Ice Sheet System Model 2008
User Guide

Authors:

Éric Larour[1]
Mathieu Morlighem[2,4]
Hélène Seroussi[2,4]
Ala Khazendar[2]
Éric Rignot[2,3]

[1]Division 35, Thermal and Cryogenics Section,
Mechanical Division, MS 157-316.
Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109.

[2]Division 33, Radar Science and Enginnering Section,
Communications, Tracking and Radar Division, MS 157-316.
Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109.

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

[4]Laboratoire de Mécanique des sols, Structures et matériaux (MSSMat)
École Centrale Paris, CNRS UMR 8579
Grande Voie des Vignes, 92295 Châtenay-Malabry Cedex, FRANCE

February 19, 2009

Summary

This manual explains how to use the ISSM code, capable of modeling ice flow in 2d and 3d using finite elements.

Key-words: ISSM, ice flow, finite elements, 2-D, 3-D, matlab

Contents

Chapter 1

Installation

1.1 Fetching code from repository

ISSM is actively managed using a code versioning system called CVS. The code is stored in a repository, and can be remotely fetched, modified, updated, and uploaded. This allows for multiple users to develop the code in an organized way. In order to fetch a version of the code, users will need to install CVS on their machine. This versioning system is available at the following address: http://www.nongnu.org/cvs/. Once CVS, issue the following command:

```
> cvs -d useraccount@wilkes.jpl.nasa.gov:/home/larour/Ice_Repository  co -r ice2 -P ice1
```

This command will download the stable version of the code *ISSM* named *ice*2 from the repository on the wilkes machine, located in the $/home/larour/Ice_Repository$ directory, onto the current local directory (replace *useraccount* by your user name on wilkes). Users are free to choose whichever location they want and to rename *ice*1. Users should not forget the $-P$ option. Failing to do so would allow CVS to fetch empty directories. This would result in a code structure not intended by the developers.

1.2 Setting up *ISSM*

In the main directory of *ISSM* (initially named *ice*1), open the file *install.sh* and modify the configuration if you need to:

- $CONFIG =$ "$config\_linux32.mk$" for a 32 bits linux OS,
- $CONFIG =$ "$config\_linux64.mk$" for a 64 bits linux OS,
- $CONFIG =$ "$config\_mac.mk$" for Mac OS X

Once the configuration files are properly setup, users should type *shinstall.sh* at the top level directory ice1 to run the installation sequence (compiling *C* routines and modifying the startup file).

```
> sh  install.sh
```

1.3 Code structure.

ISSM is a set of tools which can carry out various functions, all related to ice flow modeling. Its structure is apparent when typing $ls\ -v$ at the command prompt.

```
CVS  CieloBindings  Cron  Doc  Etc  Makefile  Mesh  Model  Public  Solver  Tests  Trash
Utils  archive.sh contributors.txt  install.sh  mailinglist  startup.m  todo
```

The *CVS* directory is proper to the versioning system. You can just ignore it. The startup.m file we already alluded to. The code is mainly composed of a mesher (Mesh directory), a model structure (Model directory) to hold the mesh and other input parameters to an ice flow model, and a solver (Solver directory) that works on the model itself. There is also a documentation (Doc directory) and a repository for old code (Trash directory). Finally, the Utils directory is a place holder for miscellaneous code ranging from interpolation tools to display plotting routines, etc ...

Chapter 2

Getting started

2.1 Location of the *startup.m* file

When matlab is launched, all the *ISSM* tools need to be correctly located. This is done by the *startup.m* file located in the *ice*1 directory. This file needs to be located in the starting directory of matlab, or in the directory where matlab will establish its root.

2.2 Creation of a model

To create a new model, you can type the following command in Matlab's Command window:

```
> md=model;
```

That will create a new model named "md" whose class is "model". The structure of a model contains much information: the mesh, the boundary conditions, the materials' properties, the results of the runs, etc...

When one creates a new model, all these fields are empty or *NaN* (not a number), but "md" is ready to be used as a model. Type *md* in the command window if you want Matlab to print the fields contained in *md*.

One can add a note and/or a name to the model in order to remember easily what it deals with:

```
> md.name='PineIslandGlacier';
> md=addnote(md,'Pine Island Glacier test 1, geometry of 1996');
```

2.3 Mesh and Geometry.

2.3.1 Required files.

In order to set up properly the model, four files are required:

- A file that countains the coordinates of the domain outline, i.e. the domain one wishes to mesh: *DomainOutline.exp*. The contour must be closed and may contain holes.
- A file that contains the grounding line of the glacier: *Iceshelves.exp*. The contour must be closed and may contain holes.
- A file that contains the islands or ice rises in the ice shelf: *Islands.exp*. The contour must be closed and may contain holes.

- A file that contains all the properties of the ice, the constants used in the simulation and other parameters: *Parameters.par*.

Usually, the first 3 files (with an "exp" extension) come from ARGUS and have the following shape:

```
## 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 other file is a Matlab file that plugs all the parameters into the model *md*.

2.3.2 Meshing the domain

To mesh the domain, you need the file 'DomainOutline.exp' from ARGUS as explained above. Once again, the contour described in this file must be closed and can contain holes if needed. Then the following command will create the mesh:

```
> md=mesh(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 2 grids). To see how the mesh looks like, one can type:

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

2.4 Geography of the domain

The solver will use two different boundary conditions depending on the nature of the ice. 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. The files used come from ARGUS[1]. To add this information to the model, type the following command:

```
> md=geography(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).

2.5 Parametrization of the model

To run a simulation, the solver needs many parameters: physical constants, number of iterations, relaxation constant, thickness and surface of the glacier, etc. All this information must be located in a file (The description of this file is given in Chapter 7). To plug all these data to the model, use the following command:

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

The first argument is the model, and the second argument between quotes is the file containing all the parameters' values. One can see chapter 7 for more information on the parameter file.

[1] If the whole domain is an ice shelf, use $md = geography(md,'all','')$. If the whole domain is grounded, use $md = geography(md,'','')$.

2.6 Extrusion of the domain (optional)

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

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

The first argument is the model ad 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 require 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 happening near the bedrock usually and users might want to refine more the lower layers than the upper ones. An extrusion exponent of 1 will create a mesh with layers vertically equally distributed. The higher the extrusion exponent, the more refined the base. An extrusion exponent of 3 is generally enough.

2.7 Setting types of elements

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

- Hutter's ice sheet model (2d and 3d)
- MacAyeal's shelfy stream model (2d and 3d)
- Pattyn's higher order model (3d: extruded mesh only)
- Full Stokes' model (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=setelementstype(md,'pattyn','Pattyn.exp','macayeal',md.elementoniceshelf,'fill','hutter');
```

The routine *setelementstype* works like *plotmodel*: it works with an even number of inputs (without counting *md* itself). There are five possible options: $'hutter'$, $'macayeal'$, $'pattyn'$, $'stokes'$ and $'fill'$. The first four options must be followed by one of the following argument:

- 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 option) assigned to. All options are not required to be used. The previous example assigns the model of Pattyn for the element inside the contour $Pattyn.exp$, 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=setelementstype(md,'macayeal','all');
```

Chapter 3

Solving and plot of the simulations' results

3.1 Solver.

To run a simulation, type the following command:

```
> md=solve(md,'diagnostic','ice');
```

The first argument is the model, the second is the nature of the simulation one wants to run (it can be *'diagnostic'*, *'prognostic'*, *'thermalsteady*, *'thermaltransient'*, *'transient' 'control'*, *'parameters'* or *'mesh2grid'*), and the last argument is the package the user wants to use to calculate the solution (*ice* or *cielo*). If no solver is specified, the default solver is *'ice'*.

- If one runs a simulation using *'diagnostic'*, the solver will compute the velocity, the horizontal components only if the mesh is 2d, the three components if the mesh is 3d
- The *'prognostic'* solution will compute the new thickness of the ice sheet system depending of the time step of the model and the velocity field that must be already computed. The time step is given by *md.dt*
- The *'thermalsteady'* simulation will solve the thermal fields of the model, ie the temperature and melting. To do so one needs to create a 3d mesh and to compute the velocity first. It solves a stationary solution for thermal, ie $\frac{\partial T}{\partial t} = 0$
- The *'thermaltransient'* simulation will solve the thermal fields of the model with a temporal evolution. An initial temperature is needed to run a *'thermaltransient'* computation. One can precise the time step and number of iterations depends using *md.dt* and *md.ndt*. All the intermediary results are saved in $md.thermaltransient_results$
- The *'transient'* simulation will solve both velocity and thermal fields of the model with a temporal evolution. One can precise the time step and number of iterations depends using *md.dt* and *md.ndt*. All the intermediary results are saved in $md.transient_results$
- The *'control'* solution is a control method applied on the model Chapter 4 a detailed description

You can compute additional fields with the *'parameters'* solution, using *md.parameteroutput'*, such as *'stress'*, *'deviatoricstress'*, *'viscousheating'*, *'strainrate'* or any combination of these fields given a a structure:

```
> md.parameteroutput={'stress','deviatoricstress'};
> md=solve(md,'parameters');
```

3.2 Plots

3.2.1 Generality

Once the simulation has converged, there are many fields of the model that one can plot very easily with the plotmodel function of the ISSM code. plotmodel takes the model *md* as first argument and then an even number of options (like *setelementstype*). To plot a given field, use the option *'data'* followed by the field one wants to plot. For the thickness:

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

or alternatively

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

You can plot several fields in the same time but you have to add the argument 'data' before each field you want to plot:

```
> plotmodel(md, 'data', 'thickness','data','surface','data','vel','data','elements_type')
```

Most of the fields can be plotted: thickness, nodes on ice shelf, x-component of the velocity, thickness and surface of the glacier, etc...

3.2.2 Graphics options

We saw in the previous section the option *'data'*. The option *'data'* can be followed by:

- any field of the model structure
- *'boundaries'*: this will draw all the segment boundaries to the model, including rifts.
- *'deviatoricstress_tensor'*: plot the components of the deviatoric stress tensor (tauxx, tauyy, tauzz, tauxy, tauxz, tauyz) if computed
- *'deviatoricstress_principal'*: plot the deviatoric stress tensor principal axis and principal values
- *'deviatoricstress_principalaxis*1*'*: arrow plot the first principal axis of the deviatoric stress tensor(replace 1 by 2 or 3 if needed)
- *'elements_type'*: model used for each element
- *'elementnumbering'*: numbering of elements
- *'gridnumbering'*: numbering of grids
- *'highlight'*: highlights certain grids or elements when using 'gridnumbering' or 'elementnumbering' option
- *'mesh'*: draw mesh
- *'quiver'*: arrow plot of the velocity in 2d
- *'quiver*3*'*: arrow plot of the velocity in 3d
- *'quivervel'*: arrow plot of the velocity superimposed with its magnitude
- *'riftvel'*: velocities along rifts
- *'riftrelvel'*: relative velocities along rifts
- *'riftpenetration'*: penetration levels for a fault
- *'strainrate_tensor'*: plot the components of the strain rate tensor (exx, eyy, ezz, exy, exz, eyz) if computed
- *'strainrate_principal'*: plot the strain rate tensor principal axis and principal values)
- *'strainrate_principalaxis*1*'*: arrow plot the first principal axis of the strain rate tensor (replace 1 by 2 or 3 if needed)
- *'stress_tensor'*: plot the components of stress tensor (sxx,syy,szz,sxy,sxz,syz) if computed
- *'stress_principal'*: plot the stress tensor principal axis and principal values)

- *'stress_principalaxis1'*: arrow plot the first principal axis of the stress tensor (replace 1 by 2 or 3 if needed)
- *'transient_results'*: this will display all the time steps of a transient run
- *'transient_movie'*: this will display the time steps of a given field of a transient run
- *'thermaltransient_results'*: this will display all the time steps of a thermal transient run

But there are other options such as:

- *'caxis'*: modify colorbar range. (array of type $[a \quad b]$ where $b \leq a$)
- *'colorbar'*: add colorbar (string *'on'* or *'off'*)
- *'colormap'*: same as standard matlab option
- *'wrapping'*: repeat n times the colormap (n must be an integer);
- *'edgecolor'*: same as standard matlab option
- *'fontsize'*: same as standard matlab option
- *'fontweight'*: same as standard matlab option
- *'resolution'*: resolution used by section value (array of type [*horizontal_resolutionvertical_resolution*]) *horizontal_resolution* must be in meter, and *vertical_resolution* a number of layers
- *'showsection'*: show section used by 'sectionvalue' (string *'yes'*)
- *'sectionvalue'*: give the value of data on a profile given by an Argus file
- *'smooth'*: smooth element data (string *'yes'*)
- *'title'*: same as standard matlab option
- *'view'*: same as standard matlab option
- *'xlim'*: same as standard matlab option
- *'ylim'*: same as standard matlab option
- *'zlim'*: same as standard matlab option
- *'xlabel'*: same as standard matlab option
- *'ylabel'*: same as standard matlab option

Any options (except *'data'*) can be followed by *'#i'* where *'i'* is the subplot number, or *'#all'* if applied to all plots. For example: to plot the velocity and the mesh at the same time, with a colorbar for both plots bot a 2d view for the mesh:

```
> plotmodel(md,'data','vel','data','mesh','view#2',3,'colorbar#all','on')
```

3.3 Saving the model

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

```
> save square.model md
```

That will create a file *square.model* made from the model md. To load this file, type:

```
> loadmodel square.model
```

the loaded model will be named *md*.

Chapter 4

Control Methods

ISSM allows control method on the drag (*drag*) and the viscosity parameter (B) spatial distribution for 2d meshes, when an observed velocity field is known and plugged in *md.vx_obs*, *md.vy_obs* and *md.vel_obs* (See parameter file section for more details). This section explains how to launch a control method and which parameters must be tuned.

4.1 Misfits

The control methods can use different types of misfit. The misfit must be entered in $md.fit$. it can be *Absolute*, *Relative* or *Logarithmic*.

4.1.1 Absolute misfit

This is the classic way of calculating a misfit between a modeled and observed velocity field:

$$J = \iint_{\Omega \cup \partial\Omega} \frac{1}{2} \left((u - u^{obs})^2 + (v - v^{obs})^2 \right) d\Omega \tag{4.1}$$

where:

- u: is the glacier modeled velocity projected on the x-axis
- v: is the glacier modeled velocity projected on the y-axis
- u^{obs}: is the glacier modeled velocity projected on the x-axis
- v^{obs}: is the glacier modeled velocity projected on the y-axis

4.1.2 Relative misfit

The relative misfit is defined as follows:

$$J = \iint_{\Omega \cup \partial\Omega} \frac{1}{2} \left(\frac{\overline{vel}^2}{(u^{obs} + \varepsilon)^2} (u - u^{obs})^2 + \frac{\overline{vel}^2}{(v^{obs} + \varepsilon)^2} (v - v^{obs})^2 \right) d\Omega \tag{4.2}$$

where:

- u: is the glacier modeled velocity projected on the x-axis
- v: is the glacier modeled velocity projected on the y-axis
- u^{obs}: is the glacier modeled velocity projected on the x-axis

- v^{obs}: is the glacier modeled velocity projected on the y-axis
- $\overline{vel}$: is an averaged velocity used for dimensional purposes, usually, one takes $md.meanvel = 1000$ m/year (to be converted into m/s)
- ε: is a minimum velocity used to avoid the observed velocity being equal to zero. One takes usually $md.eps = eps$ (Matlab's epsilon)

4.1.3 Logarithmic misfit

The relative misfit is defined as follows:

$$J = \iint_{\Omega \cup \partial\Omega} 4\,\overline{vel}^2 \left(ln\left(\frac{vel + \varepsilon}{vel^{obs} + \varepsilon} \right) \right)^2 d\Omega \tag{4.3}$$

where:

- vel: is the glacier modeled velocity
- vel^{obs}: is the glacier modeled velocity
- $\overline{vel}$: is an averaged velocity used for dimensional purposes, usually, one takes $md.meanvel = 1000$ m/year (to be converted into m/s)
- ε: is a minimum velocity used to avoid the observed velocity being equal to zero. One takes usually $md.eps = eps$ (Matlab's epsilon)

4.2 Control Methods parameters

4.2.1 Control type

One can choose which parameter to optimize. This is registered in *md.control_type* in a cell. For exemple, if one wants to optimize the viscosity parameter spatial distribution:

```
> md.control_type={'B'};
```

If one wants to optimize several parameters such as *drag* and B:

```
> md.control_type={'B','drag'};
```

4.2.2 Constraints

Usually, the viscosity or the drag are constrained. For example the viscosity can not be negative. To prevent the inversed parameters from being out of the physical range, enter the constraints values in *md.mincontrolconstraint* and *md.maxcontrolconstraint*. If one does not wish to constrain the inverse parameters, enter NaN.

For example, if one wants to optimize the *drag* and the viscosity parameter B, with the following constraints:

$$0 \leq drag \qquad 10 \leq B \leq 10^9 \tag{4.4}$$

type:

```
> md.mincontrolconstraint=[0,10];
> md.maxcontrolconstraint=[NaN,10^9];
```

4.2.3 optimization parameters

4.2.3.1 *md.nsteps*

This is done a given number of times set in *md.nsteps*. For example, if one wants 100 iterations, type

```
> md.nsteps=100;
```

The other control parameters (*md.optscal*, $md.fit$ and *md.maxiter*) must have a length equal to *md.nsteps*.

4.2.3.2 *md.optscal*

At each iteration, the routine evaluates the gradients of the misfit with respect to the inversed parameters spatial distribution and updates each parameter spatial distribution using a scalar as follows:

$$P^{new} = P^{old} + dimensionalscalar \times scalar \times \frac{\overrightarrow{gradient}}{\max(\overrightarrow{gradient}_i)} \tag{4.5}$$

The scalar is calculated by a C routine and is constrained between 0 and 1. The dimensional scalar is a constant and must be entered in *md.optscal*. We recommand to use 10^8 for the viscosity and 10 for the drag. For example, if one inverses the drag and the viscosity:

```
> md.optscal=[10*ones(md.nsteps,1) 10^8*ones(md.nsteps,1)];
```

4.2.3.3 $md.fit$

We aw that there were three possible misfits. A misfit of 0 is associated to an absolute misfit, 1 to a relative misfit and 2 to a logarithmic misfit. If one wants half logarithmic and half absolute, use the following command:

```
> md.misfit=[2*ones(floor(md.nsteps/2),1); 0*ones(ceil(md.nsteps/2),1)];
```

4.2.3.4 *md.tolx* and *md.maxiter*

At each iteration, the routine search for a scalar between 0 and 20. It stops when the difference of the cost function between 2 scalars is less than *md.tolx*, or when it reaches a maximum number of iteration *md.maxiter*. We recommend :

```
> md.tolx=10^{-4};
> md.maxiter=20*ones(md.nsteps,1);
```

4.3 Launching a control method

Control methods are not performed with the package *ice* for computational time reasons. Only *CIELO* has control methods implemented. To launch a control method, fill all the previous fields and type:

```
>  md=solve(md,'control','cielo');
```

Chapter 5

Cluster computing

ISSM can use the *CIELO* software to run in parallel on a cluster. This section shows how to use this capability.

5.1 Setting up the environment to use Cielo

We assume users have correctly setup *CIELO* in the directory *CIELODIR*, on the cluster *CLUSTERNAME*. In order for *ISSM* to recognize the settings for this cluster, users should edit the file *cielo.rc* found in the Etc/ directory of *ISSM*. This file list all the clusters recognized by *CIELO*. Two fields in particular are of interest, *server_codepath*, which should be *CIELODIR/server_delivery*, and *server_executionpath*, which should be the name of a directory where the logs of the parallel computation will be dumped. Let's call it *TESTINGDIR* for now. Once those two paths are setup, *ISSM* should be ready to be used with *CIELO*.

5.2 Using Cielo

Given a model md, users can check where the computation will occur by typing:

```
> md.solpar
```

A list of solution parameters will ensue. The part regarding use of a cluster is displayed here:

```
'parallelisation'
cluster: astrid      (set to 'cluster_name' to run in cluster, 'none' to run serially)
np: 8      (number of CPUS requested on cluster)
'ice'
scheduler_configuration: local      (name of cluster configuration used from Parallel Matlab toolbox)
'cielo'
batch: 1      (is cluster running in batch mode (batch=1), or in client/server mode (batch=0, default
exclusive: 0      (set to 1 if CPUS used are not to be shared with other users, 0 otherwise)
time: 10      (amount of time requested on cluster)
alloc_cleanup: 0      (allocation cleanup before starting a job, default 1)
queue:       (queue name)
```

This indicates that the cluster being used is 'astrid', with 8 processors (np=8), in batch mode (batch=1). Those three parameters are needed to run using cielo.

```
md.cluster='wilkes';
md.np=8;
md.batch=1;
```

Once those parameters are setup, users can use three solution sequences (for now): diagnostic_horiz, control and thermalsteady. The solution sequences are called the same way, but specifying cielo as the computational engine.

```
md=solve(md,'diagnostic_horiz','cielo');
```

Once the computation is over (check the logs in $TESTINGDIR$), users can load the results using:

```
md=loadresultsfromcluster(md,'diagnostic_horiz');
```

The following fields 'errlog' and 'outlog' in the model structure will hold the run logs. If errlog is not empty, something probably went wrong.

5.3 Faq

```
The following message  appears in the errlog file when launching my job in batch mode.
mpdrun_wilkes.jpl.nasa.gov: cannot connect to local mpd (/tmp/mpd2.console_larour); possible causes:
  1. no mpd is running on this host
    2. an mpd is running but was started without a "console" (-n option)
~
~
~
~
```

This message means that the MPI (Message Passing Interface) server, called mpd, is not running. Therefore, no parallel jobs can run on the cluster. To solve this issue, just type, at the command prompt on the server side (if for example your cluster has 8 cpus):

```
mpd --ncpus=8 &
```

This will laucn the MPI server to manage 8 cpus on the cluster.

Chapter 6

Rifts

ISSM allows simulation of rifts. This section explains how to create a model that includes rifts, and how to control their behaviour.

6.1 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 interset 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=mesh(md,'DomainOutline.exp','Rifts.exp',4000);
> md=meshprocessrifts(md);
> md=geography(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.

6.2 Rifts tip refining

Rifts in a mesh will not modify the type of meshing occuring 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=mesh(md,'DomainOutline.exp','Rifts.exp',4001);
> md=rifttipsrefine(md,2000,30000);
> md=meshprocessrifts(md);
> md=geography(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.

6.3 Rifts in parameter file

The structure 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(i).fill=1 %include water in the rifts
>   md.rifts(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.

6.4 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,'diagnostic','ice');
```

will suffice. Bear in mind, 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 which 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.

6.5 Rifts plotting

Rifts can be plotted using the following special plots:

```
> plotmodel(md,'data','riftpenetration','data','riftvel','data','riftrelvel');
> end
```

these three plots will give users a view of which parts of the rifts are opening, closing, at which relative speed, etc ...

Chapter 7

Parameter file

This chapter indicates all the fields the parameter file must contain. These fields are divided into five parts: material parameters, other physical parameters, boundary conditions, observations and solution parameters. One can type *md* to see all the parameters that the model md contains.
This file is created for a two dimensional mesh. The parameters will be automatically extruded if a part of the mesh is extruded. Number of elements, number of grids refer to the number of elements and grids in the two dimensional mesh.

7.1 Material parameters

This section indicates all the fields of the model dealing with the material parameters. One must add all these fields in the parameter file. One can access to all these fields typing *md.mat* The list of all these fields is presented below:

- *md.rho_ice*: ice density (in kg/m^3)
- *md.rho_water*: water density (in kg/m^3)
- *md.heatcapacity*: ice heat capacity (in $J/kg/K$)
- *md.thermalconductivity*: ice thermal conductivity (in $W/m/K$)
- *md.beta*: depression of the melting point due to pressure (in K/Pa)
- *md.B*: viscosity parameter, given in $Pa/s^{\frac{1}{n}}$. This field must be an array of size the number of elements
- *md.n*: empirical constant of Glen's flow law. This field must be an array of size the number of elements

7.2 Other physical parameters

This section indicates all the fields of the model dealing with all the other physical and geometrical parameters

7.2.1 Physical parameters

- *md.g*: constant of gravity (in m/s^2)
- *md.drag_type*: type of drag for the basal friction. 0 for none, 1 for plastic and 2 for viscous

- *md.drag*: drag coefficent k defined as

$$\overrightarrow{\tau_b} = -k^2 N_{eff}^r \|\overrightarrow{v}\|^{s-1} \overrightarrow{u_b} \tag{7.1}$$

  This field must be an array of size the number of grids. (see Theory guide p. 11)

- *md.p*: drag constant described above ($r = q/p$ and $s = 1/p$). This field must be an array of size the number of elements

- *md.q*: drag constant described above ($r = q/p$ and $s = 1/p$). This field must be an array of size the number of elements

7.2.2 Geometrical parameters

- *md.surface*: height of the glacier surface (in m). This field must be an array of size the number of grids. The surface must be positive for the grids on ice shelf

- *md.thickness*: glacier thickness (in m). This field must be an array of size the number of grids. The thickness must be positive for every grid

- *md.firn_layer*: firn thickness on top of the ice (in m). This field must be an array of size the number of grids. The thickness of firn must be positive for every grid

- *md.bed*: bed height of the ice (in m). This field must be an array of size the number of grids

7.3 Boundary conditions

This section indicates all the fields of the model dealing with all the boundary conditions.

7.3.1 Diagnostic parameters

- *md.gridondirichlet_diag*: 1 if the grid has a dirichlet boundary condition and 0 else. This field must an array of size the number of grids

- *md.dirichletvalues_diag*: value of the dirichlet boundary condition. This field must an array of size the number of grids

- *md.segmentonneumann_diag*: matrix of 3 columns. Each line holds the characteristics of one segment located on a Neumann type boundary condition. The first column is the first grid number of the segment, the second column is the second grid number and the last column is the element number that holds the two previous grids.

- *md.neumannvalues_diag*: value of the Neumann boundary condition (in N), NaN if there is only the water pressure. This field must an array of size the number of segments on the boundaries

7.3.2 Prognostic parameters

- *md.gridondirichlet_prog*: 1 if the grid has a Dirichlet boundary condition and 0 else. This field must an array of size the number of grids

- *md.dirichletvalues_prog*:value of the Dirichlet boundary condition. This field must an array of size the number of grids

- *md.segmentonneumann_prog*: matrix of 3 columns. Each line holds the characteristics of one segment located on a Neumann type boundary condition. The first column is the first grid number of the segment, the second column is the second grid number and the last column is the element number that holds the two previous grids.

- *md.neumannvalues_prog*: value of the neumann boundary condition, NaN if there is only the water pressure. This field must an array of size the number of segments on the boundaries

- *md.segmentonneumann_prog2*: matrix of 3 columns. Each line holds the characteristics of one segment located on a Neumann type boundary condition. The first column is the first grid number of the segment, the second column is the second grid number and the last column is the element number that holds the two previous grids.
- *md.neumannvalues_prog2*: value of the Neumann boundary condition, NaN if there is free radiation. This field must an array of size the number of segments on the boundaries

7.3.3 Thermal parameters

- *md.gridondirichlet_thermal*: 1 if the grid has a thermal Dirichlet boundary condition and 0 else. This field must an array of size the number of grids
- *md.dirichletvalues_thermal*: value of the thermal Dirichlet boundary condition (in K). This field must an array of size the number of grids

7.4 Observations

This section indicates all the fields of the model dealing with all the observations.

- *md.vx_obs*: velocity field in the x direction (in m/a). This field must be an array of size the number of grids
- *md.vy_obs*: velocity field in the y direction (in m/a). This field must be an array of size the number of grids
- *md.vel_obs*: velocity field (norm of the velocity in m/a). This field must be an array of size the number of grids
- *md.observed_temperatures*: field of surface temperatures of ice (in K). This field must be an array of size the number of grids
- *md.accumulation*: field of accumulation of snow (in m/a). This field must be an array of size the number of grids
- *md.melting*: field of melting rate (in m/a). This field must be an array of size the number of grids
- *md.geothermal_flux*: field of geothermal flux (in W/m^2). This field must be an array of size the number of grids

7.5 Solution parameters

This section indicates all the fields of the model dealing with all the solution parameters.

7.5.1 Parallelization parameters

The list below deals with the parameters used for the parallelization of the code.

- *md.cluster*: indicates the name of the cluster one wants to use ('none' for the package 'ice')
- *md.bash*: indicates if the bash mode is activated (0 for the package 'ice', 1 for CIELO)
- *md.np*: number of CPUs to use for the cluster (8 for *wilkes*)
- *md.exclusive*: 1
- *md.time*: 1

7.5.2 Statics parameters

The list below deals with the parameters needed for a static solution.

- *md.eps_rel*: relative velocity convergence criterion
- *md.eps_abs*: absolute velocity convergence criterion (NaN if not desired)
- *md.penalty_offest*: magnitude of penalty stiffness for contact problems: $\kappa = 10^{penalty\_offest} \times max(abs(K))$ where K is the initial stiffness matrix
- *md.lowmem*: field for memory. 0 unless you are running low on cluster memory, then 1
- *md.sparsity*: matrix sparsity
- *md.acceleration*: 1 if accelerated Mac Ayeal's solution, else 0 ($'ice'$ only)
- *md.debug*: 1 if user wants to print the detail of each non linear iteration.

7.5.3 Dynamics parameters

The list below deals with the parameters needed for a dynamic solution

- *md.dt*: time step in year
- *md.ndt*: time span in year
- *md.artificial_diffusivity*: turn on or off the artificial viscosity for prognostic and transient solutions. This field must be 1 or 0
- *md.minh*: minimum thickness to avoid stiffness singularity.

7.5.4 Control parameters

The list below deals with the parameters needed for a control method

- *md.control$_t$ype*: list of parameters where inverse control is carried out, for example 'drag' or 'B'
- *md.fit*: type of fit for the control method. This field can be 'absolute', 'relative' or 'logarithmic'. The default value is 'absolute'
- *md.meanvel*: this constant is used for relative and logarithmic fits
- *md.epsvel*: minimum velocity used for relative and logarithmic fits
- *md.maxiter*: maximum iterations during one optimization search
- *md.nsteps*: number of optimization searches
- *md.optscal*: scaling factor on gradient during optimization
- *md.mincontrolconstraint*: list containting the minimum tolerated for the inversed parameters
- *md.maxcontrolconstraint*: list containting the maximum tolerated for the inversed parameters

7.6 Example of parameter file

Here is an example of a parameter file for a square ice shelf. We assume that the file *Front.exp* already exists and is a closed contour holding the nodes in the ice front.
The program *ArgusContourToMesh* is a Matlab routine located in the directory *Utils*. It flags the grids or elements that are within a contour from an Argus contour and a mesh.

```
%Ok, start defining model parameters here

%material parameters
    md.g=9.8;
    md.rho_ice=917;
    md.rho_water=1023;
    di=md.rho_ice/md.rho_water;
    md.yts=365*24*3600;
    md.heatcapacity=2009;
    md.thermalconductivity=2.2; %W/mK
    md.beta=9.8*10^-8;

%Solution parameters
    %parallelization
    md.cluster='none';
    md.np=2;
    md.time=1;
    md.exclusive=0;

    %statics
    md.lowmem=1;
    md.eps_abs=10;
    md.eps_rel=0.01;
    md.penalty_offset=3;
    md.penalty_melting=10^7;
    if md.numberofgrids<1000000,
    md.sparsity=.001;
    else
    md.sparsity=.0001;
    end

    %dynamics
    md.dt=1*md.yts; %1 year
    md.ndt=md.dt*10;
    md.artificial_diffusivity=1;

    %control
    md.control_type={'drag'}; %'drag', 'B'
    md.nsteps=5;
    md.tolx=10^-4;
    md.maxiter=20;
    md.optscal=10;
    md.fit='logarithmic'; %'absolute','relative','logarithmic'
    md.meanvel=1000/md.yts; %1000 meters/year
    md.epsvel=eps;

%Geometry
    disp('      creating thickness');
    hmin=300;
    hmax=1000;
```

```
    ymin=min(md.y);
    ymax=max(md.y);
    md.thickness=hmax+(hmin-hmax)*(md.y-ymin)/(ymax-ymin);
    md.firn_layer=10*ones(md.numberofgrids,1);
    md.bed=-di*md.thickness;
    md.surface=md.bed+md.thickness;

    disp('      creating velocities');
    md.vx_obs=zeros(md.numberofgrids,1);
    md.vy_obs=zeros(md.numberofgrids,1);
    md.vel_obs=sqrt(md.vx_obs.^2+md.vy_obs.^2);

    disp('      creating drag');
    md.drag_type=2; %0 none 1 plastic 2 viscous
    md.drag=200*ones(md.numberofgrids,1); %q=1.
    %Take care of iceshelves: no basal drag
    pos=find(md.elementoniceshelf);
    md.drag(md.elements(pos,:))=0;
    md.p=ones(md.numberofelements,1);
    md.q=ones(md.numberofelements,1);

    disp('      creating temperature');
    md.observed_temperature=(273-20)*ones(md.numberofgrids,1);

    disp('      creating flow law paramter');
    md.B=paterson(md.observed_temperature);
    md.n=3*ones(md.numberofelements,1);

    disp('      creating accumulation rates');
    md.accumulation=ones(md.numberofgrids,1)/md.yts; %1m/a
    md.melting=0*ones(md.numberofgrids,1)/md.yts; %1m/a

%Boundary conditions:
    disp('      boundary conditions for diagnostic model: ');
    %Build gridonicefront, array of boundary grids belonging to the icefront:
    gridinsideicefront=ArgusContourToMesh(md.elements,md.x,md.y,expread('Front.exp',1),'node');
    gridonicefront=double(md.gridonboundary & gridinsideicefront);

    md.gridondirichlet_diag=zeros(md.numberofgrids,1);
    pos=find(md.gridonboundary & ~gridonicefront);md.gridondirichlet_diag(pos)=1;
    md.dirichletvalues_diag=zeros(md.numberofgrids,2);

    pos=find(gridonicefront(md.segments(:,1)) | gridonicefront(md.segments(:,2)));
    md.segmentonneumann_diag=md.segments(pos,:);
    %dynamic boundary conditions (water pressure)
    md.neumannvalues_diag=NaN*ones(length(md.segmentonneumann_diag),1);

    disp('      boundary conditions for prognostic model: ');
    md.gridondirichlet_prog=zeros(md.numberofgrids,1);
    md.dirichletvalues_prog=zeros(md.numberofgrids,1);
    pos=find(gridonicefront(md.segments(:,1)) | gridonicefront(md.segments(:,2)));
    md.segmentonneumann_prog=md.segments(pos,:);
    md.neumannvalues_prog=zeros(size(md.segmentonneumann_prog,1),1);
    md.neumannvalues_prog(:)=NaN; %free radiation

    pos=find(gridonicefront(md.segments(:,1)) | gridonicefront(md.segments(:,2)));
```

```
md.segmentonneumann_prog2=md.segments(pos,:);
md.neumannvalues_prog2=zeros(size(md.segmentonneumann_prog2,1),1);
md.neumannvalues_prog2(:)=NaN; %free radiation

disp('      boundary conditions for thermal model: ');
md.gridondirichlet_thermal=ones(md.numberofgrids,1); %surface temperature
md.dirichletvalues_thermal=md.observed_temperature;
md.geothermalflux=zeros(md.numberofgrids,1);
pos=find(md.elementonicesheet);md.geothermalflux(md.elements(pos,:))=50*10^-3;%50 mW/m^2
```

Chapter 8

Example

This section gives the reader an example of simulation using *gdal*, *Argus* and *ISSM* to calculate the velocity field of Amery Glacier.

8.1 Radar Image

In order to build a proper domain outline, one needs an image of the glacier. This image is often a radar image that needs to be resized. The tools for accomplishing this feat are *gdal_translate* and *gdalwarp*, both part of the *GDAL* suite of utilities (www.gdal.org). For this example we will be using the Antarctica mosaic LIMA, available on http://lima.nasa.gov/

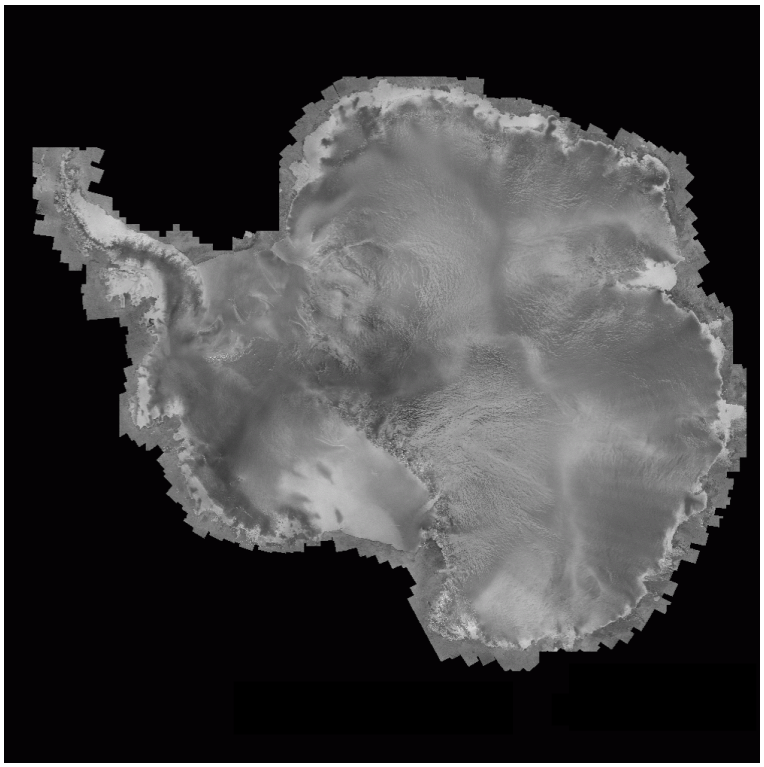


Figure 8.1: Original radar image

To know the original mosaic properties, type

```
> gdalinfo file.gif
```

where *file.gif* is the name of this mosaic. At the end of the properties, the coordinates of the corners are displayed:

```
Corner Coordinates:
Upper Left  (-3174450.000,2406320.000)
Lower Left  (-3174450.000,-2816080.000)
Upper Right (2867150.000,2406320.000)
Lower Right (2867150.000,-2816080.000)
Center      (-153650.000,-204880.000)
```

The first step is to determine the coordinates for the box of interest. In this case we will just pull out Amery Glacier. The bounding box is approximately 1549950, 951120 to 2259950, 554120 (upper left to lower right). To extract Amery from the mosaic, we use:

```
> gdal_translate -a_ullr 1549950 951120 2259950 554120
  -projwin 1549950 951120 2259950 554120 file.tif Amery.tif
```

The *−projwin* option specifies the area one wants to clip out of the mosaic using the coordinate system. We also used the *−a_ullr* option to force the output image to have the bounding coordinates we want. The result is:

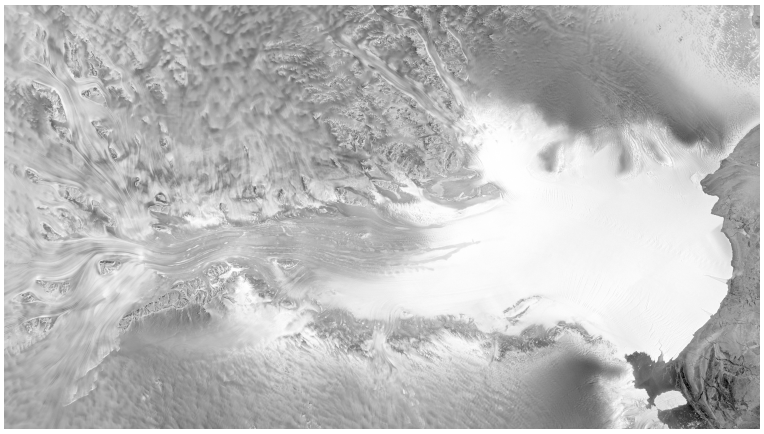


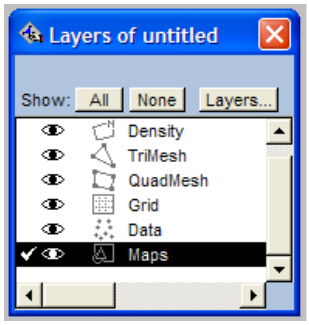
Figure 8.2: Amery Glacier

8.2 Loading the image in *ARGUS ONE*

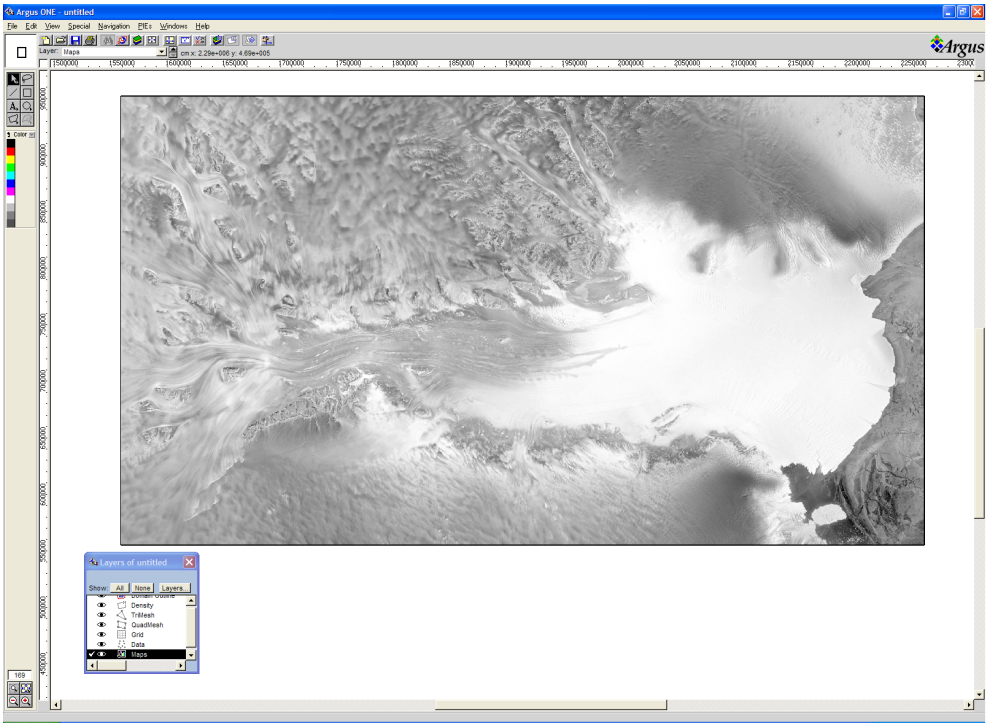
To build all the files required by *ISSM*, we use the software *ARGUS ONE*. The first step is to import the box coordinates of the radar image in *ARGUS*. To do so, we type a text file that holds the image corners coordinates. There are 5 points since *ARGUS* requires a closed box:

```
## Name:
## Icon:0
# Points Count Value
# X pos Y pos
1550000.000000 951000.000000
550000.000000 554000.000000
2260000.000000 554000.000000
2260000.000000 951000.000000
1550000.000000 951000.000000
```

The extension of this file must be *.exp*. In this example, this file is named *Amery_box.exp*. Then, open *ARGUS*, File/Import Domain Outline.../Text file and select *Amery_box.exp*. The new box in Argus has exactly the coordinates of *Amery.tif*. Now, the image must be downloaded and placed in the box. Select *map* in the *Layers* window:

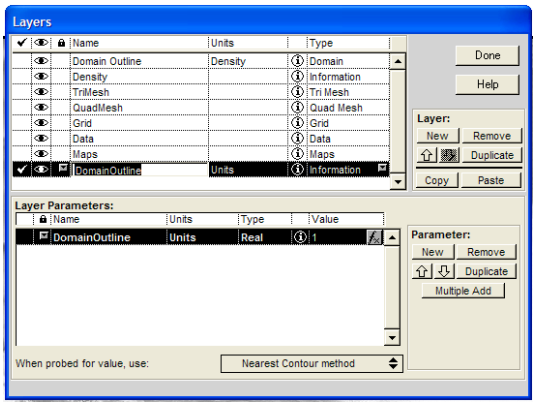


To download the image: File/Place Image... and select *Amery.tif*. Then, resize the image and move it to fit in the black frame.



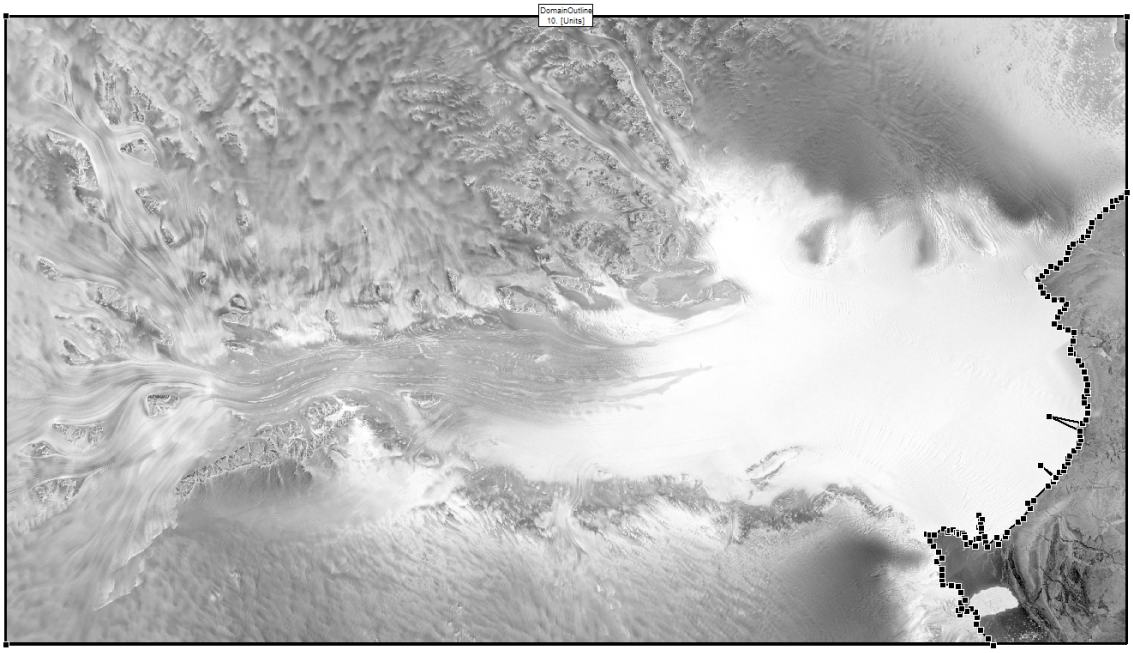
8.3 Building the *DomainOutline.exp* file

In the *Layers* window, click on *Layers*, then *New* and name it *DomainOutline*, and enter the value 1 (The value needs to be different than 0):



Then click on *Done* and select the icon *Contour* on the left of the window:

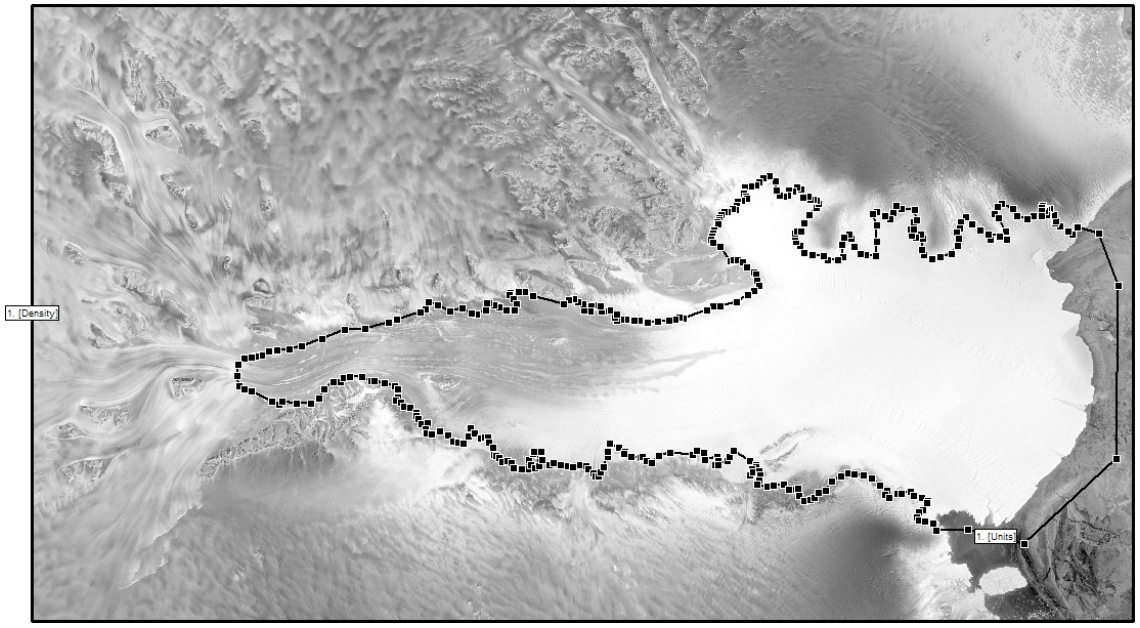
Select the domain outline of the glacier (it must be a closed contour, do not take sea ice) and double click to end:



Then click on File/Export/Export DomainOutline... and save the file *DomainOutline.exp*.

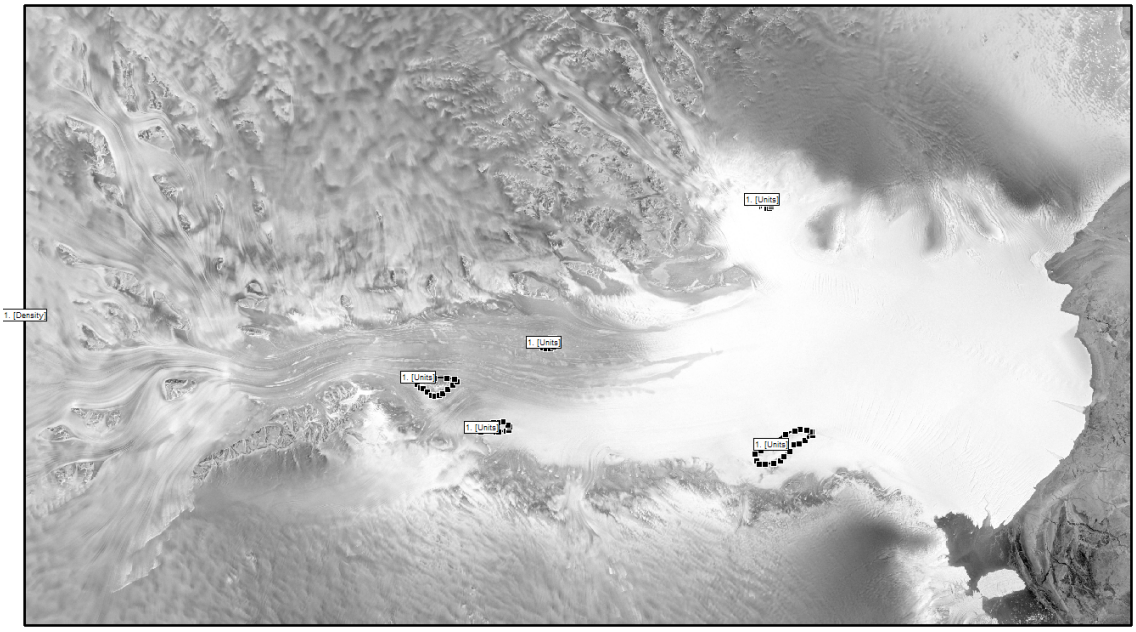
8.4 Building the *Iceshelves.exp* file

Repeat the previous step with a new layer named *Iceshelves*. In this step, one must know the position of the grouding line. The contour must be closed and include the ice shelf as follows:



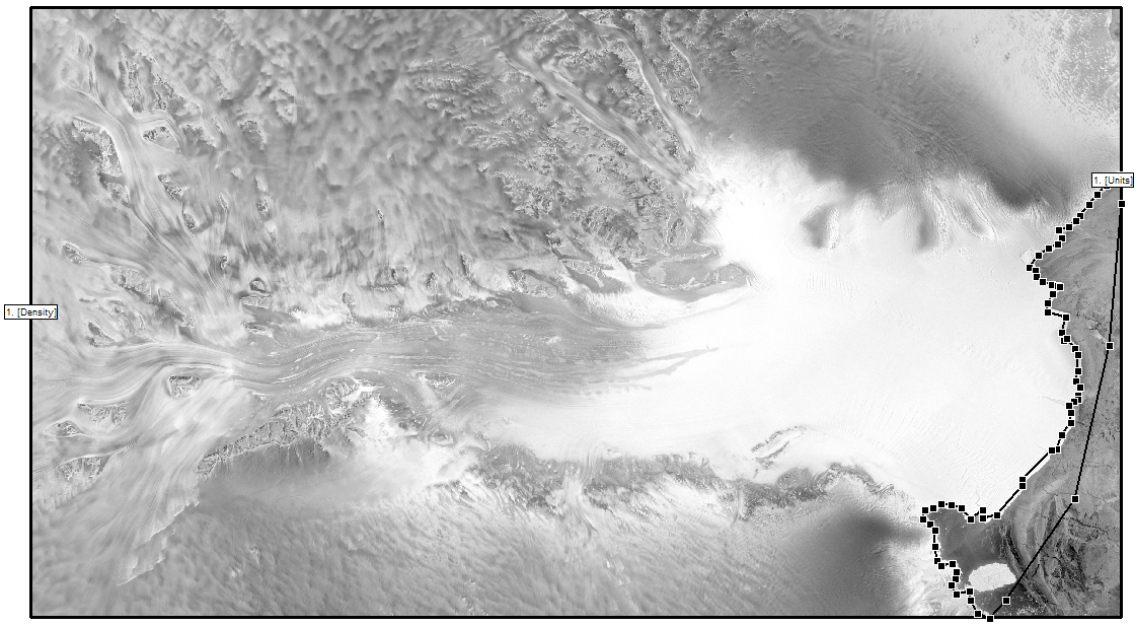
8.5 Building the *Islands.exp* file

Skip this step if there is no island in the ice shelf part of the glacier. Repeat the previous step with a new layer named *Islands*. This layer holds several contours that stand for islands in the ice shelf:



8.6 Building the *Front.exp* file (needed in Amery.par)

Repeat the previous step with a new layer named *Front*. In this step, the contour must be closed and include the ice front as follows:



8.7 Building the *Amery.par* parameter file

Here is just an example of *Amery.par*. Several data are needed (surface, thickness, surface temperature,...). The content of these files is not detailed here and the user will have to change the parts of the parameter file that refer to these data to fit his.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Some hardcoded parameters for Amery%%%%%%%%%%

%some parameterization for this parameter file :)
package='ice';
modeldatapath='/home/larour/ModelData';
thicknesspath=[modeldatapath '/BedMap/gridded/thickness'];
firnpath=[modeldatapath '/BroekeFirn1km/firn'];
surfacepath=[modeldatapath '/BamberDEMAntarctica1km/surface_smooth30_lowslope'];
thicknessiceshelvespath=[modeldatapath '/HartmutThicknessAntarctica/HartmutThickness'];
mosaicpath=[modeldatapath '/RignotAntarcticaVelMosaicRampErsAlos/RignotAntVel'];
temperaturepath=[modeldatapath '/GiovinettoZwallyTemperatures92/Giovinetto_Temperatures'];
heatfluxpath=[modeldatapath '/HeatfluxAntarctica/RignotHeatFlux'];

%material parameters
    md.g=9.8;
    md.rho_ice=917;
    md.rho_water=1023;
    di=md.rho_ice/md.rho_water;
    md.yts=365*24*3600;
    md.heatcapacity=2009;
    md.thermalconductivity=2.2; %W/mK
    md.beta=9.8*10^-8;

%Solution parameters
    %parallelization
    md.cluster='cosmos';
    md.exclusive=1;
    md.batch=1;
    md.alloc_cleanup=0;
```

```
md.np=10;
md.time=60;
md.connectivity=10;

%solver
md=solversettoasm(md);

%Set solver
%statics
md.eps_rel=.01; %1 per cent
md.eps_abs=10; %10 m/yr
md.penalty_offset=3;
md.lowmem=1;
md.sparsity=.001;

%dynamics
md.dt=100*md.yts; %1 year
md.ndt=md.dt*100;
md.artificial_diffusivity=1;

%control
md.control_type={'drag'};
md.nsteps=50;
md.tolx=10^-4;
md.maxiter=30*ones(md.nsteps,1);
md.optscal=500*ones(md.nsteps,1);
md.fit=2*ones(md.nsteps,1); %absolute fit
md.meanvel=1000/md.yts; %1000 meters/year
md.epsvel=eps;
md.debug=0;
md.plot=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('      reading bedmap thicknesses');
load(thicknesspath);
md.thickness=DataInterp(x_m,y_m,thickness,md.x,md.y,package); clear thickness

disp('      reading firn layer');
load(firnpath);
md.firn_layer=DataInterp(x_m,y_m,firn,md.x,md.y,package); clear firn

disp('      reading Bamber dem');
load(surfacepath);
md.surface=DataInterp(x_m,y_m,antarctica_surface,md.x,md.y,package);
clear antarctica_surface

minsurf=1;
pos=find(isnan(md.surface) | (md.surface<=0));
md.surface(pos)=minsurf;
pos=find(isnan(md.thickness) | (md.thickness<=0));
md.thickness(pos)=minsurf/(1-di);
md.bed=md.surface-md.thickness;

disp('      reading ice thickness over ice shelves, from Hartmut');
load(thicknessiceshelvespath);
```

```
iceshelf_thickness=DataInterp(x_m,y_m,hartmut_thickness,md.x,md.y,package);
clear hartmut_thickness

pos=find((md.gridoniceshelf) & (iceshelf_thickness~=0));
md.thickness(pos)=iceshelf_thickness(pos);
md.bed(pos)=-di*md.thickness(pos);
md.surface(pos)=(1-di)*md.thickness(pos);
clear iceshelf_thickness

disp('      reading velocities from Rignot');
load(mosaicpath);
posting=1000; x1=max(md.x);x0=min(md.x);y1=max(md.y);y0=min(md.y);
i0=round((x0-x_m(1))/posting)-1; i1=round((x1-x_m(1))/posting)+1;
j0=round((y0-y_m(1))/posting)-1; j1=round((y1-y_m(1))/posting)+1;
x_m=x_m(i0:i1); y_m=y_m(j0:j1);
vx=vx(j0:(j1-1),i0:(i1-1));
vy=vy(j0:(j1-1),i0:(i1-1));

md.vx_obs=DataInterp(x_m,y_m,vx,md.x,md.y,package); clear vx
md.vy_obs=DataInterp(x_m,y_m,vy,md.x,md.y,package); clear vy
md.vel_obs=sqrt(md.vx_obs.^2+md.vy_obs.^2);

%drag md.drag or stress
md.drag_type=2; %0 none 1 plastic 2 viscous
md.drag=300*ones(md.numberofgrids,1); %q=1.

%Take care of iceshelves: no drag md.drag
pos=find(md.elementoniceshelf);
md.drag(md.elements(pos,:))=0;
md.p=ones(md.numberofelements,1);
md.q=ones(md.numberofelements,1);

%Load md.temperature from Giovinetto:
disp('      loading temperature');
load(temperaturepath);
md.temperature=griddata(xt,yt,Ts92,md.x,md.y); clear Ts92

%flow law
disp('      creating flow law paramters');
md.n=3*ones(md.numberofelements,1);
md.B=paterson(md.temperature);

disp('      creating accumulation rates');
md.accumulation=ones(md.numberofgrids,1); %1m/a
md.melting=0.1*ones(md.numberofgrids,1); %.1m/a

%Deal with boundary conditions:
disp('      boundary conditions for diagnostic model');
gridinsideicefront=ArgusContourToMesh(md.elements,md.x,md.y,expread('Front.exp',1),'node');
gridonicefront=double(md.gridonboundary & gridinsideicefront);
md.gridondirichlet_diag=zeros(md.numberofgrids,1);
pos=find(md.gridonboundary & ~gridonicefront);md.gridondirichlet_diag(pos)=1;
md.dirichletvalues_diag=zeros(md.numberofgrids,2);

pos=find(md.gridoniceshelf(md.segments(:,1)) | md.gridoniceshelf(md.segments(:,2)));
if ~isempty(pos),
```

```
    md.segmentonneumann_diag=md.segments(pos,:);
else
    md.segmentonneumann_diag={};
end
md.neumannvalues_diag=NaN*ones(length(md.segmentonneumann_diag),1);

disp('      boundary conditions for prognostic model');
md.gridondirichlet_prog=zeros(md.numberofgrids,1);
md.dirichletvalues_prog=zeros(md.numberofgrids,1);
pos=find(md.gridoniceshelf(md.segments(:,1)) | md.gridoniceshelf(md.segments(:,2)));
if ~isempty(pos),
    md.segmentonneumann_prog=md.segments(pos,:);
else
    md.segmentonneumann_prog={};
end
md.neumannvalues_prog=zeros(size(md.segmentonneumann_prog,1),1);
md.neumannvalues_prog(:)=NaN; %free radiation

pos=find(md.gridoniceshelf(md.segments(:,1)) | md.gridoniceshelf(md.segments(:,2)));
if ~isempty(pos),
    md.segmentonneumann_prog2=md.segments(pos,:);
else
    md.segmentonneumann_prog2={};
end
md.neumannvalues_prog2=zeros(size(md.segmentonneumann_prog2,1),1);
md.neumannvalues_prog2(:)=NaN; %free radiation

disp('      thermal model');
md.melting=zeros(md.numberofgrids,1);
md.observed_temperature=md.temperature;
disp('      watch out: screwing up thermal parameterization on purpose!');
md.gridondirichlet_thermal=ones(md.numberofgrids,1);
md.gridondirichlet_thermal(1)=0;
md.dirichletvalues_thermal=md.temperature;

disp('      reading geothermal flux');
load(heatfluxpath);
md.geothermalflux=DataInterp(x_m,y_m,heatflux_Antarctica,md.x,md.y,package);
pos=find(md.geothermalflux==0);md.geothermalflux(pos)=80;
md.geothermalflux=md.geothermalflux/1000; %map is given in mW/m^2, we need it in W/m^2
```

8.8 Launching a 2d simulation

Once the four files *DomainOutline.exp*, *GroundingLine.exp*, *Islands.exp*, *Front.exp* and *Amery.par* are in the same directory, launch *Matlab*. The first thing to be done is to create a model:

```
> md=model;
```

Then one can mesh the entire domain outline. The refinement is $4000m$ here:

```
> md=mesh(md,'DomainOutline.exp',4000);
```

Load the geography of the model (To determine which part of the model is on the ice shelf, on the ice sheet or on the ice front). If there is no island, the third argument must be ''.

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

The parameters can now be loaded (the file *Front.exp* is required for this step)

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

The model is now ready to be extruded (10 layers, and an extrusion exponent of 3):

```
> md=extrude(md,10,3);
```

Last step: assigning the models to the elements (here, MacAyeal on the ice shelf and Pattyn for the grounded ice)

```
> md=setelementstype(md,'MacAyeal',md.elementoniceshelf,'fill','Pattyn');
```

Then, one can launch a diagnostic:

```
> md=solve(md,'diagnostic','ice');
```

Once it has ended, one can plot the velovity field

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

8.9 Control method on the viscosity parameter spatial distribution

One can optimize the viscosity parameter spatial distribution in order to be closer to an observed velocity field. The first things to do are to plug the observed velocity field (here named $vxobs, vyobs, velobs$) into the model:

```
>md.obs_vx=vxobs;
>md.obs_vy=vyobs;
>md.obs_vel=velobs;
```

Then one can type the following command to setup the model (See control method section 4 for more details):

```
> md.control_type={'B'};
> md.nsteps=100;
> md.misfit=[2*ones(floor(md.nsteps/2),1); 0*ones(ceil(md.nsteps/2),1)]
> md.optscal=[2*10^7*ones(floor(md.nsteps/2),1); 10^7*ones(ceil(md.nsteps/2),1)];
> md.mincontrolconstraint=10;
> md.maxcontrolconstraint=10^9;
```

Then, one can launch a simulation on CIELO (provided that the fields related to parallelization are correctly filled):

```
> md=solve(md,'control','cielo');
```