_s$ is the surface accumulation (in m/yr of ice equivalent, positive for accumulation)
- $\dot{M}_b$ is the basal melting (in m/yr of ice equivalent, positive for melting)

For full-Stokes models, free surface equations are solved for the upper surface and the base of floating ice:

$$\frac{\partial s}{\partial t} + v_x(s)\frac{\partial s}{\partial x} + v_y(s)\frac{\partial s}{\partial y} - v_z(s) = \dot{M}_s \tag{4.18}$$

and

$$\frac{\partial b}{\partial t} + v_x(b)\frac{\partial b}{\partial x} + v_y(b)\frac{\partial b}{\partial y} - v_z(b) = \dot{M}_b \tag{4.19}$$

where

- s is the elevation of the ice upper surface
- b is the elevation of the floating ice lower surface
- $(v_x(s), v_y(s), v_z(s))$ are the ice velocity components at the upper surface s
- $(v_x(b), v_y(b), v_z(b))$ are the ice velocity components at the base b

4.3.1.2 Boundary conditions

Ice thickness is imposed at the inflow boundary:

$$H = H_{obs} \text{ on } \Gamma_- \tag{4.20}$$

For free surfaces models, both b and s are constrained at the inflow boundary.

4.3.1.3 Numerical implementation

Mass transport is solved using finite elements in space, and implicit finite difference in time. To stabilize the equation, artificial diffusion might be added to the left hand side:

$$\frac{\partial H}{\partial t} + \nabla \cdot (H\bar{\mathbf{v}}) + \nabla (\mathfrak{D}\nabla H) = \dot{M}_s - \dot{M}_b \tag{4.21}$$

where $\mathfrak{D}$ is the artificial diffusivity. We take

$$\mathfrak{D} = \frac{h}{2} \begin{pmatrix} |vx| & 0 \\ 0 & |vy| \end{pmatrix} \tag{4.22}$$

4.3.2 Model parameters

The parameters relevant to the mass transport solution can be displayed by typing:

```
>> md.masstransport
```

- `md.masstransport.spcthickness`: thickness constraints (`NaN` means no constraint)
- `md.masstransport.hydrostatic_adjustment`: adjustment of ice shelves upper and lower surfaces: `'Incremental'` or `'Absolute'`
- `md.masstransport.stabilization`: 0: no stabilization, 1: Artificial diffusivity 3: Discontinuous Galerkin (experimental)
- `md.masstransport.penalty_factor`: offset used by penalties

$$\kappa = 10^{\text{penalty\_offset}} \max_{i,j} |K_{ij}| \tag{4.23}$$

- `md.masstransport.vertex_pairing`: pairs of vertices that are penalized (for periodic boundary conditions only)

The solution will also use the following model fields:

- `md.surfaceforcings.ablation_rate`: surface ablation rate (in meters)
- `md.surfaceforcings.mass_balance`: surface mass balance (in meters)
- `md.initialization.vx`: x component of velocity
- `md.initialization.vy`: y component of velocity
- `md.basalforcings.groundedice_melting_rate`: basal melting rate applied on grounded ice (positive if melting)
- `md.basalforcings.floatingice_melting_rate`: basal melting rate applied on floating ice (positive if melting)
- `md.surfaceforcings.mass_balance`: surface mass balance (in meters/year ice equivalent)
- `md.timestepping.time_step`: length of time steps (in years)

4.3.3 Running a simulation

To run a simulation, use the following command:

```
>> md=solve(md,MasstransportSolutionEnum);
```

The first argument is the model, the second is the nature of the simulation one wants to run. This will compute one time step of the mass transport equation; use the transient solution for multiple time steps.

4.4 Thermal Model

4.4.1 Physical basis

4.4.1.1 Thermal state

The heat transport equation is derived from the balance equation of internal energy E combined with Fourier's law of heat transfer and reads:

$$\rho\left(\frac{\partial E}{\partial t}+\mathbf{v}\cdot\nabla E\right)=-\nabla\left(\kappa(E)\nabla E\right)+\mathrm{Tr}\left(\boldsymbol{\sigma}\cdot\dot{\boldsymbol{\varepsilon}}\right) \tag{4.24}$$

where radiative sources have been neglected, and

- $\mathbf{v}$ is the velocity vector
- $\dot{\boldsymbol{\varepsilon}}$ is the strain rate tensor
- E is the internal energy density
- κ is the specific heat conductivity, which can depend on the heat density
- $\boldsymbol{\sigma}$ is the Cauchy stress tensor.

For constant heat conductivity and heat capacity c_i, the previous equation reduces to

$$\rho c_i\left(\frac{\partial T}{\partial t}+\mathbf{v}\cdot\nabla T\right)=-c_i\kappa\Delta T+\mathrm{Tr}\left(\boldsymbol{\sigma}\cdot\dot{\boldsymbol{\varepsilon}}\right) \tag{4.25}$$

4.4.1.2 Boundary conditions

Dirichlet boundary conditions can be applied at the ice surface:

$$T(z=s)=T_s, \tag{4.26}$$

and Neumann boundary conditions at the ice base:

$$q(z=b)=-\kappa(E)\nabla E=q_{\mathrm{geo}} \tag{4.27}$$

where

- s is the elevation of the ice upper surface
- b is the elevation of the floating ice lower surface

When using the enthalpy formulation, the basal boundary condition scheme from *Aschwanden et al.* [2012], figure 5 is used instead of the previous equation.

4.4.1.3 Numerical implementation

The heat equation is solved using linear finite elements in space, and semi-implicit finite difference in time (time stepping should therefore satisfy the CFL condition). To stabilize the equation, we either add an isotropic artificial diffusion to the left hand side:

$$\rho\left(\frac{\partial E}{\partial t}+\mathbf{v}\cdot\nabla E\right)+\nabla\left(\kappa(E)\nabla E\right)+\nabla\left(\mathfrak{D}\nabla E\right)=\mathrm{Tr}\left(\boldsymbol{\sigma}\cdot\dot{\boldsymbol{\varepsilon}}\right) \tag{4.28}$$

where $\mathfrak{D}$ is the artificial diffusivity. We take

$$\mathfrak{D}=\frac{h}{2}\begin{pmatrix}|vx| & 0 & 0\\ 0 & |vy| & 0\\ 0 & 0 & |vz|\end{pmatrix} \tag{4.29}$$

or rely on the Streamline upwind/Petrov-Galerkin formulation (SUPG) from *Franca et al.* [2006].

4.4.2 Model parameters

The parameters relevant to the heat equation solution can be displayed by typing:

```
>> md.thermal
```

- `md.thermal.spctemperature`: temperature constraints (`NaN` means no constraint)
- `md.thermal.stabilization`: type of stabilization (0: no stabilization; 1: artificial diffusion; 2: Streamline-Upwind Petrov-Galerkin)
- `md.thermal.maxiter`: maximum number of iterations for thermal solver
- `md.thermal.penalty_lock`: stabilize unstable thermal constraints that keep zigzagging after n iteration (default is 0, no stabilization)
- `md.thermal.penalty_threshold`: threshold to declare convergence of thermal solution (default is 0)
- `md.thermal.penalty_factor`: offset used by penalties(default is 3):

$$\kappa=10^{\mathrm{penalty\_factor}}\max_{i,j}|K_{ij}| \tag{4.30}$$

- `md.thermal.isenthalpy`: are we using the enthalpy formulation (Aschwanden et al., 2012)? (0: no, 1: yes)
- `md.thermal.isdynamicbasalspc`: are we allowing changing basal boundary conditions for transient runs?
- `md.thermal.requested_outputs`: specify further requested outputs here.

The solution will also use the following model fields:

- `md.initialization.temperature`: temperature field (in K)
- `md.initialization.waterfraction`: water fraction in ice (between 0 and 1)
- `md.basalforcings.geothermalflux`: geothermal heat flux (in W/m^2)
- `md.basalforcings.meltingrate`: basal melting rate (in m/yr w.e.)
- `md.timestepping.time_step`: length of time steps (in yrs)

4.4.3 Running a simulation

To run a simulation solving only the thermal state, use the following command:

```
>> md=solve(md,ThermalSolutionEnum);
```

This will compute one time step of the thermal equation; use the transient solutionfor multiple time steps. To run a simulation solving both the thermal and stressbalance state in a coupled fashion, use the following command:

```
>> md=solve(md,SteadystateSolutionEnum);
```

The first argument is the model, the second is the nature of the simulation one wants to run.

4.5 Dual continuum Hydrology model

4.5.1 Physical basis

Using the dual continuum porous equivalent approach, the inefficient and efficient drainage components are both modeled as sediment layers with the use of a specific activation scheme for the efficient drainage system. This approach defines in a continuous manner the location where the efficient drainage system is most likely to develop.

4.5.1.1 Water Distribution

The model consist of two analyses, one for the Inefficient Drainage System (IDS) and the other for the Efficient Drainage System(EDS). Each compute the water head by using a vertically integrated diffusion equation based on Darcy's law. The two are coupled through a transfer term, which is implicitly computed at the same time as the water head. In the following equation, the index j (subscript or superscript) may either refer to the IDS ($j = \mathrm{i}$) or to the EDS ($j = \mathrm{e}$).

$$S_j \frac{\partial h_j}{\partial t} - \nabla \cdot (T_j \nabla h_j) = Q_j. \tag{4.31}$$

where:

- S_j is the storage coefficient of porous media $[SU]$
- h_j is the water head of the porous media $[m]$
- T_j is the transmissivity of porous media $[m^2\, s^{-1}]$
- Q_j is the water input $[m\, s^{-1}]$

Storage coefficient and transmissivities are the descriptive parameters of the porous layers. They are defined as:

$$T_j = e_j K_j \tag{4.32}$$

and

$$S_j = \rho_w \omega_j g e_j \left[\beta_w - \frac{\alpha}{\omega_j}\right], \tag{4.33}$$

where

- e_j is the thickness of the considered layer $[m]$
- K_j is the permeability of the porous media $[m\, s^{-1}]$
- ρ_w is the density of fresh water $[kg\, m^{-3}]$
- ω_j is the porosity of the media $[SU]$
- g is the gravitational acceleration $[m\, s^{-2}]$
- β_w is the compressibility of water $[Pa^{-1}]$
- α is the compressibility of the solid phase of the porous media $[Pa^{-1}]$

4.5.1.2 Specificities of the IDS

The main specificity of the IDS is that it allows us to set up a maximum limit for the water head. This is dealt with by a penalization method from which the residual is kept, in order to be re-injected into the EDS.

The source term for the IDS is the sum of three possible sources:

- surfacic input given by the melting at the base of the glacier $[m]$
- local input at a given point representing moulin input $[m^{-3}\, s^{-1}]$
- input due to the transfer between the two layers which is dealt with in an implicit matter (See Layer Transfer)

4.5.1.3 Specificities of the EDS

The model could be run without introducing this layer. In this case, it is possible that the model does not conserve the mass of water, depending on the setting of the upper limit for the IDS. If the layer is used, it is usually not active on the whole domain. The initial activation process is driven by the water head in the IDS and then by the water head in the EDS. More information about the activation process can be found in *de Fleurian et al.* [2014]. Improvements from the version presented in *de Fleurian et al.* [2014] include a varying thickness for the EDS layer, which allows us to close back the EDS when the water volume becomes too low and can be evacuated by the IDS only. The thickness evolution is defined as follows:

$$\frac{\partial e_e}{\partial t} = g\frac{\rho_w e_e K_e}{\rho_{ice} L_{ice}} (\nabla h_e)^2 - 2\left[\frac{N}{Bn}\right]^n \tag{4.34}$$

where:

- ρ_{ice} is the density of the ice $[kg\, m^{-3}]$
- L_{ice} is the latent heat of fusion for the ice $[J\, kg^{-1}]$
- N is the effective pressure $[Pa]$
- B is the ice hardness or rigidity $[Pa\, s^{1/n}]$
- n is Glen's flow law exponent, generally taken as equal to 3 $[SU]$

4.5.2 Transfer equation

The transfer between the two layers is based on the water head difference in the two systems. The transfer term Q_t is as follows:

$$Q_{\mathrm{t}} = \varphi(h_i - h_e). \tag{4.35}$$

where:

- φ is the leakage time from one layer to the other $[s^{-1}]$

The leakage time φ is a characteristic time needed for the water to pass from one drainage system to the other. This corresponds to the crossing of a less permeable layer inbetween the inefficient and efficient layers.

4.5.2.1 Boundary conditions

The natural boundary condition is a no flow condition, which is what is kept on the upstream model boundaries. The water head is then fixed at the snouts of glaciers.

4.5.3 Model parameters

The parameters relevant to the hydrology solution can be displayed by typing:

```
>> md.hydrology
```

These parameters are of three different types

4.5.3.1 General parameters

- `md.hydrology.water_compressibility`: compressibility of water [Pa^-1]
- `md.hydrology.isefficientlayer`: do we use an efficient drainage system (1: true; 0: false)
- `md.hydrology.penalty_factor`: exponent of the value used in the penalization method (dimensionless)
- `md.hydrology.penalty_lock`: stabilize unstable constraints that keep zigzagging after n iteration (default is 0, no stabilization)
- `md.hydrology.rel_tol`: tolerance of the nonlinear iteration for the transfer between layers (dimensionless)
- `md.hydrology.max_iter`: maximum number of nonlinear iteration
- `md.hydrology.sedimentlimit_flag`: what kind of upper limit is applied for the inefficient layer
  - 0: no limit
  - 1: user defined: sedimentlimit
  - 2: hydrostatic pressure
  - 3: normal stress
- `md.hydrology.transfer_flag`: what kind of transfer method is applied between the layers
  - 0: no transfer
  - 1: constant leakage factor: leakage_factor
- `md.hydrology.leakage_factor`: user defined leakage factor [m]
- `md.hydrology.basal_moulin_input`: water flux at a given point [m3 s-1]

4.5.3.2 IDS parameters

Also called sediment layer

- `md.hydrology.spcsediment_head`: sediment water head constraints (`NaN` means no constraint) (m above MSL)
- `md.hydrology.sediment_compressibility`: sediment compressibility [Pa^-1]
- `md.hydrology.sediment_porosity`: sediment (dimensionless)
- `md.hydrology.sediment_thickness`: sediment thickness [m]
- `md.hydrology.sediment_transmitivity`: sediment transmitivity [m^2/s]

4.5.3.3 EDS parameters

Also called EPL layer (Equivalent Porous Layer)

- `md.hydrology.spcepl_head`: epl water head constraints (NaN means no constraint) [m above MSL]
- `md.hydrology.mask_eplactive_node`: active (1) or not (0) EPL
- `md.hydrology.epl_compressibility`: epl compressibility [Pa^-1]
- `md.hydrology.epl_porosity`: epl [dimensionless]
- `md.hydrology.epl_initial_thickness`: epl initial thickness [m]
- `md.hydrology.epl_max_thickness`: epl maximal thickness [m]
- `md.hydrology.epl_conductivity`: epl conductivity [m^2/s]

4.5.4 Running a simulation

To run a transient simulation, use the following command:

```
>> md=solve(md,TransientSolutionEnum);
```

The first argument is the model, the second is the nature of the simulation one wants to run. The default for the transient simulation does not include the resolution of the hydrological model. One should introduce the following lines in the run launchers to enable the hydrology:

- For a standalone hydrology model

```
>> md.transient=deactivateall(md.transient);
>> md.transient.ishydrology=1;
```

- To add the hydrology to a transient simulation

```
>> md.transient.ishydrology=1;
```

Running a steady state simulation, is done with the following command:

```
>> md=solve(md,HydrologySolutionEnum);
```

4.6 Shreve's Hydrology model

4.6.1 Physical basis

This model is the one described in *Le Brocq et al.* [2009]. Here we present only the main equations.

4.6.1.1 Water column

The model applied here is the most simplistic form of the water-film model, as described by the Weertman theory [*Weertman*, 1957]. The model solves for the thickness w of the water-film as follows:

$$\frac{\partial w}{\partial t} = S - \nabla \cdot \mathbf{u}_w w \tag{4.36}$$

where:

- S is the source term $[m\,s^{-1}]$
- $\mathbf{u}_w$ is the water velocity vector $[m\,s^{-1}]$

The water velocity vector $\mathbf{u}_w$ is a depth-averaged two dimensional horizontal vector, which is computed using a theoretical treatment of laminar flow between two parallel plates:

$$\mathbf{u}_w = \frac{w^2}{12\mu}\nabla\phi \tag{4.37}$$

- ϕ is the hydraulic potential $[Pa]$
- μ is the water viscosity $[Pa\,s]$

In this model, the hydraulic potential ϕ is defined following the Shreve approximation [*Shreve*, 1972], which hypothesizes a null effective pressure. Assuming this null effective pressure gives the hydraulic potential gradient as follows:

$$\nabla\phi = \rho_{ice} g \nabla s + (\rho_w - \rho_{ice})\, g \nabla h \tag{4.38}$$

where

- ρ_{ice} is the density of the ice $[kg\, m^{-3}]$
- ρ_w is the density of fresh water $[kg\, m^{-3}]$
- s is the surface elevation $[m]$
- g is the gravitational acceleration $[m\, s^{-2}]$
- h is the bedrock elevation $[m]$

4.6.1.2 Numerical implementation

To stabilize the equation, artificial diffusion might be added to the left hand side:

$$\frac{\partial w}{\partial t} + \nabla\left(\mathfrak{D}\nabla w\right) = S - \nabla\cdot\mathbf{u}_w w \tag{4.39}$$

where $\mathfrak{D}$ is the artificial diffusivity. We take

$$\mathfrak{D} = \frac{h}{2}\begin{pmatrix} |vx| & 0 \\ 0 & |vy| \end{pmatrix} \tag{4.40}$$

4.6.2 Model parameters

The parameters relevant to the water column solution can be displayed by typing:

```
>> md.hydrology
```

- `md.hydrology.spcwatercolumn`: water thickness constraints (`NaN` means no constraint) $[m]$
- `md.hydrology.stabilization`: artificial diffusivity (default is 1).

4.6.3 Running a simulation

To run a simulation, use the following command:

```
>> md=solve(md,HydrologySolutionEnum);
```

4.7 Damage mechanics

4.7.1 Physical basis

Damage is a state variable introduced to account for the influence of fractures on ice flow, while maintaining a continuum representation of the ice domain. For purely viscous ice flow modeling, damage is linked to flow enhancement–specifically the increase in strain rate–due to a fracture or a multitude of fractures in the ice.

4.7.1.1 Inferring damage from remote sensing data

Remote sensing data can be used to calculate damage from the static stress balance in the ice. At present, this is only implemented in two dimensions for the SSA approximations to ice flow. Damage can be inferred in one of two ways:

- Inverting for damage directly
- Inverting for ice rigidity B and then post-processing to determine damage (and optionally backstress)

4.7.1.2 Inverting for damage directly

For the SSA equations, the damage-dependent ice viscosity (μ) is:

$$\mu = \frac{(1-D)\,B}{2\dot{\varepsilon}_e^{\frac{n-1}{n}}} \tag{4.41}$$

where:

- D is damage
- B is the ice rigidity
- $\dot{\varepsilon}_e$ is the effective strain rate
- n is the flow law exponent

Damage can be calculated using an inverse control method in the same manner as an inversion for the ice rigidity B. Simply specify the following field in `md.inversion`:

- `md.inversion.control_parameters=['DamageDbar']` (Python)
- `md.inversion.control_parameters={'DamageDbar'}` (MATLAB)

The remainder of the inversion procedure is described on the Inverse Methods page. This was the procedure followed by *Borstad et al.* [2012] in determining the damage for the Larsen B ice shelf prior to its collapse (see the ISSM Publications List for a link to the paper).

4.7.1.3 Post-processing to determine damage

Damage can also be calculated from the results of an inverse method solution for ice rigidity B. This procedure uses the analytical solution for the strain rate of a damaged ice shelf, derived by *Borstad et al.* [2013]:

$$\dot{\varepsilon}_{xx} = \theta \left[\frac{1/2\rho_i \left(1-\rho_i/\rho_w\right) gH - \sigma_b}{(1-D)\,B} \right]^n \tag{4.42}$$

where:

- $\dot{\varepsilon}_{xx}$ is the longitudinal strain rate
- θ accounts for the lateral and shear strain rate terms
- ρ_i and ρ_w are the densities of ice and seawater, respectively
- g is gravitational acceleration
- H is the ice thickness
- σ_b is the backstress resisting the flow
- D is the damage
- B is the ice rigidity
- n is the flow law exponent

To determine damage, an inverse control method solution for ice rigidity B is first carried out. The initial guess $B_\circ$ for the control method (contained in `md.materials.rheology_B!`) is assumed to be based on a temperature parameterization, given a reasonable estimate of the depth-averaged temperature of the ice. Damage is then calculated in locations where the inverse solution for B is less than the ice rigidity appropriate for the local temperature of the ice. A post-processing function carries out this calculation directly:

```
>> md=damagefrominversion(md);
```

Additionally, the scalar backstress can be calculated from the inversion results:

```
>> md=backstressfrominversion(md);
```

This procedure for calculating damage and backstress was used in *Borstad et al.* [2013] for the Larsen C ice shelf (see the ISSM Publications List for a link to the paper).

4.7.2 Damage Evolution (Under Construction)

A differential equation describing damage evolution in time–both the advection of damage with ice flow as well as the evolution of damage as the stress state changes–is being implemented in ISSM. Check back for updates.

4.8 Transient (time dependent projection)

4.8.1 Physical basis

4.8.1.1 Transient solution

The transient solution is a combination of all the other solutions and modules that allow us to run a model forward in time (between a start time and a final time) using finite differences in time. At each time step of the simulation the following steps are performed in the order noted below:

- thermal solution
- hydrology solution
- stressbalance solution
- damage evolution model
- masstransport solution
- grounding line migration (and geometry update)
- gia solution

Not all solutions have to be included in the transient runs, and each of these functionalities can be activated or deactivated prior to launching the simulation.

4.8.2 Model parameters

The parameters relevant to the transient solution can be displayed by typing:

```
>> md.transient
```

- `md.transient.ismasstransport`: indicates whether a masstransport solution is used in the transient
- `md.transient.isstressbalance`: indicates whether a stressbalance solution is used in the transient

- `md.transient.isthermal`: indicates whether a thermal solution is used in the transient
- `md.transient.isgroundingline`: indicates whether a grounding line migration is used in the transient
- `md.transient.isgia`: indicates whether a postglacial solution is used in the transient
- `md.transient.isdamageevolution`: indicates whether damage evolution is used in the transient
- `md.transient.islevelset`: level set, not implemented yet
- `md.transient.ishydrology`: indicates whether a hydrology solution is used in the transient
- `md.transient.requested_outputs`: list of additional outputs requested

The solution will also use fiels from the following classes for each of the solution used:

- `md.masstransport`: for parameters related to the masstransport solution
- `md.stressbalance`: for parameters related to the stressbalance solution
- `md.thermal`: for parameters related to the thermal solution
- `md.groundingline`: for parameters related to grounding line migration
- `md.gia`: for parameters related to the postglacial solution
- `md.damage`: for parameters related to damage evolution
- `md.hydrology`: for parameters related to the hydrology solution
- `md.initialization`: for initial values of model fields (velocity, temperature, ...)
- `md.timestepping`: for parameters related to time stepping (initial time, final time, length of time steps, ...)

4.8.2.1 Time stepping

Each solution requested is computed at each time step. The time step has either a fixed duration (specified by the user before the simulation is launched) or a varying duration based on the CFL (CourantâĂŞFriedrichsâĂŞLewy) condition (necessary condition for the stability of certain partial differential equations).

The parameters relevant to the time stepping can be displayed by typing:

```
>> md.timestepping
```

- `md.timestepping.start_time`: simulation starting time (year)
- `md.timestepping.final_start`: final time to stop the simulation (year)
- `md.timestepping.time_step`: length of time steps (year)
- `md.timestepping.time_adapt`: to indicate if the CFL condition is used to define time step ?
- `md.timestepping.cfl_coefficient`: coefficient applied to cfl condition
- `md.timestepping.interp_forcings`: interpolate in time between requested forcing values ? (0 or 1)

4.8.3 Forcing a transient

To specify a transient forcing, the user must add a time value to the end (i.e. in the `end+1` position) of the variable to be forced. This means that a transient forcing will no longer be a single column of length `n`. Instead, it will be a matrix (or a series of columns), and each column will be of length `n+1`.

For example, let `smb` be values of surface mass balance. Below, we impose `smb` at year 10 and then impose a decrease of 1 m/yr in surface mass balance everywhere at year 20.

```
>> md.surfaceforcings.mass_balance = [ smb smb-1];
>> md.surfaceforcings.mass_balance = [ md.surfaceforcings.mass_balance; [10 20]];
```

Prior to first and after last imposed time, ISSM will impose constant surface mass balance values. In the example above, the surface mass balance is assumed constant prior to year 10, and again after year 20. Forcing values will be equal to `smb` prior to year 10 and `smb-1` after year 20.

Between years 10 and 20, ISSM will treat all forcings according to the value set in the time stepping parameter `interp_forcings`.

By default, `md.timestepping.interp_forcings = 1`. This means that between the user-imposed times, forcings are linearly interpolated. For the example above, the model will linearly increase surface mass balance from `smb` to `smb-1` between years 10 and 20.

The user must set `md.timestepping.interp_forcings = 0` to turn this feature off and impose a step-wise forcing. When `interp_forcings = 0`, the forcing value will change only at the times designated by the user. After the last user-specified time, the forcing will remain constant. In the example above, the surface mass balance will be equal to `smb` up until time 20. At time 20, the surface mass balance will be changed to `smb-1`, and will remain at these values until the end of the simulation.

4.8.4 Running a simulation

To run a simulation, use the following command:

```
>> md=solve(md,TransientSolutionEnum);
```

The first argument to `solve` is the model, the second is the nature of the simulation one wants to run.

4.9 Grounding Lines

4.9.1 Physical basis

4.9.1.1 Hydrostatic equilibrium

The position of the grounding line is determined by a floatation criterion: ice is floating if its thickness, H, is equal or lower than the floating height H_f defined as:

$$H_f = -\frac{\rho_w}{\rho_i} r,\ r < 0 \tag{4.43}$$

where ρ_i is the ice density, ρ_w the ocean density and r the bedrock elevation (negative if below sea level). Grounding line is therefore located where $H = H_f$:

$$\begin{array}{lcll} H & > & H_f & \text{ice is grounded} \\ H & = & H_f & \text{grounding line position} \\ H & < & H_f & \text{ice is floating} \end{array} \tag{4.44}$$

Each element of the mesh is either grounded or floating: floatation criterion is determined on each vertex of the triangle and if at least one vertex of the triangle is floating, the element is considered floating and no friction is applied. Otherwise, if the three vertices are grounded, the element is considered grounded. We refer to this type of grounding line migration as `'SoftMigration'`.

Sub-element parameterization can also be used to track the position of the grounding line within an element and improve accuracy of the results. The floating condition is a 2D field and the grounding line position is determined by the line where $H = H_f$, so it is located anywhere within an element. Some elements are therefore partly grounded and partly floating. Two different schemes of sub-element parameterizations have been implemented.

In the first case, the basal friction coefficient C is reduced to match the amount of grounded ice in the element as proposed by *Pattyn et al.* [2006] and *Gladstone et al.* [2010] but for a 2D element:

$$C_g = C\,\frac{A_g}{A} \tag{4.45}$$

where C_g is the applied basal friction coefficient for the element partially grounded, A_g is the area of grounded ice of this element and A is the total area of the element. We refer to this type of grounding line parameterization as `'SubelementMigration'`.

In the second case, the basal friction computed for partly grounded elements is integrated only on the part of the element that is grounded. This can be done simply by changing the integration area from the initial element to the grounded part of the element, over which the basal friction is unchanged. We refer to this type of grounding line parameterization as `'SubelementMigration2'`.

The sub-element parameterizations are described in details in *Seroussi et al.* [2014a].

4.9.1.2 Contact mechanics

Grounding line migration can be advantageously based on contact mechanics when solving the stress balance equations with a full-Stokes models [*Nowicki and Wingham*, 2008; *Durand et al.*, 2009].

This capability is currently under development.

4.9.2 Model parameters

The parameters relevant to the grounding line migration can be displayed by typing:

```
>> md.groundingline
```

- `md.groundingline.migration`: type of grounding line migration: `'SoftMigration'`,`'AgressiveMigration'`,`'SubelementMigr` or `'None'`

4.9.3 Running a simulation

To compute grounding line migration, the transient solution must be used and all solutions except the grounding line migration must be deactivated (see Transient solution):

```
>> md=solve(md,TransientSolutionEnum);
```

The first argument to `solve` is the model, the second is the nature of the simulation one wants to run.

4.10 Glacial Isostatic Adjusment (GIA)

4.10.1 Physical basis

The ISSM/GIA model assumes that the ice sheet rests on top of the solid Earth, which is considered to be a simple two-layered incompressible continuum with upper elastic lithosphere floating on the viscoelastic (Maxwell material) mantle half-space. Coordinate transformations allow simple axisymmetric solutions for the deformation of pre-stressed solid Earth (subject to a normal surface traction of ice/ocean) to retrieve semi-analytical solutions of vertical displacement at the lithosphere surface.

4.10.2 Vertical surface displacement

Vertical displacement at the lithosphere surface (i.e., ice/ocean-bedrock interface), $w(r,t)$, is the most relevant field variable for GIA assessment. For brevity, hereinafter, this is referred to as the GIA solution. Semi-analytical GIA solution is given by *Ivins and James* [1999]:

$$w(r,t) = \int_0^\infty k \left[\frac{4\mu_1^e \alpha}{2k\mu_1^e + \rho_1 g} \hat{Q}_0(k,t) J_1(k\alpha) \right] J_0(kr)\, \mathrm{d}k, \tag{4.46}$$

where:

- r is the radial distance from the centre of the cylindrical disc load
- t is the evaluation time
- k is the Hankel transform variable of r (or wavenumber)
- α is the radius of the cylindrical disc load
- μ_1^e is the shear modulus of elasticity of lithosphere
- ρ_1 is the lithosphere density
- g is the vertical component of the gravity vector
- $J_v(kr)$ is the v-th order Bessel function of the first kind
- $\hat{Q}_0(k,t)$ accounts for the integrated influence of ice loading history (cf. Figure 1) at the evaluation time t. (Note that $\hat{f}_v(k)$ is the v-th order Hankel transform of function $f(r)$.)

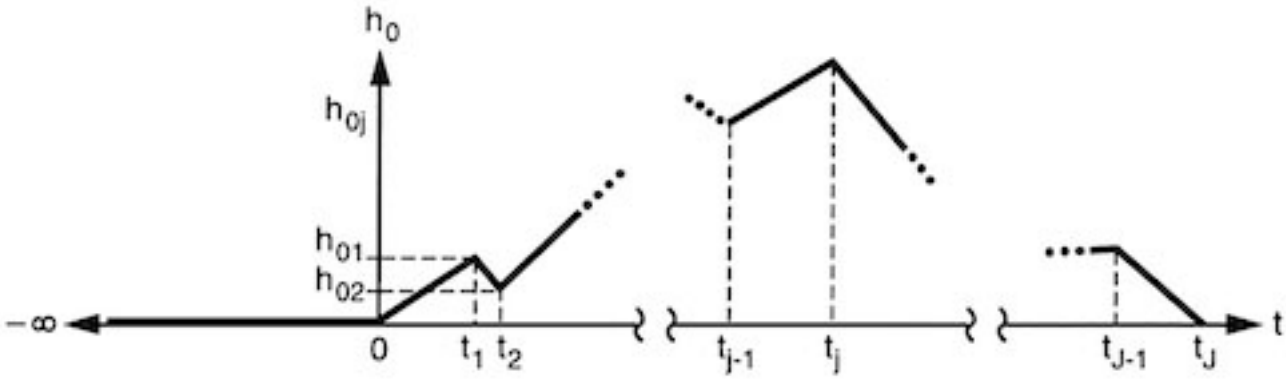


Figure 4.3: Schematic of evolution of piecewise continuous load height, h_0, with J linear segments (from *Ivins and James* [1999]). For j-th segment, we can compute m_j and b_j (cf. Eqs. 3–4) based on the ice load at time t_{j-1} and t_j. At t_j, for example, ice load at the lithosphere surface is given by $\rho_0 g h_{0j}$, where ρ_0 is the ice density.

Assuming $t_{J-1} < t \leq t_J$, the term $\hat{Q}_0(k,t)$ can be written as follows

$$\hat{Q}_0(k,t) = \sum_{j=1}^{J} {}_j\hat{Q}_0(k,t). \tag{4.47}$$

For $j \leq (J-1)$,

$${}_j\hat{Q}_0(k,t) = \sum_{p=1}^{2} \left\{ \frac{m_j \xi_p}{\gamma_p^2} \left[(\gamma_p t_j - 1)\, \mathrm{e}^{\gamma_p (t_j - t)} - (\gamma_p t_{j-1} - 1)\, \mathrm{e}^{\gamma_p (t_{j-1} - t)} \right] + \frac{b_j \xi_p}{\gamma_p} \left[\mathrm{e}^{\gamma_p (t_j - t)} - \mathrm{e}^{\gamma_p (t_{j-1} - t)} \right] \right\}, \tag{4.48}$$

and for $j = J$ (i.e. the last load segment),

$${}_j\hat{Q}_0(k,t) = \sum_{p=1}^{2} \left\{ \frac{m_j \xi_p}{\gamma_p^2} \left[(\gamma_p t - 1) - (\gamma_p t_{j-1} - 1)\, \mathrm{e}^{\gamma_p (t_{j-1} - t)} \right] + \frac{b_j \xi_p}{\gamma_p} \left[1 - \mathrm{e}^{\gamma_p (t_{j-1} - t)} \right] \right\} + \left(c_2 + \frac{1}{4k\mu_1^e} \right) (m_j t + b_j), \tag{4.49}$$

where:

- m_j is the slope of the linear j-th load segment
- b_j is the y-intercept of the linear j-th load segment
- γ_p is the inverse decay time
- ξ_p is the amplitude factor

For $p = 1, 2$, the inverse decay times are given by

$$\gamma_p = \frac{d_1 \pm \sqrt{d_1^2 - 4d_0}}{2}, \tag{4.50}$$

and the amplitude factors by

$$\xi_p = \frac{(-1)^p}{(\gamma_2 - \gamma_1)} \left[(-c_2\gamma_p + c_1)\,\gamma_p - c_0\right]. \tag{4.51}$$

Parameters appearing in Eqs. (5) and (6) are defined as follows

$$c_0 = \frac{h_1}{\mu_2^e \tau_m^2} c_0',\ c_1 = \frac{h_1}{\mu_2^e \tau_m} c_1',\ c_2 = \frac{h_1}{\mu_2^e} c_2',\ d_0 = \frac{1}{\tau_m^2} d_0' \text{ and } d_1 = \frac{1}{\tau_m} d_1', \tag{4.52}$$

where:

- h_1 is the lithosphere thickness
- $\tau_m = \eta/\mu_2^e$ is the Maxwell relaxation time
- η is the effective viscosity of mantle
- μ_2^e is the shear modulus of elasticity of mantle
- parameters with primes, e.g. c_0', are dimensionless (listed in Table 1)

with the following dimensionless parameters:

- $d_2' = b_0' + b_1' + b_2' + b_3' + b_4' + b_5' + b_6' + b_7'$
- $d_1' = \left[b_2' + b_3' + b_4' + 2\,(b_5' + b_6' + b_7')\right]/d_2'$
- $d_0' = (b_5' + b_6' + b_7')/d_2'$
- $c_2' = (a_0' + a_1' + a_2' + a_3')/d_2'$
- $c_1' = \left[a_1' + 2\,(a_2' + a_3')\right]/d_2'$
- $c_0' = (a_2' + a_3')/d_2'$

where:

- $a_0' = -2k' \left\{1 + \mathrm{e}^{2k'} \left[1 + 2k'\,(1 + k')\right]\right\}$
- $a_1' = 4k'\,R_\mu^e - R_\rho^- \left\{1 + \mathrm{e}^{2k'} \left[1 + 2k'\,(1 + k')\right]\right\}$
- $a_2' = -2k' \left(R_\mu^e\right)^2 \left[1 - \mathrm{e}^{2k'} - 2k'\,\mathrm{e}^{2k'}\,(1 + k')\right]$
- $a_3' = R_\mu^e R_\rho^- \left[1 - \mathrm{e}^{2k'}\,(1 + 2k')\right]$
- $b_0' = 4(k')^2 R_\mu^e \left[1 + \mathrm{e}^{4k'} + 2\,\mathrm{e}^{2k'} \left(1 + 2(k')^2\right)\right]$
- $b_1' = -2k' R_\rho^1 \left(1 - \mathrm{e}^{4k'} + 4k'\,\mathrm{e}^{2k'}\right)$
- $b_2' = -8(k')^2 \left(R_\mu^e\right)^2 \left(1 - \mathrm{e}^{4k'}\right)$
- $b_3' = 2k' R_\mu^e \left[R_\rho^+ \left(1 + \mathrm{e}^{4k'}\right) + 2R_\rho^-\,\mathrm{e}^{2k'} \left(1 + 2(k')^2\right)\right]$
- $b_4' = -R_\rho^1 R_\rho^- \left(1 - \mathrm{e}^{4k'} + 4k'\,\mathrm{e}^{2k'}\right)$
- $b_5' = 4(k')^2 \left(R_\mu^e\right)^3 \left[\left(1 - \mathrm{e}^{2k'}\right)^2 - 4(k')^2 \mathrm{e}^{2k'}\right]$
- $b_6' = -2k' \left(R_\mu^e\right)^2 R_\rho^2 \left(1 - \mathrm{e}^{4k'} - 4k'\,\mathrm{e}^{2k'}\right)$

- $b'_7 = R^e_\mu R^1_\rho R^-_\rho \left(1 - \mathrm{e}^{2k'}\right)^2$

The following set of non-dimensionlized parameters are defined, as needed to express dimensionless terms listed in Table 2

$$k' = kh_1, R^e_\mu = \frac{\mu^e_1}{\mu^e_2}, R^1_\rho = \frac{gh_1\rho_1}{\mu^e_2}, R^2_\rho = \frac{gh_1\rho_2}{\mu^e_2}, R^+_\rho = \frac{gh_1\left(\rho_2+\rho_1\right)}{\mu^e_2}, R^-_\rho = \frac{gh_1\left(\rho_2-\rho_1\right)}{\mu^e_2}, \tag{4.53}$$

where:

- ρ_2 is the mantle density

4.10.3 Numerical implementation

In the Cartesian frame of ISSM, we treat the size of ice load as the property of mesh element and compute the GIA solution at each node of the element [*Adhikari et al.*, 2014]. Individual 2-D (xy-plane) mesh elements are defined as the equivalence of footprint (i.e., projection onto the xy-plane) of cylindrical disc loads, ensuring that the corresponding element and disc both share the same origin and plan-form area. The height of ice load is then assigned to each element such that the average normal tractional force on the corresponding area of bedrock is conserved. At each node within the domain, the final GIA solutions are computed by integrating the solutions due to individual disc loads, defined as the property of mesh elements.

4.10.4 Model parameters

The parameters relevant to the GIA solution can be displayed by typing:

```
>> md.gia
```

- `md.gia.mantle_viscosity`: mantle viscosity (in Pa s)
- `md.gia.lithosphere_thickness`: lithosphere thickness (in km)
- `md.gia.cross_section_shape`: shape of the cylindrical disc load; 1: square-edged (default) 2: elliptical

The solution will also use the following model fields:

- `md.materials.lithosphere_shear_modulus`: shear modulus of lithosphere (in Pa)
- `md.materials.lithosphere_density`: lithosphere density (in g/cm^3)
- `md.materials.mantle_shear_modulus`: shear modulus of mantle (in Pa)
- `md.materials.mantle_density`: mantle density (in g/cm^3)
- `md.timestepping.start_time`: GIA evaluation time t (in yr)
- `md.timestepping.final_time`: $t_J(>t)$ in Figure 1 (in yr).
- `md.geometry.thickness`: ice loading history in the $J \times 2$ matrix form; the j-th row, for example, should be defined as $[h_{0j}, t_j]$ (cf. Figure 1).

4.10.5 ISSM Configuration

To activate the GIA model, add the following in the configuration script and compile ISSM:

```
--with-math77-dir="$ISSM_DIR/externalpackages/math77/install"
```

4.10.6 Running a simulation

To run a simulation, use the following command:

```
>> md=solve(md,GiaSolutionEnum);
```

The first argument is the model, the second is the nature of the simulation one wants to run.

Chapter 5

Parameterization of physical processes

5.1 Positive Degree Day (PDD)

5.1.1 Physical basis

5.1.1.1 Positive degree day method

A standard positive degree day (PDD) method is used to compute the surface masse balance (ice ablation and accumulation) from the temperature and precipitation fields. The hourly temperatures are assumed to have a normal distribution, of standard deviation $\sigma_{PDD} = 5.5\,^{\circ}$C, around the monthly mean (T_m). The number of days for which the temperature is above $0\,^{\circ}$C in a year is computed as follows:

$$\mathrm{PDD} = \frac{1}{\sigma_{PDD}\sqrt{2\pi}} \int_0^{1year} \int_{0\,^{\circ}\mathrm{C}}^{T_m+2.5\sigma_{PDD}} Texp\left[\frac{-(T-T_m)^2}{2\sigma_{PDD}^2}\right] dT\,dt \tag{5.1}$$

The amount of snow and ice that melts is assumed to be proportional to the number of positive degree days. Snow is melted first and the remaining positive degree days are used to melt ice. A dependence to the mean June/July/August temperature (T_{jja}) is added to get the ablation rate factor for snow (γ_{snow}) and ice (γ_{ice}):

$$\begin{aligned}
\gamma_{\mathrm{ice}} &= \begin{cases} 17.22 \text{ mm/PDD} & T_{jja} \le -1\,^{\circ}\mathrm{C}, \\ 0.0067 \times (10\text{-}T_{jja})^3 + 8.3 \text{ mm/PDD} & -1\,^{\circ}\mathrm{C} < T_{jja} < 10\,^{\circ}\mathrm{C}, \\ 8.3 \text{ mm/PDD} & 10\,^{\circ}\mathrm{C} \le T_{jja} \end{cases} \\
&\text{and} \\
\gamma_{\mathrm{snow}} &= \begin{cases} 2.65 \text{ mm/PDD} & T_{jja} \le -1\,^{\circ}\mathrm{C}, \\ 0.15 \times T_{jja} + 2.8 \text{ mm/PDD} & -1\,^{\circ}\mathrm{C} < T_{jja} < 10\,^{\circ}\mathrm{C}, \\ 4.3 \text{ mm/PDD} & 10\,^{\circ}\mathrm{C} \le T_{jja} \end{cases}
\end{aligned} \tag{5.2}$$

A fraction of the melted snow is refrozen. The amount of superimposed ice for a year is:

$$\text{superimposed ice} = \begin{cases} \min[\mathrm{Pr} + \mathrm{M},\ 2.2 \times (\mathrm{Ps} - \mathrm{M}) - \mathrm{d} \times \mathrm{ci}\ /\mathrm{L} \times \min(\mathrm{Tsurf}\ ,\ 0\,^{\circ}\mathrm{C})] & \mathrm{M} < \mathrm{Ps}\ , \\ \min[\mathrm{Pr} + \mathrm{M},\ \mathrm{d}\ \times \mathrm{ci}\ /\mathrm{L} \times \min(\mathrm{Tsurf}\ ,\ ^{\circ}\mathrm{C})] & \mathrm{M} > \mathrm{Ps} \end{cases} \tag{5.3}$$

where:

- Pr is the rainfall in a year
- Ps is the snow fall in a year
- M is the snow melt in a year
- 2.2 is the capillarity factor
- d is the active thermodynamic layer (set to 1 m)

- ci is the ice specific heat capacity ($152.5 + 7.122\mathrm{T}\ Jkg^{-1}K^{-1}$)
- L is the latent heat fusion ($3.35 \times 10^5\ Jkg^{-1}$)
- $Tsurf$ is the surface temperature

A normal distribution of the hourly temperature is also assumed to compute the amount of snow accumulation from the precipitation. A lower standard deviation $\sigma_{RS} = \sigma_{PDD} - 0.5$ is assumed in that case to take into account the smaller temperature variability during cloudy days. Precipitation is considered to be snow when the temperature is below $0\,^{\circ}\mathrm{C}$.

$$\frac{\text{accumulation}}{\text{precipitation}} = \frac{\rho_i}{\rho_w \sigma_{RS}\sqrt{2\pi}} \int_0^{1year} \int_{T_m - 2.5\sigma_{RS}}^{0\,^{\circ}\mathrm{C}} exp\left[\frac{-(T-T_m)^2}{2\sigma_{RS}^2}\right] dTdt \tag{5.4}$$

5.1.1.2 Temperature and precipitation forcing (Under development)

If precipitations come from another elevation than the surface elevation of the ice, it can be adjusted to take into account the elevation desertification effect.

If the forcing temperatures are provided for a constant altitude, a lapse rate of 6.5°/km is used to adjust them to the surface elevation of each step.

5.1.2 Model parameters

The parameters relevant to the positive degree day and $\delta^{18}O$ parameterization methods can be displayed by typing: The lapse rate is computed as an weighted mean of the present day ($rlaps$) and LGM ($rlapslgm$) lapse rate as

$$rtlaps = TdiffTime * rlapslgm + (1. - TdiffTime) * rlaps \tag{5.5}$$

where TdiffTime is the time interpolation parameter (Tdiff) at the integration time.

The surface temperature ($Tsurf$) is the yearly average temperature computed from the monthly temperature tstar. tstar is computed as the present day temperature plus the temperature difference, $tdiffh$, between LGM and present day.

$$tstar = tdiffh + TemperaturesPresentday[imonth] - rtlaps \times \max st, sealev \times 0.001; \tag{5.6}$$

st is the difference between the surface elevation and the elevation from temperature source

$$st = (s - s0t)/1000 \tag{5.7}$$

and tdiffh is the weighted mean between the present day and lgm temperature

$$tdiffh = TdiffTime \times (TemperaturesLgm[imonth] - TemperaturesPresentday[imonth]) \tag{5.8}$$

```
>> md.surfaceforcings
```

- `isdelta18o`: is temperature and precipitation delta18o parametrization activated (0 or 1, default is 0)
- `desfac`: desertification elevation factor (between 0 and 1, default is 0.5) (m)
- `s0p`: should be set to elevation from precipitation source (between 0 and a few 1000s m, default is 0) (m)
- `s0t`: should be set to elevation from temperature source (between 0 and a few 1000s m, default is 0) [m]
- `rlaps`: present day lapse rate (degree/km)
- `rlapslgm`: LGM lapse rate (degree/km)
- `Pfac`: time interpolation parameter for precipitation, 1D (year)
- `Tdiff`: time interpolation parameter for temperature, 1D (year)

- `sealev`: sea level (m)
- `monthlytemperatures`: monthly surface temperatures (K), required if pdd is activated and delta18o not activated
- `precipitation`: surface precipitation (m/yr water eq)
- `temperatures_presentday`: monthly present day surface temperatures (K), required if pdd is activated and delta18o activated
- `temperatures_lgm`: monthly LGM surface temperatures (K), required if pdd is activated and delta18o activated
- `precipitations_presentday`: monthly surface precipitation (m/yr water eq), required if pdd is activated and delta18o activated
- `delta18o`: delta18o, required if pdd is activated and delta18o activated
- `delta18o_surface`: surface elevation of the delta18o site, required if pdd is activated and delta18o activated

5.1.3 Running a simulation

To turn this module on in a simulation, use the following command:

```
>>    md.surfaceforcings=SMBpdd();
```

5.2 Other surface mass balance models (SMB)

5.2.1 SMB (default)

The default surface mass balance model applies the surface mass balance that's provided by the model without any modifications. This model can be selected by typing:

```
>> md.surfaceforcings = SMB();
```

One can display the following fields by typing:

```
>> md.surfaceforcings
```

- `md.surfaceforcings.mass_balance`: surface mass balance (in m/yr ice equivalent)

5.2.2 SMB components

The `SMBcomponents` model computes surface mass balance using the component parameters provided. The components expected are: accumulation, runoff, and evaporation. All components are typically expected to be given as positive values. In the model computation of surface mass balance, runoff and evaporation are considered as mass lost and accumulation is considered as mass gain.

The components model can be selected by typing:

```
>> md.surfaceforcings = SMBcomponents();
```

One can display the following fields by typing:

```
>> md.surfaceforcings
```

- `surface forcings parameters (SMB=accumulation-runoff-evaporation)`:
- `md.surfaceforcings.accumulation`: accumulated snow [m/yr ice eq]
- `md.surfaceforcings.runoff` : amount of ice melt lost from the ice column [m/yr ice eq]
- `md.surfaceforcings.evaporation` : amount of ice lost to evaporative processes [m/yr ice eq]

5.2.3 SMB melt components

Like the SMBcomponents model, the SMBmeltcomponents model computes surface mass balance using the component parameters provided by the user. The components expected are: accumulation, evaporation, melt, and refreeze. All components are typically expected to be given as positive values. In the model computation of surface mass balance, melt and evaporation are considered as mass lost while accumulation and refreeze are considered as mass gain.

The melt components model can be selected by typing:

```
>> md.surfaceforcings = SMBmeltcomponents();
```

```
>> md.surfaceforcings
```

- `surface forcings parameters with melt (SMB=accumulation-evaporation-melt+refreeze)`:
- `md.surfaceforcings.accumulation`: accumulated snow [m/yr ice eq]
- `md.surfaceforcings.evaporation` : amount of ice lost to evaporative processes [m/yr ice eq]
- `md.surfaceforcings.melt` : amount of ice melt in ice column [m/yr ice eq]
- `md.surfaceforcings.refreeze` : amount of ice melt refrozen in ice column [m/yr ice eq]

5.2.4 SMB gradients method

This surface mass balance model is based on the mass balance gradients method described in *Helsen et al.* [2012]. To activate this method, the user must provide a climatology and a reference ice surface profile. The method will evolve the surface mass balance forcing through time, according to deviations of ice surface height. Required parameters include, at each vertex: (1) a reference surface mass balance field; (2) a reference ice elevation at each vertex; (3) a pre-determined slope of the linear regression between positive surface mass balance and ice surface height; and (4) a pre-determined slope of the linear regression between negative surface mass balance and ice surface height. Surface mass balance values are expected in units of millimeters of water equivalent per year and elevations are expected in meters.

The gradients model can be selected by typing:

```
>> md.surfaceforcings = SMBgradients();
```

```
>> md.surfaceforcings
```

- `md.surfaceforcings.href` : reference elevation from which deviation is used to calculate SMB adjustment in smb gradients method [m]
- `md.surfaceforcings.smbref`: reference smb from which deviation is calculated in smb gradients method [mm/yr water equiv]
- `md.surfaceforcings.b_pos` : slope of hs - smb regression line for accumulation regime required if smb gradients is activated
- `md.surfaceforcings.b_neg` : slope of hs - smb regression line for ablation regime required if smb gradients is activated

5.3 Basal friction

5.3.1 Default Friction law

The default friction law is defined as [*Paterson*, 1994] (p 151):

$$v_b \propto N^{-q} \tau_b^p \tag{5.9}$$

where:

- v_b is the basal velocity magnitude
- τ_b is the basal stress magnitude
- N is the effective pressure
- p and q are friction law exponents

In ISSM, this friction law is implemented in terms of basal stress:

$$\boldsymbol{\tau}_b = -k^2 N^r \|\mathbf{v}_b\|^{s-1} \mathbf{v}_b \tag{5.10}$$

where:

- k is a friction coefficient (variable in space)
- r and s are friction law exponents:

$$r = q/p \qquad\qquad s = 1/p \tag{5.11}$$

This friction law can be selected as follows:

```
>> md.friction = friction();
```

One can display the following fields by typing:

```
>> md.friction
```

- `md.friction.coefficient`: friction coefficient
- `md.friction.p`: p exponent
- `md.friction.q`: q exponent

5.3.2 Weertman Friction law

A Weertman friction law is currently under development. The Weertman friction law reads:

$$v_b = C\tau_b^m \tag{5.12}$$

- C is a friction coefficient (variable in space)
- m is a friction law exponent

In ISSM, this friction law is implemented in terms of basal stress:

$$\boldsymbol{\tau}_b = C^{-1/m} \|\mathbf{v}_b\|^{1/m-1} \mathbf{v}_b \tag{5.13}$$

This friction law can be selected as follows:

```
>> md.friction = frictionweertman();
```

One can display the following fields by typing:

```
>> md.friction
```

- `md.friction.C`: friction coefficient
- `md.friction.m`: m exponent

5.3.3 Thin water layer friction law

The thin water layer friction law is similar to the default friction law except that the effective pressure includes a specified layer of water at the bed:

$$N = g\left(\rho_i H + \rho_w\left(b - w\right)\right) \tag{5.14}$$

when the bedrock is below sea level, and:

$$N = g\left(\rho_i H - \rho_w w\right) \tag{5.15}$$

when the bedrock is above sea level, with:

- N the effective pressure
- ρ_i the ice density
- ρ_w the water density
- H and b ice thickness and bed elevation
- w the water thickness at the ice base

This friction law can be selected as follows:

```
>> md.friction = frictionwaterlayer();
```

One can display all these fields by typing:

```
>> md.friction
```

- `md.friction.coefficient`: friction coefficient
- `md.friction.p`: p exponent
- `md.friction.q`: q exponent
- `md.friction.water_layer`: thin water layer thickness (meters)

Chapter 6

Cluster/Cloud computing

6.1 Cluster computing

ISSM can be run in parallel on clusters or on multi-core computers. This subsection shows how to use this capability.

6.1.1 Setting up the environment to use the parallel mode

We assume users have correctly setup ISSM. Every cluster is different and one might need to create a new cluster file in `$ISSM_DIR/src/m/classes/clusters/`. In most cases, the `generic` cluster can be used:

```
>> md.cluster=generic('name',machine_name);
```

For a local machine, the command `oshostname()` can be used:

```
>> md.cluster=generic('name',oshostname());
```

Many parameters, such as the number of processors, are fields of `md.cluster`. Once those parameters are setup, the solution sequences are called the same way:

```
md=solve(md,StressbalanceSolutionEnum);
```

6.1.2 password-less SSH login

In order to facilitate use of clusters that might be protected by passwords, or to avoid having to input password for each run, one can either set-up a public key authentication or a tunnel between the local host and the cluster.

6.1.2.1 Step 1: simplifying the way you ssh

It is useful to simplify the way you log into your cluster. For example, if your username is `myusername` and the complete cluster name is `mycluster.ess.uci.edu`, you need to log in using this ssh command:

```
$ssh myusername@mycluster.ess.uci.edu
```

This can be simplified by adding the following lines to `~/.ssh/config`:

```
Host mycluster mycluster.ess.uci.edu
   HostName mycluster.ess.uci.edu
   User myusername
```

Now, to log into the cluster, you only need to type:

```
$ssh mycluster
```

you will obviously need to adapt these lines to your own user name and cluster name.

6.1.2.2 Step 2: creating an SSH public/private key

You need to have a SSH public/private key pair. You can check whether the following file exists `~/.ssh/id_rsa.pub`. If you do have a public key, you can skip this step and go to the next step. If you do not, you can create a SSH public/private key pair by typing the following command and following the prompts (no passphrase necessary):

```
$your_localhost% ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/username/.ssh/id_rsa):RETURN
Enter passphrase (empty for no passphrase):RETURN
Enter same passphrase again:RETURN
Your identification has been saved in /Users/username/.ssh/id_rsa.
Your public key has been saved in /Users/username/.ssh/id_rsa.pub.
```

Two files were created: your private key `/Users/username/.ssh/id_rsa`, and the public key `/Users/username/.ssh/id_rsa.pub`. The private key is read-only and only for you, it is used to decrypt all correspondence encrypted with the public key.

6.1.2.3 Step 3: SSH passthrough

The contents of your RSA public key need to be copied to `~/.ssh/authorized_keys` on the system you wish to SSH to without being prompted for a password:

```
$your_localhost%scp ~/.ssh/id_rsa.pub username@your_remotehost:~
```

Now on your remote host, copy the content of `id_rsa.pub`:

```
$your_remotehost%cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
$your_remotehost%rm ~/id_rsa.pub
```

6.1.3 Tunneling

Another possibility is to establish an SSH tunnel between the local host and the cluster. Open a terminal, and connect to the cluster using ssh, by typing:

```
$ ssh -L 1025:localhost:22 login@cluster
```

The port that will be tunneled is the port 1025. To be able to use the tunnel, you will have to change the port setting in the `md.cluster.port` from 0 to 1025. Once this is done, solutions can be solved the exact same way.

6.2 Cloud computing

ISSM can be deployed on the EC2 Amazon Cloud servers. Images have been created to run ISSM, which greatly improves the ease of use/installation provided users have an Amazon account.

6.3 Introduction

We rely on the Amazon Elastic Compute Cloud solution (EC2) to run ISSM on the amazon cloud. We provide everything required in the `externalpackages` directory and this documentation to run ISSM, provided an Amazon EC2 account has already been setup by the user. EC2 is the backbone for our cloud computing, however, it does not provide facilities to deploy an MPI cluster on an instance of the cloud. To deploy such a cluster, we rely on the third party software, StarCluster. This library is a suite of python scripts which communicate with the EC2 servers, and automatically setup a cluster, given a configuration file, using the EC2 account of the user. Once the cluster is spun-up, it can be used to carry out computations using the ISSM framework. To facilitate the install of ISSM we provide images (AMI), which can be loaded directly and hence avoiding the need for extensive installation/compilation. Such images are now private, but could potentially be made available to the ISSM community of users in the near-future.

6.4 Installation

We assume here that you have setup an EC2 account, and that you have the following: the AWS access key id, and the AWS secret access key. Nothing else is needed on the EC2 side.

You need to also install StarCluster. First, be sure that you have successfully installed python from the ISSM `externalpackages` directory, otherwise, you will get permission issues upon install of the StarCluster package. Then just run the `install.sh` script in the StarCluster external package of `issm/trunk`. This should install all the scripts necessary to run StarCluster successfully.

6.5 Configuration

StarCluster needs a configuration file, which will be used to store information proper to your EC2 account, and templates for the clusters you will be spinning up. This file is well described on the StarCluster website, found here: StarCluster user manual. For ease of use, we have created an ISSM StarCluster configuration file that you will find in the `issm-jpl/proj-group/CloudComputing` directory. In order to use this configuration file, you should create an alias in your local settings:

```
alias st='starcluster -c $PATH_TO_CONFIG_FILE '
```

For the ISSM team, the StarCluster configuration file should be replaced by: `issm-jpl/proj-group/CloudComputing/starcluster.c`

6.5.1 StarCluster configuration file for ISSM

Here are some of the sections of the `starcluster.config` file, which we explain within the framework of the ISSM runs:

```
[aws info]
AWS_ACCESS_KEY_ID = put_you_aws_access_key_id_here
AWS_SECRET_ACCESS_KEY = put_your_aws_secrate_access_key_here
AWS_USER_ID = put_your_ec2_acount_loging_here
AWS_REGION_NAME = us-east-1
AWS_REGION_HOST = ec2.us-east-1.amazonaws.com
```

This section holds your settings for the AWS account. You need to provide your access key id and secret access key, along with the user id for the account. The region name and region hosts determine on which region of the Amazon EC2 cloud you will be running your cluster. Beware, some of the most powerful machines are not always available in all regions. Typically, the `us-east-1` has the best instance types, the `cc2.8xlarge` machine. This is what we will rely on here.

```
[key issm-jpl]
KEY_LOCATION=~/.ssh/issm-jpl.rsa
```

This section specifies the locations where the ssh key for the EC2 account is located. Before you can start using the EC2 account with StarCluster, StarCluster needs to create this key. Run the following command to do so:

```
st createkey  issm --outputfile=issm.rsa
```

Once the key is created, move it to the `KEY\_LOCATION` specified in the section above. Here, we would move `issm.rsa` to the ssh directory in your home. If the key has already been created and you don't have it, just download it from your EC2 account console or, request it from someone who owns the account and can send you the rsa key. Do not attempt to destroy the key by running `st removekey issm.rsa`, as this will also deactivate all the other users that were using this key.

```
[cluster issm-jpl]
KEYNAME = issm-jpl
CLUSTER_SIZE = 1
CLUSTER_USER = username
CLUSTER_SHELL = bash
NODE_IMAGE_ID = ami-59106030 root
NODE_INSTANCE_TYPE = cc2.8xlarge
PLUGINS = createusers-jpl, mpich2
#SPOT_BID = 0.27

[plugin createusers-jpl]
setup_class = starcluster.plugins.users.CreateUsers
usernames =  username
download_keys = True

[plugin mpich2]
SETUP_CLASS = starcluster.plugins.mpich2.MPICH2Setup
```

These sections really describe your cluster settings. `CLUSTER\_SIZE` is the number of instances that will be launched for the cluster. A special EC2 cluster group will be created, ensuring the all the instances of your cluster have the best connectivity. The user of the cluster will have a corresponding user name created.

The `NODE\_INSTANCE\_TYPE` can be found here: EC2 instance types. The most powerful one for the purpose of running ISSM is the `cc2.8xlarge`, which can be found on the `us-east1` region.

The `NODE\_IMAGE\_ID` is the image (residing on the EC2 servers) from which the instances will be created. This image is a template for the cluster you will be launching. Here, we use ISSM images that have been created specifically. Please ask the ISSM team for an image if you don't want to create one from scratch.

`SPOT\_BID` is a setting that you can activate to make a spot request at a certain price. The EC2 cloud will provide you instances once the spot price goes below your requested price. This is a way to run low-cost solutions using the market driven EC2 prices.

The plugin sections ensure that the usernames are created and mpich2 is run as the backbone of the MPI cluster.

6.6 Running ISSM with StarCluster

In order to run ISSM on the EC2 cloud, you will need to create an instance of your cluster, by running the following:

```
st start -c issm-jpl issm1
```

This will spin-up an issm-jpl type cluster, named issm1. Once the cluster is created, you can log into it by doing the following:

```
st sshmaster issm1 -u username
```

You can run ISSM locally, or better, rely on the @cloud cluster class already implemented in ISSM. The only thing to do to run on the issm1 cluster is to activate the @cloud class in ISSM using the `'issm1'` name:

```
md.cluster=cloud('name','issm1','np',num_cpus_requested);
```

Upon a call to solve, ISSM will be launched on the issm1 cloud instance.

If you wish to receive an e-mail when the run is complete, a `.sge_request` file must be created in the home directory of the issm-jpl image (either locally on the instance after it is started or before the image itself is created). The `.sge_request` file should look like this:

```
-M your_email_address@your_domain.com
-m e
```

Once you are done running on the cluster, terminate the cluster by doing:

```
st terminate issm1
```

Chapter 7

Advanced features

7.1 Inverse methods

7.1.1 Introduction

Inversions are used to constrain poorly known model parameters such as basal friction. The method consists of finding a set of model inputs that minimizes the cost function $\mathcal{J}$ that measures the misfit between model and observations. For example, inverse methods are used to infer the basal friction k:

$$\boldsymbol{\tau}_b = -k^2 N^r \|\mathbf{v}\|^{s-1} \mathbf{v}_b \tag{7.1}$$

and/or the depth-averaged ice hardness, B, in Glen's flow law:

$$\mu = \frac{B}{2\left(\dot{\varepsilon}_e^{1-\frac{1}{n}}\right)} \tag{7.2}$$

This section explains how to launch an inverse method and how optimization parameters must be tuned.

7.1.2 Cost functions

7.1.2.1 Absolute misfit

This is the classic way of calculating a misfit between a modeled and observed velocity field:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2}\left(\left(v_x - v_x^{\text{obs}}\right)^2 + \left(v_y - v_y^{\text{obs}}\right)^2\right) dS \tag{7.3}$$

Where:

- v_x is the x component of the glacier modeled velocity
- v_y is the y component of the glacier modeled velocity
- $\text{v}_\text{x}^{\text{obs}}$ is the x component of the glacier observed velocity
- $\text{v}_\text{y}^{\text{obs}}$ is the y component of the glacier observed velocity

7.1.2.2 Relative misfit

The relative misfit is defined as follows:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2}\left(\frac{\left(v_x - v_x^{\text{obs}}\right)^2}{\left(v_x^{\text{obs}} + \varepsilon\right)^2} + \frac{\left(v_y - v_y^{\text{obs}}\right)^2}{\left(v_y^{\text{obs}} + \varepsilon\right)^2}\right) dS \tag{7.4}$$

Where:

- ε is a minimum velocity used to avoid the observed velocity being equal to zero.

7.1.2.3 Logarithmic misfit

$$\mathcal{J}(\mathbf{v}) = \int_S \left(\log\left(\frac{\|\mathbf{v}\| + \varepsilon}{\|\mathbf{v}^{\text{obs}}\| + \varepsilon} \right) \right)^2 dS \tag{7.5}$$

Where:

- v is the glacier modeled velocity magnitude
- v^{obs} is the glacier observed velocity magnitude
- ε is a minimum velocity used to avoid the observed velocity being equal to zero.

7.1.2.4 Thickness misfit

$$\mathcal{J}(H) = \int_\Omega \frac{1}{2} \left(H - H^{\text{obs}} \right)^2 d\Omega \tag{7.6}$$

Where:

- H is the ice thickness
- H^{obs} is the measured ice thickness

7.1.2.5 Drag gradient

$$\mathcal{J}(k) = \int_B \gamma \frac{1}{2} \|\nabla k\|^2 dB \tag{7.7}$$

Where:

- γ is a Tikhonov regularization parameter

7.1.2.6 Thickness gradient

$$\mathcal{J}(k) = \int_\Omega \gamma \frac{1}{2} \|\nabla H\|^2 d\Omega \tag{7.8}$$

Where:

- γ is a Tikhonov regularization parameter

7.1.3 Model parameters

The parameters relevant to the stress balance solution can be displayed by typing:

```
>> md.inversion
```

- `md.inversion.iscontrol`: 1 if inversion is activated, 0 for a forward run (default)
- `md.inversion.incomplete_adjoint`: 1 linear viscosity, 0 non-linear viscosity
- `md.inversion.control_parameters`: parameters that is inferred (ex: `{'FrictionCoefficient'}` or `{'MaterialsRheologyBbar'}`
- `md.inversion.cost_functions`: list of individual cost functions that are summed to calculate the final cost function $\mathcal{J}$ to be minimized (ex: `[101,501]`)
- `md.inversion.cost_functions_coefficients`: weight of each individual cost function previously defined for each vertex (more/no weight can be put on certain regions)

- `md.inversion.min_parameters`: minimum value for the inferred parameter
- `md.inversion.max_parameters`: maximum value for the inferred parameter
- `md.inversion.vx_obs`: x component of the surface velocity
- `md.inversion.vy_obs`: y component of the surface velocity
- `md.inversion.vel_obs`: surface velocity magnitude
- `md.inversion.thickness_obs`: measured ice thickness

7.1.4 Minimization algorithms

Depending on the class of `md.inversion`, several optimization algorithm are available:

- Brent search algorithm (`md.inversion=inversion()`, the default)
- Toolkit for Advanced Optimization (TAO) (`md.inversion=taoinversion()`)
- M1QN3 algorithm (`md.inversion=m1qn3inversion()`)

Each minimizer has its own optimization parameters described below.

7.1.4.1 Brent search minimizers

- `md.inversion.nsteps`: number of optimization searches (gradient evaluations)
- `md.inversion.maxiter_per_step`: maximum iterations during each optimization step
- `md.inversion.step_threshold`: decrease threshold for next step (default is 30%)
- `md.inversion.gradient_scaling`: scaling factor on gradient direction during optimization, for each optimization step

$$\alpha \in [0, \texttt{gradient\_scaling}] \qquad p^{\text{new}} = p^{\text{old}} - \alpha \, \nabla_p \mathcal{J} / \|\nabla_p \mathcal{J}\| \tag{7.9}$$

7.1.4.2 Toolkit for Advanced Optimization (TAO)

ISSM has an interface to the Toolkit for Advanced Optimization (TAO) [*Munson et al.*, 2012]. Here is a list of the relevant parameters:

- `md.inversion.maxsteps`: maximum number of iterations (gradient computation)
- `md.inversion.maxiter`: maximum number of Function evaluation (forward run)
- `md.inversion.algorithm`: inimization algorithm. ex: `'tao_blmvm'`, `'tao_cg'`, `'tao_lmvm'`
- `md.inversion.fatol`: cost function absolute convergence criterion (defined below)
- `md.inversion.frtol`: cost function relative convergence criterion (defined below)
- `md.inversion.gatol`: gradient absolute convergence criterion (defined below)
- `md.inversion.grtol`: gradient relative convergence criterion (defined below)
- `md.inversion.gttol`: gradient relative convergence criterion 2 (defined below)

with the following convergence criteria:

$$\begin{array}{lcl} f(X) - f(X^*) & < & \epsilon_{fatol} \\ |f(X) - f(X^*| \, / \, |f(X^*)| & < & \epsilon_{frtol} \\ \|g(X)\| & < & \epsilon_{gatol} \\ \|g(X)\| / \, |f(X)| & < & \epsilon_{grtol} \\ \|g(X)\| / \|g(X_0)\| & < & \epsilon_{gttol} \end{array} \tag{7.10}$$

where:

- $f(X)$ is the cost function at X
- $g(X)$ is the cost function gradient with respect to X
- X^* is the estimated "true" minimum
- X_0 is the initial guess

7.1.4.3 M1QN3

ISSM has an interface to M1QN3 (Inria) [*Gilbert and Lemaréchal*, 1989]. This interface was largely based on *Nardi et al.* [2009]. Here is a list of the relevant parameters:

- `md.inversion.maxsteps`: maximum number of iterations (gradient computation)
- `md.inversion.maxiter`: maximum number of Function evaluation (forward run)
- `md.inversion.dxmin`: convergence criterion: two points less than dxmin from eachother (sup-norm) are considered identical
- `md.inversion.gttol`: gradient relative convergence criterion 2 (defined below)

7.1.5 Running an inversion

To launch an inversion, run a stress balance solution with `md.inversion.iscontrol=1`:

```
>> md=solve(md,StressbalanceSolutionEnum);
```

7.2 Rifts

ISSM allows for the simulation of rifts. This section explains how to create a model that includes rifts, and how to control their behavior.

7.2.1 Rifts creation

Rifts can be included right between the phase where the mesh is created, and the phase where the geography is setup. Rifts that should be included in the model must be present in an ARGUS type file. Each rift should be represented by an open loop set of points. Infinite numbers of rifts can be included, provided they do not intersect with the domain outline, or any other rift. This point is particularly important as there are no checks on intersections at the meshing phase. For example, a file including two straight rifts could look like, `Rifts.exp`:

```
## Name:Rift1
## Icon:0
# Points Count  Value
2 1.000000
# X pos Y pos
0 0
50000  0

## Name:Rift2
## Icon:0
# Points Count  Value
2 1.000000
# X pos Y pos
0 10000
50000  10000
```

this file includes two horizontal rifts of 50 km long, separated by 10 km. In order to create a model with these rifts, one would do:

```
>> md=model;
>> md=triangle(md,'DomainOutline.exp','Rifts.exp',4000);
>> md=meshprocessrifts(md);
>> md=setmask(md,'Iceshelves.exp','Islands.exp');
>> etc ...
```

The rest of the process is similar. This will create a `rifts` structure in the model `md`. The `rifts` structure holds as many members as there are rifts in `Rifts.exp`. The key fields in the rifts structure are the `fill` and `friction`. Fill can be either 1 (for water), 2 (for air) and 3 (for ice). Fill determines the pressure on each flank of the rifts that is being applied. Friction is a coefficient between the shear stress exerted on the rift flanks and the differential tangential velocity between both flanks.

7.2.2 Rift tip refining

Rifts in a mesh will not modify the type of meshing occurring during the mesh phase. To impact the mesh, one can use the `riftstiprefine.m` routine. This routine will ensure that the rift tips are correctly refined, to take into account the tip stress singularity. Use of this routine is as follows:

```
>> md=model;
>> md=triangle(md,'DomainOutline.exp','Rifts.exp',4001);
>> md=rifttipsrefine(md,2000,30000);
>> md=meshprocessrifts(md);
>> md=setmask(md,'Iceshelves.exp','Islands.exp');
>> etc ...
```

the first argument is the model, the second argument the tip area resolution, and the third is the size of the circle around the tips where mesh refinement should occur.

7.2.3 Rifts in parameter file

The structure of rifts can be modified in any parameter file. We do not advise touching anything except the fill and friction for each one of the rifts in the structure. For example, inclusion of the following lines in the parameter file should be enough:

```
>> for i=1:md.numrifts,
>>    md.rifts.riftstruct(i).fill=WaterEnum() %include water in the rifts
>>    md.rifts.riftstruct(i).friction=10^11 %friction parameter sigma=10^11*dv_t
>> end
```

Of course, different frictions and fill could be applied, according to the physics being captured.

7.2.4 Solving for rifts

Rifts are only allowed when using MacAyeal type elements, in 2d meshes. For now, 3d meshes are not supported. Nothing is needed to take rifts into account in the solve phase. A simple:

```
>> md=solve(md,StressbalanceSolutionEnum);
```

will suffice. Bear in mind that rifts are handled using penalty methods to ensure that penetration of rift flanks does not occur. This can be very computationally expensive, as penalty methods tend to lead to zigzagging of contact. A stable set of constraints strategy has been implemented, which should guarantee convergence, but can be slow. Users should also try to minimize zigzagging by refining the mesh where needed. In case zigzagging becomes too intense, locking of the zigzagging penalties will occur, which ensures convergence, but which can lead to bad results in a physical sense. Detecting penalty locking should give users an idea on where to refine the mesh.

7.2.5 Rifts plotting

Rifts can be plotted using the following special plots:

```
>> plotmodel(md,'data','rifts','data','riftpenetration','data','riftvel','data','riftrelvel');
```

these three plots will give users a view of which parts of the rifts are opening, closing, at which relative speed, etc ...

7.2.6 Rifts when using Yams mesh adaptation

Rifts can be used in conjunction with the Yams mesh adaptation routine, by adding the `Rifts.exp` file defining rift contours to the `'riftoutline'` option of `yams`. For example:

```
>>md=yams(md,'domainoutline','DomainOutline.exp','riftoutline','Rifts.exp','velocities','vel.mat');
```

7.2.7 Adding rifts to an existing mesh

In case users want to use an existing mesh, rifts can still be added on. The format for the rifts file is in this case slightly different:

```
## Name:ContourAroundRift1
## Icon:0
# Points Count  Value
5 1
# X pos Y pos
-100 -100
50100 -100
50100 +100
-100 +100
-100 -100

## Name:Rift1
## Icon:0
# Points Count  Value
2 500
# X pos Y pos
0 0
50000  0

## Name:ContourAroundRift2
## Icon:0
# Points Count  Value
5 1
# X pos Y pos
-100 900
50100 900
50100 1100
-100 1100
-100 900

## Name:Rift2
## Icon:0
# Points Count  Value
2 1000
# X pos Y pos
0 10000
50000  10000
```

The format is made of pairs of rift contours with the corresponding rift profile. The rift contour is a closed contour that envelopes the rift. The rift that follows needs to be completely included in it. The rift density

(here, 500 and 1000 respectively) is very important, as it will decide the density of the mesh around the rift. Do not specify 1, as this will try to include a rift in the mesh with a 1 m mesh density, which will probably result in a memory exhaustion problem for the local machine running ISSM.

7.3 Quantifications of Margins and Uncertainties with Dakota

7.3.1 Physical basis

The methods for Quantification of Margins and Uncertainties (QMU) are based on the Design Analysis Kit for Optimization and Terascale Applications (DAKOTA) software [*Eldred et al.*, 2008], which is embedded within ISSM [*Larour et al.*, 2012a, b]. Available DAKOTA analyses include sensitivity and sampling analyses, which we respectfully rely on to: 1) understand the sensitivity of model diagnostics to local variations in model fields and 2) identify how variations in model fields impact uncertainty in model diagnostics. Diagnostics of interest include ice volume, maximum velocity, and mass flux across user-specified profiles.

7.3.1.1 Mesh Partitioning

QMU analyses are carried out on partitions of the model domain. Each partition consists of a collection of vertices. The ISSM partitioner is versatile. For example, the partitioner can assign one vertex for each partition (linear partitioning); the same number of vertices per partition (un-weighted partitioning); or it can weight partitions by a specified amount (equal-area by default - to remove area-specific dependencies). Advanced partitioning is accomplished using the CHACO Software for Partitioning Graphs [*Hendrickson and Leland*, 1995], prior to setting up the model parameters for QMU analysis.

7.3.1.2 Sensitivity

Sensitivity, or local reliability, analysis computes the local derivative of diagnostics with respect to model inputs. It is used to assess the spatial distribution of this derivative, for the purpose of spatially ranking the influence of various inputs.

Given a response r that is a function of multiple variables x_i in a local reliability analysis *Coleman and Steele* [1999], we have:

$$r = r(x_1, x_2, ..., x_n) \tag{7.11}$$

where the sensitivities are defined as:

$$\theta_i = \frac{\delta r}{\delta x_i} \tag{7.12}$$

If each of the variables is independent, the error propagation equation defines the variance of r as:

$$\sigma_r^2 = \sum_{i=1}^{n} \theta_i^2 \sigma_i^2 \tag{7.13}$$

where σ_i is the standard deviation of x_i and σ_r is the standard deviation of r.

Importance factors for each x_i are determined by dividing the error propagation equation by ${\sigma_r}^2$. Note that the mean of the response is taken to be the response for the nominal value of each variable x_i.

Sensitivities are computed from the function evaluations using finite differences. The finite difference step size is user-defined by a parameter in the ISSM model. This analysis imposes the finite-difference step size as a small perturbation to x_i. The resulting sensitivities quantify how the location of errors impact a specified model diagnostic (r).

First, DAKOTA calls one ISSM model solve for an un-perturbed control simulation. Then, for every x_i, ISSM perturbs each partition one at a time, and calls an ISSM solve for each. At every partition, p, a resulting sensitivity, $\theta_i(p)$ is assigned. Each θ_i (defined above) is dependent on how much the outcome diverges from

the control run. The result is a spatial mapping of sensitivities and importance factors of r for every x_i. For Transient solves, sensitivities are determined only at the completion of a forward run.

Method inputs: σ_i for each x_i at every partition and the finite difference step

Method outputs: sensitivities (θ_i) and importance factors for each x_i at every partition

7.3.1.3 Sampling

Sampling analysis quantifies how input errors propagate through a model to impact a specified diagnostic, r. It a Monte-Carlo-style method that relies upon repeated execution (samples) of the same model, where input variables are perturbed by different amounts at each partition for each individual run. Resulting statistics (mean, standard deviations, cumulative distribution functions) are calculated after the specified number of samples are run.

For a particular sample, every x_i is perturbed by a different amount at each partition. Input values are perturbed randomly, per partition, within a prescribed range (described by a statistical distribution, e.g. normal or uniform). Once the variables are perturbed, the ISSM model solve is called.

Distributions: A normal distribution for a particular partition is fully described by an average, μ_i, and a standard deviation, σ_i. By definition, normal distributions cluster around μ_i and decrease towards the tails, in a Gaussian bell curve ranging from $\mu_i \pm 3\sigma_i$. A uniform distribution places greater emphasis on values closer to the tails, where probability of occurrence is equal for any given value within a specified minimum and maximum value.

If a user chooses so, any x_i can be treated as a scaled value. In this case, the distribution definitions are given in percentages, relative to a μ_i value of 1.

For example, at the beginning of a particular sample for a scaled x_i, DAKOTA chooses a random percentage perturbation $P_i(p)$ at each partition p. The value of the random percentage will fall within the defined error distribution, and the new value of x_i for duration of this sample run is perturbed by $x_i P_i(p)$. The generation algorithm for $P_i(p)$ is user-specified (e.g. Monte-Carlo or LHS [*Swiler and Wyss*, 2004]).

In the case where the user wants to sample n variables at the same time, a $P_i(p)$ is chosen separately for each x_i before a particular sample run. Resulting statistics reflect the combined effects of the errors due to $x_1, x_2, ..., x_n$.

For Transient simulations, $P_i(p)$ remains constant for the duration of a particular sample run. Note that statistics are determined only at the completion of each forward run.

Method inputs: The number of samples to be run and for every x_i, a definition of error distribution (error ranges may vary spatially by partition)

Method outputs: For r, mean, standard deviations, and cumulative distribution functions resulting from errors due to $x_1, x_2, ..., x_n$

7.3.2 Model parameters

The parameters relevant to uncertainty quantification can be displayed by typing:

```
>> md.qmu
```

- `md.qmu.isdakota`: 1 to activate qmu analysis, or else 0
- `md.qmu.variables`: arrays of each `variable` class
- `md.qmu.responses`: arrays of each `diagnostics` class
- `md.qmu.numberofresponses`: number of responses
- `md.qmu.params`: array of method-independent parameters
- `md.qmu.results`: holder class for information from dakota result files

- `md.qmu.partition`: user provided, the partition each vertex belongs to
- `md.qmu.numberofpartitions`: number of partitions
- `md.qmu.variabledescriptors`: list of user-defined descriptors for variables
- `md.qmu.responsedescriptors`: list of user-defined descriptors for diagnostics
- `md.qmu.method`: array of `dakota_method` class
- `md.qmu.mass_flux_profile_directory`: directory for mass flux profiles
- `md.qmu.mass_flux_profiles`: list of `mass_flux` profiles
- `md.qmu.mass_flux_segments`: used by `process_qmu_response_data` to store processed profiles
- `md.qmu.adjacency`: adjacency matrix from connectivity table, partitioner computes it by default
- `md.qmu.vertex_weight`: weight for each vertex, partitioner sets it from connectivity by default

7.3.3 Building the CHACO and DAKOTA packages

In order to run DAKOTA with ISSM, you must compile and install the external package dakota. In addition, for complex partitioning (more than one vertex per partition), you must compile and install the external package CHACO.

In addition, your configure script should include the following:

```
--with-chaco-dir=$ISSM_DIR/externalpackages/chaco/install \
--with-dakota-dir=$ISSM_DIR/externalpackages/dakota/install \
```

More recent versions of DAKOTA also require the external package boost. If installed, it should also be added to your configure script:

```
--with-boost-dir=$ISSM_DIR/externalpackages/boost/install/ \
```

7.3.4 Partitioning a Mesh

To partition your mesh using chaco, use the following commands:

```
>>md.qmu.numberofpartitions=1000; % Note: chaco can crash if too large
>>md=partitioner(md,'package','chaco','npart',md.qmu.numberofpartitions,'weighting','on');
%weighting on for weighted partitioning (equal-area by default), off for equal vertex partitioning
>>md.qmu.partition=md.qmu.partition-1; %With chaco, partition numbers must be adjusted by 1
```

OR, for a 1-to-1 mapping of vertices to partitions:

```
>>md.qmu.numberofpartitions=md.mesh.number_of_vertices;
>>md=partitioner(md,'package','linear');
```

7.3.5 Setting up the QMU

7.3.5.1 For sensitivity

```
>>md.qmu.method=dakota_method('nond_l');
```

This sets the method to local reliability (sensitivity). Other sensitivity settings:

```
>>md.qmu.params.fd_gradient_step_size='0.1'; %finite difference step size, 0.001 by
default
```

7.3.5.2 For sampling

```
>>md.qmu.method=dakota_method('nond_samp');
>>md.qmu.method(end)=...
dmeth_params_set(md.qmu.method(end),'seed',1234,'samples',500,'sample_type','lhs');
```

Where `'seed'` is used for reproducibility of results and `'samples'` is the number of samples requested. Other sampling settings:

```
>>md.qmu.params.tabular_graphics_data=true; %Output all the information needed to create
histograms of results
```

7.3.5.3 Other simple default settings for both sampling and sensitivity

```
>>md.qmu.params.evaluation_concurrency=1;
>>md.qmu.params.analysis_driver='';
>>md.qmu.params.analysis_components='';
>>md.qmu.params.direct=true;
```

7.3.6 Setting your QMU variables

Example: Here, the input of interest is md.friction.coefficient, scaled, with error defined as a normal distribution with a mean of 1 and a standard deviation of 10%.

```
>>md.qmu.variables.drag_coefficient=normal_uncertain('scaled_FrictionCoefficient',1,0.1);
```

This sets the standard deviation to a constant value at every partition. After it is initialized as above, the standard deviation can be set manually, so that it varies spatially by partition:

```
md.qmu.variables.drag_coefficient.stddev=uncertainty_on_partition;
```

See also:

```
>>help normal_uncertain
>>help uniform_uncertain
>>help AreaAverageOntoPartition
```

7.3.7 Setting your diagnostics

Example: Here, diagnostics of interest are (1) maximum velocity and (2) mass flux through two different gates. Mass flux gates are defined by the ARGUS files `'../Exp/MassFlux1.exp'` and `'../Exp/MassFlux2.exp'`.

```
%responses
md.qmu.responses.MaxVel=response_function('MaxVel',[],[0.01 0.25 0.5 0.75 0.99]);
md.qmu.responses.MassFlux1=response_function('indexed_MassFlux_1',[],[0.01 0.25 0.5 0.75 0.99]);
md.qmu.responses.MassFlux2=response_function('indexed_MassFlux_2',[],[0.01 0.25 0.5 0.75 0.99]);

%mass flux profiles
md.qmu.mass_flux_profiles={'../Exp/MassFlux1.exp','../Exp/MassFlux2.exp'};
md.qmu.mass_flux_profile_directory=pwd;
```

For more options see:

```
>>help response_function
```

7.3.8 Running a simulation

Note: You must set your stress balance tolerance to 10^{-5} or smaller in order to avoid the accumulation of numerical residuals between consecutive samples.

```
>>md.stressbalance.restol=10^-5;
```

To initiate the analysis of choice, use the following commands:

```
>> md.qmu.isdakota=1;
>> md=solve(md,MasstransportSolutionEnum);
```

The first argument is the model, the second is the nature of the simulation one wants to run.

Chapter 8

Plotting

8.1 MATLAB plots

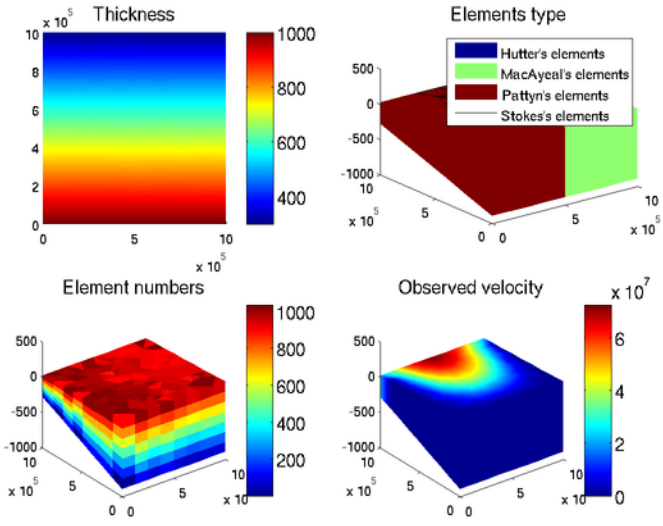
8.1.1 plotmodel

`plotmodel` takes the model `md` as first argument and then an even number of options (as in the function `setelementstype`, or `solve`). To plot a given field, use the option `'data'` followed by the field one wants to plot. For the thickness:

```
>> plotmodel(md,'data',md.geometry.thickness)
```

You can plot several fields at the same time but you have to add the argument `'data'` before each field you want to plot:

```
>> plotmodel(md,'data',md.geometry.thickness,'data','mesh','data',[1:md.mesh.numberofelements])
```



This can work for any field of length `md.mesh.numberofelements` or `md.mesh.numberofvertices`.

8.1.2 Options

Options in `plotmodel` come as pairs: the option name must be followed by its value. For example, if one wants to remove the color bar, the option name is `'colorbar'` and the value `0`:

```
>> plotmodel(md,'data',md.initialization.vel,'colorbar',0)
```

any options (except `'data'`) can be followed by `'#i'` where `i` is the subplot number, or `'#all'` if applied to all plots. Example:

```
>> plotmodel(md,'data',md.initialization.vel,'data','mesh','view#2',3,'colorbar#all','on','axis#1','off equal')
```

8.1.2.1 axis

Same as as standard axis MATLAB option

```
>> plotmodel(md,'data',md.vel,'axis','tight')
```

8.1.2.2 view

Same as as standard view MATLAB option

```
>> plotmodel(md,'data',md.vel,'view',2)
```

8.1.2.3 xlim, ylim, zlim

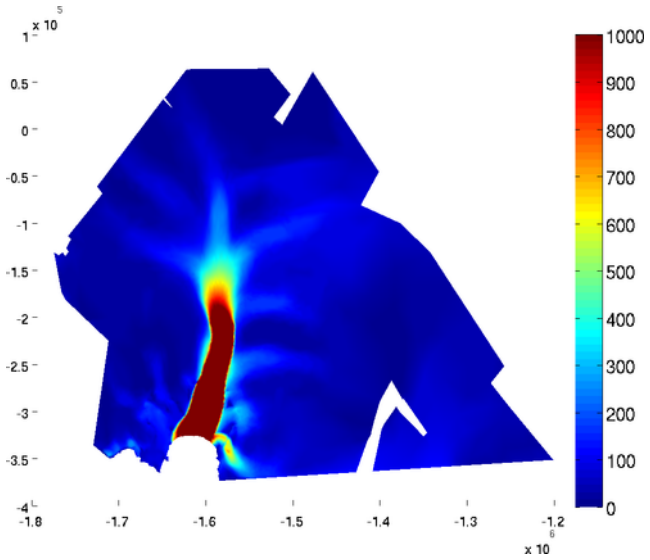
Same as as standard xlim MATLAB option

```
>> plotmodel(md,'data',md.vel,'xlim',[10^5 2*10^5])
```

8.1.2.4 caxis

Same as as standard caxis matlab option (control the extreme values of the colorbar).

```
>> plotmodel(md,'data',md.vel,'caxis',[0 1000])
```



8.1.2.5 colorbar

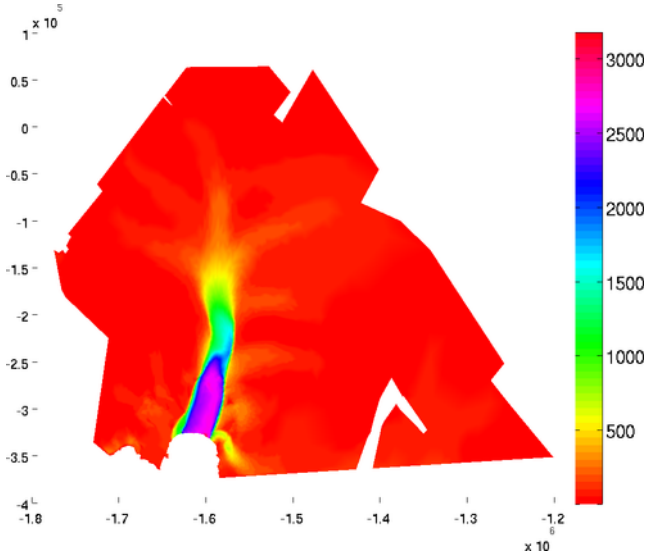
This option is used to control the colorbar display: `'on'` or `'off'`.

```
>> plotmodel(md,'data',md.vel,'colorbar','off')
```

8.1.2.6 colormap

Same as as standard colormap matlab option (control the extreme values of the colorbar).

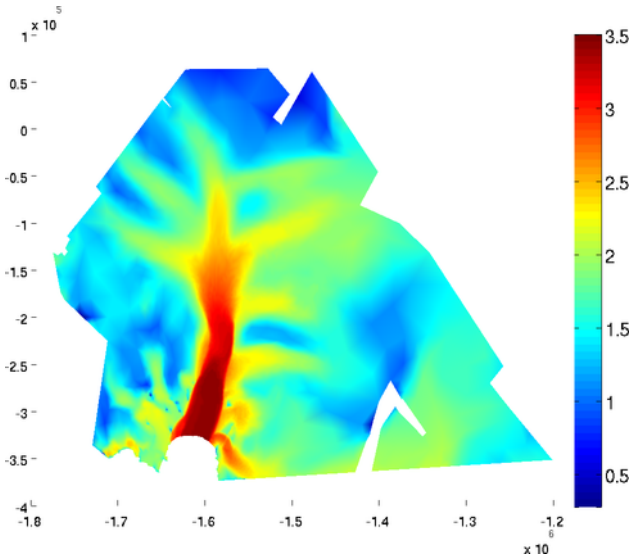
```
>> plotmodel(md,'data',md.vel,'colormap','hsv')
```



8.1.2.7 log

To get a logarithmic colorbar, use the `'log'` option followed by `10` for a decimal logarithm.

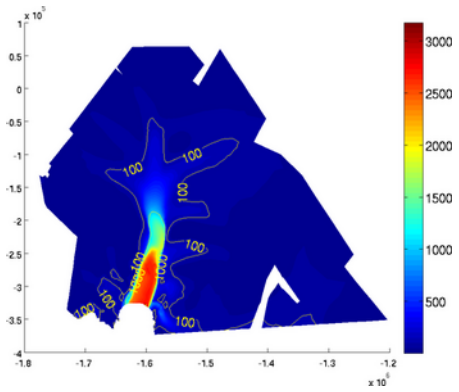
```
>> plotmodel(md,'data',md.vel,'log',10)
```



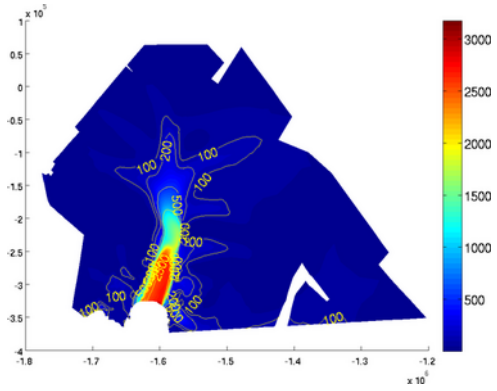
8.1.2.8 contourlevels

Contours of equi-value can be added to the plot by using the `'contourlevels'` option. The number of contours can be chosen by using the `'contourlevels'` options. The user can specify a number of levels or a cell containing the values of color changes (See examples below).

```
>> plotmodel(md,'data',md.vel,'contourlevels',3)
```



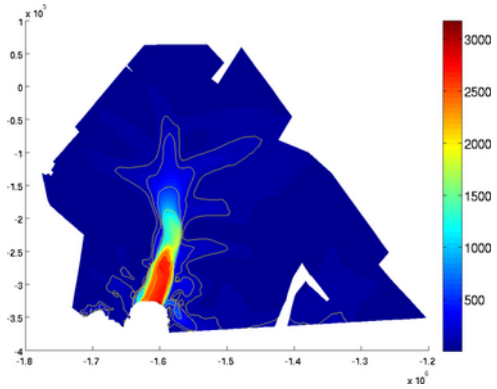

```
>> plotmodel(md,'data',md.vel,'contourlevels',{100,200,500,1000,2000,2500})
```



8.1.2.9 contourticks

If the user does not want to display the contour levels ticks, use the `'contourticks'` set as `'off'`:

```
>> plotmodel(md,'data',md.vel,'contourlevels',{100,200,500,1000,2000,2500},'contourticks','off')
```



8.1.2.10 contouronly

If the user wants to display the contours only, use the `'contouronly'` set as `'on'`.

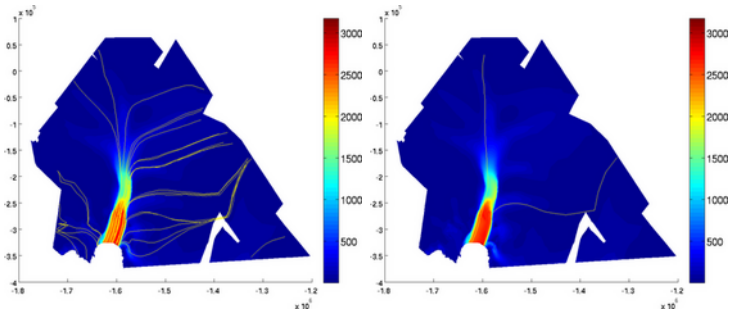
```
>> plotmodel(md,'data','vel','contourlevels',{100,200,500,1000,2000,2500},'contouronly','on')
```

8.1.2.11 streamlines

Streamlines can be displayed by using the `'streamlines'` option followed by a number of streamlines or a cell containing the coordinates of seed points:

```
>> plotmodel(md,'data',md.initialization.vel,'streamlines',50)
```

```
>> plotmodel(md,'data',md.initialization.vel,'streamlines',{10^6*[-1.45 -0.27],10^6*[-1.6 0]})
```

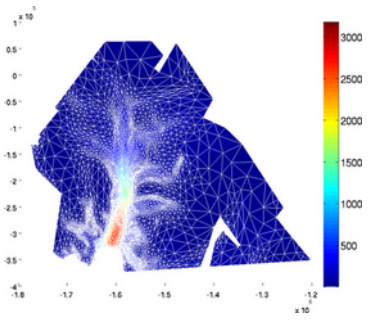


Note: streamlines use the velocities that are in `md.initialization`. Make sure you transfer the calculated velocities to `md.initialization` if you want to display the calculated streamlines.

8.1.2.12 edgecolor

The mesh can be superposed to the plot by using the `'edgecolor'` option followed by a color.

```
>> plotmodel(md,'data',md.initialization.vel,'edgecolor','w')
```



8.1.2.13 expdisp

Any ARGUS file can be displayed with the `'expdisp'` option followed by the name of the ARGUS file.

```
>> plotmodel(md,'data',md.initialization.vel,'expdisp','Iceshelves.exp')
```

8.1.2.14 expstyle

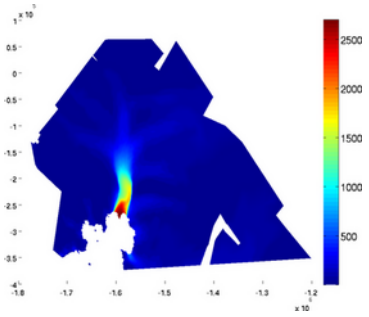
The style of the ARGUS profile can be controlled with the `'expstyle'` option, followed by the desired line style. Here is an example for a yellow dotted line:

```
>> plotmodel(md,'data',md.initialization.vel,'expdisp','Iceshelves.exp','expstyle','--y')
```

8.1.2.15 mask

If one does not want to display the value of the field on a mask only, use the `'mask'` option followed by a vector that holds 0 for the vertices whose values are hidden:

```
>> plotmodel(md,'data',md.initialization.vel,'mask',md.mask.groundedice_levelset<0)
```



8.1.2.16 northarrow

An arrow pointing North can be added with the `'northarrow'` option followed by `'on'`. The shape and position of the arrow can be controlled by using `[x0 y0 length [ratio [width]]]` instead of `'on'`.

```
>> plotmodel(md,'data',md.initialization.vel,'northarrow','on')
```

8.1.2.17 scaleruler

A scale ruler can be added. As for the North arrow, the default display is done by `'on'` but the shape and position of the scale ruler can be controlled by `[x0 y0 length width numberofticks]` where (x0,y0) are the coordinates of the lower left corner.

```
>> plotmodel(md,'data',md.initialization.vel,'scaleruler','on')
```

8.1.2.18 title

Same as as standard title MATLAB option

```
>> plotmodel(md,'data',md.vel,'title','Ice velocity [m/yr]')
```

8.1.2.19 fontsize

Same as as standard fontsize MATLAB option

```
>> plotmodel(md,'data',md.vel,'title','Ice velocity [m/yr]','fontsize',8)
```

8.1.2.20 fontweight

Same as as standard fontweight MATLAB option

```
>> plotmodel(md,'data',md.vel,'title','Ice velocity [m/yr]','fontweight','b')
```

8.1.2.21 xlabel, ylabel

Same as as standard xlabel MATLAB option

```
>> plotmodel(md,'data',md.vel,'xlabel','x axis [m]')
```

8.1.3 Special plots

8.1.3.1 basaldrag

The special plot `'basal_drag'` displays the norm of the basal drag friction in kPa following formula:

$$\boldsymbol{\tau}_b = -k^2 N^r \|\mathbf{v}\|^{s-1} \mathbf{v}_b \tag{8.1}$$

Basal drag relies on the velocity provided in `md.initialization`. The x and y components of the basal drag can be displayed with the `'basal_dragx'` or `'basal_dragy'` special plots:

```
>> plotmodel(md,'data','basal_drag')
```

```
>> plotmodel(md,'data','basal_dragx')
```

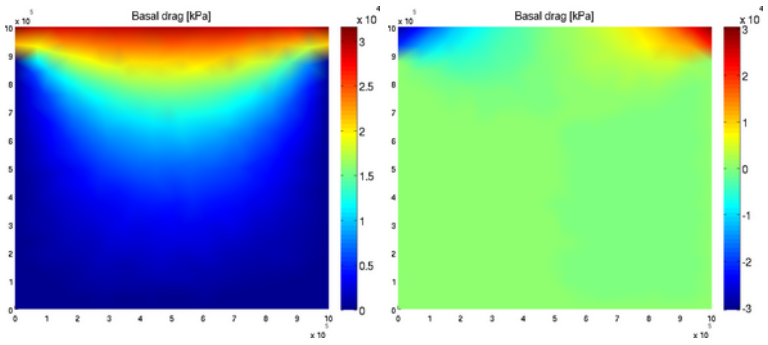
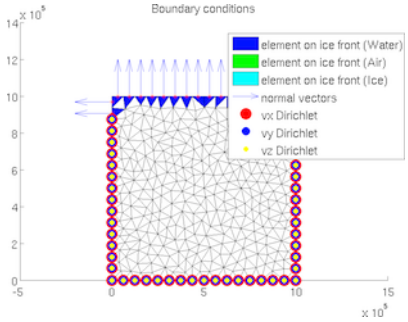


Figure 8.1: Basal friction norm and Basal friction x-component

8.1.3.2 BC

The special plot `'BC'` displays all boundary conditions (Newmann and Dirichlet) for 2d and 3d meshes.

```
>> plotmodel(md,'data','BC')
```

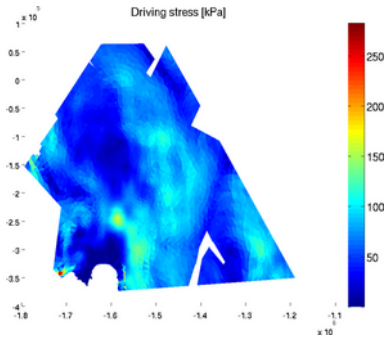


8.1.3.3 driving_stress

The special plot `'driving_stress'` displays the basal drag friction in kPa following formula:

$$\boldsymbol{\tau}_d = \rho g H \, \nabla s \tag{8.2}$$

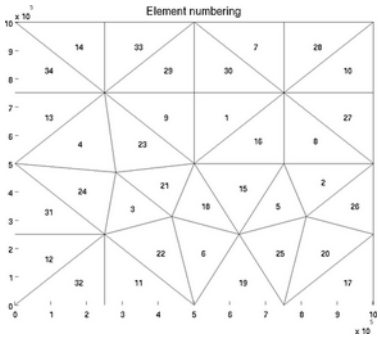
```
>> plotmodel(md,'data','driving_stress')
```



8.1.3.4 elementnumbering

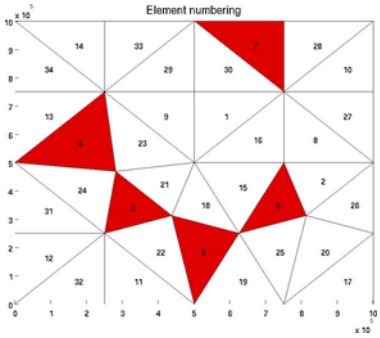
In the debugging process, it is often very useful to display all the elements next to their numbers. This is what the special plot `'elementnumbering'` does:

```
>> plotmodel(md,'data','elementnumbering')
```



A given list of elements can be highlighted with te `'highlight'` option:

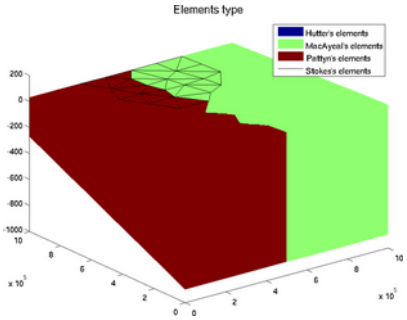
```
>> plotmodel(md,'data','elementnumbering','highlight',[3 4 5 6 7])
```



8.1.3.5 elements_type

The special plot `'elements_type'` displays the elements with a specific color for each formulation.

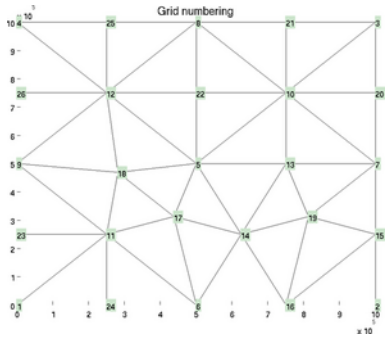
```
>> plotmodel(md,'data','elements_type')
```



8.1.3.6 vertexnumbering

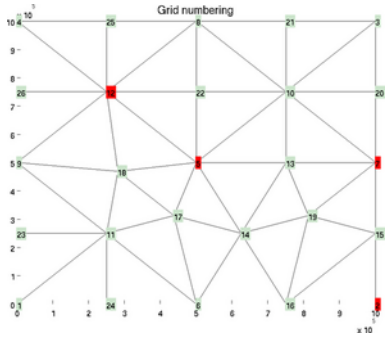
In the debugging process, it is often very useful to display all the vertices next to their numbers. This is what the special plot `'vertexnumbering'` does:

```
>> plotmodel(md,'data','vertexnumbering')
```



A given list of vertices can be highlighted with the `'highlight'` option:

```
>> plotmodel(md,'data','vertexnumbering','highlight',[2 5 7 12])
```

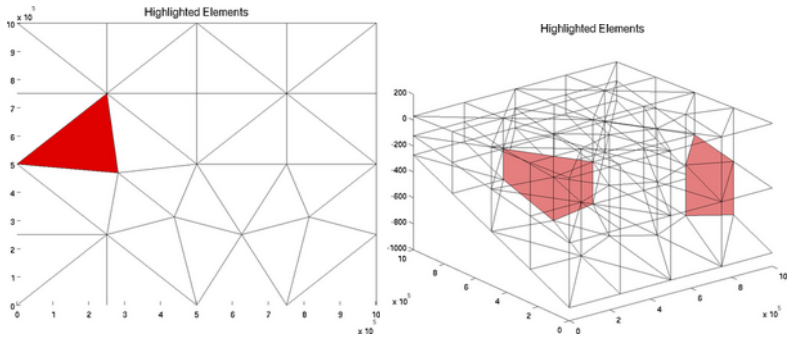


8.1.3.7 highlightelements

The special plot `'highlightelements'` is very similar to the plot `'elementnumbering'`. It is another possiblity to highlight one or several grids, but without indicating the number of all the elements. It is way faster for large models.

```
>> plotmodel(md,'data','highlightelements','highlight',5)
```

```
>> plotmodel(md,'data','highlightelements','highlight',[5 12])
```

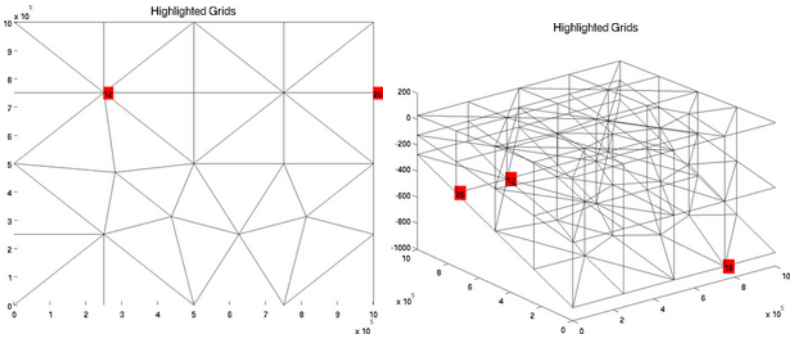


8.1.3.8 highlightgrids

The special plot `'highlightgrids'` is very similar to `'gridnumbering'`. It is another possibility to highlight grids without indicating all the grids numbers. It is way faster for big models.

```
>> plotmodel(md,'data','highlightgrids','highlight',[12 20])
```

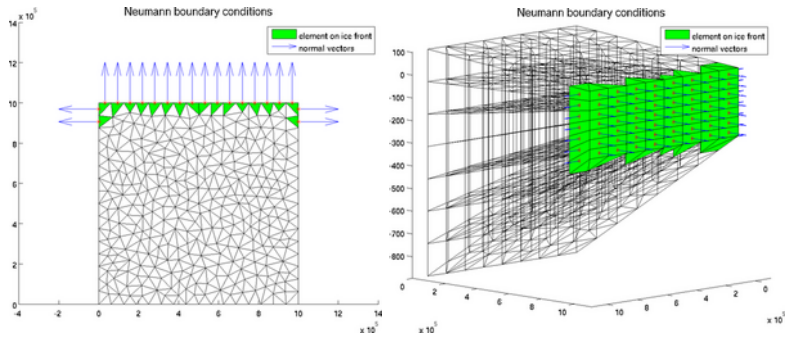
```
>> plotmodel(md,'data','highlightgrids','highlight',[12 16 26])
```



8.1.3.9 icefront

The special plot `'icefront'` displays the neumann boundary conditions, ie all the segments on ice front and the normal to these segments, for a 2d or 3d mesh.

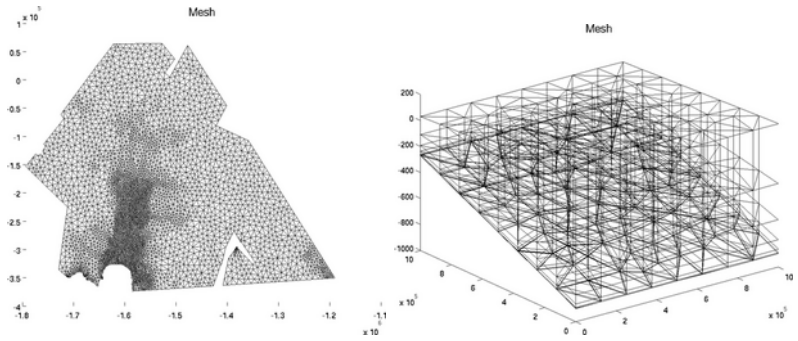
```
>> plotmodel(md,'data','icefront')
```



8.1.3.10 mesh

The special plot `'mesh'` displays the mesh of 2d or 3d model.

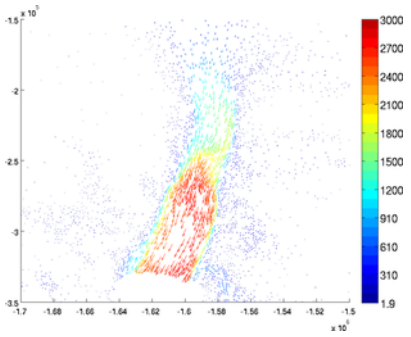
```
>> plotmodel(md,'data','mesh')
```



8.1.4 Quiver plot

For 2d or 3d fields, a generic color plot cannot be used (except component by component). The `'data'` used by the function `plotmodel` must be a matrix of 2 or 3 columns. For example:

```
>> plotmodel(md,'data',[md.vx md.vy])
```

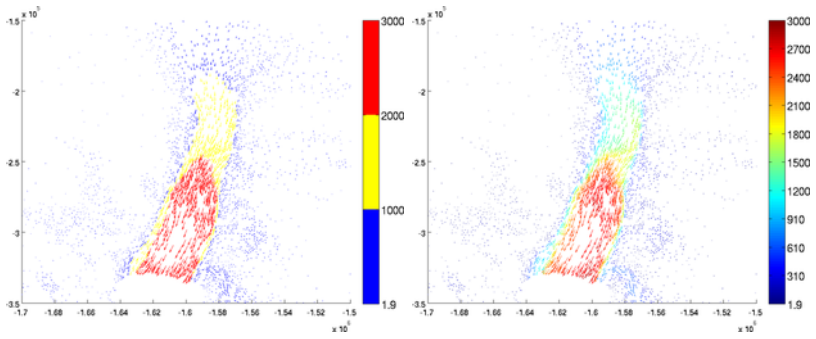


8.1.4.1 ColorLevels

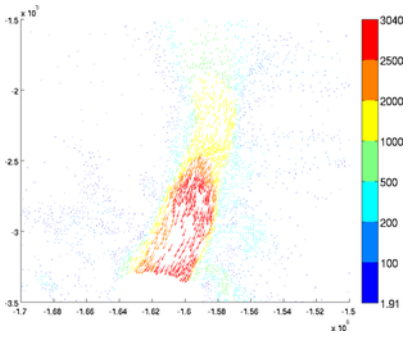
The number of colors can be chosen by using the `'colorlevels'` options. The user can specify a number of levels or a cell containing the values of color changes (See examples below).

```
>> plotmodel(md,'data',[md.vx md.vy],'colorlevels',3)
```

```
>> plotmodel(md,'data',[md.vx md.vy],'colorlevels',100)
```




```
>> plotmodel(md,'data',[md.vx md.vy],'colorlevels',{100,200,500,1000,2000,2500})
```

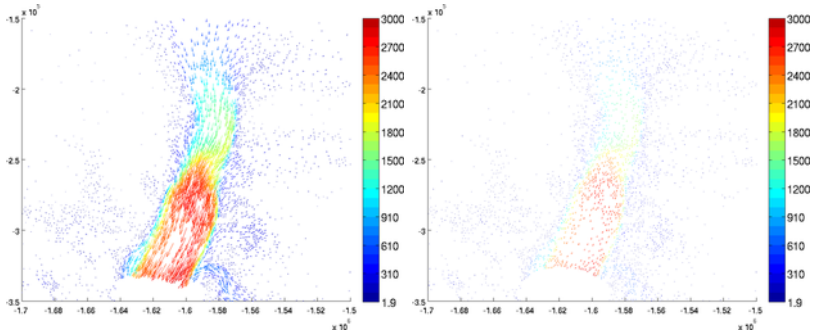


8.1.4.2 Scaling

The arrows length can be modified with the `'scaling'` options. The default value is 0.4. A higher scaling value will result in longer arrows.

```
>> plotmodel(md,'data',[md.vx md.vy],'scaling',1)
```

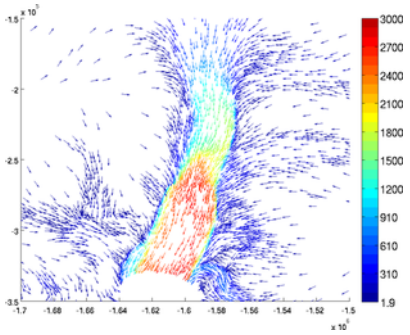
```
>> plotmodel(md,'data',[md.vx md.vy],'scaling',0.1)
```



8.1.4.3 Autoscale

If the user wants all the arrows to have the same length, use the option `'autoscale'` set as `'off'`.

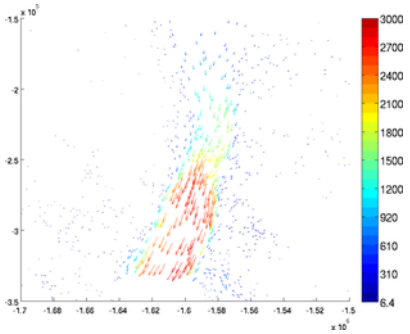
```
>> plotmodel(md,'data',[md.vx md.vy],'autoscale','off')
```



8.1.4.4 Density

The number of arrows can be reduced with the option `'density'`. If the density is set as 3, only one arrow out of 3 will be displayed. This option is very useful when the mesh is very refined.

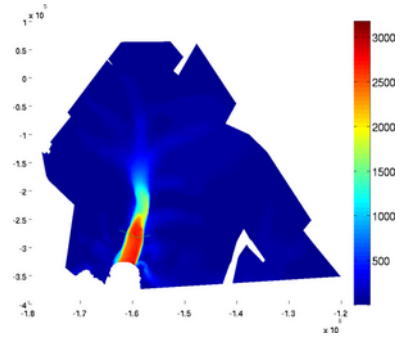
```
>> plotmodel(md,'data',[md.vx md.vy],'density',3)
```



8.1.5 Cross section

The section plot can be used to display the value of a field on a given track. The option `'sectionvalue'` must be followed by the name of an ARGUS file which contained the coordinates of the points describing the profile (this file can be generated by `exptool.m`). The resulting plot will be a curve in 2d and a colored surface in 3d (See example below).

```
>> plotmodel(md,'data',md.vel,'expdisp','track.exp')
```




```
>> plotmodel(md,'data',md.vel,'sectionvalue','track.exp')
```

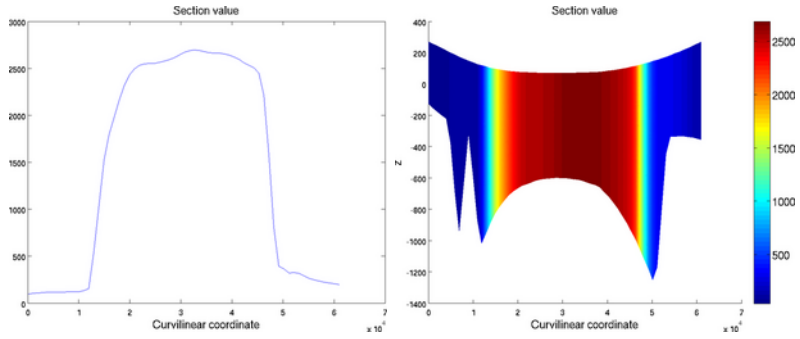


Figure 8.2: Section plot for 2D (left) and 3d (right) models

8.1.5.1 Resolution

The horizontal and vertical (in 3d) resolution can be specified by the `'resolution'` option. It must be a list with the horizontal resolution followed by the vertical resolution (in meters). When not specified, the default resolution is displayed.

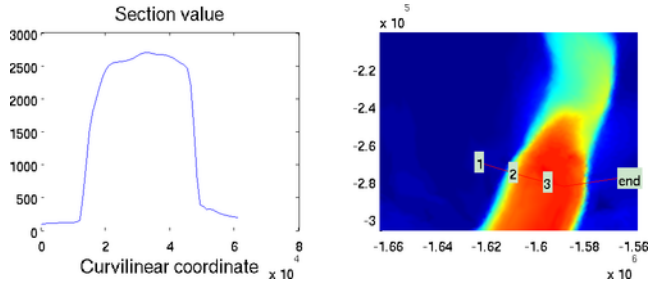
```
>> plotmodel(md,'data',md.vel,'sectionvalue','track.exp','resolution',[2*10^4 0])
```

```
>> plotmodel(md,'data',md.vel,'sectionvalue','track.exp','resolution',[10^3 0])
```

8.1.5.2 Show section

The profile used to create the section plot can be also plotted with the `'showsection'` option.

```
>> plotmodel(md,'data',md.vel,'showsection','on')
```



Chapter 9

Miscellaneous Tools

Several tools are available to help the user analyze the results and set up the models. These tools are just briefly mentioned below. Interested user can learn how to use these tools by typing `help function_name` in the MATLAB prompt for any of the following functions.

9.0.1 Mesh

- `triangle` generate a mesh from a domain outline
- `bamg` anisotropic mesh generation and adaptation
- `yams` anisotropic mesh adaptation
- `meshexprefine` refine a region of a mesh
- `meshprocessrift` process mesh when rifts are present
- `MeshQuality` compute mesh quality
- `rifttiprefine` refine mesh near rift tips

9.0.2 Model parameterization

- `extrude` vertically extrude a model
- `setmask` establish boundaries between grounded and floating ice
- `modelextract` extract the model over a subdomain
- `parameterize` model general parameterization
- `setflowequation` set stressbalance elements type
- `solversettoasm` set PETSc solver to ASM
- `solversettomumps` set PETSc solver to MUMPS
- `solversettosor` set PETSc solver to SOR
- `SetIceSheetBC` set ice sheet boundary conditions
- `SetIceShelfBC` set ice shelf boundary conditions
- `SetMarineIceSheetBC` set marine ice sheet boundary conditions

9.0.3 Mask

- `contourenvelope` create a list of segments enveloping an ARGUS contour
- `ContourToMesh` get elements and/or nodes inside an ARGUS contour
- `GetAreas` compute the area of each element
- `xy2ll` convert lat/lon to (x,y)
- `ll2xy` convert (x,y) to lat/lon
- `utm2ll` convert UTM to lat/lon

9.0.4 Interpolation

- `InterpFromGridToMesh` interpolation from a grid to a list of (x,y)
- `InterpFromMeshToGrid` interpolation from a 2d mesh to a grid
- `InterpFromMeshToMesh2d` interpolation from a 2d mesh to a list of (x,y)

9.0.5 ARGUS files

- `expcoarsen` coarsen or refine the resolution a contour
- `exptool` create and manage ARGUS files
- `expread` read an ARGUS file
- `expwrite` write an ARGUS file

9.0.6 Results analysis

- `averaging` data averaging over a mesh
- `basalstress` compute the basal stress
- `contourmassbalance` compute the mass balance of a contour
- `DepthAverage` depth averaging of a 3d field
- `drivingstress` compute the driving stress
- `flowlines` compute the coordinates of one or several flowlines
- `paterson` compute B from a temperature
- `project2d` project a 3d field on a layer
- `project3d` extrude a 2d field on every layer
- `SectionValues` compute the value of a field on a section or line
- `thicknessevolution` compute dh/dt

Chapter 10

FAQ

Here are a list of frequently asked questions:

10.0.1 Compilation troubleshooting

The installation of some external packages may require some tweaking depending on your platform. If you have compilation or configuration issues, try one of the following links:

- External Packages installation
  - PETSc 3.2
  - PETSc 3.1
  - PETSc 2.3.2
- ISSM configuration and compilation
- Python related issues
  - Python 2.7
  - Python 3.2

10.0.2 Using ISSM

We list here the questions related to the use of ISSM: the interface, error messages, etc.

- MATLAB's interface
- Debugging with valgrind
- MPICH error messages

10.0.3 Other

- svn tricks

10.1 PETSc 3.2

10.1.1 Error message in configure.log, when compiling downloaded PLAPACK:

```
icc -I...  /home1/...//externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h(29):
```

```
error: "MPI_COMPLEX" has already been declared in the current scope
MPI_Datatype MPI_COMPLEX;
             ^

icc -I...  /home1/...//externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h(36):
error: "MPI_COMPLEX" has already been declared in the current scope
MPI_Datatype MPI_DOUBLE_COMPLEX;
             ^
```

The only solution for now is to edit `$ISSM_DIR/externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h` and comment the lines:

```
//#ifndef MPI_COMPLEX
//   #if MANUFACTURE != SGI && ! (MANUFACTURE == CRAY && MACHINE_TYPE == CRAYPVP)
//      MPI_Datatype MPI_COMPLEX;
//      #define PLA_MPI_COMPLEX TRUE
//   #endif
//#endif
//
//#ifndef MPI_DOUBLE_COMPLEX
//   #if MANUFACTURE != SGI && ! (MANUFACTURE == CRAY && MACHINE_TYPE == CRAYPVP)
//      MPI_Datatype MPI_DOUBLE_COMPLEX;
//      #define PLA_MPI_DOUBLE_COMPLEX TRUE
//   #endif
//#endif
```

Then, recompile the PLAPACK package:

```
cd $ISSM_DIR/externalpackages/petsc/src/externalpackages/PLAPACKR32-hg
make all
```

and relaunch the PETSc's installation without downloading PLAPACK, but instead specifying the location of the compiled PLAPACK package in the `$PETSC_DIR/externalpackages/` directory:

```
--with-plapack-dir=$ISSM_DIR/externalpackages/petsc/src/$ISSM_ARCH
```

10.1.2 Running conftest on Pleiades

At the end of Petsc configuration, the following message appear:

```
================================================================================
Since your compute nodes require use of a batch system or mpiexec you must:
1) Submit ./conftest-linux-gnu-ia64-intel.py to 1 processor of your batch system or system you
are cross-compiling for; this will generate the file reconfigure.py
2) Run ./reconfigure-linux-gnu-ia64-intel.py.py (to complete the configure process).
================================================================================
```

The easiest way to do that is to create a file: `script.queue` with the following content:

```
#PBS -S /bin/bash
#PBS -q debug
#PBS -l select=1:ncpus=1:model=har
#PBS -l walltime=200
#PBS -W group_list=s1010
#PBS -m e
. /usr/share/modules/init/bash
```

```
module load comp-intel/11.1.046
module load mpi/mpt.1.25
module load math/intel_mkl_64_10.0.011
export PATH="$PATH:."
export MPI_GROUP_MAX=64
mpiexec -np 1 ./conftest-linux-gnu-ia64-intel.py
```

Then submit the job

```
qsub script.queue
```

Once the job is completed, check the file `./reconfigure-linux-gnu-ia64-intel.py.py` and continue the installation.

10.2 PETSc 3.1

10.2.1 Error message in configure.log, when compiling downloaded PLAPACK:

```
icc -I...  /home1/...//externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h(29):
error: "MPI_COMPLEX" has already been declared in the current scope
MPI_Datatype MPI_COMPLEX;
             ^

icc -I...  /home1/...//externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h(36):
error: "MPI_COMPLEX" has already been declared in the current scope
MPI_Datatype MPI_DOUBLE_COMPLEX;
             ^
```

The only solution for now is to edit `$ISSM_DIR/externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h` and comment the lines:

```
//#ifndef MPI_COMPLEX
//   #if MANUFACTURE != SGI && ! (MANUFACTURE == CRAY && MACHINE_TYPE == CRAYPVP)
//      MPI_Datatype MPI_COMPLEX;
//      #define PLA_MPI_COMPLEX TRUE
//   #endif
//#endif
//
//#ifndef MPI_DOUBLE_COMPLEX
//   #if MANUFACTURE != SGI && ! (MANUFACTURE == CRAY && MACHINE_TYPE == CRAYPVP)
//      MPI_Datatype MPI_DOUBLE_COMPLEX;
//      #define PLA_MPI_DOUBLE_COMPLEX TRUE
//   #endif
//#endif
```

Then, recompile the PLAPACK package:

```
cd $ISSM_DIR/externalpackages/petsc/src/externalpackages/PLAPACKR32-hg
make all
```

and relaunch the PETSc's installation without downloading PLAPACK, but instead specifying the location of the compiled PLAPACK package in the `$PETSC_DIR/externalpackages/` directory:

```
--with-plapack-dir=$ISSM_DIR/externalpackages/petsc/src/$ISSM_ARCH
```

10.2.2 Running conftest on Pleiades

At the end of Petsc configuration, the following message appear:

```
================================================================================
Since your compute nodes require use of a batch system or mpiexec you must:
1) Submit ./conftest to 1 processor of your batch system or system you are
cross-compiling for; this will generate the file reconfigure.py
2) Run ./reconfigure.py (to complete the configure process).
================================================================================
```

The easiest way to do that is to create a file: `script.queue` with the following content:

```
#PBS -S /bin/bash
#PBS -q debug
#PBS -l select=1:ncpus=1:model=har
#PBS -l walltime=200
#PBS -W group_list=s1010
#PBS -m e
. /usr/share/modules/init/bash
module load comp-intel/11.1.046
module load mpi/mpt.1.25
module load math/intel_mkl_64_10.0.011
export PATH="$PATH:."
export MPI_GROUP_MAX=64
mpiexec -np 1 ./conftest
```

Once the job is completed, check the file `reconfigure.py` and continue the installation.

10.2.3 Error message when compiling ISSM: _intel_fast_memcpy

```
if icpc -DHAVE_CONFIG_H -I ....".deps/libpISSM_a-BamgGeom.Tpo"; exit 1; fi
/home1/.../externalpackages/petsc/install/include/petscsys.h(1775):
error: identifier "_intel_fast_memcpy" is undefined
_intel_fast_memcpy((char*)(a),(char*)(b),n);
^

/home1/.../externalpackages/petsc/install/include/petscsys.h(1824):
error: identifier "_intel_fast_memset" is undefined
_intel_fast_memset((char*)a,0,n);
^
```

This is due to a problem in some versions of the Intel compilers. Comment all lines related to `PETSC_HAVE__INTEL_FAST*` in `$ISSM_DIR/externalpackages/petsc/install/include/petscconf.h`:

```
//#ifndef PETSC_HAVE__INTEL_FAST_MEMSET
//#define PETSC_HAVE__INTEL_FAST_MEMSET 1
//#endif

#ifndef PETSC_HAVE_TIME
#define PETSC_HAVE_TIME 1
#endif

//#ifndef PETSC_HAVE__INTEL_FAST_MEMCPY
//#define PETSC_HAVE__INTEL_FAST_MEMCPY 1
//#endif
```

This should fix ISSM's compilation.

10.3 PETSc 2.3.2

10.3.1 Error message in configure.log, when compiling downloaded MUMPS:

```
gfortran -arch x86_64 -m64 ... -WF,-Dpord -I. -I../include -c dmumps_bloc2.F
ar: creating archive libpord.a
cc1: error: unrecognized command line option "-WF,-Dpord"
make[1]: *** [dmumps_bloc2.o] Error 1
make: *** [double] Error 2
```

The only solution for now is to go into the `$PETSC_DIR/externalpackages/MUMPS_*/Makefile.inc` and comment the line:

```
#ORDERINGSF = -WF,-Dpord
```

Then, recompile the MUMPS package (don't forget to clean first: make clean, make all), and relaunch the Petsc installation without downloading MUMPS, but instead specifying the location of the compiled MUMPS package in the `$PETSC_DIR/externalpackages/` directory.

10.3.2 Error message in configure.log, when compiling downloaded PLAPACK:

```
icc -I...  /home1/...//externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h(29):
error: "MPI_COMPLEX" has already been declared in the current scope
MPI_Datatype MPI_COMPLEX;
             ^

icc -I...  /home1/...//externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h(36):
error: "MPI_COMPLEX" has already been declared in the current scope
MPI_Datatype MPI_DOUBLE_COMPLEX;
             ^
```

The only solution for now is to edit `$ISSM_DIR/externalpackages/petsc/src/linux-gnu-ia64-intel.py/include/PLA_Misc.h` and comment the lines:

```
//#ifndef MPI_COMPLEX
//   #if MANUFACTURE != SGI && ! (MANUFACTURE == CRAY && MACHINE_TYPE == CRAYPVP)
//      MPI_Datatype MPI_COMPLEX;
//      #define PLA_MPI_COMPLEX TRUE
//   #endif
//#endif
//
//#ifndef MPI_DOUBLE_COMPLEX
//   #if MANUFACTURE != SGI && ! (MANUFACTURE == CRAY && MACHINE_TYPE == CRAYPVP)
//      MPI_Datatype MPI_DOUBLE_COMPLEX;
//      #define PLA_MPI_DOUBLE_COMPLEX TRUE
//   #endif
//#endif
```

Then, recompile the PLAPACK package:

```
cd $ISSM_DIR/externalpackages/petsc/src/externalpackages/PLAPACKR32-hg
make all
```

and relaunch the PETSc's installation without downloading PLAPACK, but instead specifying the location of the compiled PLAPACK package in the `$PETSC_DIR/externalpackages/` directory:

```
--with-plapack-dir=$ISSM_DIR/externalpackages/petsc/src/$ISSM_ARCH
```

10.4 ISSM configuration and compilation

10.4.1 MPICH2 linking error

```
/trunk/externalpackages/mpich2/install/lib//libmpich.a(init.o):
In function ‘MPI_Init’:
init.c:(.text+0x48): undefined reference to ‘MPL_env2str’
init.c:(.text+0x93): undefined reference to ‘MPL_env2bool’
```

This error message indicates that you are using `mpich2` version higher than 1.3, and that `-lmpl` is missing in `--with-mpi-lib=`. Edit your ISSM configuration script (`configure.sh`) and change the corresponding line to:

```
--with-mpi-lib="-L$ISSM_DIR/externalpackages/mpich2/install/lib/ -lmpich -lmpl"
```

10.4.2 configure: error: Couldn't find mex... check your installation of matlab

This error message typically happens on Macs, because MATLAB does not support XCode7.0. MATLAB provides a patch available here.

If this fix does not work, you can do the following maneuver manually. In MATLAB, open the xml file that specifies where the SDK is located:

```
edit ([matlabroot '/bin/maci64/mexopts/clang++_maci64.xml'])
```

Toward the bottom, you will see a line that look like this, referencing "10.10":

```
<dirExists name="$$/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk" />
```

Copy and paste this line (immediately after the first) and change the second one to "10.11", like this:

```
<dirExists name="$$/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk" />
<dirExists name="$$/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.11.sdk" />
```

Do this against for another instance in this file, *both must be modified*. This should allow you to link against the OS X 10.11 SDK, but please be aware that this is configuration that has not been validated by MathWorks.

If you want to check that `mex` is now working, you can try to compile the following file:

```
#include <mex.h>
void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs[]){}
```

Let's call this file `conftest.cpp`. You can compile it using:

```
mex conftest.cpp
```

If it fails, your mex compiler is still not working, please contact MATLAB support and send them this file.

10.4.3 MATLAB MEX compile error

```
In file included from /Applications/MATLAB_R2013a.app/extern/include/mex.h:58:
In file included from /Applications/MATLAB_R2013a.app/extern/include/matrix.h:294:
/Applications/MATLAB_R2013a.app/extern/include/tmwtypes.h:819:9: error: unknown type name 'char16_t'
typedef char16_t CHAR16_T;
```

This happens after an upgrade of XCode, which also upgraded the C/C++ compiler, which MATLAB is not aware of. Add the following line to `$ISSM_DIR/configure.sh`

```
export CXXFLAGS=" -std=c++11"
```

Reconfigure and recompile.

10.4.4 X11 Library not found

```
g++: /usr/lib64/libX11.so: No such file or directory
```

This error message indicates that your X11 library is not located in `/usr/lib64/libX11.so`, which is the location that was provided in the ISSM configuration script (`configure.sh`). You need to find where this library is located and change the path in the configuration script:

```
--with-graphics-lib="/usr/lib/x86_64-linux-gnu/libX11.so"
```

10.4.5 *** No rule to make target

If you get the following error message:

```
make[3]: *** No rule to make target ‘objects/Gauss/GaussTria.cpp’, needed by
‘libISSMCore_a-GaussTria.o’.  Stop.
make[2]: *** [all-recursive] Error 1
make[1]: *** [all-recursive] Error 1
make: *** [all] Error 2
```

This is because you just updated ISSM and you need to clean the trunk:

```
$ cd $ISSM_DIR
$ make distclean
```

and then you will need to reconfigure:

```
$ cd $ISSM_DIR
$ ./scripts/automakererun.sh
```

ISSM can then be configured for the given OS. Several scripts exist to configure ISSM, they are all located in `$ISSM_DIR/configs`. Users should use the configuration script that is the closest to their environment. The configuration file must be copied in `$ISSM_DIR` and executed:

```
$ ./configure.sh
$ make
$ make install
```

10.4.6 Error message when compiling, with unresolved symbols in Petsc

The issue may lie with the externalpackage petsc, which you should compile with shared libraries enabled. Otherwise, at run time, on some platforms, python will not be able to resolve the Petsc symbols using statically compiled petscs libraries.

10.4.7 Linkage Error for drand48 and srand48 specifications in Windows

```
c:\issmuci\trunk-jpl\externalpackages\petsc\install\include\petscfix.h(12) : error
C2732: linkage specification contradicts earlier specification for 'drand48'
c:\issmuci\trunk-jpl\externalpackages\petsc\install\include\petscfix.h(12) : see declaration
of 'drand48'
c:\issmuci\trunk-jpl\externalpackages\petsc\install\include\petscfix.h(13) : error C2732:
linkage specification contradicts earlier specification for 'srand48'
c:\issmuci\trunk-jpl\externalpackages\petsc\install\include\petscfix.h(13) : see
declaration of 'srand48'
```

This error message indicates that petsc is not linking properly with your issm due to a C-specific definition in a header file. The petsc header file needs to be patched. Open `$ISSM_DIR/externalpackages/petsc/install/include/petscfix.h`, and edit the file by commenting out lines 10 and 14 like so:

```
10 //extern "C" {
11 int getdomainname(char *, int);
12 double drand48();
13 void   srand48(long);
14 //}
```

10.5 MATLAB's interface

MATLAB does not recognize any ISSM command

```
>> md=model;
??? Undefined function or variable 'model'.
```

This error message shows that ISSM tools have not been loaded by MATLAB. See this page for more info.

10.5.1 MATLAB complains about __gfortran_transfer_array_write symbol

In some cases, MATLAB complains about missing symbols in mex modules. That is due to the fact that MATLAB uses its own libraries that are not the ones you compiled the mex modules with. For example, you might have the following error message:

```
Invalid MEX-file '/Users/rtwalker/ISSM/trunk/lib/TriMesh.mexmaci64':
dlopen(/Users/rtwalker/ISSM/trunk/lib/TriMesh.mexmaci64, 6): Symbol not found:
__gfortran_transfer_array_write
```

This problem has been reported on macs. There are two ways to fix this problem:

10.5.1.1 Option 1 (preferred)

1. Locate where your `gfortran` library is (for example: `/usr/local/gfortran/lib/`).
2. copy MATLAB's `.matlab7rc.sh` in your home directory. For example:

   ```
   cp /Applications/MATLAB_R2014b.app/bin/.matlab7rc.sh ~/
   ```

3. open `~/.matlab7rc.sh` with your favorite editor, you will see a `"case"` with different architecture: `glnx86|glnxa64` for linux, `mac|maci|maci64` for mac and `*` for other architectures (windows etc). Go to the case that corresponds to your machine's architecture
4. uncomment the following line:

   ```
   #            LDPATH_PREFIX='$MATLAB/sys/opengl/lib/$ARCH'
   ```

   and change the path to reflect where your libgfortran.so is located (step 1). For example:

   ```
              LDPATH_PREFIX='/usr/local/gfortran/lib/'
   ```

Restart matlab and it should now work.

10.5.1.2 Option 2 (requires admin priviledges)

The second fix consists of replacing MATLAB's library with the one that are on your system, but you will need to have admin priviledges.

We show here the steps for the following MATLAB path: `/Applications/MATLAB_R2013a.app/` and `libgfortran` path: `/usr/local/gfortran/lib/`.

Before changing the libraries, make a backup:

```
cd /Applications/MATLAB_R2013a.app/sys/os/maci64/
mkdir OLD
mv libgfortran.* OLD
```

Then subsitute these libraries by the current ones used by gfortran (copy or symlink)

```
ln -s /usr/local/gfortran/lib/libgfortran.dylib .
ln -s /usr/local/gfortran/lib/libgfortran.3.dylib .
```

Contact us if the problem is not fixed.

10.5.2 MATLAB complains GLIBCXX libraries

In some cases, MATLAB complains about its own libraries. That is due to the fact that MATLAB uses its own libraries that might not be the ones you compiled the mex modules with. For example, you might have the following error message:

```
libstdc++.so.6: version 'GLIBCXX_3.4.9' not found
```

We found a fix on an Ubuntu forum that we copied here. The idea is to replace MATLAB's library with the one that was used to compile the mex modules, but you will need to have admin priviledges.

We show here the steps for the following MATLAB path: `/usr/local/matlab80/` and gcc libraries in: `/usr/lib`.

Before changing the libraries, make a backup:

```
cd /usr/local/matlab80/sys/os/glnxa64
mkdir OLD
mv libstdc++.so* OLD/
mv libgcc_s.so* OLD/
```

Then subsitute the last two by the current ones used by gcc (copy or symlink):

```
ln -s /usr/lib/libstd* .
ln -s /lib/libgcc_s.so* .
```

10.5.3 MATLAB complains about intel_fast_memm symbol

If you compile mex modules with intel compilers, MATLAB might complain about missing symbols. That is due to the fact that MATLAB uses its own `libirc.so` library that are not the ones you compiled the mex modules with. For example, you might have the following error message:

```
Invalid MEX-file '/users/username/test/issm/install/lib/IssmConfig.mexa64':
/users/username/test/issm/src/externalpackages/petsc/install/lib/libmetis.so: undefined symbol:
_intel_fast_memmove
```

Here is how you can fix this problem:

1. Locate where your `libirc.so` library is (for example: `/opt/share/intel/composer_xe_2013_sp1.3.174/compiler/lib/intel`
2. copy MATLAB's `.matlab7rc.sh` in your home directory. For example:

   ```
   cp /nasa/mw/2013b/bin/.matlab7rc.sh ~/
   ```

3. open `~/.matlab7rc.sh` with your favorite editor, you will see a `"case"` with different architecture: `glnx86|glnxa64` for linux, `mac|maci|maci64` for mac and `*` for other architectures (windows etc). Go to the case that corresponds to your machine's architecture
4. uncomment the following line:

   ```
   #           LDPATH_PREFIX='$MATLAB/sys/opengl/lib/$ARCH'
   ```

   and change the path to reflect where your libgfortran.so is located (step 1). For example:

   ```
   LDPATH_PREFIX='/opt/share/intel/composer_xe_2013_sp1.3.174/compiler/lib/intel64/'
   ```

Restart matlab and it should now work.

Contact us if the problem is not fixed.

MATLAB crashes unexpectedly

There are many causes that might make MATLAB crash. A possible cause is that PETSc is conflicting with java (this happens on some linux machines). The workaround is to use MATLAB in command line by deactivating the gui:

```
matlab -nojvm
```

Why can't I see what I am typing in the terminal after I exit MATLAB

This is a bug of MATLAB when running with `-nojvm` or `-nodesktop` flags under bash. The solution proposed by MathWorks consists of reseting the terminal after MATLAB exits by running `reset` command in the terminal window:

```
reset
```

The following message appears when I launch MATLAB:

```
Warning: Executing startup failed in matlabrc.
This indicates a potentially serious problem in your MATLAB setup,
which should be resolved as soon as possible.  Error detected was:
MATLAB:m_illegal_reserved_keyword_usage
Error: File: trunk/src/m/classes/qmu/normal_uncertain.m Line: 38
Column: 5
Illegal use of reserved keyword "end".
> In matlabrc at 220
```

This message indicates that your MATLAB version is too old (less than 7.6), and does not support MATLAB's new Class-Definition syntax. In this case, contact us, and we will help you convert all ISSM's MATLAB classes to the older syntax.

10.6 Debugging with valgrind

How to debug a crash in issm.exe?

If there is crash during the solve phase, we strongly advise to use valgrind. Install valgrind in the directory `$ISSM_DIR/externalpackages/valgrind`. valgrind is embedded in ISSM and can detect segmentation faults as well as memory leaks. Set the model debugging field to 1 and use only one cpu:

```
md.debug.valgrind=1;
md.cluster.np=1;
```

Launch the solution sequence and read the 'errlog' file.

How to debug a matlab crash?

If there is a crash that is not in `issm.exe` (sometimes shown as by PETSc's error manager), one should also use valgrind. Use the following command:

```
matlab -nojvm -nosplash -r "your matlab commands" -D"valgrind \
--error-limit=no --tool=memcheck -v --log-file=valgrind.log"
```

Valgrind's output file `valgrind.log` should help (look for Invalid read and Invalid write).

How to debug a python crash?

If there is a crash that is not in `issm.exe` (sometimes shown as by PETSc's error manager), one should also use valgrind. Use the following command:

```
valgrind  --error-limit=no --tool=memcheck -v --log-file=valgrind.log \
python -E -tt ./yourpythonscript.py
```

Valgrind's output file `valgrind.log` should help (look for Invalid read and Invalid write).

Note: if line numbers are not displayed for mac users, add the following option `--dsymutil=yes`

10.7 MPICH error messages

The following message appears in the errlog file when launching my job in parallel:

```
mpdrun_wilkes.jpl.nasa.gov: cannot connect to local mpd (/tmp/mpd2.console_name);
possible causes:
1. no mpd is running on this host
2. an mpd is running but was started without a "console" (-n option)
~
~
```

This message means that the MPI (Message Passing Interface) server, called mpd, is not running. Therefore, no parallel jobs can run on the cluster. To solve this issue, just type, at the command prompt on the server side (if for example your cluster has 8 cpus):

```
mpd --ncpus=8 &
```

This will launch the MPI server to manage 8 cpus on the cluster.

10.8 svn tricks

How do I control the list of files that are ignored by svn?

Go to the directory you are interested in and use the following command:

```
svn propedit svn:ignore .
```

You will get a list of all ignored files. You can modify this list and then commit.

Bibliography

Adhikari, S., E. Ivins, E. Larour, H. Seroussi, M. Morlighem, and S. Nowicki, Future antarctic bed topography and its implications for ice sheet dynamic, *Solid Earth*, *5*(1), 569–584, doi:{10.5194/se-5-569-2014}, 2014.

Aschwanden, A., E. Bueler, C. Khroulev, and H. Blatter, An enthalpy formulation for glaciers and ice sheets, *J. Glaciol.*, *58*(209), 441–457, doi:10.3189/2012JoG11J088, 2012.

Blatter, H., Velocity and stress-fields in grounded glaciers: A simple algorithm for including deviatoric stress gradients, *J. Glaciol.*, *41*(138), 333–344, 1995.

Borstad, C. P., A. Khazendar, E. Larour, M. Morlighem, E. Rignot, M. P. Schodlok, and H. Seroussi, A damage mechanics assessment of the Larsen B ice shelf prior to collapse: Toward a physically-based calving law, *Geophys. Res. Lett.*, *39*(L18502), 1–5, doi:10.1029/2012GL053317, 2012.

Borstad, C. P., E. Rignot, J. Mouginot, and M. P. Schodlok, Creep deformation and buttressing capacity of damaged ice shelves: theory and application to Larsen C ice shelf, *Cryosphere*, *7*, 1931–1947, doi:10.5194/tc-7-1931-2013, 2013.

Box, J. E., Greenland Ice Sheet Mass Balance Reconstruction. Part II: Surface Mass Balance (1840-2010), *J. Clim.*, *26*(18), 6974–6989, doi:10.1175/JCLI-D-12-00518.1, 2013.

Box, J. E., and W. Colgan, Greenland Ice Sheet Mass Balance Reconstruction. Part III: Marine Ice Loss and Total Mass Balance (1840-2010), *J. Clim.*, *26*(18), 6990–7002, doi:{10.1175/JCLI-D-12-00546.1}, 2013.

Box, J. E., et al., Greenland Ice Sheet Mass Balance Reconstruction. Part I: Net Snow Accumulation (1600-2009), *J. Clim.*, *26*(11), 3919–3934, doi:{10.1175/JCLI-D-12-00373.1}, 2013.

Coleman, W., and J. Steele, *Experimentation and Uncertainties Analysis for Engineers*, John Wiler, 1999.

Cuffey, K., and W. S. B. Paterson, *The Physics of Glaciers, 4th Edition*, Elsevier, Oxford, 2010.

de Fleurian, B., O. Gagliardini, T. Zwinger, G. Durand, E. Le Meur, D. Mair, and P. Råback, A double continuum hydrological model for glacier applications, *Cryosphere*, *8*(1), 137–153, doi:10.5194/tc-8-137-2014, 2014.

Durand, G., O. Gagliardini, T. Zwinger, E. Le Meur, and R. Hindmarsh, Full Stokes modeling of marine ice sheets: influence of the grid size, *Ann. Glaciol.*, *50*(52), 109–114, 2009.

Eldred, M. S., et al., DAKOTA, a Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis, Version 4.2 User's Manual, Technical Report SAND 2006-6337, *Tech. rep.*, Sandia National Laboratories, PO Box 5800, Albuquerque, NM 87185, 2008.

Ettema, J., M. R. van den Broeke, E. van Meijgaard, W. J. van de Berg, J. L. Bamber, J. E. Box, and R. C. Bales, Higher surface mass balance of the Greenland Ice Sheet revealed by high-resolution climate modeling, *Geophys. Res. Lett.*, *36*, 1–5, doi:10.1029/2009GL038110., 2009.

Franca, L., G. Hauke, and A. Masud, Revisiting stabilized finite element methods for the advective-diffusive equation, *Comput. Methods Appl. Mech. Engrg.*, *195*(13-16), 1560–1572, doi:10.1016/j.cma.2005.05.028, 2006.

Frey, P. J., Yams, a fully automatic adaptive isotropic surface remeshing procedure, *Tech. Rep. RT-0252*, INRIA, Rocquencourt, 2001.

Gilbert, J. C., and C. Lemaréchal, Some numerical experiments with variable-storage quasi-Newton algorithms, *Math. Program.*, *45*(1-3), 407–435, doi:10.1007/BF01589113, 1989.

Gladstone, R. M., V. Lee, A. Vieli, and A. J. Payne, Grounding line migration in an adaptive mesh ice sheet model, *J. Geophys. Res.*, *115*, 1–19, doi:10.1029/2009JF001615, 2010.

Glen, J., The creep of polycrystalline ice, *Proc. R. Soc. A*, *228*(1175), 519–538, 1955.

Hecht, F., BAMG: Bi-dimensional anisotropic mesh generator, *Tech. rep.*, FreeFem++, 2006.

Helsen, M. M., R. S. W. van de Wal, M. R. van den Broeke, W. J. van de Berg, and J. Oerlemans, Coupling of climate models and ice sheet models by surface mass balance gradients: application to the Greenland Ice Sheet, *Cryosphere*, *6*(2), 255–272, doi:10.5194/tc-6-255-2012, 2012.

Hendrickson, B., and R. Leland, The Chaco user's guide, version 2.0, Technical Report SAND-95-2344, *Tech. rep.*, Sandia National Laboratories, Albuquerque, NM 87185-1110, 1995.

Ivins, E. R., and T. S. James, Simple models for late Holocene and present-day Patagonian glacier fluctuations and predictions of a geodetically detectable isostatic response, *Geophys. J. Int.*, *138*(3), 601–624, doi:10.1046/j.1365-246x.1999.00899.x, 1999.

Larour, E., M. Morlighem, H. Seroussi, J. Schiermeier, and E. Rignot, Ice flow sensitivity to geothermal heat flux of Pine Island Glacier, Antarctica, *J. Geophys. Res. - Earth Surface*, *117*(F04023), 1–12, doi:{10.1029/2012JF002371}, 2012a.

Larour, E., J. Schiermeier, E. Rignot, H. Seroussi, M. Morlighem, and J. Paden, Sensitivity Analysis of Pine Island Glacier ice flow using ISSM and DAKOTA, *J. Geophys. Res.*, *117, F02009*, 1–16, doi:10.1029/2011JF002146, 2012b.

Le Brocq, A., A. Payne, M. Siegert, and R. Alley, A subglacial water-flow model for west antarctica, *J. Glaciol.*, *55(193)*, 879–888, 2009.

MacAyeal, D., A tutorial on the use of control methods in ice-sheet modeling, *J. Glaciol.*, *39*(131), 91–98, 1993.

MacAyeal, D. R., Large-scale ice flow over a viscous basal sediment: Theory and application to Ice Stream B, Antarctica, *J. Geophys. Res.*, *94*(B4), 4071–4087, 1989.

Morland, L., Unconfined ice shelf flow, *Proceedings of Workshop on the Dynamics of the West Antarctic Ice Sheet, University of Utrecht, May 1985. Published by Reidel*, *ed. C.J. van der Veen and J. Oerlemans*, 99–116, 1987.

Munson, T., J. Sarich, S. Wild, S. Benson, and L. C. McInnes, TAO 2.0 Users Manual, *Technical Memorandum ANL/MCS-TM-322*, Mathematics and Computer Science Division, Argonne National Laboratory, http://www.mcs.anl.gov/tao, 2012.

Nardi, L., C. Sorror, F. Badran, and S. Thiria, YAO: A Software for Variational Data Assimilation Using Numerical Models, in *LNCS 5593, Computational Science and Its Applications - ICCSA 2009*, edited by O. Gervasi, D. Taniar, B. Murgante, A. Laganà, Y. Mun, and M. L. Gavrilova, pp. 621–636, Springer-Verlag, 2009.

Nowicki, S., et al., Insights into spatial sensitivities of ice mass response to environmental change from the SeaRISE ice sheet modeling project II: Greenland, *J. Geophys. Res.*, *118*, 1–20, doi:10.1002/jgrf.20076, 2013.

Nowicki, S. M. J., and D. J. Wingham, Conditions for a steady ice sheet-ice shelf junction, *Earth Planet. Sci. Lett.*, *265*(1-2), 246–255, 2008.

Paterson, W., *The Physics of Glaciers*, 3rd ed., Pergamon Press, Oxford, London, New York, 1994.

Pattyn, F., A new three-dimensional higher-order thermomechanical ice sheet model: Basic sensitivity, ice stream development, and ice flow across subglacial lakes, *J. Geophys. Res.*, *108*(B8), 1–15, doi:10.1029/2002JB002329, 2003.

Pattyn, F., A. Huyghe, S. De Brabander, and B. De Smedt, Role of transition zones in marine ice sheet dynamics, *J. Geophys. Res. - Earth Surface*, *111*(F2), 1–10, doi:10.1029/2005JF000394, 2006.

Pfeffer, W., T. Illangasekare, and M. Meier, Analysis and modeling of melt-water refreezing in dry snow, *J. Glaciol.*, *36*(123), 238–246, 1990.

Pfeffer, W., M. Meier, and T. Illangasekare, Retention of Greenland Runoff by Refreezing: Implications for Projected Future Sea-Level Rise, *J. Geophys. Res. - Oceans*, *96*(C12), 22,117–22,124, doi:10.1029/91JC02502, 1991.

Schlegel, N.-J., E. Larour, H. Seroussi, M. Morlighem, and J. E. Box, Decadal-scale sensitivity of Northeast Greenland ice flow to errors in surface mass balance using ISSM, *J. Geophys. Res. - Earth Surface*, *118*, 1–14, doi:10.1002/jgrf.20062, 2013.

Schlegel, N.-J., E. Larour, H. Seroussi, M. Morlighem, and J. E. Box, Ice discharge uncertainties in Northeast Greenland from boundary conditions and climate forcing of an ice flow model, *J. Geophys. Res. - Earth Surface*, *120*(1), 29–54, 2015.

Seroussi, H., M. Morlighem, E. Larour, E. Rignot, and A. Khazendar, Hydrostatic grounding line parameterization in ice sheet models, *Cryosphere*, *8*(6), 2075–2087, doi:10.5194/tc-8-2075-2014, 2014a.

Seroussi, H., M. Morlighem, E. Rignot, J. Mouginot, E. Larour, M. P. Schodlok, and A. Khazendar, Sensitivity of the dynamics of Pine Island Glacier, West Antarctica, to climate forcing for the next 50 years, *Cryosphere*, *8*(5), 1699–1710, doi:10.5194/tc-8-1699-2014, 2014b.

Shewchuk, J. R., Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, in *Applied Computational Geometry: Towards Geometric Engineering*, *Lecture Notes in Computer Science*, vol. 1148, edited by M. C. Lin and D. Manocha, pp. 203–222, Springer-Verlag, from the First ACM Workshop on Applied Computational Geometry, 1996.

Shreve, R. L., Movement of water in glaciers, *J. Glaciol.*, *11*(62), 205–214, 1972.

Swiler, L. P., and G. D. Wyss, A User's Guide to Sandia's Latin Hypercube Sampling Software: LHS UNIX Library/Standalone Version, Technical Report SAND2004-2439, *Tech. rep.*, Sandia National Laboratories, PO Box 5800, Albuquerque, NM 87185-0847, 2004.

van Angelen, J. H., M. R. van den Broeke, and W. J. van de Berg, Momentum budget of the atmospheric boundary layer over the Greenland ice sheet and its surrounding seas, *J. Geophys. Res. - Atmospheres*, *116*, 1–14, doi:10.1029/2010JD015485, 2011.

van den Broeke, M., J. Bamber, J. Ettema, E. Rignot, E. Schrama, W. J. van de Berg, E. van Meijgaard, I. Velicogna, and B. Wouters, Partitioning Recent Greenland Mass Loss, *Science*, *326*(5955), 984–986, doi:10.1126/science.1178176, 2009.

van Meijgaard, E., L. H. van Ulft, W. J. Van de Berg, F. C. Bosvelt, B. J. J. M. Van den Hurk, G. Lenderink, and A. P. Siebesma, The knmi regional atmospheric model racmo version 2.1, technical report 302, *Tech. rep.*, KNMI, De Bilt, The Netherlands, 2008.

Weertman, J., On the sliding of glaciers, *J. Glaciol.*, *3*, 33–38, 1957.