

Ice-sheet and Sea-level System Model 2025 (4.24)

User Guide

Authors:

Mathieu Morlighem¹
Hélène Seroussi¹
Éric Larour²
Nicole Schlegel³
Surendra Adhikari²
Aleah Sommers¹
Felicity McCormack⁴

¹Dartmouth College, Hanover, NH 03755, USA

² Jet Propulsion Laboratory - Caltech, Pasadena, CA 91109, USA

³ NOAA/OAR/Geophysical Fluid Dynamics Laboratory, Princeton, NJ, USA

⁴ Monash University, Melbourne, Australia

Contents

1	Introduction	7
2	Installation	8
2.1	Introduction	8
2.1.1	Python Interface	8
2.1.2	External Packages	8
2.2	Linux	10
2.2.1	Precompiled Distributable	10
2.2.2	Compiling ISSM from Source	10
2.2.3	Environment	10
2.2.4	System Packages	11
2.2.5	Scripting Interfaces	11
2.2.6	External Packages	12
2.2.7	Configuring and Compiling ISSM	13
2.3	macOS	15
2.3.1	Precompiled Distributable	15
2.3.2	Compiling ISSM from Source	15
2.3.3	Environment	15
2.3.4	System Packages	16
2.3.5	Scripting Interfaces	16
2.3.6	External Packages	17
2.3.7	Configuring and Compiling ISSM	18
2.4	Windows	20
2.4.1	Precompiled Distributable	20
2.4.2	Compiling ISSM from Source	20
2.4.3	MSYS2	20

2.4.4	Scripting Interfaces	22
2.4.5	Shell profile	22
2.4.6	Microsoft MPI	23
2.4.7	External Packages	23
2.4.8	Configuring and Compiling ISSM	23
2.4.9	(Optional) SSHd	25
2.5	High-Performance Computing (HPC)	27
2.5.1	A Note About HPC Environments	27
2.5.2	Configuration and Compiling ISSM	27
2.6	Advanced Features	28
2.6.1	Development Build	28
2.6.2	Debugging Build	28
2.6.3	Dakota	28
2.6.4	Solid Earth	29
2.6.5	Automatic Differentiation	29
3	Getting Started	32
3.1	Introduction	32
3.2	Loading ISSM	33
3.2.1	Environment	33
3.2.2	Scripting Interfaces	33
3.3	Model Class	37
3.3.1	Saving/Loading a Model	38
3.4	Solutions	39
3.5	Plotting	40
3.5.1	Plotting in MATLAB	40
3.5.2	Plotting in Python	69
4	Using ISSM	70
4.1	Introduction	70
4.1.1	Introduction to Capabilities Video Series	70
4.2	Tutorials	71
4.2.1	Datasets	72
4.2.2	Square Ice Shelf	73

4.2.3	Mesh Adaptation	75
4.2.4	Ice Flow Models	89
4.2.5	Ice Sheet Model Intercomparison Project (ISMIP) Tests	90
4.2.6	Inversions	114
4.2.7	Modeling the Greenland Ice Sheet (SeaRISE)	129
4.2.8	Modeling the Greenland Ice Sheet Using IceBridge Data	140
4.2.9	Modeling Pine Island Glacier	152
4.2.10	Pine Island Glacier Sensitivity Study	160
4.2.11	Uncertainty Quantification (UQ)	167
4.2.12	Pine Island Glacier Stochastic Forcing (StISSM)	177
4.2.13	Modeling Jakobshavn Isbræ	183
4.2.14	Subglacial Channel Formation From a Single Moulin (SHAKTI)	187
4.2.15	Modeling Helheim Glacier	191
4.2.16	Subglacial Hydrology of Helheim Glacier (SHAKTI)	194
4.2.17	Adaptive Mesh Refinement (AMR)	199
4.2.18	Sea-Level Fingerprints (GRACE)	203
4.3	Capabilities	206
4.3.1	Mesh Generation	207
4.3.2	Setting a Mask	211
4.3.3	Interpolation Routines	212
4.3.4	Glacial Flow Approximation	213
4.3.5	Stress Balance Solution	214
4.3.6	Mass Transport Solution	219
4.3.7	Thermal Solution	222
4.3.8	Hydrology Solution	225
4.3.9	Damage Evolution	238
4.3.10	Transient Solution	240
4.3.11	Grounding Lines	243
4.3.12	Ice Front Migration (level-set method)	245
4.3.13	Model parameters	245
4.3.14	Running a simulation	245
4.3.15	Glacial Isostatic Adjustment (GIA) Solution	246
4.3.16	Elastostatic Adjustment Solution	250

4.3.17 Sea-level Fingerprints Solution	252
4.3.18 Verbosity	254
4.4 Parameterization of Physical Processes	255
4.4.1 Parameter Files	256
4.4.2 Positive Degree Day (PDD)	258
4.4.3 Surface Mass Balance (SMB)	261
4.4.4 Basal Friction	263
4.4.5 Calving	267
4.4.6 Basal Melt	268
4.4.7 Empirical Scalar Tertiary Anisotropy Regime (ESTAR)	271
4.5 Advanced Features	274
4.5.1 Inversions	275
4.5.2 Rifts	279
4.5.3 Adaptive Mesh Refinement - AMR	283
4.5.4 Quantifications of Margins and Uncertainties (QMU) with Dakota	286
4.5.5 Stochastic Forcing with StISSM	292
5 Supplements	296
5.1 Utilities	296
5.1.1 Mesh	296
5.1.2 Model parameterization	296
5.1.3 Mask	297
5.1.4 Interpolation	297
5.1.5 ARGUS files	297
5.1.6 Results analysis	297
5.1.7 SSH	299
5.2 Changelog	301
5.3 Validation Guide	311
6 Troubleshooting	312
6.1 Troubleshooting	312
6.2 Configuring and Compiling External Packages	313
6.3 Configuring and Compiling ISSM	315
6.4 Runtime Errors	316

6.5 MATLAB Interface	317
6.6 Python Interface	326
6.7 Debugging	327

Chapter 1

Introduction

The following documentation is also available [online](#).

Copying and Pasting Code

Please note that code copied and pasted from this document may need to be adjusted as whitespace and line breaks are not always reproduced accurately in PDF files. It may be easiest to navigate to the corresponding section in the online version of this documentation and copy and paste the relevant section from there.

Chapter 2

Installation

2.1 Introduction

Most users should navigate to the page corresponding to their operating system and choose to either download and install a precompiled distributable or follow the instructions for configuring, compiling, and installing ISSM from source.

Known configurations and other notes for compiling and installing and running ISSM on a number of ‘High-Performance Computing (HPC)’ systems are available and updated as new configurations are discovered.

Configuring and compiling ISSM with extended capabilities (e.g. solid-earth, automatic differentiation) or for development is covered in the ‘Advanced Features’ section.

A Note About Anaconda

Anaconda environments can cause all sorts of toolchain conflicts at configuration and compile time that are hard to diagnose. If you use Anaconda and are having issues compiling the external packages, you might try disabling your Anaconda environment with,

```
conda deactivate [<ENV_NAME>]
```

If disabling your Anaconda environment resolves configuration/compilation issues, you might consider disabling it by default in your shell profile (after the `conda initialize` block). This and other compile time and runtime issues are covered in the ‘Troubleshooting’ chapter.

2.1.1 Python Interface

There are currently two interfaces to ISSM: MATLAB (preferred) or Python (not fully supported). To run ISSM with a Python interface, you will need to install a few packages. The current best practice for achieving this is to do so through a virtual environment. Please see the page corresponding to your operating system for further instructions.

2.1.2 External Packages

Compiling ISSM requires a few external packages. Some of these may be installed via package manager, but we also provide installation scripts which include known, valid configurations for a variety of

external packages on all major operating systems and architectures. These scripts are located in `${ISSM_DIR}/externalpackages/` .

There is no guarantee that compilation of a given external package will work on all systems. Some tweaking of the installation script may be necessary, especially the configuration. Some known gotchas are covered in the ‘[Troubleshooting](#)’ chapter. Feel free as well to search or post troubleshooting questions and issues to the [ISSM Forum](#), or ISSM GitHub repository [Discussions](#) or [Issues](#).

2.2 Linux

2.2.1 Precompiled Distributable

The quickest way to get started with ISSM is to download one of our precompiled distributables. They have been tested on the latest Debian and Ubuntu distributions. Note that for the Python interface, you will also need to follow the setup instructions in the ‘Scripting Interfaces’ → ‘Python Interface’ section.

- MATLAB ↗
- Python 3 ↗ (currently compiled against Python 3.11)

After downloading the distributable, unpack it with the Archive Manager (or similar utility) or in a terminal with,

```
tar -zxvf <DISTIBUTABLE>
```

and move it to the desired location on disk.

You are now ready to [get started with ISSM!](#)

2.2.2 Compiling ISSM from Source

To get started, clone or fork a copy of the ISSM source code repository from [GitHub](#).

2.2.3 Environment

Configuring, compiling, and running ISSM requires at least one environment variable, which can be achieved by running the following,

```
bash, zsh
```

```
export ISSM_DIR=<ISSM_PATH>
```

```
csh
```

```
setenv ISSM_DIR <ISSM_PATH>
```

where `<ISSM_PATH>` is the path to the copy of the ISSM source code that you checked out in the previous step (ex: `${HOME}/ISSM/src`).

2.2.4 System Packages

NOTE: The following assumes use of the GNU compiler collection (i.e. `gcc`, `gfortran`), the APT package manager, and `sudo` privileges.

Install a basic package set with,

```
sudo apt-get install build-essential gfortran libssl-dev
```

If you will be installing the GDAL external package, install dependencies with,

```
sudo apt-get install swig
```

If you will be installing the PROJ external package, install dependencies with,

```
sudo apt-get install libsqlite3-dev sqlite3
```

NOTE: Some systems may require a link to be created so that the linker can find `libstdc++`, which can be accomplished with,

```
sudo ln -s /usr/lib/x86_64-linux-gnu/libstdc++.so.6  
/usr/lib/x86_64-linux-gnu/libstdc++.so
```

2.2.5 Scripting Interfaces

Follow the instructions for setting up the interface that you wish to use with ISSM.

2.2.5.1 MATLAB Interface

Download and install the desired version of MATLAB from the [MathWorks website](#). Make sure to install the optional ‘Mapping Toolbox’.

2.2.5.2 Python Interface

NOTE: There are various methods for installing the required packages. The following is our suggestion.

Install Python 3 with,

```
sudo apt-get install python3-minimal
```

Install required Python 3 header files with,

```
sudo apt-get install python3-dev
```

Install `venv` module with,

```
sudo apt-get install python3-venv
```

Install pip with,

```
sudo apt-get install python3-pip
```

Create a Python virtual environment for ISSM and activate it with,

```
mkdir ${HOME}/.venv  
python3 -m venv ${HOME}/.venv/ISSM  
source ${HOME}/.venv/ISSM/bin/activate
```

Important

- You can create your virtual environment anywhere on disk that you have permission to write to.
- You will have to manually reactivate this virtual environment for each session that you want to run ISSM in, or add the activation command to your shell configuration file.

Then, use pip to install NumPy, SciPy, and dependencies,

```
pip install matplotlib netcdf4 nose numpy pyshp scipy
```

2.2.6 External Packages

The following packages and installation scripts are recommended for a basic installation of ISSM on Linux,

```
autotools  install-linux.sh  
cmake      install.sh  
petsc       install-3.22-linux.sh  
triangle   install-linux.sh  
mlqn3      install.sh
```

Important

After successfully compiling and installing a package, run,

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

or,

```
source ${ISSM_DIR}/etc/environment.csh
```

so that the newly-installed package can be found by subsequent packages that may depend on it.

2.2.7 Configuring and Compiling ISSM

We rely on the Autotools to allow us to configure and compile ISSM on a variety of systems. To begin, run,

```
cd ${ISSM_DIR}
autoreconf -ivf
```

Next, create a configuration file called `configure.sh` in `${ISSM_DIR}` . Its contents will depend on the interface you will be using, the external packages and capabilities you wish to use, and the location of certain libraries and executables on disk. The following examples can be used for an installation of ISSM with basic capabilities:

2.2.7.1 MATLAB Interface

```
./configure \
--prefix=${ISSM_DIR} \
--with-matlab-dir="" \
--with-fortran-lib="-L/usr/local/gfortran/lib -lgfortran" \
--with-mpi-include="${ISSM_DIR}/externalpackages/petsc/install/include" \
 \
--with-mpi-libflags="-L${ISSM_DIR}/externalpackages/petsc/install/lib \
-lmpi -lmpicxx -lmpifort" \
--with-metis-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-parmetis-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-petsc-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-triangle-dir="${ISSM_DIR}/externalpackages/triangle/install" \
 \
--with-m1qn3-dir="${ISSM_DIR}/externalpackages/m1qn3/install"
```

where <MATLAB_PATH> is the path to the MATLAB installation that you wish to use (e.g. /usr/local/MATLAB/R2024a).

2.2.7.2 Python Interface

```
./configure \
--prefix="${ISSM_DIR}" \
--with-python="${HOME}/.venv/issm/bin/python" \
--with-fortran-lib="-L/usr/local/gfortran/lib -lgfortran" \
--with-mpi-include="${ISSM_DIR}/externalpackages/petsc/install/include" \
\
--with-mpi-libflags="-L${ISSM_DIR}/externalpackages/petsc/install/lib \
-lmpi -lmpicxx -lmpifort" \
--with-metis-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-parmetis-dir="${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="${ISSM_DIR}/externalpackages/petsc/install" \
--with-petsc-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-triangle-dir="${ISSM_DIR}/externalpackages/triangle/install" \
\
--with-m1qn3-dir="${ISSM_DIR}/externalpackages/m1qn3/install"
```

Again, you might need to make adjustments to the above configurations based on your system and needs. We have a number of resources to help here:

- Various external package install scripts and configurations in \${ISSM_DIR}/jenkins
- ‘High-Performance Computing’ section
- ‘Advanced Features’ section
- ISSM Forum ↗
- ISSM GitHub repository [Discussions](#) ↗

If the configuration completed without any errors, ISSM can now be compiled with,

```
make
make install
```

You are now ready to [get started with ISSM!](#)

2.3 macOS

2.3.1 Precompiled Distributable

The quickest way to get started with ISSM is to download one of our precompiled distributables. They have been tested on the latest version of macOS.

Note that for the Python interface, you will also need to follow the setup instructions in the ‘[Scripting Interfaces](#)’ → ‘[Python Interface](#)’ section.

2.3.1.1 Silicon-based Macs

- [MATLAB](#)
- [Python 3](#) (currently compiled against Python 3.11)

2.3.1.2 Intel-based Macs

- [MATLAB](#)
- [Python 3](#) (currently compiled against Python 3.11)

After downloading the distributable, simply unzip it and move it to the desired location on disk.

You are now ready to [get started with ISSM!](#)

2.3.2 Compiling ISSM from Source

To get started, clone or fork a copy of the ISSM source code repository from [GitHub](#).

2.3.3 Environment

Configuring, compiling, and running ISSM requires at least one environment variable, which can be achieved by running the following,

```
bash, zsh
```

```
export ISSM_DIR=<ISSM_PATH>
```

```
csh
```

```
setenv ISSM_DIR <ISSM_PATH>
```

where `<ISSM_PATH>` is the path to the copy of the ISSM source code that you checked out in the previous step (e.g. `${HOME}/ISSM/src`).

2.3.4 System Packages

NOTE: The following assumes use of the GNU compiler collection (i.e. `gcc`, `gfortran`), the Homebrew package manager [↗](#), and `sudo` privileges.

In order to install ISSM on macOS, you will need the Xcode Command Line Tools, which can be installed with,

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

Alternatively, you can use the compiler and other build tools that come with Xcode [↗](#), but if you do not otherwise use Xcode as an IDE, we recommend installing only the Command Line Tools.

Important

Having both the Command Line Tools and Xcode installed can cause toolchain conflicts in some cases. If you are experiencing difficult-to-debug issues during configuration or compilation, try the following, which will select the Command Line Tools as the default,

```
sudo xcode-select --switch /Library/Developer/CommandLineTools
```

Unfortunately, both the Command Line Tools and Xcode lack a Fortran compiler, which is required for various external packages. We recommend either of the following methods for installing `gfortran`,

- [FX Coudert's GitHub repository for macOS installers for GNU Fortran ↗](#).
- Via Homebrew with,

```
brew install gfortran
```

If you will be installing the PROJ external package, you can install dependencies via Homebrew with,

```
brew install sqlite3
```

2.3.5 Scripting Interfaces

Follow the instructions for setting up the interface that you wish to use with ISSM.

2.3.5.1 MATLAB Interface

Download and install the desired version of MATLAB from the [MathWorks website ↗](#). Make sure to install the optional ‘Mapping Toolbox’.

2.3.5.2 Python Interface

NOTE: - We assume use of copy of Python 3 supplied by the Command Line Tools. - There are various methods for installing the required packages. The following is our suggestion.

Create a Python virtual environment for ISSM and activate it with,

```
mkdir ${HOME}/.venv
python3 -m venv ${HOME}/.venv/ISSM
source ${HOME}/.venv/ISSM/bin/activate
```

Important

- You can create your virtual environment anywhere on disk that you have permission to write to.
- You will have to manually reactivate this virtual environment for each session that you want to run ISSM in, or add the activation command to your shell configuration file.

Then, use pip to install NumPy, SciPy, and dependencies,

```
pip install matplotlib netcdf4 nose numpy pyshp scipy
```

2.3.6 External Packages

The following packages and installation scripts are recommended for a basic installation of ISSM on Linux,

```
autotools  install-linux.sh
cmake      install.sh
petsc       install-3.22-mac.sh
triangle   install-mac.sh
mlqn3      install.sh
```

Important

After successfully compiling and installing a package, run,

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

or,

```
source ${ISSM_DIR}/etc/environment.csh
```

so that the newly-installed package can be found by subsequent packages that may depend on it.

2.3.7 Configuring and Compiling ISSM

We rely on the Autotools to allow us to configure and compile ISSM on a variety of systems. To begin, run,

```
cd ${ISSM_DIR}
autoreconf -ivf
```

Next, create a configuration file called `configure.sh` in `${ISSM_DIR}`. Its contents will depend on the interface you will be using, the external packages and capabilities you wish to use, and the location of certain libraries and executables on disk.

NOTE: Both of the following interface configurations refer to `<LIBGFORTRAN_PATH>`, which should be substituted for the path to the parent directory of `libgfortran`. Common locations of `<LIBGFORTRAN_PATH>` on macOS are,

- `/usr/local/gfortran/lib` if you installed using one of the packages from FX Coudert's GitHub repository for macOS installers for GNU Fortran
- `/opt/homebrew/lib/gcc/current` if you installed via Homebrew on a Silicon-based Mac
- `/usr/local/Cellar/gcc/<VER>/lib/gcc/current` (where `<VER>` is the version of `gcc`) if you installed via Homebrew on an Intel-based Mac

The following examples can be used for an installation of ISSM with basic capabilities:

2.3.7.1 MATLAB Interface

```
./configure \
--prefix=${ISSM_DIR} \
--with-matlab-dir="" \
--with-fortran-lib="-L<LIBGFORTRAN_PATH> -lgfortran" \
--with-mpi-include="${ISSM_DIR}/externalpackages/petsc/install/include" \
 \
--with-mpi-libflags="-L${ISSM_DIR}/externalpackages/petsc/install/lib \
-lmpi -lmpicxx -lmpifort" \
--with-metis-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-parmetis-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-petsc-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-triangle-dir="${ISSM_DIR}/externalpackages/triangle/install" \
 \
--with-m1qn3-dir="${ISSM_DIR}/externalpackages/m1qn3/install"
```

where `<MATLAB_PATH>` is the path to the MATLAB installation that you wish to use (e.g. `/Applications/MATLAB_R2024a.app`).

2.3.7.2 Python Interface

```
./configure \
--prefix="${ISSM_DIR}" \
--with-python="${HOME}/.venv/issm/bin/python" \
--with-fortran-lib="-L<LIBGFORTRAN_PATH> -lgfortran" \
--with-mpi-include="${ISSM_DIR}/externalpackages/petsc/install/include" \
\
--with-mpi-libflags="-L${ISSM_DIR}/externalpackages/petsc/install/lib \
-lmpi -lmpicxx -lmpifort" \
--with-metis-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-parmetis-dir="${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="${ISSM_DIR}/externalpackages/petsc/install" \
--with-petsc-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-triangle-dir="${ISSM_DIR}/externalpackages/triangle/install" \
\
--with-m1qn3-dir="${ISSM_DIR}/externalpackages/m1qn3/install"
```

Again, you might need to make adjustments to the above configurations based on your system and needs. We have a number of resources to help here:

- Various external package install scripts and configurations in `${ISSM_DIR}/jenkins`
- ‘High-Performance Computing’ section
- ‘Advanced Features’ section
- ISSM Forum ↗
- ISSM GitHub repository [Discussions](#) ↗

If the configuration completed without any errors, ISSM can now be compiled with,

```
make
make install
```

You are now ready to [get started with ISSM!](#)

2.4 Windows

2.4.1 Precompiled Distributable

The quickest way to get started with ISSM is to download one of our precompiled distributables. They have been tested on Windows 10 and 11.

- [MATLAB](#)

NOTE

- ISSM is currently limited to basic capabilities on Windows. We will be working soon on supporting external packages such as Dakota and advanced capabilities like modelling of solid earth processes.
- Currently, only the MATLAB interface to ISSM is supported on Windows.

After downloading the distributable, unpack it with the ‘Extract’ feature and move it to the desired location on disk.

You are now ready to [get started with ISSM!](#)

2.4.2 Compiling ISSM from Source

The following instructions detail how to create an environment for compiling ISSM from source on Windows. We rely on MSYS2 to provide a Linux-like interface and the MinGW compiler chain to generate native Windows executables and libraries. There may be other methods for achieving the above, which we invite you to share on [the ISSM forum](#).

NOTE: You will have to use an Administrator user account for some of the following to work as intended

2.4.3 MSYS2

2.4.3.1 Install MSYS2

- Navigate to [the MSYS2 website](#)
- Download the installer
- When the download completes, run the installer
- Use the default value for ‘Installation Folder’
- Use the default value for ‘Start Menu shortcuts’
- Deselect ‘Run MSYS 64bit now’ and click the ‘Finish’ button

2.4.3.2 Set up shortcut for MSYS2 terminal emulator

- In the Windows ‘Search Bar’, search for "MSYS2"
- The ‘Best match’ should be ‘MSYS2 MSYS’; click ‘Open file location’

- In the resulting File Explorer window, right-click ‘MSYS2 MinGW 64-bit’ and select ‘Send to’ → ‘Desktop (create shortcut)’
- Right-click on the newly-created desktop shortcut and select ‘Properties’
 - click the ‘Advanced...’ button
 - check the box labeled ‘Run as administrator’
 - click the ‘OK’ button
 - click the ‘Apply’ button
 - click the ‘OK’ button

2.4.3.3 Update the package database and install required packages

- Double-click the ‘MSYS2 MinGW 64-bit’ desktop shortcut
- At the resulting command prompt run,

```
$ pacman -Syu
```

to update the database and base packages, entering "Y" when prompted

- The previous step will result in the window being closed, so double-click the ‘MSYS2 MinGW 64-bit’ desktop shortcut again
- At the resulting command prompt run,

```
$ pacman -Su
```

until the resulting output is,

```
:: Starting core system upgrade...
there is nothing to do
:: Starting full system upgrade...
there is nothing to do
```

- Install necessary packages with,

```
$ pacman -S --needed base-devel git openssh python
python-setuptools subversion unzip mingw-w64-x86_64-autotools
mingw-w64-x86_64-cmake mingw-w64-x86_64-gcc-fortran
mingw-w64-x86_64-toolchain
```

entering "Y" or simply hitting the return key as needed

- (Optional) Install Vim text editor with,

```
$ pacman -S vim
```

2.4.4 Scripting Interfaces

Follow the instructions for setting up the interface that you wish to use with ISSM.

2.4.4.1 MATLAB

Download and install the desired version of MATLAB from the [MathWorks website](#). Make sure to install the optional ‘Mapping Toolbox’.

2.4.4.2 Python

The Python interface to ISSM on Windows is currently under development.

2.4.5 Shell profile

2.4.5.1 .bash_profile

- Open `/c/msys64/home/<USER>/ .bash_profile` for editing (the easiest way to do this is with vim if you installed it in the previous step)
- Add the following to the bottom of the file,

```
# Allow for NTFS symbolic links
export MSYS=winsymlinks:nativestrict
```

2.4.5.2 .bashrc

- Open `/c/msys64/home/<USER>/ .bashrc` for editing and add the following at the bottom of the file,

```
## MATLAB
#
MATLAB_VER=<MATLAB_VER> # Allows for easy resetting of MATLAB
version added to path
export MATLAB_PATH=$(cygpath -u $(cygpath -ms "/c/Program
Files/MATLAB/${MATLAB_VER}"))
export PATH="${MATLAB_PATH}/bin:${PATH}"

## ISSM
#
export ISSM_DIR=<ISSM_PATH>
export ISSM_DIR_WIN=$(cygpath -ms "${ISSM_DIR}") # Needed by
MATLAB
```

where `<MATLAB_VER>` is the version of MATLAB that you have installed (for example, "R2023b") and `<ISSM_DIR>` is the path to the copy of the ISSM source code that you checked out (e.g. `/c/Users/<USER>/ISSM/src`, where `<USER>` is your username)

2.4.6 Microsoft MPI

- Navigate to the ‘Microsoft MPI Release Notes’ webpage [↗](#)
- Click the link for ‘Microsoft Download Center’ that corresponds with the latest release (take note of the version number that you download for the next step; it can also be found by going to ‘Settings’ / ‘Apps & Features’)
- Click the ‘Download’ button
- Make sure both boxes are checked, then click the ‘Next’ button
- Click the ‘Save File’ button for each file
- When the download completes, run each installer
- Follow the prompts, using the default installation directories

2.4.7 External Packages

The following packages and installation scripts are recommended for a basic installation of ISSM on Windows,

```
msmpi      install.sh
petsc      install-3.14-win-msys2-mingw-msmpi.sh
triangle   install-win-msys2-mingw.sh
mlqn3      install-win-msys2-mingw.sh
```

NOTE

After successfully compiling and installing a package, run,

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

or,

```
source ${ISSM_DIR}/etc/environment.csh
```

so that the newly-installed package can be found by subsequent packages that may depend on it.

2.4.8 Configuring and Compiling ISSM

We rely on the Autotools to allow us to configure and compile ISSM on a variety of systems. To begin, run,

```
cd ${ISSM_DIR}
autoreconf -ivf
```

NOTE:

- Replace `<NUM_CPUS>` with the number of available CPU's
- `MATLAB_PATH` was defined previously in `bashrc`.
- `MSMPI_ROOT` will be defined after running,

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

- You may need to update the path to `libgfortran` in the `--with-fortran-lib` option

Next, create a configuration file called `configure.sh` in `${ISSM_DIR}`. Its contents will depend on the interface you will be using, the external packages and capabilities you wish to use, and the location of certain libraries and executables on disk. The following examples can be used for an installation of ISSM with basic capabilities,

2.4.8.1 MATLAB Interface

```
./configure \
--prefix=${ISSM_DIR} \
--with-numthreads=<NUM_CPUS> \
--with-matlab-dir=${MATLAB_PATH} \
--with-mpi-include="${MSMPI_ROOT}/include" \
--with-mpi-libdir="-Wl,-L${MSMPI_ROOT}/lib" \
--with-mpi-libflags="-Wl,-lmsmpi" \
--with-fortran-lib="-Wl,-L/c/msys64/mingw64/lib -Wl,-lgfortran" \
--with-metis-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-parmetis-dir="${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="${ISSM_DIR}/externalpackages/petsc/install" \
--with-petsc-dir="${ISSM_DIR}/externalpackages/petsc/install" \
--with-triangle-dir="${ISSM_DIR}/externalpackages/triangle/install" \
 \
--with-m1qn3-dir="${ISSM_DIR}/externalpackages/m1qn3/install" \
--with-semic-dir="${ISSM_DIR}/externalpackages/semic/install"
```

Again, you might need to make adjustments to the above configurations based on your system and needs. We have a number of resources to help here:

- Various external package install scripts and configurations in `${ISSM_DIR}/jenkins`
- ‘High-Performance Computing’ section
- ‘Advanced Features’ section
- ISSM Forum ↗
- ISSM GitHub repository [Discussions](#) ↗

If the configuration completed without any errors, ISSM can now be compiled,

```
$ cd ${ISSM_DIR}
$ make
$ make install
```

You are now ready to [get started with ISSM!](#)

2.4.9 (Optional) SSHd

NOTE: The following is probably not applicable to most users.

- Navigate to [the MSYS2 ‘Setting up SSHd’ webpage](#) ↗
- Copy the contents of the code block and paste to a new text file
- Set the value of the variable `UNPRIV_NAME` to the desired user
 - If the user does not already exist, it will be created
 - If the user does already exist, note the default password is the same that is used to log in to the Windows user account
- Save the file out to the location of your choice
- In a MSYS2 MinGW 64-bit shell instance, run the script
- If errors occur with messages about missing packages, install those packages and run the script again
- You can disregard the message,

```
cygrunsrv: Error removing a service: OpenService: Win32 error
1060:
The specified service does not exist as an installed service.
```

- The following message indicates that setup was successful,

```
The MSYS2 sshd service is starting.
The MSYS2 sshd service was started successfully.
```

- You can test that the service and your log in are working correctly by running,

```
ssh -l <UNPRIV_NAME> localhost
```

where `<UNPRIV_NAME>` is the same user that we authorized to use the service. You should be prompted to accept an ECDSA fingerprint, you which you respond "yes". Then, enter the password for this account. If all goes well, you should now have a prompt that reads,

```
<UNPRIV_NAME>@<HOSTNAME> MSYS ~
```

- Once logged in to the target machine, open `/etc/ssh/sshd_config` for editing, add,

```
AcceptEnv MSYSTEM
```

then save out the changes. On the client machine, open `/etc/ssh/ssh_config` for editing, add,

```
SendEnv MSYSTEM
```

to the file (you can add `MSYSTEM` to the list of environment variables if `SendEnv` already exists), then save out the changes. You can now prefix your SSH commands like,

```
MSYSTEM=MINGW64 ssh [...]
```

in order to log in to the MSYS2 MinGW 64-bit shell (other possible values are `MSYS2` and `MINGW32`).

- If an attempted SSH connection from a remote machine stalls out or is denied, it may be the case that you are running Windows Defender Firewall and need to open port 22. To do so,
 - in the Windows search bar, search for "Defender" and select 'Windows Defender Firewall with Advanced Security'
 - click 'Inbound Rules' in the left sidebar
 - click 'New Rule...' in the right sidebar
 - select 'Port', then click the 'Next' button
 - select 'TCP'
 - select 'Specific local ports', set the field to "22", then click the 'Next' button
 - select 'Allow the connection' then click the 'Next' button (if you are using a third-party firewall application, it is up to you to determine how to open port 22)
 - uncheck the 'Public' box (if a subsequent attempted connection stalls out or is denied, you may need to edit this rule, checking the 'Public' box, but try first without it), then click the 'Next' button
 - set the 'Name' field to "SSH", then click the 'Next' button
- The `ssh-keygen` utility can be used to create a more secure SSH connection and to protect your Windows user password
- If you later decide that you want to stop the `sshd` service, you can do so with,

```
net stop msys2_sshd
```

and can remove the service altogether with,

```
cygrunsrv -R msys2_sshd
```

Sources:

- [MSYS2 'Setting up SSHd' webpage ↗](#)
- [Sam Hocevar's GitHub Gist page on setting up SSHd under MSYS2 ↗](#)

2.5 High-Performance Computing (HPC)

2.5.1 A Note About HPC Environments

MATLAB and Python are used only for model setup and post-processing of simulation results (e.g. plotting). As such, when leveraging the power of HPC, our general strategy is to install one copy of ISSM with the MATLAB and/or Python wrappers on a local machine, and a second copy of ISSM with only the binaries (e.g. `issm.exe`) on the cluster. We can achieve this second type of build by configuring ISSM with the `--without-wrappers` option. The MATLAB or Python interface can then send the binary input files to the cluster and fetch the output file once the run is completed.

Note as well that the ‘local’ machine in the above case may be one that is physically remote to you. For example, you might install ISSM with its Python interface so that it is available in a remote Jupyter Lab/Hub environment. This paradigm is becoming increasingly popular with computing centers that provide access to HPC.

2.5.2 Configuration and Compiling ISSM

Please see the [ISSM Development Wiki](#) for notes on configuring and compiling on various HPC systems.

We will be working soon to migrate the content from the wiki to this documentation.

2.6 Advanced Features

2.6.1 Development Build

If you plan on making changes to your installation of ISSM and/or contributions to the code base, you can enable development by adding the following to your configuration,

```
--enable-development
```

This prevents MATLAB/Python scripts from being copied from their respective locations in `${ISSM_DIR}/src` to `${ISSM_DIR}/bin` when ISSM is recompiled

2.6.2 Debugging Build

Debugging features can be enabled by adding the following to your ISSM configuration with,

```
--enable-debugging
```

This will make several additional fields available for inspecting program state in certain modules of the ISSM core.

We can debug crashes during the solve phase by installing Valgrind using the appropriate installation script available at `${ISSM_DIR}/externalpackages/valgrind`.

See the '[Debugging](#)' section for more info.

2.6.3 Dakota

The [Dakota project](#)  delivers both state-of-the-art research and robust, usable software for optimization and UQ. Broadly, the Dakota software's advanced parametric analyses enable design exploration, model calibration, risk analysis, and quantification of margins and uncertainty with computational models.

Dakota can be enabled by compiling and installing the following additional external packages,

Linux

```
gsl      install.sh
boost    install-1.7-linux.sh
dakota   install-6.2-linux.sh
chaco    install-linux.sh
```

macOS

```
gsl      install.sh
boost    install-1.7-mac.sh
dakota   install-6.2-mac.sh
chaco    install-mac.sh
```

and making the following additions to the ISSM configuration, reconfiguring, and recompiling,

```
--with-gsl-dir=${ISSM_DIR}/externalpackages/gsl/install \
--with-boost-dir=${ISSM_DIR}/externalpackages/boost/install \
--with-dakota-dir=${ISSM_DIR}/externalpackages/dakota/install \
--with-chaco-dir=${ISSM_DIR}/externalpackages/chaco/install
```

2.6.4 Solid Earth

Capabilities for modelling solid earth processes can be enabled by compiling and installing the following additional external packages,

Linux

```
curl      install-7-mac.sh
hdf5      install-1.sh
netcdf    install-4.sh
proj      install-6.sh
gdal      install-3.sh
gshhg     install.sh
gmt       install-6-mac.sh
gmsh      install-4-mac.sh
```

macOS

```
curl      install-7-linux.sh
hdf5      install-1.sh
netcdf    install-4.sh
proj      install-6.sh
gdal      install-3.sh
gshhg     install.sh
gmt       install-6-linux.sh
gmsh      install-4-linux.sh
```

and making the following additions to the ISSM configuration, reconfiguring, and recompiling,

```
--with-hdf5-dir="${ISSM_DIR}/externalpackages/hdf5/install" \
--with-proj-dir="${ISSM_DIR}/externalpackages/proj/install"
```

NOTE: If you will be using the Python interface to ISSM, use the `install-3-python.sh` script to compile GDAL.

2.6.5 Automatic Differentiation

Automatic differentiation is only supported under Linux and Mac.

2.6.5.1 CoDiPack installation (recommended)

CoDiPack can be enabled by compiling and installing the following additional external packages,

```
gsl      install.sh
codipack  install.sh
medipack  install-linux.sh
```

Note that the PETSc libraries are incompatible with automatic differentiation, but we still use PETSc to install other external packages to solve linear systems. As such, we remove the `--with-petsc-dir` option from the ISSM configuration. You will also need to deactivate ISSM's kriging capability. The following is an example configuration script:

```
export CXXFLAGS="-g -O3 -fPIC -std=c++11 -DCODI_ForceInlines"
./configure \
--prefix=${ISSM_DIR} \
--enable-tape-alloc \
--without-kriging \
--enable-debugging \
--without-kml \
--without-Love \
--without-Sealevelchange \
--with-numthreads=<NUM_CPUS> \
--with-matlab-dir=<MATLAB_PATH> \
--with-fortran-lib="-L/usr/lib/x86_64-linux-gnu -lgfortran" \
--with-mpi-libflags="-L${ISSM_DIR}/externalpackages/petsc/install/lib \
-lmp -lmpicxx -lmpifort" \
--with-mpi-include=${ISSM_DIR}/externalpackages/petsc/install/include \
 \
--with-blas-lapack-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-metis-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-parmetis-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-scalapack-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-mumps-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-gsl-dir=${ISSM_DIR}/externalpackages/gsl/install \
--with-triangle-dir=${ISSM_DIR}/externalpackages/triangle/install \
--with-m1qn3-dir="$ISSM_DIR/externalpackages/m1qn3/install" \
--with-codipack-dir="$ISSM_DIR/externalpackages/codipack/install" \
--with-medipack-dir="$ISSM_DIR/externalpackages/medipack/install"
```

where `<NUM_CPUS>` is the number of available CPU's.

Reconfigure and recompile ISSM and it will now be fully adjoinable.

2.6.5.2 ADOL-C installation

ADOL-C can be enabled by compiling and installing the following additional external packages,

Linux

```
adjoinablempi install-linux.sh
ADOL-C      install.sh
```

macOS

```
adjoinablempi install-mac.sh
adolc      install.sh
```

Note that the PETSc libraries are incompatible with automatic differentiation, but we still use PETSc to install other external packages to solve linear systems. As such, we remove the `--with-petsc-dir` option from the ISSM configuration. You will also need to deactivate ISSM's kriging capability. The following is an example configuration script:

```
./configure \
--prefix=${ISSM_DIR} \
--enable-debugging \
--without-kriging \
--without-kml \
--without-Sealevelchange \
--without-Love \
--with-numthreads=<NUM_CPUS> \
--with-matlab-dir=<MATLAB_PATH> \
--with-fortran-lib="-L/usr/lib/x86_64-linux-gnu -lgfortran" \
--with-mpi-libflags="-L${ISSM_DIR}/externalpackages/petsc/install/lib \
-lmpi -lmpicxx -lmpifort" \
--with-mpi-include=${ISSM_DIR}/externalpackages/petsc/install/include \
 \
--with-blas-lapack-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-metis-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-parmetis-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-scalapack-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-mumps-dir=${ISSM_DIR}/externalpackages/petsc/install \
--with-gsl-dir=${ISSM_DIR}/externalpackages/gsl/install \
--with-triangle-dir=$ISSM_DIR/externalpackages/triangle/install \
--with-m1qn3-dir="$ISSM_DIR/externalpackages/m1qn3/install" \
--with-adolc-dir=$ISSM_DIR/externalpackages/adolc/install \
--with-ampi-dir=$ISSM_DIR/externalpackages/adjoinablempi/install
```

where `<NUM_CPUS>` is the number of available CPU's.

Reconfigure and recompile ISSM and it will now be fully adjoinable.

Chapter 3

Getting Started

3.1 Introduction

The following pages will guide you through the basic workflow of using ISSM: loading ISSM in the interface of your choice, defining a model, requesting a solution, and plotting the results.

Note

For new MATLAB users, we highly recommend enrolling in the free [MATLAB Onramp](#)↗ course offered by MathWorks. This two hour, self-paced course will familiarize users with MATLAB's features and syntax.

3.2 Loading ISSM

By default, MATLAB and Python cannot locate ISSM and its external packages. We outline here how to load ISSM in your chosen interface.

3.2.1 Environment

First, we need to make sure our interface can find ISSM and its external packages by running,

`bash, zsh`

```
export ISSM_DIR=<ISSM_PATH>
source ${ISSM_DIR}/etc/environment.sh
```

`csh`

```
setenv ISSM_DIR <ISSM_PATH>
source ${ISSM_DIR}/etc/environment.csh
```

where `<ISSM_PATH>` is the path to ISSM on disk (ex: `${HOME} /ISSM/src`). This works whether you downloaded one of our precompiled distributables or compiled ISSM from source.

3.2.2 Scripting Interfaces

3.2.2.1 MATLAB

NOTE: There does not currently exist a method for starting the MATLAB GUI using its launcher icon with ISSM loaded.

Start MATLAB in a terminal by running,

```
matlab
```

If the `matlab` executable is not on your environment's path, you will need to add its parent directory to the `PATH` environment variable, create an alias for `matlab`, or start MATLAB with the full path to the `matlab` executable.

After starting MATLAB, you must tell it where to find ISSM's binaries and libraries by running,

```
addpath <ISSM_PATH>/bin <ISSM_PATH>/lib
```

again, where `<ISSM_PATH>` is the path to ISSM on disk. Alternatively, you can use the '[Set Path](#)' dialog  to add these paths manually. If you installed ISSM by downloading one of our precompiled distributables, you will also need to run,

```
addpath <ISSM_PATH>/share
```

Yet another alternative is to feel the `addpath` command as a statement to the `matlab` startup command, for example with,

```
matlab -r 'addpath ${ISSM_DIR}/bin ${ISSM_DIR}/lib'
```

You can verify that ISSM is findable by running,

```
issmversion
```

which should print a message similar to,

```
Ice-sheet and Sea-level System Model (ISSM) Version 4.22
(website: http://issm.jpl.nasa.gov contact:issm@jpl.nasa.gov)

Build date: Wed Sep 18 14:00:06 PDT 2023
Copyright (c) 2009-2023 California Institute of Technology

to get started type: issmdoc
```

To avoid having to manually load ISSM each time you start up MATLAB, you might create an alias in your shell configuration file like,

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

If you will not be using MATLAB's GUI, you might create aliases like,

```
alias matlab_no_gui="matlab -nosplash -nodesktop -nodisplay"
alias matlab_no_gui-issm="matlab -r 'addpath ${ISSM_DIR}/bin
${ISSM_DIR}/lib'"
```

3.2.2.2 Python

Before starting Python, activate the virtual environment that you set up during installation with, for example,

```
source ${HOME}/.venv/ISSM/bin/activate
```

After starting Python, you must tell it where to find ISSM's binaries and libraries by running,

```
import os
import sys
```

```
ISSM_DIR = os.getenv('ISSM_DIR')
sys.path.append(ISSM_DIR + '/bin')
sys.path.append(ISSM_DIR + '/lib')
```

If you installed ISSM by downloading one of our precompiled distributables, you will also need to run,

```
sys.path.append(os.getenv('ISSM_DIR') + '/share')
```

You can verify that ISSM is findable by running,

```
from issmversion import issmversion
```

which should print a message similar to,

```
Ice-sheet and Sea-level System Model (ISSM) Version 4.22
(website: http://issm.jpl.nasa.gov contact:issm@jpl.nasa.gov)

Build date: Wed Sep 18 14:00:06 PDT 2023
Copyright (c) 2009-2023 California Institute of Technology
```

To avoid having to manually load ISSM each time you start up Python, you might create a startup script, for example, `${HOME}/ISSM/bin/sitecustomize.py` with the following contents,

```
#!/usr/bin/env python3
import os
import sys
ISSM_DIR = os.getenv('ISSM_DIR')
sys.path.append(ISSM_DIR + '/bin')
sys.path.append(ISSM_DIR + '/lib')
```

Then, add the following to the bottom of your virtual environment activation script (again, for example, `${HOME}/.venv/ISSM/bin/activate`):

```
export PYTHONPATH="${HOME}/ISSM/bin:$PYTHONPATH"
```

ISSM should now be findable by Python whether you are in an interactive or non-interactive session as long as you have first activated your virtual environment.

3.2.2.3 Development Build

If you configured and compiled ISSM for development, you will have to load ISSM a bit differently.

MATLAB

```
addpath <ISSM_PATH>/src/m/dev  
devpath
```

again, where `<ISSM_PATH>` is the path to ISSM on disk.

Python Before starting Python, run,

```
export PYTHONPATH="${ISSM_DIR}/src/m/dev"  
export PYTHONSTARTUP="${PYTHONPATH}/devpath.py"
```

For IPython users, instead launch the interface with,

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

3.3 Model Class

All the data belonging to a model (geometry, node coordinates, results, etc.) is held in the same object `model`. To create a new model in MATLAB, run,

```
md = model
```

and in Python,

```
from model import *
md = model()
```

This will create a new model named `md` whose class is `model`. The information contained in the model `md` is grouped by class, each of which are comprised of fields related to that particular aspect of the model (e.g. mesh, material properties, friction, stressbalance solution, solution results). When one creates a new model, all of 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 by running, in MATLAB,

```
>> md
md =
    mesh: [1x1 mesh2d]          -- mesh properties
    mask: [1x1 mask]           -- defines grounded and
        floating elements
    geometry: [1x1 geometry]      -- surface elevation,
        bedrock topography, ice thickness, ...
        [...]
    results: [1x1 struct]         -- model results
    radaroverlay: [1x1 radaroverlay]   -- radar image for plot
        overlay
    miscellaneous: [1x1 miscellaneous] -- miscellaneous fields
```

or, in Python,

```
>>> print(md)
    mesh: [1x1 mesh2d]          -- mesh properties
    mask: [1x1 mask]           -- defines grounded and
        floating elements
    geometry: [1x1 geometry]      -- surface elevation,
        bedrock topography, ice thickness, ...
        [...]
    radaroverlay: [1x1 radaroverlay]   -- radar image for plot
        overlay
    miscellaneous: [1x1 miscellaneous] -- miscellaneous fields
    stochasticforcing: [1x1 stochasticforcing] -- stochasticity
        applied to model forcings
```

Likewise, you can display all the fields associated with, for example, the model's mesh by running,

```
>> md.mesh
```

or,

```
>>> print(md.mesh)
```

3.3.1 Saving/Loading a Model

You can save the model with all its fields so that the saved file contains all of the information in the model by running,

```
save squaremodel md
```

This will create a file `squaremodel.mat` made from the model `md`. Likewise, to load this file, run,

```
>> loadmodel squaremodel
```

The loaded model will be named `md`.

3.4 Solutions

After parameterizing your model, you are ready to request a solution. In MATLAB, this is done by running,

```
>> md = solve(md, <solution_type>);
```

and in Python,

```
>>> md = solve(md, <solution_type>)
```

where `<solution_type>` is a string representing a given solution type, for example, `'Stressbalance'`.

The following pages provide more in-depth information on the various solution types,

- [thermal](#)
- [hydrology](#)
- [stress-balance](#)
- [mass transport](#)
- [Glacial Isostatic Adjustment \(GIA\)](#)

Running one or more of the above solutions over time is detailed on the Transient Solutions page.

3.5 Plotting

3.5.1 Plotting in MATLAB

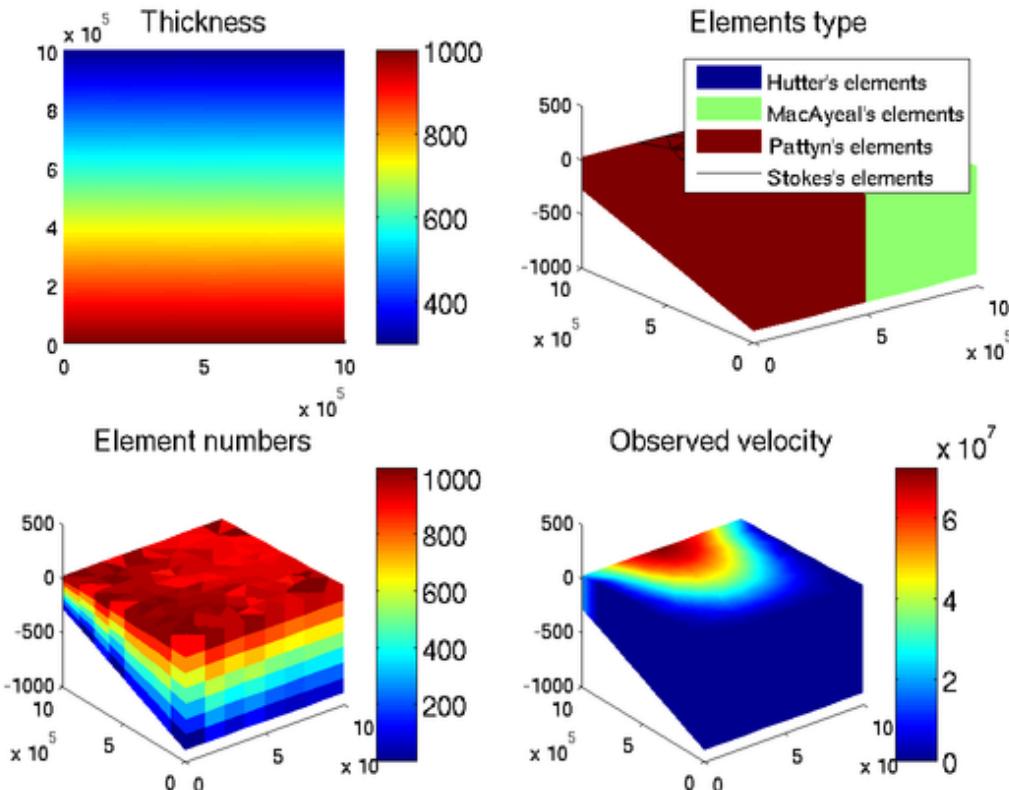
3.5.1.1 plotmodel

`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`.

3.5.1.2 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. For example:

```
>> plotmodel(md, 'data', md.initialization.vel, 'data', 'mesh',
    'view#2', 3, 'colorbar#all', 'on', 'axis#1', 'off equal')
```

axis

Same as standard `axis` MATLAB option:

```
>> plotmodel(md, 'data', md.vel, 'axis', 'tight')
```

view

Same as standard `view` MATLAB option:

```
>> plotmodel(md, 'data', md.vel, 'view', 2)
```

xlim, ylim, zlim

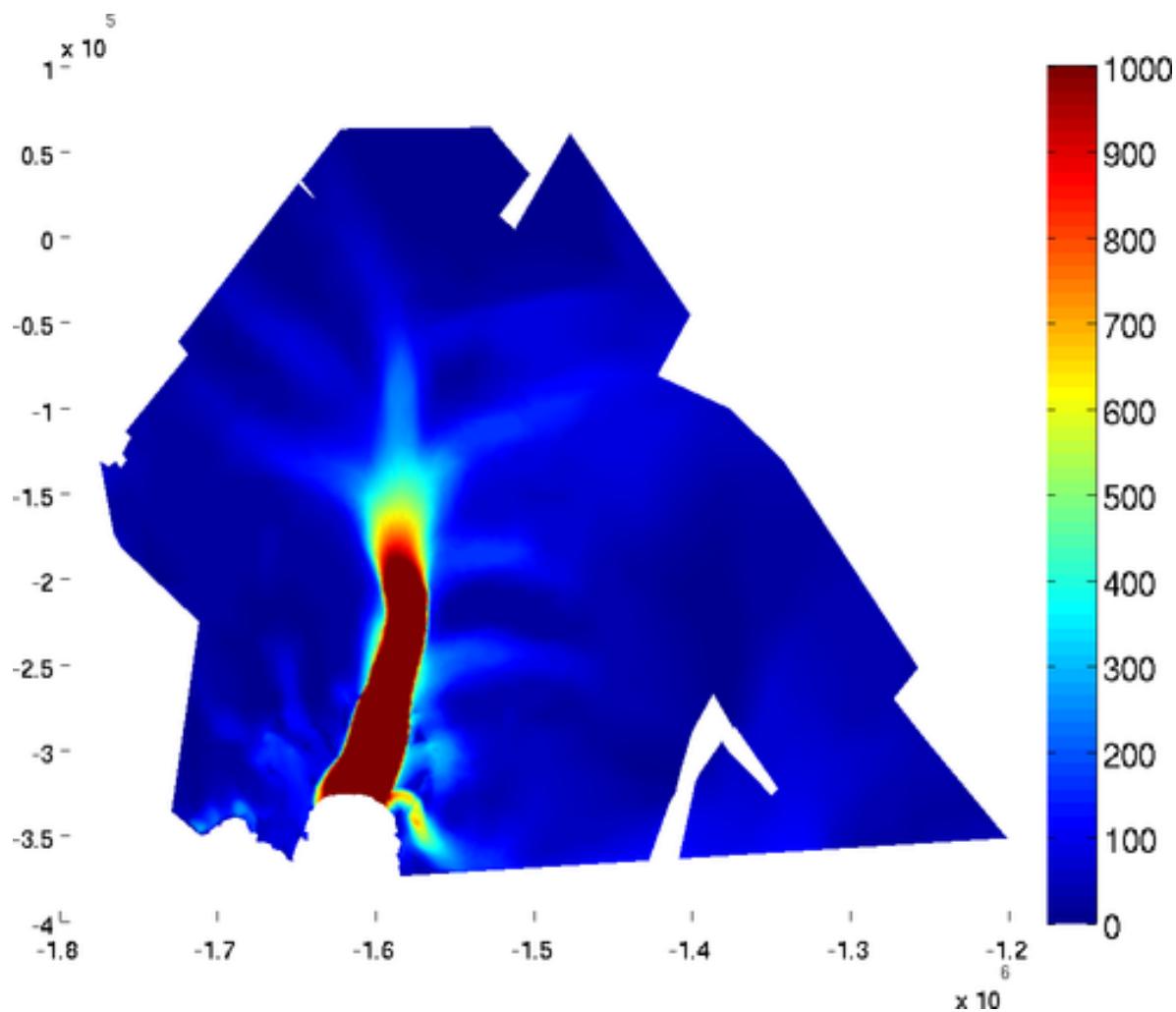
Same as standard `xlim` MATLAB option:

```
>> plotmodel(md, 'data', md.vel, 'xlim', [10^5 2*10^5])
```

caxis

Same as standard `caxis` MATLAB option (control the extreme values of the colorbar):

```
>> plotmodel(md, 'data', md.vel, 'caxis', [0 1000])
```



colorbar

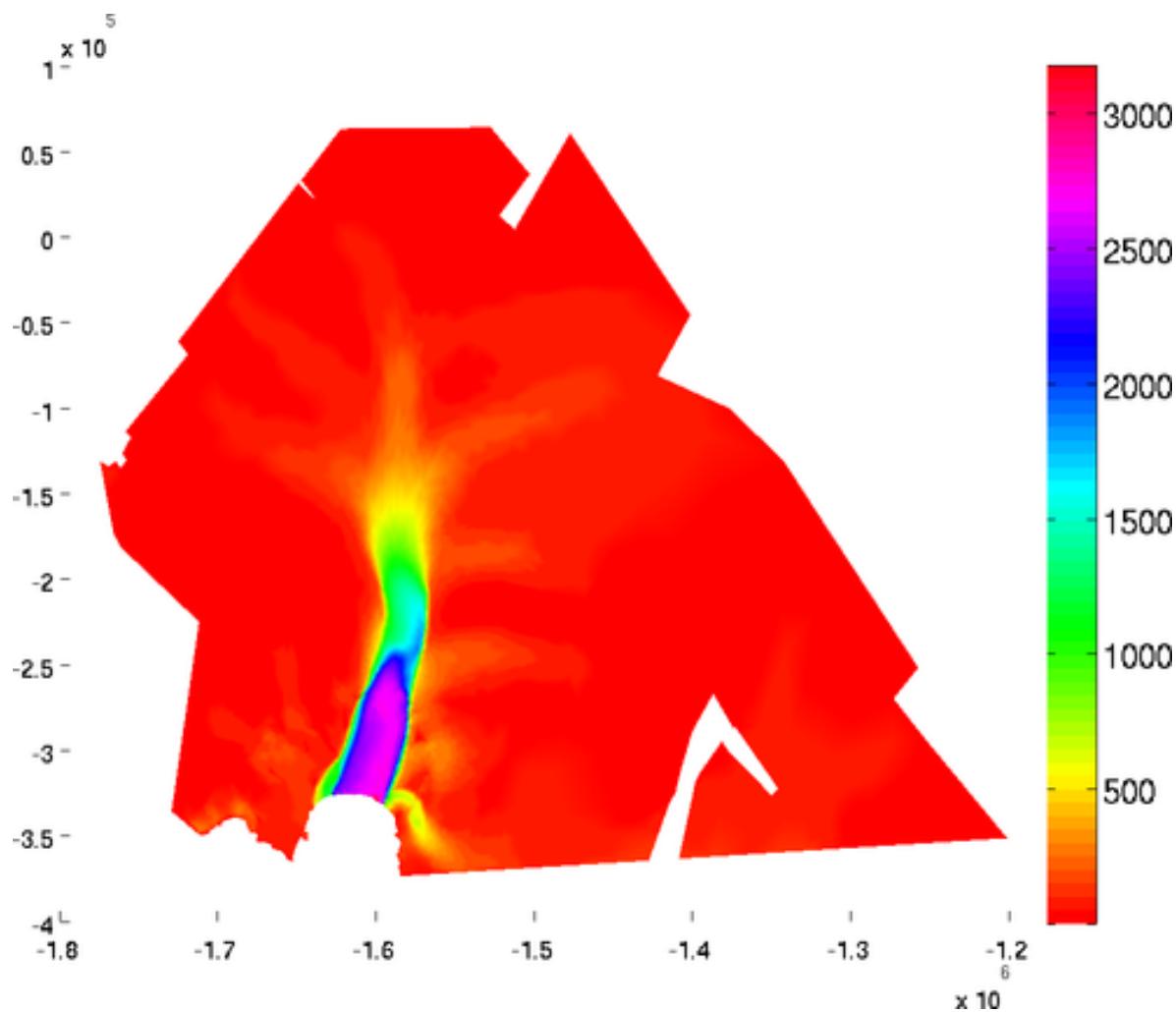
This option is used to control the colorbar display ('on' or 'off'):

```
>> plotmodel(md, 'data', md.vel, 'colorbar', 'off')
```

colormap

Same as standard colormap MATLAB option (control the extreme values of the colorbar):

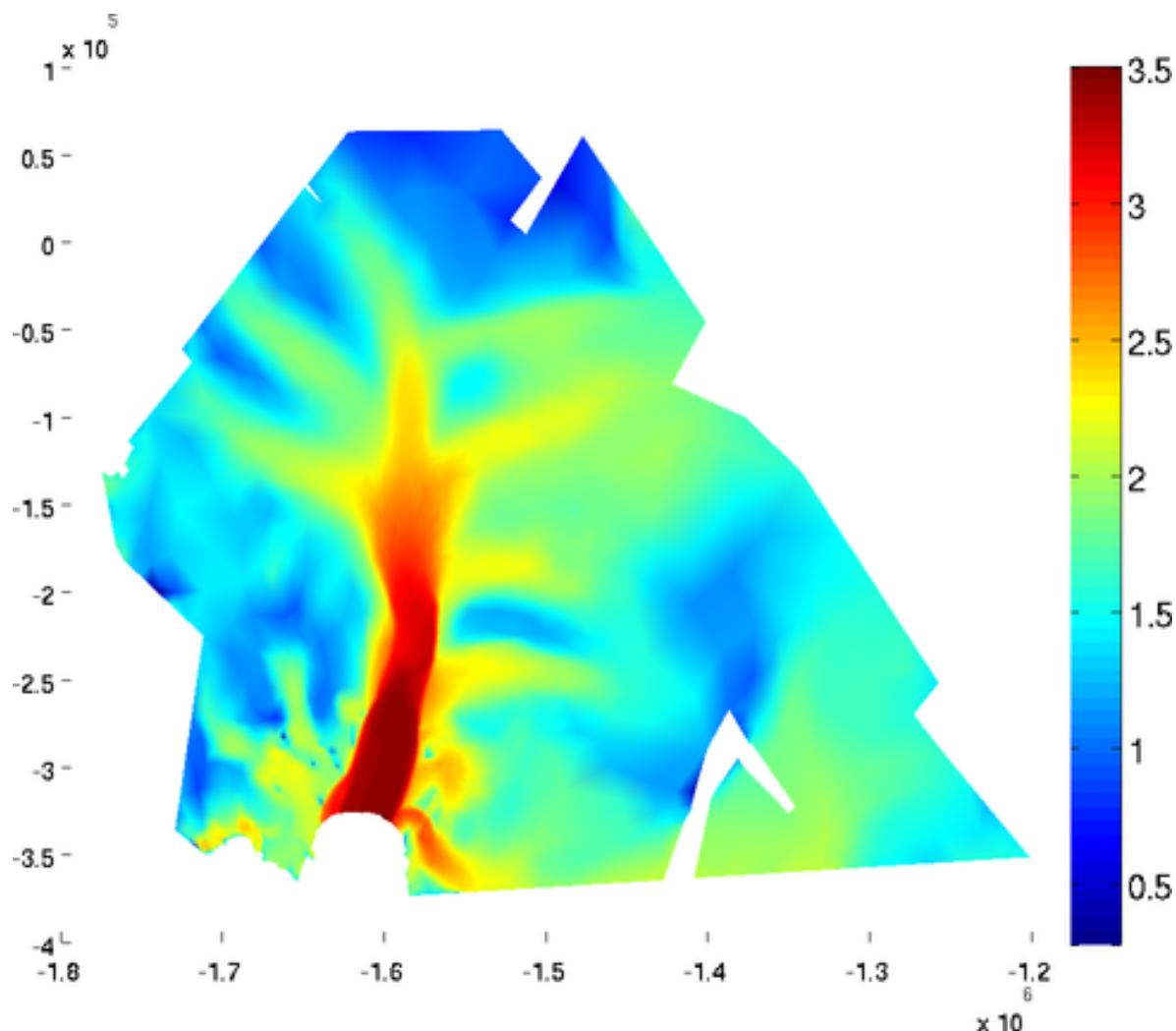
```
>> plotmodel(md, 'data', md.vel, 'colormap', 'hsv')
```



log

To get a logarithmic colorbar, use the 'log' option followed by `10` for a decimal logarithm:

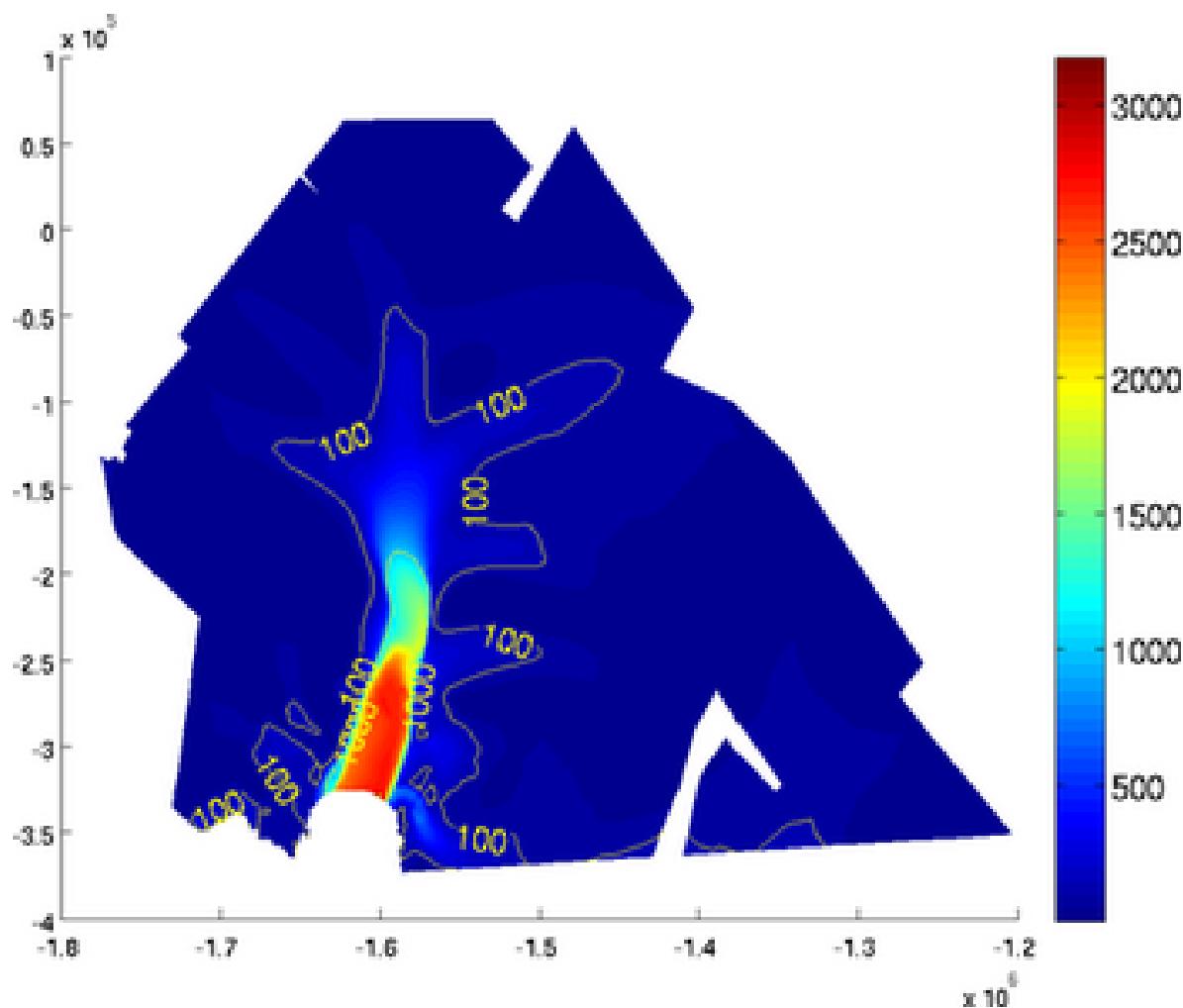
```
>> plotmodel(md, 'data', md.vel, 'log', 10)
```



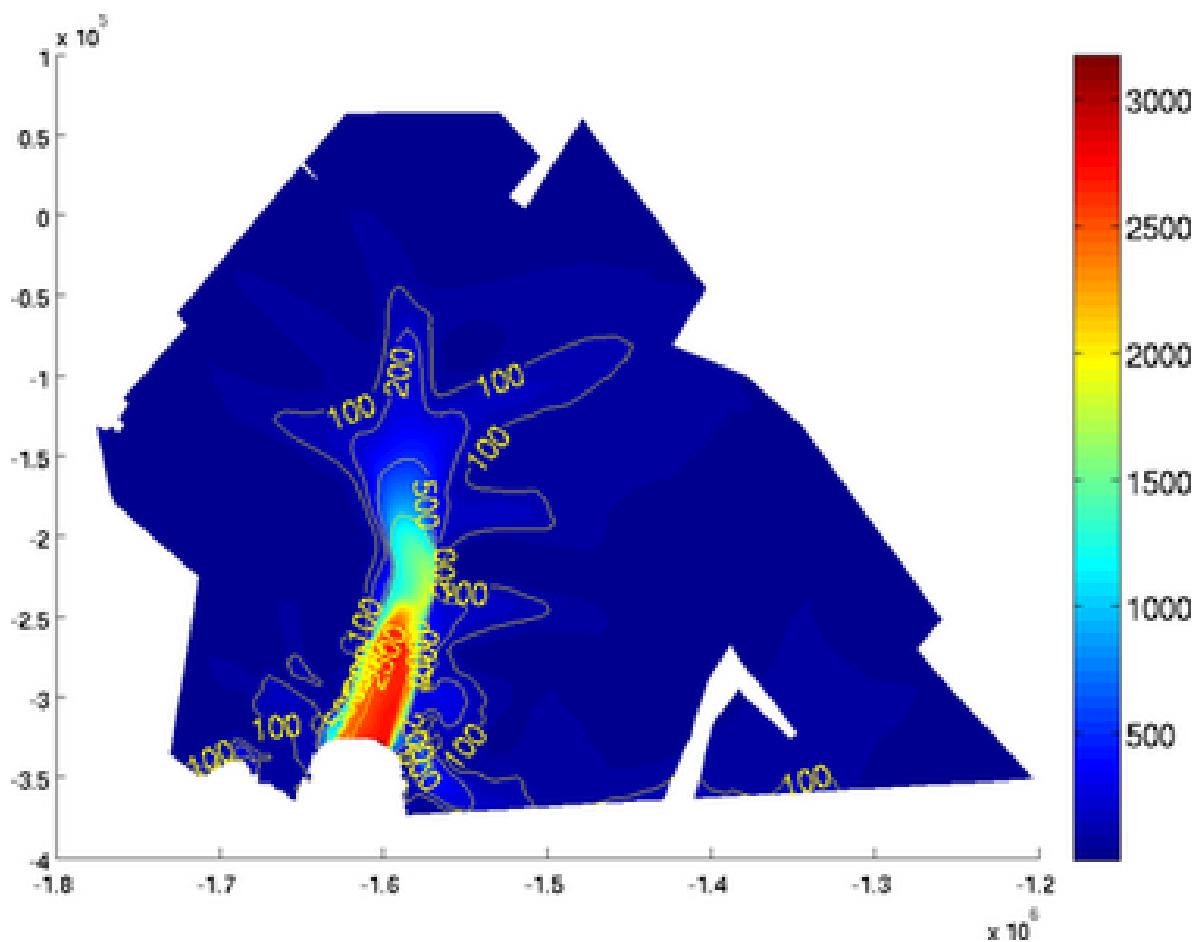
contourlevels

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. For example:

```
>> plotmodel(md, 'data', md.vel, 'contourlevels', 3)
```



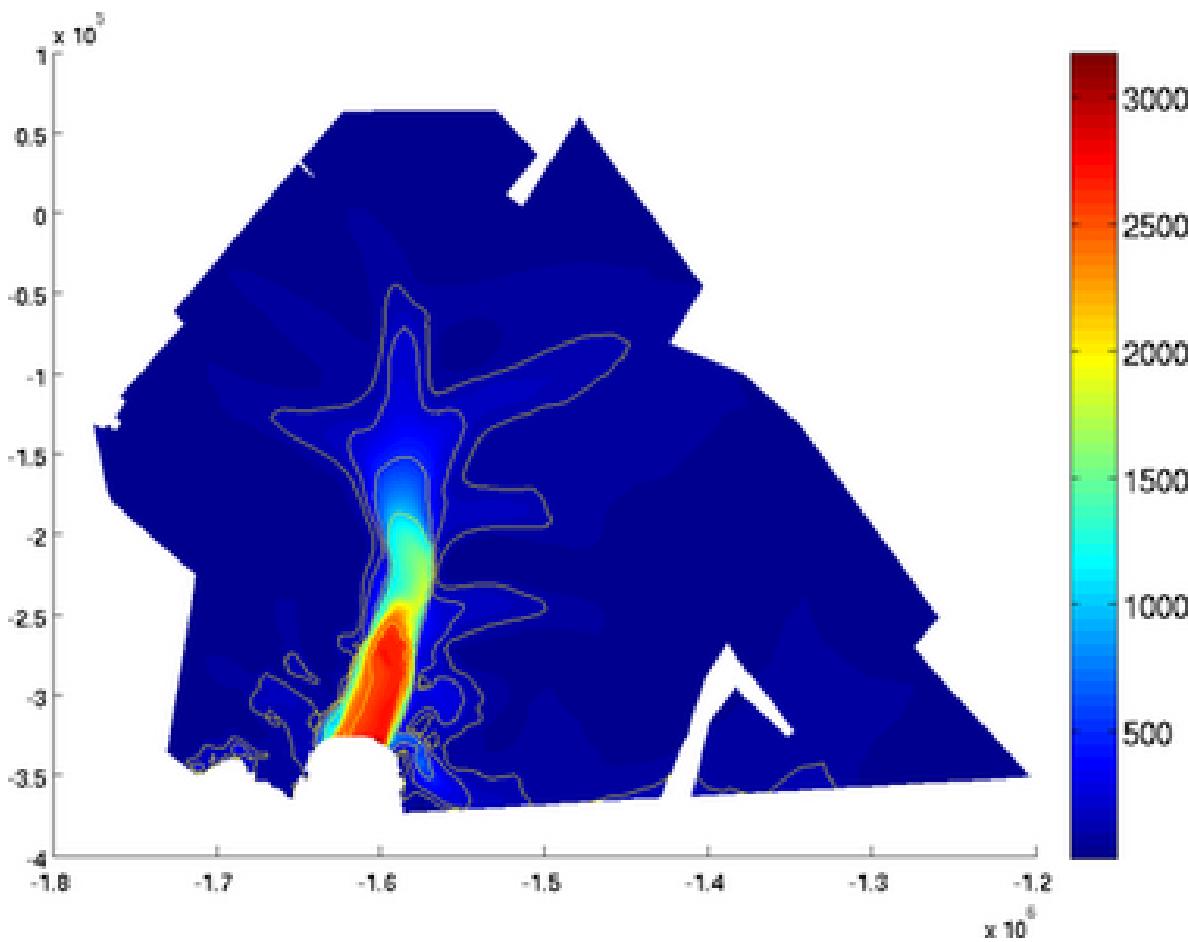
```
>> plotmodel(md, 'data', md.vel, 'contourlevels', {100, 200, 500,
1000, 2000, 2500})
```



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')
```



contouronly

If the user wants to display the contours only, use the 'contouronly' set as 'on' :

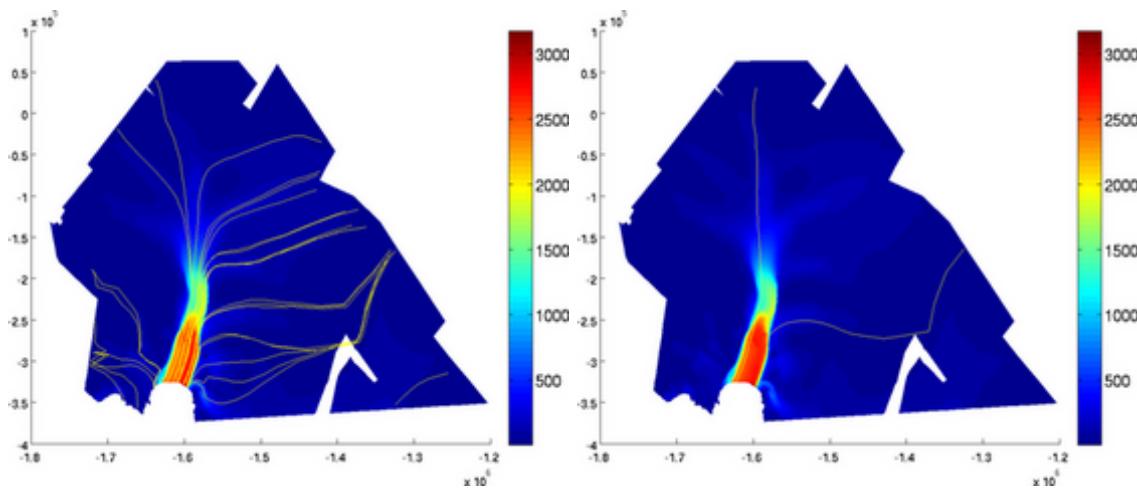
```
>> plotmodel(md, 'data', 'vel', 'contourlevels', {100, 200, 500,
1000, 2000, 2500}, 'contouronly', 'on')
```

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]})
```

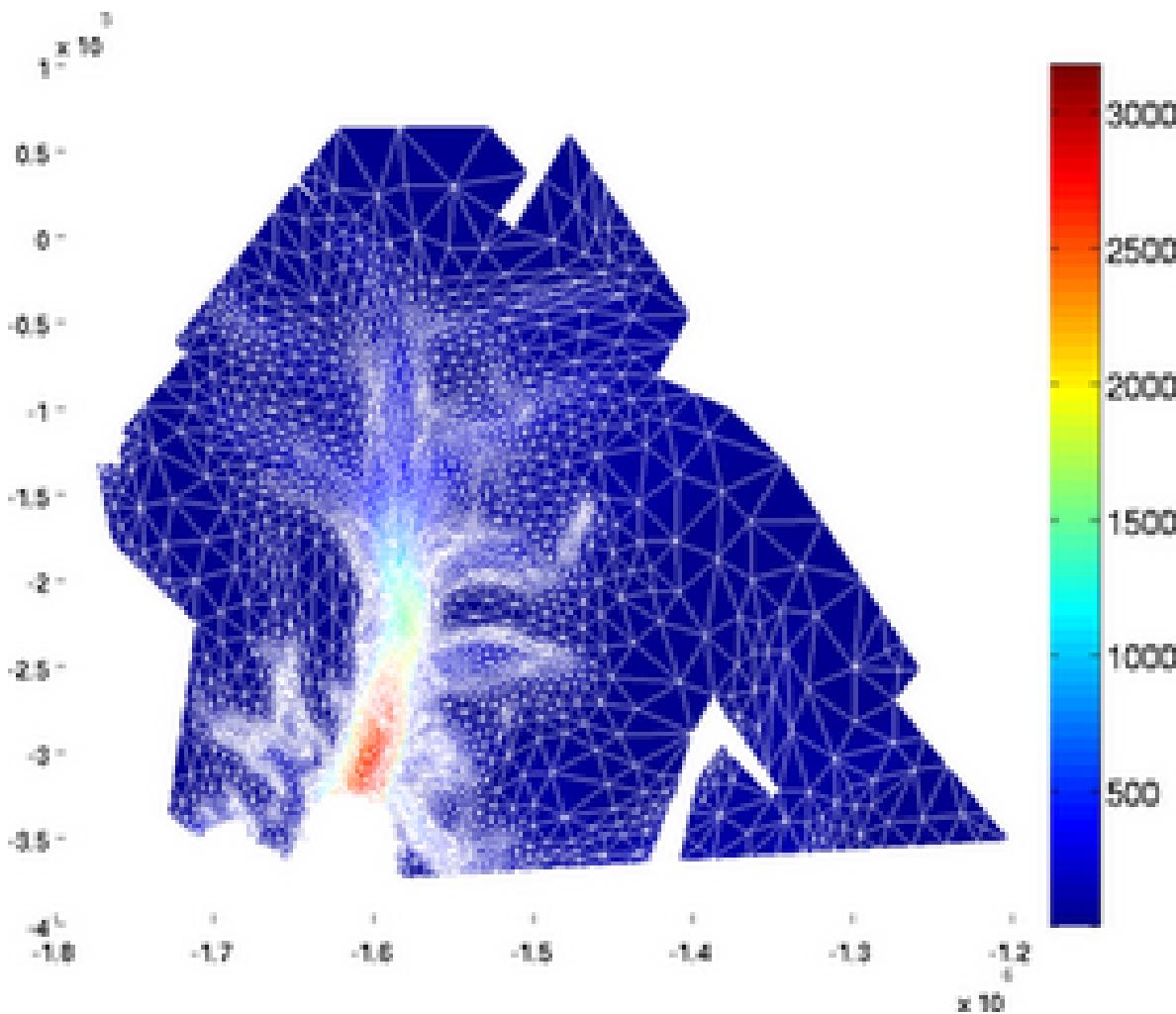


NOTE: Streamlines use the velocities that are in `md.initialization`. Make sure you transfer the calculated velocities to `md.initialization` if you want to display the calculated streamlines.

edgecolor

The mesh can be superimposed onto the plot by using the '`edgecolor`' option followed by a color:

```
>> plotmodel(md, 'data', md.initialization.vel, 'edgecolor', 'w')
```



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')
```

expstyle

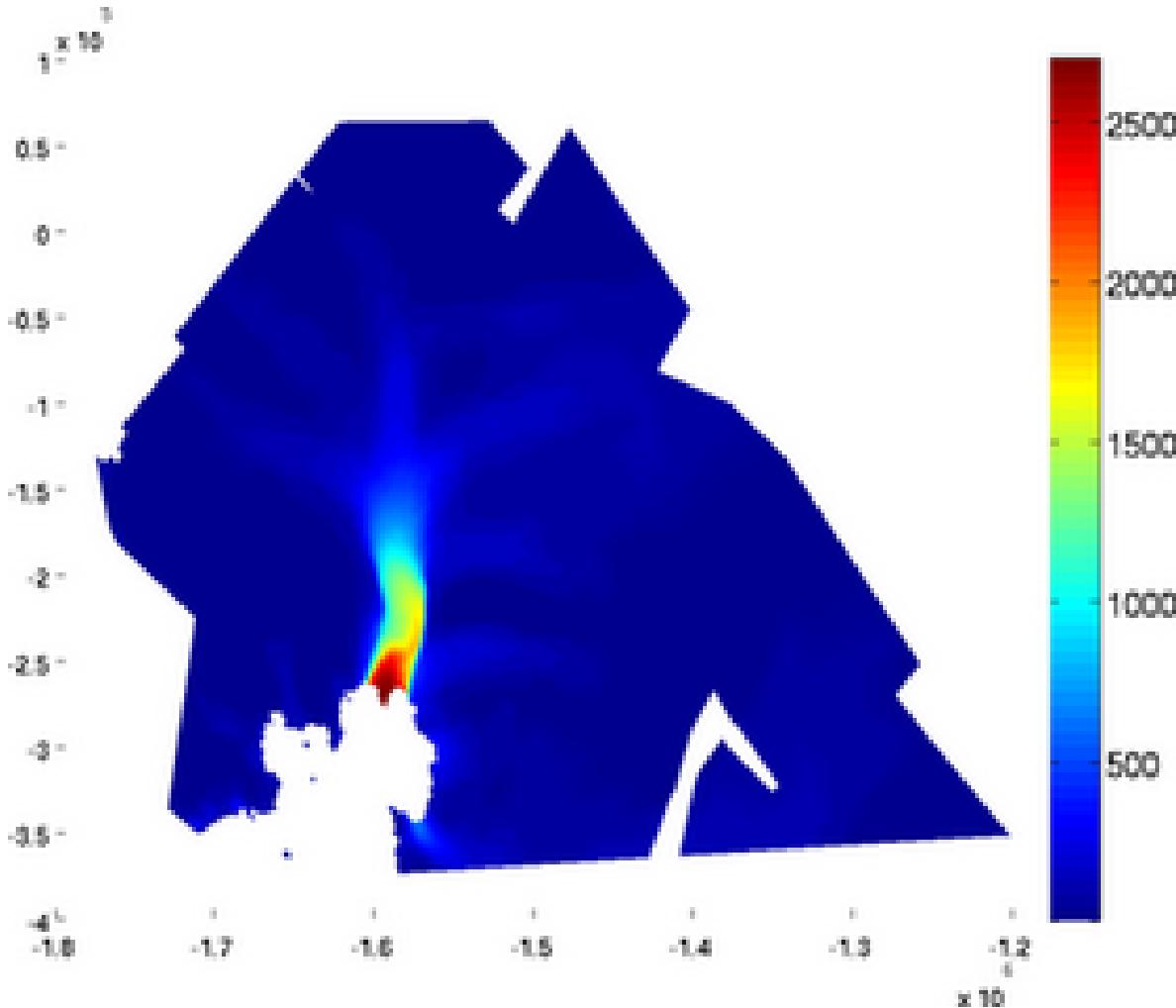
The style of the ARGUS profile can be controlled with the '`expstyle`' option, followed by the desired line style. Here is an example for a yellow dotted line:

```
>> plotmodel(md, 'data', md.initialization.vel, 'expdisp',
    'Iceshelves.exp', 'expstyle', '--y')
```

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.ocean_levelset < 0)
```

**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')
```

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')
```

title

Same as standard [title](#) MATLAB option:

```
>> plotmodel(md, 'data', md.vel, 'title', 'Ice velocity [m/yr]')
```

fontsize

Same as standard [fontsize](#) MATLAB option:

```
>> plotmodel(md, 'data', md.vel, 'title', 'Ice velocity [m/yr]',  
    'fontsize', 8)
```

fontweight

Same as standard [fontweight](#) MATLAB option:

```
>> plotmodel(md, 'data', md.vel, 'title', 'Ice velocity [m/yr]',  
    'fontweight', 'b')
```

xlabel, ylabel

Same as standard [xlabel](#) MATLAB option:

```
>> plotmodel(md, 'data', md.vel, 'xlabel', 'x axis [m]')
```

3.5.1.3 Special plots**basaldrag**

The special plot '[basal_drag](#)' displays the norm of the basal drag friction in kPa following formula:

$$\tau_b = -k^2 N^r \|\mathbf{v}\|^{s-1} \mathbf{v}_b \quad (3.1)$$

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')
```

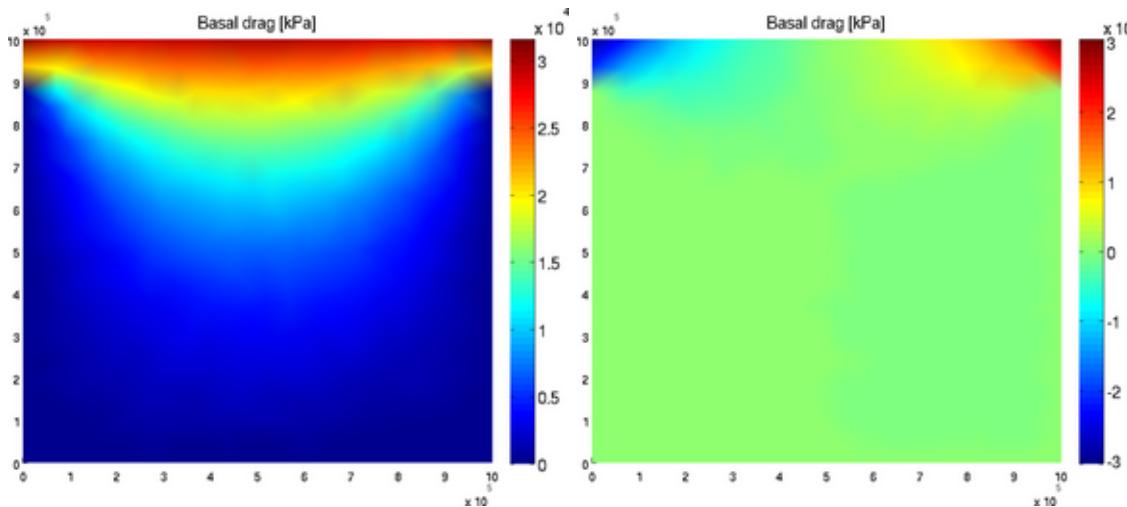
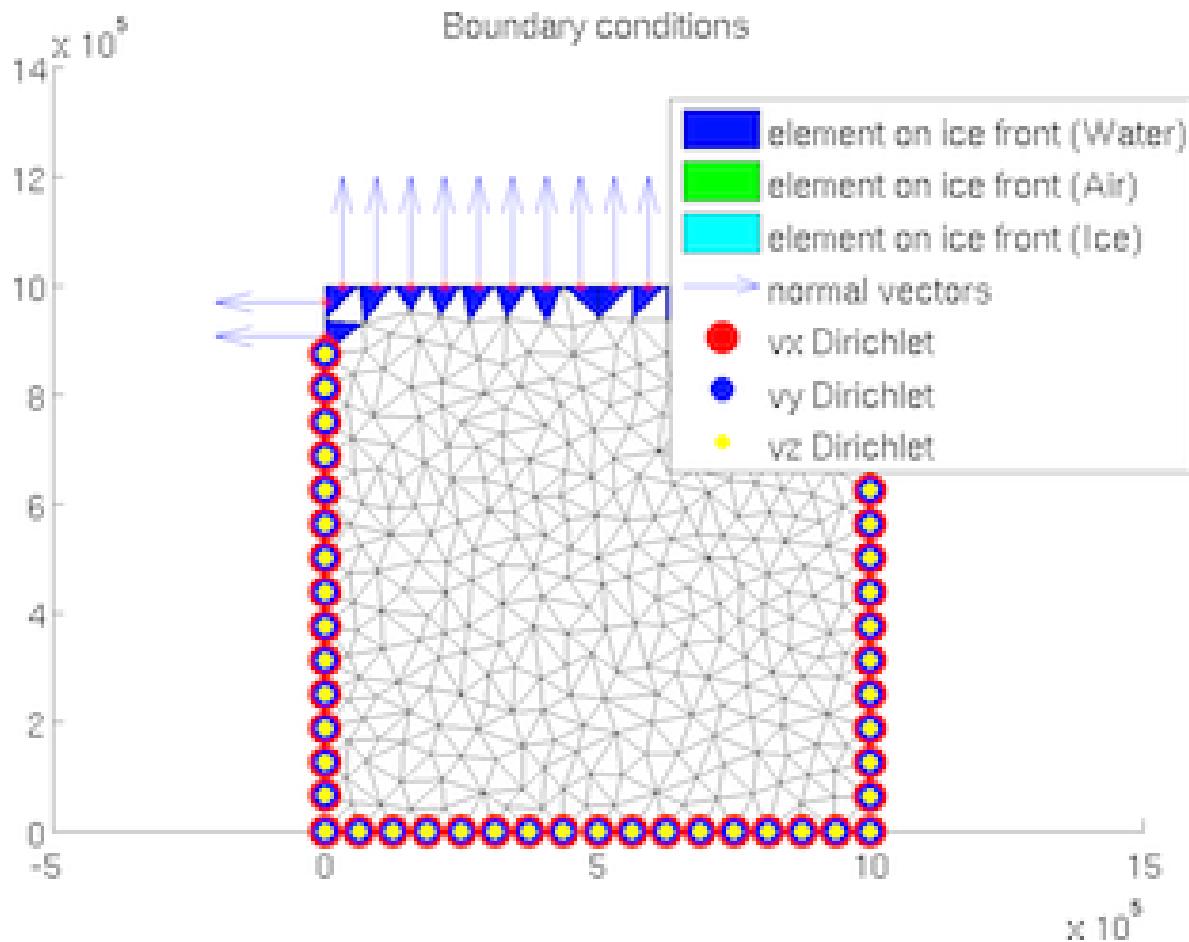


Figure 3.1: Basal friction norm and Basal friction x-component

BC

The special plot '`'BC'`' displays all boundary conditions (Newmann and Dirichlet) for 2D and 3D meshes:

```
>> plotmodel(md, 'data', 'BC')
```

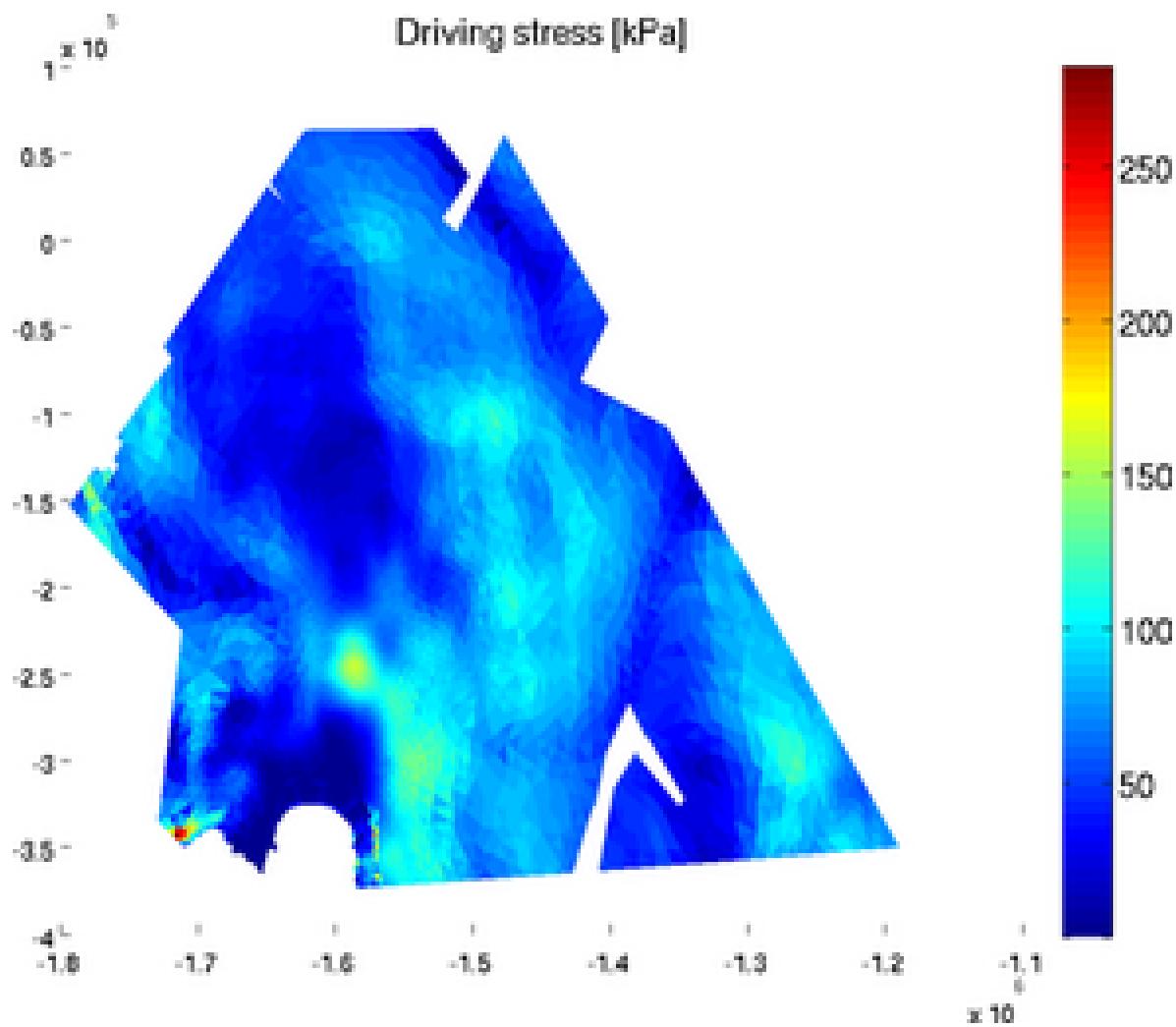


driving_stress

The special plot '`driving_stress`' displays the basal drag friction in kPa following formula:

$$\tau_d = \rho g H \nabla s \quad (3.2)$$

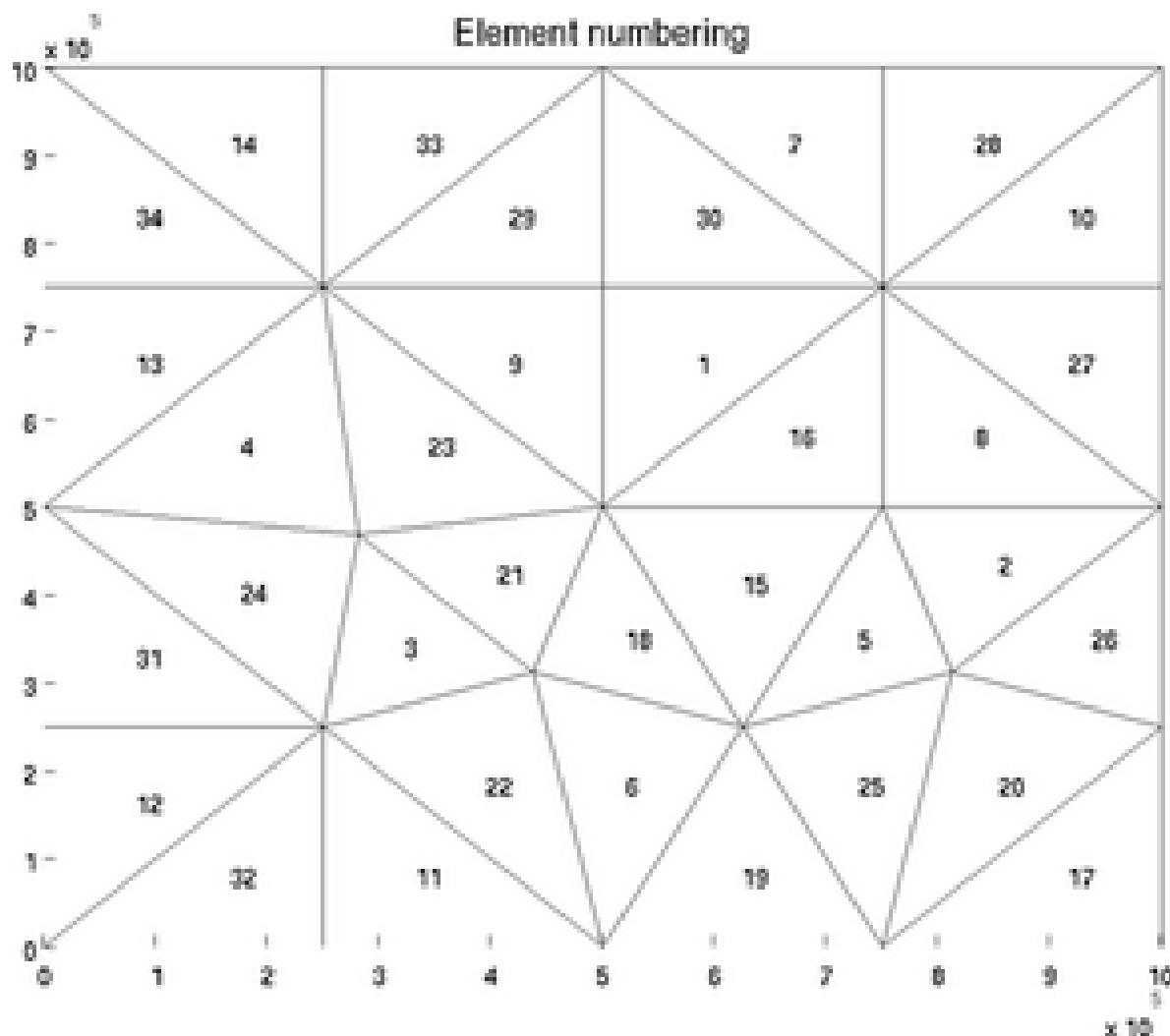
```
>> plotmodel(md, 'data', 'driving_stress')
```



elementnumbering

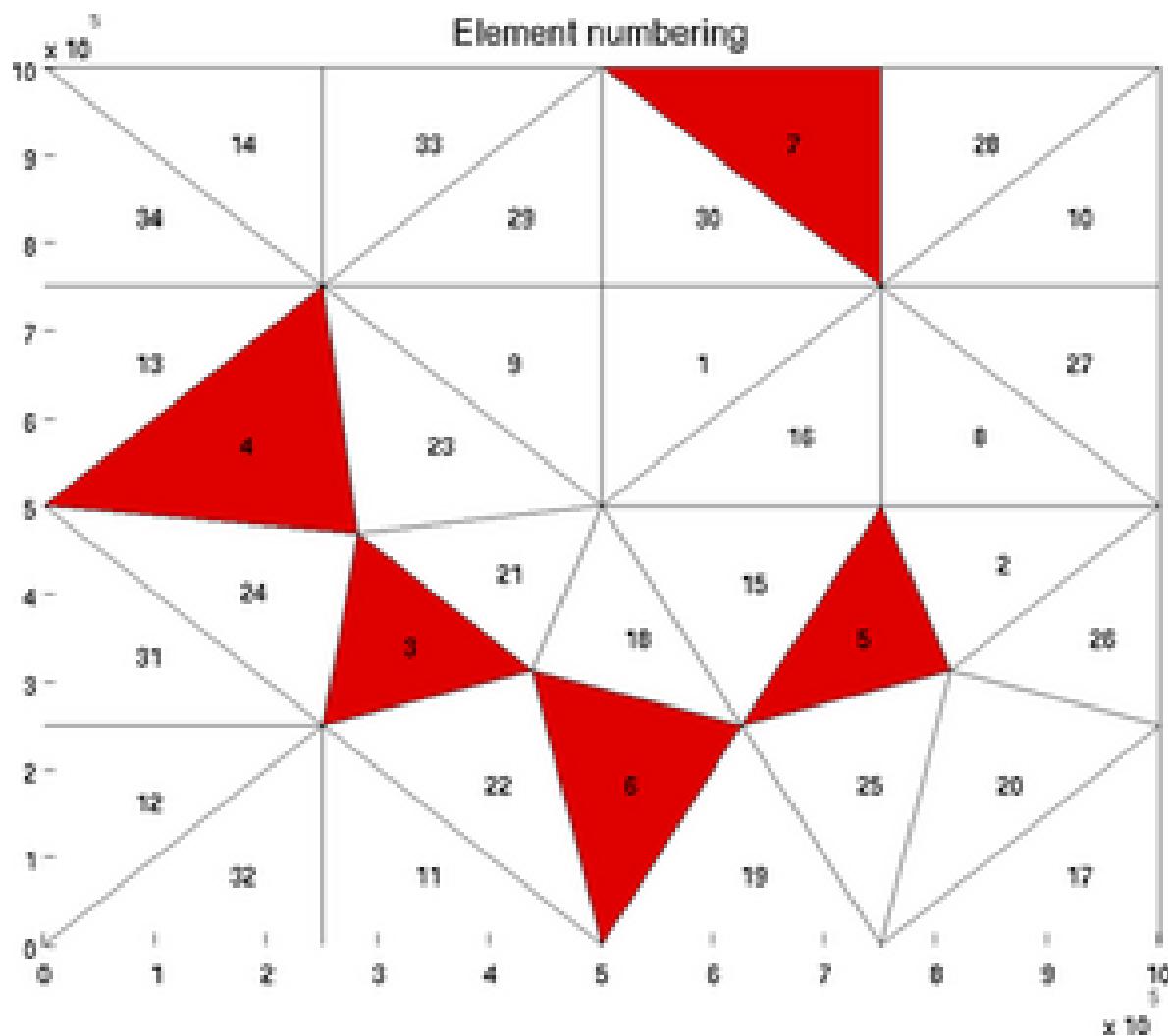
In the debugging process, it is often very useful to display all the elements next to their numbers. This is what the special plot 'elementnumbering' does:

```
>> plotmodel(md, 'data', 'elementnumbering')
```



A given list of elements can be highlighted with the 'highlight' option:

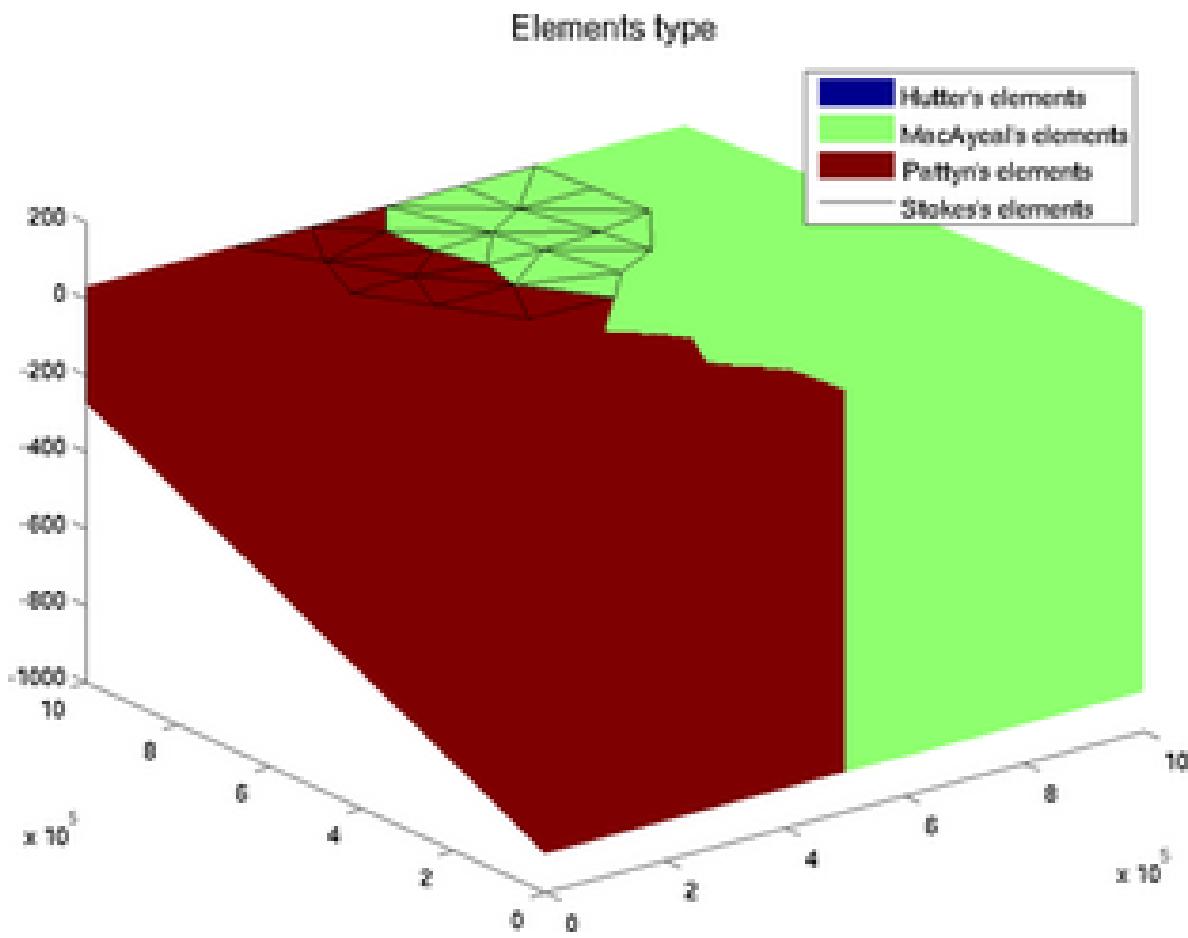
```
>> plotmodel(md, 'data', 'elementnumbering', 'highlight', [3 4 5 6  
7])
```



elements_type

The special plot 'elements_type' displays the elements with a specific color for each formulation:

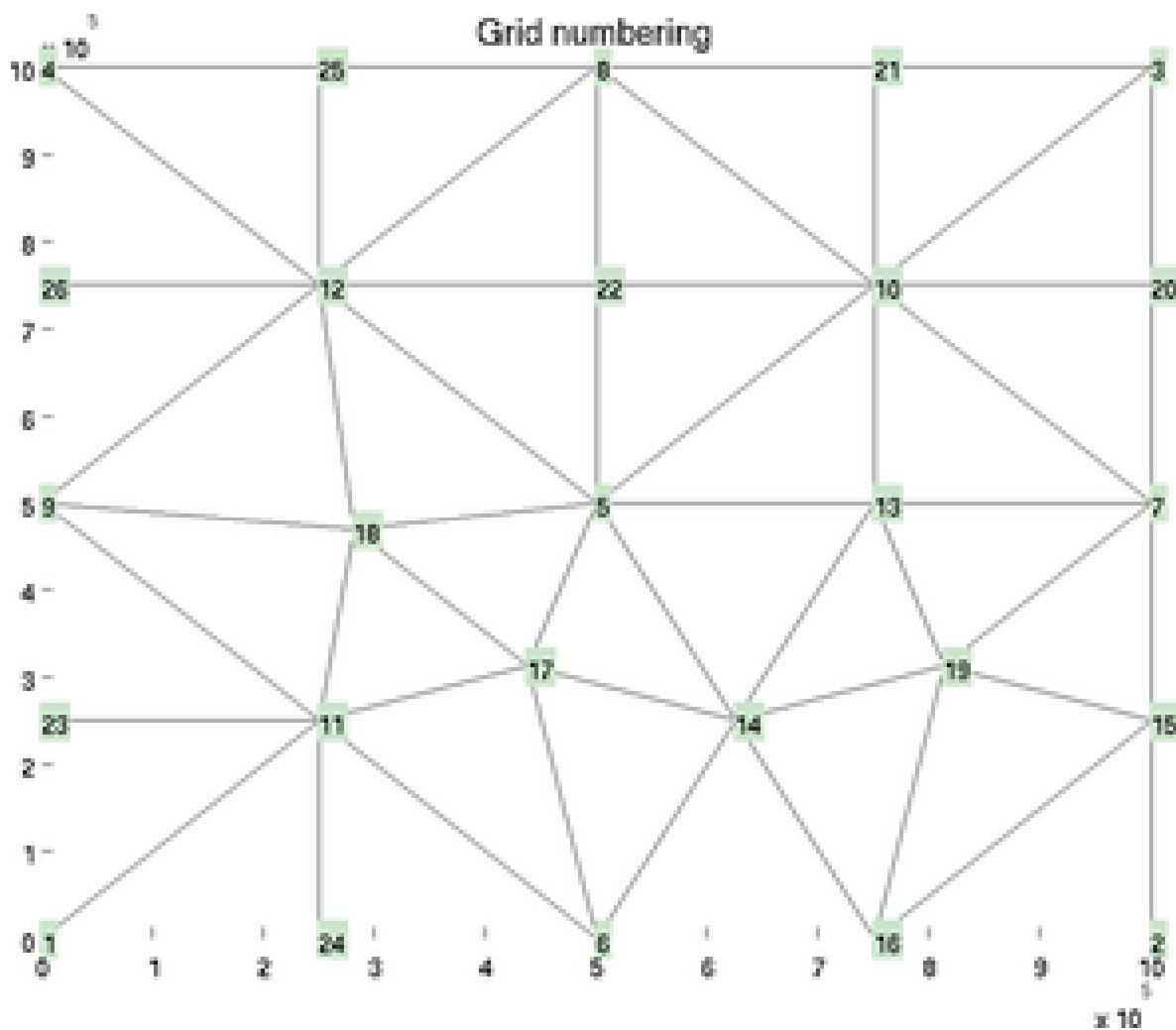
```
>> plotmodel(md, 'data', 'elements_type')
```



vertexnumbering

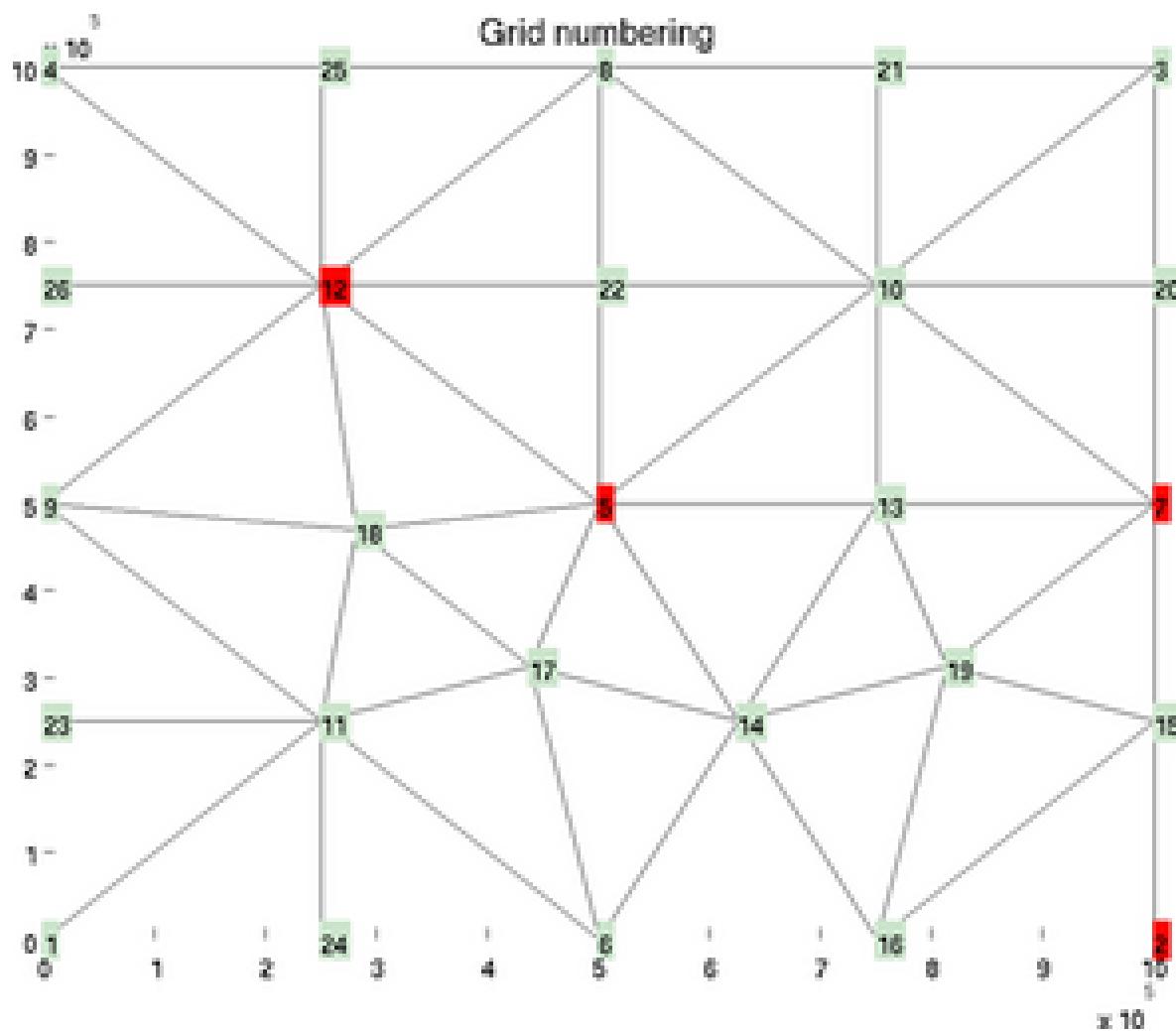
In the debugging process, it is often very useful to display all the vertices next to their numbers. This is what the special plot 'vertexnumbering' does:

```
>> 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])
```

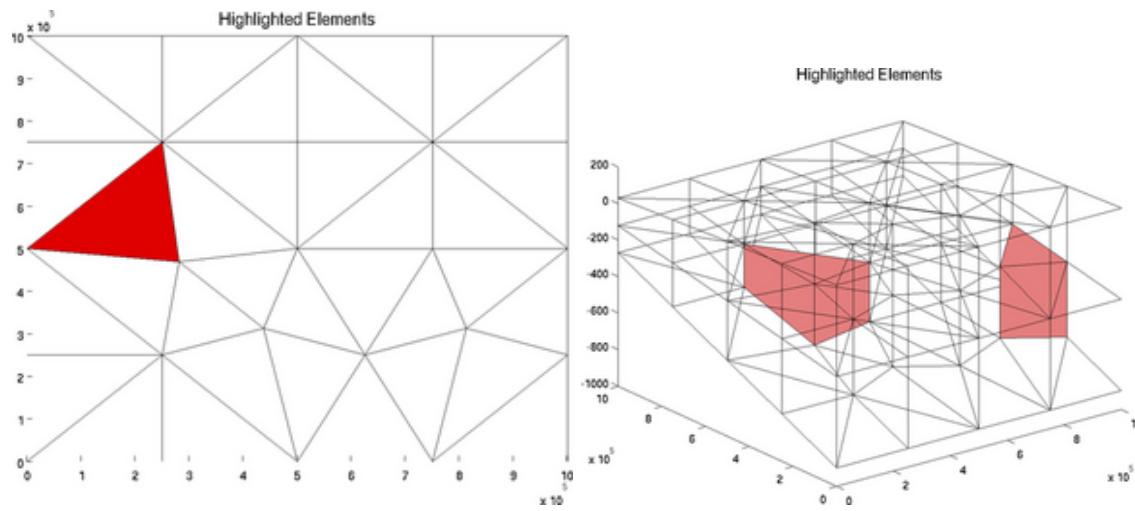


highlightelements

The special plot 'highlightelements' is very similar to the plot 'elementnumbering'. It is another possibility to highlight one or several grids, but without indicating the number of all the elements. It is a lot faster for large models:

```
>> plotmodel(md, 'data', 'highlightelements', 'highlight', 5)
```

```
>> plotmodel(md, 'data', 'highlightelements', 'highlight', [5 12])
```

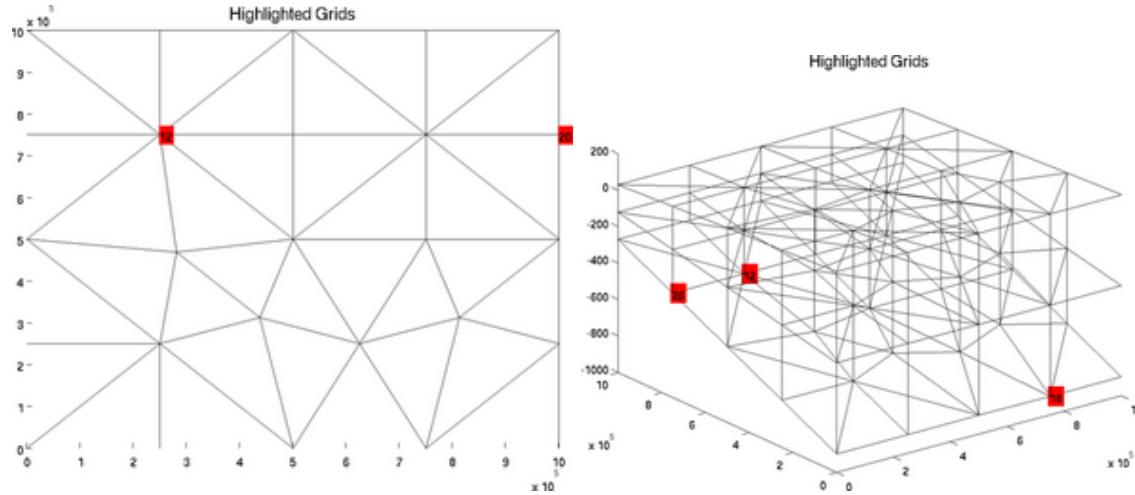


highlightgrids

The special plot 'highlightgrids' is very similar to 'gridnumbering'. It is another possibility to highlight grids without indicating all the grids numbers. It is a lot faster for big models:

```
>> plotmodel(md, 'data', 'highlightgrids', 'highlight', [12 20])
```

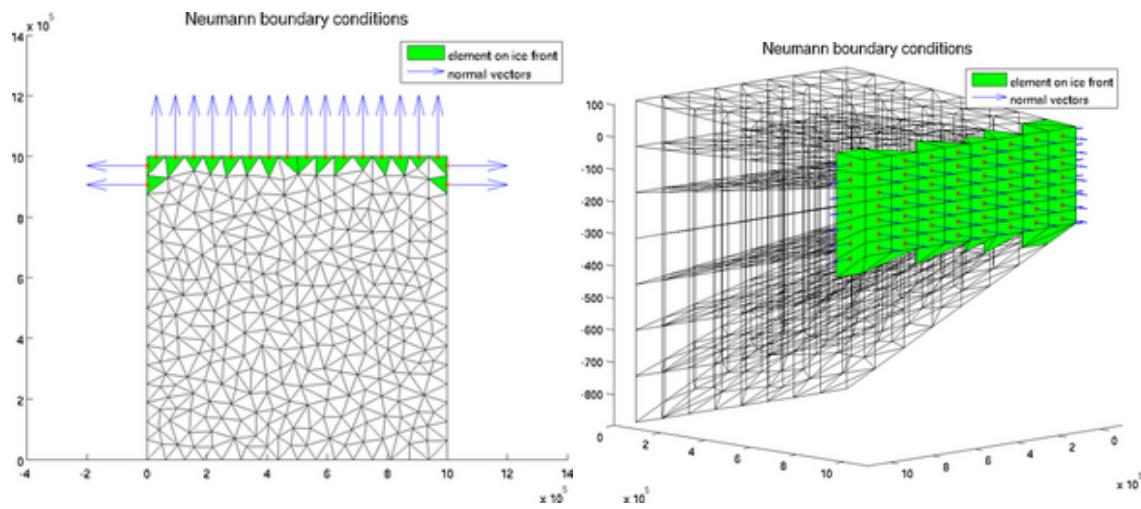
```
>> plotmodel(md, 'data', 'highlightgrids', 'highlight', [12 16 26])
```



icefront

The special plot 'icefront' displays the Neumann boundary conditions, i.e. all the segments on ice front and the normal to these segments, for a 2D or 3D mesh:

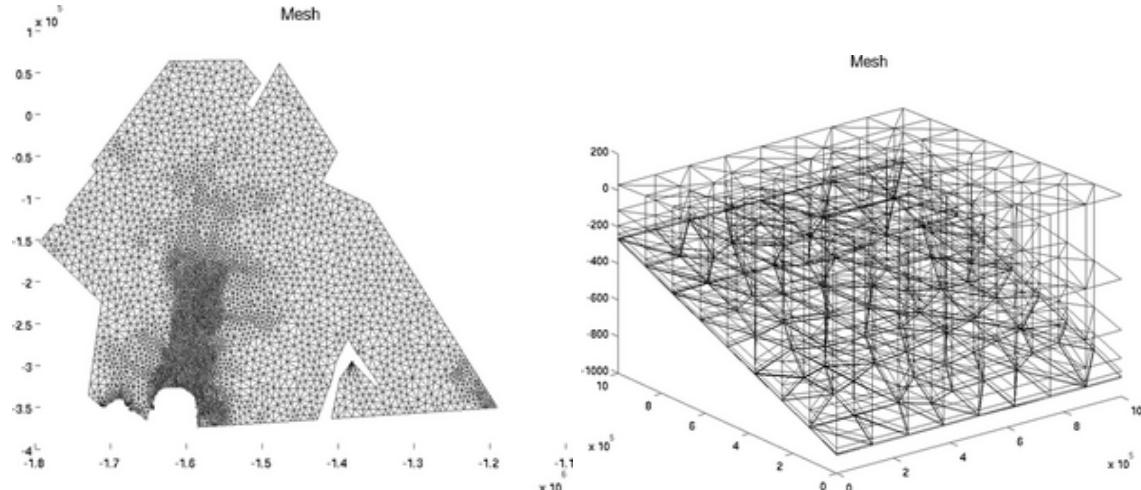
```
>> plotmodel(md, 'data', 'icefront')
```



mesh

The special plot 'mesh' displays the mesh of 2D or 3D model:

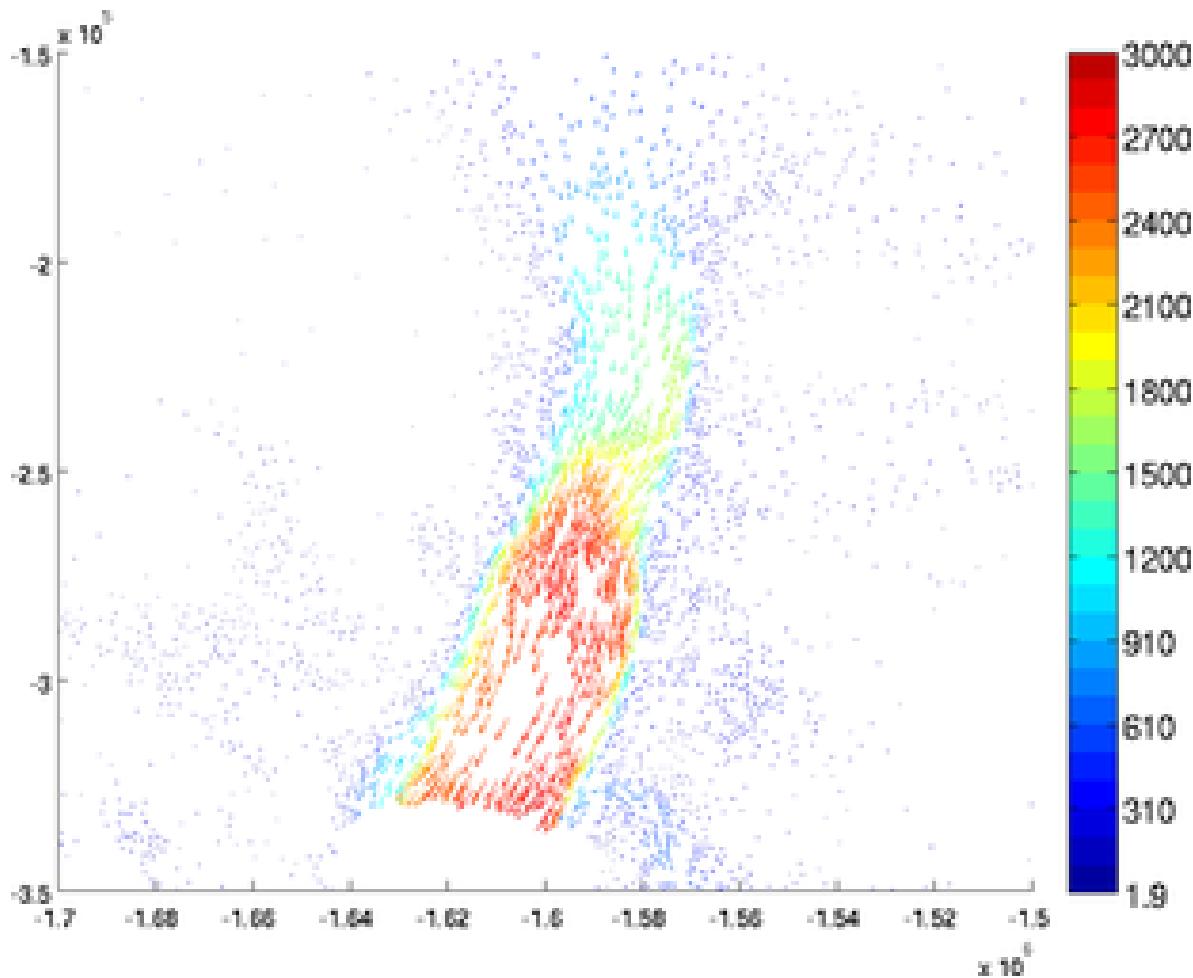
```
>> plotmodel(md, 'data', 'mesh')
```



3.5.1.4 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])
```

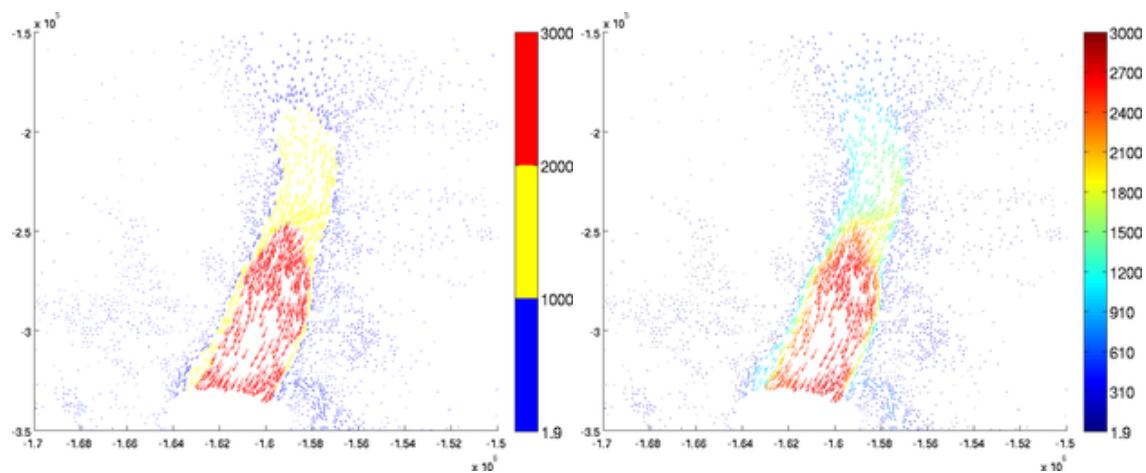


ColorLevels

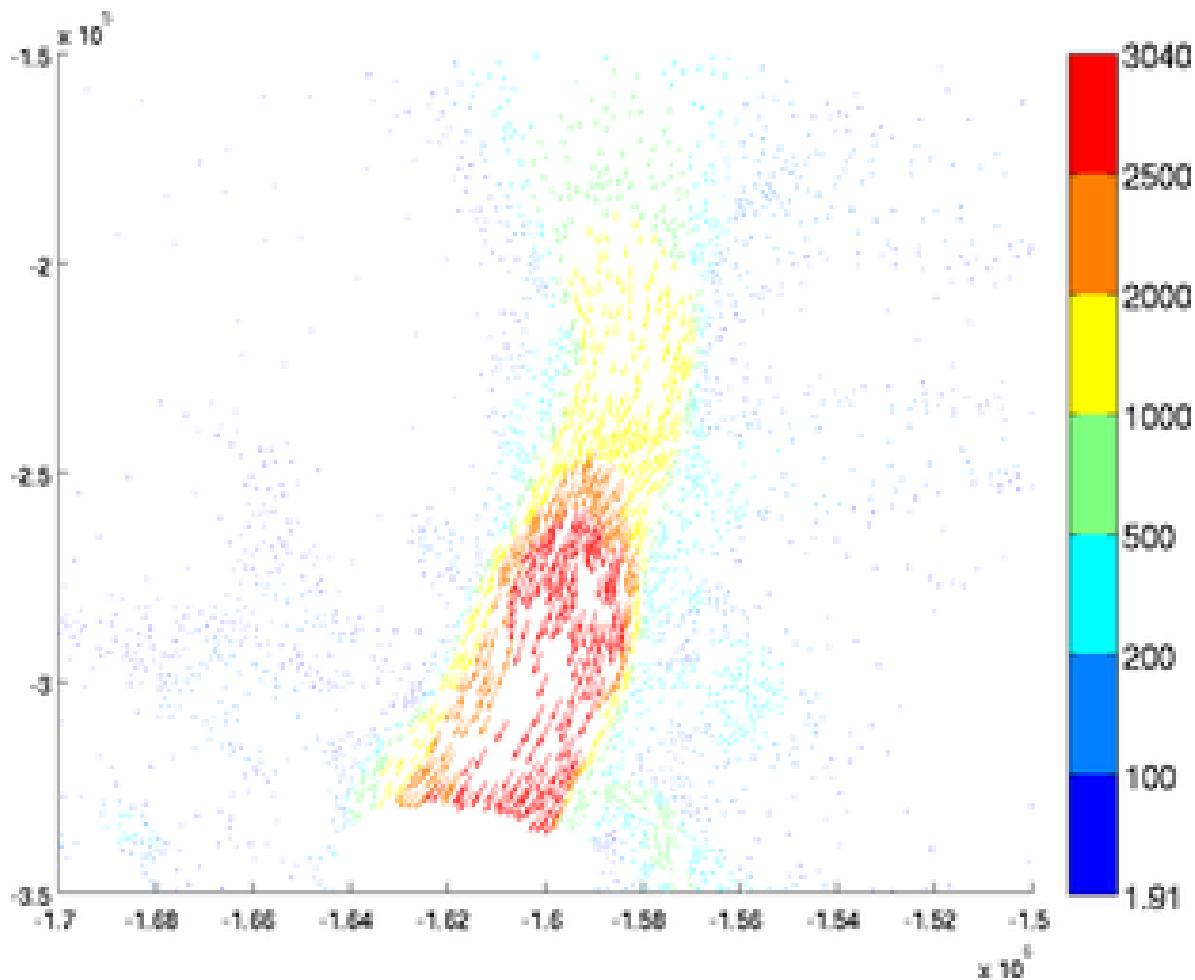
The number of colors can be chosen by using the `'colorlevels'` options. The user can specify a number of levels or a cell containing the values of color changes. For example:

```
>> 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})
```

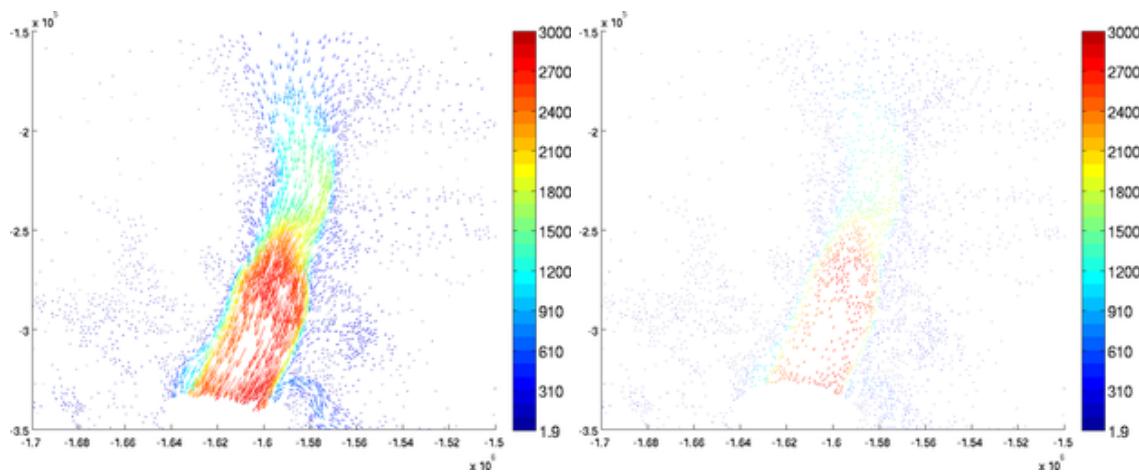


Scaling

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)
```

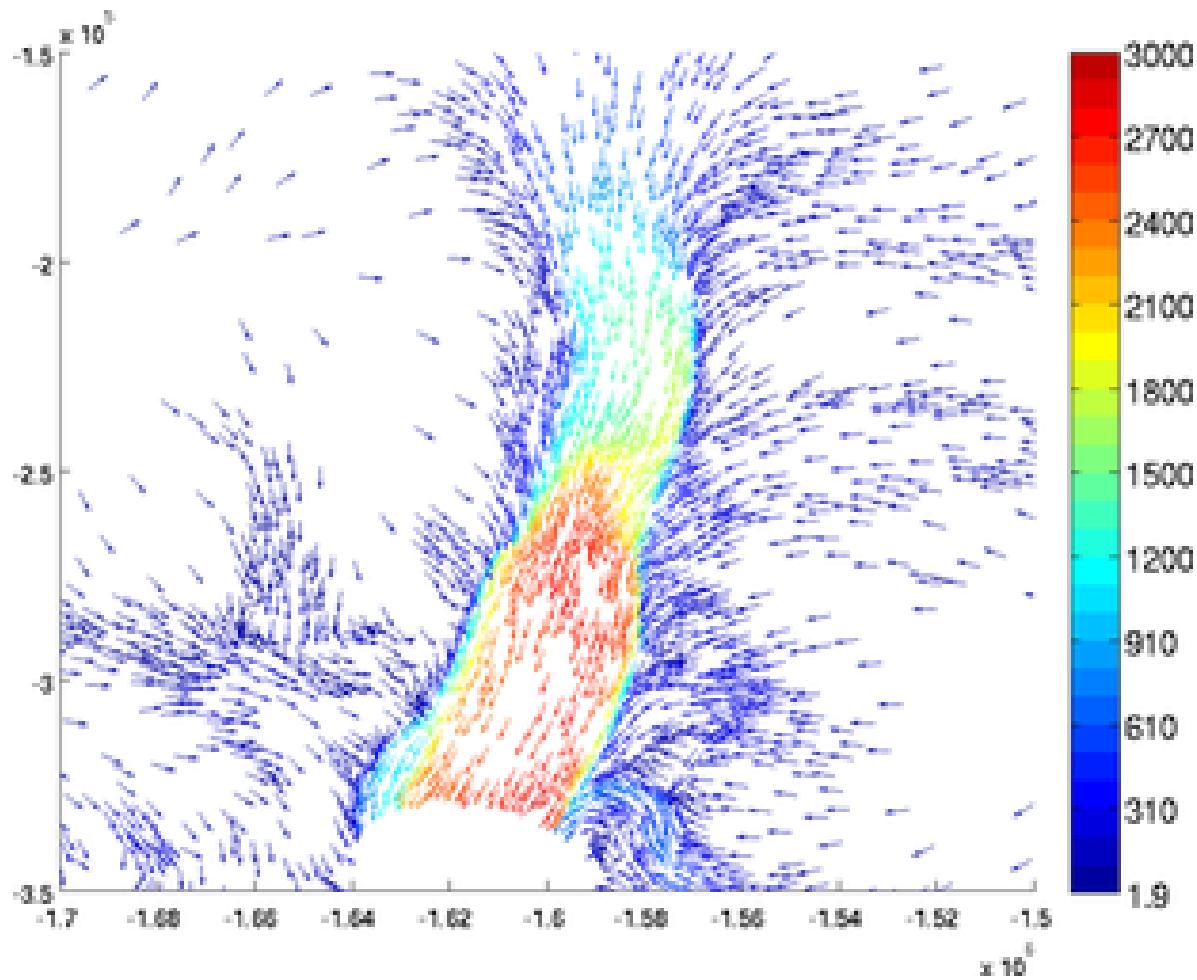
```
>> plotmodel(md, 'data', [md.vx md.vy], 'scaling', 0.1)
```



Autoscale

If the user wants all the arrows to have the same length, use the option 'autoscale' set as 'off' :

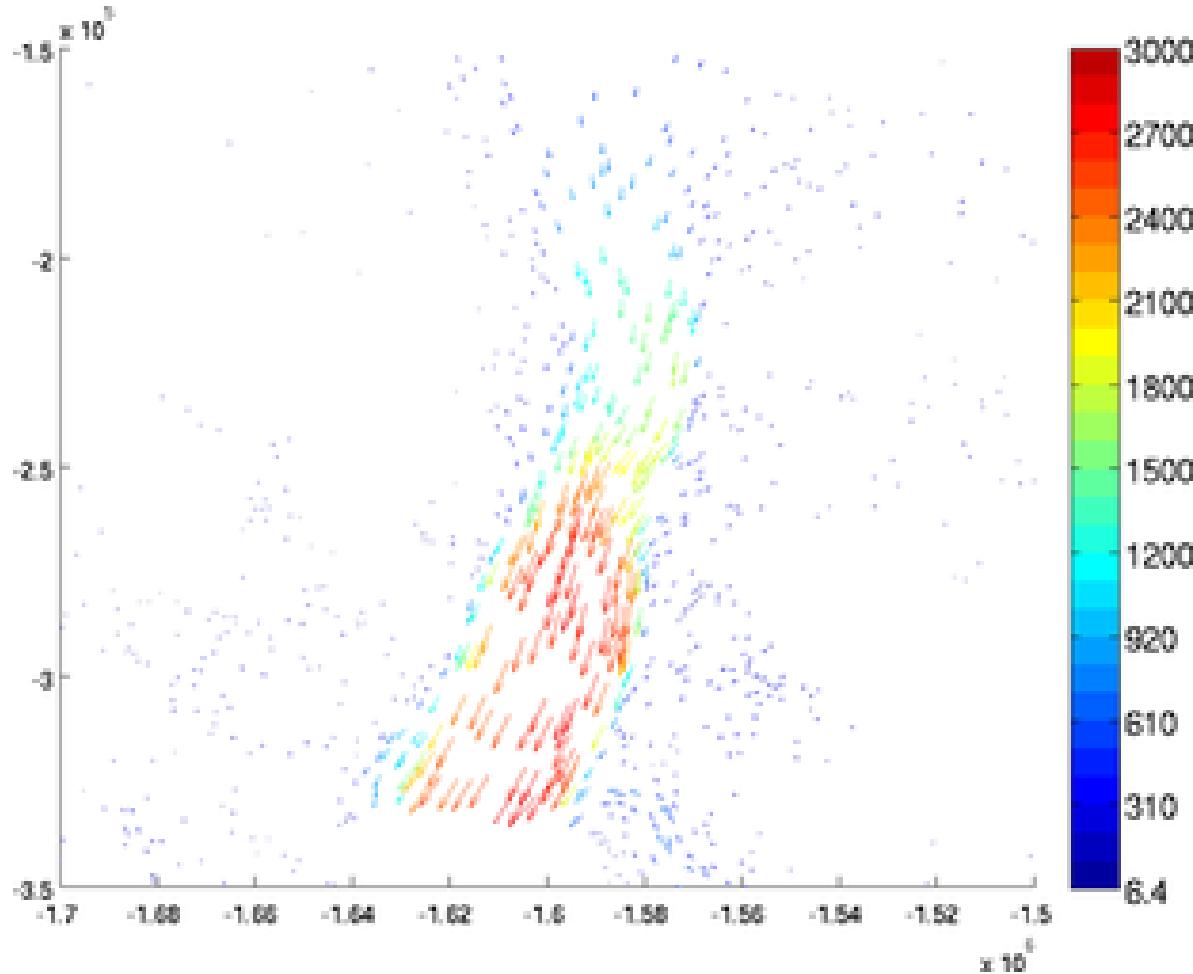
```
>> plotmodel(md, 'data', [md.vx md.vy], 'autoscale', 'off')
```



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)
```



3.5.1.5 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. For example:

```
>> plotmodel(md, 'data', md.vel, 'expdisp', 'track.exp')
```

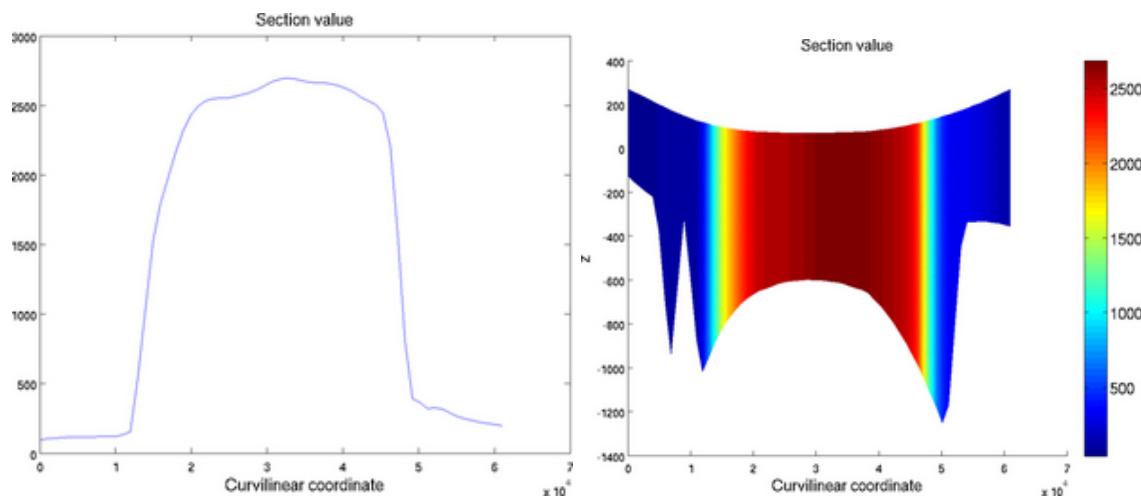
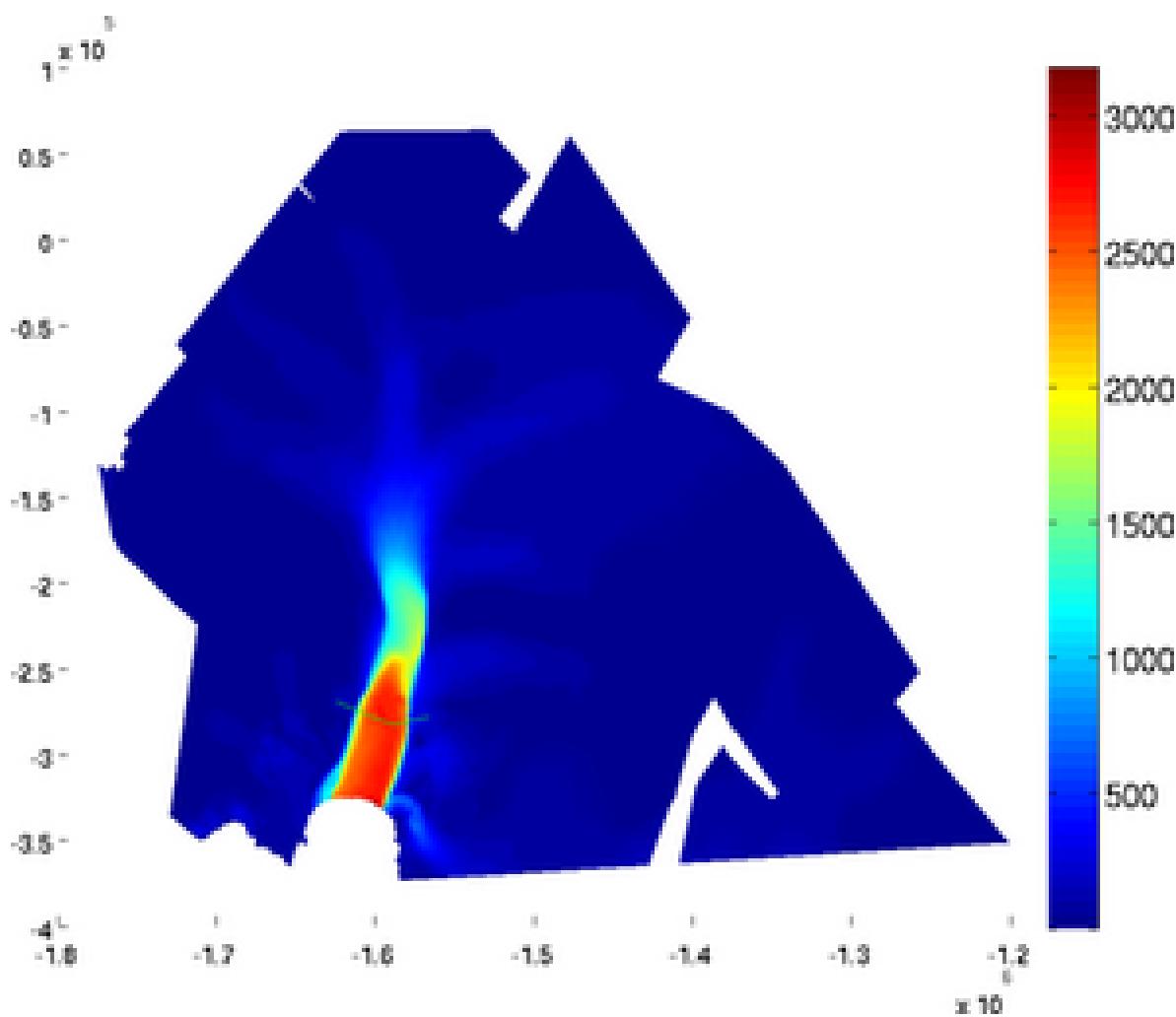


Figure 3.2: Section plot for 2D (left) and 3D (right) models

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:

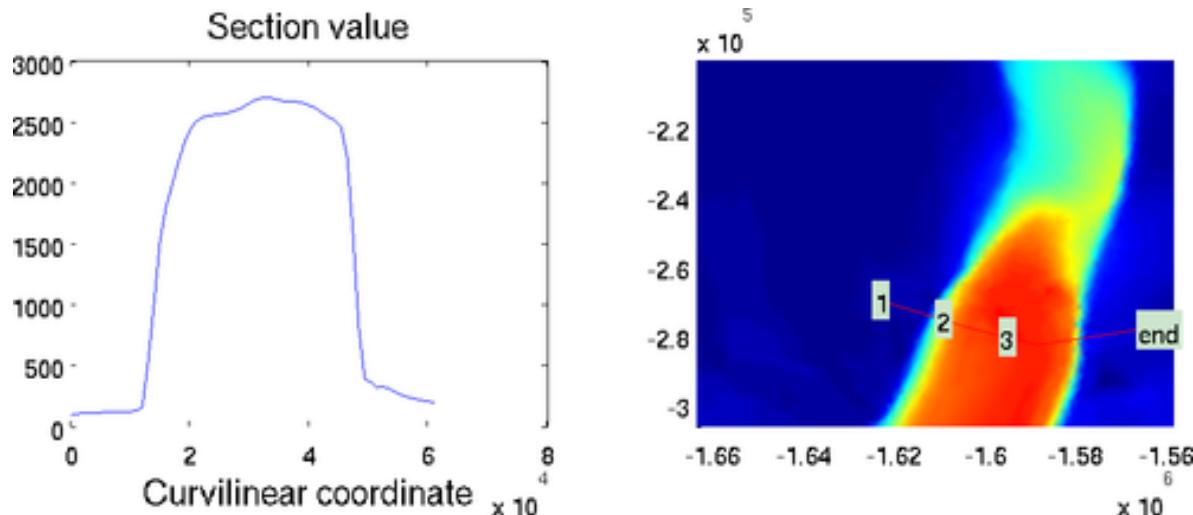
```
>> plotmodel(md, 'data', md.vel, 'sectionvalue', 'track.exp',
    'resolution', [2*10^4 0])
```

```
>> plotmodel(md, 'data', md.vel, 'sectionvalue', 'track.exp',
    'resolution', [10^3 0])
```

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')
```



3.5.2 Plotting in Python

We are currently working on a standard API for plotting ISSM results in a Python interface.

Chapter 4

Using ISSM

4.1 Introduction

- [Tutorials](#)
- [Capabilities](#)
- [Parameterization](#)
- [Advanced](#)

4.1.1 Introduction to Capabilities Video Series

Dr. Eric Larour and members of the ISSM development team have developed a series of videos to introduce viewers to practical uses of a number of ISSM's capabilities.

- [ISSM Tutorial 1 ↗](#)
- [ISSM Tutorial 2 ↗](#)
- [ISSM Tutorial 3 ↗](#)
- [ISSM Tutorial 4 ↗](#)
- [ISSM Tutorial 5 ↗](#)

4.2 Tutorials

In order to run the tutorials, you need to download some Datasets

- Square Ice Shelf
- Mesh Adaptation
- Ice Flow Models (under development)
- Ice Sheet Model Intercomparison Project (ISMIP) Tests
- Inversions
- Modeling the Greenland Ice Sheet (SeaRISE)
- Modeling the Greenland Ice Sheet Using IceBridge Data
- Modeling Pine Island Glacier
- Pine Island Glacier Sensitivity Study
- Pine Island Glacier Uncertainty Quantification (requires Dakota)
- Pine Island Glacier Stochastic Forcing (StISSM)
- Modeling Jakobshavn Isbrae
- Subglacial Channel Formation From a Single Moulin (SHAKTI)
- Modeling Helheim Glacier
- Subglacial Hydrology of Helheim Glacier (SHAKTI)
- Adaptive Mesh Refinement (AMR)
- Sea-Level Fingerprints (GRACE)

4.2.1 Datasets

To run the tutorials, you will need to download a number of datasets. The easiest way to do this is to run,

```
${ISSM_DIR}/scripts/DownloadExamplesDatasets.sh
```

The default behavior of this script is to download the datasets to the `../examples/Data` relative to the location of the script. If you wish to download them to another location, supply the optional path argument.

If you would prefer to download the datasets manually, use the following links, downloading each file to `${ISSM_DIR}/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 Isbræ bed map [↗](#)
(we only need `grids/Jakobshavn_2008_2011_Composite_XYZGrid.txt`)
- GRACE and supporting datasets for SESAW tutorials [↗](#)

4.2.2 Square Ice Shelf

This is an example of velocity computation in steady state for a square ice shelf. First, launch MATLAB. In the left sidebar, select `ISSM_DIR` (the directory in which ISSM is stored) as your Current Directory. Then, navigate to `examples/SquareIceShelf`, which you can also do via the left sidebar or by running the following in the MATLAB Command Window:

```
>> cd examples/SquareIceShelf
```

You can create an empty model structure by running:

```
>> 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` (which you can see exists in the current directory by inspecting the left sidebar):

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

Define all elements as SSA:

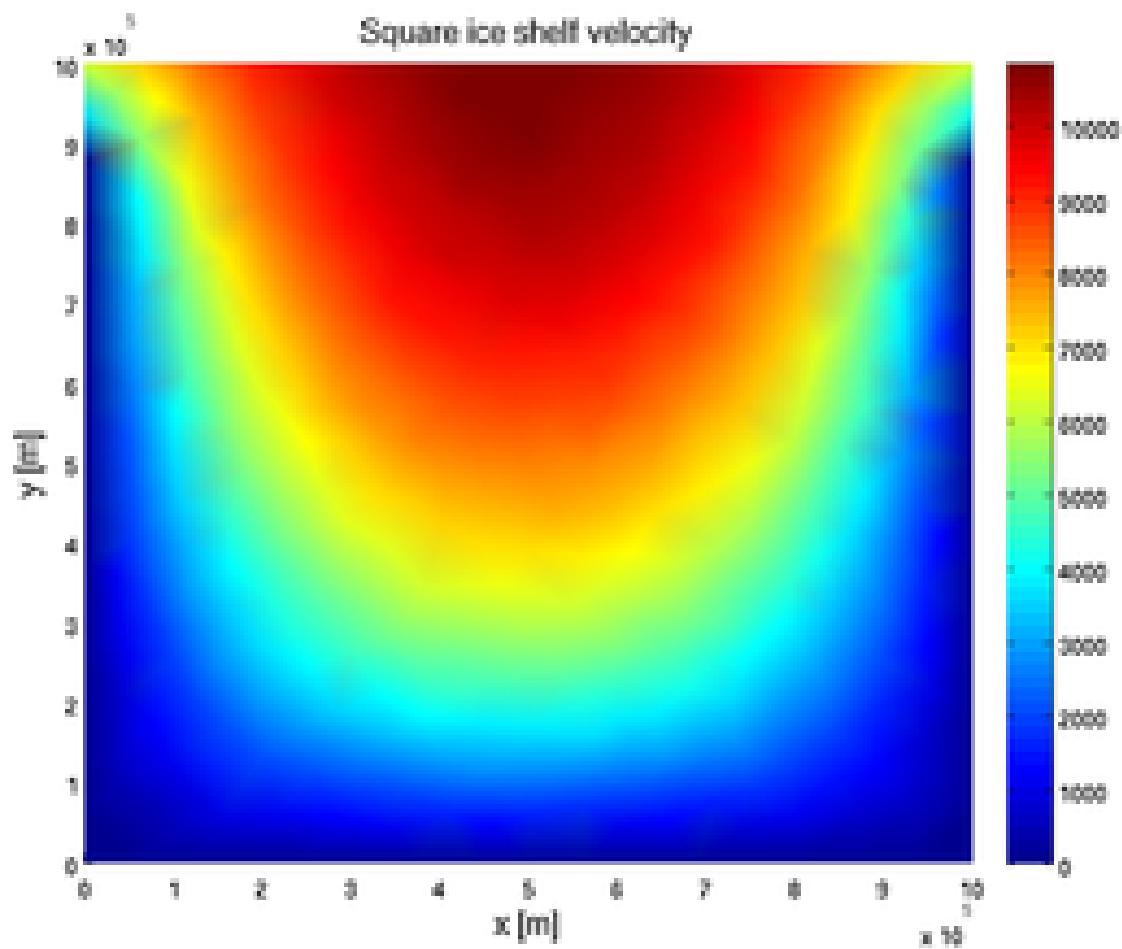
```
>> md = setflowequation(md, 'SSA', 'all');
```

Compute the velocity field of the ice shelf:

```
>> md = solve(md, 'Stressbalance');
```

Finally, generate a plot of the velocity:

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



4.2.3 Mesh Adaptation

4.2.3.1 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 `<ISSM_DIR>/examples/Mesh/` to do this tutorial.

4.2.3.2 Squaremesh

`squaremesh` generates structured uniform meshes for rectangular domains.

Usage

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

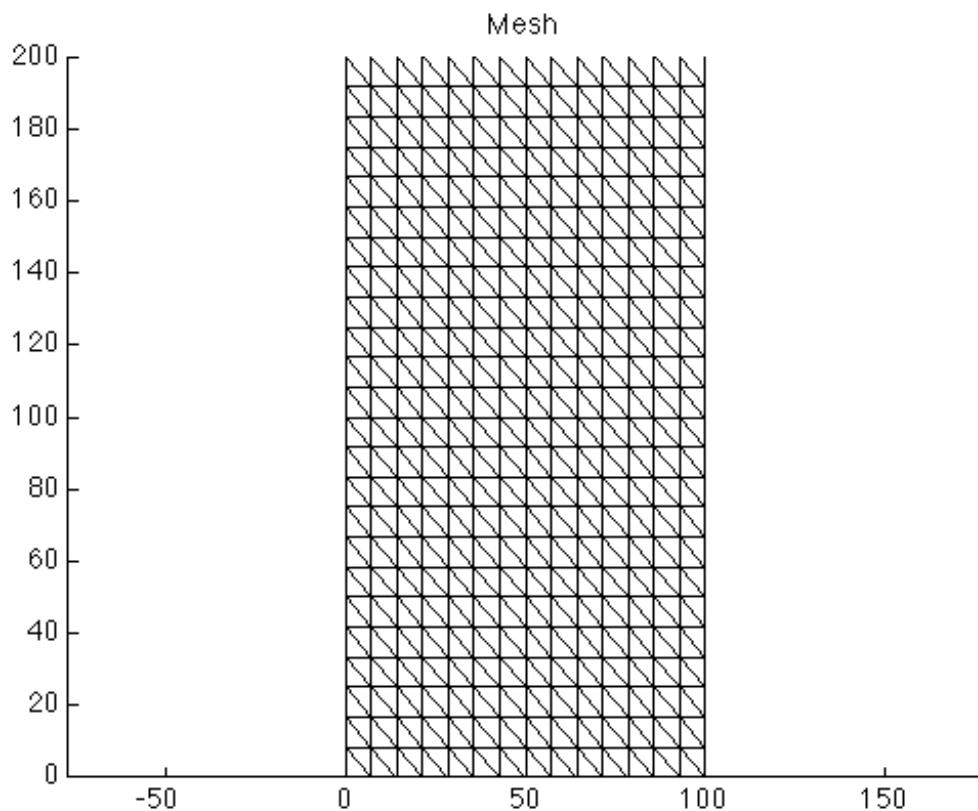
`squaremesh` takes the following arguments:

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

Example

The previous command creates the mesh shown below:

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



4.2.3.3 Roundmesh

`roundmesh` generates unstructured uniform meshes for circular domains.

Usage

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

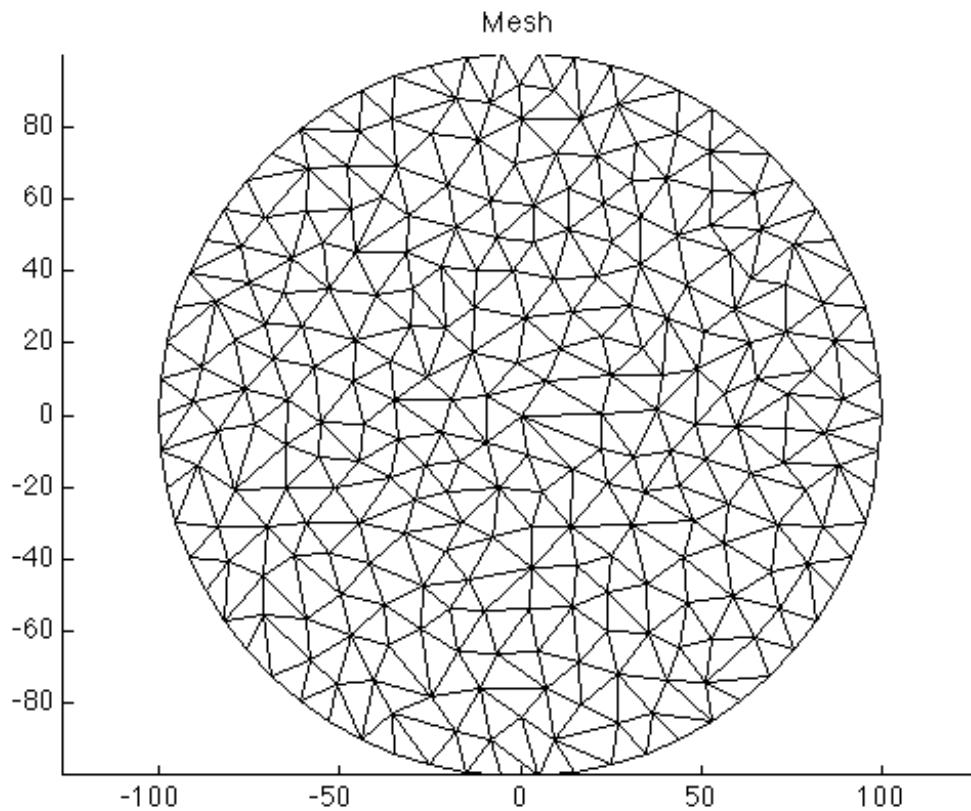
`roundmesh` takes the following arguments:

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

Example

The previous command creates the mesh shown below:

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



4.2.3.4 Triangle

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

Usage

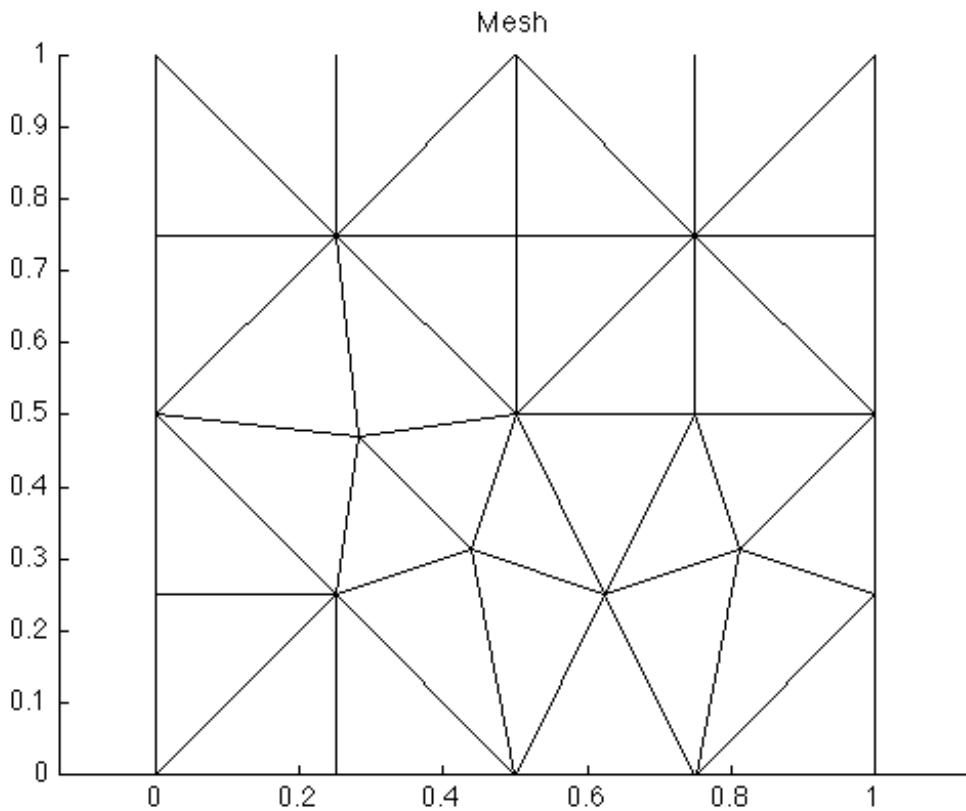
```
>> md = triangle(model, 'Square.exp', .2);
```

`triangle` takes the following arguments:

1. model
2. ARGUS file of the domain outline (`.exp` extension, see the '[Capabilities](#)' → '[Mesh Generation](#)' section 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.

4.2.3.5 Bamg

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

Usage

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

`bamg` takes as its 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)

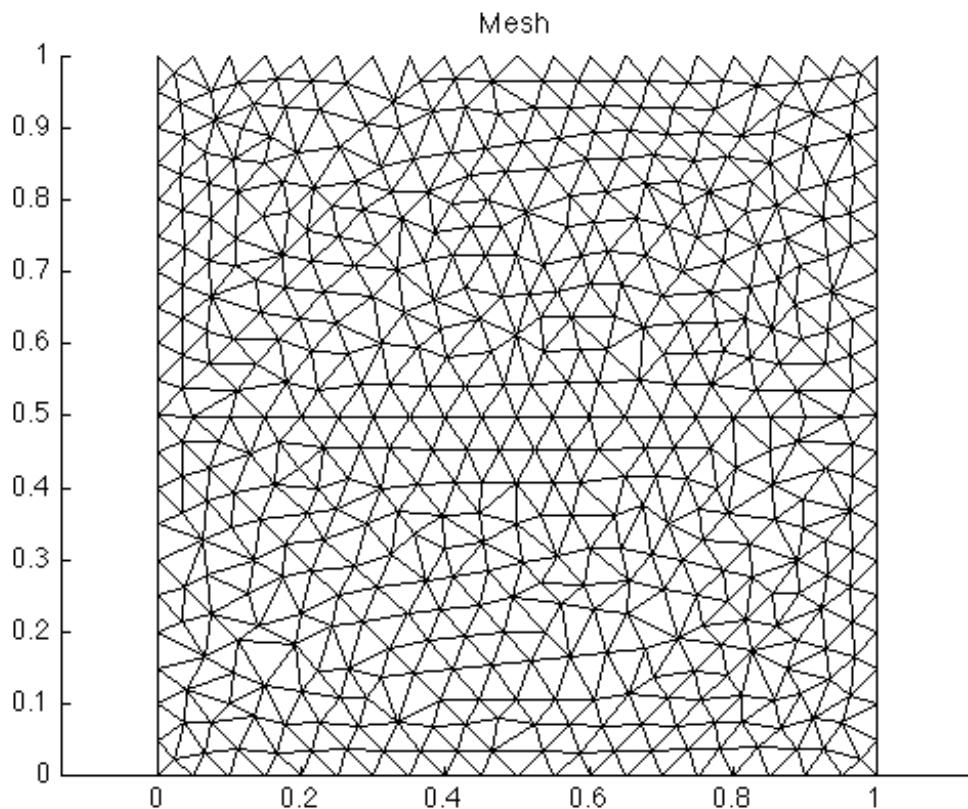
Uniform mesh

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.

Non-Uniform mesh

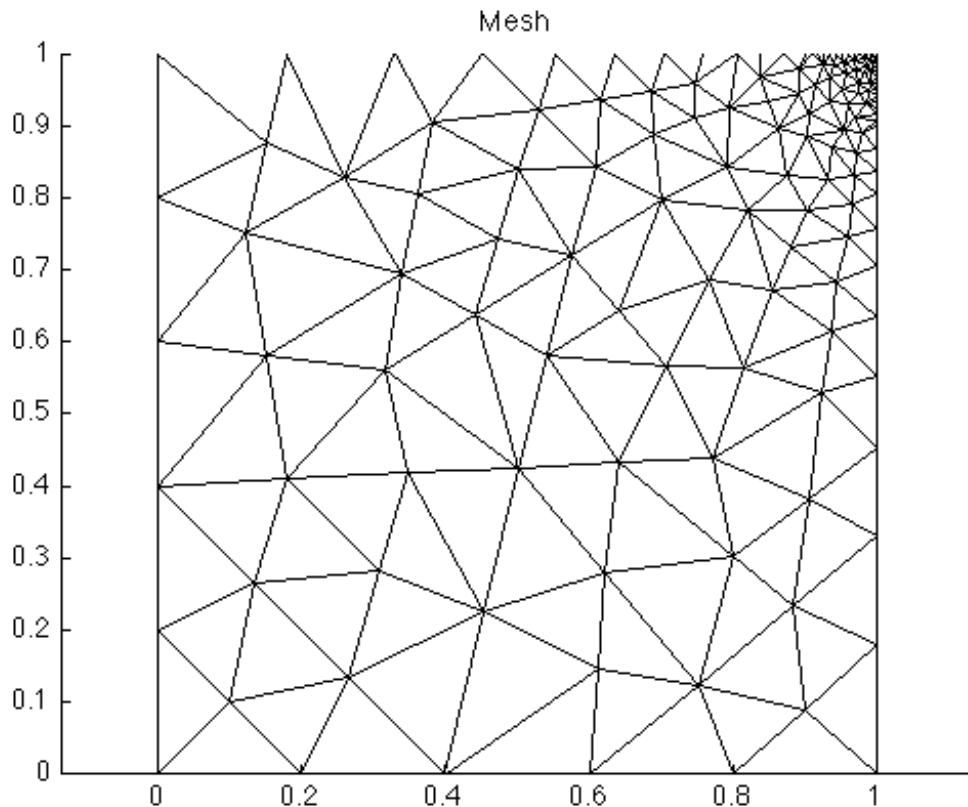
To create a non-uniform mesh, use the following options:

1. 'domain' followed by the domain name
2. 'hvvertices' 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);
```

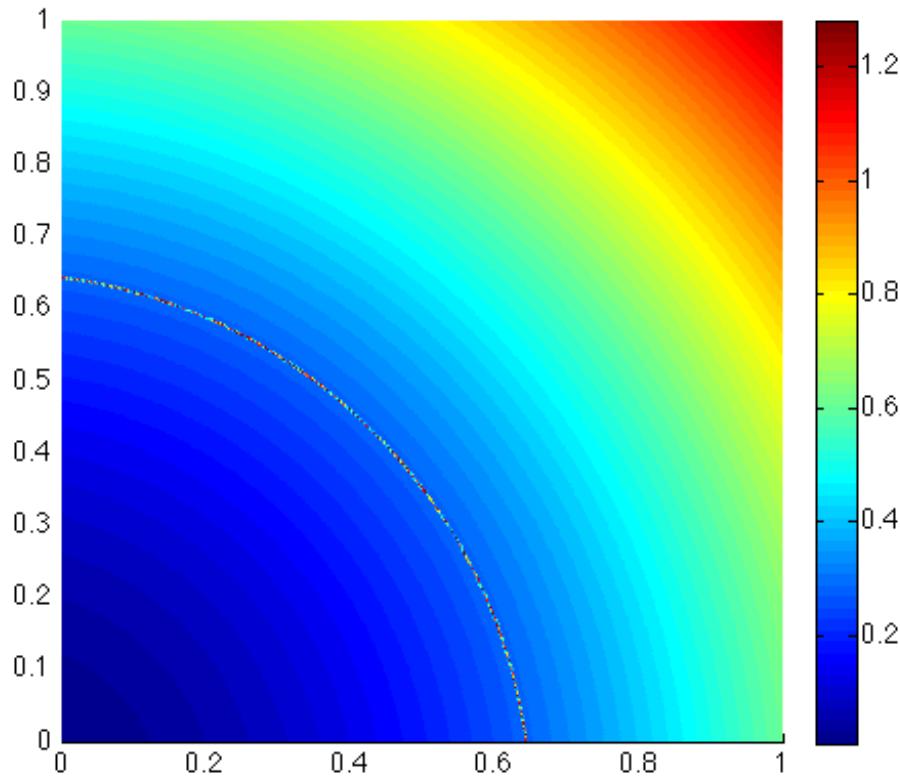
Use the `plotmodel(md, 'data', 'mesh')` command to visualize the newly defined mesh:



Mesh adaptation

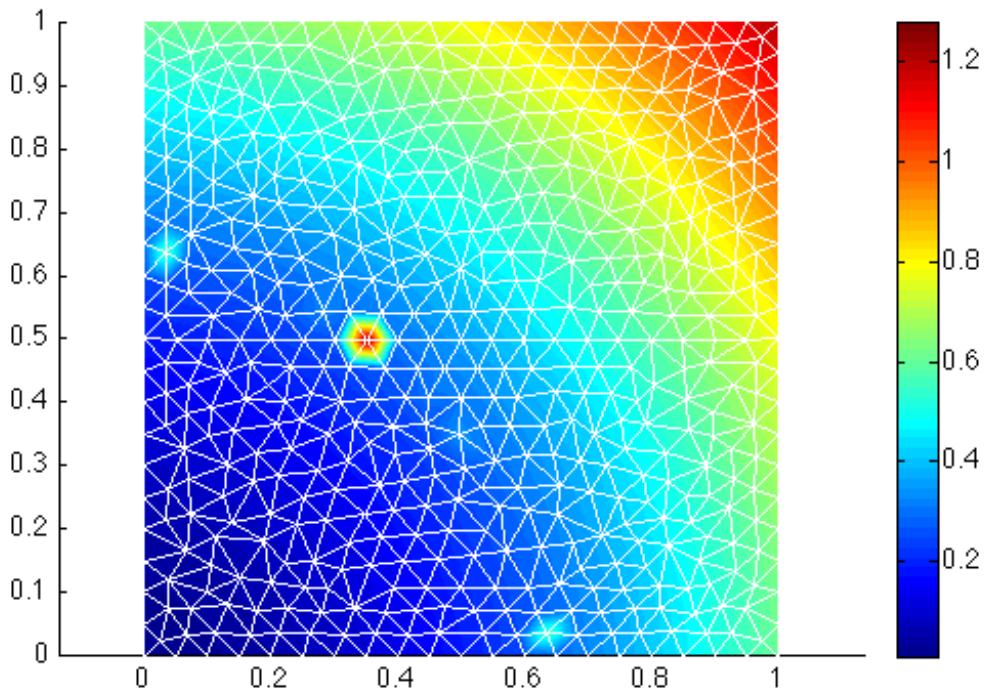
We can use observations to generate a mesh that is adapted to the solution we are trying to model. Given a solution field, `bamg` will calculate a metric based on the field's Hessian matrix (second derivative) to generate an anisotropic mesh that minimize the interpolation error (assuming that linear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



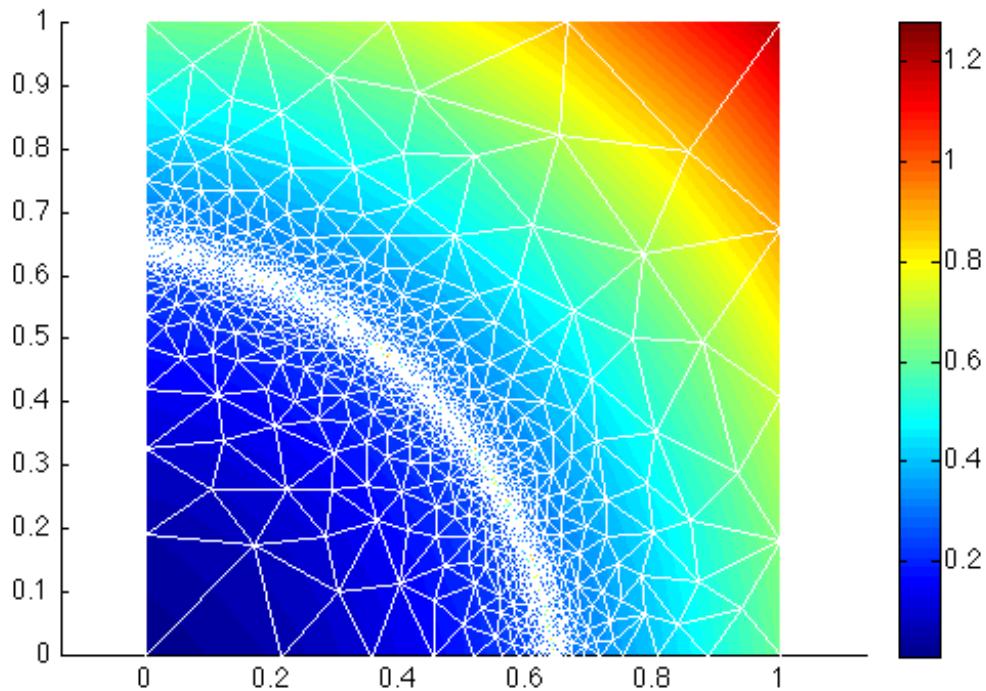
With a simple uniform mesh, the discontinuity is not captured. It is best to start with a finer mesh, which captures the discontinuity rather well, and interpolate the observations on this finer mesh to adapt the mesh anisotropically.

```
>> 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:

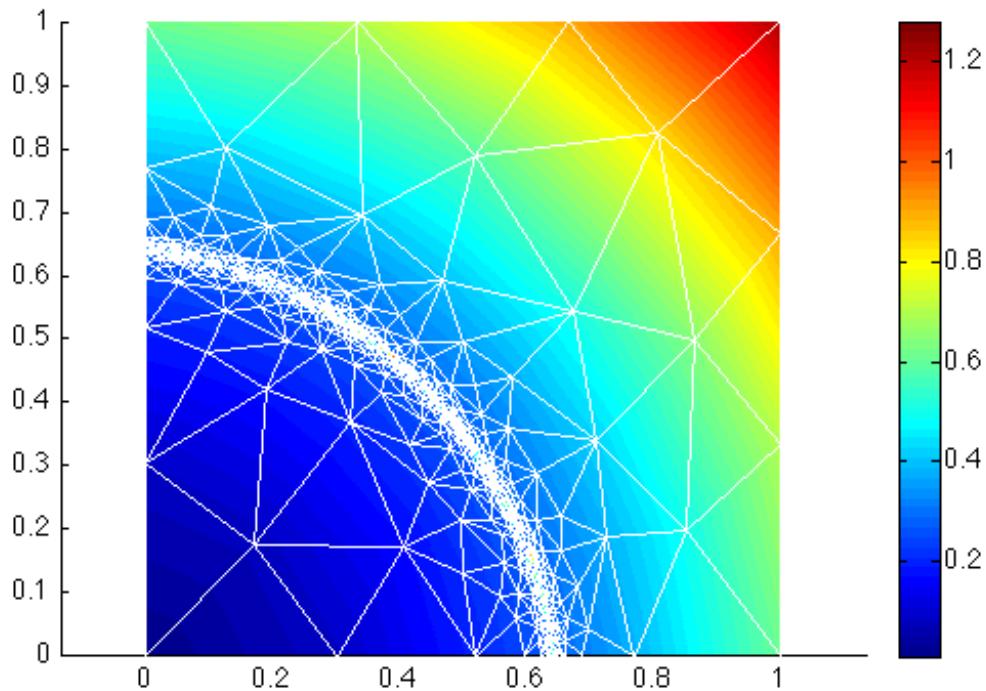
1. `'field'` followed by `vel`, the field we want to adapt the mesh to
2. `'err'` the allowed interpolation error (Here, the field must be captured within 0.05)
3. `'hmin'` minimum edge length
4. `'hmax'` maximum edge length

```
>> 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');
```



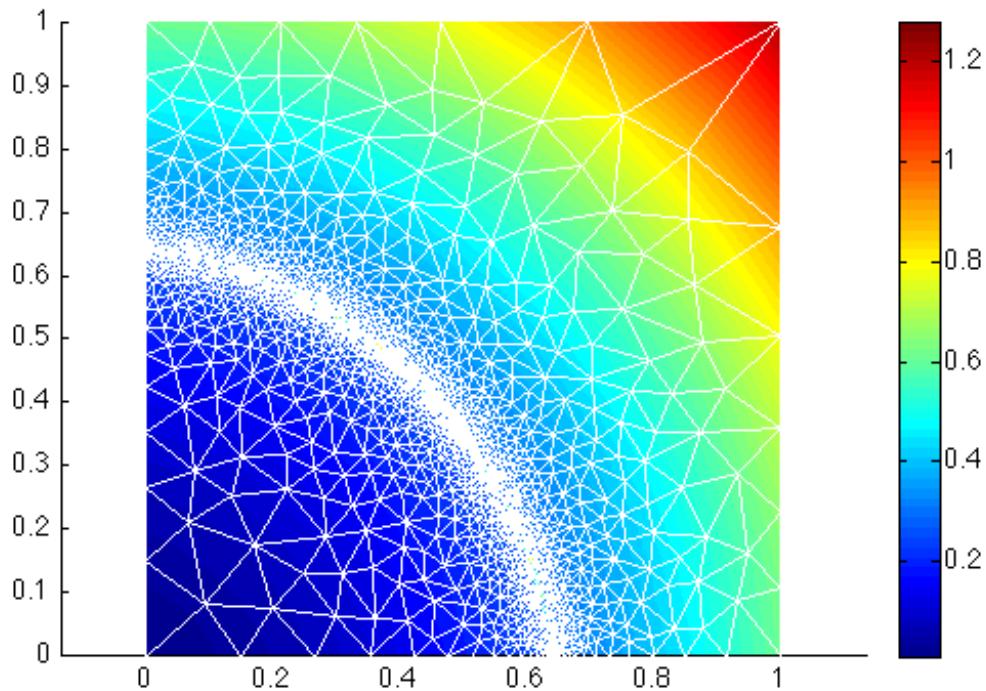
You can change the option `'err'` to 0.03, to see the effect of `'err'`. The ratio between two consecutive edges can be controlled by the option `'gradation'`.

```
>> 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 anisotropy (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');
```



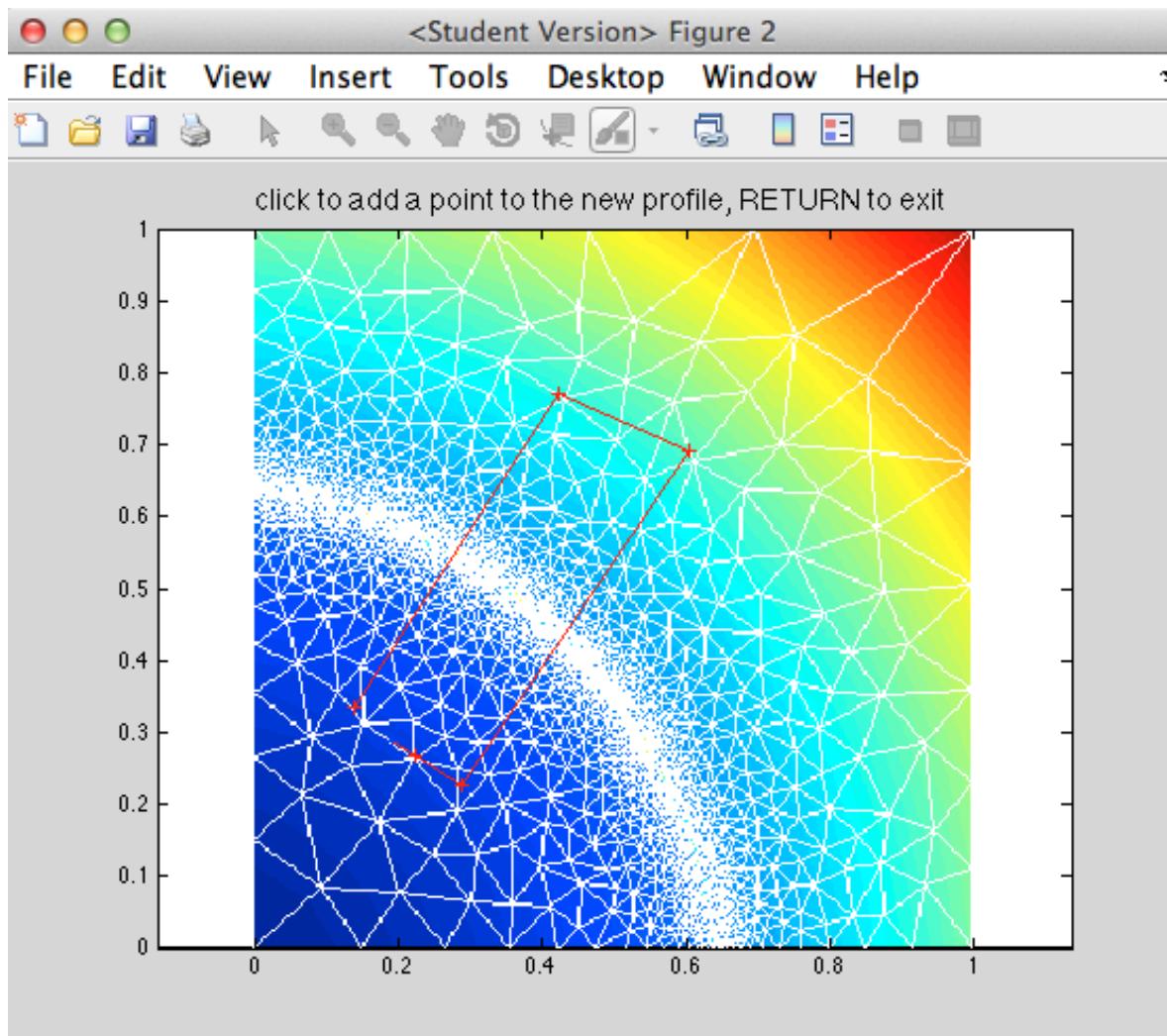
You can also try to refine a mesh using the function `circles.m`, which is provided in the same directory.

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.

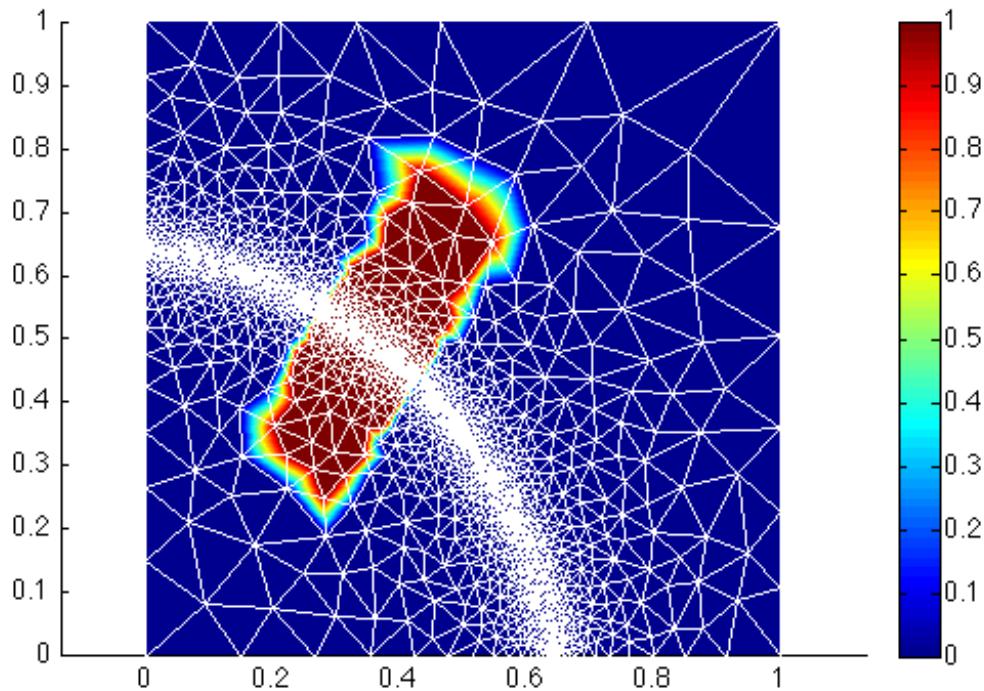
```
>> plotmodel(md, 'data', vel, 'edgecolor', 'w');
>> exptool('refinement.exp')
```



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:

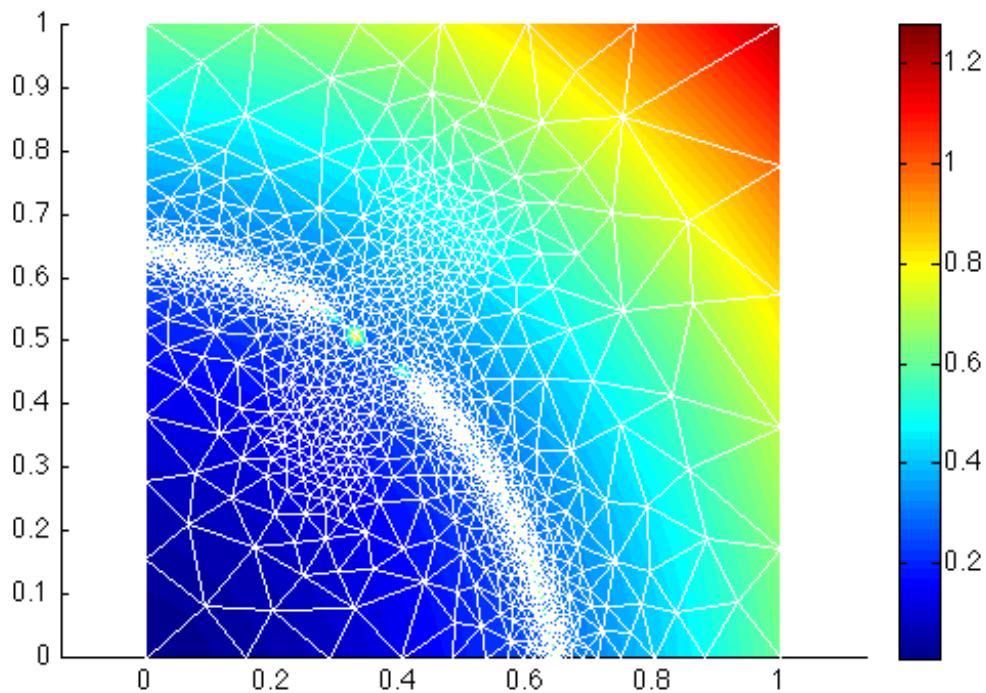
```
>> h = Nan * ones(md.mesh.numberofvertices, 1);
>> in = ContourToNodes(md.mesh.x, md.mesh.y, 'refinement.exp', 1);
>> h(find(in)) = 0.02;
>> plotmodel(md, 'data', in, 'edgecolor', 'w');
```

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');
```



Another example

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

4.2.4 Ice Flow Models

UNDER DEVELOPMENT

4.2.5 Ice Sheet Model Intercomparison Project (ISMIP) Tests

4.2.5.1 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) (see [publication ↗](#) for more information about these tests)

Go to `<ISSM_DIR>/examples/ISMIP/` to do this tutorial.

4.2.5.2 Introduction/How To

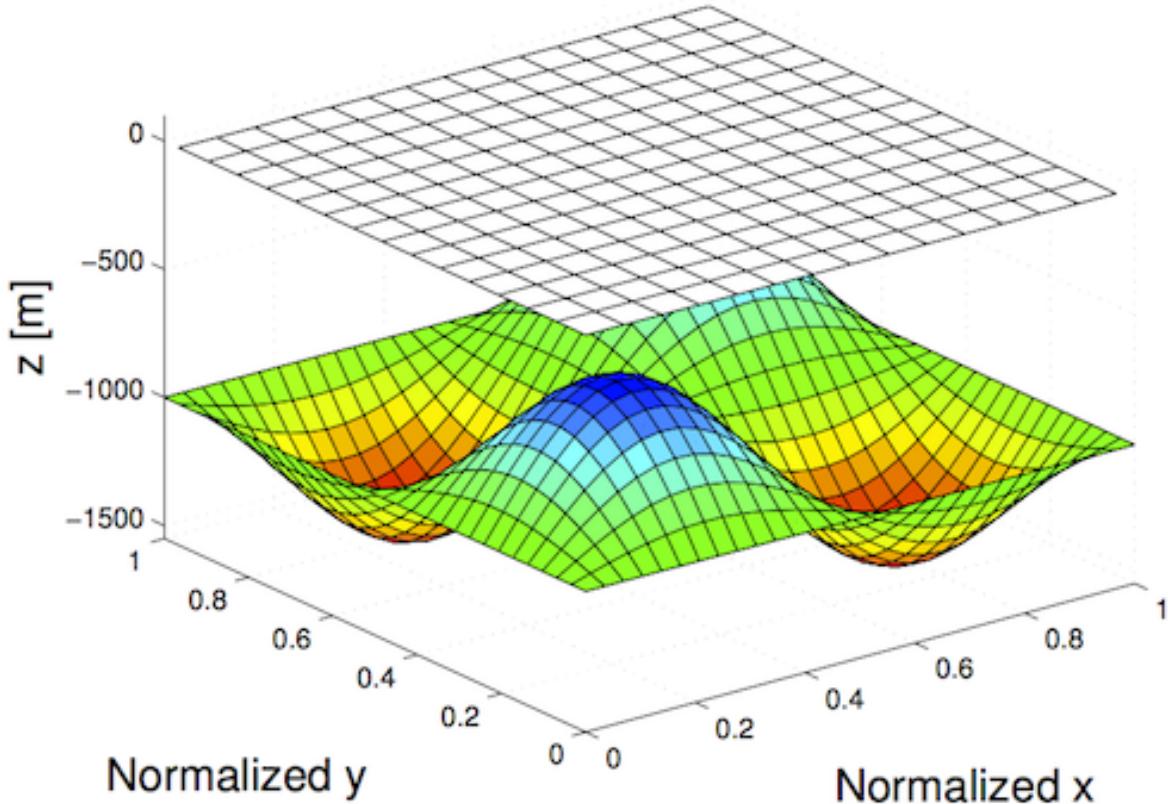
The `runme.m` file and `*par` files give a layout of the simulation that has to be modified.

- 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
- See the solutions below if you get stuck.

4.2.5.3 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



4.2.5.4 Simulation File Layout and Organization

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

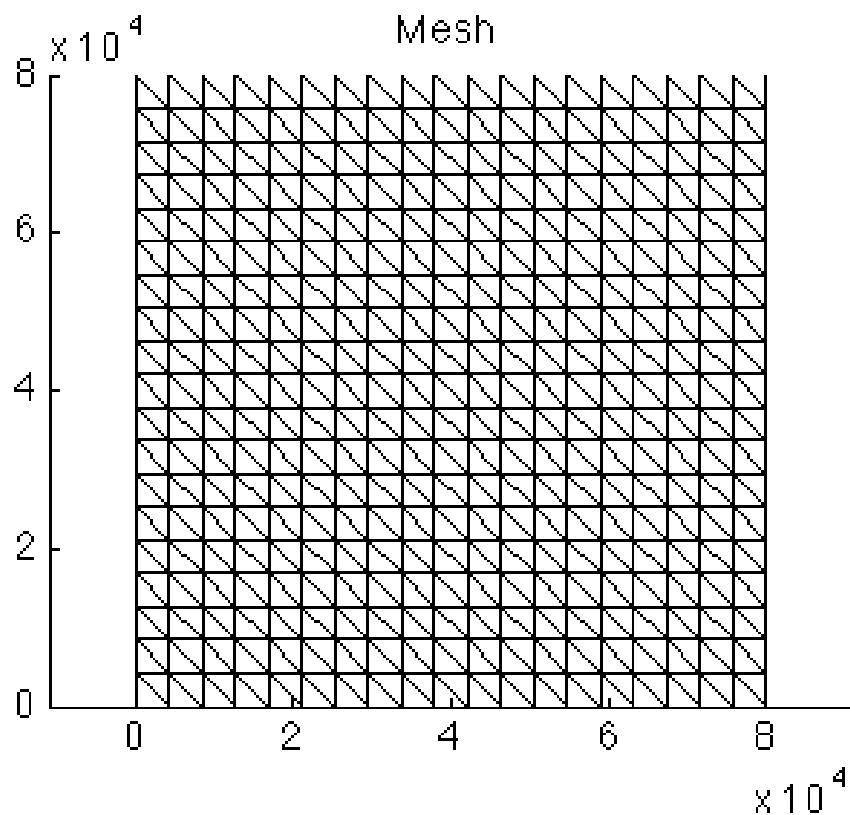
- Model loading
- Performing an action
- Model saving

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

4.2.5.5 Mesh

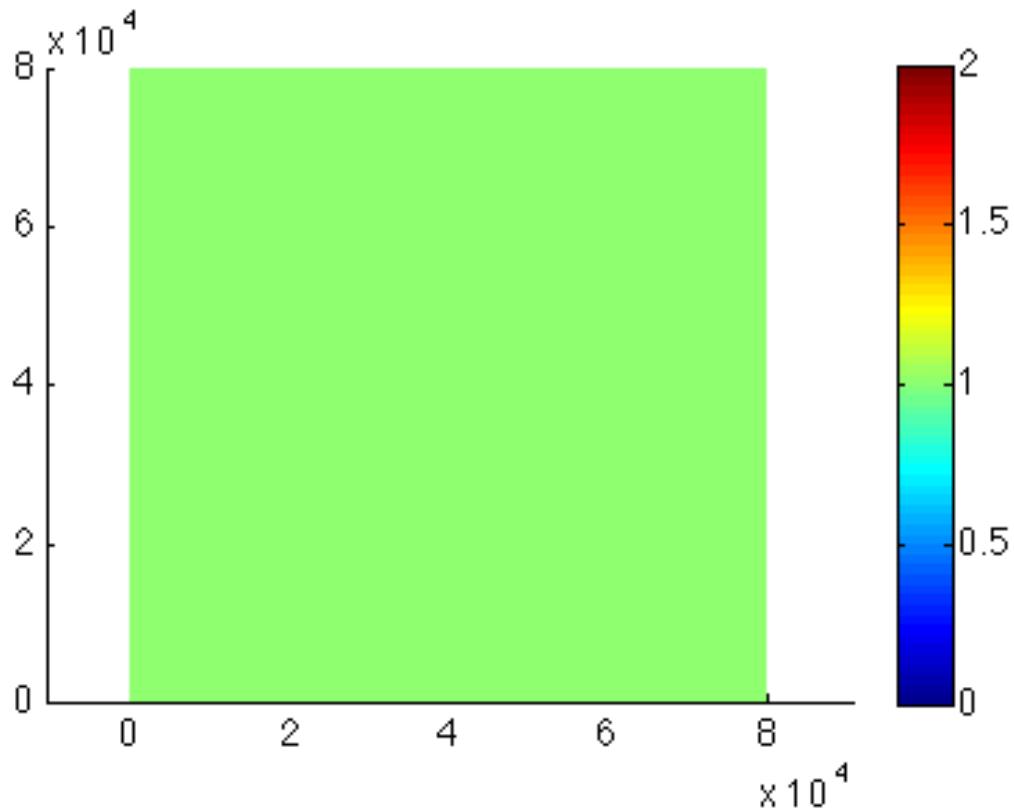
In place of loading a preceding model we initialize one. The action here is the generation of a mesh. To do this initialize `md` as a new model (`#help model`) and generate a `squaremesh` (`#help squaremesh`) with the following parameters. Afterward, plot the mesh and save the model.

- 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.

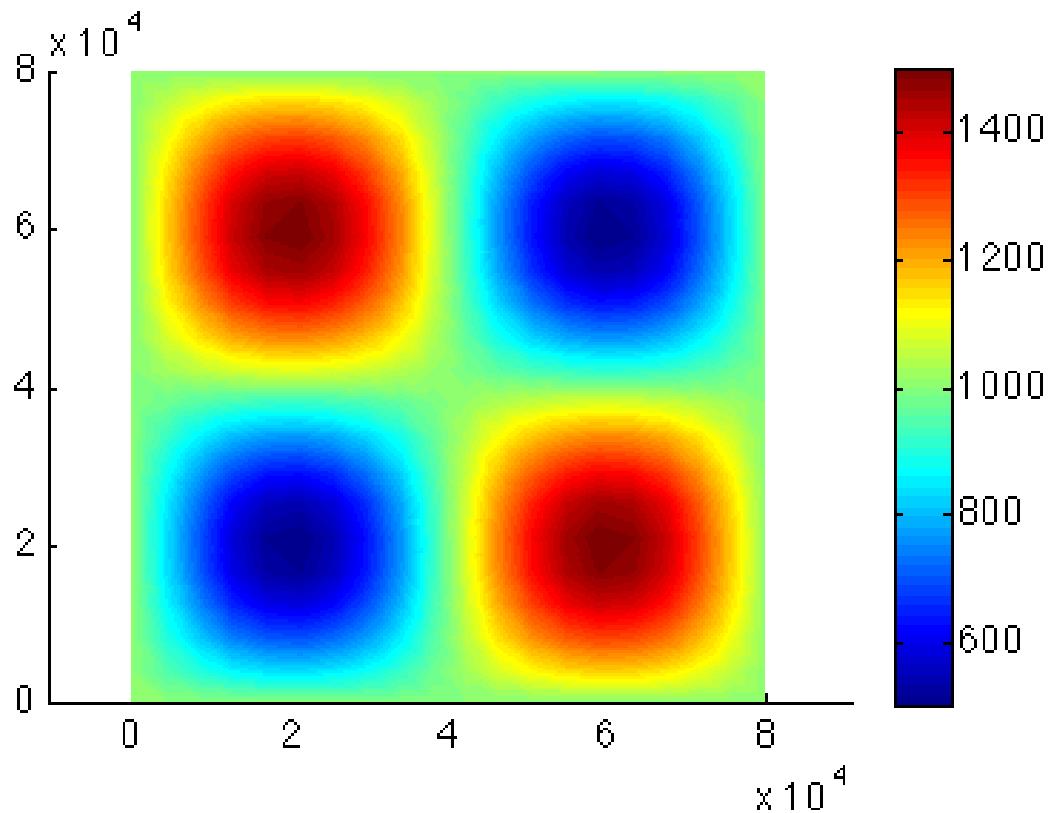
- Mesh size: 80,000 meters
- Nodes in each direction: 20
- All grounded: default



4.2.5.6 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.

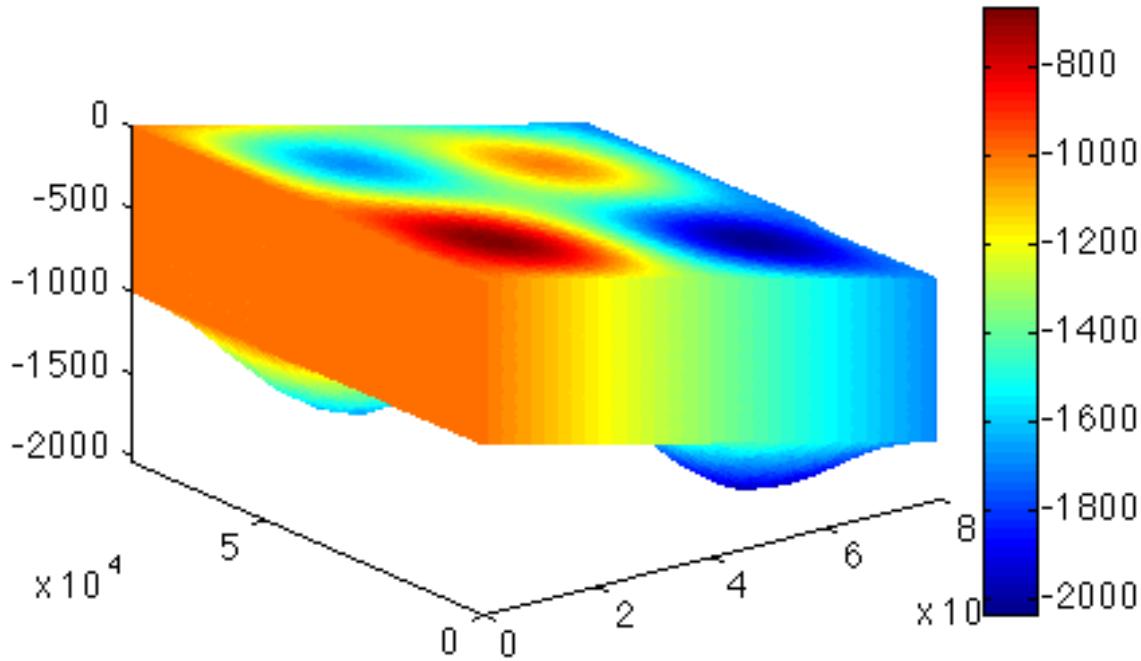
- Mesh size: 80,000 meters
- Nodes in each direction: 20
- All grounded: default
- Ice-flow parameter: `B = 6.8067 x 10^7 Pa s^1/n`
- Glen's exponent: `n = 3`



4.2.5.7 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 parameter: $B = 6.8067 \times 10^7 \text{ Pa s}^{1/n}$
- Glen's exponent: $n = 3$
- 5 layer extrusion



4.2.5.8 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

4.2.5.9 Boundary Conditions

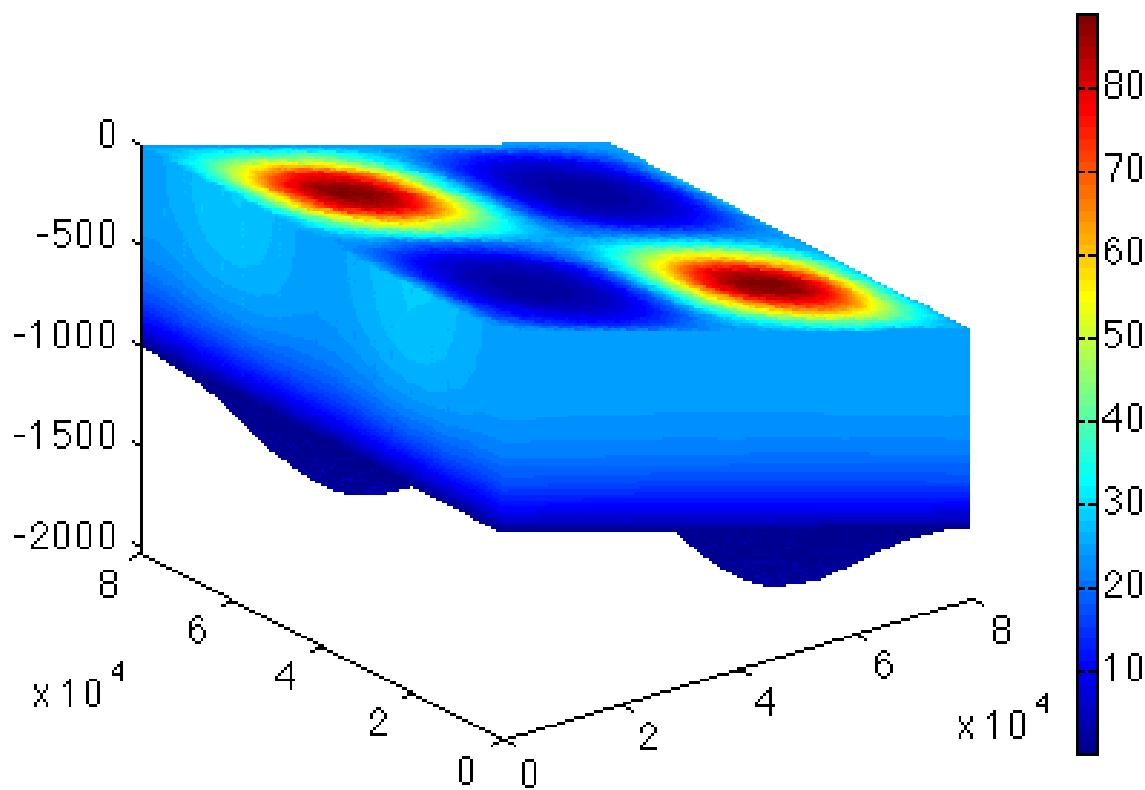
Load the `SetFlow` model. Dirichlet boundary condition are known as SPC's, where ice is frozen to the base with no velocity. SPC's are initialized at NaN one value per vertex. Extract the node-numbers at the base (`#md.mesh.vertexonbase`) and set the sliding to zero on the bed (`Vx` and `Vy`). Periodic 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: $B = 6.8067 \times 10^6 \text{ Pa s}^{1/n}$
- Glen's exponent: $n = 3$
- 5 layer extrusion
- Flow model: HO

4.2.5.10 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.

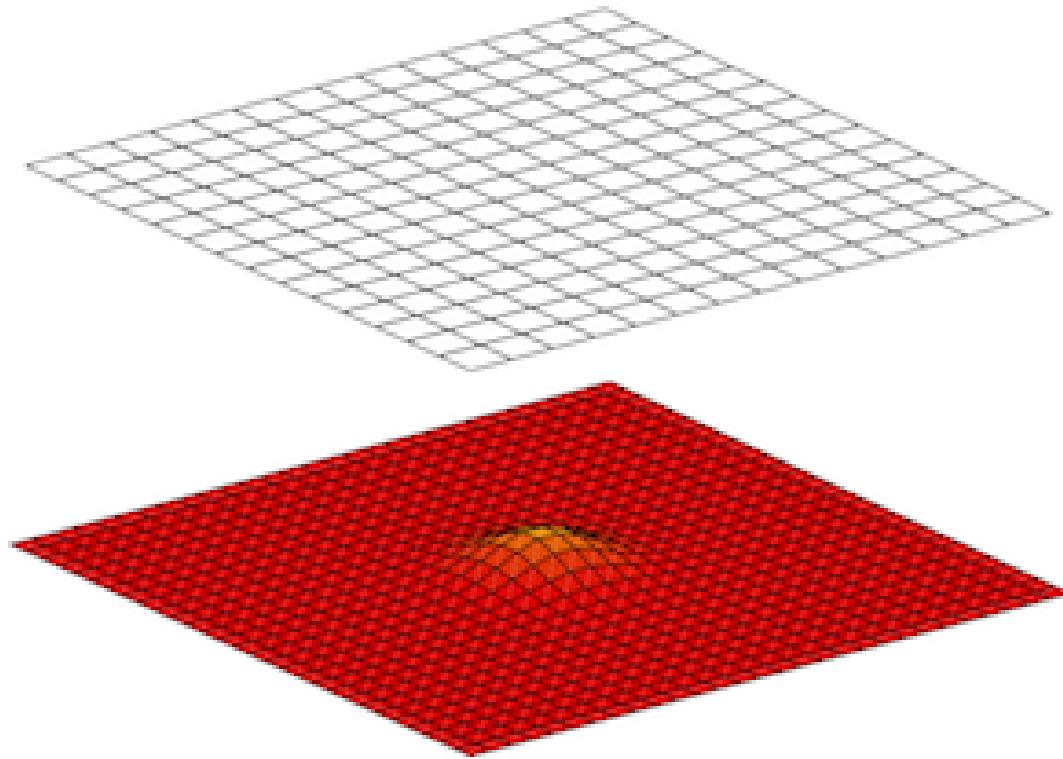
- Mesh size: 80,000 meters
- Nodes in each direction: 20
- All grounded: default
- Ice-flow parameter: $B = 6.8067 \times 10^7 \text{ Pa s}^{1/n}$
- Glen's exponent: $n = 3$
- 5 layers extrusion
- Flow model: HO



4.2.5.11 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



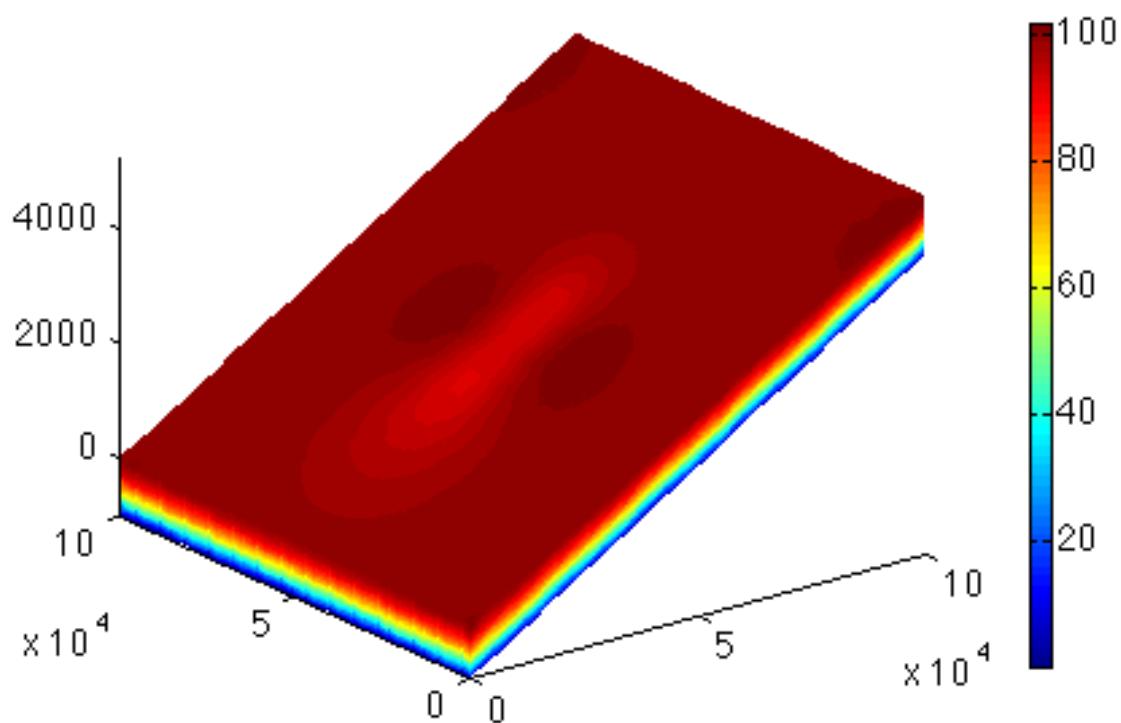
Model Setup

- Mesh size: 100,000 meters
- Nodes in each direction: 30
- All grounded: default
- Ice-flow parameter: $B = 1.4734 \times 10^{14} \text{ Pa s}^{1/n}$ (or, $B = A^{-1/n}$ where $n = 1$ and $A = 2.140373 \times 10^{-7} \text{ Pa}^{-1} \text{ yr}^{-1}$)
- Glen's exponent: $n = 1$
- 5 layers extrusion
- Flow model: HO

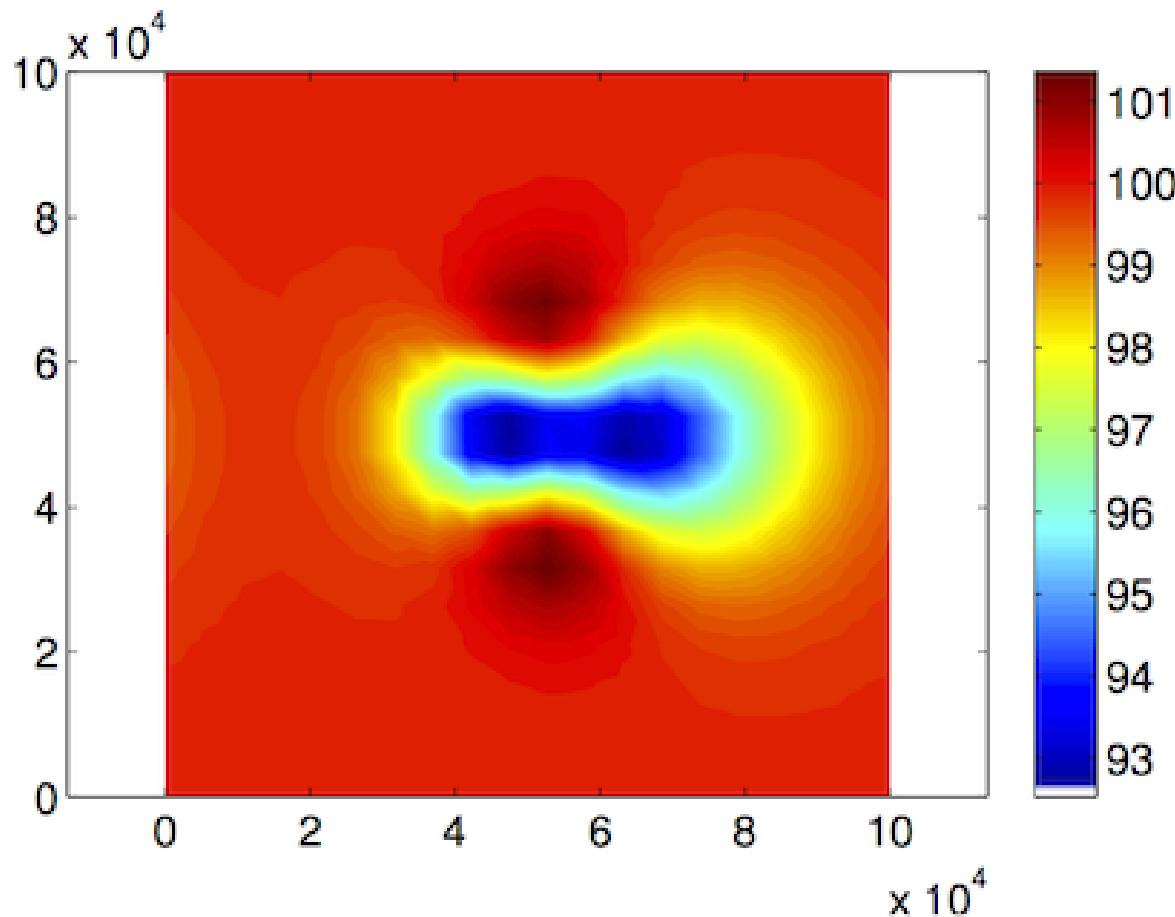
4.2.5.12 Actual Work and Results

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. Here is the upper surface velocity:

Side view:



Top view:



4.2.5.13 Solution for runme.m (MATLAB)

```
%which steps to perform; steps are from 1 to 8
%step 7 is specific to ISMIPA
%step 8 is specific to ISMIPF

steps = [1:7]; %ISMIPA
%steps = [1:6, 8]; %ISMIPB

% parameter file to be used, choose between IsmipA.par or IsmipF.par
ParamFile = 'IsmipA.par'
%ParamFile = 'IsmipF.par'

%Run Steps

%Mesh Generation #1
if any(steps == 1)
    %initialize md as a new model #help model
    %-
    md = model();
    % generate a squaremesh #help squaremesh
    % Side is 80 km long with 20 points
    %-
    if(ParamFile == 'IsmipA.par'),
```

```

        md = squaremesh(md, 80000, 80000, 20, 20);
elseif(ParamFile == 'IsmipF.par'),
    md = squaremesh(md, 100000, 100000, 30, 30);
end
% plot the given mesh #plotdoc
%-
plotmodel(md, 'data', 'mesh')
% save the given model
%-
save ./Models/ISMIP.Mesh_generation md;
end

%Masks #2
if any(steps == 2)
    % load the preceding step #help loadmodel
    % path is given by the organizer with the name of the given step
%-
md = loadmodel('./Models/ISMIP.Mesh_generation');
% set the mask #help setmask
% all MISMIP nodes are grounded
%-
md = setmask(md, '', '');
% plot the given mask #md.mask to locate the field
%-
plotmodel(md, 'data', md.mask.ocean_levelset);
% save the given model
%-
save ./Models/ISMIP.SetMask md;
end

%Parameterization #3
if any(steps == 3)
    % load the preceding step #help loadmodel
    % path is given by the organizer with the name of the given step
%-
md = loadmodel('./Models/ISMIP.SetMask');
% parametrize the model # help parameterize
% you will need to fill up the parameter file defined by the
% ParamFile variable
%-
md = parameterize(md, ParamFile);
% save the given model
%-
save ./Models/ISMIP.Parameterization md;
end

%Extrusion #4
if any(steps == 4)

    % load the preceding step #help loadmodel
    % path is given by the organizer with the name of the given step
%-
md = loadmodel('./Models/ISMIP.Parameterization');
% vertically extrude the preceding mesh #help extrude
% only 5 layers exponent 1
%-

```

```

md = extrude(md, 5, 1);
% plot the 3D geometry #plotdoc
%-
plotmodel(md, 'data', md.geometry.base)
% save the given model
%-
save ./Models/ISMIP.Extrusion md;
end

%Set the flow computing method #5
if any(steps == 5)

    % load the preceding step #help loadmodel
    % path is given by the organizer with the name of the given step
    %-
    md = loadmodel('./Models/ISMIP.Extrusion');
    % set the approximation for the flow computation #help
        setflowequation
    % We will be using the Higher Order Model (HO)
    %-
    md = setflowequation(md, 'HO', 'all');
    % save the given model
    %-
    save ./Models/ISMIP.SetFlow md;
end

%Set Boundary Conditions #6
if any(steps == 6)

    % load the preceding step #help loadmodel
    % path is given by the organizer with the name of the given step
    %-
    md = loadmodel('./Models/ISMIP.SetFlow');
    % dirichlet boundary condition are known as SPCs
    % ice frozen to the base, no velocity #md.stressbalance
    % SPCs are initialized at NaN one value per vertex
    %-
    md.stressbalance.spcvx = NaN * ones(md.mesh.numberofvertices, 1);
    %-
    md.stressbalance.spcvy = NaN * ones(md.mesh.numberofvertices, 1);
    %-
    md.stressbalance.spcvz = NaN * ones(md.mesh.numberofvertices, 1);
    % extract the nodenumbers at the base #md.mesh.vertexonbase
    %-
    basalnodes = find(md.mesh.vertexonbase);
    % set the sliding to zero on the bed
    %-
    md.stressbalance.spcvx(basalnodes) = 0.0;
    %-
    md.stressbalance.spcvy(basalnodes) = 0.0;
    % periodic boundaries have to be fixed on the sides
    % Find the indices of the sides of the domain, for x and then for y
    % for x
    % create maxX, list of indices where x is equal to max of x (use
        >> help find)
    %-

```

```

maxX = find(md.mesh.x == max(md.mesh.x));
% create minX, list of indices where x is equal to min of x
%-
minX = find(md.mesh.x == min(md.mesh.x));
% for y
% create maxY, list of indices where y is equal to max of y
% but not where x is equal to max or min of x
% (i.e, indices in maxX and minX should be excluded from maxY and
% minY)
%-
maxY = find(md.mesh.y == max(md.mesh.y) & md.mesh.x ~==
    max(md.mesh.x) & md.mesh.x ~== min(md.mesh.x));
% create minY, list of indices where y is equal to max of y
% but not where x is equal to max or min of x
%-
minY = find(md.mesh.y == min(md.mesh.y) & md.mesh.x ~==
    max(md.mesh.x) & md.mesh.x ~== min(md.mesh.x));
% set the node that should be paired together, minX with maxX and
% minY with maxY
% #md.stressbalance.vertex_pairing
%-
md.stressbalance.vertex_pairing = [minX, maxX; minY, maxY];
if (ParamFile == 'IsmipF.par')
    % if we are dealing with IsmipF the solution is in
    % masstransport
    md.masstransport.vertex_pairing =
        md.stressbalance.vertex_pairing;
end
% save the given model
%-
save ./Models/ISMIP.BoundaryCondition md;
end

%Solving #7
if any(steps == 7)
    % load the preceding step #help loadmodel
    % path is given by the organizer with the name of the given step
    %-
    md = loadmodel('./Models/ISMIP.BoundaryCondition');
    % Set cluster #md.cluster
    % generic parameters #help generic
    % set only the name and number of process
    %-
    md.cluster = generic('name', oshostname(), 'np', 2);
    % Set which control message you want to see #help verbose
    %-
    md.verbose = verbose('convergence', true);
    % Solve #help solve
    % we are solving a StressBalanc
    %-
    md = solve(md, 'Stressbalance');
    % save the given model
    %-
    save ./Models/ISMIP.StressBalance md;
    % plot the surface velocities #plotdoc
    %-

```

```

    plotmodel(md, 'data', md.results.StressbalanceSolution.Vel)
end

%Solving #8
if any(steps == 8)
    % load the preceding step #help loadmodel
    % path is given by the organizer with the name of the given step
    %->
    md = loadmodel('./Models/ISMIP.BoundaryCondition');
    % Set cluster #md.cluster
    % generic parameters #help generic
    % set only the name and number of process
    %->
    md.cluster = generic('name', oshostname(), 'np', 2);
    % Set which control message you want to see #help verbose
    %->
    md.verbose = verbose('convergence', true);
    % set the transient model to ignore the thermal model
    % #md.transient
    %->
    md.transient.isthermal = 0;
    % define the timestepping scheme
    % everything here should be provided in years #md.timestepping
    % give the length of the time_step (4 years)
    %->
    md.timestepping.time_step = 4;
    % give final_time (20 * 4 years time_steps)
    %->
    md.timestepping.final_time = 4 * 20;
    % Solve #help solve
    % we are solving a TransientSolution
    %->
    md = solve(md, 'Transient');
    % save the given model
    %->
    save ./Models/ISMIP.Transient md;
    % plot the surface velocities #plotdoc
    %->
    plotmodel(md, 'data', md.results.TransientSolution(20).Vel)
end

```

4.2.5.14 Solution for runme.m (Python)

```

import numpy as np
from model import *
from squaremesh import squaremesh
from plotmodel import plotmodel
from export_netCDF import export_netCDF
from loadmodel import loadmodel
from setmask import setmask
from parameterize import parameterize
from setflowequation import setflowequation
from socket import gethostname

```

```

from solve import solve

#which steps to perform; steps are from 1 to 8
#step 7 is specific to ISMIPA
#step 8 is specific to ISMIPF

steps = [1, 2, 3, 4, 5, 6, 8]

# parameter file to be used, choose between IsmipA_cor.py or
# IsmipF_cor.py
ParamFile = 'IsmipF_cor.py'

#Run Steps

#Mesh Generation #1
if 1 in steps:
    print("Now generating the mesh")
    #initialize md as a new model help(model)
    #-->
    md = model()
    # generate a squaremesh help(squaremesh)
    # Side is 80 km long with 20 points
    #-->
    if ParamFile == 'IsmipA_cor.py':
        md = squaremesh(md, 80000, 80000, 20, 20)
    elif ParamFile == 'IsmipF_cor.py':
        md = squaremesh(md, 100000, 100000, 30, 30)

    # plot the given mesh plotdoc()
    #-->
    plotmodel(md, 'data', 'mesh', 'figure', 1)
    # save the given model
    #-->
    export_netcdf(md, "./Models/ISMIP-Mesh_generation.nc")

#Masks #2
if 2 in steps:
    print("Setting the masks")
    # load the preceding step help(loadmodel)
    # path is given by the organizer with the name of the given step
    #-->
    md = loadmodel("./Models/ISMIP-Mesh_generation.nc")
    # set the mask help(setmask)
    # all MISMIP nodes are grounded
    #-->
    md = setmask(md, ' ', ' ')
    # plot the given mask #md.mask to locate the field
    #-->
    plotmodel(md, 'data', md.mask.ocean_levelset, 'figure', 2)
    # save the given model
    #-->
    export_netcdf(md, "./Models/ISMIP-SetMask.nc")

#Parameterization #3
if 3 in steps:
    print("Parameterizing")

```

```

# load the preceding step #help loadmodel
# path is given by the organizer with the name of the given step
#->
md = loadmodel("./Models/ISMIP-SetMask.nc")
# parametrize the model # help parameterize
# you will need to fill up the parameter file (given by the
# ParamFile variable)
#->
md = parameterize(md, ParamFile)
# save the given model
#->
export_netcdf(md, "./Models/ISMIP-Parameterization.nc")

#Extrusion #4
if 4 in steps:
    print("Extruding")
    # load the preceding step #help loadmodel
    # path is given by the organizer with the name of the given step
    #->
    md = loadmodel("./Models/ISMIP-Parameterization.nc")
    # vertically extrude the preceding mesh #help extrude
    # only 5 layers exponent 1
    #->
    md = md.extrude(5, 1)
    # plot the 3D geometry #plotdoc
    #->
    plotmodel(md, 'data', md.geometry.base, 'figure', 3)
    # save the given model
    #->
    export_netcdf(md, "./Models/ISMIP-Extrusion.nc")

#Set the flow computing method #5
if 5 in steps:
    print("setting flow approximation")
    # load the preceding step #help loadmodel
    # path is given by the organizer with the name of the given step
    #->
    md = loadmodel("./Models/ISMIP-Extrusion.nc")
    # set the approximation for the flow computation #help
        setflowequation
    # We will be using the Higher Order Model (HO)
    #->
    md = setflowequation(md, 'HO', 'all')
    # save the given model
    #->
    export_netcdf(md, "./Models/ISMIP-SetFlow.nc")

#Set Boundary Conditions #6
if 6 in steps:
    print("setting boundary conditions")
    # load the preceding step #help loadmodel
    # path is given by the organizer with the name of the given step
    #->
    md = loadmodel("./Models/ISMIP-SetFlow.nc")
    # dirichlet boundary condition are known as SPCs
    # ice frozen to the base, no velocity #md.stressbalance

```

```

# SPCs are initialized at NaN one value per vertex
#->
md.stressbalance.spcvx = np.nan *
    np.ones((md.mesh.numberofvertices))
#->
md.stressbalance.spcvy = np.nan *
    np.ones((md.mesh.numberofvertices))
#->
md.stressbalance.spcvz = np.nan *
    np.ones((md.mesh.numberofvertices))
# extract the nodenumbers at the base #md.mesh.vertexonbase
#->
basalnodes = np.nonzero(md.mesh.vertexonbase)
# set the sliding to zero on the bed (Vx and Vy)
#->
md.stressbalance.spcvx[basalnodes] = 0.0
#->
md.stressbalance.spcvy[basalnodes] = 0.0
# periodic boundaries have to be fixed on the sides
# Find the indices of the sides of the domain, for x and then
    for y
# for x
# create maxX, list of indices where x is equal to max of x (use
    >> help find)
#->
maxX = np.squeeze(np.nonzero(md.mesh.x == np.nanmax(md.mesh.x)))
# create minX, list of indices where x is equal to min of x
#->
minX = np.squeeze(np.nonzero(md.mesh.x == np.nanmin(md.mesh.x)))
# for y
# create maxY, list of indices where y is equal to max of y
# but not where x is equal to max or min of x
# (i.e, indices in maxX and minX should be excluded from maxY
    and minY)
#->
maxY = np.squeeze(np.logical_and.reduce((md.mesh.y ==
    np.nanmax(md.mesh.y), md.mesh.x != np.nanmin(md.mesh.x),
    md.mesh.x != np.nanmax(md.mesh.x))))
# create minY, list of indices where y is equal to max of y
# but not where x is equal to max or min of x
#->
minY = np.squeeze(np.logical_and.reduce((md.mesh.y ==
    np.nanmin(md.mesh.y), md.mesh.x != np.nanmin(md.mesh.x),
    md.mesh.x != np.nanmax(md.mesh.x))))
# set the node that should be paired together, minX with maxX
    and minY with maxY
# #md.stressbalance.vertex_pairing
#->
md.stressbalance.vertex_pairing = np.hstack((np.vstack((minX +
    1, maxX + 1)), np.vstack((minY + 1, maxY + 1)))).T
if ParamFile == 'IsmipF_cor.py':
    # if we are dealing with IsmipF the solution is in
        masstransport
    md.masstransport.vertex_pairing =
        md.stressbalance.vertex_pairing

```

```

# save the given model
#->
export_netcdf(md, "./Models/ISMIP-BoundaryCondition.nc")

#Solving #7
if 7 in steps:
    print("running the solver for the A case")
    # load the preceding step #help loadmodel
    # path is given by the organizer with the name of the given step
    #->
    md = loadmodel("./Models/ISMIP-BoundaryCondition.nc")
    # Set cluster #md.cluster
    # generic parameters #help generic
    # set only the name and number of process
    #->
    md.cluster = generic('name', gethostname(), 'np', 2)
    # Set which control message you want to see #help verbose
    #->
    md.verbose = verbose('convergence', True)
    # Solve #help solve
    # we are solving a StressBalance
    #->
    md = solve(md, 'Stressbalance')
    # save the given model
    #->
    export_netcdf(md, "./Models/ISMIP-StressBalance.nc")
    # plot the surface velocities #plotdoc
    #->
    plotmodel(md, 'data', md.results.StressbalanceSolution.Vel,
               'figure', 4)

#Solving #8
if 8 in steps:
    print("running the solver for the F case")
    # load the preceding step #help loadmodel
    # path is given by the organizer with the name of the given step
    #->
    md = loadmodel("./Models/ISMIP-BoundaryCondition.nc")
    # Set cluster #md.cluster
    # generic parameters #help generic
    # set only the name and number of process
    #->
    md.cluster = generic('name', gethostname(), 'np', 2)
    # Set which control message you want to see #help verbose
    #->
    md.verbose = verbose('convergence', True)
    # set the transient model to ignore the thermal model
    # #md.transient
    #->
    md.transient.isthermal = 0
    # define the timestepping scheme
    # everything here should be provided in years #md.timestepping
    # give the length of the time_step (4 years)
    #->
    md.timestepping.time_step = 1
    # give final_time (20 * 4 years time_steps)

```

```

#->
md.timestepping.final_time = 1 * 20
# Solve #help solve
# we are solving a TransientSolution
#->
md = solve(md, 'Transient')
# save the given model
#->
export_netcdf(md, "./Models/ISMIP-Transient.nc")
# plot the surface velocities #plotdoc
#->
plotmodel(md, 'data', md.results.TransientSolution[19].Vel,
           'layer', 5, 'figure', 5)

```

4.2.5.15 Solution for IsmipA.par (MATLAB)

```

%Parameterization for ISMIP A experiment

%Set the Simulation generic name #md.miscellaneous
%->

%Geometry
disp(' Constructing Geometry');

%Define the geometry of the simulation #md.geometry
%surface is [-x * tan(0.5 * pi / 180)] #md.mesh
%->
md.geometry.surface = -md.mesh.x * tan(0.5 * pi / 180.);
%base is [surface - 1000 + 500 * sin(x * 2 * pi / L) .* sin(y * 2 *
pi / L)]
%L is the size of the side of the square #max(md.mesh.x) -
min(md.mesh.x)
%->
L = max(md.mesh.x) - min(md.mesh.x);
md.geometry.base = md.geometry.surface - 1000.0 + 500.0 *
sin(md.mesh.x * 2.0 * pi / L) .* sin(md.mesh.y * 2.0 * pi / L);
%thickness is the difference between surface and base #md.geometry
%->
md.geometry.thickness = md.geometry.surface - md.geometry.base;
%plot the geometry to check it out
%->
plotmodel(md, 'data', md.geometry.thickness);

disp(' Defining friction parameters');

%These parameters will not be used but need to be fixed #md.friction
%one friction coefficient per node (md.mesh.numberofvertices, 1)
%->
md.friction.coefficient = 200.0 * ones(md.mesh.numberofvertices, 1);
%one friction exponent (p, q) per element
%->
md.friction.p = ones(md.mesh.numberofelements, 1);
%->

```

```

md.friction.q = ones(md.mesh.numberofelements, 1);

disp(' Construct ice rheological properties');

%The rheology parameters sit in the material section #md.materials
%B has one value per vertex
%-
md.materials.rheology_B = 6.8067e7 * ones(md.mesh.numberofvertices,
    1);
%n has one value per element
%-
md.materials.rheology_n = 3 * ones(md.mesh.numberofelements, 1);

disp(' Set boundary conditions');

%Set the default boundary conditions for an ice-sheet
% #help SetIceSheetBC
%-
md = SetIceSheetBC(md);

```

4.2.5.16 Solution for IsmipA.py (Python)

4.2.5.17 Solution for IsmipF.par (MATLAB)

```

%Parameterization for ISMIP F experiment

%Set the Simulation generic name #md.miscellaneous
%-
%Geometry
disp(' Constructing Geometry');

%Define the geometry of the simulation #md.geometry
%surface is [-x * tan(3.0 * pi / 180)] #md.mesh
%-
md.geometry.surface = -md.mesh.x * tan(3.0 * pi / 180.0);
%base is [surface - 1000 + 100 * exp(-((x - L / 2) .^ 2 + (y - L /
2) .^ 2) / (10000.^2))]
%L is the size of the side of the square #max(md.mesh.x) -
min(md.mesh.x)
%-
L = max(md.mesh.x) - min(md.mesh.x);
%-
md.geometry.base = md.geometry.surface - 1000.0 + 100.0 *
exp(-((md.mesh.x - L / 2.0) .^ 2.0 + (md.mesh.y - L / 2.0) .^
2.0) / (10000.^2.0));
%thickness is the difference between surface and base #md.geometry
%-
md.geometry.thickness = md.geometry.surface - md.geometry.base;

```

```
%plot the geometry to check it out
%-
plotmodel(md, 'data', md.geometry.thickness);

disp(' Defining friction parameters');

%These parameters will not be used but need to be fixed #md.friction
%one friction coefficient per node (md.mesh.numberofvertices, 1)
%conversion from year to seconds with #md.constants.yts
%-
md.friction.coefficient = sqrt(md.constants.yts / (1000 * 2.140373 *
    10^-7)) * ones(md.mesh.numberofvertices, 1);
%one friction exponent (p, q) per element
%-
md.friction.p = ones(md.mesh.numberofelements, 1);
%-
md.friction.q = zeros(md.mesh.numberofelements, 1);

disp(' Construct ice rheological properties');

%The rheology parameters sit in the material section #md.materials
%B has one value per vertex
%-
md.materials.rheology_B = (1 / (2.140373 * 10^-7 /
    md.constants.yts)) * ones(md.mesh.numberofvertices, 1);
%n has one value per element
%-
md.materials.rheology_n = 1 * ones(md.mesh.numberofelements, 1);

disp(' Set boundary conditions');

%Set the default boundary conditions for an ice-sheet
% #help SetIceSheetBC
%-
md = SetIceSheetBC(md);

disp(' Initializing velocity and pressure');

%initialize the velocity and pressurefields of #md.initialization
%-
md.initialization.vx = zeros(md.mesh.numberofvertices, 1);
%-
md.initialization.vy = zeros(md.mesh.numberofvertices, 1);
%-
md.initialization.vz = zeros(md.mesh.numberofvertices, 1);
%-
md.initialization.pressure = zeros(md.mesh.numberofvertices, 1);
```

4.2.5.18 Solution for IsmipF.py (Python)

```
import numpy as np
from plotmodel import plotmodel
from SetIceSheetBC import SetIceSheetBC
```

```

#Parameterization for ISMIP F experiment

#Set the Simulation generic name #md.miscellaneous
#->
md.miscellaneous.name = 'IsmipF_cor'
#Geometry
print('    Constructing Geometry')

#Define the geometry of the simulation #md.geometry
#surface is [-x * tan(3.0 * pi / 180)] #md.mesh
#->
md.geometry.surface = md.mesh.x * np.tan(3.0 * np.pi / 180.0)
#base is [surface - 1000 + 100 * exp(-((x - L / 2) .^ 2 + (y - L /
2) .^ 2) / (10000.^2))]
#L is the size of the side of the square #max(md.mesh.x) -
min(md.mesh.x)
#->
L = np.nanmax(md.mesh.x) - np.nanmin(md.mesh.x)
#->
md.geometry.base = md.geometry.surface - 1000.0 + 100.0 *
np.exp(-((md.mesh.x - L / 2.0) ** 2.0 + (md.mesh.y - L / 2.0) ** 2.0) / (10000.***2.0))
#thickness is the difference between surface and base #md.geometry
#->
md.geometry.thickness = md.geometry.surface - md.geometry.base
#plot the geometry to check it out
#->
plotmodel(md, 'data', md.geometry.thickness)

print('    Defining friction parameters')

#These parameters will not be used but need to be fixed #md.friction
#one friction coefficient per node (md.mesh.numberofvertices,1)
#conversion form year to seconds with #md.constants.yts
#->
md.friction.coefficient = np.sqrt(md.constants.yts / (1000 *
2.140373 * 1e-7)) * np.ones((md.mesh.numberofvertices))
#one friction exponent (p, q) per element
#->
md.friction.p = np.ones((md.mesh.numberofelements))
#->
md.friction.q = np.zeros((md.mesh.numberofelements))

print('    Construct ice rheological properties')

#The rheology parameters sit in the material section #md.materials
#B has one value per vertex
#->
md.materials.rheology_B = (1 / (2.140373 * 1e-7 / md.constants.yts))
    * np.ones((md.mesh.numberofvertices))
#n has one value per element
#->
md.materials.rheology_n = np.ones((md.mesh.numberofelements))

print('    Set boundary conditions')

```

```
#Set the default boundary conditions for an ice-sheet
# #help SetIceSheetBC
#->
md = SetIceSheetBC(md)

print('    Initializing velocity and pressure')

#initialize the velocity and pressurefields of #md.initialization
#->
md.initialization.vx = np.zeros((md.mesh.numberofvertices))
#->
md.initialization.vy = np.zeros((md.mesh.numberofvertices))
#->
md.initialization.vz = np.zeros((md.mesh.numberofvertices))
#->
md.initialization.pressure = np.zeros((md.mesh.numberofvertices))
```

4.2.6 Inversions

4.2.6.1 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

4.2.6.2 Introduction

Several model input parameters, such as the ice rigidity B (`md.materials.rheology_B`) and basal friction 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:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2} \left((v_x - v_x^{\text{obs}})^2 + (v_y - v_y^{\text{obs}})^2 \right) dS \quad (4.1)$$

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

Inversions were first introduced to glaciology by ? 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.

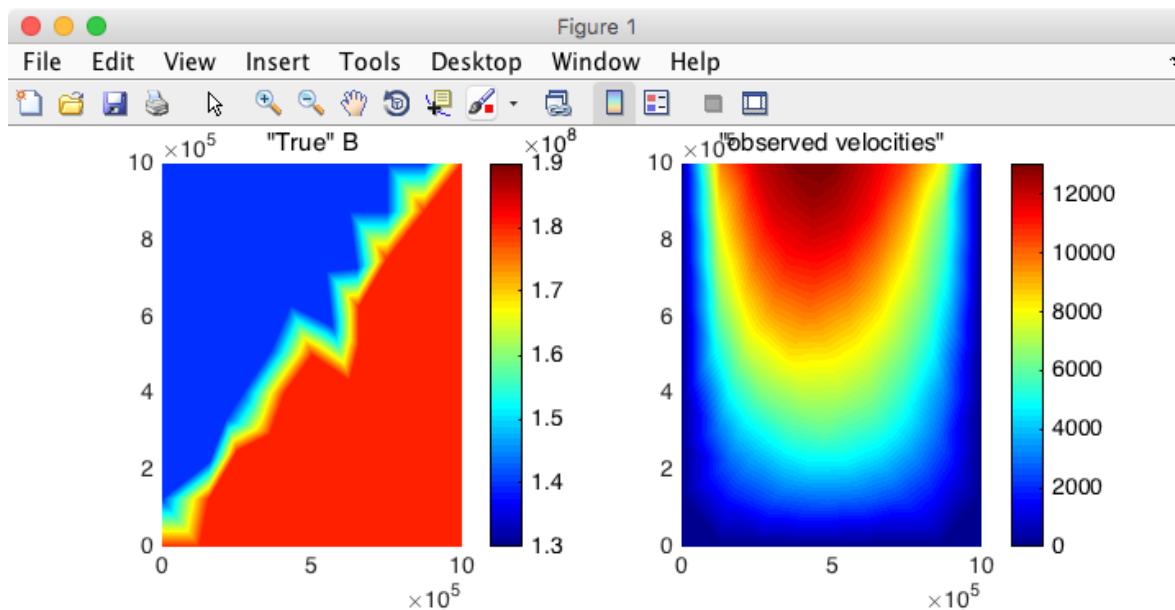
4.2.6.3 Hands on 1 (ice rigidity, B)

Step 1: Generating Observations

First, go to `<ISSM_DIR>/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
```

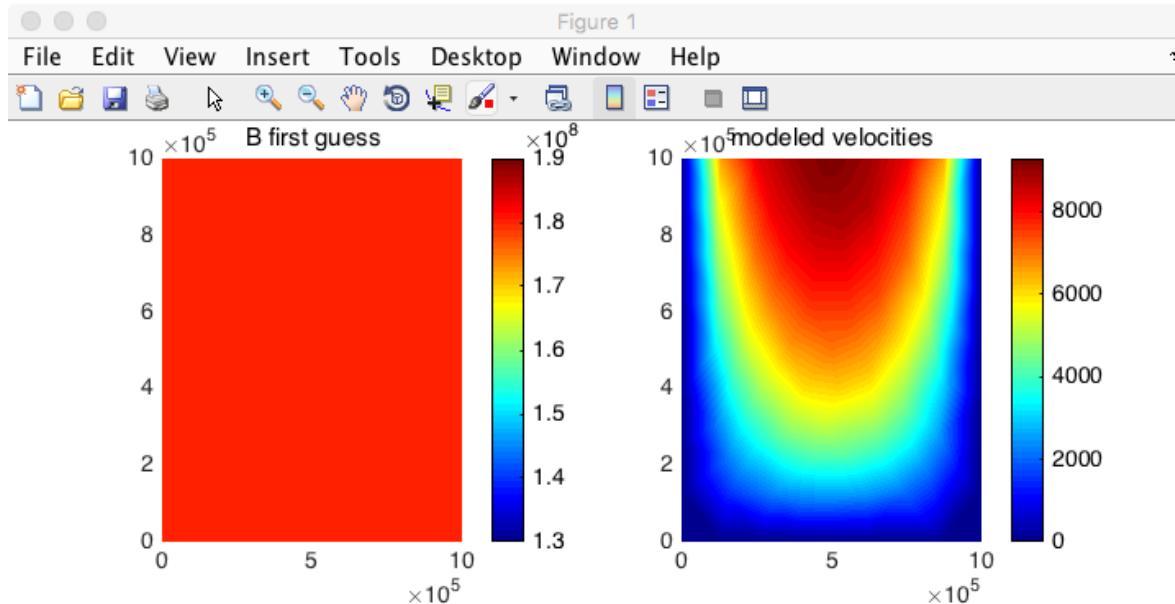
You will see on the left our prescribed rigidity, B , and to the right the calculated velocities. We choose a pattern with 2 distinct values for B for the upper left region, and stiffer ice for the lower right, with a sharp transition.



In the next step, we are 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.

Step 2: Initial guess and initial velocity

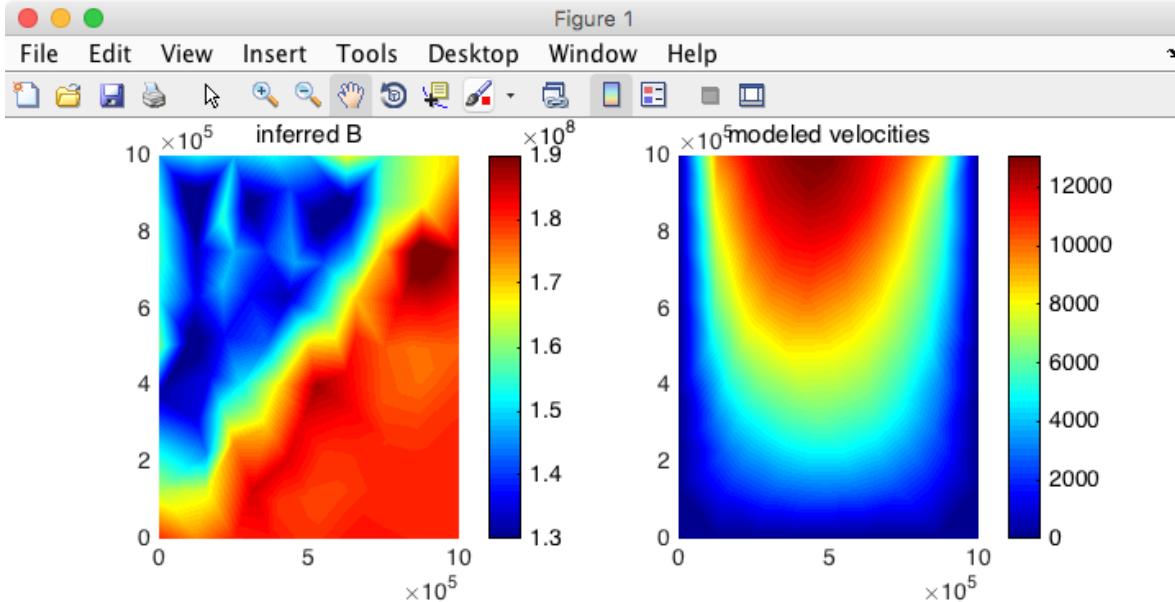
We now change the rigidity, B , and make it uniform. The results of the previous step are taken as observations (but we will only use them in step 3). Open `runme.m` and set `step = 2`. Save the file and execute step 2 in MATLAB as above.



We now see that the left panel is constant, and the velocity is symmetrical. 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.

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 are *not* physical. Adding regularization that penalizes wiggles in the control parameter stabilizes the inversion.

Step 4: Adding regularization

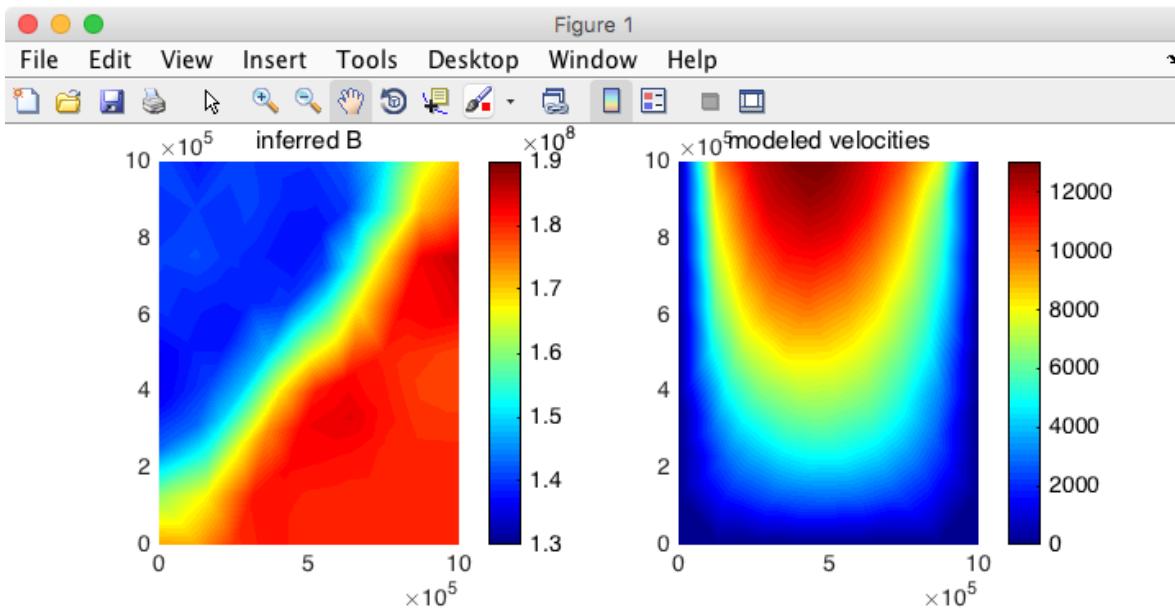
Here, we would like to add a term of regularization to our cost function:

$$\mathcal{J}(B) = \int_S w_1 \frac{1}{2} \left((v_x - v_x^{\text{obs}})^2 + (v_y - v_y^{\text{obs}})^2 \right) dS + \int_b w_2 \frac{1}{2} \|\nabla B\|^2 db \quad (4.2)$$

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.

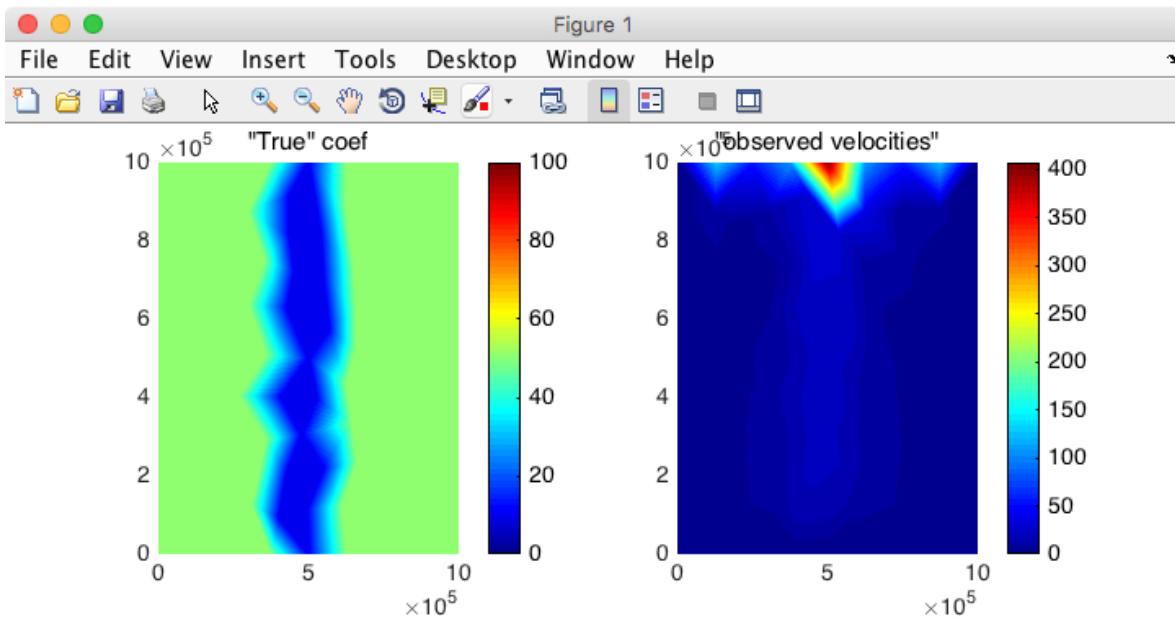
4.2.6.4 Hands on 2 (friction)

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 additions and/or modifications to the `runme.m` script as described below.

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 = 1.8×10^8
4. Friction coefficient: 50, and 10 for $400,000 < x < 600,000$
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.

Solutions to step 1 (MATLAB)

```
%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, 'Stressbalance');

%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
```

Solutions to step 1 (Python)

```
import numpy as np
from model import *
from setmask import setmask
from parameterize import parameterize
from setflowequation import setflowequation
from generic import generic
from solve import solve
from plotmodel import plotmodel
from export_netCDF import export_netCDF
from mlqn3inversion import mlqn3inversion
from verbose import verbose
from loadmodel import loadmodel
from cuffey import cuffey

steps = [5]
Clims = [1.3 * 1e8, 1.9 * 1e8]

if 1 in steps:
    #Generate observations
    md = model()
    md = triangle(md, 'DomainOutline.exp', 100000)
    #CHANGES START
    md = setmask(md, '', '')
    #CHANGES END
    md = parameterize(md, 'Square.py')
    #CHANGES START
    md.geometry.base = md.geometry.base + 100.
    md.geometry.surface = md.geometry.surface + 100.
    md.friction.coefficient[:] = 50
    pos = np.nonzero(np.logical_and(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, 'Stressbalance')
    #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

    export_netCDF(md, 'model1.nc')
```

```

if 2 in steps:
    #Modify rheology, now constant
    md = loadmodel('modell.nc')
    md.materials.rheology_B[:] = 1.8 * 1e8

    #results of previous run are taken as observations
    md.inversion = m1qn3inversion()
    md.inversion.vx_obs = md.results.StressbalanceSolution.Vx
    md.inversion.vy_obs = md.results.StressbalanceSolution.Vy
    md.inversion.vel_obs = md.results.StressbalanceSolution.Vel

    #CHANGES START
    md.friction.coefficient[:] = 50
    #CHANGES END

    md = solve(md, 'Stressbalance')
    #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
    export_netcdf(md, 'model2.nc')

if 3 in steps:
    #invert for ice rigidity
    md = loadmodel('model2.nc')

    #Set up inversion parameters
    maxsteps = 20
    md.inversion.iscontrol = 1
    #CHANGES START
    md.inversion.control_parameters = ['FrictionCoefficient']
    #CHANGES END
    md.inversion.maxsteps = maxsteps
    md.inversion.cost_functions = [101]
    md.inversion.cost_functions_coefficients =
        np.ones((md.mesh.numberofvertices, 1))
    #CHANGES START
    md.inversion.min_parameters = np.ones((md.mesh.numberofvertices,
                                           1))
    md.inversion.max_parameters = 100. *
        np.ones((md.mesh.numberofvertices, 1))
    #CHANGES END

    #Go solve!
    md.verbose = verbose(0)
    md = solve(md, 'Stressbalance')
    #CHANGES START
    plotmodel(md, 'axis#all', 'tight', 'data',
              md.results.StressbalanceSolution.FrictionCoefficient,
              'caxis', [0, 100], 'title', '"True" coef', 'data',
              md.results.StressbalanceSolution.Vel, 'title', '"observed velocities"')
    #CHANGES END

```

```

if 4 in steps:
    #invert for ice rigidity
    md = loadmodel('model2.nc')

    #Set up inversion parameters
    maxsteps = 20
    md.inversion.iscontrol = 1
    md.inversion.control_parameters = ['FrictionCoefficient']
    md.inversion.maxsteps = maxsteps
    #CHANGES START
    md.inversion.cost_functions = [101, 103]
    md.inversion.cost_functions_coefficients =
        np.ones((md.mesh.numberofvertices, 2))
    md.inversion.cost_functions_coefficients[:, 0] = 3000
    md.inversion.cost_functions_coefficients[:, 1] = 1
    #CHANGES END
    md.inversion.min_parameters = np.ones((md.mesh.numberofvertices,
        1))
    md.inversion.max_parameters = 100. *
        np.ones((md.mesh.numberofvertices, 1))

    #Go solve!
    md.verbose = verbose(0)
    md = solve(md, 'Stressbalance')
    #CHANGES START
    plotmodel(md, 'axis#all', 'tight', 'data',
        md.results.StressbalanceSolution.FrictionCoefficient,
        'caxis', [0, 100], 'title', '"True" coef', 'data',
        md.results.StressbalanceSolution.Vel, 'title', '"observed
        velocities"')
    #CHANGES END

if 5 in steps:
    #invert for ice rigidity
    md = loadmodel('model2.nc')

    #Set up inversion parameters
    maxsteps = 20
    md.inversion.iscontrol = 1
    md.inversion.control_parameters = ['FrictionCoefficient']
    md.inversion.maxsteps = maxsteps
    #CHANGES START
    md.inversion.cost_functions = [101, 103, 501]
    md.inversion.cost_functions_coefficients =
        np.ones((md.mesh.numberofvertices, 3))
    #CHANGES END
    md.inversion.cost_functions_coefficients[:, 0] = 3000
    md.inversion.cost_functions_coefficients[:, 1] = 1
    #CHANGES START
    md.inversion.cost_functions_coefficients[:, 2] = 0.01
    #CHANGES END
    md.inversion.min_parameters = np.ones((md.mesh.numberofvertices,
        1))
    md.inversion.max_parameters = 100. *
        np.ones((md.mesh.numberofvertices, 1))

```

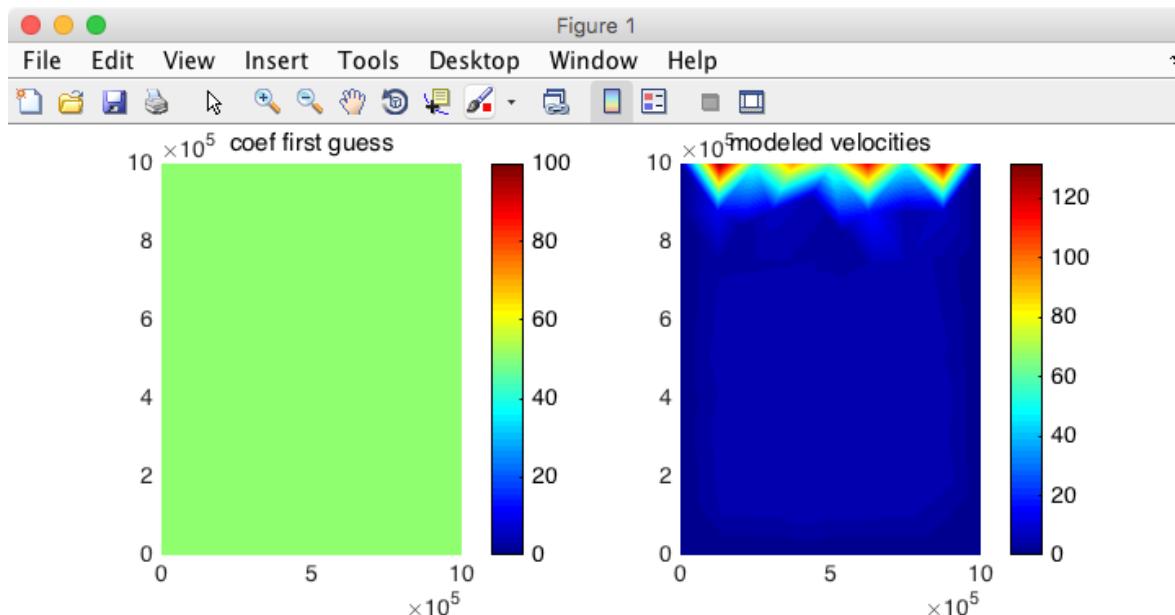
```
#Go solve!
md.verbose = verbose(0)
md = solve(md, 'Stressbalance')
plotmodel(md, 'axis#all', 'tight', 'data',
          md.results.StressbalanceSolution.FrictionCoefficient,
          'caxis', [0, 100], 'title', 'inferred B', 'data',
          md.results.StressbalanceSolution.Vel, 'title', 'modeled
velocities')
```

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.

Solutions to step 2

```
%Modify rheology, now constant
loadmodel('modell.mat');
md.materials.rheology_B(:) = 1.8 * 10^8;

%results of previous run are taken as observations
```

```

md.inversion = mlqn3inversion();
md.inversion.vx_obs      = md.results.StressbalanceSolution.Vx;
md.inversion.vy_obs      = md.results.StressbalanceSolution.Vy;
md.inversion.vel_obs     = md.results.StressbalanceSolution.Vel;

%CHANGES START
md.friction.coefficient(:) = 50;
%CHANGES END

md = solve(md, 'Stressbalance');
%CHANGES START
plotmodel(md, 'axis#all', 'tight', 'data', md.friction.coefficient,
    'caxis', [0 100], 'title', 'coeff first guess', ...
    'data', md.results.StressbalanceSolution.Vel, 'title', 'modeled
    velocities')
%CHANGES END
save model12 md

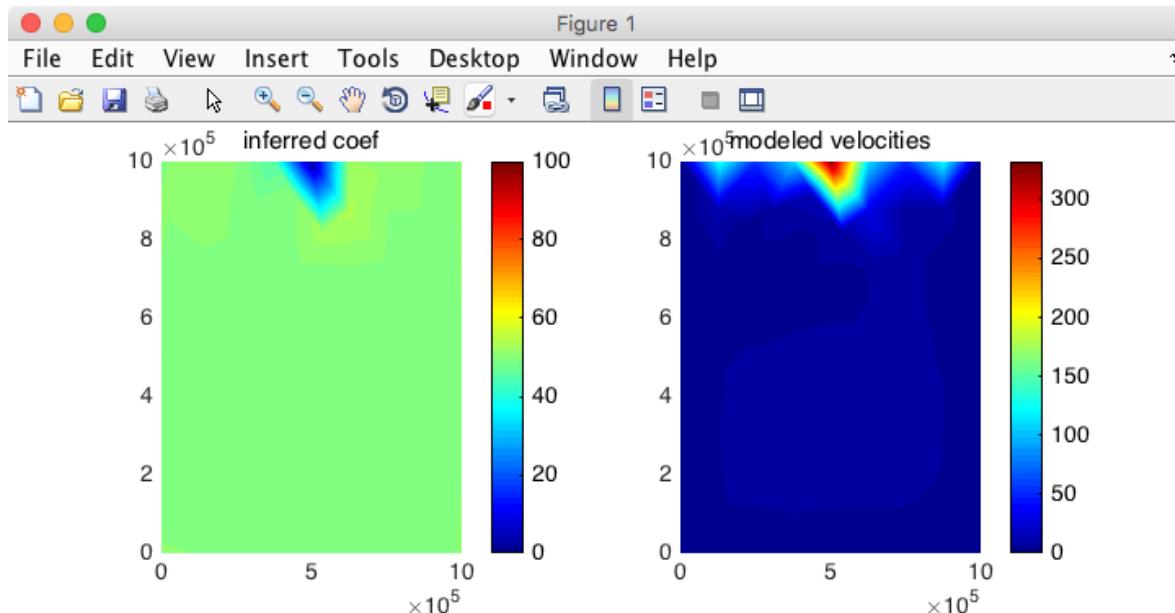
```

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 as follows,

Solutions to step 3

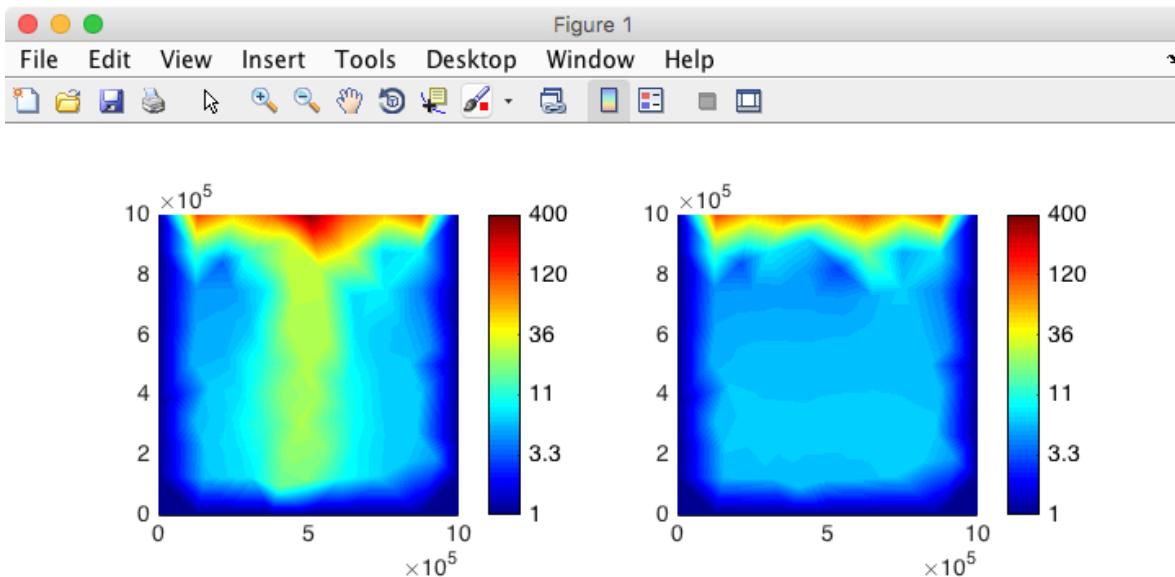
```
%invert for ice rigidity
loadmodel('model2.mat');

%Set up inversion parameters
maxsteps = 20;
md.inversion.iscontrol = 1;
%CHANGES START
md.inversion.control_parameters = {'FrictionCoefficient'};
%CHANGES END
md.inversion.maxsteps = maxsteps;
md.inversion.cost_functions = 101;
md.inversion.cost_functions_coefficients =
    ones(md.mesh.numberofvertices, 1);
%CHANGES START
md.inversion.min_parameters = ones(md.mesh.numberofvertices, 1);
md.inversion.max_parameters = 100*ones(md.mesh.numberofvertices, 1);
%CHANGES END

%Go solve!
md.verbose = verbose(0);
md = solve(md, 'Stressbalance');
%CHANGES START
plotmodel(md, 'axis#all', 'tight', 'data',
    md.results.StressbalanceSolution.FrictionCoefficient, 'caxis' ,
    [0 100], 'title', 'inferred coeff', ...
    'data', md.results.StressbalanceSolution.Vel, 'title', 'modeled
    velocities')
%CHANGES END
```

As you can see, we get more sliding close to the front, but the rest of the domain is unchanged. 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 (note, 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,
    'caxis#all', [1 400])
```



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.

Changing the cost function

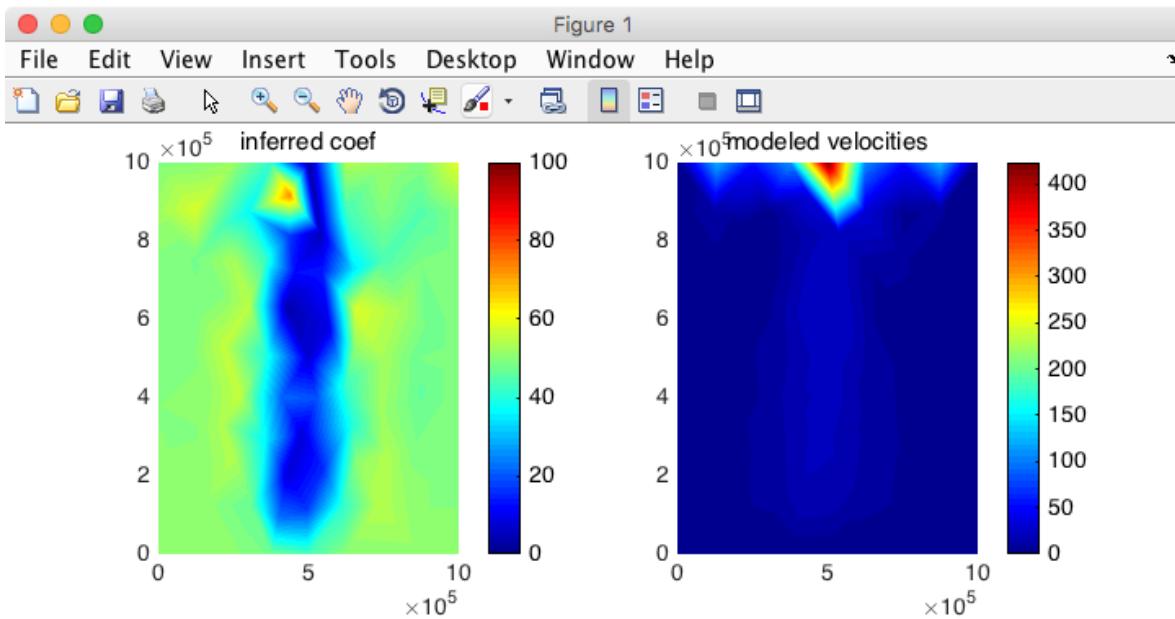
We want the cost function to include an additional term:

$$\mathcal{J}(\mathbf{v}) = \int_S w_1 \frac{1}{2} \left((v_x - v_x^{\text{obs}})^2 + (v_y - v_y^{\text{obs}})^2 \right) dS + \int_S w_2 \left(\log \left(\frac{\|\mathbf{v}\| + \varepsilon}{\|\mathbf{v}^{\text{obs}}\| + \varepsilon} \right) \right)^2 dS \quad (4.3)$$

The ‘Advanced Features’ → ‘Inversions’ 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.

Solutions to step 3b

```
%invert for ice rigidity
loadmodel('model2.mat');

%Set up inversion parameters
maxsteps = 20;
md.inversion.iscontrol = 1;
md.inversion.control_parameters = {'FrictionCoefficient'};
md.inversion.maxsteps = maxsteps;
%CHANGES START
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;
%CHANGES END
md.inversion.min_parameters = ones(md.mesh.numberofvertices, 1);
md.inversion.max_parameters = 100 * ones(md.mesh.numberofvertices,
1);

%Go solve!
md.verbose=verbose(0);
md = solve(md, 'Stressbalance');
%CHANGES START
plotmodel(md, 'axis#all', 'tight', 'data',
    md.results.StressbalanceSolution.FrictionCoefficient, 'caxis', [0
    100], 'title', 'inferred coeff', ...
    'data', md.results.StressbalanceSolution.Vel, 'title', 'modeled
    velocities')
```

```
%CHANGES END
```

Adding regularization

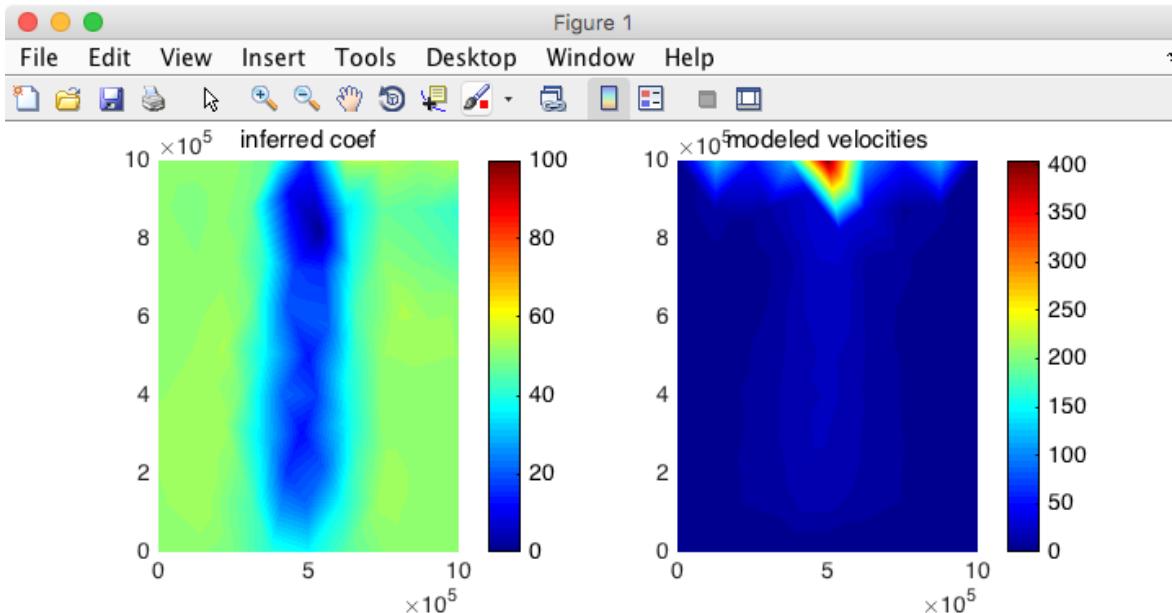
We want the cost function to include a regularization term:

$$\mathcal{J} = \int_S w_1 \frac{1}{2} \left((v_x - v_x^{\text{obs}})^2 + (v_y - v_y^{\text{obs}})^2 \right) dS + \int_S w_2 \left(\log \left(\frac{\|\mathbf{v}\| + \varepsilon}{\|\mathbf{v}^{\text{obs}}\| + \varepsilon} \right) \right)^2 dS + \int_B w_3 \frac{1}{2} \|\nabla \alpha\|^2 dB \quad (4.4)$$

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!

Solutions to step 3c

```
%invert for ice rigidity
loadmodel('model2.mat');

%Set up inversion parameters
maxsteps = 20;
md.inversion.iscontrol = 1;
md.inversion.control_parameters = {'FrictionCoefficient'};
```

```
md.inversion.maxsteps = maxsteps;
%CHANGES START
md.inversion.cost_functions = [101 103 501];
md.inversion.cost_functions_coefficients =
    ones(md.mesh.numberofvertices, 3);
%CHANGES END
md.inversion.cost_functions_coefficients(:, 1) = 3000;
md.inversion.cost_functions_coefficients(:, 2) = 1;
%CHANGES START
md.inversion.cost_functions_coefficients(:, 3) = 0.01;
%CHANGES END
md.inversion.min_parameters = ones(md.mesh.numberofvertices, 1);
md.inversion.max_parameters = 100 * ones(md.mesh.numberofvertices,
1);

%Go solve!
md.verbose = verbose(0);
md = solve(md, 'Stressbalance');
plotmodel(md, 'axis#all', 'tight', 'data',
    md.results.StressbalanceSolution.FrictionCoefficient, 'caxis', [0
100], 'title', 'inferred coeff', ...
'data', md.results.StressbalanceSolution.Vel, 'title', 'modeled
velocities')
```

4.2.7 Modeling the Greenland Ice Sheet (SeaRISE)

4.2.7.1 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 `<ISSM_DIR>/examples/Greenland/` to do this tutorial.

4.2.7.2 Introduction

In this tutorial, you will learn how to set up a continental Greenland model using the SeaRISE ice sheet model input dataset [?]. 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

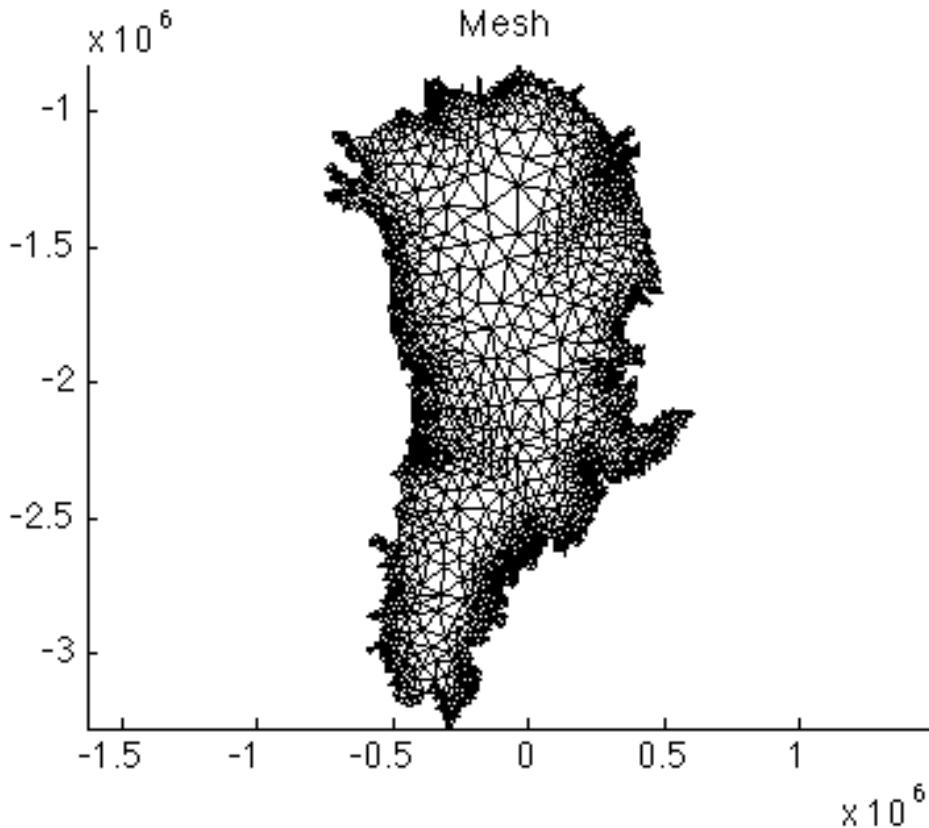
4.2.7.3 Mesh

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 400 km where the velocity gradient is small. The velocity 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 x and y coordinates from a NetCDF file)
- 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:

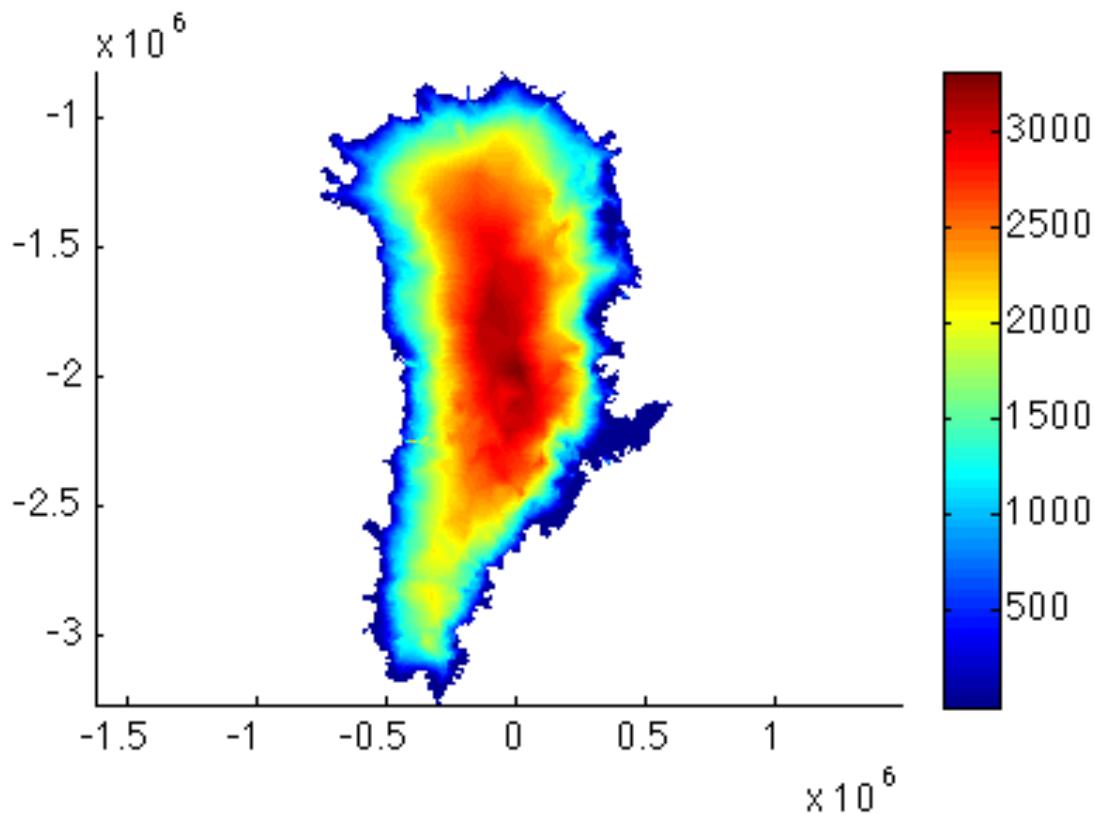


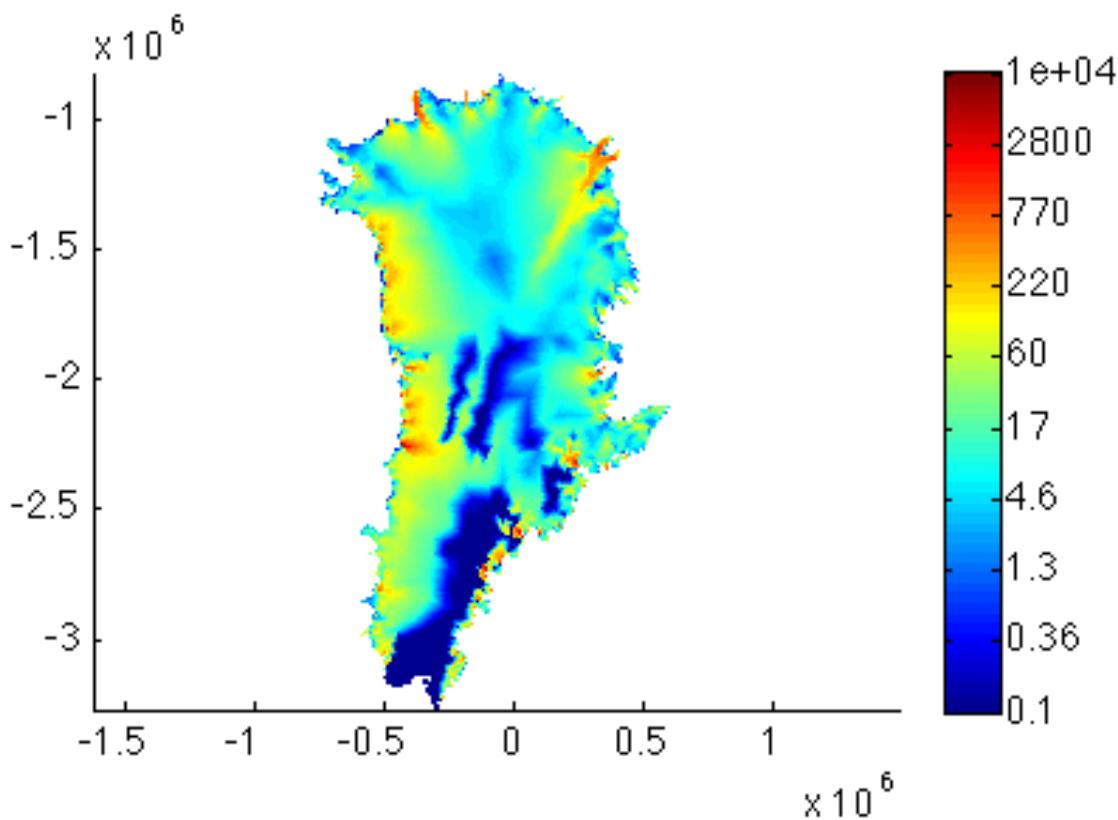
4.2.7.4 Parameterization

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)
```





4.2.7.5 Stress Balance

Use control methods to inversely solve for Greenland FrictionCoefficient (Step 3, lines 44-81).

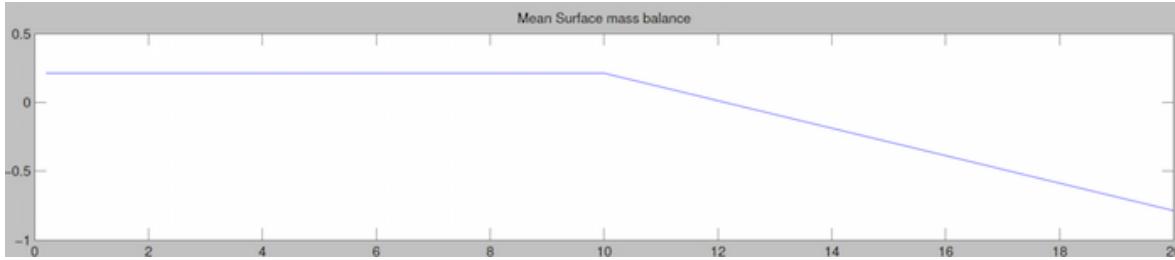
NOTE: Remember that `md.inversion` can be called for help!

- Set three different cost functions
 - Absolute value of surface velocity
 - Log of surface velocity
 - Drag coefficient gradient
- Set cost functions coefficients to 350, 0.2, 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.

4.2.7.6 Transient

You are now ready to run a transient! In Step 4, we will simulate a simple constant warming trend over Greenland 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]]
```

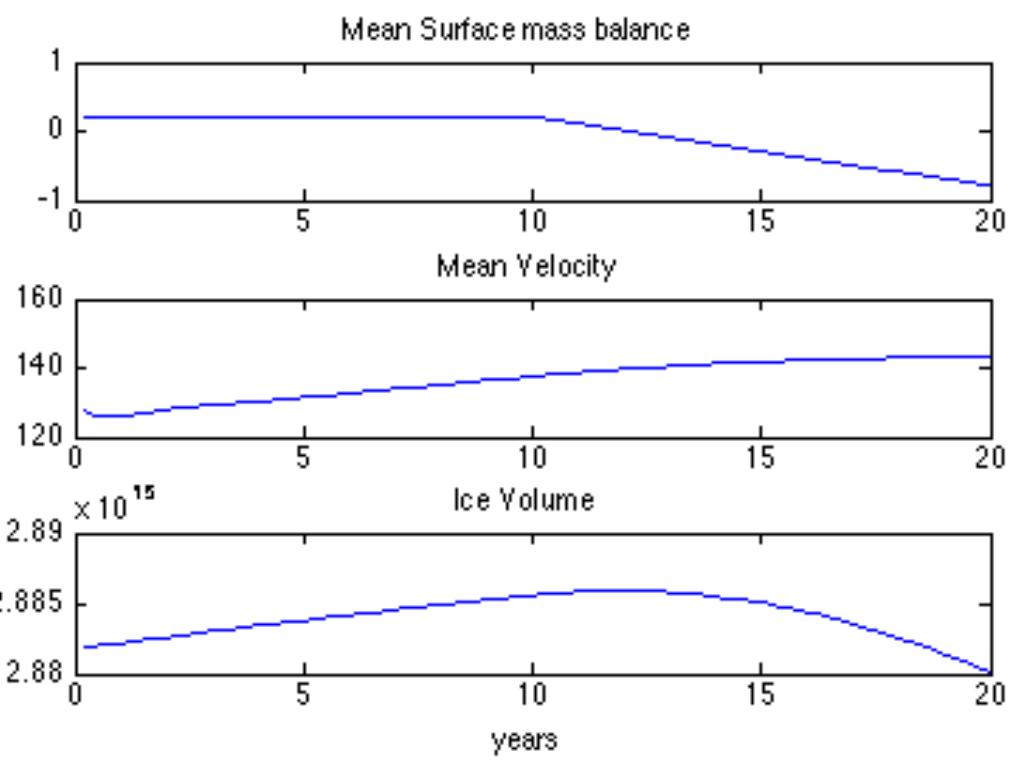
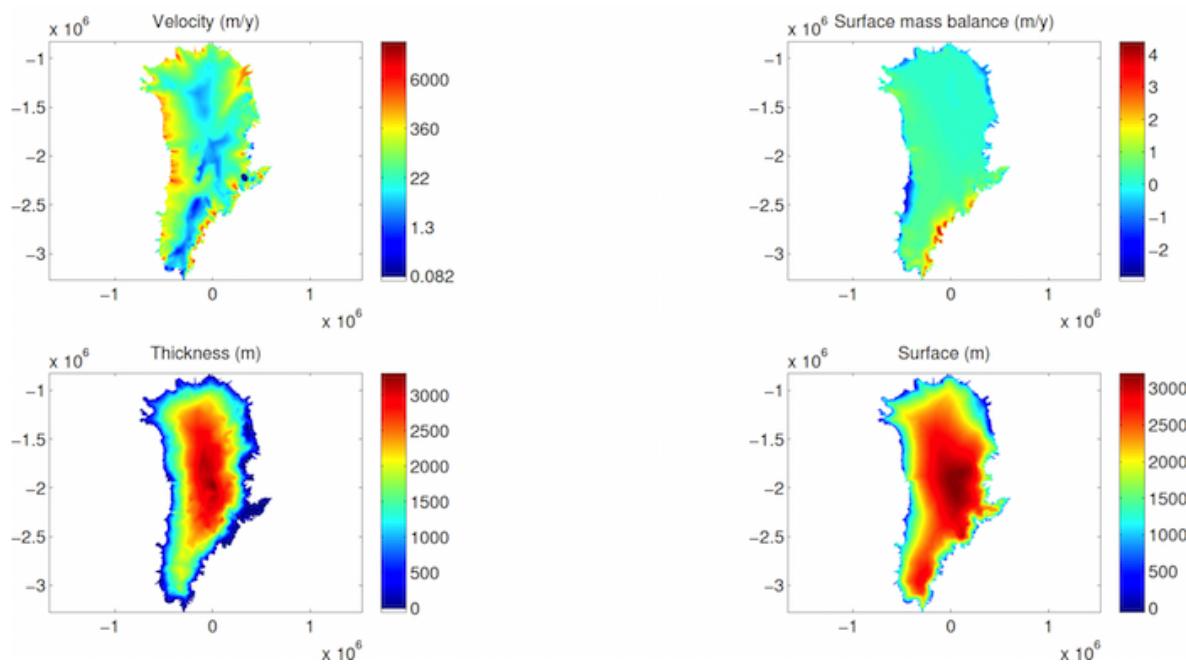
By default, ISSM will linearly interpolate surface mass balance between time 10 and time 20 in this example. Prior to first and after last imposed time, forcing values remain constant. In order to turn interpolation off (i.e. use a step function), you would set `md.timestepping.interp_forcings = 0`.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

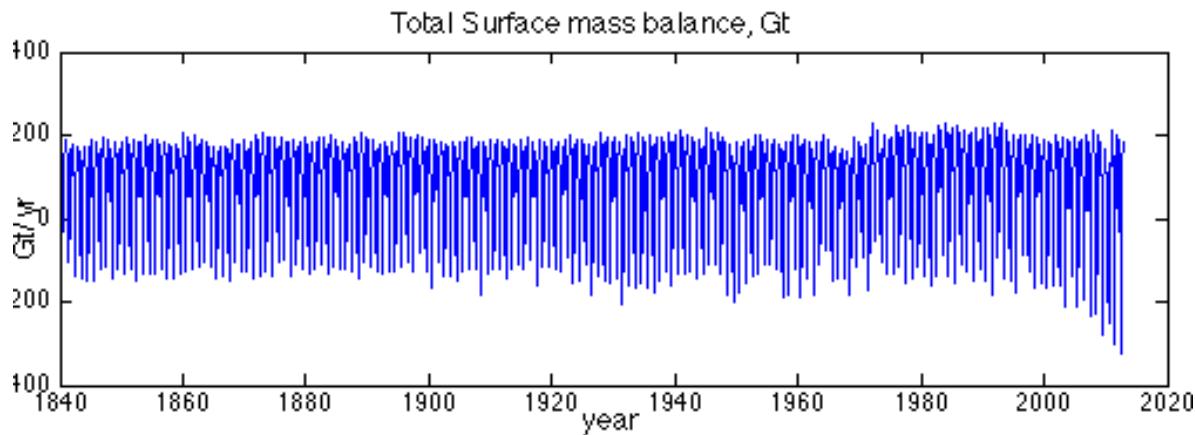


4.2.7.7 Exercise

Now, let's run our transient with historical mass balance! Use Jason Box's surface mass balance (SMB) time series as forcing [??].¹

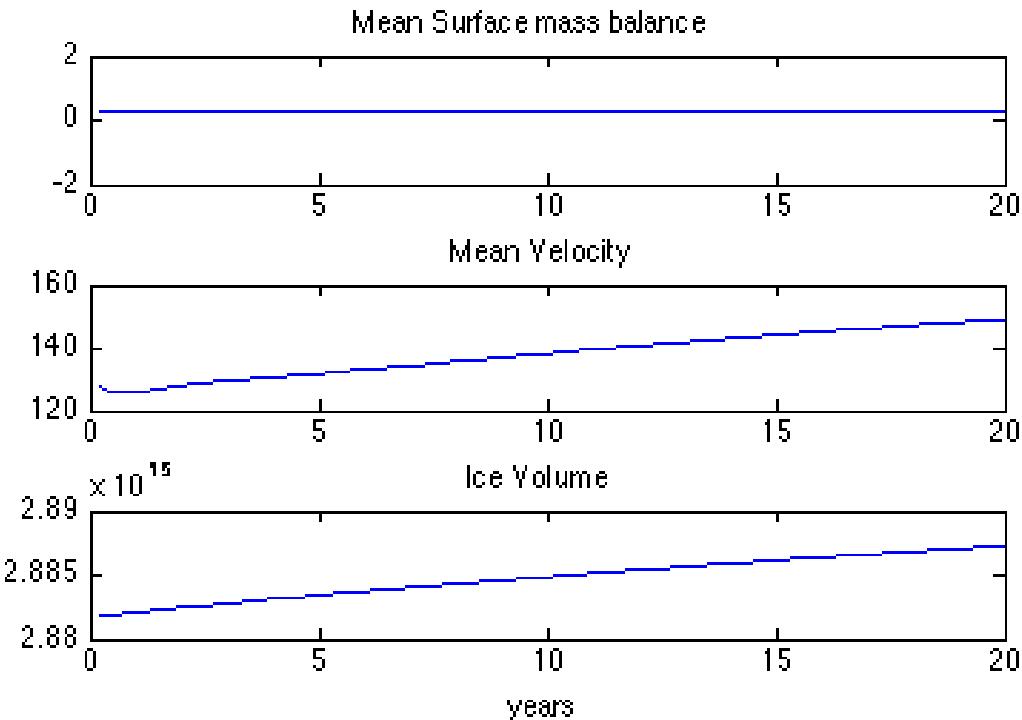
First, format the SMB provided. In Step 6 of the `runme.m` file, we extract the SMB timeseries from

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.



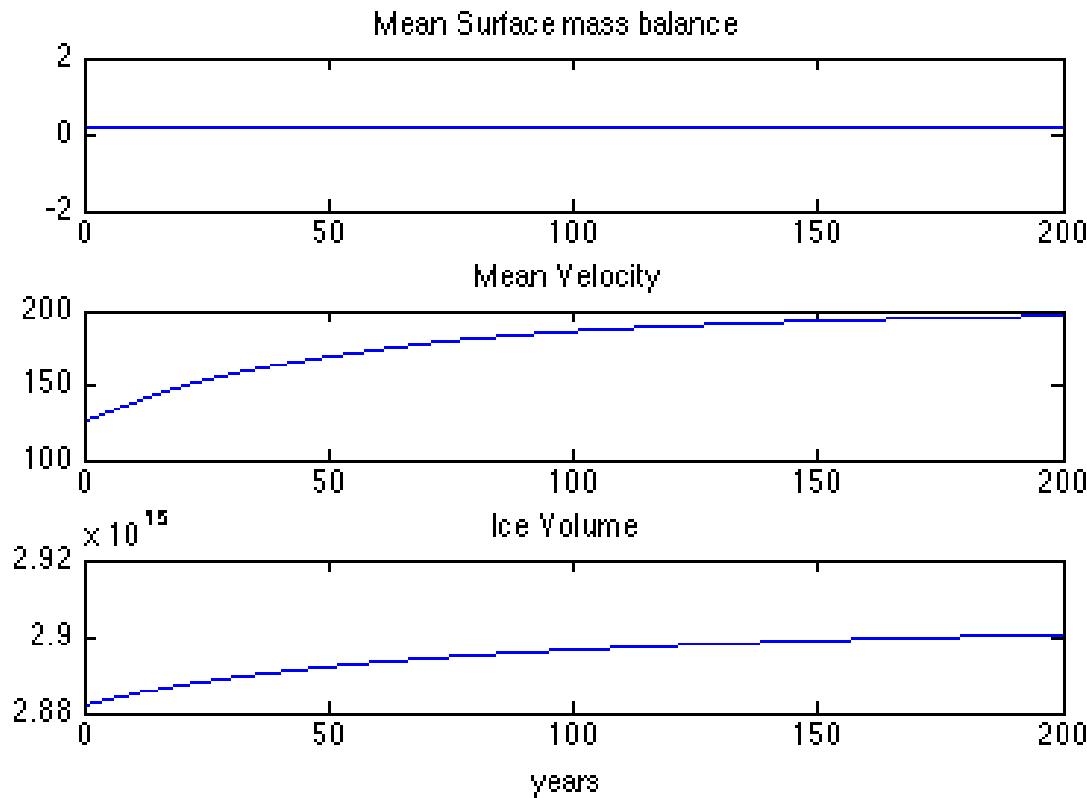
To reach equilibrium, the model should run on the order of 3000 years. Since, 3000 years might take quite a long time to run on a personal computer, you may want to try running for 200 years instead.

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 original 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:

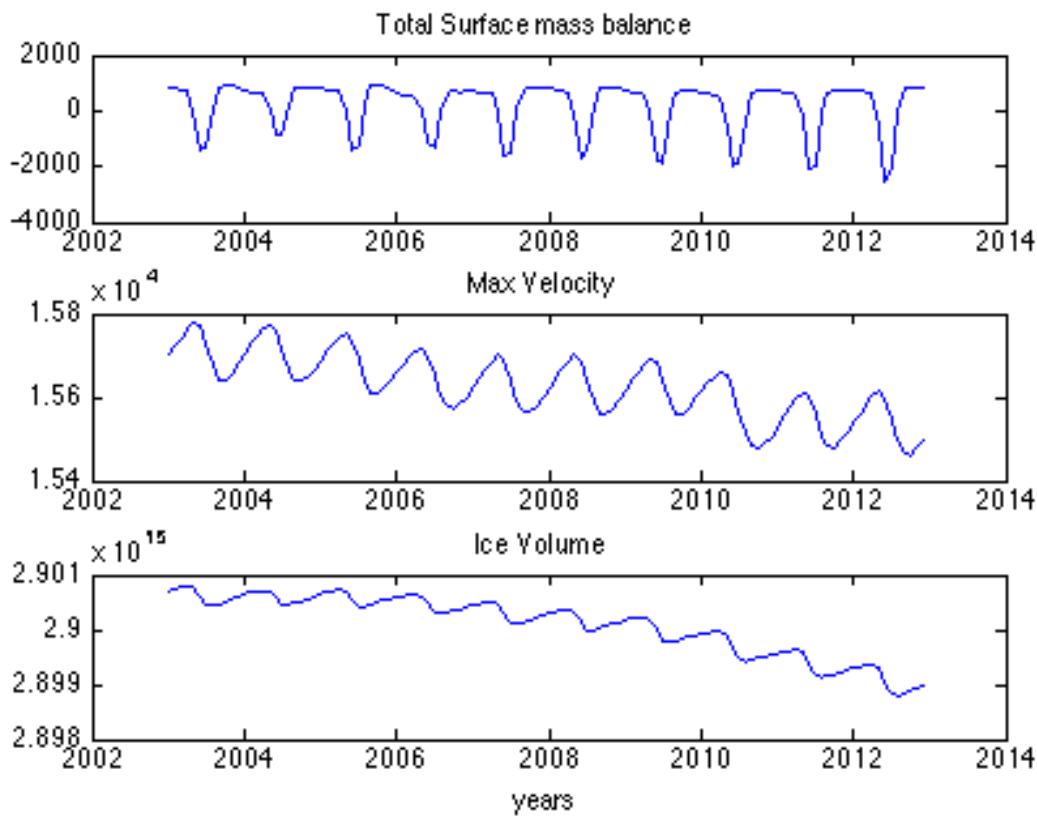


In Step 9, we will use the 200 year relaxed ice sheet as a starting condition for a historic transient run. To do so we need to save the 200 year resulting geometry and velocities into the model state. To load your past results see lines 254-259.

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 execute 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.



4.2.7.8 Solution for step 7

```

if any(steps == 7)
    disp(' Step 7: Historical Relaxation run');
    md = loadmodel('./Models/Greenland.Control_drag');

    load smbbox

    %convert mesh x,y into the Box projection
    [md.mesh.lat, md.mesh.long] = xy2ll(md.mesh.x, md.mesh.y, +1, 39,
        71);
    [xi, yi] = ll2xy(md.mesh.lat, md.mesh.long, +1, 45, 70);

    %Interpolate and set surface mass balance
    index = BamgTriangulate(x1(:), y1(:));
    smb_mo = InterpFromMeshToMesh2d(index, x1(:), y1(:), smbmean(:),
        xi, yi);
    smb = smb_mo * 12 / 1000 * md.materials.rho_freshwater /
        md.materials.rho_ice;
    md.smb.mass_balance = [smb; 1];

```

```
%Set transient options, run for 20 years, saving every 5 timesteps
md.timestepping.time_step = 0.2;
md.timestepping.final_time = 200;
md.settings.output_frequency = 5;

%Additional options
md.inversion.iscontrol = 0;
md.transient.requested_outputs = {'IceVolume', 'TotalSmb', ...
    'SmbMassBalance'};
md.verbose = verbose('solution', true, 'module', true);

%Go solve
md.cluster = generic('name', oshostname, 'np', 2);
md = solve(md, 'Transient');

save ./Models/Greenland.HistoricTransient_200yr md;
end
```

4.2.7.9 Solution for step 8

```
if any(steps == 8)
    %Load historic transient model
    md = loadmodel('./Models/Greenland.HistoricTransient_200yr');

    %Create Line Plots of relaxation run. Create a figure.
    figure;

    %Save surface mass balance, by looping through 200 years (1000
    %steps)
    %Note, the first output will always contain output from time step 1
    surfmb = [];
    for i=2:201;
        surfmb = [surfmb md.results.TransientSolution(i).SmbMassBalance];
    end

    %Plot surface mass balance time series in first subplot
    subplot(3, 1, 1);
    plot([1:200], mean(surfmb));

    %Title this plot Mean surface mass balance
    title('Mean Surface mass balance');

    %Save velocity by looping through 200 years
    vel = [];
    for i=2:201;
        vel = [vel md.results.TransientSolution(i).Vel];
    end

    %Plot velocity time series in second subplot
    subplot(3, 1, 2);
    plot([1:200], mean(vel));

    %Title this plot Mean Velocity
```

```

title('Mean Velocity');

%Save Ice Volume by looping through 200 years
volume=[];
for i=2:201;
    volume = [volume md.results.TransientSolution(i).IceVolume];
end

%Plot volume time series in third subplot
subplot(3, 1, 3);
plot([1:200], volume);

%Title this plot Mean Velocity and add an x label of years
title('Ice Volume');
xlabel('years');
end

```

4.2.7.10 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!

4.2.7.11 Footnotes

1. The year 1840-2012 Greenland near surface air temperature (T) and land ice SMB reconstruction after Box [2013] is calibrated to RACMO2 output [????]. 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 (\sim 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 [??] 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.

4.2.8 Modeling the Greenland Ice Sheet Using IceBridge Data

4.2.8.1 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 `<ISSM_DIR>/examples/IceBridge/` to do this tutorial.

4.2.8.2 Introduction

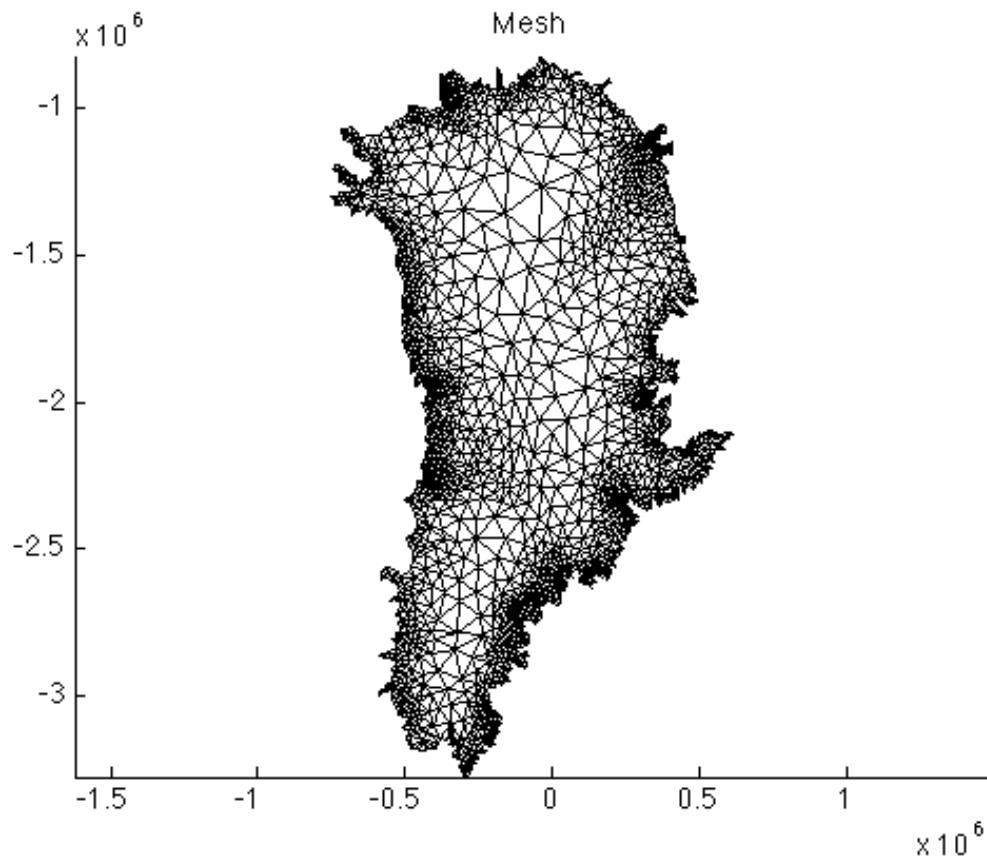
Tutorial steps to be taken:

- 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.

4.2.8.3 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:

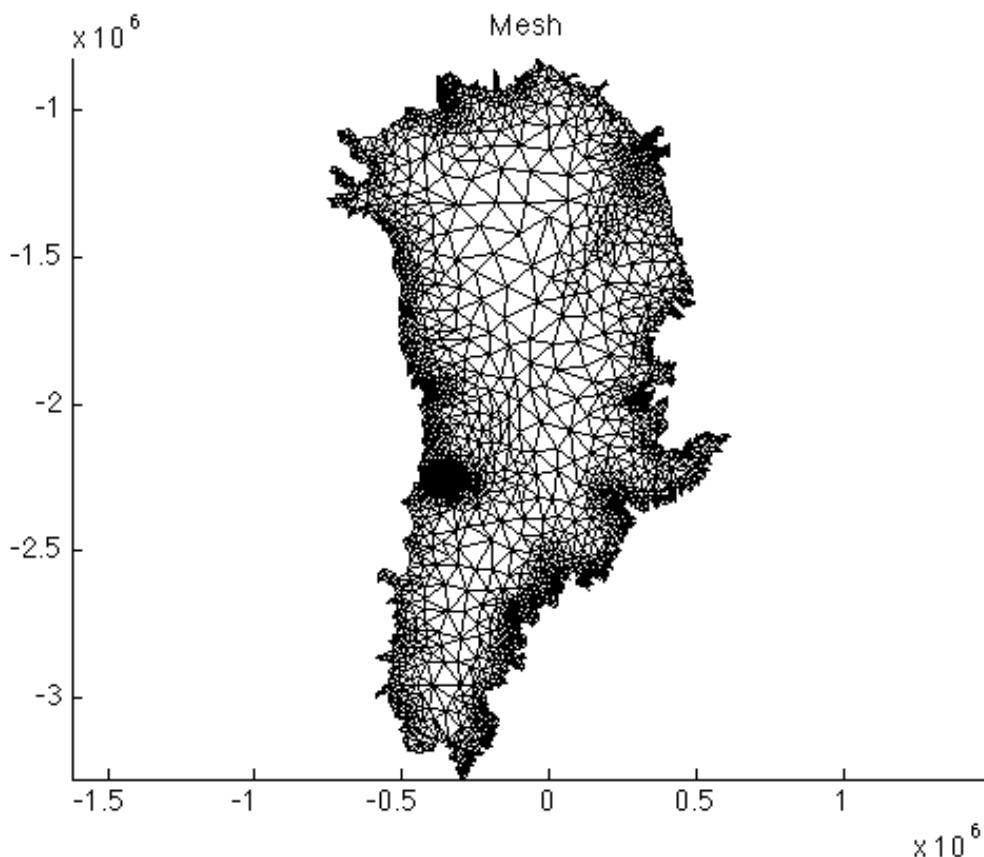
```
>> plotmodel(md, 'data', 'mesh');
```



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');
```

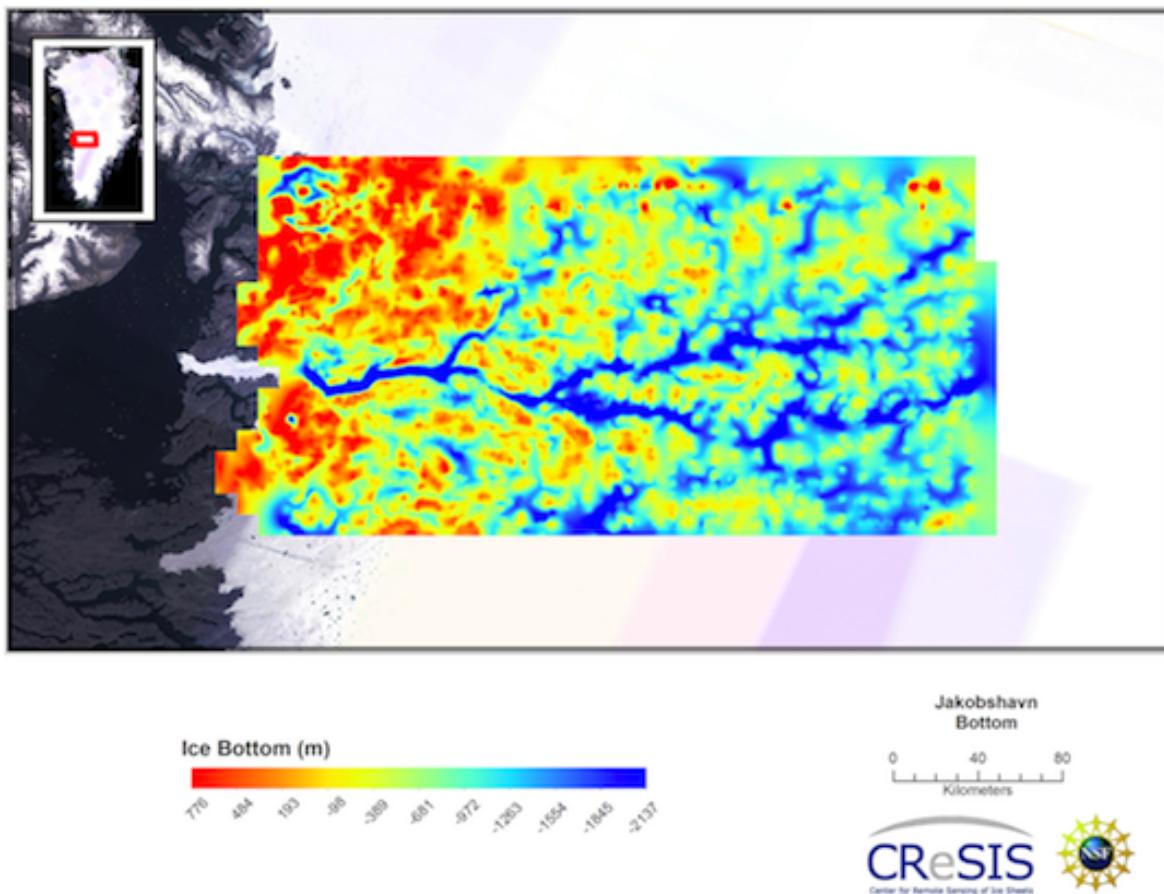
Next, we modify the `bamg` command by imposing a 3 km resolution within the JI area using `hmaxVertices`. Note that, to implement the changes noted above you must deactivate the first occurrence of the `bamg` command in step 1, as well as the `return` command. Do this by commenting out these lines, and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



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

4.2.8.4 Parameterization

We want to include high-resolution bedrock and surface elevation data acquired in the OIB mission. The data is accessible on University of Kansas' CReSIS [Open Polar Radar Data Products page](#). 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/grids/Jakobshavn_2008_2011_Composite'
titles = fgets(fid);
data = fscanf(fid, '%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, ...
'Jak_grounded.exp', 'node', 1);
bed_jks(~in) = NaN;
pos = find(~isnan(bed_jks));
md.geometry.base(pos) = bed_jks(pos);
```

```
md.geometry.thickness = md.geometry.surface - md.geometry.base;
```

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, ...
    '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:

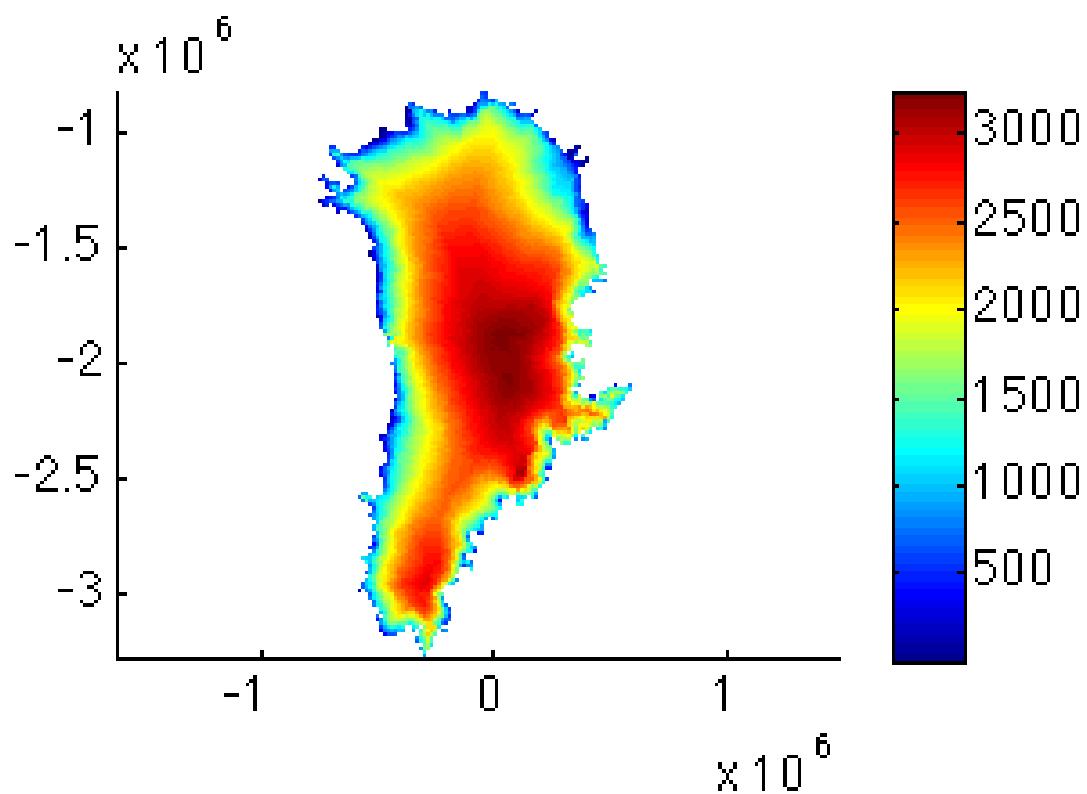


Figure 4.1: `plotmodel(md, 'data', md.geometry.surface)`

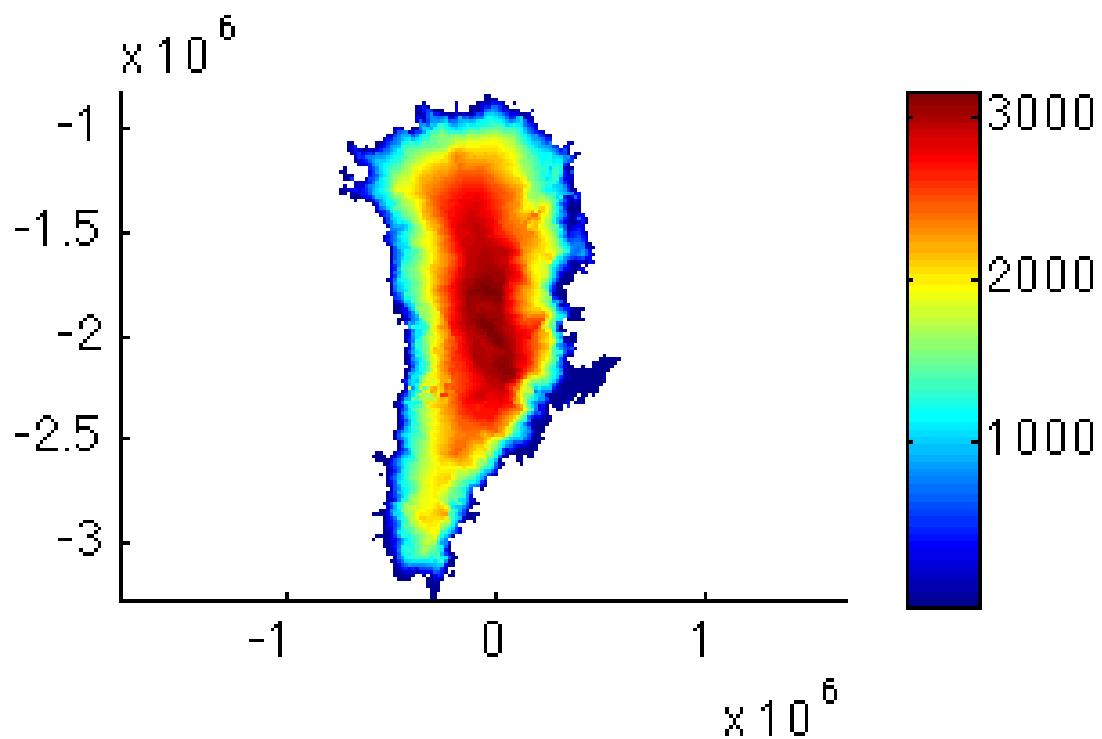


Figure 4.2: `plotmodel(md, 'data', md.geometry.thickness)`

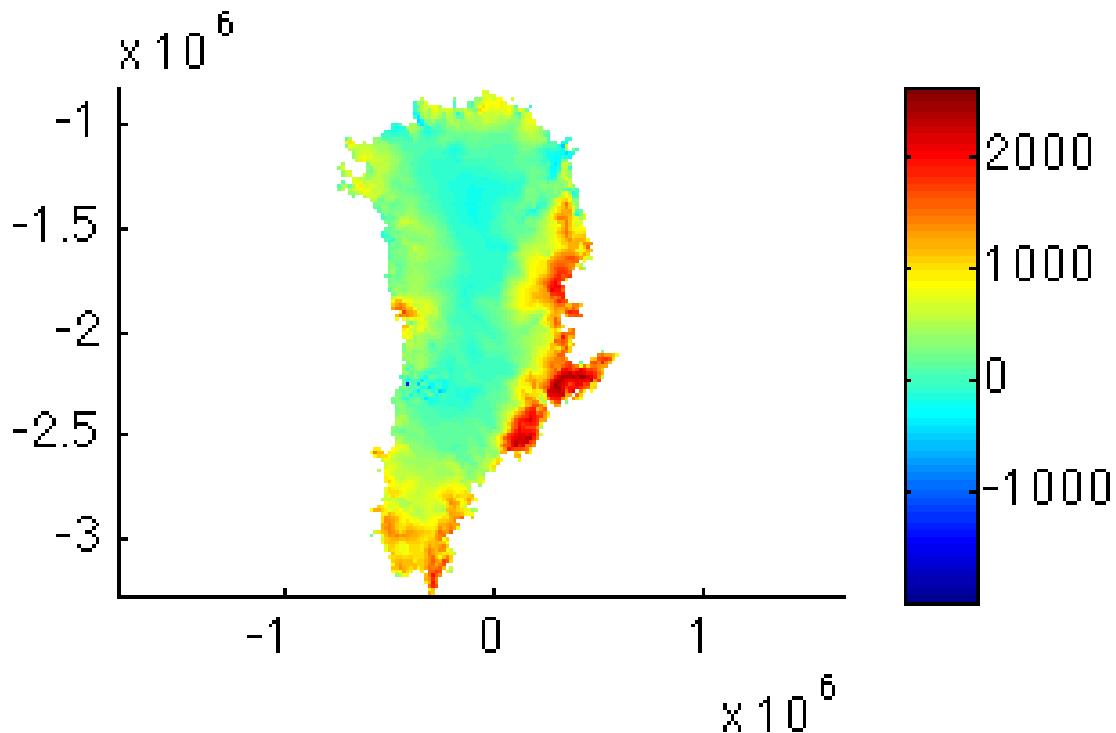
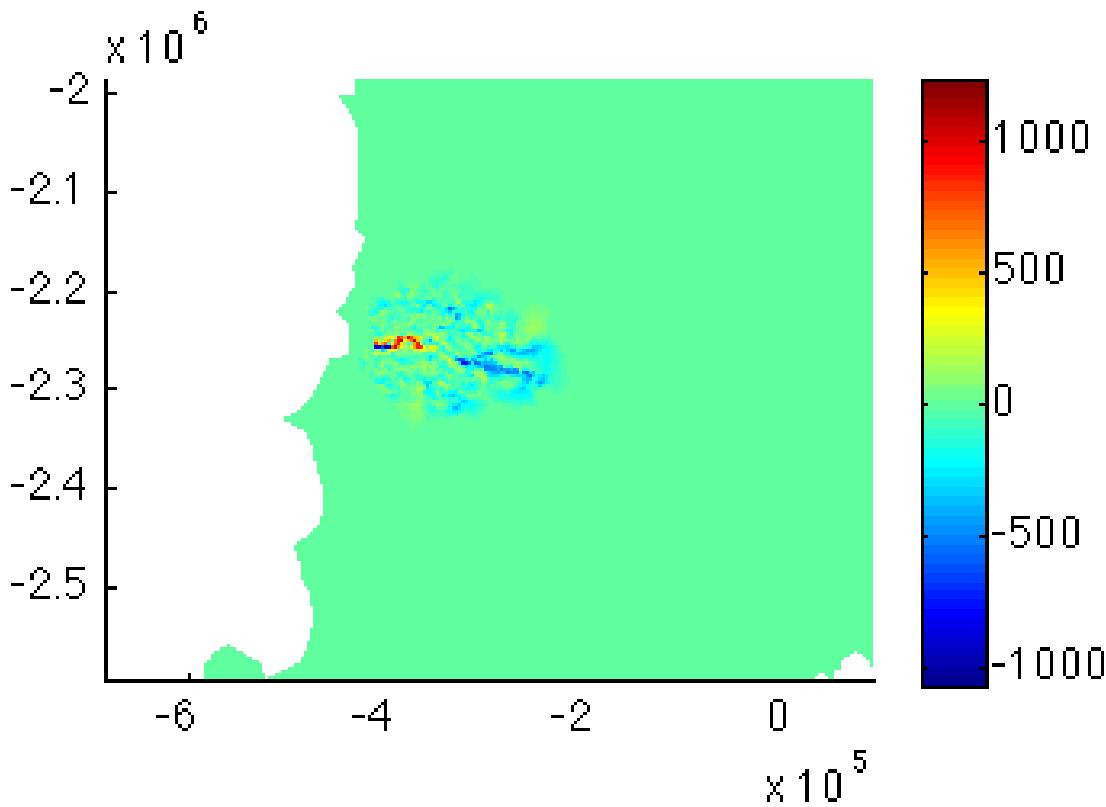


Figure 4.3: `plotmodel(md, 'data', md.geometry.base)`

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.



4.2.8.5 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')
```

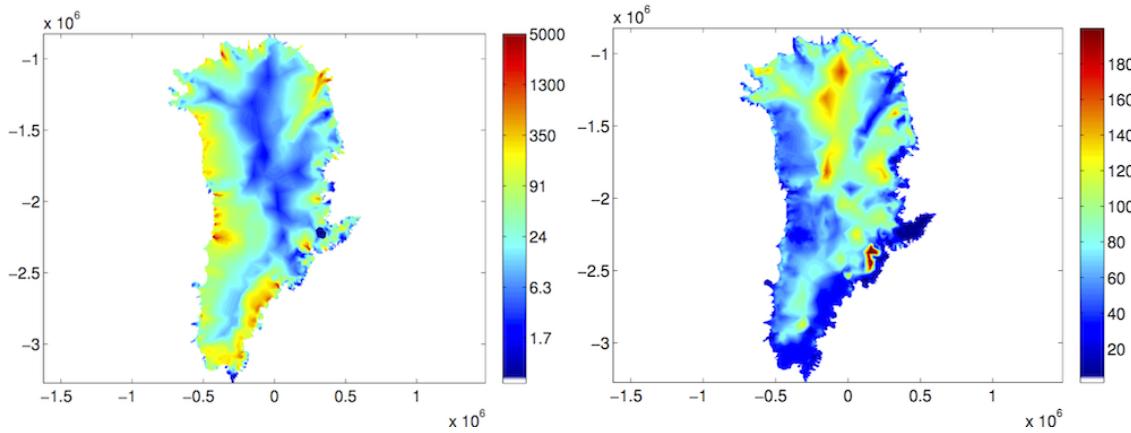
Exclude these data gaps in the inversion by giving them zero weight during the inversion process:

```
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:



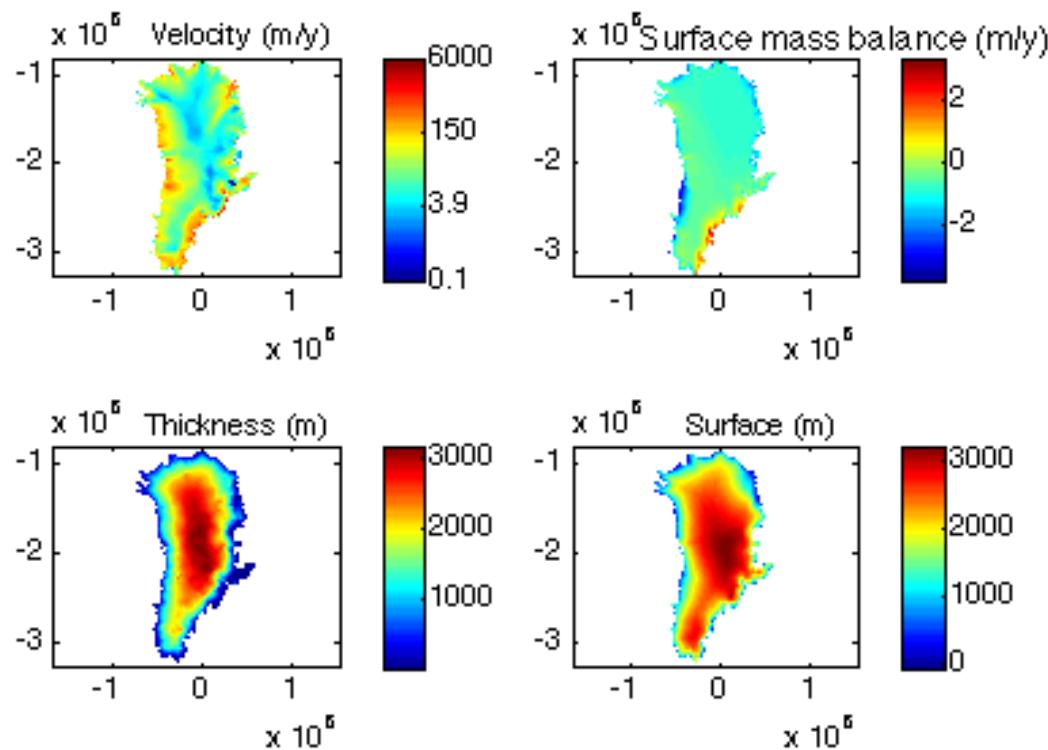
Even at this coarse resolution we can identify the high friction values inland and lower values towards the coast, which may be related to the basal thermal regime of the ice sheet.

4.2.8.6 Transient

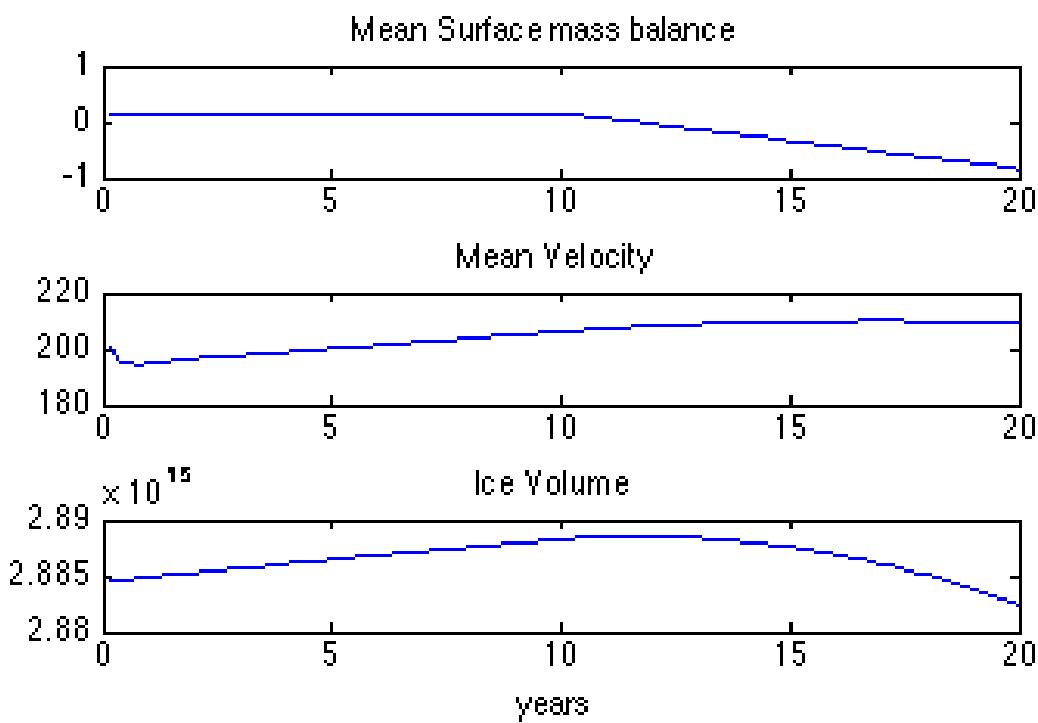
Finally, do a transient run (step 4) for 20 years, and decrease the surface mass balance linearly by 1 m w.e./yr over the last 10 years (`ncdata = '../Data/Greenland_5km_dev1.2.nc';`).

```
%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:



4.2.8.7 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?

4.2.9 Modeling Pine Island Glacier

4.2.9.1 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

4.2.9.2 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:

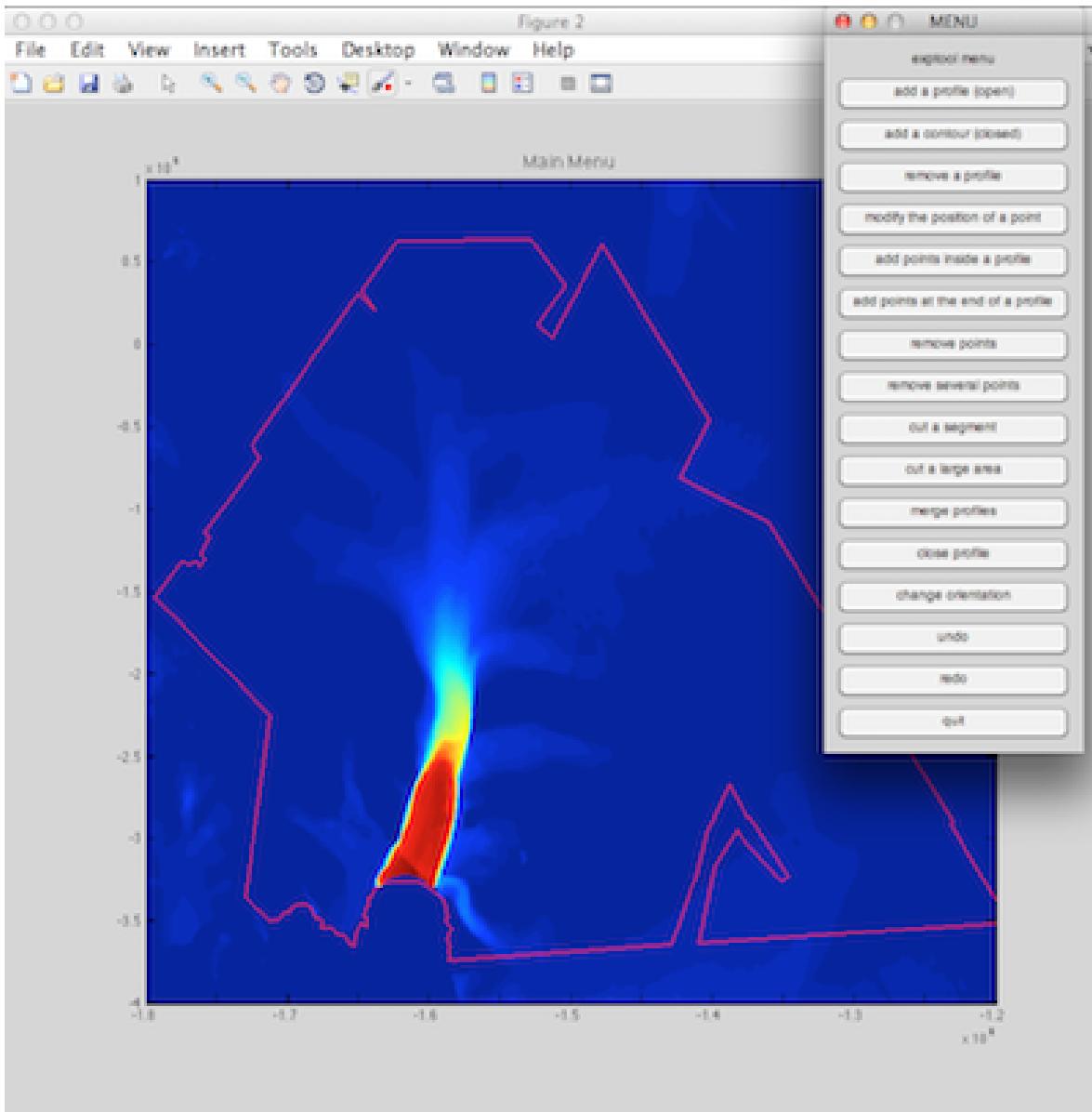
- Define the model region
- Create a mesh
- Apply masks
- Parameterize the model
- Invert friction coefficient
- Plot results
- Run higher-order simulation

Files needed for this tutorial can be found in `<ISSM_DIR>/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](#).

4.2.9.3 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`. Rename this file `ModelDomain.exp` (which will, effectively, erase your contour):

```
>>!mv DomainOutline.bkp DomainOutline.exp
```

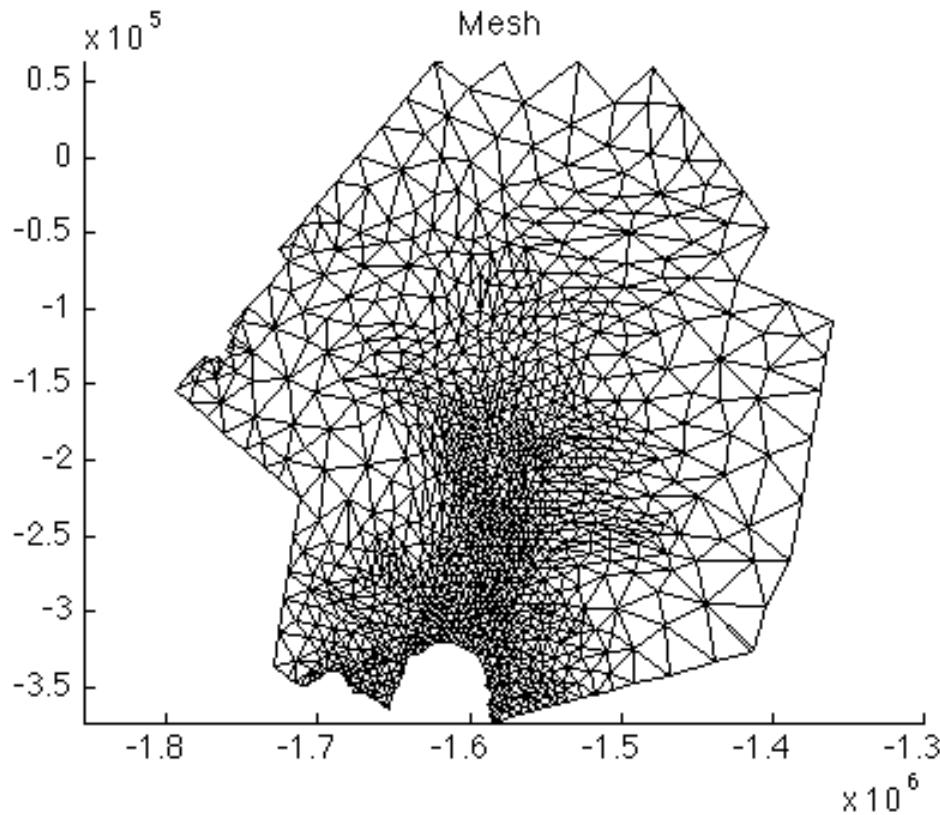
4.2.9.4 Mesh

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 `steps`, at the top of the file, is set to `steps = [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:



4.2.9.5 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:

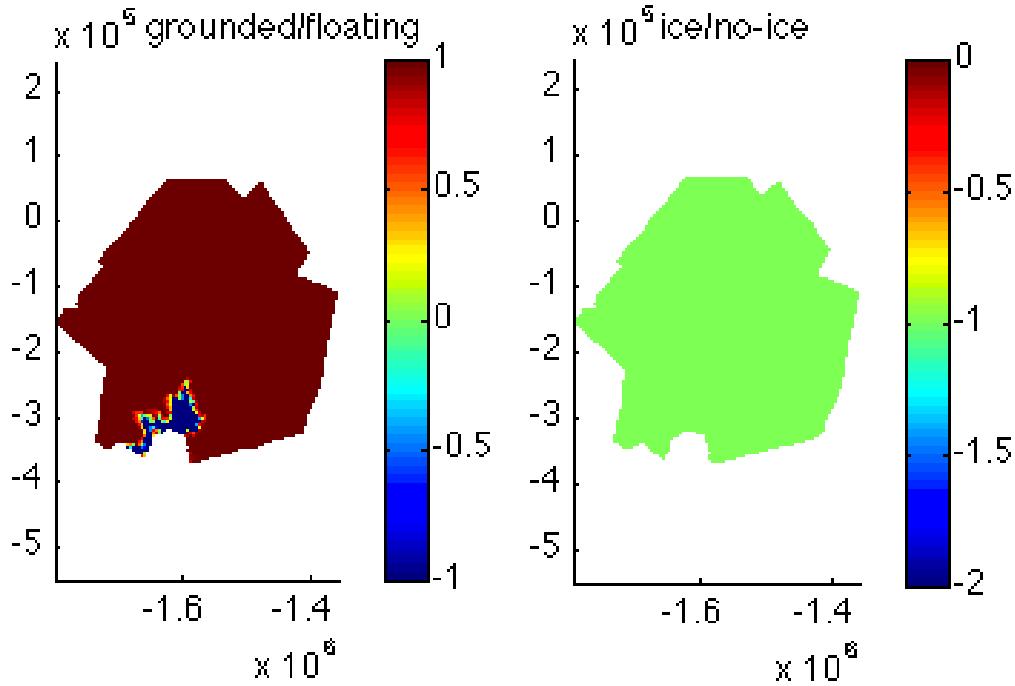
- The field `md.mask.ocean_levelset` contains this information
 - Ice is grounded if `md.mask.ocean_levelset` is positive
 - Ice is floating if `md.mask.ocean_levelset` is negative
 - The grounding line lies where `md.mask.ocean_levelset` equals zero

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 positive
 - The ice front lies where `md.mask.ice_levelset` equals zero

Open `runme.m` and set `steps = [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:



4.2.9.6 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.

4.2.9.7 Inversion of basal friction

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

$$\tau_b = -\beta^2 N^r \|\mathbf{v}_b\|^{s-1} \mathbf{v}_b \quad (4.5)$$

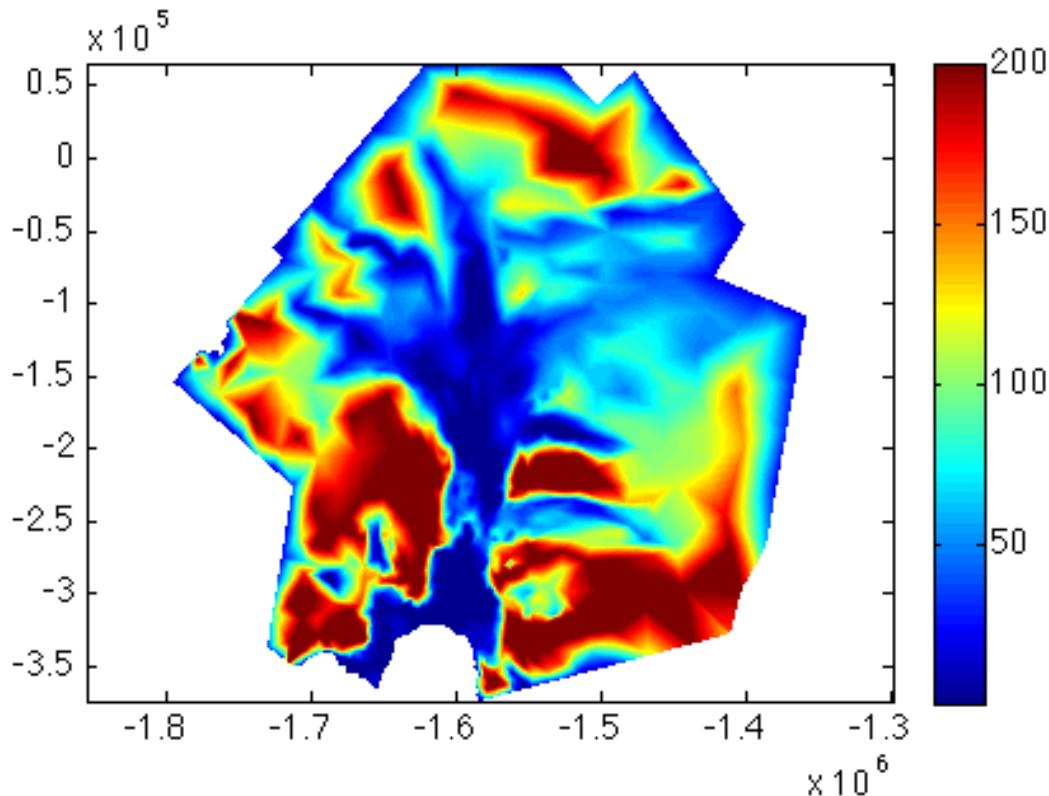
- τ_b : Basal drag
- N : Effective pressure
- v_b : Basal velocity (equal surface in SSA approximation)
- r : Exponent (equals q/p of the parameter file)
- s : Exponent (equals $1/p$ of the parameter file)

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`.

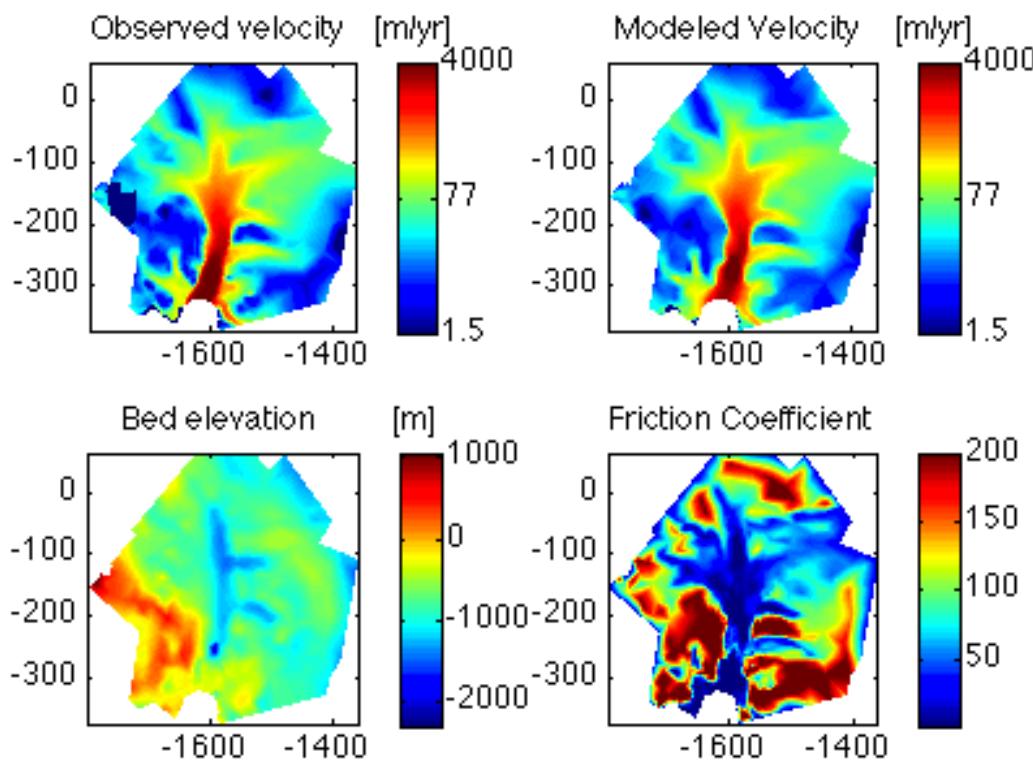
Run step 4 and look at the results, they should be similar to the figure below:



4.2.9.8 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:



4.2.9.9 Higher Order (HO) Ice Flow Model

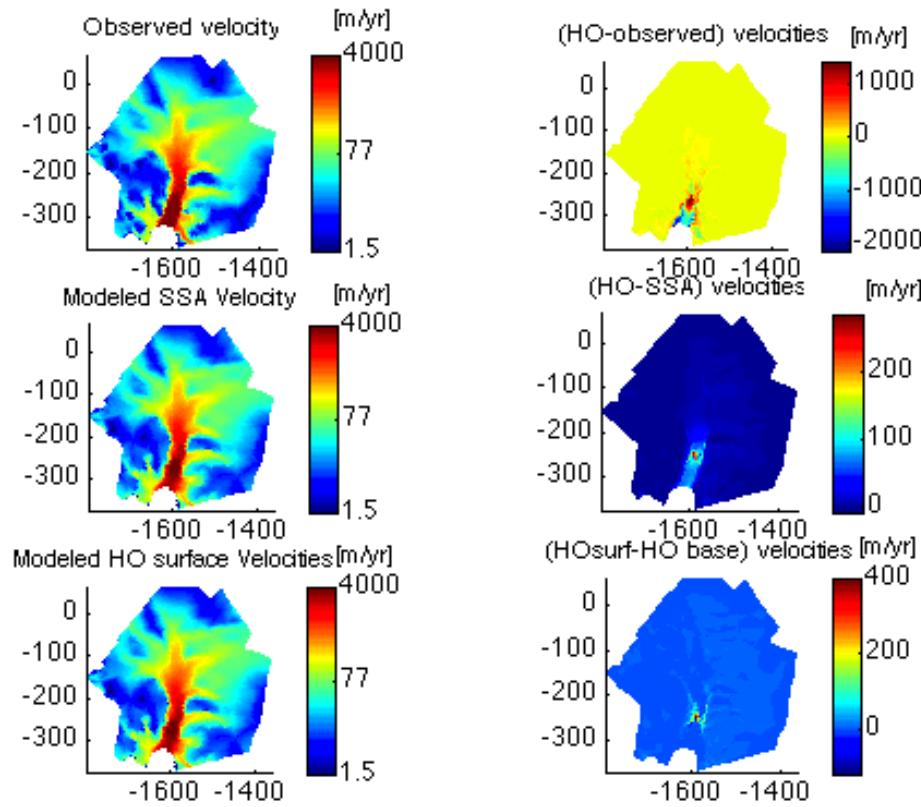
The last step of this tutorial is to run a forward model of Pine Island Glacier with the Higher-Order stress balance approximation.

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:



4.2.9.10 Solution for step 6

```

if any(steps == 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, 'Stressbalance');

    save ./Models/PIG_ModelHO md;
end

```

4.2.10 Pine Island Glacier Sensitivity Study

4.2.10.1 Goals

This example is adapted from the results presented in ?. 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 `<ISSM_DIR>/examples/PigSensitivity/`. This tutorial relies on experience gained from completing the [Pine Island Glacier](#) and [Greenland Ice Sheet](#) modeling tutorials, so make sure to complete them first.

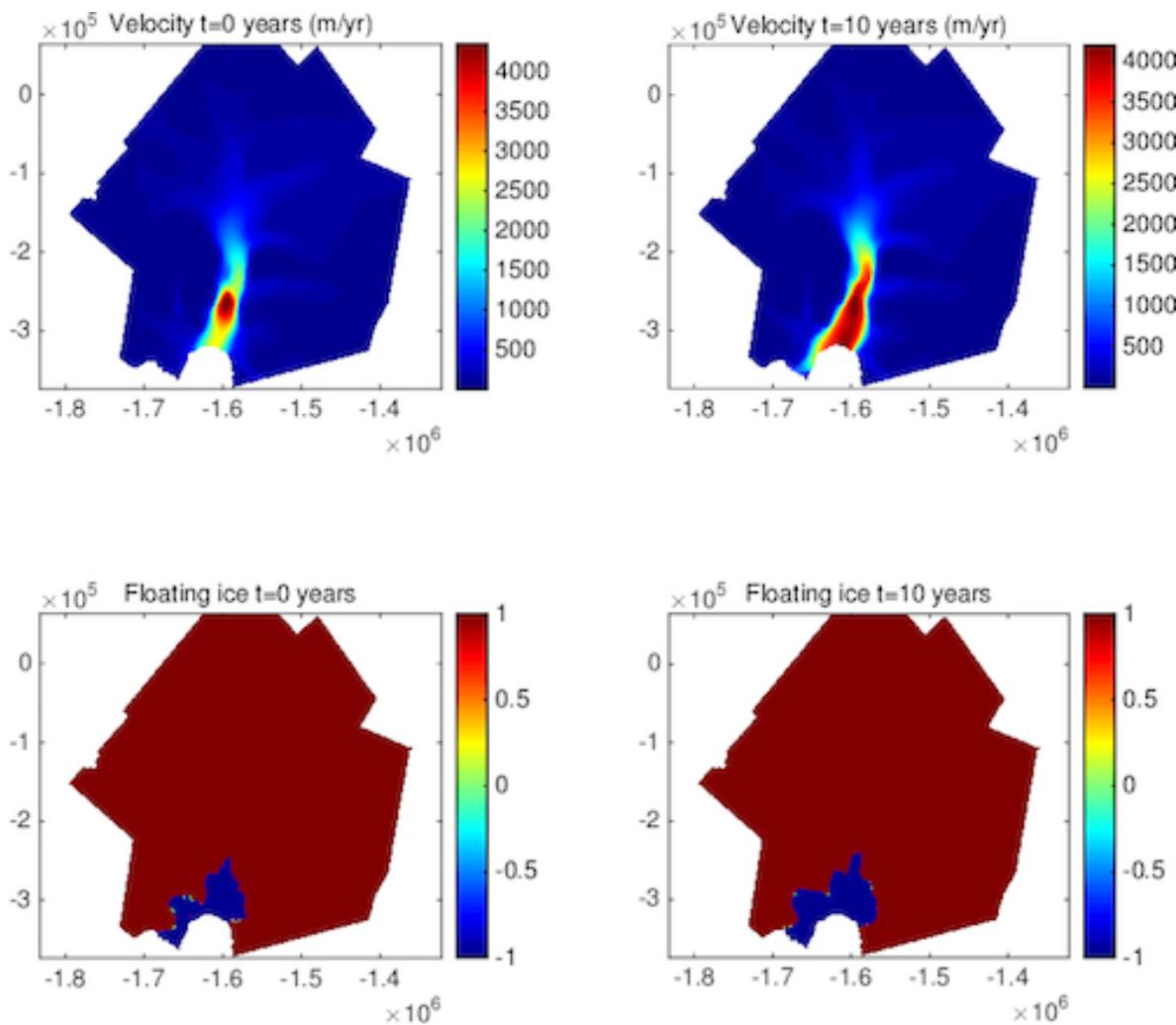
4.2.10.2 Evolution over 10 years

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 `steps`, at the top of the file, is set to `steps = [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.issstressbalance`
 - `md.transient.isthermal`
 - `md.transient.isgroundingline`
 - `md.transient.ismovingfront`
- Request additional outputs
- Solve transient solution

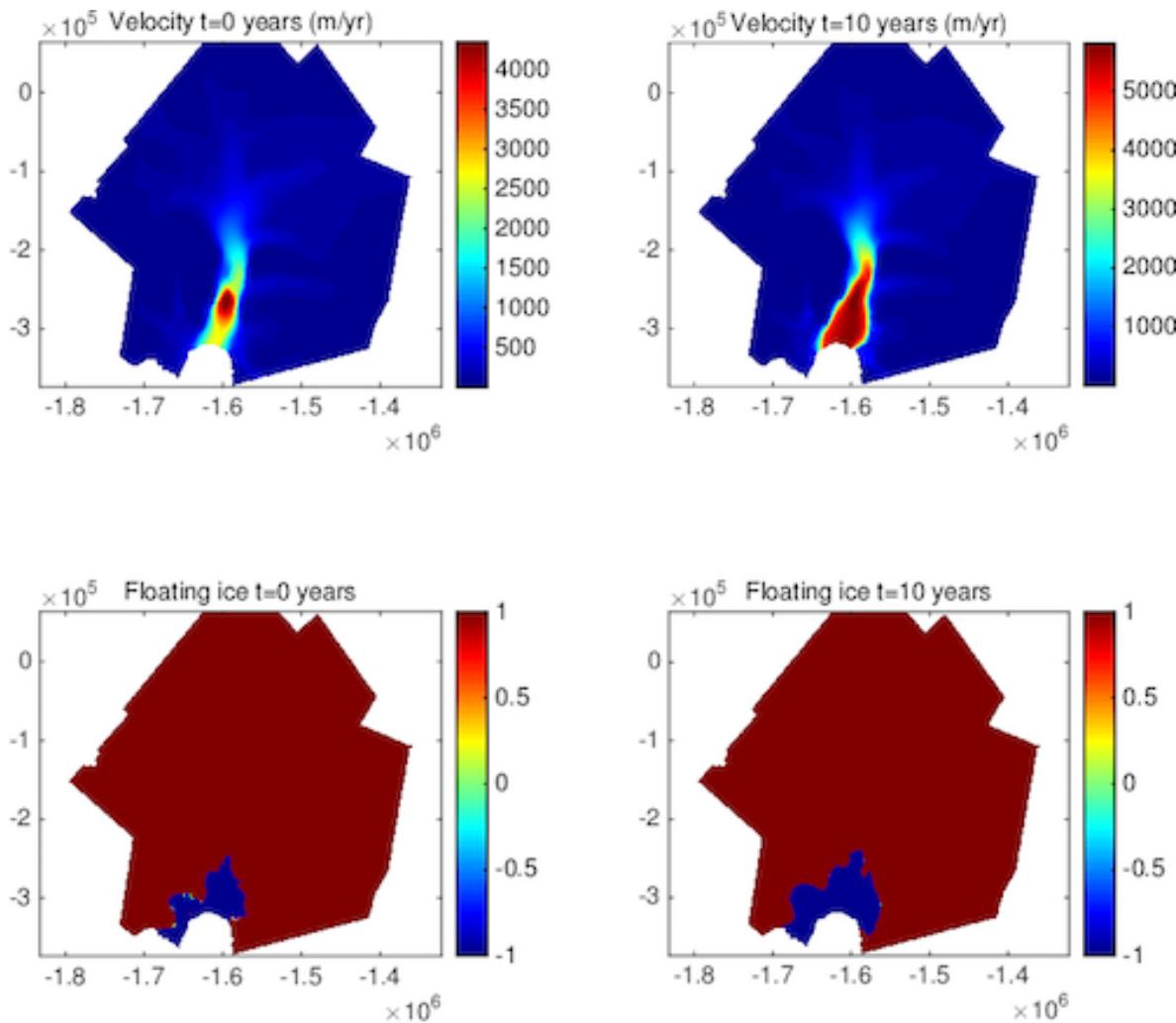
Execute `runme` to perform step 1. The following figure shows the evolution of the ice velocity and grounding line positions at the beginning and at the end of the simulation:



4.2.10.3 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:



4.2.10.4 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.

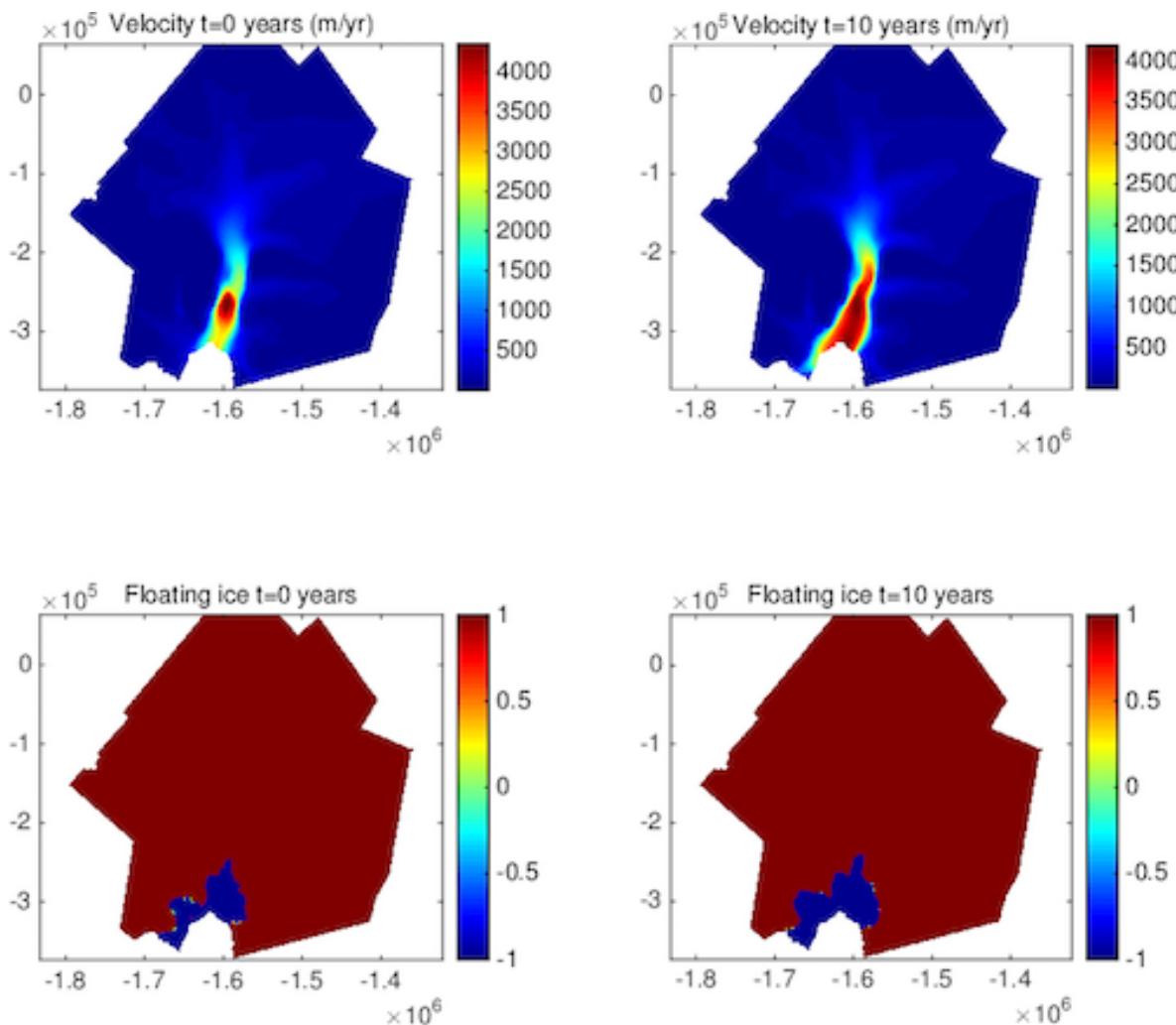
Then extract the domain from the initial model, excluding the `RetreatFront.exp` area using the `extrude` routine:

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

As this operation changes the model domain, some parameters and boundary conditions have to 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:



4.2.10.5 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'};

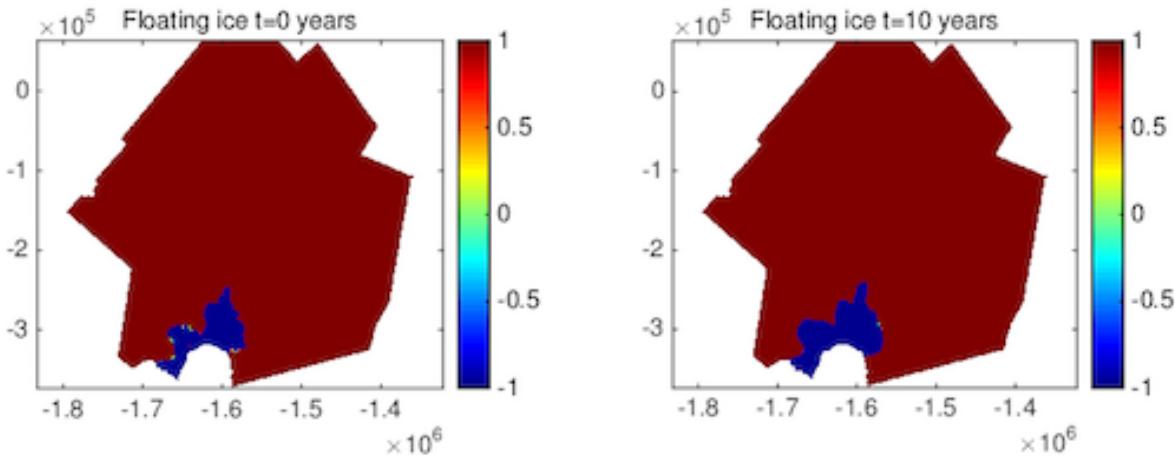
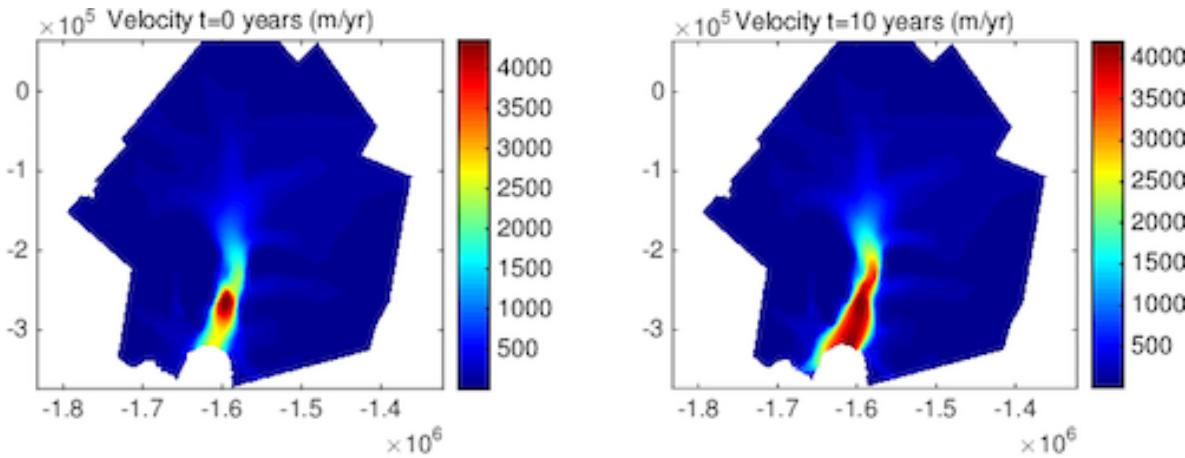
    %Solve
    md = solve(md, 'Transient');

    %Plot
    plotmodel(md, 'data', md.results.TransientSolution(1).Vel, ...
        'title#1', 'Velocity t=0 years (m/yr)', ...
        'data', md.results.TransientSolution(end).Vel, ...
        'title#2', 'Velocity t=10 years (m/yr)', ...
        'data', md.results.TransientSolution(1).MaskOceanLevelset, ...
        'title#3', 'Floating ice t=0 years', ...
        'data', md.results.TransientSolution(end).MaskOceanLevelset, ...
        'title#4', 'Floating ice t=10 years', ...
        'caxis#1', ([0 4500]), 'caxis#2', ([0 4500]), ...
        'caxis#3', ([-1, 1]), 'caxis#4', ([-1, 1]));

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

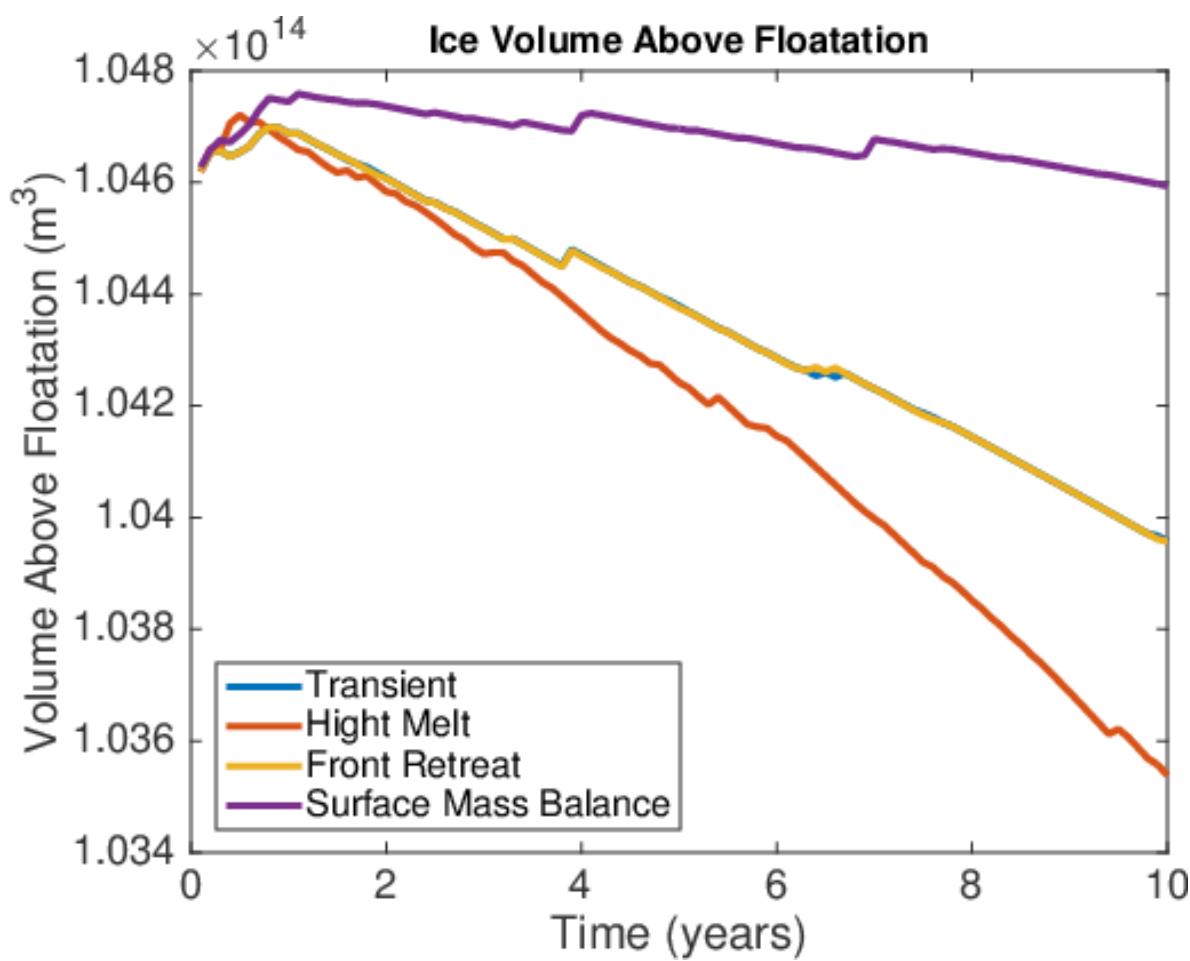
```

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



4.2.10.6 Evolution of the ice volume above flotation

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 flotation for the four scenarios performed previously:



4.2.11 Uncertainty Quantification (UQ)

4.2.11.1 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

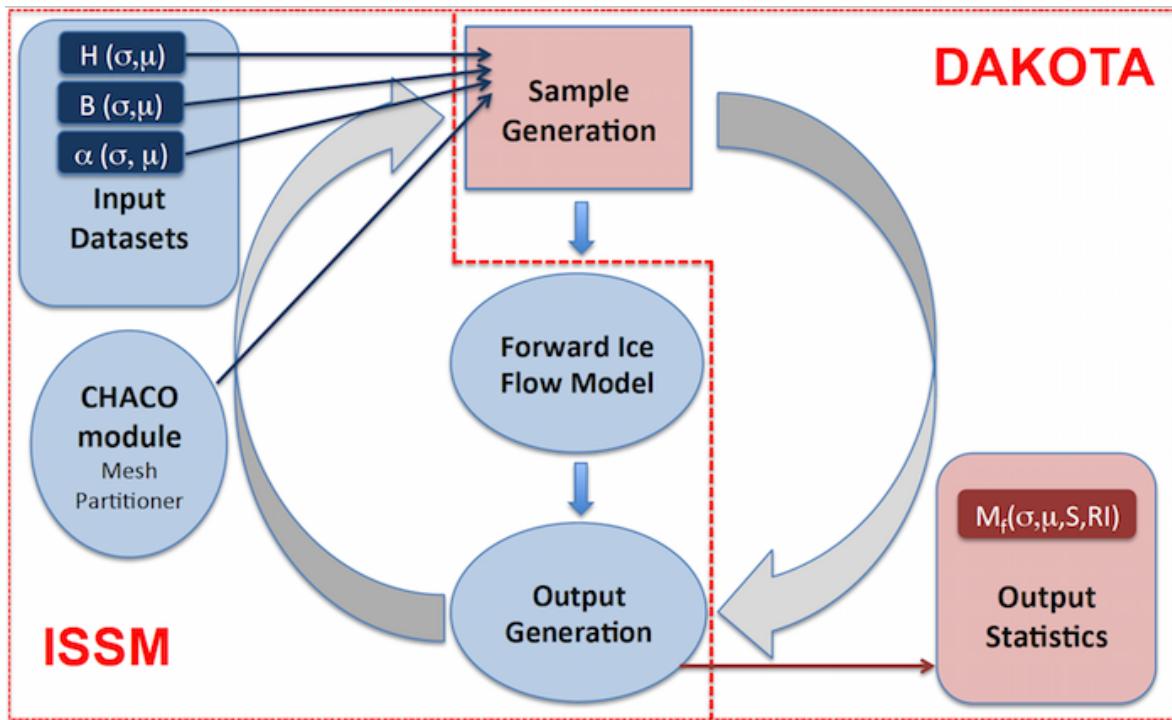
Go to `<ISSM_DIR>/examples/UncertaintyQuantification/` to do this tutorial.

4.2.11.2 Introduction

This experiment will use the model of Pine Island Glacier that was saved in the previous [Pine Island Glacier modeling 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.

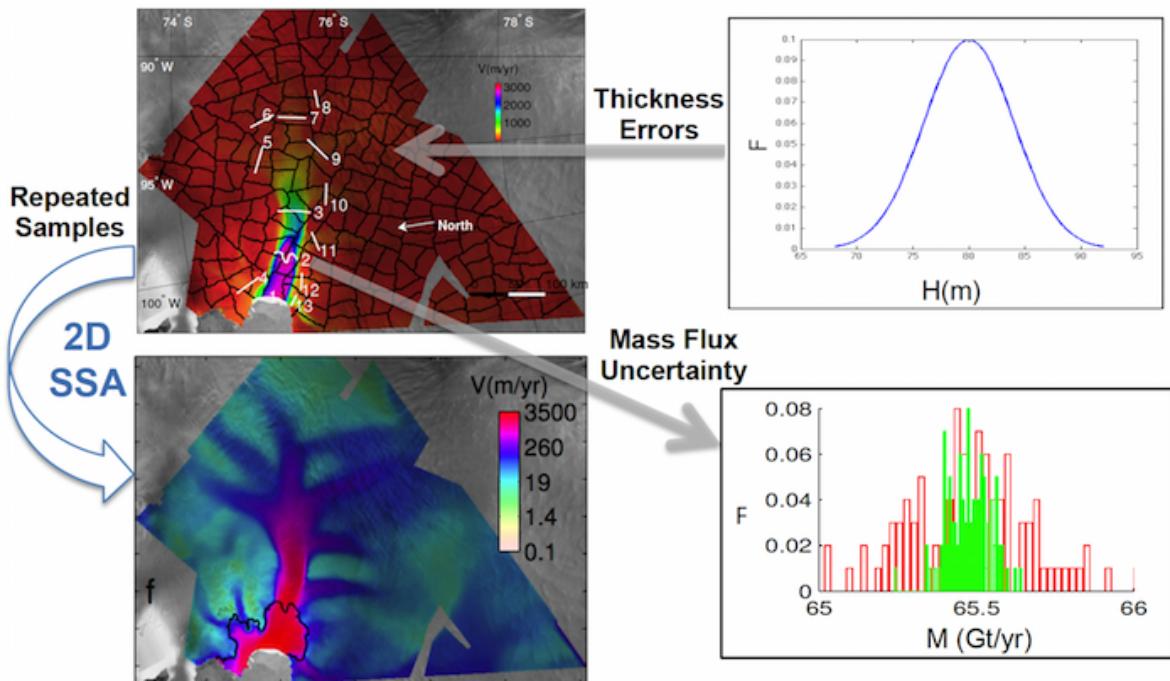
Our Uncertainty Quantification (UQ) methods are based on the Design Analysis Kit for Optimization and Terascale Applications (Dakota) software [?], which is embedded in ISSM. The following diagram illustrates the relationship between ISSM and Dakota. The ISSM mesh must be partitioned (i.e. vertices can 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 responsible 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.).



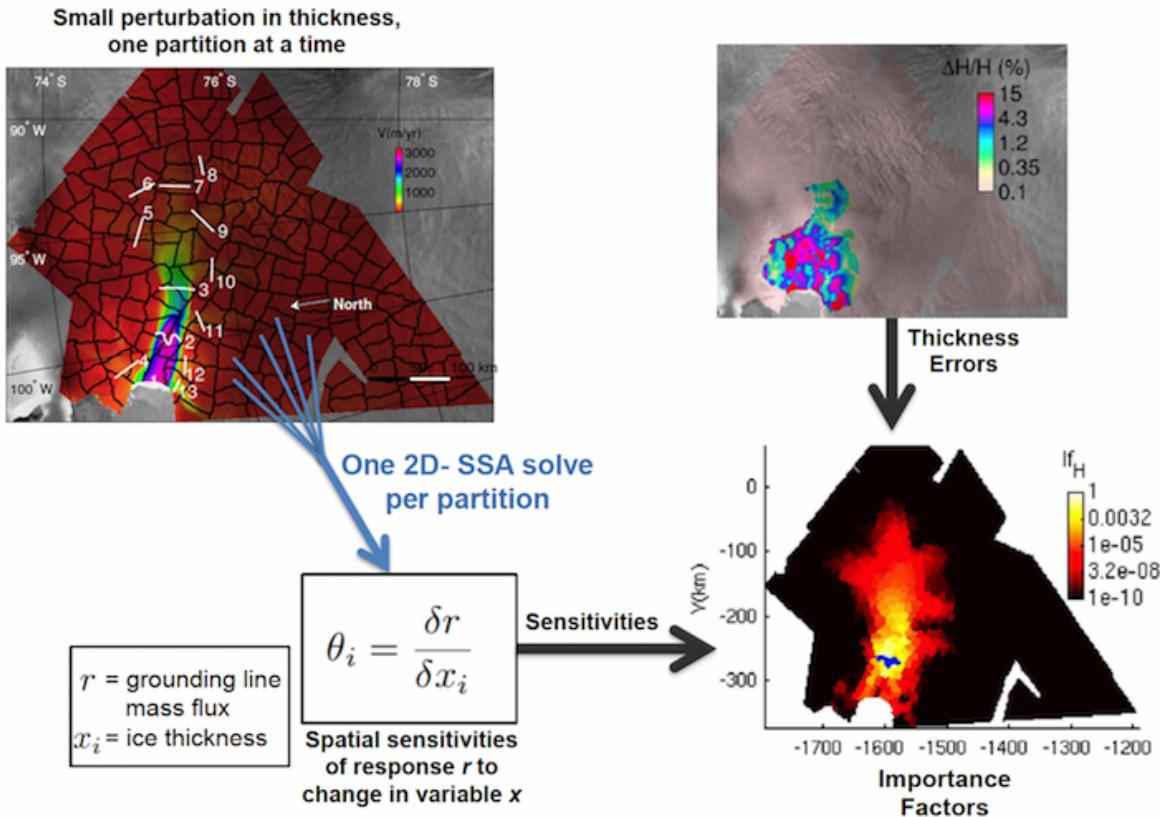
Tutorial steps to be taken:

- 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

Sampling 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 diagnostic, in importance. One such example of Dakota output is the ‘importance factor’, or sensitivities scaled by error margins [??], illustrated below as UQ sensitivity analysis output for mass flux gate 2.



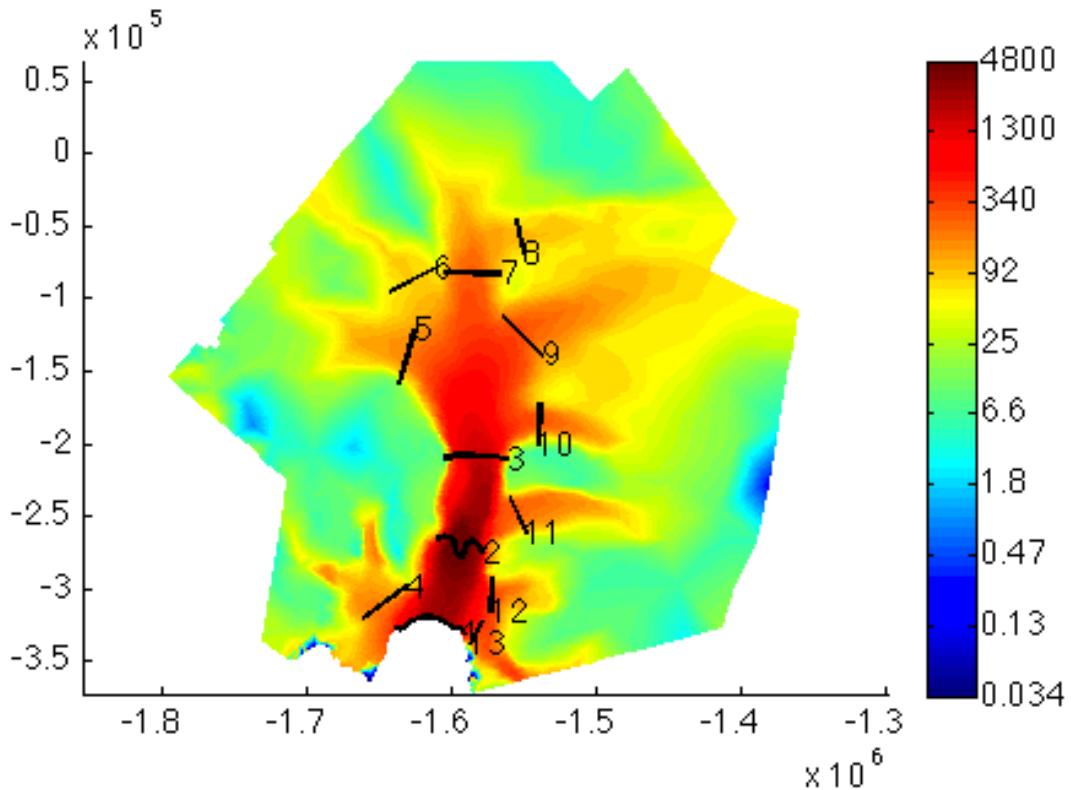
For manuscript examples of these studies, see ????.

4.2.11.3 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 overlaid over the PIG surface velocities.



4.2.11.4 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.

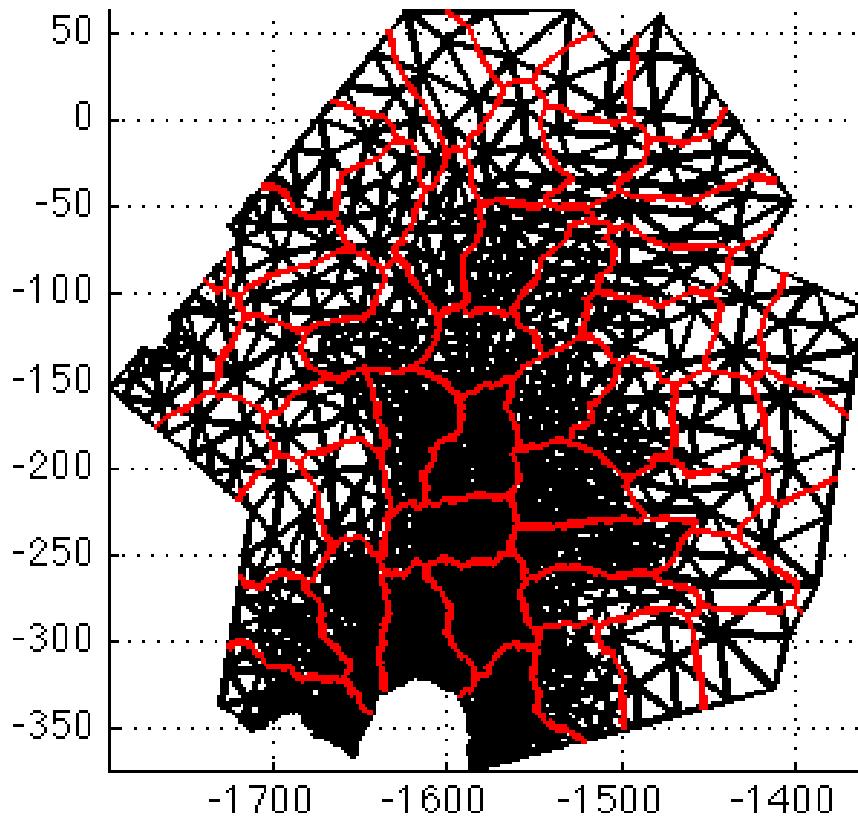
4.2.11.5 Sampling Analysis

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 69-72 in the `runme.m` file
- Run step 3

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

- See lines 155-162
- Run step 4



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 74-82

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 84-97

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 99-115

Finally, we need to designate a sampling strategy. Options include '`nond_samp`' for sampling or '`nond_1`' 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:

- See lines 117-124

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

- See lines 126-131

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

- See line 133

Because the ISSM-Dakota framework now runs in parallel, our implementation requires that Dakota runs with a master/slave configuration. This means that at least 2 CPUs 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 142-145

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

- See lines 147-149

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

4.2.11.6 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% perturbation range:
 - See lines 178-190
- To specify new sensitivity method, tell Dakota to use local reliability or '`nond_1`' :
 - See line 226

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:

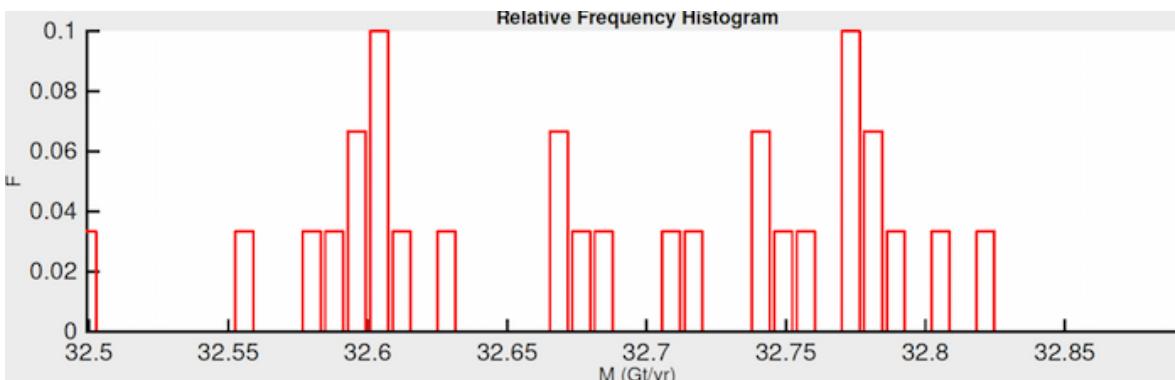
- See lines 239-252

Run step 5 to launch the sensitivity runs.

4.2.11.7 Plot Results

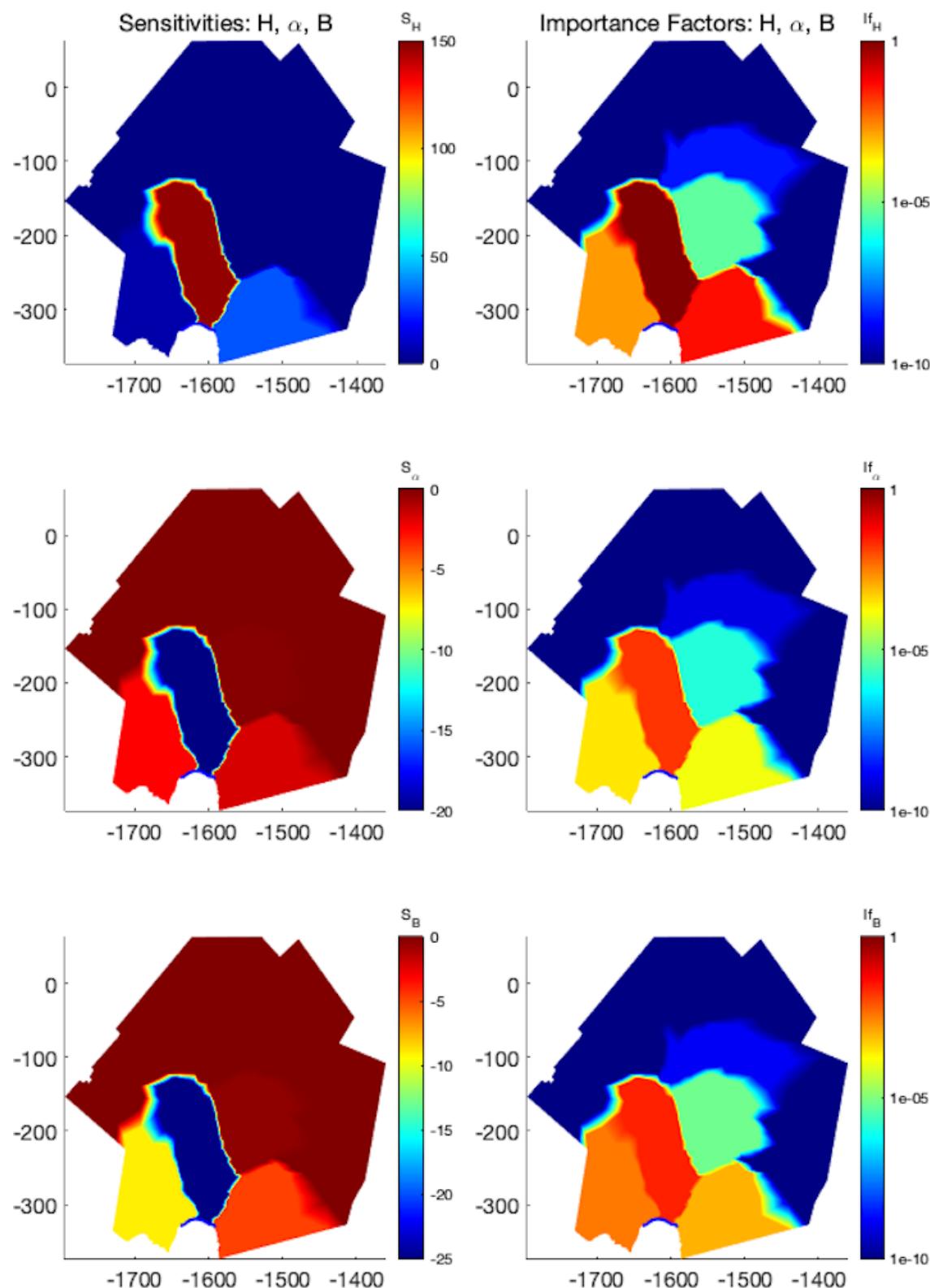
Plot Sampling Results: In order to plot the results, we extract the results for one of the mass flux gates, and display a histogram of the sampling results for that particular gate. ISSM has a plotting function for this, 'plot_hist_norm'. Note that ISSM mass flux results are in mass flux in m^3 water 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 260-273



Plot Sensitivity Results:

- To retrieve sensitivities for each model input:
 - See lines 288-290
- To plot sensitivities:
 - See lines 292-300
- To retrieve importance factors for each model input:
 - See lines 303-305
- To plot the importance factors:
 - See lines 307-314
- 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).



4.2.11.8 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!

4.2.12 Pine Island Glacier Stochastic Forcing (StISSM)

4.2.12.1 Goals

- Introduction to the use of the stochastic capabilities implemented in ISSM (StISSM)
- Model Pine Island Glacier as in previous tutorials, but with stochastic forcings

4.2.12.2 Introduction

The main goals of this tutorial are 1) to become familiar with the use of StISSM, 2) to learn how to parameterize stochastic variables of the model, and 3) to launch transient stochastic simulations. The first steps follow what was done in previous tutorials: setting up the model domain and general configuration for the Pine Island Glacier. The organization of the tutorial is as follows:

- Step 1: Generate a model mesh
- Step 2: Set up the ice and ocean masks
- Step 3: Parameterization of the model
- Step 4: Set up the stochastic SMB parameterization
- Step 5: Transient run
- Step 6: Set up the stochastic calving parameterization
- Step 7: Second transient run starting from the results of the first one

Files needed for this tutorial can be found in `trunk/examples/StISSM/`. The `runme.m` file contains the structure of the overall simulation, while the `.par` file includes most parameters needed for the model setup. The `.exp` files are domain files that define geometric boundaries of the simulation. Observed datasets needed for the parameterization also need to be [downloaded](#).

4.2.12.3 Mesh

This step follows what is done in the Pine Island Glacier tutorial. We simply set up the model mesh from the `.exp` files. Set `step = 1` in the `runme.m` file to execute it.

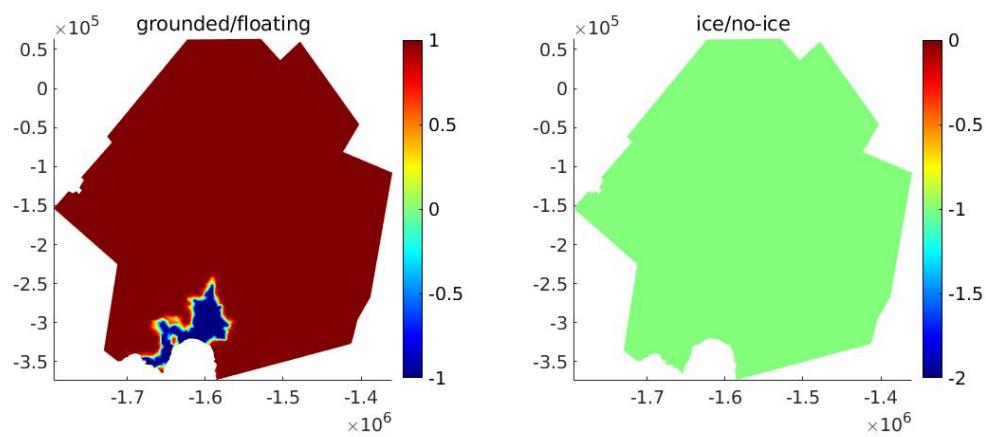
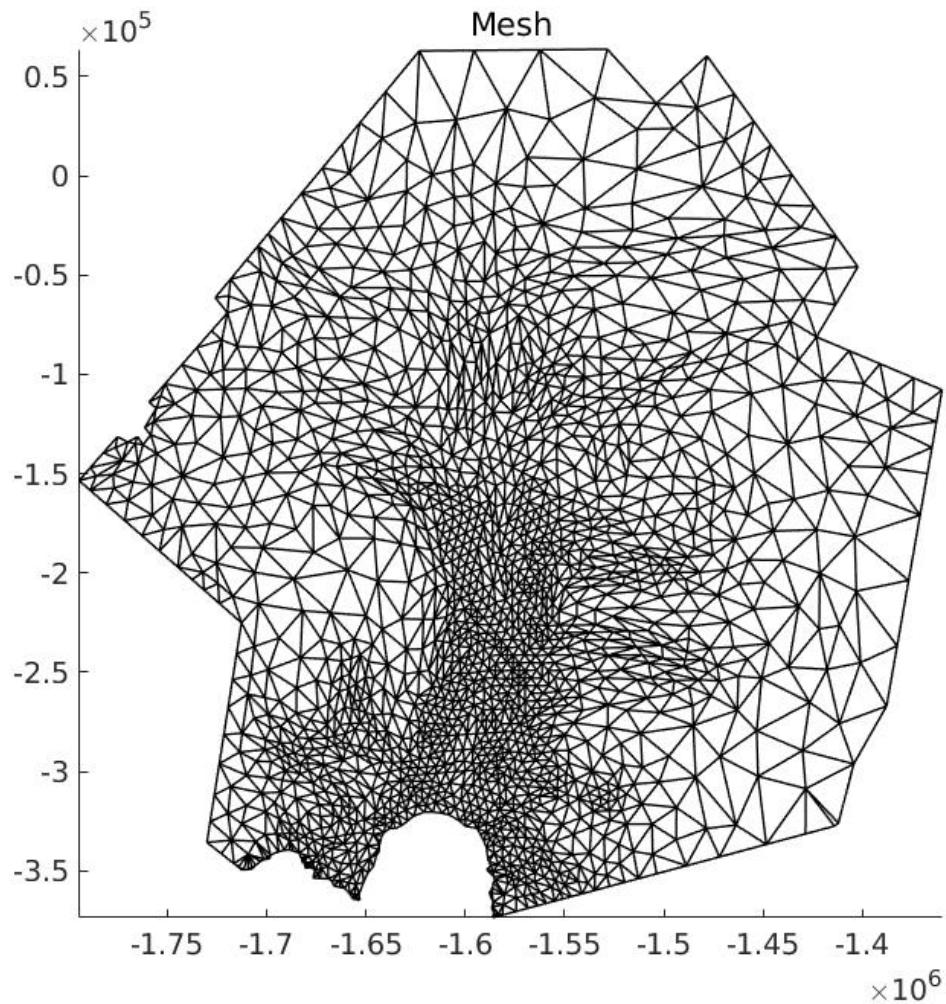
4.2.12.4 Mask

This step follows what is done in the Pine Island Glacier tutorial. We define the masks where ice is present/absent and where ice is grounded/floating. Set `step = 2` in the `runme.m` file to execute it.

4.2.12.5 Parameterization

This step follows what is done in the Pine Island Glacier tutorial. We use the `PigStISSM.par` file to parameterize the following fields:

- Geometry
- Initialization parameters



- Material parameters
- Forcings
- Friction coefficient
- Ice rheology
- Boundary conditions

Set `step = 3` in the `runme.m` file to execute it.

4.2.12.6 Parameterization

From here, we start to focus on the specifics of StISSM. We set up an Autoregressive Moving-Average (ARMA) model for SMB. In other words, the evolution of SMB follows the following equation:

$$SMB_t = \mu_t + \sum_{i=1}^p \varphi_i (SMB_{t-i} - \mu_{t-i}) + \sum_{j=1}^q \theta_j \epsilon_{y,t-j} + \epsilon_t \quad (4.6)$$

where μ_t is a deterministic function of time, φ are the autoregressive (AR) coefficients, and θ are the moving-average coefficients (MA). The values of p and q are the orders of the AR and MA part of the ARMA model, respectively. The term ϵ_t is a Gaussian noise term generated at time step t .

We define two different subdomains, with separate ARMA processes. The subdomains are separated at 1/3rd of the x-axis. For the deterministic function μ in Eq. (1), we use a piecewise linear function with a single breakpoint:

$$\begin{cases} \mu_t = c_0 + a_0(t - t_0) & \text{if } t \leq t_{brk} \\ \mu_t = c_1 + a_1(t - t_{brk}) & \text{if } t > t_{brk} \end{cases} \quad (4.7)$$

where t_0 is the initial time of the ARMA model, t_{brk} is the breakpoint (a date in time), the c terms are constant values, and the a terms are trends in time. All the coefficients and parameters of Eqs (1) and (2) are prescribed in the `runme.m` file.

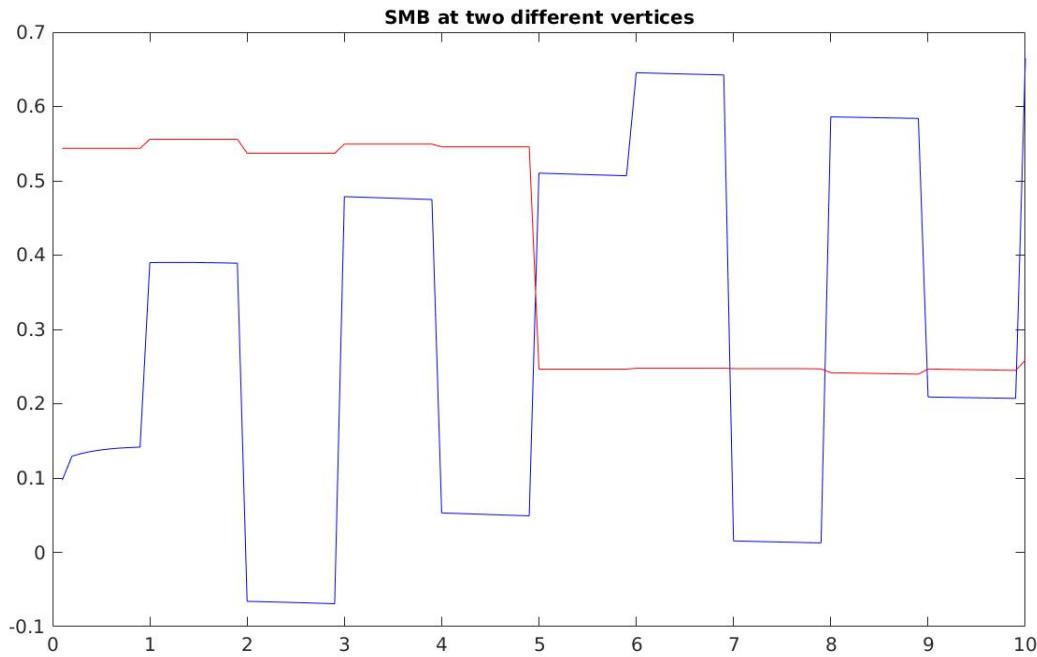
Next, we also define some SMB lapse rates. Lapse rates are elevation gradient of SMB. The lapse rate values must be associated with an elevation range, such that lapse rate 1 applies below elevation 1, lapse rate 2 applies between elevation 1 and elevation 2, etc.

After that, we set up the covariance matrix that will define the stochastic perturbations. We use different amplitudes of variability in the two subdomains, and a moderate correlation (0.5) between the subdomains. Notice that the covariance matrix is simply computed as:

$$\Sigma = KCK \quad (4.8)$$

where K is the diagonal matrix with the individual standard deviations on the diagonal, and C is the correlation matrix.

The final step of the stochasticity configuration is to set up the parameterization of `md.stochasticforcing`. This only entails activating stochasticity, specifying which model field is stochastic (SMBarma here), the time step of stochasticity (the frequency at which random perturbations are generated), and assigning the covariance matrix. Set `step = 4` in the `runme.m` file to execute this step.

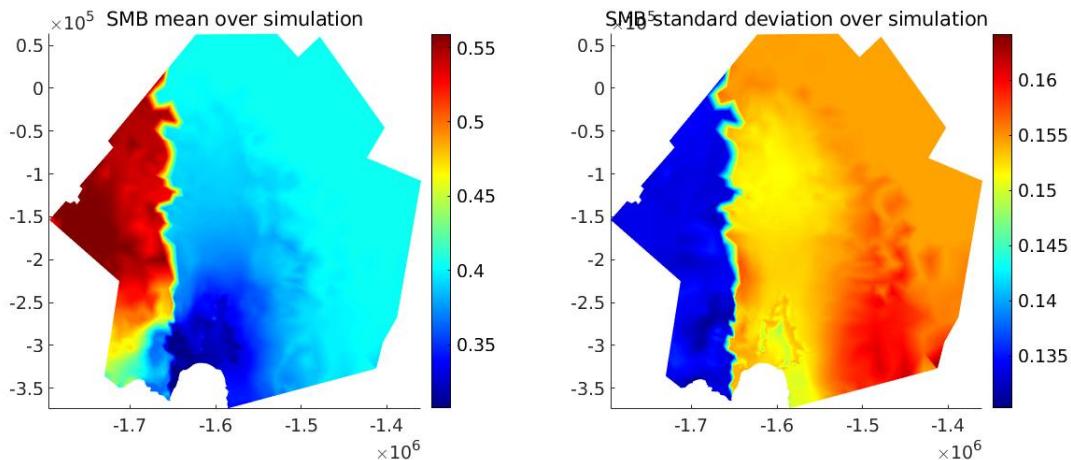


4.2.12.7 Transient run 1

Now that the entire model is configured, we run a transient simulation and plot some results. The plots generated are an example of the SMB results that you could reach. Note that all runs will have different SMB fields, due to stochasticity. Set `step = 5` in the `runme.m` file to execute it.

4.2.12.8 Stochastic calving

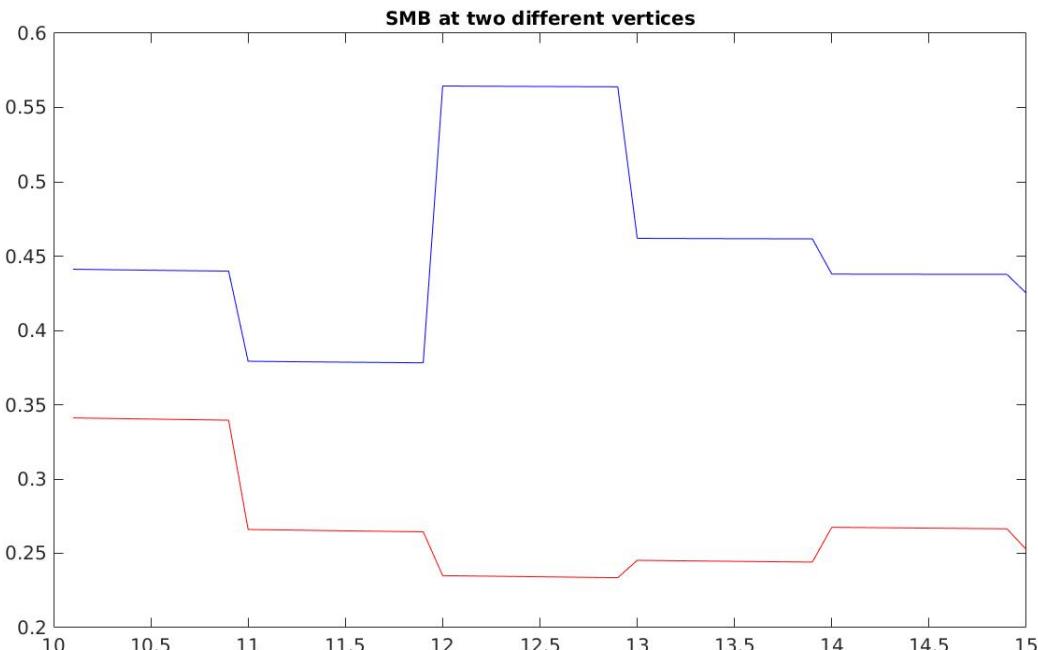
In the last step of this tutorial, we also want to activate stochastic calving. First, we need to specify that calving at the ice front is activated, and specify the background calving values. Notice also that imposing calving means that we need to allow for the ice front to migrate by setting `md.transient.ismovingfront` to 1. Here, we assign the same subdimensions for calving as for

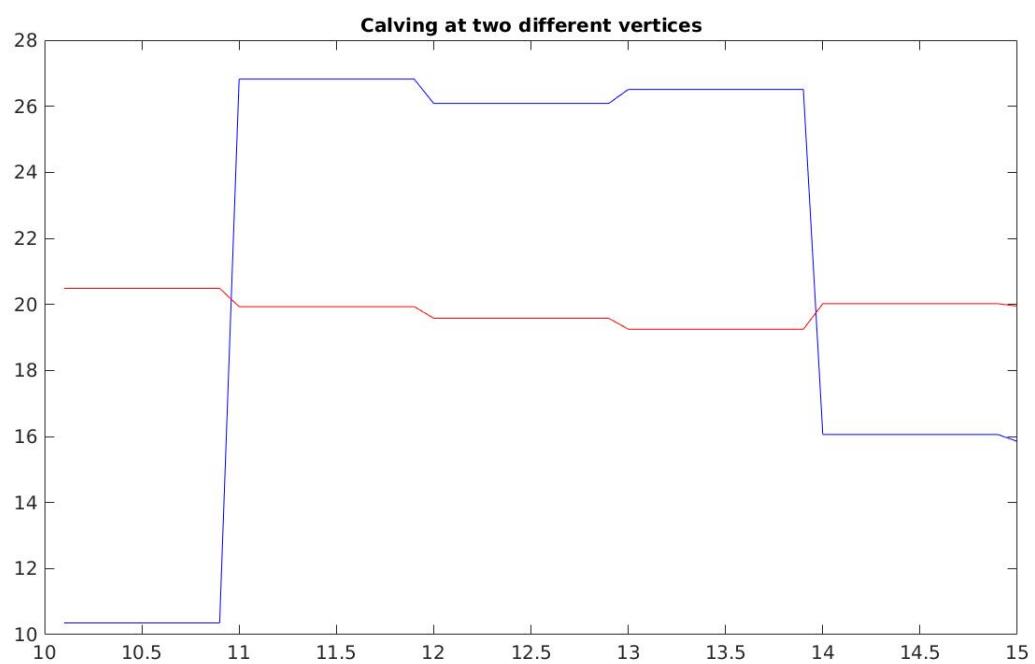


SMB. Next, we need to set the covariance matrix for calving. This involves setting the individual standard deviations for the two subdomains, the correlation matrix (here, we set no correlation), and applying Eq. (3). Finally, we modify the `md.stochasticforcing` class. We specify that two fields are stochastic `[{'SMBarma'}, {'DefaultCalving'}]`. We stack the SMB and calving covariance matrices together, here assuming no correlation between these different fields. As a final note, calving is not an ARMA model, thus its subdimensions must be passed as the default dimensions of the `md.stochasticforcing` class. Set `step = 6` in the `runme.m` file to execute this step.

4.2.12.9 Transient run 2

We want to launch a second transient run. First, we load the final geometry, masks, and velocities of the previous transient run, and set these as initial conditions for the second transient run. The model is now fully configured for the second transient run. We launch the transient simulation, and plot some results. Again, the plots generated are an example of possible results, which will vary due to the stochastic nature of the model run. Set `step = 7` in the `runme.m` file to execute this step.





4.2.13 Modeling Jakobshavn Isbræ

4.2.13.1 Goals

- Construct a 2-dimensional model of Jakobshavn Isbræ, 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 `<ISSM_DIR>/examples/Jakobshavn/` to do this tutorial.

4.2.13.2 Introduction

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

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](#) section).

4.2.13.3 runme file

The `runme.m` file in `<ISSM_DIR>/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`.

4.2.13.4 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
```

4.2.13.5 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 smb.precipitation specified: values set as zero
    no basalforcings.melting_rate specified: values set as zero
    no balancethickness.thickening_rate specified: values set as
        zero
```

4.2.13.6 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 modeled surface velocities match the observed ones. The mismatch between observation and modeled 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.

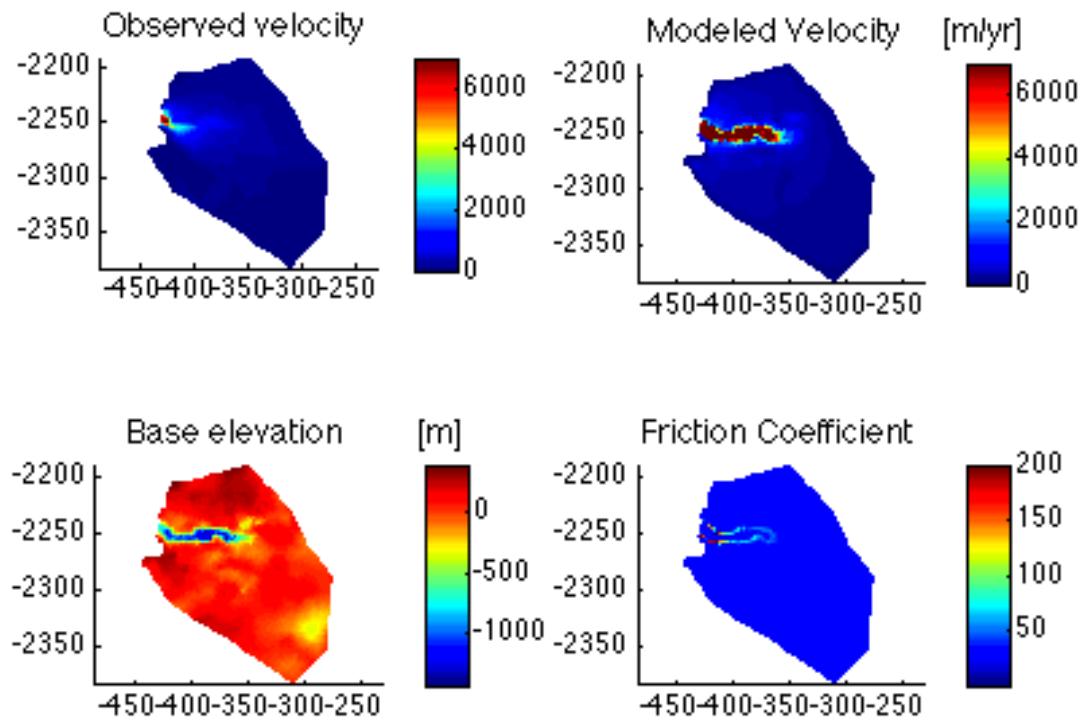
Open `runme.m`, make sure that the third step is activated (`steps = [3];`), then run `runme.m`:

```
>> 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
    ....
```

4.2.13.7 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:



4.2.14 Subglacial Channel Formation From a Single Moulin (SHAKTI)

4.2.14.1 Goals

- Learn to set up a subglacial hydrology simulation using the SHAKTI model (Subglacial Hydrology and Kinetic, Transient Interactions, ?),
- Run a test with steady input into a single moulin to see an efficient drainage pathway develop from the moulin to the outflow, and obtain the corresponding effective pressure, hydraulic head, and basal water flux distributions.

Go to `<ISSM_DIR>/examples/shakti/` to do this tutorial.

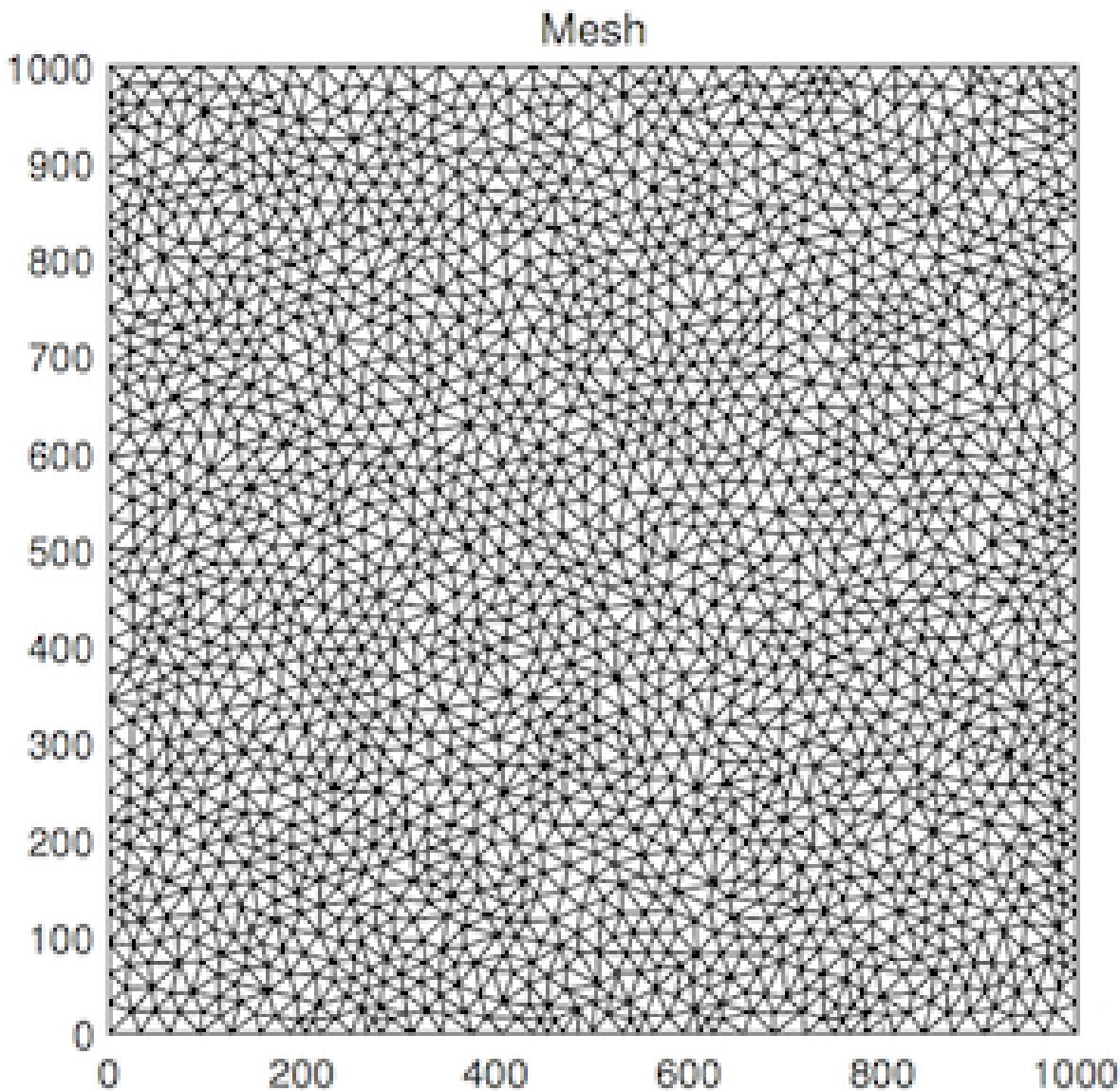
4.2.14.2 Introduction

The `runme.m` file and `moulin.par` go through the steps and basic structure to set up and run a subglacial hydrology model with steady input into a single moulin at the center of a 1 km square, tilted slab of ice. These files can be altered to create simulations on different domains and geometries, with different meltwater inputs (distributed or into moulin, steady or time-varying). The `runme.m` script is set up as three distinct steps, saving the model at each stage:

1. Mesh generation
2. Parameterization
3. Hydrology solution

4.2.14.3 Mesh Generation

Run step 1 in `runme.m` to generate an unstructured mesh on a 1 km square with typical element edge length of 20 m. This mesh shown here has 4,032 elements and 2,096 vertices. To plot your mesh, use `plotmodel(md, 'data', 'mesh')` :



4.2.14.4 Parameterization

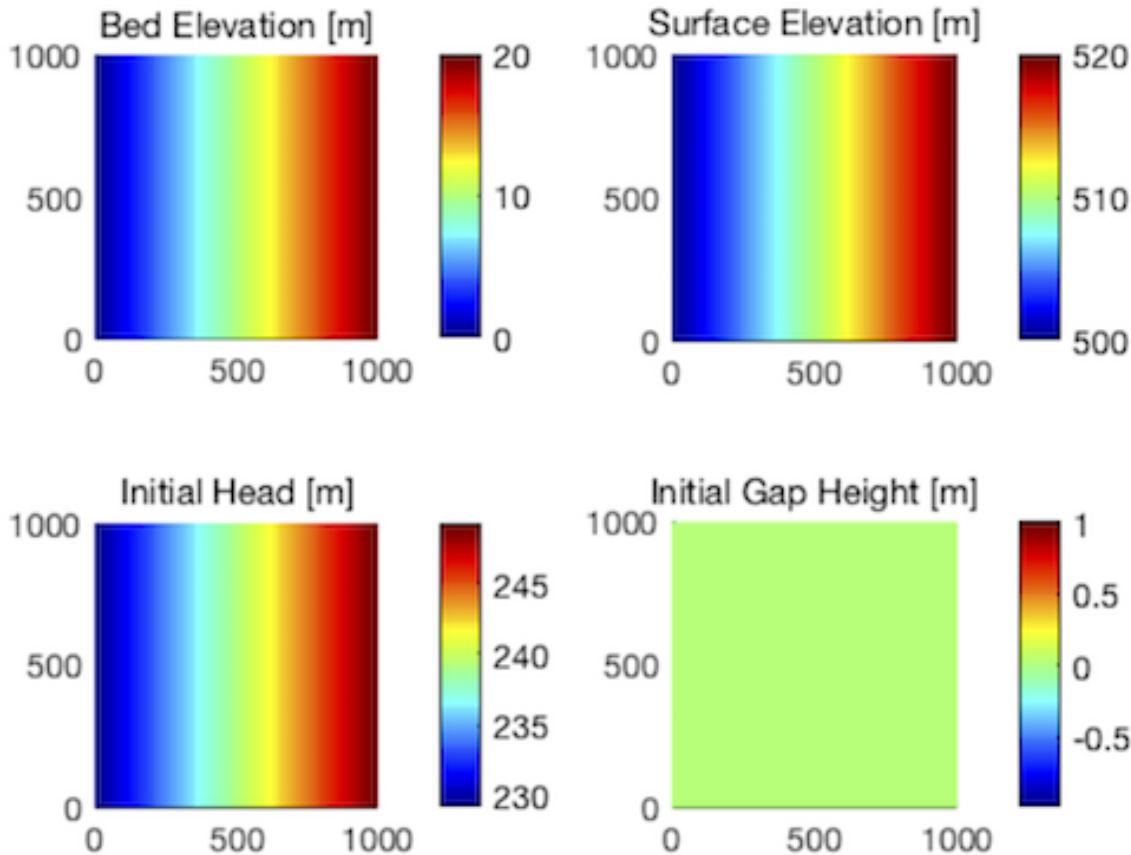
Run step 2 in `runme.m` to define the model parameters. First we call on standard parameters defined in the `moulin.par` file (bed and ice geometry, sliding velocity, material properties, etc.). Then we define hydrology-specific parameters for the SHAKTI model (initial hydraulic head, Reynolds number, subglacial gap height, boundary conditions, etc.).

The model domain is set up as a 500 m thick slab of ice, with bed and surface slope of 0.02. We begin by assuming the hydraulic head is such that the water pressure is equal to 50% of the ice overburden pressure, $Re=1,000$ everywhere, and the initial gap height is 0.01 m. The outflow boundary ($x=0$ m) is set to atmospheric pressure ($h = z_b$) with a "Type 1" (Dirichlet) condition. We set the distributed input from the englacial system to the subglacial system as zero (for this example, we will define the moulin input in the next step).

To look at the bed topography, ice surface, initial head, and initial gap height, you can plot them in MATLAB:

```
plotmodel(md, ...)
```

```
'data', md.geometry.base, 'title', 'Bed Elevation [m]', ...
'data', md.geometry.surface, 'title', 'Surface Elevation [m]', ...
'data', md.hydrology.head, 'title', 'Initial Head [m]', ...
'data', md.hydrology.gap_height, 'title', 'Initial Gap Height [m]')
```



4.2.14.5 Hydrology solution

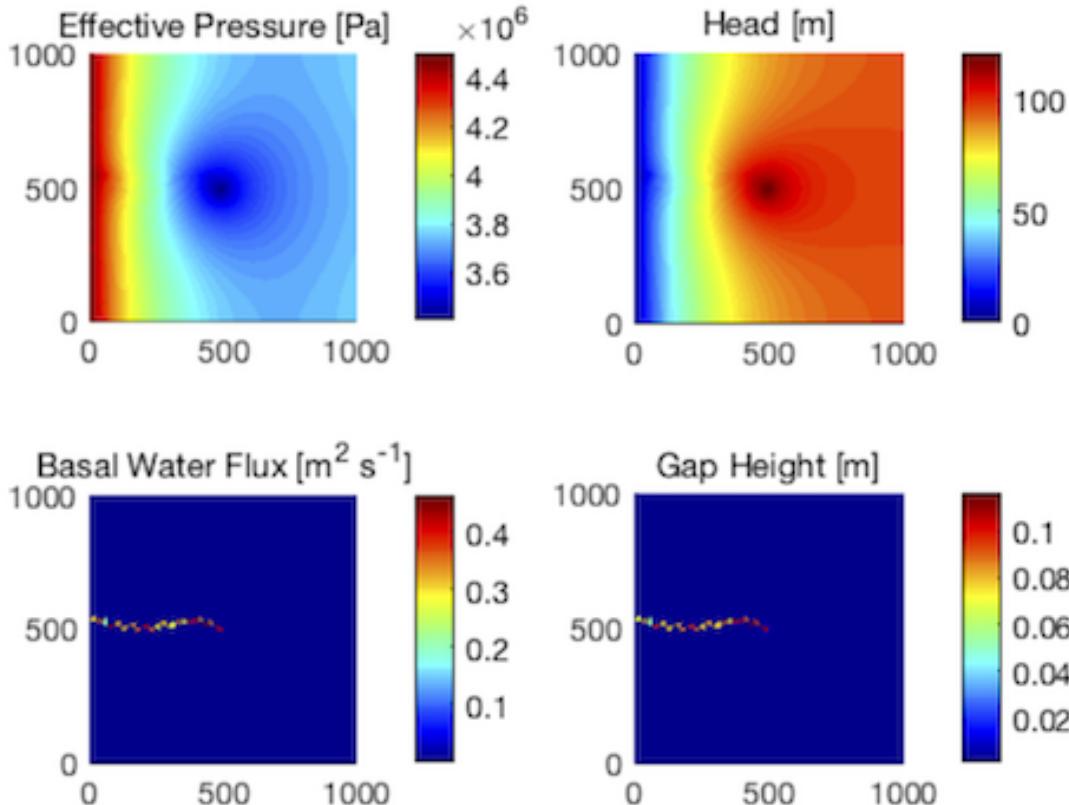
In step 3, we specify which machine we want to run the model on, including number of processors to be used, define the model time step, final time, and prescribe the moulin inputs. In this example, we put a steady moulin input of $4 \text{ m}^3 \text{ s}^{-1}$ at the center of the domain ($x=500 \text{ m}$, $y=500 \text{ m}$). We also impose a no-flux "Type 2" (Neumann) boundary condition at all boundaries (except the outflow, where we have our Dirichlet condition defined already in step 2).

Now that the set up is complete, we can run the model:

```
md = solve(md, 'Transient');
```

The final steady configurations for effective pressure, hydraulic head, basal water flux, and gap height can be visualized by plotting:

```
plotmodel(md, 'data',
    md.results.TransientSolution(end).EffectivePressure, 'title',
    'Effective Pressure [Pa]', ...
    'data', md.results.TransientSolution(end).HydrologyHead, 'title',
    'Head [m]', ...
    'data', md.results.TransientSolution(end).HydrologyBasalFlux,
    'title', 'Basal Water Flux [m^2 s^{-1}]', ...
    'data', md.results.TransientSolution(end).HydrologyGapHeight,
    'title', 'Gap Height [m]')
```



You can see that a distinct pathway has formed from the moulin at the center to the outflow at the left. Hydraulic head (related to water pressure) is highest directly around the moulin, and the head is lower in the channel than in the areas above and below it in the y-direction.

To watch the evolution through time in an animation, use the command:

```
plotmodel(md, 'data', 'transient_movie')
```

You will be prompted to select which parameter to animate, and can watch an efficient subglacial channel emerge from the moulin to the outflow!

4.2.15 Modeling Helheim Glacier

4.2.15.1 Goals

- Create an ice-flow model of Helheim Glacier (southeast Greenland)
- Run an inversion model to infer basal friction

4.2.15.2 Introduction

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

- Define the model region
- Create a mesh
- Parameterize the model
- Invert for friction coefficient
- Plot results

Files needed for this tutorial can be found in `<ISSM_DIR>/examples/Helheim/`. 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` file contains coordinates that define the model domain boundaries.

Observed datasets needed for the parameterization need to be downloaded.

4.2.15.3 Mesh

The first step is to create the model domain outline and mesh.

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 `steps`, at the top of the file, is set to `steps = [1]`. In the code, you will see that in Step 1 the following actions are implemented:

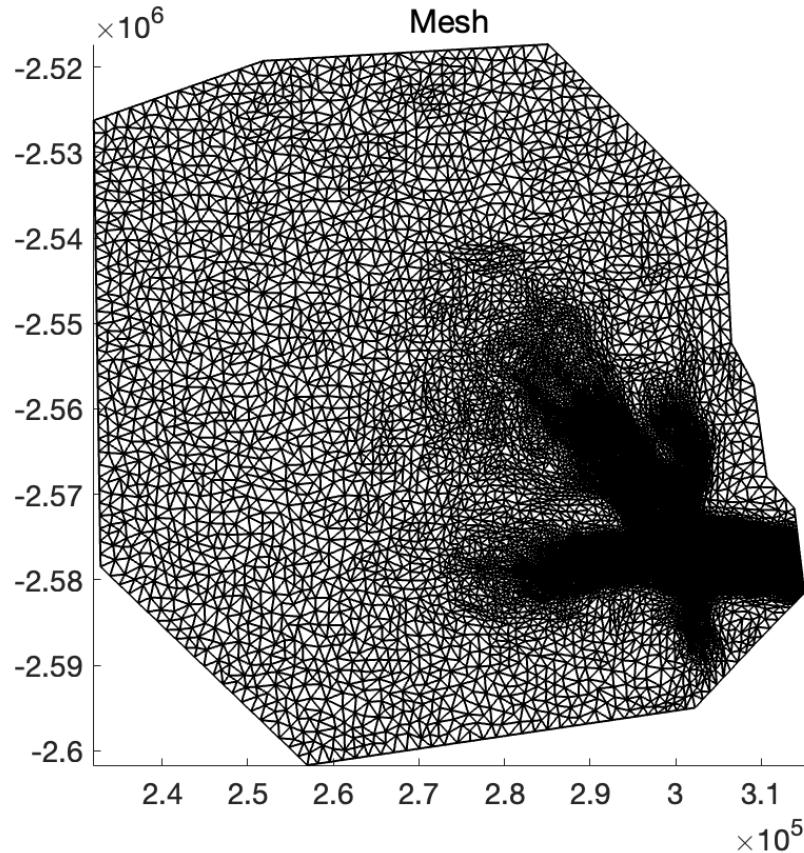
- Create a uniform mesh
- Refine the mesh using anisotropic mesh refinement based on surface velocity
- Set the mesh parameters
- Load observed surface velocities
- Interpolate velocity onto a coarse mesh. Adapt the mesh to minimize error in velocity interpolation
- Save the model

Execute the `runme.m` file to perform step 1. Plot the mesh with:

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

You should see the following figure, with finer resolution in the two main branches of Helheim and its main trunk:

Try experimenting with different values in the mesh generation to create finer or coarser meshes.



4.2.15.4 Parameterization

Most of the model parameterization is done through a different file (`Greenland.par`). In this example, we parameterize the following model fields:

- Geometry (ice surface elevation and bed topography, using BedMachine)
- Initialization parameters
- Material parameters
- Friction coefficient
- Boundary conditions

Run step 2 in the `runme.m` file to perform the parameterization.

4.2.15.5 Inversion for basal friction

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

$$\tau_b = -\beta^2 N^r \|\mathbf{v}_b\|^{s-1} \mathbf{v}_b \quad (4.9)$$

- τ_b : Basal stress (basal drag)
- N : Effective pressure (ice overburden pressure - water pressure at the bed)

- v_b : Basal velocity (equal to surface velocity 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:

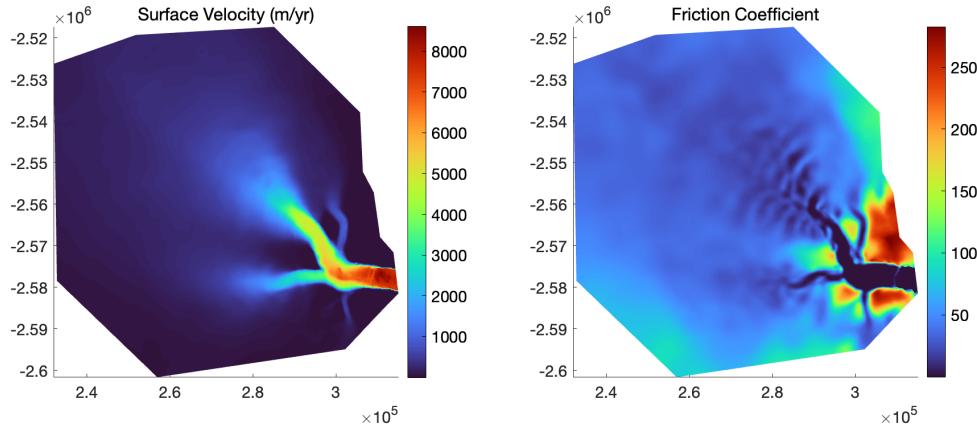
- Velocity is computed from the SSA approximation
- Misfit of the cost function is computed
- Friction coefficient is modified following the gradient of the cost function

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

Run step 3. This will take a few minutes to run. Plot the resulting velocity and friction coefficient:

```
plotmodel(md, 'data', md.initialization.vel, 'title', 'Surface
Velocity (m/yr)', ...
'data', md.friction.coefficient, 'title', 'Friction Coefficient')
```

You should see something similar to the figure below:



Try experimenting with different cost-function values and other parameters in `md.inversion`.

Now that you have run an inversion for basal friction and have a working stress-balance model of Helheim Glacier, you are ready for the next tutorial on simulating subglacial hydrology at Helheim using the SHAKTI model.

4.2.16 Subglacial Hydrology of Helheim Glacier (SHAKTI)

4.2.16.1 Goals

- Use SHAKTI model to simulate subglacial hydrology of Helheim Glacier in southeast Greenland
- Follow an example of how to set up a SHAKTI hydrology simulation on a real-world glacier
- Learn how to run a two-way coupled hydrology–velocity model with SHAKTI-ISSM
- Run a coupled SHAKTI-ISSM simulation of winter base-state hydrology
- Run a coupled SHAKTI-ISSM simulation with transient seasonal meltwater inputs (distributed and point inputs)

4.2.16.2 Introduction

In this example, the main goal is to set up a subglacial hydrology simulation using the SHAKTI model, coupled with ice velocity on a real Greenland outlet glacier. In order to build an operational simulation of Helheim Glacier as an example, we will follow these steps:

- Load your Helheim Glacier model created in the ‘Modeling Helheim Glacier tutorial
- Set the hydrology model to SHAKTI and set up friction coupling
- Set SHAKTI-specific hydrology parameters
- Run a transient two-way coupled simulation with zero meltwater input to generate the winter base state drainage system
- Set up and run two-way coupled seasonal simulations with prescribed distributed meltwater input and point meltwater inputs

Files needed for this tutorial can be found in `<ISSM_DIR>/examples/HelheimSHAKTI/`. This tutorial begins from a model of Helheim Glacier generated in the ‘[Modeling Helheim Glacier](#)’ tutorial.

4.2.16.3 Load model

The first step in the `runme.m` file is to load the model of Helheim Glacier created in the previous tutorial. We turn off the inversion now.

4.2.16.4 Set up SHAKTI subglacial hydrology model

```
In the runme.m file, we set the hydrology model to SHAKTI:  
md.hydrology = hydrologyshakti();
```

Next, we initialize the SHAKTI-specific hydrological parameters:

- Distributed meltwater input ("englacial input") (m yr^{-1})
- Point meltwater input ("moulin input") ($\text{m}^3 \text{s}^{-1}$)
- Initial hydraulic head (m)
- Initial subglacial gap height (m)
- Typical bed bump height and spacing (m)
- Initial Reynolds number
- Boundary conditions (prescribed head for thin ice and ice-free elements)

4.2.16.5 Define coupling and friction

We turn on the coupling between SHAKTI and ISSM through `md.transient` and `md.friction.coupling`. This tutorial uses the Budd-type sliding law.

- To turn on SHAKTI, set `md.transient.ishydrology = 1`.
- To solve for stress balance in the transient simulation, set `md.transient.isstressbalance = 1` and `md.friction.coupling = 4`.
- To run stand-alone SHAKTI without evolving velocity, set `md.transient.isstressbalance = 0`.

4.2.16.6 Run a winter simulation

The final step before running is to define the time step and final time of the simulation. Note that the time step and final time set in years, so make sure to convert appropriately. Small time steps on the order of 1 hour are typically functional for SHAKTI, but feel free to experiment with this.

The model will take a while to run, exactly how long will vary depending on your final time, time step, how many processors you are using, and mesh resolution. If you are running a long simulation, you might not want to save model output at every time step and can reduce the output file size through `md.settings.output_frequency` (for example, with a time step of 1 hour, you would set `md.settings.output_frequency = 24`; to save output once every day).

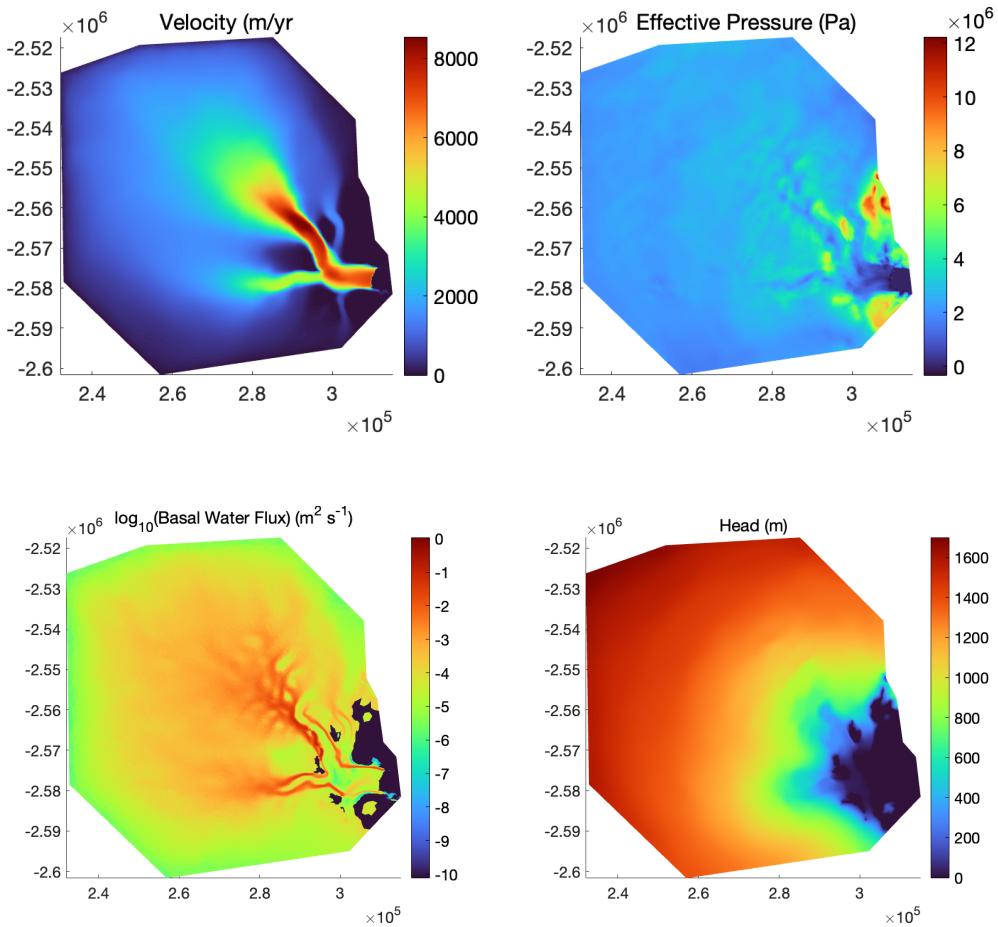
The model you have set up currently specifies zero meltwater inputs (`md.hydrology.englacial_input` and `md.hydrology.moulin_inputs` are both zero everywhere). This represents winter conditions, when all water at the bed is generated via basal melt by geothermal flux, frictional heat from sliding, and turbulent dissipation.

You can examine the output spatially by plotting different quantities. For example, here are plots of velocity, effective pressure, basal water flux, and hydraulic head after 30 days:

```
plotmodel(md, 'data', md.results.TransientSolution(end).Vel,
          'title', 'Velocity (m/yr', ...
          'data', md.results.TransientSolution(end).EffectivePressure,
          'title', 'Effective Pressure (Pa)')
```

```
plotmodel(md, 'data',
          log10(md.results.TransientSolution(end).HydrologyBasalFlux,
          'title', 'log_{10}(Basal Water Flux) (m^2 s^{-1})', ...
          'data', md.results.TransientSolution(end).HydrologyHead, 'title',
          'Head (m)')
```

If you are interested in a steady state, check convergence to a steady winter state by comparing `md.results.TransientSolution(end).Vel` and `md.results.TransientSolution(end).EffectivePressure` with the previous time step. You will probably need to run for a year or two for the system to fully equilibrate.



4.2.16.7 Continuing a simulation

You may find it helpful to continue a simulation from the end state of a previous simulation. This can be useful for running long simulations in segments or exploring different forcing from a common initialized state. Use the script below to continue a previous simulation, which sets the relevant initial parameters accordingly:

```
% runme script to continue a coupled SHAKTI-ISSM simulation,
% continuing from
% end state of a previous simulation

clear all; close all

% Load the model you want to begin from
load Models/your_shaktiissm_model_name.mat

% Starting conditions from end of previous simulation
md.hydrology.head = md.results.TransientSolution(end).HydrologyHead;
md.hydrology.gap_height =
    md.results.TransientSolution(end).HydrologyGapHeight;
md.hydrology.reynolds =
    md.results.TransientSolution(end).HydrologyBasalFlux ./ 1.787e-6;
md.hydrology.reynolds(md.hydrology.reynolds == 0) = 1;
```

```

md.initialization.vx = md.results.TransientSolution(end).Vx;
md.initialization.vy = md.results.TransientSolution(end).Vy;
md.initialization.vel = md.results.TransientSolution(end).Vel;

% Time-stepping
md.timestepping.time_step = 3600 / md.constants.yts; % Time step (in
years)
md.timestepping.final_time = 365 / 365;
md.settings.output_frequency = 24;

md.cluster = generic('np', 8);
md.verbose.solution = 1;
md = solve(md, 'Transient');

```

4.2.16.8 Seasonal meltwater inputs

Meltwater inputs can be added through two options:

- Distributed meltwater input (e.g. for highly crevassed regions): `md.hydrology.englacial_input` (units of m yr^{-1})
- Point inputs to represent discrete crevasses or moulin: `md.hydrology.moulin_input` (units of $\text{m}^3 \text{s}^{-1}$)

Both types of meltwater input are spatially variable and can be prescribed as either constant in time (`size(md.mesh.numberofvertices, 1)`) or time-varying (`size(md.mesh.numberofvertices + 1, length(timevec))`). The sample code below includes example syntax for setting these meltwater inputs. Get creative and experiment with different combinations!

```

% Time-stepping
md.timestepping.time_step = 3600 / md.constants.yts; % Time step (in
years)
md.timestepping.final_time = 365 / 365;
timevec = 0:md.timestepping.time_step:md.timestepping.final_time;
md.settings.output_frequency = 24;

% To prescribe distributed meltwater inputs:
% Steady distributed meltwater inputs:
md.hydrology.englacial_input = zeros(md.mesh.numberofvertices, 1);
md.hydrology.englacial_input(:) = **set values here, can vary
spatially**;

% Time-varying distributed meltwater inputs:
% Initialize the matrix as zeros
md.hydrology.englacial_input = zeros(md.mesh.numberofvertices + 1,
length(timevec));
% Set the final column to be a time index
md.hydrology.englacial_input(end, :) = timevec;
% Example: low-elevation distributed meltwater inputs
le = find(md.geometry.surface <= 900); % Find low-elevation vertices
below 900 m
for nv=1:length(le)

```

```
md.hydrology.englacial_input(le(nv), :) = <<your data or function
    here>>;
end

% To prescribe point meltwater inputs:
% Example: Steady input into firn aquifer crevasse drainage points
% Find high-elevation crevasse input points from firn aquifer at
1500 m
he = find(md.geometry.surface >= 1500 & md.geometry.surface <= 1515);
highieb = 1.5855 / size(he, 1); % Use 50e6 m^3/yr, converted to
m^3/s, divided evenly between eligible points
md.hydrology.moulin_input = zeros(md.mesh.numberofvertices, 1);
md.hydrology.moulin_input(he) = highieb;

% For time-varying point inputs:
md.hydrology.moulin_input = zeros(md.mesh.numberofvertices + 1,
length(timevec));
md.hydrology.moulin_input(end, :) = timevec;
md.hydrology.moulin_input(1:end-1, :) = **set values here, can vary
spatially and temporally**
```

4.2.16.9 Resources

For more details about the SHAKTI model and applications to Helheim Glacier, please see the following references: ???.

4.2.17 Adaptive Mesh Refinement (AMR)

4.2.17.1 Goals

In this tutorial, we show how to use the mesher BAMG to run a simulation with AMR:

- Learn how to set up the AMR properties and a refinement criterion;
- Run a transient simulation with AMR using the MISMIP3d setup to track the grounding line migration.

Go to `<ISSM_DIR>/examples/AMR/` to do this tutorial.

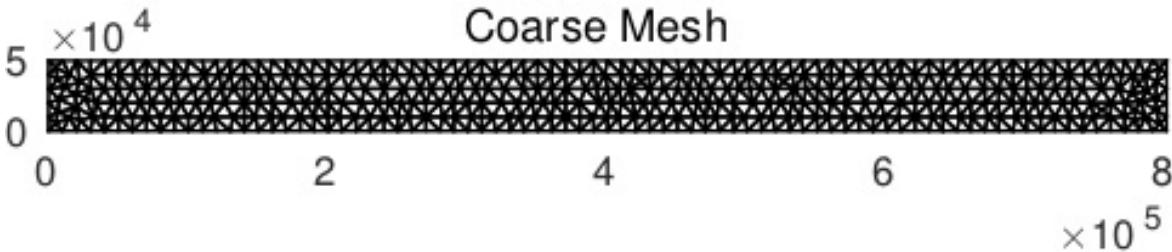
4.2.17.2 Introduction

The `runme.m` file and `mismip.par` go through the steps and basic structure to set up and run the MISMIP3d experiment with adaptive mesh refinement to track the grounding line positions. The `runme.m` script is set up as three distinct steps, saving the model at each stage:

1. Mesh generation
2. Parameterization
3. Transient solution with AMR

4.2.17.3 Mesh Generation

Run step 1 in `runme.m` to generate an unstructured coarse mesh on a 800 x 50 km domain with typical element edge length of 10,000 m (10 km). This coarse mesh shown here has 820 elements and 496 vertices. To plot your coarse mesh, use `plotmodel(md, 'data', 'mesh', 'fontsize', 12);`:



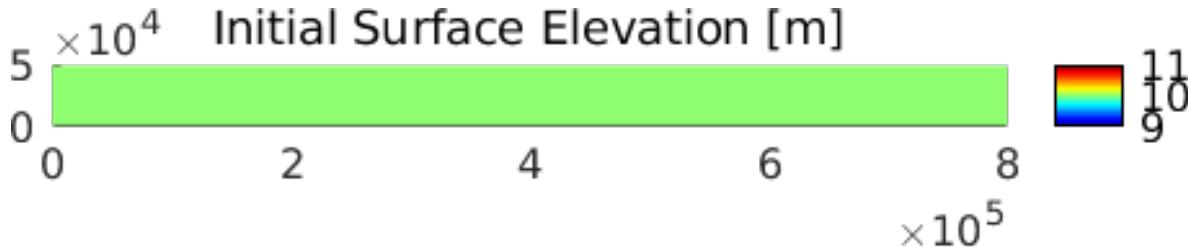
4.2.17.4 Parameterization

Run step 2 in `runme.m` to define the model parameters. First we call on standard parameters defined in the `mismip.par` file (bed and ice geometry, sliding velocity, material properties, etc.). Then we define AMR-specific parameters to run an AMR transient simulation (resolution at the grounding line, distance to the grounding line used as criterion, ratio between two consecutive edges, etc.).

The MISMIP3d domain is initially set up as a 100 m thick slab of ice. The MISMIP3d bed is defined as $r = -100 - x/1000$ (in [m], negative if below sea level). The surface mass balance is constant over the domain and equal to 0.5 m/yr. A Weertman-type friction law is applied on the grounded ice. The basal friction coefficient is uniform over the domain and equal to $10^7 \text{ Pa m}^{-1/3} \text{s}^{1/3}$. The ice viscosity parameter, B ($= A^{1/n}$) is equal to $2.15 \times 10^8 \text{ Pas}^{-1/3}$.

To look at the initial ice surface, you can plot it in MATLAB:

```
plotmodel(md, 'data', md.geometry.surface, 'title', 'Initial Surface
Elevation [m]', 'fontsize', 12);
```



4.2.17.5 Transient solution with AMR

In step 3, we specify which machine we want to run the model on, including number of processors to be used, define the model time step, final time, and prescribe the AMR frequency, i.e, how often the mesh needs to be updated. In this example, we run 500 yr forward in time to track the grounding line movement as soon as the initial thin ice slab starts to grounded on the bedrock. The ice starts to grounded in $x=0$, the boundary of the ice divide ($v_x=0$ at $x=0$). We set the AMR frequency equal to 1, what means that the mesh is update (refined/coarsen) very time step). In this example, a time step equal to 1 yr is imposed. The SSA equations are used as the flow model.

Now that the set up is complete, we can run the model:

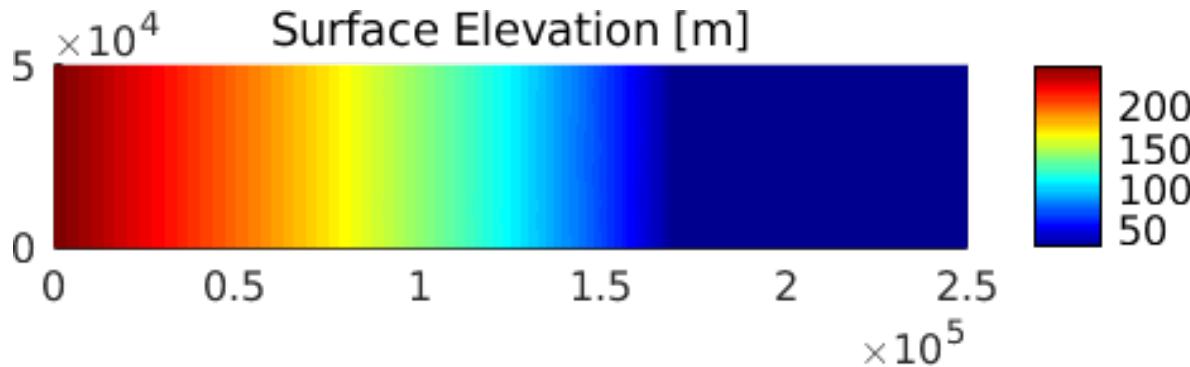
```
md = solve(md,'Transient');
```

The solutions at the end of transient simulation ($t=500$ yr) can be visualized by plotting. Here, we plot the ice surface near the ice divide boundary ($x=0$ to $x=250$ km):

```
finalstep = length(md.results.TransientSolution);
plotmodel(md, 'data',
          md.results.TransientSolution(finalstep).Surface, 'title',
          'Surface Elevation [m]', 'amr', finalstep, 'xlim', [0 250000],
          'fontsize', 12);
```

or in Python:

```
finalstep = len(md.results.TransientSolution) - 1
plotmodel(md, 'data', md.results.TransientSolution[-1].Surface,
          'title', 'Surface Elevation [m]', 'amr', finalstep, 'xlim', [0,
          250000], 'fontsize', 12, 'figure', 1)
```

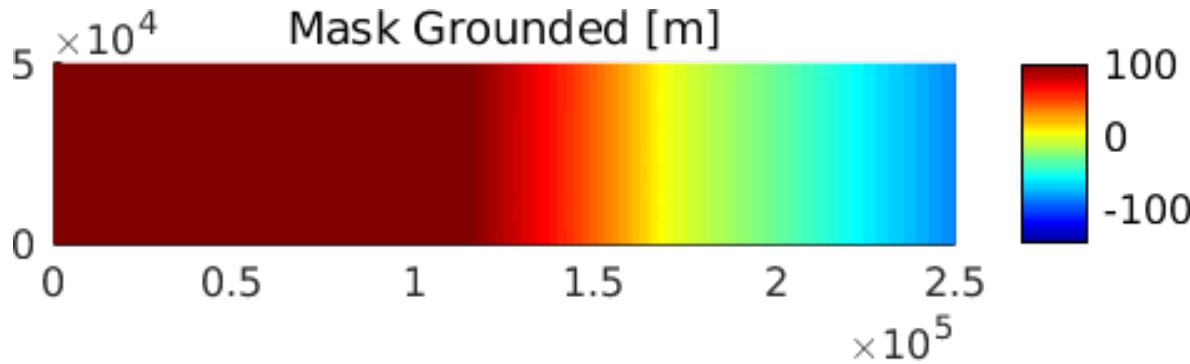


Here, we are plotting the mask grounded level set, which indicates if the ice is grounded (positive) or floating (negative). The value 0 indicates the position of the grounding line:

```
finalstep = length(md.results.TransientSolution);
plotmodel(md, 'data',
          md.results.TransientSolution(finalstep).MaskGroundediceLevelset,
          'title', 'Mask Grounded [m]', 'amr', finalstep, 'xlim', [0 250000], 'caxis', [-150 100], 'fontsize', 12);
```

or in Python:

```
plotmodel(md, 'data',
          md.results.TransientSolution[-1].MaskOceanLevelset,
          'title', 'Mask Grounded [m]', 'amr', finalstep, 'xlim', [0, 250000],
          'caxis', [-150, 100], 'fontsize', 12, 'figure', 2)
```



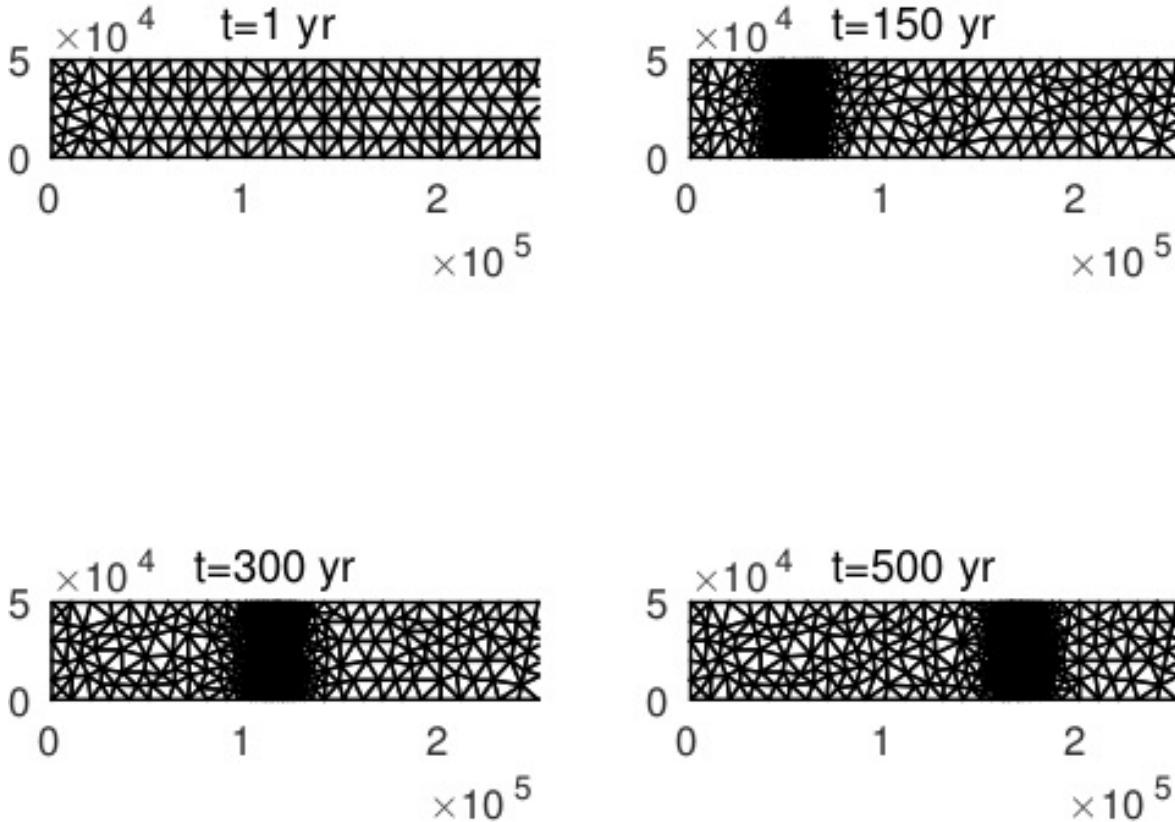
You can see the grounding line position is about $x=170$ km at the end of this example. To visualize the adaptive meshes, you can plot them at any saved time specifying the corresponding step:

```
>> plotmodel(md, 'data', 'mesh', 'amr', 1, 'xlim', [0 250000],
              'title', 't=1 yr', 'fontsize', 12, ...
              'data', 'mesh', 'amr', 16, 'xlim', [0 250000], 'title', 't=150
              yr', 'fontsize', 12, ...
              'data', 'mesh', 'amr', 31, 'xlim', [0 250000], 'title', 't=300
              yr', 'fontsize', 12, ...
```

```
'data', 'mesh', 'amr', 51, 'xlim', [0 250000], 'title', 't=500
yr', 'fontsize', 12);
```

or in Python:

```
plotmodel(md, 'data', 'mesh', 'amr', 0, 'xlim', [0, 250000],
'title', 't=1 yr', 'fontsize', 12,
'data', 'mesh', 'amr', 15, 'xlim', [0, 250000], 'title', 't=150
yr', 'fontsize', 12,
'data', 'mesh', 'amr', 30, 'xlim', [0, 250000], 'title', 't=300
yr', 'fontsize', 12,
'data', 'mesh', 'amr', 50, 'xlim', [0, 250000], 'title', 't=500
yr', 'fontsize', 12, 'figure', 3)
```



To watch the evolution through time in an animation, we print the results and the respective meshes in .VTK-type file format, see the folder `<ISSM_DIR>/examples/AMR/`. These files can be seen using [ParaView](#).

In ParaView, you will select which result to animate, and can watch the mesh tracking the grounding line movement as soon as the ice starts to grounded on the bedrock. The result and the mesh can be simultaneously displayed using selecting `Surface With Edges` in the box next to the field/result box selection.

4.2.18 Sea-Level Fingerprints (GRACE)

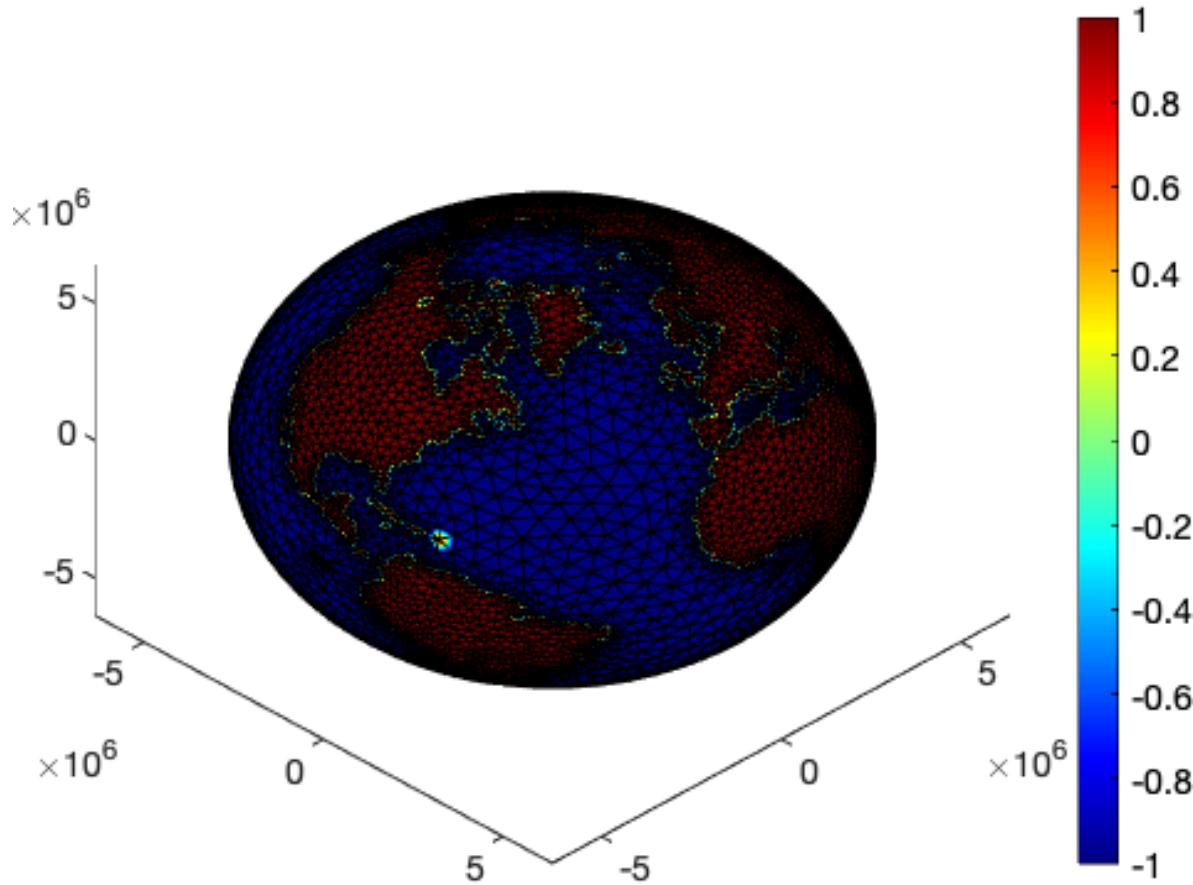
4.2.18.1 Goals

- Setup a ISSM-SESAW model with GRACE-based forcing
- Run the model to compute sea-level fingerprints

Go to `<ISSM_DIR>/examples/SlrGRACE/` to do this tutorial.

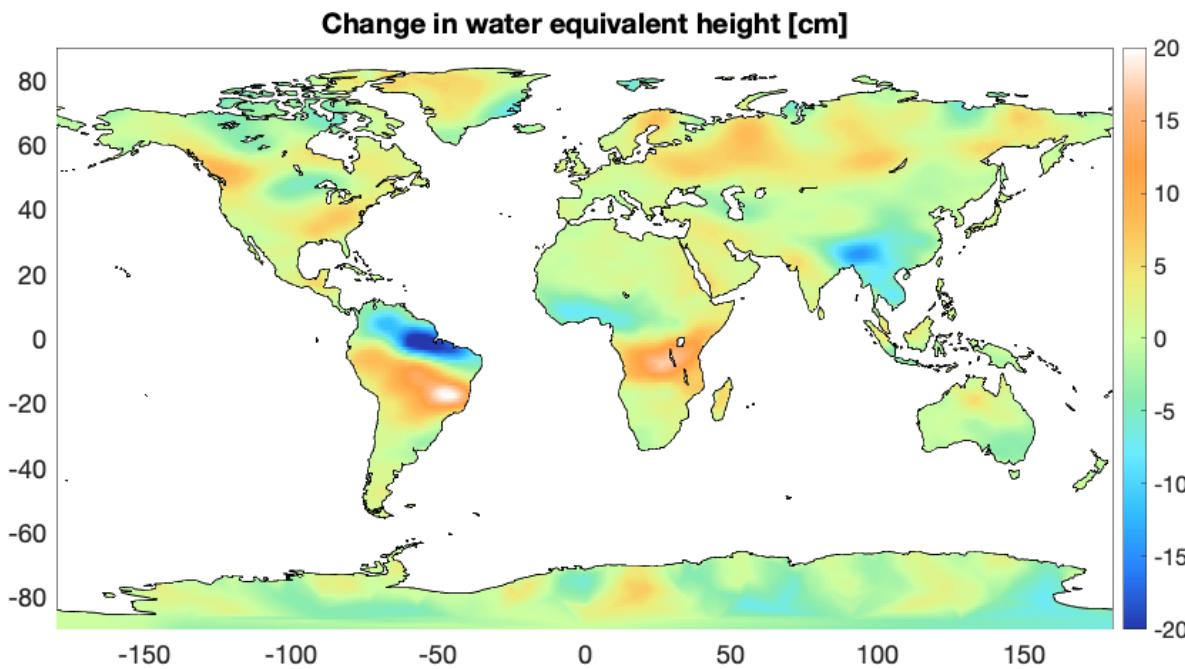
4.2.18.2 Mesh

Set `steps = 1` to create an unstructured global mesh. Choose `mindistance_coast`, `mindistance_lan`, and `maxdistance` as you wish for generating a mesh. The nominal parameters should generate the following mesh:



4.2.18.3 GRACE loads

Set `steps = 2` to load GRACE-based estimate of water equivalent height (WEH) change for a chosen month. Choose `year_month` as you wish. The nominal month is January 2007 and here is the load model (cf. `steps = 5` for plotting):



4.2.18.4 Parameterization

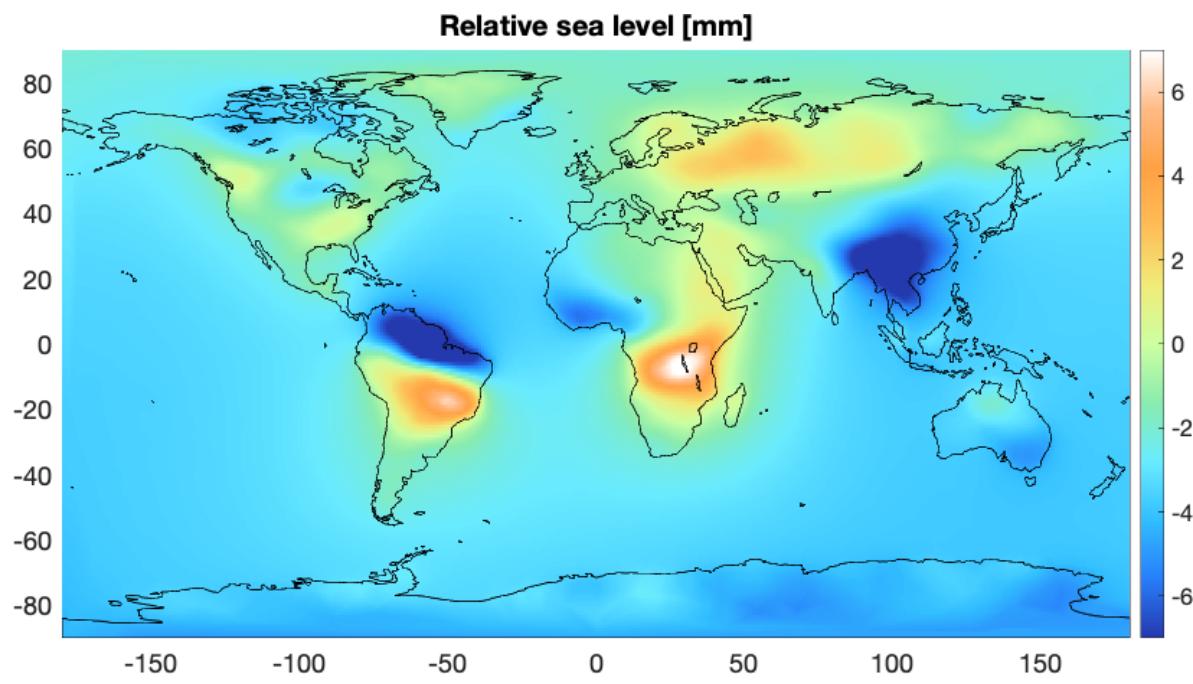
In the next step, you will load the Earth model. The nominal model is PREM; `lovenumbers` reads the associated Love numbers. You will also have to set up some standard parameters regarding ice sheets for passing the consistency.

4.2.18.5 Solve Model

In `steps = 4`, you will choose the solid-Earth physics (e.g., gravitation, viscoelasticity, and rotation) that you wish to consider. You may also request model outputs (e.g., sea level and bedrock motion). You must set `masstransport` and `slc` flags on before solving the `transient` model. See `steps = 6` for running a model with multiple (transient) loads.

4.2.18.6 Some Results

Once the model run is completed, you may plot results. Some useful plotting scripts are located in `steps = 5` and `steps = 7`. In the latter, you can also find a script to make an animation. Here is an example result for January 2007:



4.3 Capabilities

- Mesh Generation
- Setting a Mask
- Interpolation Routines
- Glacial Flow Approximation
- Stress Balance Solution
- Mass Transport
- Thermal Solution
- Hydrology Solutions
 - Dual Continuum Porous Equivalent Approach
 - GlaDS
 - SHAKTI
 - Shreve Approximation
- Damage Mechanics
- Transient Solution
- Grounding Line Migration
- Ice Front Migration (level-set method)
- Glacial Isostatic Adjustment (GIA) Solution
- Elastostatic Adjustment Solution
- Sea-level Fingerprints Solution
- Verbosity

4.3.1 Mesh Generation

4.3.1.1 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.

4.3.1.2 triangle

`triangle` is a wrapper of [triangle](#) developed by [Jonathan Shewchuk](#) [?]. 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 run:

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

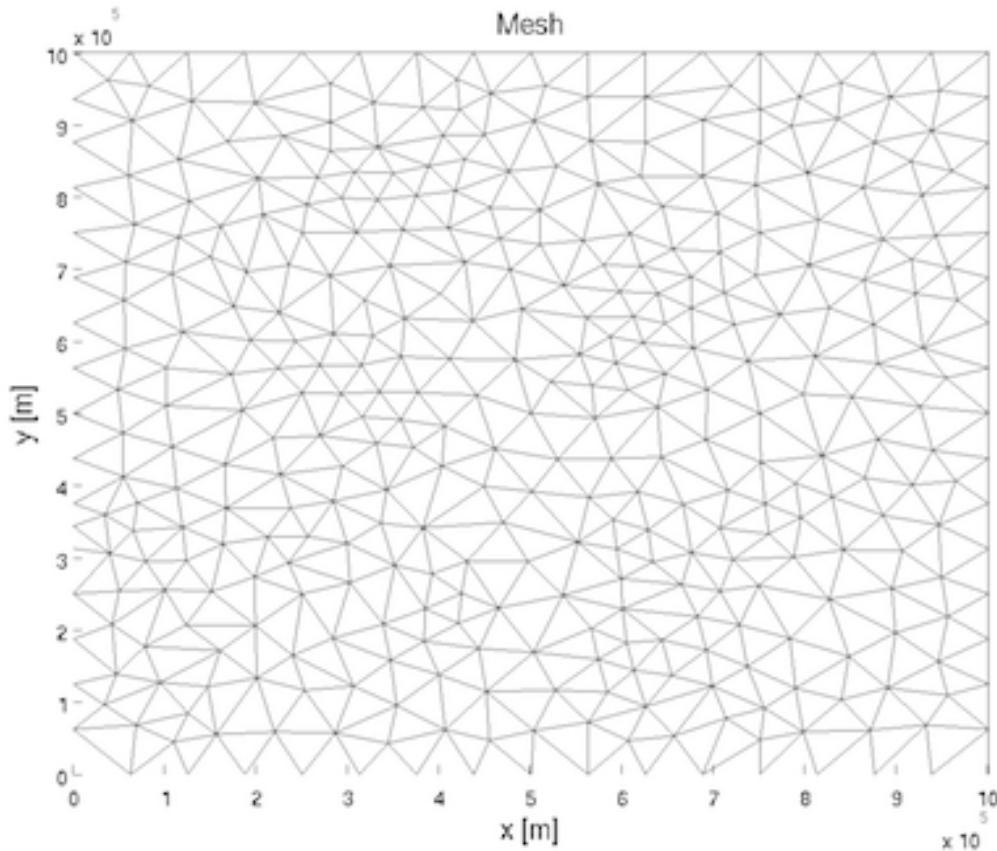


Figure 4.4: Mesh

ISSM includes a mesh adaptation capability embedded in the code, inspired by [BAMG](#) developed by Frederic Hecht [?], and YAMS developed by Pascal Frey [?].

4.3.1.3 Bamg

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`, run:

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

hmin/hmax

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

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

hVertices

One can specify the edge length of domain outline vertices. Use `NaN` if an edge length value is not required/available:

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

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);
```

Multiple fields can also be used:

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

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);
```

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);
```

NOTE: Users using Intel compilers (`icc`, `icpc`) shoud use the flag `-fp-model precise` to disable optimizations that are not value-safe on floating-point data. This will prevent `bamg` from being compiler dependent (see [here](#)).

4.3.1.4 Extrusion (3D)

One can extrude the mesh, in order to use a three-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, run 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.

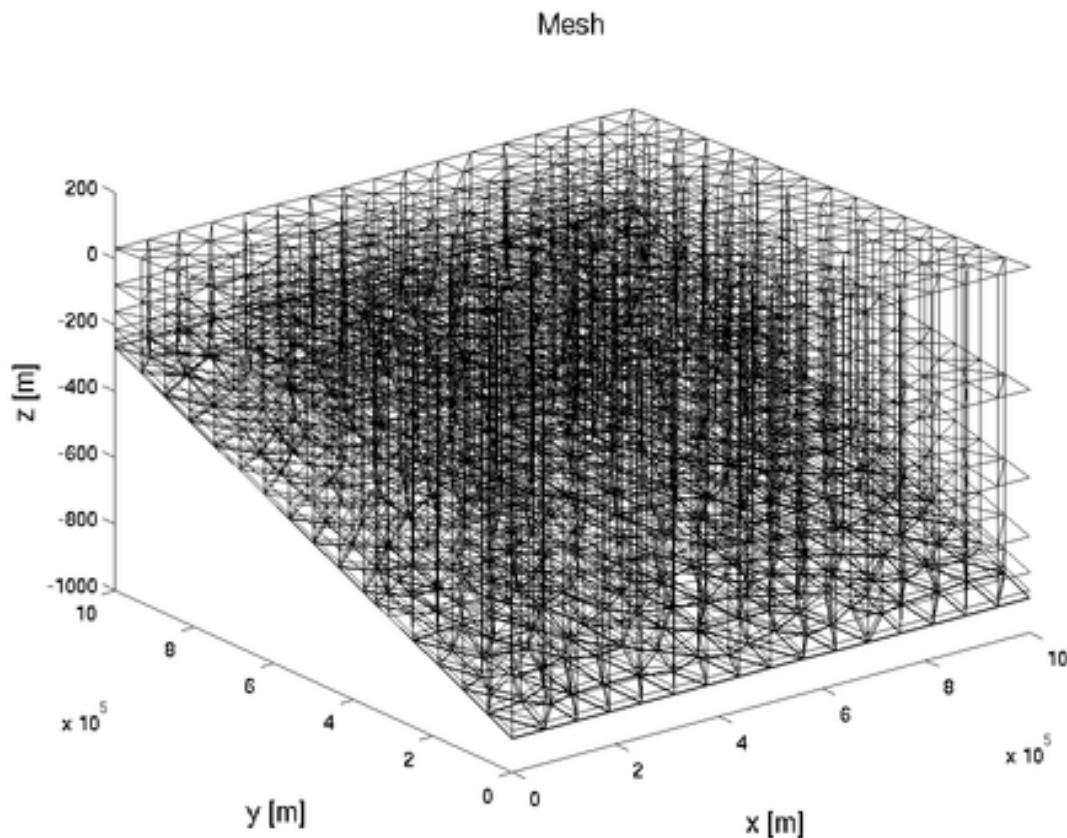


Figure 4.5: Extruded mesh

4.3.2 Setting a Mask

The solver will use two different boundary conditions depending on the nature of the ice sheet system. An ice shelf will slide on the water whereas there is friction between an ice sheet and the bedrock for the grounded ice. The model must contain this field that tells whether the element is on an ice shelf or on an ice sheet. If the whole domain is an ice shelf, the following command may be used,

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

and for an ice sheet,

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

If the geometry is more complex (marine ice sheet, ice shelf with ice rises), ARGUS files must be used. The following command will set the geometry,

```
>> md = setmask(md, 'Iceshelves.exp', 'Islands.exp');
```

The first argument is the model and the two other arguments are the files containing the coordinates of the ice shelf included in the Domain Outline and the part of grounded ice included in the ice shelf part (islands or ice rises).

4.3.3 Interpolation Routines

Several interpolation routines can be used in order to interpolate datasets onto the mesh vertices.

`ContourToMesh` is used to flag the nodes and/or elements that are within a contour from an Argus contour and a mesh. For example,

```
gridinsidefront = ContourToMesh(md.mesh.elements, md.mesh.x,
    md.mesh.y, expread('Front.exp', 1), 'node');
```

To interpolate a field from a structured grid to an unstructured mesh (or any list of points), one can use `InterpFromGridToMesh`,

```
data_mesh = InterpFromGridToMesh(x_grid, y_grid, data, x_mesh,
    y_mesh)
```

To interpolate a field from a 2d mesh to a 2d mesh (or any list of points), one can use `InterpFromMeshToMesh2d`,

```
data_mesh2 = InterpFromMeshToMesh2d(index_mesh1, x_mesh1, y_mesh1,
    data, x_mesh2, y_mesh2)
```

To interpolate a field from a 3d mesh to a 3d mesh (or any list of points), one can use `InterpFromMeshToMesh3d`,

```
data_mesh2 = InterpFromMeshToMesh3d(index_mesh1, x_mesh1, y_mesh1,
    z_mesh1, data, x_mesh2, y_mesh2, z_mesh2)
```

4.3.4 Glacial Flow Approximation

ISSM has the capability to compute the flow of a glacier with 4 different models:

- Shallow Ice Approximation (SIA, 2d and 3d)
- MacAyeal/Morland's Shelfy Stream Approximation (SSA, 2d and 3d)
- Blatter/Pattyn's Higher-Order model (HO, 3d: extruded mesh only)
- Full-Stokes' model (FS, 3d: extruded mesh only)

The ice flow model is specified for each element of the mesh. To assign the models to the elements, as an example the following command can be used:

```
>> md = setflowequation(md, 'HO', 'Pattyn.exp', 'SSA',
    md.mask.ocean_levelset < 0., 'fill', 'SIA');
```

The routine `setflowequation` works like `plotmodel`: it requires an even number of arguments (without counting `md` itself). There are five possible options:

- '`SIA`'
- '`SSA`'
- '`HO`'
- '`FS`'
- '`fill`'

The first four options must be followed by one of the following arguments:

- An ARGUS file containing a closed contour, the elements inside the contour will be assigned to the model given by the option. If user wants to assign the model to the elements outside the domain, add ' ' to the name of the domain file (ex: ' Pattyn.exp')
- A vector of size `md.numberofelements` holding 0, and 1 on the elements that the user had flagged. The model given by the option will be assigned to the elements flagged only.
- '`all`' if the user wants to assign the model to all the elements

The last option '`fill`' must be followed by the name of the model that the user wants the other elements (that have not been flagged by the other options) assigned to. All options are not required to be used. The previous example assigns the model of Pattyn for the elements inside the contour `Pattyn.exp` and the model of MacAyeal for the elements located on the ice shelf. The other elements are Hutter's elements. If the user wants to use MacAyeal's model only, type the following command:

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

4.3.5 Stress Balance Solution

4.3.5.1 Physical basis

Conservation of linear momentum

The conservation of momentum reads:

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \quad (4.10)$$

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:

$$\nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} = \mathbf{0} \quad (4.11)$$

Conservation of angular momentum

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

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^T \quad (4.12)$$

Ice constitutive equations

Ice is treated as a purely viscous incompressible material ?. Its constitutive equation therefore only involves the deviatoric stress and the strain rate tensor:

$$\boldsymbol{\sigma}' = 2 \mu \dot{\boldsymbol{\epsilon}} \quad (4.13)$$

where:

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

Ice is a non-Newtonian fluid, its viscosity follows the generalized Glen's flow law ?:

$$\mu = \frac{B}{2 \dot{\boldsymbol{\epsilon}}_e^{\frac{n-1}{n}}} \quad (4.14)$$

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:

$$\dot{\varepsilon}_e = \sqrt{\frac{1}{2} \sum_{i,j} \dot{\varepsilon}_{ij}^2} = \frac{1}{\sqrt{2}} \|\dot{\varepsilon}\|_F \quad (4.15)$$

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

Full-Stokes (FS) field equations

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

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 ??:

$$\begin{aligned} \nabla \cdot (2\mu \dot{\varepsilon}_{HO1}) &= \rho g \frac{\partial s}{\partial x} \\ \nabla \cdot (2\mu \dot{\varepsilon}_{HO2}) &= \rho g \frac{\partial s}{\partial y} \end{aligned} \quad (4.16)$$

with:

$$\dot{\varepsilon}_{HO1} = \begin{bmatrix} 2 \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \\ \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \\ \frac{1}{2} \frac{\partial v_x}{\partial z} \end{bmatrix} \quad \dot{\varepsilon}_{HO2} = \begin{bmatrix} \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \\ \frac{\partial v_x}{\partial x} + 2 \frac{\partial v_y}{\partial y} \\ \frac{1}{2} \frac{\partial v_y}{\partial z} \end{bmatrix} \quad (4.17)$$

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 [??]:

$$\begin{aligned} \nabla \cdot (2\bar{\mu} H \dot{\varepsilon}_{SSA1}) - \alpha^2 v_x &= \rho g H \frac{\partial s}{\partial x} \\ \nabla \cdot (2\bar{\mu} H \dot{\varepsilon}_{SSA2}) - \alpha^2 v_y &= \rho g H \frac{\partial s}{\partial y} \end{aligned} \quad (4.18)$$

with:

$$\dot{\varepsilon}_{SSA1} = \begin{bmatrix} 2\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \\ \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \end{bmatrix} \quad \dot{\varepsilon}_{SSA2} = \begin{bmatrix} \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \\ \frac{\partial v_x}{\partial x} + 2\frac{\partial v_y}{\partial y} \end{bmatrix} \quad (4.19)$$

where:

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

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:

$$\begin{aligned} \boldsymbol{\sigma} \cdot \mathbf{n} &= \mathbf{0} && \text{on } \Gamma_s \\ (\boldsymbol{\sigma} \cdot \mathbf{n} \cdot \mathbf{n} + \alpha^2 \mathbf{v})_{\parallel} &= \mathbf{0} && \text{on } \Gamma_b \\ \mathbf{v} \cdot \mathbf{n} &= 0 && \text{on } \Gamma_b \\ \boldsymbol{\sigma} \cdot \mathbf{n} &= \rho_w g z \mathbf{n} && \text{on } \Gamma_w \end{aligned} \quad (4.20)$$

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:

$$\begin{aligned} \dot{\varepsilon}_{HO1} \cdot \mathbf{n} &= 0 & \dot{\varepsilon}_{HO2} \cdot \mathbf{n} &= 0 && \text{on } \Gamma_s \\ 2\mu \dot{\varepsilon}_{HO1} \cdot \mathbf{n} &= -\alpha^2 v_x & 2\mu \dot{\varepsilon}_{HO2} \cdot \mathbf{n} &= -\alpha^2 v_y && \text{on } \Gamma_b \\ 2\mu \dot{\varepsilon}_{HO1} \cdot \mathbf{n} &= f_w n_x & 2\mu \dot{\varepsilon}_{HO2} \cdot \mathbf{n} &= f_w n_y && \text{on } \Gamma_w \end{aligned} \quad (4.21)$$

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

For SSA, these boundary conditions are:

$$\dot{\varepsilon}_{SSA1} \cdot \mathbf{n} = 0 \quad \dot{\varepsilon}_{SSA2} \cdot \mathbf{n} = 0 \quad \text{on } \Gamma_s \quad (4.22)$$

$$\begin{aligned} 2\bar{\mu} H \dot{\varepsilon}_{SSA1} \cdot \mathbf{n} &= \left(\frac{1}{2} \rho g H^2 - \frac{1}{2} \rho_w g b^2 \right) n_x && \text{on } \Gamma_w \\ 2\bar{\mu} H \dot{\varepsilon}_{SSA2} \cdot \mathbf{n} &= \left(\frac{1}{2} \rho g H^2 - \frac{1}{2} \rho_w g b^2 \right) n_y \end{aligned} \quad (4.23)$$

4.3.5.2 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.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:

$$\kappa = 10^{\text{penalty_factor}} \max_{i,j} |K_{ij}| \quad (4.24)$$

- `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`: z component of velocity (used as an initial guess)

4.3.5.3 Running a simulation

To run a simulation, use the following command:

```
>> md = solve(md, 'Stressbalance');
```

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

4.3.6 Mass Transport Solution

4.3.6.1 Physical basis

Conservation of mass

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

$$\frac{\partial H}{\partial t} = -\nabla \cdot (H \bar{v}) + \dot{M}_s - \dot{M}_b \quad (4.25)$$

where:

- \bar{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:

$$\frac{\partial s}{\partial t} + v_x(s) \frac{\partial s}{\partial x} + v_y(s) \frac{\partial s}{\partial y} - v_z(s) = \dot{M}_s \quad (4.26)$$

and:

$$\frac{\partial b}{\partial t} + v_x(b) \frac{\partial b}{\partial x} + v_y(b) \frac{\partial b}{\partial y} - v_z(b) = \dot{M}_b \quad (4.27)$$

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

Boundary conditions

Ice thickness is imposed at the inflow boundary:

$$H = H_{obs} \text{ on } \Gamma_- \quad (4.28)$$

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

Numerical implementation

Mass transport is solved using finite elements in space, and implicit finite difference in time. To stabilize the equation, a stabilization term might be added to the left hand side, for example:

$$\frac{\partial H}{\partial t} + \nabla \cdot (H \bar{v}) - \nabla \cdot (\mathfrak{D} \nabla H) = \dot{M}_s - \dot{M}_b \quad (4.29)$$

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

$$\mathfrak{D} = \frac{h}{2} \begin{pmatrix} |vx| & 0 \\ 0 & |vy| \end{pmatrix} \quad (4.30)$$

There are other stabilization schemes available in ISSM: (1) Artificial Diffusion, (2) Streamline Upwinding, (3) Discontinuous Galerkin (DG), (4) Flux Corrected Transport (FCT), and (5) Streamline Upwind Petrov-Galerkin (SUPG). They can be used by setting:

```
>> md.masstransport.stabilization = 1;
```

4.3.6.2 Model parameters

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

```
>> 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 diffusion, 2: streamline upwinding 3: discontinuous Galerkin, 4: flux corrected transport (FCT), 5: streamline upwind Petrov-Galerkin (SUPG)
- `md.masstransport.penalty_factor`: offset used by penalties

$$\kappa = 10^{\text{penalty_offset}} \max_{i,j} |K_{ij}| \quad (4.31)$$

- `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.smb.ablation_rate`: surface ablation rate (in meters)
- `md.smb.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.smb.mass_balance`: surface mass balance (in meters/year ice equivalent)
- `md.timestepping.time_step`: length of time steps (in years)

4.3.6.3 Running a simulation

To run a simulation, use the following command:

```
>> md = solve(md, 'Masstransport');
```

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.

4.3.7 Thermal Solution

4.3.7.1 Physical basis

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:

$$\rho \left(\frac{\partial E}{\partial t} + \mathbf{v} \cdot \nabla E \right) = -\nabla (\kappa(E) \nabla E) + \text{Tr}(\boldsymbol{\sigma} \cdot \dot{\boldsymbol{\epsilon}}) \quad (4.32)$$

where radiative sources have been neglected, and:

- \mathbf{v} is the velocity vector
- $\dot{\boldsymbol{\epsilon}}$ 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 becomes:

$$\rho c_i \left(\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T \right) = -c_i \kappa \Delta T + \text{Tr}(\boldsymbol{\sigma} \cdot \dot{\boldsymbol{\epsilon}}) \quad (4.33)$$

Boundary conditions

Dirichlet boundary conditions should be applied at the ice surface:

$$T(z = s) = T_s, \quad (4.34)$$

and Neumann boundary conditions at the ice base:

$$q(z = b) = -\kappa(E) \nabla E = q_{\text{geo}} \quad (4.35)$$

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 ?, figure 5 is used instead of the previous equation.

NOTE: For regional model, make sure to set a Dirichlet condition on the inflow boundary (throughout the ice column) to avoid advection of noise.

Numerical implementation

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

$$\rho \left(\frac{\partial E}{\partial t} + \mathbf{v} \cdot \nabla E \right) + \nabla (\kappa(E) \nabla E) + \nabla (\mathcal{D} \nabla E) = \text{Tr}(\boldsymbol{\sigma} \cdot \dot{\boldsymbol{\epsilon}}) \quad (4.36)$$

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

$$\mathfrak{D} = \frac{h}{2} \begin{pmatrix} |vx| & 0 & 0 \\ 0 & |vy| & 0 \\ 0 & 0 & |vz| \end{pmatrix} \quad (4.37)$$

or rely on the Streamline upwind/Petrov-Galerkin formulation (SUPG) from ?.

4.3.7.2 Model parameters

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

```
>> 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):

$$\kappa = 10^{\text{penalty_factor}} \max_{i,j} |K_{ij}| \quad (4.38)$$

- `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/m²)
- `md.basalforcings.meltingrate`: basal melting rate (in m/yr w.e.)
- `md.timestepping.time_step`: length of time steps (in yrs)

4.3.7.3 Running a simulation

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

```
>> md = solve(md, 'Thermal');
```

This will compute *one time step only* of the thermal equation; use the [transient solution](#) for multiple time steps.

To run a thermal steady-state simulation (i.e. $\partial T / \partial t = 0$), you need to first set the time stepping as 0:

```
>> md.timestepping.time_step = 0  
>> md = solve(md, 'Thermal');
```

4.3.8 Hydrology Solution

ISSM features various hydrology model and solution types,

- [Dual Continuum Porous Equivalent Approach](#)
- [GlaDS](#)
- [SHAKTI](#)
- [Shreve Approximation](#)

4.3.8.1 Hydrology Solution - Dual Continuum Porous Equivalent Approach

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.

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 ($j = i$) or to the EDS ($j = e$):

$$S_j \frac{\partial h_j}{\partial t} - \nabla \cdot (T_j \nabla h_j) = Q_j. \quad (4.39)$$

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:

$$T_j = e_j K_j \quad (4.40)$$

and:

$$S_j = \rho_w \omega_j g e_j \left[\beta_w - \frac{\alpha}{\omega_j} \right], \quad (4.41)$$

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}]

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 manner (See Layer Transfer)

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 ?. Improvements from the version presented in ? 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:

$$\frac{\partial e_e}{\partial t} = g \frac{\rho_w e_e K_e}{\rho_{ice} L_{ice}} (\nabla h_e)^2 - 2 \left[\frac{N}{Bn} \right]^n \quad (4.42)$$

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]

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:

$$Q_t = \varphi(h_i - h_e). \quad (4.43)$$

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 in between the inefficient and efficient layers.

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.

Model parameters

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

```
>> md.hydrology
```

These parameters are of three different types:

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 [$m^3 s^{-1}$]

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]

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]

Running a simulation

To run a transient simulation, use the following command:

```
>> md = solve(md, 'Transient');
```

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, 'Hydrology');
```

4.3.8.2 Hydrology Solution - GlaDS

Description

The two-dimensional Glacier Drainage System model (GlaDS, ?) couples a distributed water sheet model – a continuum description of a linked cavity drainage system [?] – with a channelized water flow model – modeled as R channels [??]. The coupled system collectively describes the evolution of hydraulic potential ϕ , water sheet thickness h , and water channel cross-sectional area S .

Sheet model equations

- Mass conservation: The mass conservation equation describes water storage changes over longer timescales (dictated by cavity opening due to sliding) as well as shorter timescales (e.g. due to surface melt water input):

$$\frac{e_v}{\rho_w g} \frac{\partial \phi}{\partial t} + \frac{\partial h}{\partial t} - \nabla \cdot \mathbf{q} - m_b = 0, \quad (4.44)$$

where: e_v is the englacial void ratio, ρ_w is water density (kg m^{-3}), g is gravitational acceleration (kg m^{-3}), ϕ is the hydraulic potential (Pa), and h is the sheet thickness (m). The water discharge \mathbf{q} ($\text{m}^2 \text{s}^{-1}$) is given by:

$$\mathbf{q} = -k_s h^{\alpha_s} |\nabla \phi|^{\beta_s - 2} \nabla \phi, \quad (4.45)$$

where k_s is the sheet conductivity (m s kg^{-1}), and $\alpha_s=5/4$ and $\beta_s=3/2$ are two constant exponents. Finally, the melt source term m_b (m s^{-1}) is given by:

$$m_b = \frac{G + |\boldsymbol{\tau}_b \cdot \mathbf{u}_b|}{\rho_i L}, \quad (4.46)$$

where G is the geothermal heat flux (W m^{-2}), $|\boldsymbol{\tau}_b \cdot \mathbf{u}_b|$ is the frictional heating (W m^{-2}), for basal stress $\boldsymbol{\tau}_b$ (Pa), ρ_i is ice density (kg m^{-3}), and L is latent heating (J kg^{-1}).

- Sheet thickness:

$$\frac{\partial h}{\partial t} = w_s - v_s. \quad (4.47)$$

Here, w_s is the cavity opening rate due to sliding over bed topography (m s^{-1}), given by:

$$w_s(h) = \begin{cases} \frac{|\mathbf{u}_b|}{l_r} (h_r - h), & \text{if } h < h_r \\ 0, & \text{otherwise,} \end{cases} \quad (4.48)$$

where h_r and l_r are both constants (m), and \mathbf{u}_b is the basal sliding velocity vector (provided by the ice flow model). The cavity closing rate due to ice creep v_s (m s^{-1}), is given by:

$$v_s(h, \phi) = \frac{2A}{n^n} h |N|^{n-1} N, \quad (4.49)$$

where A is the basal flow parameter ($\text{Pa}^{-3} \text{s}^{-1}$), related to the ice hardness by $B = A^{-1/3}$, n is the Glen flow relation exponent, and $N = \phi_0 - \phi$ is the effective pressure. The overburden hydraulic potential is given by $\phi_0 = \phi_m + p$, with the ice pressure $p = \rho_i g H$ and elevation potential $\phi_m = \rho_w g b$, all of which are given in units of Pa.

Channel model equations

- Channel discharge (along mesh edges):

$$\frac{\partial Q}{\partial s} + \frac{\Xi - \Pi}{L} \left(\frac{1}{\rho_i} - \frac{1}{\rho_w} \right) - v_c - m_c = 0, \quad (4.50)$$

where Q is the channel discharge ($\text{m}^3 \text{s}^{-1}$), which evolves with respect to the horizontal coordinate along the channel s , Ξ is the channel potential energy dissipated per unit length and time (W m^{-1}), Π is the channel pressure melting/refreezing (W m^{-1}), v_c is the channel closing rate ($\text{m}^2 \text{s}^{-1}$) and m_c is the source term ($\text{m}^2 \text{s}^{-1}$). The discharge Q is defined as:

$$Q = \underbrace{-k_c S^{\alpha_c} \left| \frac{\partial \phi}{\partial s} \right|^{\beta_c - 2} \frac{\partial \phi}{\partial s}}_{K_c}, \quad (4.51)$$

where k_c is the channel conductivity, and $\alpha_c=3$ and $\beta_c=2$ are two exponents. The term v_c is the closing of the channels by ice creep, and is given by:

$$v_c(S, \phi) = \frac{2A}{n^n} S |N|^{n-1} N, \quad (4.52)$$

where S is the channel cross-sectional area (m^2). Finally, m_c , the channel source term balancing the flow of water out of the adjacent sheet, is:

$$m_c = \mathbf{q} \cdot \mathbf{n}|_{\partial\Omega_{i1}} + \mathbf{q} \cdot \mathbf{n}|_{\partial\Omega_{i2}}. \quad (4.53)$$

where \mathbf{n} is the normal to the channel edge.

- Channel dissipation of potential energy:

$$\Xi(S, \phi) = \left| Q \frac{\partial \phi}{\partial s} \right| + \left| l_c q_c \frac{\partial \phi}{\partial s} \right|, \quad (4.54)$$

where l_c is the channel width (m), and q_c is the discharge in the sheet (flowing in the direction of the channel; $\text{m}^2 \text{s}^{-1}$), given by:

$$q_c(h, \phi) = -k_s h^{\alpha_s} \underbrace{\left| \frac{\partial \phi}{\partial s} \right|^{\beta_s - 2} \frac{\partial \phi}{\partial s}}_{K_s}, \quad (4.55)$$

with k_s , α_s , and β_s as given above.

- Channel melt/refreeze rate:

$$\Pi(S, \phi) = -c_t c_w \rho_w (Q + f l_c q_c) \frac{\partial \phi - \partial \phi_m}{\partial s}, \quad (4.56)$$

Here, c_t is the Clapeyron slope (K Pa^{-1}), c_w is the specific heat capacity of water ($\text{J kg}^{-1} \text{K}^{-1}$), and f is a switch parameter that accounts for the fact that the channel area cannot be negative, turning off the sheet flow for refreezing as $S \rightarrow 0$, i.e.:

$$f = \begin{cases} 1, & \text{if } S > 0 \text{ or } q_c \partial(\phi - \phi_m) \partial s > 0 \\ 0, & \text{otherwise} \end{cases} \quad (4.57)$$

- Cross-sectional channel area (defined along mesh edges):

$$\frac{\partial S}{\partial t} = \frac{\Xi - \Pi}{\rho_i L} - v_c. \quad (4.58)$$

Boundary conditions

Boundary conditions for the evolution of hydraulic potential ϕ are applied on the domain boundary $\partial\Omega$, as either a prescribed pressure or water flux. The Dirichlet boundary condition is:

$$\phi = \phi_D \quad \text{on} \quad \partial\Omega_D, \quad (4.59)$$

where ϕ_D is a specific potential, and the Neumann boundary condition is:

$$\frac{\partial \phi}{\partial n} = \Phi_N \quad \text{on} \quad \partial\Omega_N, \quad (4.60)$$

corresponding to the specific discharge

$$q_N = -k_s h^{\alpha_s} |\nabla \phi|^{\beta_s - 2} \Phi_N. \quad (4.61)$$

Channels are defined only on element edges and are not allowed to cross the domain boundary, so we do not require flux conditions. Since the evolution equations for h and S do not contain their spatial derivatives, we do not require any boundary conditions for their evolution equations.

Model parameters

The parameters relevant to the GlaDS (hydrologyglads) solution can be displayed by running:

```
>> md.hydrology
```

- `md.hydrology.pressure_melt_coefficient` : Pressure melt coefficient (c_t) [K Pa⁻¹]
- `md.hydrology.sheet_conductivity` : sheet conductivity (k) [m^{7/4} kg^{-1/2}]
- `md.hydrology.cavity_spacing` : cavity spacing (l_r) [m]
- `md.hydrology.bump_height` : typical bump height (h_r) [m]
- `md.hydrology.ischannels` : Do we allow for channels? 1: yes, 0: no
- `md.hydrology.channel_conductivity` : channel conductivity (k_c) [m^{3/2} kg^{-1/2}]
- `md.hydrology.spcphi` : Hydraulic potential Dirichlet constraints [Pa]
- `md.hydrology.neumannflux` : water flux applied along the model boundary (m² s⁻¹)
- `md.hydrology.moulin_input` : moulin input (Q_s) [m³ s⁻¹]
- `md.hydrology.englacial_void_ratio` : englacial void ratio (e_v)
- `md.hydrology.requested_outputs` : additional outputs requested?
- `md.hydrology.melt_flag`: User specified basal melt? 0: no (default), 1: use `md.basalforcings.groundedice_melting_rate`

Running a simulation

To run a transient standalone subglacial hydrology simulation, use the following commands:

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

%Change hydrology class to GlaDS;
md.hydrology = hydrologyglads();

%Set model parameters here;
md = solve(md, 'Transient');
```

4.3.8.3 Hydrology Solution - SHAKTI

Description

SHAKTI (Subglacial Hydrology and Kinetic, Transient Interactions) is a transient subglacial hydrology model that has flexible geometry and treats the entire domain with one set of governing equations, allowing for any drainage configuration to arise, which can include efficient (channelized) and inefficient (distributed) features. [?]

Equations

The SHAKTI model is based upon governing equations that describe conservation of water mass, evolution of the system geometry, basal water flux (approximate momentum equation), and internal melt generation (approximate energy equation).

- Continuity equation (water mass balance):

$$\frac{\partial b}{\partial t} + \frac{\partial b_e}{\partial t} + \nabla \cdot \vec{q} = \frac{\dot{m}}{\rho_w} + i_{e \rightarrow b} \quad (4.62)$$

where b is subglacial gap height, b_e is the volume of water stored englacially per unit area of bed, \vec{q} is basal water flux, \dot{m} is melt rate, and $i_{e \rightarrow b}$ is the input rate of water from the englacial to subglacial system.

- Basal gap dynamics (subglacial geometry):

$$\frac{\partial b}{\partial t} = \frac{\dot{m}}{\rho_i} + \beta u_b - A|p_i - p_w|^{n-1}(p_i - p_w)b \quad (4.63)$$

where b is the subglacial gap height, \dot{m} is melt rate, A is the ice flow law parameter, n is the flow law exponent, p_i is the overburden pressure of ice, p_w is water pressure, β is a dimensionless parameter governing opening by sliding, and u_b is sliding velocity. According to this equation, the subglacial gap height evolves with time by: opening by both melt and sliding over bumps on the bed, and closing due to ice creep.

- Basal water flux (approximate momentum equation):

$$\vec{q} = \frac{-b^3 g}{12\nu(1 + \omega Re)} \nabla h \quad (4.64)$$

where b is subglacial gap height, g is gravitational acceleration, ν is kinematic viscosity of water, ω is a dimensionless parameter controlling the nonlinear transition from laminar to turbulent flow (for turbulent flow, the flux becomes proportional to the square root of the head gradient), Re is the Reynolds number, and h is hydraulic head:

$$h = \frac{p_w}{\rho_w g} + z_b \quad (4.65)$$

Equation (3) is a key piece of our model formulation, in that it allows for a spatially and temporally variable hydraulic transmissivity in the system, and facilitates easeful transition between laminar and turbulent flow regimes. Most existing models prescribe a hydraulic conductivity and assume the flow to be turbulent everywhere. Our momentum equation acts to "unify" different drainage modes in a single model.

- Internal melt generation (energy balance at the bed):

$$\dot{m} = \frac{1}{L}(G + \vec{u}_b \cdot \vec{\tau}_b - \rho_w g \vec{q} \cdot \nabla h - c_t c_w \rho_w \vec{q} \cdot \nabla p_w) \quad (4.66)$$

where L is latent heat of fusion of water, G is geothermal flux, u_b is basal sliding velocity, τ_b is the stress exerted by the bed onto the ice, \vec{q} is basal water flux (discharge), h is hydraulic head, c_t is the pressure melt coefficient (Clapeyron constant), c_w is the heat capacity of water, ρ_w is density of water, and p_w is water pressure. In words, melt is produced through a combination of geothermal flux, frictional heat due to sliding, and heat generated through internal dissipation (where mechanical energy is converted to thermal energy), minus the heat consumed due to changes in water pressure. We note that this form of the energy equation assumes that all heat produced is converted locally to melt and is not advected downstream. We assume that the ice and liquid water are consistently at the pressure melting point temperature. While these assumptions may not be strictly valid under certain real conditions that may have interesting implications, we leave that discussion for future work.

Following Werder et al. (2013), the englacial storage volume is a function of water pressure:

$$b_e = e_v \frac{\rho_w g h - \rho_w g z_b}{\rho_w g} = e_v(h - z_b) \quad (4.67)$$

where e_v is the englacial void ratio (zero for no englacial storage).

Equations (1), (2), (3), and (5) are combined to form a nonlinear partial differential equation (PDE) in terms of hydraulic head, h :

$$\nabla \cdot \left[\frac{-b^3 g}{12\nu(1+\omega Re)} \cdot \nabla h \right] + \frac{\partial e_v(h - z_b)}{\partial t} = \dot{m} \left[\frac{1}{\rho_w} - \frac{1}{\rho_i} \right] + A|p_i - p_w|^{n-1}(p_i - p_w)b - \beta u_b + i_{e \rightarrow b} \quad (4.68)$$

With no englacial storage ($e_v = 0$), Eq. (7) takes the form of an elliptic PDE.

Defining the hydraulic "transmissivity":

$$\vec{K} = \frac{-b^3 g}{12\nu(1+\omega Re)} \quad (4.69)$$

Equation (7) can be written more compactly as:

$$\nabla \cdot (\vec{K} \cdot \nabla h) + \frac{\partial e_v(h - z_b)}{\partial t} = \dot{m} \left(\frac{1}{\rho_w} - \frac{1}{\rho_i} \right) + A|p_i - p_w|^{n-1}(p_i - p_w)b - \beta u_b + i_{e \rightarrow b} \quad (4.70)$$

or simply as:

$$\nabla \cdot (\vec{K} \cdot \nabla h) + \frac{\partial e_v(h - z_b)}{\partial t} = RHS \quad (4.71)$$

where the forcing (RHS) is a function of the relevant dependent variables. Within each time step, this nonlinear PDE is solved using a Picard iteration to establish the head (h) distribution.

Boundary conditions

Boundary conditions can be applied as either prescribed head (Dirichlet) conditions or as flux (Neumann) conditions. We typically apply a Dirichlet boundary condition of atmospheric pressure at the edge of the ice sheet, and Neumann boundary conditions (no flux or prescribed flux, which can be constant or time-varying) on the other boundaries of the subglacial drainage domain.

Model parameters

The parameters relevant to the SHAKTI (hydrologyshakti) solution can be displayed by running:

```
>> md.hydrology
```

- `md.hydrology.head` subglacial hydrology water head (m)
- `md.hydrology.gap_height` height of gap separating ice to bed (m)
- `md.hydrology.bump_spacing` characteristic bedrock bump spacing (m)
- `md.hydrology.bump_height` characteristic bedrock bump height (m)
- `md.hydrology.englacial_input` liquid water input from englacial to subglacial system (m/yr)
- `md.hydrology.moulin_input` liquid water input from moulin (at the vertices) to subglacial system (m^3/s)
- `md.hydrology.reynolds` Reynolds' number
- `md.hydrology.neumannflux` water flux applied along the model boundary (m^2/s)
- `md.hydrology.spchead` water head constraints (NaN means no constraint) (m)
- `md.hydrology.relaxation` under-relaxation coefficient for nonlinear iteration
- `md.hydrology.storage` englacial storage coefficient (void ratio)

Running a simulation

To run a transient stand-alone subglacial hydrology simulation, use the following commands:

```
md.transient = deactivateall(md.transient);  
md.transient.ishydrology = 1;  
  
%Change hydrology class to SHAKTI  
md.hydrology = hydrologyshakti();  
  
%Set model paramters here  
  
md = solve(md, 'Transient');
```

4.3.8.4 Hydrology Solution - Shreve Approximation

Physical basis

This model is the one described in ?. Here we present only the main equations.

Water column

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

$$\frac{\partial w}{\partial t} = S - \nabla \cdot \mathbf{u}_w w \quad (4.72)$$

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:

$$\mathbf{u}_w = \frac{w^2}{12\mu} \nabla \phi \quad (4.73)$$

- 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 [?], which hypothesizes a null effective pressure. Assuming this null effective pressure gives the hydraulic potential gradient as follows:

$$\nabla \phi = \rho_{ice} g \nabla s + (\rho_w - \rho_{ice}) g \nabla h \quad (4.74)$$

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]

Numerical implementation

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

$$\frac{\partial w}{\partial t} + \nabla (\mathfrak{D} \nabla w) = S - \nabla \cdot \mathbf{u}_w w \quad (4.75)$$

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

$$\mathfrak{D} = \frac{h}{2} \begin{pmatrix} |vx| & 0 \\ 0 & |vy| \end{pmatrix} \quad (4.76)$$

Model parameters

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

```
>> md.hydrology
```

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

Running a simulation

To run a simulation, use the following command:

```
>> md = solve(md, 'Hydrology');
```

4.3.9 Damage Evolution

4.3.9.1 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.

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)

Make sure that you are using the `matdamageice` class for `md.materials`. You can do that conversion using:

```
md.materials = matdamageice(md.materials);
```

Inverting for damage directly

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

$$\mu = \frac{(1 - D) B}{2\dot{\varepsilon}_e^n} \quad (4.77)$$

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

The remainder of the inversion procedure is described on the in the ‘Inversions’ section. This was the procedure followed by ? in determining the damage for the Larsen B ice shelf prior to its collapse (see the [ISSM Documentation ‘Publications’ page](#) for a link to the paper).

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 ?:

$$\dot{\varepsilon}_{xx} = \theta \left[\frac{1/2\rho_i (1 - \rho_i/\rho_w) gH - \sigma_b}{(1 - D) B} \right]^n \quad (4.78)$$

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_0 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:

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

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

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

This procedure for calculating damage and backstress was used in ? for the Larsen C ice shelf (see the [ISSM Documentation ‘Publications’ page](#) for a link to the paper).

4.3.9.2 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.

4.3.10 Transient Solution

4.3.10.1 Physical basis

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,

1. thermal solution
2. hydrology solution
3. stress balance solution
4. damage mechanics
5. mass transport solution
6. grounding line migration (and geometry update)
7. Glacial Isostatic Adjustment (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.

4.3.10.2 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 fields 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, ...)

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 (Courant–Friedrichs–Lewy) condition (necessary condition for the stability of certain partial differential equations).

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

```
>> 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)

4.3.10.3 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.smb.mass_balance = [smb smb - 1];
>> md.smb.mass_balance = [md.smb.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.

4.3.10.4 Running a simulation

To run a simulation, use the following command:

```
>> md = solve(md, 'Transient');
```

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

4.3.11 Grounding Lines

4.3.11.1 Physical basis

Hydrostatic equilibrium

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

$$H_f = -\frac{\rho_w}{\rho_i} r, \quad r < 0 \quad (4.79)$$

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 $H = H_f$:

$$\begin{aligned} H &> H_f && \text{ice is grounded} \\ H &= H_f && \text{grounding line position} \\ H &< H_f && \text{ice is floating} \end{aligned} \quad (4.80)$$

Each element of the mesh is either grounded or floating: flotation 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 ? and ? but for a 2D element:

$$C_g = C \frac{A_g}{A} \quad (4.81)$$

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 ?.

Contact mechanics

Grounding line migration can be advantageously based on contact mechanics when solving the stress balance equations with a full-Stokes models [??].

This capability is currently under development.

4.3.11.2 Model parameters

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

```
>> md.groundingline
```

- `md.groundingline.migration`: type of grounding line migration: `'SoftMigration'`, `'AgressiveMigration'`, `'SubelementMigration'`, `'SubelementMigration2'`, or `'None'`

4.3.11.3 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, 'Transient');
```

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

4.3.12 Ice Front Migration (level-set method)

UNDER CONSTRUCTION

4.3.13 Model parameters

```
>> md.levelset
```

- stabilization : ...

4.3.14 Running a simulation

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

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

4.3.15 Glacial Isostatic Adjustment (GIA) Solution

4.3.15.1 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.

4.3.15.2 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 ?:

$$w(r, t) = \int_0^\infty k \left[\frac{4\mu_1^e \alpha}{2k\mu_1^e + \rho_1 g} \hat{Q}_0(k, t) J_1(k\alpha) \right] J_0(kr) dk, \quad (4.82)$$

where:

- r is the radial distance from the center 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 $f(r)$.)

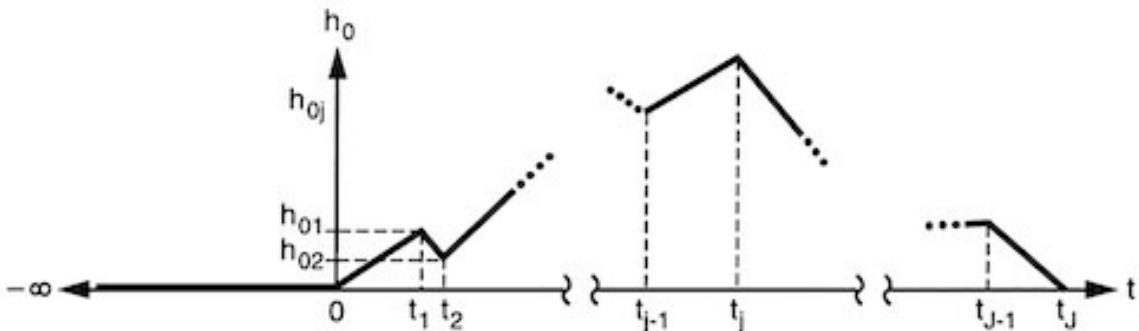


Figure 4.6: Schematic of evolution of piecewise continuous load height, h_0 , with J linear segments (from ?). For j -th segment, we can compute m_j and b_j (cf. Eqs. 3–4) based on the ice load at time t_{j-1} and t_j . At t_j , for example, ice load at the lithosphere surface is given by $\rho_0 g h_{0j}$, 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:

$$\hat{Q}_0(k, t) = \sum_{j=1}^J {}_j\hat{Q}_0(k, t). \quad (4.83)$$

for $j \leq (J - 1)$:

$${}_j\hat{Q}_0(k, t) = \sum_{p=1}^2 \left\{ \frac{m_j \xi_p}{\gamma_p^2} \left[(\gamma_p t_j - 1) e^{\gamma_p(t_j-t)} - (\gamma_p t_{j-1} - 1) e^{\gamma_p(t_{j-1}-t)} \right] + \frac{b_j \xi_p}{\gamma_p} \left[e^{\gamma_p(t_j-t)} - e^{\gamma_p(t_{j-1}-t)} \right] \right\}, \quad (4.84)$$

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

$${}_j\hat{Q}_0(k, t) = \sum_{p=1}^2 \left\{ \frac{m_j \xi_p}{\gamma_p^2} \left[(\gamma_p t - 1) - (\gamma_p t_{j-1} - 1) e^{\gamma_p(t_{j-1}-t)} \right] + \frac{b_j \xi_p}{\gamma_p} \left[1 - e^{\gamma_p(t_{j-1}-t)} \right] \right\} + \left(c_2 + \frac{1}{4k\mu_1^e} \right) (m_j t + b_j), \quad (4.85)$$

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:

$$\gamma_p = \frac{d_1 \pm \sqrt{d_1^2 - 4d_0}}{2}, \quad (4.86)$$

and the amplitude factors by:

$$\xi_p = \frac{(-1)^p}{(\gamma_2 - \gamma_1)} [(-c_2 \gamma_p + c_1) \gamma_p - c_0]. \quad (4.87)$$

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

$$c_0 = \frac{h_1}{\mu_2^e \tau_m^2} c'_0, \quad c_1 = \frac{h_1}{\mu_2^e \tau_m} c'_1, \quad c_2 = \frac{h_1}{\mu_2^e} c'_2, \quad d_0 = \frac{1}{\tau_m^2} d'_0 \text{ and } d_1 = \frac{1}{\tau_m} d'_1, \quad (4.88)$$

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:

- $d'_2 = b'_0 + b'_1 + b'_2 + b'_3 + b'_4 + b'_5 + b'_6 + b'_7$
- $d'_1 = [b'_2 + b'_3 + b'_4 + 2(b'_5 + b'_6 + b'_7)] / d'_2$
- $d'_0 = (b'_5 + b'_6 + b'_7) / d'_2$

- $c'_2 = (a'_0 + a'_1 + a'_2 + a'_3) / d'_2$
- $c'_1 = [a'_1 + 2(a'_2 + a'_3)] / d'_2$
- $c'_0 = (a'_2 + a'_3) / d'_2$

where:

- $a'_0 = -2k' \left\{ 1 + e^{2k'} [1 + 2k' (1 + k')] \right\}$
- $a'_1 = 4k' R_\mu^e - R_\rho^- \left\{ 1 + e^{2k'} [1 + 2k' (1 + k')] \right\}$
- $a'_2 = -2k' (R_\mu^e)^2 \left[1 - e^{2k'} - 2k' e^{2k'} (1 + k') \right]$
- $a'_3 = R_\mu^e R_\rho^- \left[1 - e^{2k'} (1 + 2k') \right]$
- $b'_0 = 4(k')^2 R_\mu^e \left[1 + e^{4k'} + 2 e^{2k'} (1 + 2(k')^2) \right]$
- $b'_1 = -2k' R_\rho^1 \left(1 - e^{4k'} + 4k' e^{2k'} \right)$
- $b'_2 = -8(k')^2 (R_\mu^e)^2 \left(1 - e^{4k'} \right)$
- $b'_3 = 2k' R_\mu^e \left[R_\rho^+ \left(1 + e^{4k'} \right) + 2R_\rho^- e^{2k'} (1 + 2(k')^2) \right]$
- $b'_4 = -R_\rho^1 R_\rho^- \left(1 - e^{4k'} + 4k' e^{2k'} \right)$
- $b'_5 = 4(k')^2 (R_\mu^e)^3 \left[\left(1 - e^{2k'} \right)^2 - 4(k')^2 e^{2k'} \right]$
- $b'_6 = -2k' (R_\mu^e)^2 R_\rho^2 \left(1 - e^{4k'} - 4k' e^{2k'} \right)$
- $b'_7 = R_\mu^e R_\rho^1 R_\rho^- \left(1 - e^{2k'} \right)^2$

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

$$k' = kh_1, R_\mu^e = \frac{\mu_1^e}{\mu_2^e}, R_\rho^1 = \frac{gh_1\rho_1}{\mu_2^e}, R_\rho^2 = \frac{gh_1\rho_2}{\mu_2^e}, R_\rho^+ = \frac{gh_1(\rho_2 + \rho_1)}{\mu_2^e}, R_\rho^- = \frac{gh_1(\rho_2 - \rho_1)}{\mu_2^e}, \quad (4.89)$$

where:

- ρ_2 is the mantle density

4.3.15.3 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 [?]. 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 trational 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.

4.3.15.4 Model parameters

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

```
>> 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/cm³)
- `md.materials.mantle_shear_modulus` : shear modulus of mantle (in Pa)
- `md.materials.mantle_density` : mantle density (in g/cm³)
- `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).

4.3.15.5 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"
```

4.3.15.6 Running a simulation

To run a simulation, use the following command:

```
>> md = solve(md, 'Gia');
```

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

4.3.16 Elastostatic Adjustment Solution

4.3.16.1 Physical basis

Any redistribution of mass at the Earth's surface, such as snow, water, or atmosphere, loads and deforms the underlying solid Earth. At timescales that are comparable to those of the main tidal constituents, such as the near-annual periods, solid Earth deformation is excellently approximated as an elastic response. This module employs the classical Green's function approach to solving for interior Earth responses at the surface, following the so-called load Love number formalism for a radially stratified, seismologically constrained, elastically compressible Earth.

4.3.16.2 3-D crustal motions

Let U_i (for $i = 1, 2, 3$) be the components of the 3-D crustal displacement vector, $\vec{U}(\theta, \phi, t)$, evaluated at geographic coordinates (θ, ϕ) at time t , where U_1 is the vertical displacement (up positive), U_2 is the north-south component of horizontal displacement (north positive), and U_3 is the east-west component of horizontal displacement (east positive).

For a given surface load, $H(\theta, \phi, t)$, with dimensions of ice equivalent height, these displacement components may be computed theoretically as follows:

$$\vec{U}(\theta, \phi, t) = \int \vec{G}(\alpha, \beta) H(\theta', \phi', t) d\mathcal{S}', \quad (4.90)$$

where $\vec{G}(\alpha, \beta)$ is the 3-D Green's function vector that models the influence of a specified point load evaluated at an arc distance α and direction β , from load coordinate position (θ', ϕ') . The integral in the above equation is applied over the surface of a unit sphere \mathcal{S} .

The components of \vec{G} are given by:

$$\begin{Bmatrix} G_1(\alpha, \beta) \\ G_2(\alpha, \beta) \\ G_3(\alpha, \beta) \end{Bmatrix} = \frac{3}{4\pi} \frac{\rho_i}{\rho_e} \sum_{n=0}^{\infty} \begin{Bmatrix} h'_n P_n(\cos \alpha) \\ l'_n \cos \phi dP_n(\cos \alpha)/d\alpha \\ l'_n \sin \phi dP_n(\cos \alpha)/d\alpha \end{Bmatrix}, \quad (4.91)$$

where:

- ρ_i is the ice density
- ρ_e is the Earth's global mean density
- P_n are the Legendre polynomials of degree n
- h'_n and l'_n are the load Love numbers

4.3.16.3 Numerical implementation

We use Love numbers – provided by the International Association of Geodesy (available at <http://www.srosat.com/iag-jsg/loveNb.php>) – which are the solutions of the zero frequency momentum equations with self-gravitation for a spherically symmetric and seismologically constrained Earth structure model [see, e.g., Alterman et al., 1959]. Since h'_n converges slowly toward a constant as $n \rightarrow \infty$, the requirement for generating an accurate solution for crustal deformation is stringent, demanding truncation of the series at high degree $n = 10,000$. See [?] for more details.

4.3.16.4 Model parameters

The parameters relevant to the elastostatic adjustment (ESA) solution can be displayed by running:

```
>> md.esa
```

- `md.esa.deltathickness` : thickness change: ice height equivalent [m]
- `md.solidearth.lovenumbers` : loads required Love numbers for solid Earth deformation
- `md.esa.hemisphere` : North-south, East-west components of 2-D horiz displacement vector: -1 south, 1 north
- `md.esa.degacc` : accuracy (default .01 deg) for numerical discretization of the Green's functions

4.3.16.5 Running a simulation

To run a simulation, use the following command:

```
>> md = solve(md, 'Esa');
```

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

4.3.17 Sea-level Fingerprints Solution

4.3.17.1 Physical basis

This module solves the so-called "sea-level equation" to compute spatial structure of ocean mass redistribution induced by land hydrological and cryospheric changes. Any redistribution of mass at the Earth's surface perturbs Earth's gravitational and rotational potentials; it also induces the solid Earth deformation. At timescales that are comparable to those of the main tidal constituents, such as the near-annual periods, solid Earth deformation is excellently approximated as an elastic response. This module therefore operates on a self gravitating, rotating, elastic Earth.

4.3.17.2 Relative sea-level

Let $L(\theta, \phi, t)$ be a global mass-conserving load function, such that:

$$L(\theta, \phi, t) = \rho_I H(\theta, \phi, t) \mathcal{I}(\theta, \phi) + \rho_O S(\theta, \phi, t) \mathcal{O}(\theta, \phi) \quad (4.92)$$

where H is the change in ice thickness on a (global or regional) land ice mask \mathcal{I} , S is the associated change in sea level with ocean mask \mathcal{O} , (θ, ϕ) represent the geographic coordinates, t is time, ρ_I is the ice density, and ρ_O is the ocean water density. (Note: H may be the (ice height equivalent of) land hydrological changes within hydrological domain \mathcal{I} .)

Mass changes in land ice, along with the associated variations in ocean loading, induce perturbations in the Earth's gravitational and rotational potential fields, causing further redistribution of S , which is both gravitationally and deformationally self-consistent. For an elastically compressible rotating Earth, the gravitationally consistent S is given by:

$$S(\theta, \phi, t) = \frac{R}{M} [\mathcal{G}(\alpha) \otimes L(\theta', \phi', t)] + \frac{1}{g} \sum_{m=0}^2 \sum_{i=1}^2 \Lambda_{2mi}(t) \mathcal{Y}_{2mi}(\theta, \phi) + \mathcal{E}(t) \quad (4.93)$$

where \mathcal{G} is a Green's function that models the influence of a specific point load on relative sea-level evaluated at arc distance α from the load coordinate position (θ', ϕ') , Λ_{2mi} are related to perturbations in rotational potential and associated solid Earth deformation induced by the applied loading, \mathcal{Y}_{2mi} are analytic (degree-2, order- m spherical harmonic) functions (i 's represent the cosine and sine terms), and \mathcal{E} is a spatial invariant required to conserve the mass. Parameters R , M , and g represent Earth's global mean radius, mass, and gravitational acceleration, respectively. The operator \otimes implies the spatial convolution on the surface of Earth.

4.3.17.3 Numerical implementation

Solving the second equation above for S requires a priori knowledge of S itself (see the first equation above), and we therefore solve the system of equations iteratively, as in the original study of Farrell and Clark (1972). All of our calculations were based on a novel mesh-based approach [?], which, unlike contemporary pseudo-spectral methods, remained numerically accurate and computationally efficient as the resolution requirements approached those of contemporary ice sheets or ocean models (on the order of a few kilometers). For more details on this approach, including validation against other existing methods relying on spherical harmonics, we refer the reader to ?.

4.3.17.4 Model parameters

The parameters relevant to the sea-level fingerprints (SLR) solution can be displayed by running:

```
>> md.slrf
```

- `md.slr.deltathickness` : thickness change: ice height equivalent [m]
- `md.slr.sealevel` : current sea level (prior to computation) [m]
- `md.slr.reltol` : sea level rise relative convergence criterion
- `md.slr.maxiter` : maximum number of nonlinear iterations
- `md.slr.love_h` : load Love number for radial (vertical) displacement
- `md.slr.love_l` : load Love number for horizontal displacement
- `md.slr.love_k` : load Love number for gravitational potential perturbation
- `md.slr.rigid` : flag for rigid earth gravitational potential perturbation
- `md.slr.elastic` : flag for elastic earth gravitational potential perturbation
- `md.slr.rotation` : flag for earth rotational potential perturbation
- `md.slr.ocean_area_scaling` : correction for model representation of ocean area [default: No correction]
- `md.slr.steric_rate` : rate of steric ocean expansion [in mm/yr]

4.3.17.5 Running a simulation

To run a simulation, use the following command:

```
>> md = solve(md, 'Slr');
```

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

4.3.18 Verbosity

The verbosity level is set by the field `md.VERBOSE`. This field is a MATLAB object, which activates and deactivates different verbosity levels. By default, all verbosity levels are deactivated.

For a complete list of available levels,

```
>> help verbose
VERBOSE class definition

Available verbosity levels:
mprocessor : model processing
module    : modules
solution   : solution sequence
solver     : solver info (extensive)
convergence: convergence criteria
control    : control method
qmu        : sensitivity analysis
```

To activate the levels `module` and `solution`, use,

```
>> md.VERBOSE = verbose();
>> md.VERBOSE = verbose('module', true, 'solution', true);
```

4.4 Parameterization of Physical Processes

- Parameter Files
- Positive Degree Day (PDD)
- Surface Mass Balance (SMB)
- Basal Friction
- Calving (UNDER DEVELOPMENT)
- Basal Melt
- Empirical Scalar Tertiary Anisotropy Regime (ESTAR)

4.4.1 Parameter Files

To run a simulation, the solution sequence needs many parameters: physical constants, number of iterations, relaxation constant, thickness and surface of the glacier, etc. All of this can be done during model setup in your `runme`. But for the sake of organization and/or reusability, you might want to store the parameterization commands in a separate file. For example, for the MATLAB interface, you might create a file `Parameters.par` with contents,

```
%%%%%%%%
%%%%% GEOMETRY
%%%%%

disp('      reading thicknesses');
md.geometry.thickness = InterpFromFile(md.mesh.x, md.mesh.y,
    thicknesspath, 10);

disp('      reading dem');
md.geometry.surface = InterpFromFile(md.mesh.x, md.mesh.y,
    surfacepath, 10);

%get base
md.geometry.base = md.geometry.surface - md.geometry.thickness;

%%%%% OBSERVATIONS
%%%%%

disp('      reading velocities');
md = plugvelocities(md, velocitypath, 0);

disp('      loading temperature');
md.initialization.temperature = InterpFromFile(md.mesh.x, md.mesh.y,
    temperaturepath, 253);

disp('      creating mass balance rates');
md.smb.mass_balance = InterpFromFile(md.x, md.y, massbalancepath, 1);

disp('      loading geothermal flux');
load(heatfluxpath);
md.basalforcings.geothermalflux = InterpFromGrid(x_m, y_m, heatflux,
    md.mesh.x, md.mesh.y, 80);

%%%%% MATERIAL
%%%%%

%flow law
disp('      creating flow law parameters');
md.materials.rheology_n = 3 * ones(md.mesh.numberofelements, 1);
md.materials.rheology_B = paterson(md.initialization.temperature);

%%%%% BOUNDARY CONDITIONS
%%%%%

%drag md.drag or stress
md.friction.coefficient = 300 * ones(md.mesh.numberofvertices, 1);
%q=1.

%floating ice: no drag
```

```
md.friction.coefficient(find(md.mask.ocean_levelset < 0.)) = 0.;  
md.friction.p = ones(md.mesh.numberofelements, 1);  
md.friction.q = ones(md.mesh.numberofelements, 1);  
  
%Create ice front  
md = SetMarineIceSheetBC(md);
```

As you can see, even though the file extension is `.par`, it is really just a MATLAB script. You can now parameterize your model in your `runme` with,

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

For the Python interface, we similarly use Pickle files (`.pkl`) and parameterize with,

```
>>> md = parameterize(md, 'Parameters.pkl')
```

NOTE:

- Parameterization must be done on a two dimensional mesh.
- Parameters will be automatically extruded if the mesh is extruded.

4.4.2 Positive Degree Day (PDD)

4.4.2.1 Physical basis

Positive degree day method

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

$$\text{PDD} = \frac{1}{\sigma_{PDD}\sqrt{2\pi}} \int_0^{1\text{year}} \int_{0^\circ\text{C}}^{T_m + 2.5\sigma_{PDD}} T \exp\left[\frac{-(T - T_m)^2}{2\sigma_{PDD}^2}\right] dT dt \quad (4.94)$$

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}):

$$\begin{aligned} \gamma_{ice} &= \begin{cases} 17.22 \text{ mm/PDD} & T_{jja} \leq -1^\circ\text{C}, \\ 0.0067 \times (10 - T_{jja})^3 + 8.3 \text{ mm/PDD} & -1^\circ\text{C} < T_{jja} < 10^\circ\text{C}, \\ 8.3 \text{ mm/PDD} & 10^\circ\text{C} \leq T_{jja} \end{cases} \\ \text{and} \\ \gamma_{snow} &= \begin{cases} 2.65 \text{ mm/PDD} & T_{jja} \leq -1^\circ\text{C}, \\ 0.15 \times T_{jja} + 2.8 \text{ mm/PDD} & -1^\circ\text{C} < T_{jja} < 10^\circ\text{C}, \\ 4.3 \text{ mm/PDD} & 10^\circ\text{C} \leq T_{jja} \end{cases} \end{aligned} \quad (4.95)$$

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

$$\text{superimposed ice} = \begin{cases} \min[Pr + M, 2.2 \times (Ps - M) - d \times ci / L \times \min(Tsurf, 0^\circ\text{C})] & M < Ps, \\ \min[Pr + M, d \times ci / L \times \min(Tsurf, 0^\circ\text{C})] & M > Ps \end{cases} \quad (4.96)$$

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 ($152.5 + 7.122T Jkg^{-1}K^{-1}$)
- 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°C .

$$\frac{\text{accumulation}}{\text{precipitation}} = \frac{\rho_i}{\rho_w \sigma_{RS} \sqrt{2\pi}} \int_0^{1\text{year}} \int_{T_m - 2.5\sigma_{RS}}^{0^\circ\text{C}} \exp\left[\frac{-(T - T_m)^2}{2\sigma_{RS}^2}\right] dT dt \quad (4.97)$$

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 $6.5^{\circ}/\text{km}$ is used to adjust them to the surface elevation of each step.

4.4.2.2 Model parameters

The parameters relevant to the positive degree day and $\delta^{18}\text{O}$ 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:

$$rtlaps = TdiffTime * rlapslgm + (1. - TdiffTime) * rlaps \quad (4.98)$$

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:

$$tstar = tdiffh + TemperaturesPresentday[imonth] - rlaps \times \max(st, sealev) \times 0.001; \quad (4.99)$$

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

$$st = (s - s0t)/1000 \quad (4.100)$$

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

$$tdiffh = TdiffTime \times (TemperaturesLgm[imonth] - TemperaturesPresentday[imonth]) \quad (4.101)$$

```
>> md.smb
```

- `isdelta18o` : is temperature and precipitation delta18o parameterization 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

4.4.2.3 Running a simulation

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

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

4.4.3 Surface Mass Balance (SMB)

4.4.3.1 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 running:

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

One can display the following fields by running:

```
>> md.smb
```

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

4.4.3.2 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 running:

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

One can display the following fields by running:

```
>> md.smb
```

surface forcings parameters (SMB = accumulation - runoff - evaporation):

- `md.smb.accumulation` : accumulated snow [m/yr ice eq]
- `md.smb.runoff` : amount of ice melt lost from the ice column [m/yr ice eq]
- `md.smb.evaporation` : amount of ice lost to evaporative processes [m/yr ice eq]

4.4.3.3 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 running:

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

```
>> md.smb
```

surface forcings parameters with melt (SMB = accumulation - evaporation - melt + refreeze)

- `md.smb.accumulation` : accumulated snow [m/yr ice eq]
- `md.smb.evaporation` : amount of ice lost to evaporative processes [m/yr ice eq]
- `md.smb.melt` : amount of ice melt in ice column [m/yr ice eq]
- `md.smb.refreeze` : amount of ice melt refrozen in ice column [m/yr ice eq]

4.4.3.4 SMB gradients method

This surface mass balance model is based on the mass balance gradients method described in ?. 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 predetermined slope of the linear regression between positive surface mass balance and ice surface height; and (4) a predetermined 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 running:

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

```
>> md.smb
```

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

4.4.4 Basal Friction

4.4.4.1 Introduction

All friction laws in ISSM are implemented as:

$$\tau_b = -f(\mathbf{v}_b, N) \frac{\mathbf{v}_b}{|\mathbf{v}_b|} \quad (4.102)$$

where N is the effective pressure, τ_b and \mathbf{v}_b are the basal stress and sliding velocities respectively. The friction laws described below are describing the norm of the basal stress for simplicity but all implementations are such that the oppose motion (i.e. the direction of the basal stress is the opposite of \mathbf{v}_b).

Most friction laws use a switch to define how the effective pressure, $N = p_{ice} - p_{water}$ is calculated:
`md.friction.coupling`:

- 0: $p_{water} = -\rho_w g b$ uniform sheet (negative water pressure ok, default)
- 1: $p_{water} = 0$, so that $N = p_{ice} = \rho_i g H$ is equal to the overburden pressure
- 2: $p_{water} = \max(0, -\rho_w g b)$. Same as 0, but $p_{water} \geq 0$
- 3: Use effective pressure prescribed in `md.friction.effective_pressure`
- 4: Use effective pressure dynamically calculated by the hydrology model (i.e., fully coupled)

4.4.4.2 Budd Friction law (friction)

The default friction law is defined as [?] (p 151):

$$v_b \propto N^{-q} \tau_b^p \quad (4.103)$$

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, following ?:

$$\tau_b = C_b^2 N^r v_b^s \quad (4.104)$$

where:

- C_b friction coefficient
- r and s are friction law exponents:

$$r = q/p \quad s = 1/p \quad (4.105)$$

This friction law can be selected as follows:

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

The following fields need to be specified:

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

4.4.4.3 Weertman Friction law (`weertmanfriction`)

The Weertman friction [?] law reads:

$$v_b = C_w \tau_b^m \quad (4.106)$$

- C_w 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:

$$\tau_b = C_w^{-1/m} \| \mathbf{v}_b \|^{1/m-1} \mathbf{v}_b \quad (4.107)$$

This friction law can be selected as follows:

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

One can display the following fields by running:

```
>> md.friction
```

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

4.4.4.4 Coulomb-limited sliding 1 (`frictioncoulomb`)

$$\tau_b = \min(CNub, C_c^2 N) \quad (4.108)$$

4.4.4.5 Regularized Coulomb-limited sliding 1 (`frictionregcoulomb`)

Sliding law from ?:

$$\tau_b = \frac{Cu_b^{1/m}\alpha^2 N}{\left(\frac{u_b}{u_0} + 1\right)^{1/m}} \quad (4.109)$$

4.4.4.6 Coulomb-limited sliding 2 (frictioncoulomb2)

Coulomb-limited sliding law used in MISMIP+ [?]:

$$\tau_b = \frac{Cu_b^m \alpha^2 N}{(C^{1/m} u_b + (\alpha^2 N)^{1/m})^m}, \quad (4.110)$$

where $\alpha^2 = 0.5$. Note that this friction law is exactly the same as `frictionschoof` described below, with $C_{max} = 0.5$.

4.4.4.7 Regularized Coulomb-limited sliding 2 (frictionregcoulomb2)

Sliding law from ?:

$$\tau_b = \frac{C N u_b^{1/m}}{(u_b + (K N)^m)^{1/m}} \quad (4.111)$$

4.4.4.8 Friction Tsai (frictiontsai)

from [?]:

$$\tau_b = \min(Cub^m, fN) \quad (4.112)$$

4.4.4.9 Friction Schoof (frictionschoof)

from [??] (note that we use C_s^2 to make sure it is a positive number):

$$\tau_b = \frac{C_s^2 v_b^m}{\left(1 + \left(\frac{C_s^2}{C_{max} N}\right)^{1/m} v_b\right)^m}, \quad (4.113)$$

4.4.4.10 Friction PISM (frictionpism)

Under construction

4.4.4.11 Thin water layer friction law (frictionwaterlayer)

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:

$$N = g(\rho_i H + \rho_w(b - w)) \quad (4.114)$$

when the bedrock is below sea level, and:

$$N = g(\rho_i H - \rho_w w) \quad (4.115)$$

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 running:

```
>> 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)

4.4.5 Calving

UNDER DEVELOPMENT

4.4.5.1 Model parameters

```
>> md.levelset
```

- stabilization : ...

4.4.5.2 Running a simulation

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

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

4.4.6 Basal Melt

4.4.6.1 Physical basis

This model is described in ? and ?. It consists in calculating basal melt rates under ice shelves based only on far field ocean temperature and salinity.

PICO

PICO is a box model of ocean circulation under ice shelf cavities. each ice shelf is divided in a set of boxes, and the temperature (T_k) and salinity (S_k) of each box is given by:

$$\begin{aligned} q(T_{k-1} - T_k) - A_k m_k \frac{\rho_i}{\rho_w} \frac{L}{c_p} &= 0 \\ q(S_{k-1} - S_k) - A_k m_k S_k &= 0 \end{aligned} \quad (4.116)$$

where:

- A_k is the surface area of box k
- m_k is the melt rate in box k
- $q = C(\rho_0 - \rho_1)$ is the strength of the overturning circulation

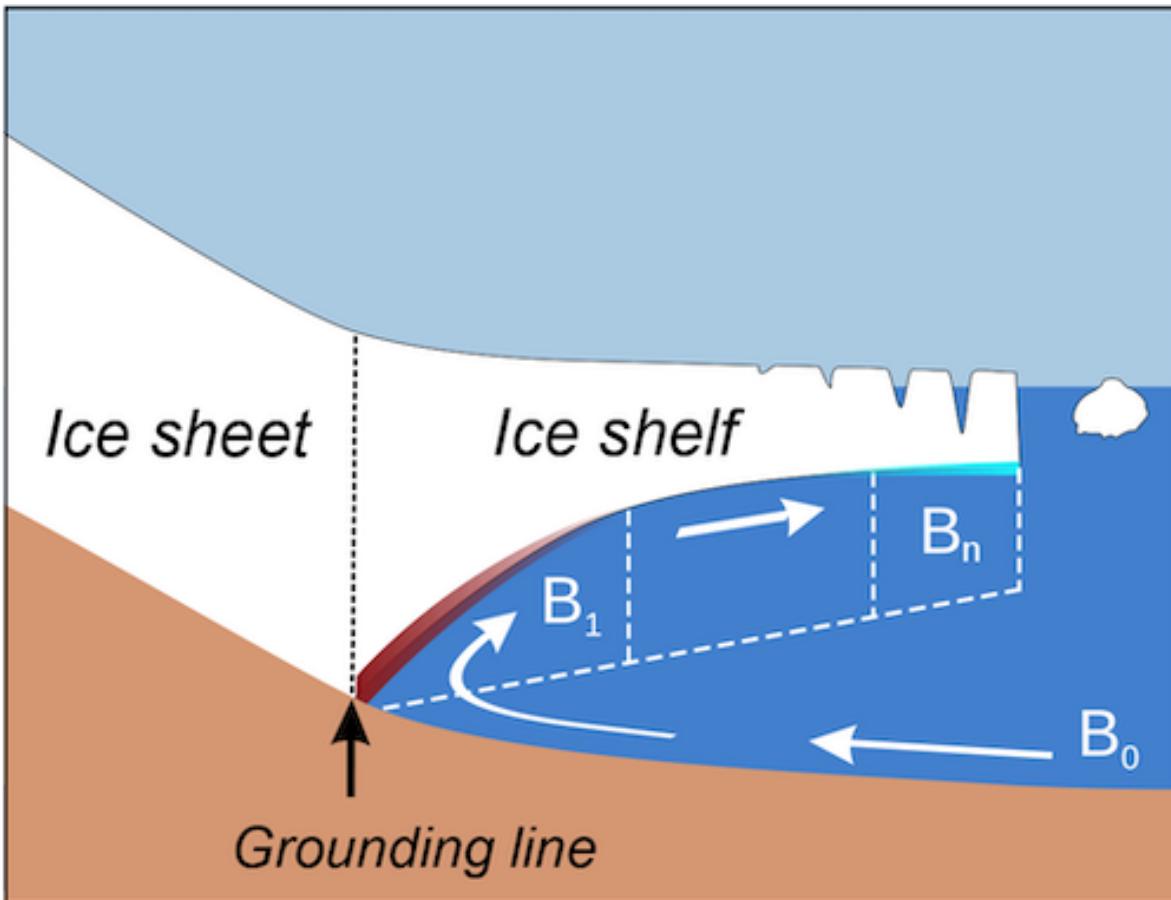


Figure 4.7: Schematic view of the PICO model (taken from ?).

PICOP

PICOP is described in ?. The idea is to use PICO to calculate the temperature and salinity in each box, but instead of using PICO's calculated melt, use these quantities to drive a plume model from ?:

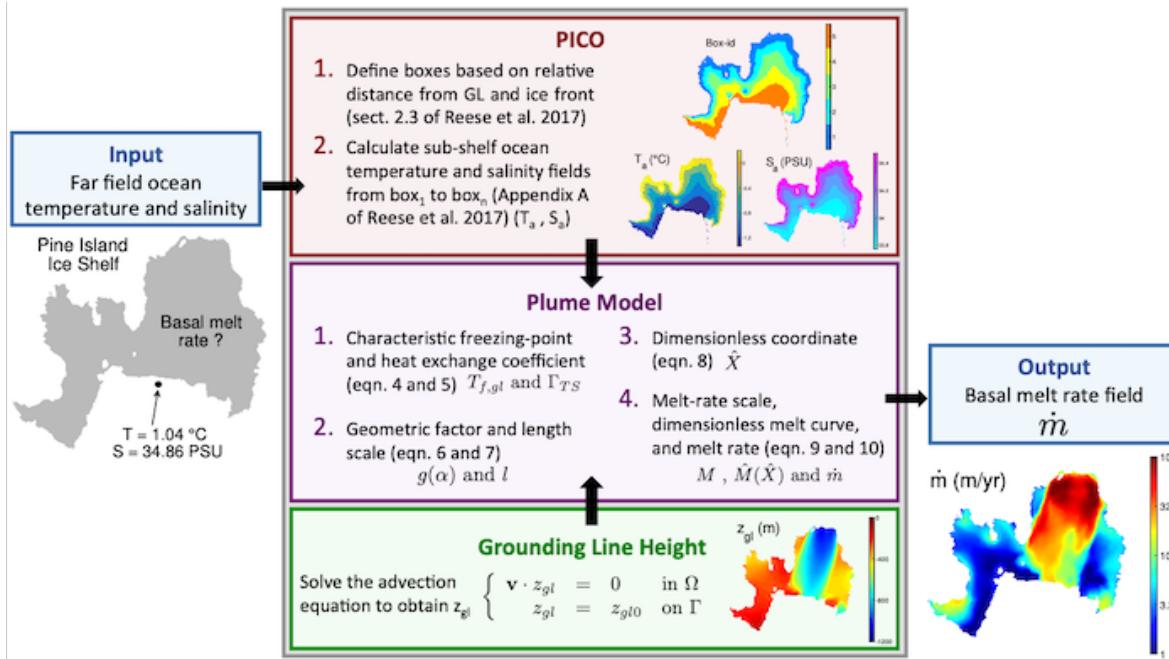


Figure 4.8: Melt calculation in PICOP, adapted from ?.

4.4.6.2 Model parameters

To activate this melt parameterization, you need to use the class `basalforcingspico`:

```
>> md.basalforcings = basalforcingspico();
```

The parameters relevant to the calculation can be displayed by running:

```
>> md.basalforcings
```

- `md.basalforcings.num_basins` : number of basins the model domain is partitioned into [unitless]
- `md.basalforcings.basin_id` : basin number assigned to each node [unitless]
- `md.basalforcings.maxboxcount` : maximum number of boxes initialized under all ice shelves
- `md.basalforcings.overturning_coeff` : overturning strength [m^3/s]
- `md.basalforcings.gamma_T` : turbulent temperature exchange velocity [m/s]
- `md.basalforcings.faroecean_temperature` : depth averaged ocean temperature in front of the ice shelf for basin i [K]

- `md.basalforcings.farocean_salinity` : depth averaged ocean salinity in front of the ice shelf for basin i [psu]
- `md.basalforcings.isplume` : boolean to use buoyant plume melt rate parameterization from Lazeroms et al., 2018 (PICOP, default false)

4.4.6.3 Example: the Amundsen sea

To set up a model of the Amundsen sea using PICOP, we only need one basin:

```
>> md.basalforcings = basalforcingspico();
>> md.basalforcings.basin_id = ones(md.mesh.numberofelements, 1);
>> md.basalforcings.num_basins = 1;
```

We generally do not need to have more than 5 boxes per ice shelf:

```
>> md.basalforcings.maxboxcount = 5;
```

and finally, we can prescribe the far field ocean properties (they can be time series):

```
>> md.basalforcings.farocean_temperature = [0.47 + 273.15]; %0.47C
    converted to K
>> md.basalforcings.farocean_salinity = [34.73]; %PSU
```

To activate PICOP instead of PICO:

```
>> md.basalforcings.isplume = 1;
```

To run a simulation, use the following command:

```
>> md = solve(md, 'Transient');
```

4.4.7 Empirical Scalar Tertiary Anisotropy Regime (ESTAR)

4.4.7.1 Description

The ESTAR (Empirical Scalar Tertiary Anisotropy Regime) flow relation [??] is a generalized constitutive relation for polycrystalline ice in steady-state (tertiary) flow. It is a scalar power law formulation based on tertiary creep rates from laboratory experiments of ice deformation under a variety of simple shear and compression stresses. While mathematically isotropic, the ESTAR flow relation describes the deformation of ice with a flow-compatible induced anisotropy – i.e. ice that has a developed anisotropic fabric that is a function of the underlying stress regime (i.e. the relative proportion of simple shear and compression stresses). The origins of ESTAR, including the laboratory experiments than contributed to its development, its derivation, and underlying assumptions are discussed in ? and ?.

Equations

Ice is treated as a purely viscous incompressible material [?], such that its material constitutive relation can be written:

$$\sigma' = 2\mu\dot{\varepsilon}, \quad (4.117)$$

where:

- σ' is the deviatoric stress tensor (Pa)
- μ is the ice effective viscosity (Pa s)
- $\dot{\varepsilon}$ is the strain rate tensor (s^{-1})

The ESTAR flow relation viscosity μ can be written:

$$\mu = \frac{B}{2E(\lambda_S)^{\frac{1}{3}}\dot{\varepsilon}_e^{\frac{2}{3}}}, \quad (4.118)$$

where:

- B is the ice hardness or rigidity. Note that $B = A(T')^{-1/3}$, where $A(T')$ is the temperature-dependent flow rate parameter and T' is the temperature relative to the pressure dependent melting point of ice.
- $E(\lambda_S)$ is an enhancement factor that characterizes the relative proportion of simple shear and compression stresses via the shear fraction λ_S

The most notable difference between the Glen and ESTAR flow relations is realized in the form of the enhancement factor, which for the ESTAR flow relation is $E(\lambda_S)$, given by:

$$E(\lambda_S) = E_C + (E_S - E_C)\lambda_S^2. \quad (4.119)$$

Here, E_C and E_S are the enhancement factors above the minimum (secondary) deformation rate for isotropic ice under compression alone or simple shear alone, respectively. Laboratory evidence suggests that a suitable ratio of E_C/E_S is 3/8 [?]. The shear fraction λ_S characterizes the contribution of simple shear to the effective stress. The collinear nature of the ESTAR flow relation allows λ_S to be expressed equivalently in terms of stresses and strain rates. The strain rate formulation is more convenient for Stokes flow modeling, and can be written:

$$\lambda_S = \frac{\dot{\varepsilon}'}{\dot{\varepsilon}_e}, \quad (4.120)$$

where $\dot{\varepsilon}'$ (s^{-1}) is the magnitude of the shear strain rate on the local non-rotating shear plane. The local non-rotating shear plane contains the velocity vector and the vorticity vector associated solely with deformation, rather than local rigid body rotation. See ? for details.

For comparison with the ESTAR viscosity, the Glen flow relation viscosity μ can be written:

$$\mu = \frac{B}{2E^{\frac{1}{n}}\dot{\varepsilon}_e^{\frac{n-1}{n}}}, \quad (4.121)$$

where E is a constant enhancement factor. For the standard Glen flow relation (the `matrice` class in ISSM), $E = 1$; to specify values of $E > 1$, the `matenhancedice` class can be used.

4.4.7.2 Model parameters

The parameters relevant to the ESTAR flow relation (the `matestar` class in ISSM) can be displayed by running:

```
>> md.materials
```

- `md.materials.rheology_B` : temperature-dependent flow relation parameter (`NaN` means no constraint)
- `md.materials.rheology_Ec` : compression enhancement factor
- `md.materials.rheology_Es` : simple shear enhancement factor
- `md.materials.rheology_law` : law for the temperature dependence of the rheology (`None` means no temperature dependence; default is `Paterson`)

4.4.7.3 Using the ESTAR flow relation

The ESTAR flow relation may be specified by:

```
>> md.materials = matestar();
```

In this case, values for B , E_C , and E_S should be explicitly set.

Alternatively, the ESTAR flow relation may be specified from conversion of a Glen type relation by the following:

```
>> md.materials = matestar(md.materials);
```

The argument is the materials class of the model. This will set the same value for B as for the Glen flow model default, with $E_S = 1$ and $E_C = 1$.

4.4.7.4 Using the enhanced Glen flow relation

It is possible to use an alternative Glen flow relation with an explicit enhancement factor, in a similar way to the ESTAR class, as follows:

```
>> md.materials = matenhancedice();
```

in which B and E should be explicitly set, or as:

```
>> md.materials = matenhancedice(md.materials);
```

in which B is inherited from the default Glen flow model and $E=1$.

4.5 Advanced Features

- Inversions
- Rifts
- Adaptive Mesh Refinement (AMR)
- Quantifications of Margins and Uncertainties (QMU) with Dakota
- Stochastic Forcing with StISSM

4.5.1 Inversions

4.5.1.1 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 :

$$\tau_b = -k^2 N^r \|\mathbf{v}\|^{s-1} \mathbf{v}_b \quad (4.122)$$

and/or the depth-averaged ice hardness, B , in Glen's flow law:

$$\mu = \frac{B}{2 \left(\dot{\varepsilon}_e^{1-\frac{1}{n}} \right)} \quad (4.123)$$

This section explains how to launch an inverse method and how optimization parameters must be tuned.

4.5.1.2 Cost functions

Absolute misfit

This is the classic way of calculating a misfit between a modeled and observed velocity field:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2} \left((v_x - v_x^{\text{obs}})^2 + (v_y - v_y^{\text{obs}})^2 \right) dS \quad (4.124)$$

where:

- v_x is the x component of the glacier modeled velocity
- v_y is the y component of the glacier modeled velocity
- v_x^{obs} is the x component of the glacier observed velocity
- v_y^{obs} is the y component of the glacier observed velocity

Relative misfit

The relative misfit is defined as follows:

$$\mathcal{J}(\mathbf{v}) = \int_S \frac{1}{2} \left(\frac{(v_x - v_x^{\text{obs}})^2}{(v_x^{\text{obs}} + \varepsilon)^2} + \frac{(v_y - v_y^{\text{obs}})^2}{(v_y^{\text{obs}} + \varepsilon)^2} \right) dS \quad (4.125)$$

where:

- ε is a minimum velocity used to avoid the observed velocity being equal to zero.

Logarithmic misfit

$$\mathcal{J}(\mathbf{v}) = \int_S \left(\log \left(\frac{\|\mathbf{v}\| + \varepsilon}{\|\mathbf{v}^{\text{obs}}\| + \varepsilon} \right) \right)^2 dS \quad (4.126)$$

where:

- \mathbf{v} is the glacier modeled velocity magnitude
- \mathbf{v}^{obs} is the glacier observed velocity magnitude
- ε is a minimum velocity used to avoid the observed velocity being equal to zero

Thickness misfit

$$\mathcal{J}(H) = \int_{\Omega} \frac{1}{2} (H - H^{\text{obs}})^2 d\Omega \quad (4.127)$$

where:

- H is the ice thickness
- H^{obs} is the measured ice thickness

Drag gradient

$$\mathcal{J}(k) = \int_B \gamma \frac{1}{2} \|\nabla k\|^2 dB \quad (4.128)$$

where:

- γ is a Tikhonov regularization parameter

Thickness gradient

$$\mathcal{J}(k) = \int_{\Omega} \gamma \frac{1}{2} \|\nabla H\|^2 d\Omega \quad (4.129)$$

where:

- γ is a Tikhonov regularization parameter

4.5.1.3 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 are 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

4.5.1.4 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 = tao_inversion()`)
- M1QN3 algorithm (`md.inversion = m1qn3inversion()`)

Each minimizer has its own optimization parameters described below.

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

$$\alpha \in [0, \text{gradient_scaling}] \quad p^{\text{new}} = p^{\text{old}} - \alpha \nabla_p \mathcal{J} / \|\nabla_p \mathcal{J}\| \quad (4.130)$$

Toolkit for Advanced Optimization (TAO)

ISSM has an interface to the Toolkit for Advanced Optimization (TAO) [?]. 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`: imization 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:

$$\begin{aligned}
 f(X) - f(X^*) &< \epsilon_{f\text{atol}} \\
 |f(X) - f(X^*)| / |f(X^*)| &< \epsilon_{f\text{rtol}} \\
 \|g(X)\| &< \epsilon_{g\text{atol}} \\
 \|g(X)\| / |f(X)| &< \epsilon_{g\text{rtol}} \\
 \|g(X)\| / \|g(X_0)\| &< \epsilon_{g\text{ttol}}
 \end{aligned} \tag{4.131}$$

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

M1QN3

ISSM has an interface to M1QN3 (Inria) [?]. This interface was largely based on ?. 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 each other (sup-norm) are considered identical
- `md.inversion.gttol` : gradient relative convergence criterion 2 (defined below)

4.5.1.5 Running an inversion

To launch an inversion, run a stress balance solution with `md.inversion.iscontrol = 1`:

```
>> md = solve(md, 'Stressbalance');
```

4.5.2 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.

4.5.2.1 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.

4.5.2.2 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.

4.5.2.3 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 = 'Water'; %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.

4.5.2.4 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, 'Stressbalance');
```

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.

4.5.2.5 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.

4.5.2.6 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');
```

4.5.2.7 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.

4.5.3 Adaptive Mesh Refinement - AMR

The adaptive mesh refinement (AMR) in ISSM relies on two independent meshers: BAMG and NeoPZ. BAMG is a bidimensional anisotropic mesh generator developed by [Frederic Hecht](#) [?] and NeoPZ is a finite element package developed by Philippe Devloo [Philippe Devloo](#) [?].

The current AMR is supported for 2D meshes (triangle elements) and for the SSA flow equations. The features of each one of these meshers are described below:

4.5.3.1 AMR using BAMG (default)

BAMG is the default mesher to run a simulation with AMR. AMR is executed specifying the required resolutions at the vertices of the mesh. The following properties can be defined by the user:

hmin/hmax

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

```
>> md.amr.hmin = 500;
>> md.amr.hmax = 5000;
```

field/err

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

```
>> md.amr.fieldname = 'Vel';
>> md.amr.err = 3;
```

gradation

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

```
>> md.amr.gradation = 1.5;
```

resolution at the grounding line

One can specify the edge length around the grounding line. The user needs to specify the distance around the grounding line (the same distance is used upstream and downstream of the grounding line) where the imposed resolution will be applied:

```
>> md.amr.groundingline_resolution = 500;
>> md.amr.groundingline_distance = 10000;
```

Set `0` in the grounding distance if this refinement is not required.

resolution at the ice front

The ice front is another region where AMR can be applied. For this, the edge length around the ice front should be specified. As for the grounding line, the user needs to specify the distance around the ice front (the same distance is used upstream and downstream to the ice front) where the imposed resolution will be applied.

```
>> md.amr.icefront_resolution = 500;
>> md.amr.icefront_distance = 10000;
```

Set 0 in the ice front distance if this refinement is not required.

Note: users using Intel compilers (`icc`, `icpc`) should use the flag `-fp-model precise` to disable optimizations that are not value-safe on floating-point data. This will prevent `bamg` from being compiler dependent (see [here](#)).

4.5.3.2 AMR using NeoPZ (requires installation)

The mesh refinement with NeoPZ is based on levels of refinement: the initial coarse mesh is refined according to the user requirement and only nested meshes are generated (it means that the initial vertices positions are kept unchanged during all the AMR simulation). NeoPZ is an external package that needs to be installed before using in ISSM. Once installed, it is necessary setting NeoPZ as the AMR package:

```
>> md.amr = amrneopz();
```

level max

Users should define the maximum level of refinement to be applied:

```
>> md.amr.level_max = 2;
```

gradation

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

```
>> md.amr.gradation = 1.5;
```

distance to the grounding line

User needs to specify the distance around the grounding line (the same distance is used upstream and downstream to the grounding line) where the elements will be refined according to the maximum level of refinement:

```
>> md.amr.groundingline_distance = 10000;
```

Set 0 as the grounding distance if this refinement is not required.

distance to the ice front

If the user wants to refine around the ice front, it is necessary to specify the distance in which the elements will be refined according to the maximum level of refinement (the same distance is used upstream and downstream to the ice front):

```
>> md.amr.icefront_distance = 10000;
```

Set 0 in the ice front distance if this refinement is not required.

Running with AMR

To ability the AMR process, one needs to define the AMR frequency in the transient field (can be 1 or larger depending on how often the mesh needs to be updated):

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

4.5.4 Quantifications of Margins and Uncertainties (QMU) with Dakota

4.5.4.1 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 [?], which is embedded within ISSM [??]. 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.

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 (unweighted 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 [?], prior to setting up the model parameters for QMU analysis.

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 ?, we have:

$$r = r(x_1, x_2, \dots, x_n) \quad (4.132)$$

where the sensitivities are defined as:

$$\theta_i = \frac{\delta r}{\delta x_i} \quad (4.133)$$

If each of the variables is independent, the error propagation equation defines the variance of r as:

$$\sigma_r^2 = \sum_{i=1}^n \theta_i^2 \sigma_i^2 \quad (4.134)$$

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 σ_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

Sampling

Sampling analysis quantifies how input errors propagate through a model to impact a specified diagnostic, r . It is 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 [?]).

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, \dots, 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, \dots, x_n

4.5.4.2 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

4.5.4.3 Building the Chaco and Dakota packages

In order to run Dakota with ISSM, you must compile and install the external package Dakota (`${ISSM_DIR}/externalpackages/dakota`). In addition, for complex partitioning (more than one vertex per partition), you must compile and install the external package Chaco (`${ISSM_DIR}/externalpackages/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 (`${ISSM_DIR}/externalpackages/boost`). If installed, it should also be added to your configure script:

```
--with-boost-dir=${ISSM_DIR}/externalpackages/boost/install/ \
```

4.5.4.4 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');
```

4.5.4.5 Setting up the QMU

For sensitivity

```
>> md.qmu.method = dakota_method('nond_1');
```

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
```

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
```

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;
```

4.5.4.6 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
```

4.5.4.7 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
```

4.5.4.8 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, 'Masstransport');
```

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

4.5.5 Stochastic Forcing with StISSM

4.5.5.1 Introduction

The stochastic component of ISSM (StISSM) allows the user to include random time-dependent fluctuations in a range of ice sheet processes. When activated for a given variable or model parameter (or ‘field’), stochastic perturbations are applied to this variable. The amplitude of stochastic variability and the frequency at which new perturbations are prescribed can be defined by the user and are independent of the simulation time steps. Stochastic perturbations are Gaussian noise terms. In other words, a stochastic variable is calculated as,

$$\begin{cases} y_t = \bar{y}_t + \epsilon_{y,t} \\ \epsilon_{y,t} \sim N(0, \sigma_y^2) \end{cases} \quad (4.135)$$

where y is a generic variable, t indicates the model time step, \bar{y}_t is the deterministic component of y_t , and $\epsilon_{y,t}$ is the stochastic perturbation applied at t to y . The distribution of ϵ_y at any time step is Normal with a variance σ_y^2 .

The model domain can be separated into different subdomains, and different perturbations are applied separately in each subdomain. In other words, the amplitude of variability can be different in the different subdomains, and the perturbations in different subdomains throughout the simulation will be different random numbers. Covariance in the stochastic perturbations can be prescribed between subdomains, as the stochastic terms are sampled from a multivariate Gaussian distribution,

$$\epsilon_t \sim N(\mathbf{0}, \Sigma) \quad (4.136)$$

where Σ is the covariance matrix, and the bold font denotes vectors. If only the diagonal terms of Σ are non-zero, all the individual entries of ϵ_t are independent of each other. Covariance between the entries can be applied by adjusting the off-diagonal terms of Σ . Stochasticity can also be applied to several variables. In this case, covariance between the different stochastic variables, and their respective subdomains, can be prescribed. Regarding the temporal correlation (i.e., for a stochastic variable to exhibit autocorrelation in time), Autoregressive Moving Average (ARMA) models have been implemented for many variables. An ARMA process follows,

$$y_t = \mu_t + \sum_{i=1}^p \varphi_i (y_{t-i} - \mu_{t-i}) + \sum_{j=1}^q \theta_j \epsilon_{y,t-j} + \epsilon_{y,t} \quad (4.137)$$

where μ_t is a deterministic function of time, φ are the autoregressive (AR) coefficients, and θ are the moving-average coefficients (MA). The values of p and q are the orders of the AR and MA part of the ARMA model, respectively.

4.5.5.2 White noise stochasticity

For variables without ARMA implementation, stochasticity is applied as Gaussian white noise (i.e., without temporal correlation, following Eq. (1)). The model parameters can be displayed by running `md.stochasticforcing`. This class includes the following fields,

- `isstochasticforcing` : determines whether the ISSM run allows for stochasticity (1) or not (0, default).
- `fields` : serves to specify which fields are modeled as stochastic variables. Note that all the available fields are automatically displayed when running `md.stochasticforcing`.
- `defaultdimension` : the number of subdomains with their separate stochastic perturbations. Note that different fields thus share the same division in subdomains. Only fields that are modeled as an ARMA process (see Section 2 below) can have their specific division in subdomains.

- `default_id`: the identification number corresponding to a given subdomain (e.g., 1, 2, 3, etc.) for each element of the mesh.
- `stochastic timestep`: this determines the frequency at which new stochastic perturbations are computed. For example, if it is set to 1 year, a stochastic perturbation is kept constant over a period of one year, after which a new stochastic perturbation is generated.
- `covariance`: this is the covariance matrix for the stochastic perturbations (in Eq. (2)). If only a single variable is modeled as stochastic, and with only a single subdomain, then the covariance is of size 1×1 (equivalent to in Eq. (1)). If there are several subdomains and/or several stochastic variables, then the covariance should be of size $D \times D$, where D is the number of subdomains times the number of stochastic variables. The marginal variance of each variable in a given subdomain are the diagonal terms of the covariance matrix. The off-diagonal terms capture the covariance between different variables and different subdomains.

4.5.5.3 ARMA processes

As mentioned above, several variables (or ‘fields’) have an ARMA model implemented. In general, the ARMA models have the same configuration of variables. First, we can focus on the parameters that are similar to the variables of `md.stochasticforcing`,

- `num_basins`: the number of subdomains for this particular variable (can be different than `md.stochasticforcing.defaultdimension`).
- `basin_id`: the identification number corresponding to a given subdomain (e.g., 1, 2, 3, etc.) for each element of the mesh.
- `arma_timestep`: the time resolution of the ARMA model. This thus corresponds to the temporal frequency at which Eq. (3) is recomputed. Note that `epsilon` in Eq. (3) is recomputed via Eq. (2) at the resolution given by `md.stochasticforcing.stochastic timestep`. Thus, Eq. (3) always uses the latest `epsilon` term computed.

Note here that the covariance parameters determining the variability in an ARMA-modeled variable should be prescribed in `md.stochasticforcing.covariance`. Second, we can focus on the parameters that are specific to ARMA-modeled variables. The deterministic background term of the ARMA process (μ_t in Eq.(3)) can be modeled as a piecewise function in time. The order of the background term function with respect to time is set by the user, for example: constant, linear, quadratic, etc. Similarly, the number of breakpoints in the background term function is set by the user, for example: no breakpoint, 1 breakpoint, 2 breakpoints, etc. The remaining parameters to be defined in the model are therefore,

- `num_breaks`: the number of breakpoints applied in the background term function.
- `datebreaks`: the dates at which the breakpoints occur.
- `ar_order`: the order of the autoregressive part of the ARMA model (i.e., in Eq. (3)).
- `ma_order`: the order of the moving-average part of the ARMA model (i.e., in Eq. (3)).
- `arlag_coefs`: the coefficients of the autoregressive part of the ARMA model (i.e., in Eq. (3)).
- `malag_coefs`: the coefficients of the moving-average part of the ARMA model (i.e., in Eq. (3)).
- `num_params`: the number of parameters in the polynomial for the background term function. `num_params` should be 1 for a constant term, 2 for a constant term and a linear trend, 3 for a constant term and a linear trend and a quadratic term, etc.

- `polynomialparams` : the parameters of the polynomial for the background term function. If several subdomains and one or more breakpoints are used, this parameter should be a three-dimensional array. The 1st dimension (along the rows) corresponds to the different subdomains. The 2nd dimension (along the columns) corresponds to the different periods separated by the breakpoints. The 3rd dimension corresponds to the polynomial terms and should be of the same size as specified in `num_params`.

Note that on top of these parameters, ARMA schemes for different variables also have different parameters that are specific to the given variable. Below are the specific parameters for some of these variables.

SMBarma

This class allows for lapse rate adjustments applied to the SMB values (i.e., elevation gradients). This is prescribed with the parameters,

- `elevationbins` : the different elevation ranges in which different lapse rate values apply. The `elevationbins` parameters are specific to the different basins of the SMBarma model.
- `lapsesrates` : the basin-specific lapse rate values applied in their corresponding elevation bin. Note that this parameter can have a third dimension of size 12 if monthly-varying lapse rate values are used (1 value per month should be provided in this case).

frontalforcingsrignotarma

This class uses the frontal melt parameterization of ? (similarly to class `frontalforcingsrignot`),

$$\dot{m} = (Ah_w q_{sg}^\alpha + B) TF^\beta \quad (4.138)$$

where h_w is the front height of the marine glacier, q_{sg} is the subglacial discharge, and A, B, α, β are calibration parameters.

Here, the TF (thermal forcing) variable is simulated as an ARMA process. In addition, sub-annual variability can be prescribed with the parameters `monthlyvals_`. Specifically, each month can have its own effect on TF by being added to the annual TF value. Each one of the 12 monthly values can be prescribed as a piecewise-linear function in time. This is implemented with the parameters,

- `monthlyvals_numbreaks` : the number of breakpoints in the piecewise-linear functions (a single value for all basins and months).
- `monthlyvals_datebreaks` : the dates at which the breakpoints occur.
- `monthlyvals_intercepts` : the intercept terms in the piecewise-linear functions (one specific entry per month and per basin).
- `monthlyvals_trends` : the trend terms in the piecewise-linear functions (one specific entry per month and per basin).

As such, each month in the StISSM simulation has a given calculated monthly value (“monthlyval”). The value of `monthlyval` is entirely defined by the time step of the model run (as a function of the piecewise-linear function). It can be positive or negative. This is added to the ARMA-calculated TF . For example, if the ARMA-calculated TF is 5K and `monthlyval` is -2K, the TF at the timestep is 3K. In addition, for `frontalforcingsrignotarma`, the subglacial discharge variable (in Eq. (4)) can also optionally be modeled as an ARMA process. This is activated by setting,

```
md.frontalforcingsrignotarma.isdischargearma = 1
```

If it is activated, all the model parameters are the same as for a usual ARMA-modeled variable. Their name is differentiated from the parameters of *TF* because they start with `sd_` (for example: `sd_arma_timestep`).

The subglacial discharge also allows for a monthly refinement of the subglacial discharge values calculated with the ARMA model by using the parameter `sd_monthlyfrac`. As an example, the ARMA model can use a yearly step (`sd_arma_timestep = 1`) but the annual value is then adjusted for each month of the StISSM simulation as a function of `sd_monthlyfrac`. The parameter `sd_monthlyfrac` is the fraction of the annual subglacial discharge occurring in each month. It should have 12 entries per row, and one row per subdomain of the `frontalforcingsrignotarma` class. The 12 entries of each row must add up to 1. Suppose that the yearly ARMA-calculated subglacial discharge value is $100\text{m}^3 \text{d}^{-1}$ and the row entries of `sd_monthlyfrac` are `[0, 0, 0, 0, 0, 0.3, 0.5, 0.2, 0, 0, 0, 0]`. If the StISSM time step is in August, then the value of subglacial discharge is calculated as $0.2 \times 100 = 20 \text{ m}^3 \text{d}^{-1}$

4.5.5.4 Additional technical information

Model Sequence

Suppose that variables y and w are stochastic variables. At any timestep of ISSM, the following sequence is executed,

- Determine if timestep t is a stochastic timestep or not (depends on `md.stochasticforcing.stochastictimestep`)
- If timestep is a stochastic timestep: draw new stochastic perturbations, $\epsilon_t \sim N(\mathbf{0}, \Sigma)$
- If timestep is not a stochastic timestep: keep ϵ_t unchanged.
- Perturb variables: $y_t = \bar{y}_t + \epsilon_{y,t}$
- All the other ISSM routines proceed as usual

Computing the stochastic terms

We use the Cholesky decomposition of the covariance matrix to compute the stochastic terms drawn from the multivariate Gaussian distribution with the specified covariance matrix. Let L be the Cholesky decomposition of Σ ,

$$\Sigma = LL^T \quad (4.139)$$

We can first draw a random vector κ from the multivariate Gaussian distribution with covariance matrix identity matrix (I),

$$\kappa \sim N(\mathbf{0}, I) \quad (4.140)$$

Then, we can generate our correlated vector of stochastic perturbation through multiplication,

$$\epsilon = L\kappa \quad (4.141)$$

Note that the random number generator implemented in ISSM to draw random numbers/vectors from univariate/multivariate Normal distributions is the linear congruential generator.

For more details about StISSM, please refer to ?.

Chapter 5

Supplements

5.1 Utilities

Several utilities are available to help the user to set up models and analyze results. Many of these tools are described below. More in-depth information can be found by running `help <FUNCTION>` in the MATLAB prompt where `<FUNCTION>` is any of the following function names. Note that many of these utilities are also available in Python, but that coverage is not 100%.

5.1.1 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
- `rifftiprefine` refine mesh near rift tips

5.1.2 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

5.1.3 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 (x,y) to lat/lon
- `ll2xy` convert lat/lon to (x,y)
- `utm2ll` convert UTM to lat/lon

5.1.4 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)

5.1.5 ARGUS files

- `expcoarsen` coarsen or refine the resolution a contour
- `exptool` create and manage ARGUS files
- `pread` read an ARGUS file
- `expwrite` write an ARGUS file

5.1.6 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

5.1.7 SSH

There are many strategies for managing SSH connections that can help to reduce repeated actions and the amount of details that you have to remember.

5.1.7.1 Aliases

Entries in `~/.ssh/config` (and `/etc/ssh/ssh_config`) allow you to For example, let's say you typically manually connect to a remote machine with,

```
ssh <USER>@<HOST>
```

where `<USER>` is your username on the remote machine and `<HOST>` is the remote machine's hostname. By adding the following entry in `~/.ssh/config`,

```
Host <ALIAS> <HOST>
  HostName <HOST>
  User <USER>
```

where `<ALIAS>` is a name of your choosing given to this connection, you can now log in to the remote machine with, simply,

```
ssh <ALIAS>
```

Likewise, you can create shell aliases that offer the same reusability and time savings. For example, if your default shell is `bash` and configuration `~/.bash_profile`, you might create the following alias,

```
alias ssh-<ALIAS>="ssh <USER>@<HOST>"
```

which would you to log in to the remote machine with, simply,

```
ssh-<ALIAS>
```

See also: Format of SSH client config file `ssh_config` | ssh.com ↗

5.1.7.2 Public Key Authentication

Some remote machines require public key authentication to establish an SSH connection. But it can also be used if it is available and you do not want to have to enter your password on each connection. To set up public key authentication, you will first need an SSH key pair. You may already have a default key pair at `~/.ssh/id_rsa[.pub]`, and it is perfectly acceptable to use this key for authenticating all of your SSH connections. That said, you may wish to create a separate key for each connection, which can be done with the `ssh-keygen` utility.

After copying the public key to `~/.ssh` on the remote machine, either with the `ssh-copy-id` utility or by manually copying its contents to `~/.ssh/authorized_keys`, you can connect with,

```
ssh -i <PRIVATE_KEY> <USER>@<HOST>
```

where `<PRIVATE_KEY>` is the path to the private key on disk (e.g. the default private key is located at `~/.ssh/id_rsa`).

You can also update any aliases you may have created for a connection to use private key authentication.

- In the case of an SSH config file, use the `IdentityFile` property.
- In the case of a shell configuration file, simply add `-i <PRIVATE_KEY>` to the alias command.

See also: [What is SSH Public Key Authentication? | ssh.com](#) ↗

5.1.7.3 Tuneling

Another possibility is to establish an SSH tunnel between the local and remote machines. For example, running,

```
$ ssh -L 1025:localhost:22 <USER>@<HOST>
```

will redirect all traffic from port 1025 on your machine to port 22 on remote machine `<HOST>`. To use the tunnel in ISSM, change `md.cluster.port` from 0 to 1025. Solutions can now be run in the typical way (e.g. `md = md.solve(md, [options])`) but will be conducted on the remote `<HOST>` rather than the local machine.

Tunneling is especially useful on clusters with multi-stop authentication (e.g. authentication required by both login node and compute nodes).

See also: [SSH Tunneling: Examples, Command, Server Config | ssh.com](#) ↗

5.2 Changelog

ISSM 4.23 (Release 2023-11-15)

5.2.0.1 New features/enhancements

- Added support for more controls (Automatic Differentiation)
- SHAKTI now works also with 3D meshes
- Added interpolation routines for common datasets in `src/m/modeldata`
- GlaDS: added turbulent/laminar flag

5.2.0.2 External packages

- Better partial support for Mac with Apple Silicon chips
- Added support for PETSc 3.19

ISSM 4.22 (Release 2022-10-27)

5.2.0.3 New features/enhancements

- Added adaptive time stepping support for GlaDS
- Added Debris modeling for mountain glaciers (still under development)
- Added autoregressive moving average statistical models for generation of SMB, thermal forcing, and floating ice melt rates
- Ability to couple thermal forcing between frontal forcings and Beckmann-Goosse floating ice melt rates

ISSM 4.21 (Release 2022-08-26)

5.2.0.4 New features/enhancements

- Glacier Energy and Mass Balance (GEMB v1.0): A model of firn processes for cryosphere research

Technical release, no other major change.

ISSM 4.20 (Release 2022-06-01)

5.2.0.5 New features/enhancements

- Improved detection of icebergs
- Added new stabilization methods for the level set equation
- Improved implementation of some discrete calving laws
- *NOTE:* `contourlevelzero.m` and `expcontourlevelzero.m` are now merged into `isoline.m`

- Improved usage of `plotmodel` for transient results. For example:

```
plotmodel(md, 'data', 'Vel', 'steps', 1, 'icefront', '-k',
'groundingline', '-r')
```
- AD/CoDiPack now works with checkpointing with multiple cost functions

5.2.0.6 External packages

- Added support for PETSc 3.16 and 3.17
- Partial support for Mac with Apple Silicon chips

ISSM 4.19 (Release 2021-12-23)

5.2.0.7 New features/enhancements

- Improved ISSM's performance and scalability
- Python 3 is now default python version
- Started stochastic forcings
- Starting Julia interface

5.2.0.8 External packages

- Added support for PETSc 3.15
- Added support for CoDiPack 2.0

ISSM 4.18 (Release 2020-12-08)

5.2.0.9 Important Name change

- *NOTE:* we have simplified `md.mask` to accommodate sea level calculations `md.mask.groundedice_levelset` is now `md.mask.ocean_levelset` so that:
 - `md.mask.ocean_levelset > 0` and `md.mask.ice_levelset > 0` : ice-free land
 - `md.mask.ocean_levelset > 0` and `md.mask.ice_levelset < 0` : grounded ice
 - `md.mask.ocean_levelset < 0` and `md.mask.ice_levelset > 0` : ocean
 - `md.mask.ocean_levelset < 0` and `md.mask.ice_levelset < 0` : floating ice

Old models are automatically converted to this new convention. Make sure to update your parameter files (you only need to rename `md.mask.groundedice_levelset` to `md.mask.ocean_levelset`, no change of sign required).

- For the Schoof friction law, we now use C^2 instead of C so make sure to take the square root of `md.friction.C` if you are loading an old model.

5.2.0.10 External packages

- Added support for PETSc 3.12, 3.13 and 3.14. mpich is now installed through PETSc

5.2.0.11 Other

- Added checkpointing capability for CoDiPack

ISSM 4.17 (Release 2020-04-01)

5.2.0.12 Other

- Entirely rewrote the management of inputs in the C++ code, which should significantly improve the efficiency of 3D models.
- *NOTE:* You will need to recompile triangle in externalpackages (See [ISSM Forum](#) for instructions to update your configuration after updating).

ISSM 4.16 (Release 2019-11-01)

5.2.0.13 New features

- ocean undercutting now has its own class `md.frontalforcings`
- Added support for Python 3
- Added Schoof and Tsai friction law
- Added ISMIP6 sub-shelf melting rates
- Added ice front and grounding line flux as possible output (`'TotalCalvingFluxLevelset'` and `'GroundinglineMassFlux'`)
- Added GlaDS subglacial hydrology solver

5.2.0.14 Other

- Significantly improved the scaling of ISSM thanks to feedback from Martin Rueckamp.

ISSM 4.15 (Release 2018-10-05)

5.2.0.15 New features

- Integrated CoDiPack and MeDiPack for automatic differentiation
- Added PDD Scheme from SICOPOLIS

ISSM 4.14 (Release 2018-08-28)

5.2.0.16 New features

- implemented sea level solver
- implemented different melt interpolation at elements crossing the grounding line
- Added PICO and PICOP melt parameterizations

5.2.0.17 Other

- `hydrologysommers()` has been renamed `hydrologicalshakti()`
- *NOTE:* we now have to specify the melt and friction parameterization at the grounding line. You should use the defaults:
 - `md.groundingline.melt_interpolation = 'NoMeltOnPartiallyFloating';`
 - `md.groundingline.friction_interpolation = 'SubelementFriction1';`
- *NOTE:* Python users are now responsible for using their own Python package and make sure that SciPy and NumPy are correctly installed. The respective directories in `externalpackages/` will be removed. We have updated the instructions on the installation page.

ISSM 4.13 (Release 2018-05-10)

5.2.0.18 New features

- *NOTE:* material parameters B, D and E are not averaged over the elements anymore, if you have non-uniform fields, your results will be different with this new version of ISSM
- new capability to compute 3-D crustal motions induced by the applied surface loads (`md.esa`)
- new capability to compute sea-level "fingerprint" induced by land hydrological changes (including ice melting) on elastically-compressible, self-gravitating, rotating Earth (`md.slr`)
- new class for adaptive (bounded) time stepping: `timesteppingadaptive`
- new Adaptive mesh refinement capability, not fully supported yet
- Implemented PICO ice shelf melt rate parameterization

5.2.0.19 Important bug fix

- Fixed a significant memory leak when `md.transient.output_frequency` is greater than 1

5.2.0.20 Other

- class `settings` has been renamed `issmsettings` to avoid shadowing MATLAB's settings function. Make sure to change `md.settings.output_frequency` when you load old models as this field cannot be recovered.
- we removed `md.stressbalance.viscosity_overshoot`, which does not speed up simulations and makes FS crash

ISSM 4.12 (Release 2017-05-19)

5.2.0.21 New features

- Higher order finite elements for thermal models (P1xP2 and P1xP3)
- Adaptive mesh refinement (still under development)

5.2.0.22 Other

- Upgraded to mpich 3.2

ISSM 4.11 (Release 2016-11-04)

5.2.0.23 New features

- Added new anisotropic rheology law ESTAR (publication in preparation)

5.2.0.24 Important bug fix

- For flowline models, the SSA and HO effective strain rates were not accounting for $\dot{\varepsilon}_{zz}$. This is now fixed.

5.2.0.25 Other

- NOTE:** All Enums have been removed from MATLAB and python. When you want to call `solve`, you now need to replace the Enum by one of the following strings:

- `'Stressbalance'` or `'sb'`
- `'Masstransport'` or `'mt'`
- `'Thermal'` or `'th'`
- `'Steadystate'` or `'ss'`
- `'Transient'` or `'tr'`
- `'Balancethickness'` or `'mc'`
- `'Balancevelocity'` or `'bv'`
- `'BedSlope'` or `'bsl'`
- `'SurfaceSlope'` or `'ssl'`
- `'Hydrology'` or `'hy'`
- `'DamageEvolution'` or `'da'`
- `'Gia'` or `'gia'`
- `'Sealevelrise'` or `'slr'`

For example: `md = solve(md, 'Stressbalance')` is now
`md = solve(md, 'Stressbalance')` or simply `md = solve(md, 'sb')`

- upgraded to PETSc 3.7

- No need to install MATLAB anymore in `externalpackages`. You will potentially need to edit your configuring script (`configure.sh`) and include the path to MATLAB instead of `$(ISSM_DIR)/externalpackages/matlab/install`. For example:

```
--with-matlab-dir="/Applications/MATLAB_R2015b.app/" \
```

ISSM 4.10 (Release 2016-04-25)

5.2.0.26 New features

- Added moving boundary capability based on level set method
- Added support for Dakota 6.2

5.2.0.27 Important bug fix

- Time series of constraints for the vertical velocities were not treated correctly for SSA and HO
- 2D SIA has been fixed

5.2.0.28 Other

- *NOTE:* `md.surfaceforcings` has been renamed `md.smb`

ISSM 4.9 (Release 2015-02-12)

- Increase performance by 20-30%
- Minor bug fixes and code enhancements
- Replaced `md.mesh.hemisphere` by `md.mesh.epsg` which gives the EPSG code of the projection being used

ISSM 4.8 (Release 2014-07-30)

- *NOTE:* `md.basalforcings.melting_rate` is now split into:
 - `md.basalforcings.floatingice_melting_rate` that is applied to floating ice only
 - `md.basalforcings.groundedice_melting_rate` that is applied to grounded ice only
- Upgraded to PETSc 3.5
- Added support for MATLAB 2014
- Added analytical validation tests
- Added new surface mass balance parameterization classes
- Minor bug fixes

ISSM 4.7 (Release: 2014-05-13)

Release for the ISSM workshop:

- Added m1qn3 optimization algorithm
- Minor bug fixes

ISSM 4.6 (Release: 2014-04-22)

5.2.0.29 New features

- Improved Taylor Hood Full-Stokes elements in 3d
- Implemented SSA and HO model in 2d flowband
- Added Cuffey and Paterson relationship between T and B (p75). One can use `cuffey.m` or `md.materials.rheology_law = 'Cuffey'`
- Damage evolution model

5.2.0.30 Other

- *NOTE:* `md.geometry.bed` has been renamed `md.geometry.base`
- *NOTE:* `md.geometry.bathymetry` has been renamed `md.geometry.bed`
- *NOTE:* `md.mesh.vertexonbed` has been renamed `md.mesh.vertexonbase`
- the `surfaceforcings` class is now divided into different subclasses depending on the model of surface mass balance needed (`SMB`, `SMBpdd` and `SMBgradients`, more to come)
- the `mesh` class is now divided into different subclasses depending on the type of mesh (2d, 3d prisms, 3d tetras, etc)

ISSM 4.5 (Release: 2013-10-28)

5.2.0.31 New features

- We now have a Full-Stokes flowband model on a fully unstructured mesh (no grounding line dynamics yet)
- Damage evolution model. If you do not want to activate this functionality (default), you will need to add the following fields to the model:

```
md.damage.D = zeros(md.mesh.numberofvertices, 1);
md.damage.spcdamage = NaN * ones(md.mesh.numberofvertices, 1);
```

5.2.0.32 Other

- `md.inversion.cost_functions` has now only one line (i.e., the cost function is no longer step dependent)

5.2.0.33 Update conflict

You might get the following error message from svn:

```
Summary of conflicts:
Tree conflicts: 1
D      C      16566  trunk/src/c/analyses
>    local unversioned, incoming add upon update
```

which can be resolved using the following:

```
cd ${ISSM_DIR}/src/c
svn resolve --accept=working analyses
svn revert --depth=infinity analyses
```

ISSM 4.4 (Release: 2013-09-16)

5.2.0.34 Installation

- configure now checks that the directories provided exist (e.g. `--with-petsc`). Make sure to remove the lines that are not necessary in your configure file.

5.2.0.35 New features

- SSA is now available with quadratic finite elements (P2 Lagrange), or with bubble functions (P1+ statically condensed or not)
- HO is now available with quadratic finite elements (P1xP2, P2xP1 and P2xP2 Lagrange), or with bubble functions (P1+ statically condensed or not)
- FS now available with Taylor-Hood finite elements (P2-P1)

5.2.0.36 Other

- `setflowequation` now takes as argument `'SIA'`, `'SSA'`, `'HO'` or `'FS'` instead of `'hutter'`, `'macayeal'`, `'pattyn'` and `'stokes'` respectively.
- `diagnostic` is now `stressbalance` and `prognostic` is `masstransport`. Make sure to update model fields and Enums accordingly.
- `md.mask` has been entirely redesigned and now has only two fields:
 - `ice_levelset` (which is positive for the nodes where ice is present), and
 - `groundedice_levelset` (which is positive for grounded ice). Example:

```
md.friction.coefficient(find(md.mask.vertexonfloatingice)) = 0.;
```

is now:

```
md.friction.coefficient(find(md.mask.groundedice_levelset < 0.))
= 0.;
```

- we removed the `startup.m` file. you now need to add the path manually when you start MATLAB or use the following command:

```
matlab -r "addpath ${ISSM_DIR}/bin/ ${ISSM_DIR}/lib/"
```

- Python users can use

```
import sys; sys.path.append('${ISSM_DIR}/bin/');
sys.path.append('${ISSM_DIR}/lib/')
```

ISSM 4.3 (Release: 2013-07-02)

5.2.0.37 Installation

- ISSM configuration file: `--with-mpi-lib` is now `--with-mpi-libflags`
- `externalpackages/mpich2` has been renamed `externalpackages/mpich`, you will need to reinstall `mpich` and `petsc` and recompile ISSM.

5.2.0.38 New features

- `'googlemaps'` option for `plotmodel`, which overlays onto satellite image from Google.
- New GIA model from Ivins et al.
- Added support for PETSc 3.4 and mpich 3.0
- Sub-element support for grounding line dynamics

5.2.0.39 Other

- `mpich2` is now `externalpackages/mpich`
- `md.petsc` is now `md.toolkits`

ISSM 4.2 (Release: 2012-06-01)

5.2.0.40 Installation

- `automake` and `autoconf` are now installed along with `libtool` in one single directory `autotools`. You can remove `externalpackages/autoconf` and `externalpackages/automake` and install `externalpackages/autotools`. You will need to source `$(ISSM_DIR)/etc/environment.sh` after the installation.
- You might need to reinstall `mpich2` and `petsc` so that shared libraries are available to libtool.

5.2.0.41 Other

- The MATLAB code has been entirely reorganized
- PETSc 3.3 is now available (we still recommend PETSc 3.2 for now)

ISSM 4.1 (Release: 2012-04-16)

5.2.0.42 Installation

- `ISSM_TIER` was renamed `ISSM_DIR` throughout the code. You will need to update your `.bashrc` and change all `ISSM_TIER` to `ISSM_DIR`
- External packages are now downloaded from the ISSM website when installed. This does not change the current install but makes the download of ISSM much faster, and the svn repository lighter

5.2.0.43 New features

- Newton's method available for SSA and HO
- Exact adjoint available for SSA and HO
- Added new Ordinary Kriging module
- Improved Python interface, still under development

5.2.0.44 Other

- The serial code has been entirely stripped out. As a consequence, the cluster `none` cannot be used anymore. The installation is faster and the code cleaner.

5.3 Validation Guide

Linked here is the validation guide for ISSM [↗](#). Several experiments using EISMINT (European Ice Sheet Modeling INiTiative) [↗](#) and ISMIP-HOM (Ice Sheet Model Intercomparison Project for Higher-Order ice sheet Models) [↗](#) have been run to validate the code. The results are compared with other existing software models.

Chapter 6

Troubleshooting

6.1 Troubleshooting

- Configuring and Compiling
 - External Packages
 - ISSM
- Runtime Errors
- Interfaces
 - MATLAB
 - Python
- Debugging

6.2 Configuring and Compiling External Packages

Chaco Multilevel Graph Partitioning Tool

6.2.0.1 Error compiling on macOS

When compiling Chaco on macOS (most likely using the `${ISSM_DIR}/externalpackages/chaco/install.sh` script), you may encounter an error that reads,

```
util/smalloc.c:6:10: fatal error: 'malloc.h' file not found
#include <malloc.h>
^~~~~~
1 error generated.
make: *** [util/smalloc.o] Error 1
```

To correct this, you will have to have either Xcode Command Line Tools (preferred) or Xcode installed.

- To install the Xcode Command Line Tools, run `xcode-select --install`
- You can install Xcode through the Mac App Store

You will then need to modify the `CPATH` environment variable in your shell profile (e.g. `~/.bashrc`),

- If you installed the Xcode Command Line Tools,

```
export
CPATH="/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/malloc:/us
```

- If you installed Xcode,

```
export
CPATH="/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Devel
```

Then, source your shell profile again and rerun the installation script.

PETSc

Error running make on <EXT_PKG>

When using PETSc to install an external package you may encounter a failure that reads,

```
Error running make on <EXT_PKG>
```

Inspection of `src/configure.log` should reveal a line similiar to,

```
Error running make on <EXT_PKG>: Could not execute
"['<CMD_STRING>']":
```

This is usually not a very helpful error message. What you can do is copy `<CMD_STRING>` and run it manually on the command line to reveal and correct the underlying errors.

6.2.0.2 Error running configure on MPICH

With GCC 10 and later, when using PETSc to install MPICH, you may encounter a failure that reads,

```
Error running configure on MPICH
```

Inspection of `src/configure.log` should reveal,

```
error: The Fortran compiler gfortran will not compile files that call
the same routine with arguments of different types.
```

One way to get around this is to add,

```
--FFLAGS="-fallow-argument-mismatch"
```

to the list of configuration arguments.

C++ error! MPI_Finalize() could not be located! / Error running make; make install on MPICH

When configuring and/or compiling MPICH via PETSc, it may fail with either of the above error messages. Inspection of `src/configure.log` should reveal something like,

```
Checking for program /opt/homebrew/bin/mpicc...found
```

The solution here is one of the following,

1. Uninstall the MPI compiler set installed via package manager (in this example, Homebrew)
2. Make sure that the path to the desired non-MPI compiler set (e.g. `gcc`, `g++`) appears in the `PATH` environment variable before the path to the offending MPI compiler set
3. Remove the `--download-mpich=1` option from the PETSc configuration and instead supply the appropriate options to use the alternate MPI implementation

6.3 Configuring and Compiling ISSM

ld: library not found for -lflapack

If you get the following error: `ld: library not found for -lflapack` (generally happens on macOS), remove the line `--with-blas-lapack-dir` from your configuration and try compiling again.

ld: file not found: @rpath/<LIBRARY_NAME>.dylib for architecture <ARCH>

If you get a variation of the above error on macOS (where `<LIBRARY_NAME>` is a version of ‘libquadmath’ or ‘libgcc’), you need to add the path to the GFortran libraries to your shell configuration file.

If you installed GFortran via FX Coudert’s GitHub repository, you should add,

```
export LIBRARY_PATH="/usr/local/gfortran/lib:${LIBRARY_PATH}"
```

If you installed GFortran via Homebrew, you should add,

```
export
LIBRARY_PATH="/usr/local/Cellar/gcc/<VER>/lib/gcc/current:${LIBRARY_PATH}"
```

6.4 Runtime Errors

cannot connect to local mpd

The following message appears in the errlog file when launching a job in parallel:

```
<SERVER>: 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 CPU's),

```
mpd --ncpus=8 &
```

This will launch the MPI server to manage 8 CPU's on the cluster.

6.5 MATLAB 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 the '[Loading ISSM](#)' section for more info.

MATLAB complains about missing symbols

In some cases, MATLAB complains about missing symbols in MEX files. If your environment is set correctly (for example, you have run `source ${ISSM_DIR}/etc/environment.sh` before starting MATLAB), the error could be due to the fact that MATLAB ships with its own copies of various libraries and manipulates the environment to prefer the location of its own libraries over the locations you have provided. We have experienced this error with the following libraries,

- Boost
- HDF5
- libgfortran

but any library that MATLAB ships with could potentially cause a conflict. There are various options for fixing the above case, but you may want to first run,

```
!ldd <PATH_TO_MEX_FILE>
```

in the MATLAB console to confirm that the cause is as described. After running the command, look through the resulting list of libraries for any that are loaded from the the MATLAB installation directory.

6.5.0.1 Option 1 (preferred)

Locate your copy of the conflicting library and provide it to `LD_PRELOAD` before launching MATLAB. For example,

```
LD_PRELOAD="${ISSM_DIR}/externalpackages/petsc/libhdf5.so" matlab
```

or,

```
export LD_PRELOAD="${ISSM_DIR}/externalpackages/petsc/libhdf5.so"
matlab
```

NOTE:

- The full path to the library including its file extension must be provided.
- Multiple libraries can be supplied to `LD_PRELOAD` by separating them by a colon or single space. For example, `LD_PRELOAD="libfoo.so:libbar.so"`.
- Be careful to avoid overwriting previous changes to `LD_PRELOAD`. For example, `LD_PRELOAD="${LD_PRELOAD}:libfoo.dylib"`.

6.5.0.2 Option 2

In some cases you may be able to provide the full path to the static copy of the library when configuring ISSM. For example, rather than providing generic link flags for libgfortran,

```
--with-fortran-lib="-L/usr/local/Cellar/gcc@10/10.4.0/lib/gcc/10
-lgfotran"
```

you might instead use,

```
--with-fortran-lib="/usr/local/Cellar/gcc@10/10.4.0/lib/gcc/10/libgfotran.a"
```

If the library in question is installed via external package using one of installation scripts provided by ISSM, you may need to reinstall the package using the appropriate installation script with suffix `-static` in order to have a static copy of the library. No further changes to the ISSM configuration or environment would have to be made in this case, but you would still have to recompile ISSM.

6.5.0.3 Option 3 (advanced)

If both dynamic and static versions of the conflicting library are available to you and you do not want to manipulate `LD_PRELOAD` or your ISSM configuration, you can instead modify `$(ISSM_DIR)/m4/issm_options.m4`: find the section of this file that handles the library in question and provided the full path to the static copy of the library rather than generic link flags (see Option 2). Then, reconfigure and recompile ISSM.

If this does not fix the problem, please search or post troubleshooting questions and issues to the [ISSM Forum](#), or ISSM GitHub repository [Discussions](#) or [Issues](#).

MATLAB complains about missing `__gfortran_transfer_array_write` symbol

In some cases, MATLAB complains about missing symbols in MEX files. That is due to the fact that MATLAB uses its own libraries that are not the ones you compiled the MEX files 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 under macOS. There are two ways to fix this problem:

6.5.0.4 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/mac164` for macOS and `*` for other architectures (windows etc).

4. Go to the case that corresponds to your machine's architecture and 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.

6.5.0.5 Option 2 (requires admin privileges)

The second fix consists of replacing MATLAB's library with the one that are on your system, but you will need to have admin privileges.

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 orig
mv libgfortran.* orig
```

Then substitute 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.
```

If this does not fix the problem, please search or post troubleshooting questions and issues to the [ISSM Forum](#), or ISSM GitHub repository [Discussions](#) or [Issues](#).

MATLAB complains about 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 files with. For example, you might have the following error message:

```
libstdc++.so.6: version 'GLIBCXX_3.4.9' not found
```

6.5.0.6 Option 1

You should locate where your `libstdc++.so.6` is, and declare it using `LD_PRELOAD` before you launch MATLAB. For example:

```
LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libstdc++.so.6 matlab
```

6.5.0.7 Option 2

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 files, but you will need to have admin privileges.

NOTE: The following uses MATLAB path `/usr/local/matlab80/` and GCC libraries in `/usr/lib` as an example. Modify the paths as needed.

Before changing the libraries, make a backup:

```
cd /usr/local/matlab80/sys/os/glnxa64
mkdir orig
mv libstdc++.so* orig
mv libgcc_s.so* orig
```

Then substitute the last two by the current ones used by `gcc` (copy or symlink):

```
ln -s /usr/lib/libstd* .
ln -s /lib/libgcc_s.so* .
```

Error using gdaltransform

When making system calls to `gdaltransform` (or other GDAL binaries), you may run into something like,

```
Error using gdaltransform (line <num>)
gdaltransform:
/usr/local/MATLAB/<MATLAB_VER>/bin/glnxa64/libtiff.so.<LIBTIFF_VER>:
no version information available
```

Much like the previous issue, this can be solved with,

```
cd /usr/local/MATLAB/<MATLAB_VER>/bin/glnxa64
sudo mkdir bak
sudo mv libtiff.so* bak
sudo ln -s /usr/lib/libtiff.so* .
```

where `<MATLAB_VER>` is your MATLAB version and `<LIBTIFF_VER>` is the major version of `libtiff` identified in the error message.

Windows: "Binary file <FILE>.outbin not found!"

You may encounter the following warning when running a solution,

```
Binary file <FILE>.outbin not found!
This typically happens when the run crashed.
Please check for error messages above or in the outlog
```

In many cases, this does indeed indicate a runtime error caused by changes made to the source code. However, if you have an unmodified copy of ISSM and you are running one of our Windows configurations, the cause may actually be the equivalent of the previous issue.

NOTE: The following is included in our [Windows installation instructions](#).

After starting a MSYS2 MinGW 64-bit shell instance, run,

```
find /c/msys64/mingw64/lib/gcc/x86_64-w64-mingw32 -name libstdc++*
```

to find all MSYS2 MinGW copies of `libstdc++*`. Then, run,

```
cd /c/Program\ Files/MATLAB/<MATLAB_VER>/bin/win64
mkdir orig
mv libstdc++<VER> orig
cp
    /c/msys64/mingw64/lib/gcc/x86_64-w64-mingw32/<GCC_VER>/libstdc++.a
.
cp
    /c/msys64/mingw64/lib/gcc/x86_64-w64-mingw32/<GCC_VER>/libstdc++.dll.a
.
```

where `<GCC_VER>` is the GCC version that is installed and `<MATLAB_VER>` is the version of MATLAB that you are running. Restart MATLAB to see if your solution now runs.

If this does not fix the problem, please search or post troubleshooting questions and issues to the [ISSM Forum](#), or ISSM GitHub repository [Discussions](#) or [Issues](#).

Windows: MATLAB hangs (no GUI) or crashes with prompt "MATLAB has encountered an internal problem and needs to close." (GUI)

Although MATLAB hangs and crashes can have many causes, this may have to do with the fact that the ISSM build configuration for MEX files needs to be [updated to use the new 64-bit API](#).

In the case of a hang (no GUI), use the Windows Task Manager to exit the MATLAB instance.

In the case of a crash while using the MATLAB GUI,

- in the prompt that reads "MATLAB has encountered an internal problem and needs to close.", click the 'Show Report' button
- scroll down to the 'Stack Trace (from fault):' section
- look for a symbol that reads `'mexfile::Inspector::needs_upgrade'` to verify the likely cause of the crash
- click 'Don't Send' and exit MATLAB

MATLAB complains about intel_fast_memm symbol

If you compile MEX files 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 files with. For example, you might have the following error message:

```
Invalid MEX-file
  '/users/username/test/issm/install/lib/IssmConfig.mexa64':
<ISSM_DIR>/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/intel64/`).

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/mac64` for mac and `*` for other architectures (Windows etc). Go to the case that corresponds to your machine's architecture
4. Uncomment the following line:

```
#          LD_LIBRARY_PATH=$MATLAB/sys/opengl/lib/$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.

If this does not fix the problem, please search or post troubleshooting questions and issues to the [ISSM Forum](#), or ISSM GitHub repository [Discussions](#) or [Issues](#).

Fatal error in MPI_Init

You may encounter a runtime error that looks something like the following,

```
Fatal error in MPI_Init: Other MPI error, error stack:  
MPIR_Init_thread(474).....  
MPID_Init(190).....: channel initialization failed  
MPIDI_CH3_Init(89).....  
MPID_nem_init(320).....  
MPID_nem_tcp_init(173).....  
MPID_nem_tcp_get_business_card(420):  
MPID_nem_tcp_init(379).....: gethostbyname failed,  
    MT-<integers> (errno 1)  
loading results from cluster
```

This issue has been observed on more recent versions of MacOS, on both the precompiled and compiled-from-source versions of ISSM.

The fix involves modifying the hosts file, `sudo vi /etc/hosts` and adding a line that reads,

```
127.0.0.1 MT-<integers>
```

where `MT-<integers>` is what is display in the original error message.

After saving the changes to `/etc/hosts` and restarting MATLAB, the issue should be resolved.

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 resetting 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, please search or post troubleshooting questions and issues to the [ISSM Forum](#), or ISSM GitHub repository [Discussions](#) or [Issues](#), and we will help you convert all ISSM's MATLAB classes to the older syntax.

**Invalid MEX-file [...] symbol not found in flat namespace
‘__ZN14ToolkitOptions11toolkittypeE’**

If you experience the above error (reportedly happens on macOS), add the following to your ISSM configuration,

```
--with-cxxoptflags="-D_DO_NOT_LOAD_GLOBALS_ -g -O2 -std=c++11"
```

reconfigure, recompile, and relaunch MATLAB.

macOS: Invalid MEX-file

On running ISSM in MATLAB under macOS, you might receive an error such as,

```
Invalid MEX-file
<ISSM_DIR>/lib/IssmConfig_matlab.mexmaci64':
dlopen(<ISSM_DIR>/lib/IssmConfig_matlab.mexmaci64,
0x0006): Library not loaded: @rpath/libMatlabEngine.dylib
Referenced from: <1523DDB9-0961-3ACC-AB28-1AC66F6E3204>
<ISSM_DIR>/lib/IssmConfig_matlab.mexmaci64
```

This can be corrected by running,

```
sudo ln -s
/Applications/<MATLAB_VER>.app/extern/bin/maci64/libMatlabEngine.dylib
/Applications/<MATLAB_VER>.app/bin/maci64/libMatlabEngine.dylib
```

where <MATLAB_VER> is the version of MATLAB running when the error was encountered.

6.6 Python Interface

Indexing and Interfacing with the C/C++ Core

Many of the model fields represented in ISSM's C/C++ core use MATLAB-like, 1-based indexing. As such, when using the Python API, make sure to adjust field indices before requesting a solution, and then again before plotting and other processing of results.

6.7 Debugging

How to debug a crash in issm.exe?

If there is crash during the solve phase, we strongly suggest using [Valgrind](#). Install Valgrind using one of the script in the directory `${ISSM_DIR}/externalpackages/valgrind`. Valgrind will subsequently be embedded in ISSM and can detect segmentation faults as well as memory leaks. To do so, set the model debugging field to 1 and use only one CPU,

```
md.debug.valgrind = 1;
md.cluster.np = 1;
```

in your script, or, simply set,

```
valgrind = true;
```

in `src/m/classes/debug.m` to apply run Valgrind on all subsequently-run scripts. Launch the solution sequence and read the `errlog` file that it outputs.

6.7.0.1 When a build includes Boost

If your build includes the Boost C++ libraries, there are additional configuration steps needed to overcome a conflict when running Valgrind. Either,

1. install Valgrind with `externalpackages/valgrind/install-<OS>.sh`
2. set `#define BOOST_MATH_PROMOTE_DOUBLE_POLICY false` in `externalpackages/valgrind/src/boost/math/tools/user.hpp` before running `bootstrap.sh` and `b2 install`
3. if you are using Dakota, this will need to be reinstalled as well

Interfaces

6.7.0.2 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).

6.7.0.3 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`

Bibliography