ISSM Workshop 2012

Ice Sheet System Model
ISMIP-HOM

Chris Borstad[1], Bao Duong[5,1], Feras Habbal[2,1], Daria Halkides[1,3], Michiel Helsen[2], Eric Larour[1], Mathieu Morlighem[2], Lan Nguyen[5,1], Gilberto Pérez[4,1], Eric Rignot[2,1], John Schiermeier[1], Nicole Schlegel[1], **Hélène Seroussi**[1]

[1] Jet Propulsion Laboratory - California Institute of Technology

[2] University of California, Irvine

[3] Joint Institute for Regional Earth System Science & Engineering, UCLA

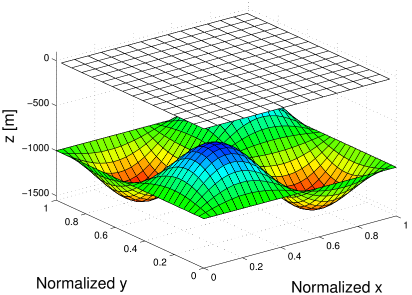
[4] University of Southern California

[5] Cal Poly Pomona

Outline

Test A

Square ice sheet flowing over a bumpy bed



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

Description of test in `ismiphom_description.pdf`

Mesh

Use `squaremesh` to create the mesh:

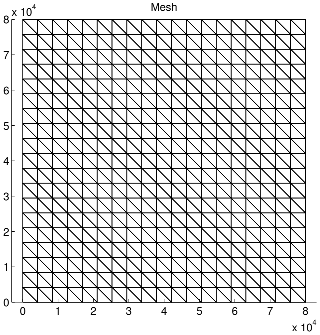
```
>> help squaremesh
   SQUAREMESH - create a structured square mesh

      This script will generate a structured square mesh
      Lx and Ly are the dimension of the domain (in meters)
      nx anx ny are the number of nodes in the x and y direction
      The coordinates x and y returned are in meters.

   Usage:
      [md]=squaremesh(md,Lx,Ly,nx,ny)
```

Example for 80 km and with 20 points in each direction

Mesh

Example for 80 km and with 20 points in each direction:

```
>> md=squaremesh(md,80000,80000,20,20)
>> plotmodel(md,'data','mesh')
```



Set Mask

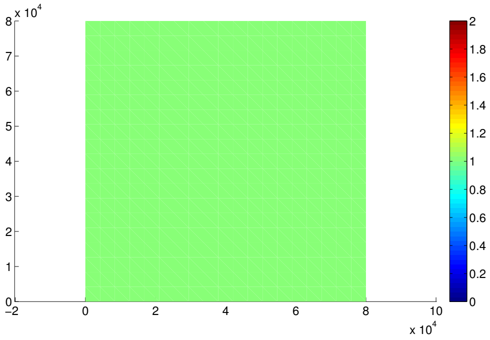
Use `setmask` to specify that everything is grounded ice

```
>> help setmask
 SETMASK- establish boundaries between grounded and floating ice.

   By default, ice is considered grounded. The contour floatingicename defines nodes
   for which ice is floating. The contour groundedicename defines nodes inside an floatingice,
   that are grounded (ie: ice rises, islands, etc ...)
   All input files are in the Argus format (extension .exp).

   Usage:
      md=setmask(md,floatingicename,groundedicename)

   Examples:
      md=setmask(md,'all','');
      md=setmask(md,'Iceshelves.exp','Islands.exp');
```

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

Set Mask

```
>> plotmodel(md,'data',md.mask.elementongroundedice)
```



Parameterization

Use `ISMIPA.par` file to parameterize the model:

```
>> help parameterize
  PARAMETERIZE - parameterize a model

     from a parameter matlab file, start filling in all the @model fields that were not
     filled in by the mesh.m and mask.m @model methods.
     Warning: the paramter file must be able to be run in Matlab

     Usage:
        md=parameterize(md,parametername)

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

Parameterization

Use `ISMIPA.par` file to parameterize the model:

```
>> help parameterize
  PARAMETERIZE - parameterize a model

     from a parameter matlab file, start filling in all the @model fields that were not
     filled in by the mesh.m and mask.m @model methods.
     Warning: the paramter file must be able to be run in Matlab

     Usage:
        md=parameterize(md,parametername)

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

```
>> md=parameterize(md,'./ISMIPA.par');}
```

Parameterization

Use `ISMIPA.par` file to parameterize the model:

First geometry:

- surface: $s(x,y) = -x\tan(\alpha)$, $\alpha = 0.5^\circ$
- bed: $b(x,y) = s(x,y) - 1000 + 500\sin(\omega x)\sin(\omega y)$, $\omega = \frac{2\pi}{L}$
- thickness: $h(x,y) = s(x,y) - b(x,y)$

Parameterization

Use `ISMIPA.par` file to parameterize the model:

First geometry:

- surface:

```
md.geometry.surface=-md.mesh.x*tan(0.5*pi/180);
```

- bed:

```
md.geometry.bed=md.geometry.surface-1000 ...
... +500*sin(md.mesh.x*2*pi/L).*sin(md.mesh.y*2*pi/L);
```

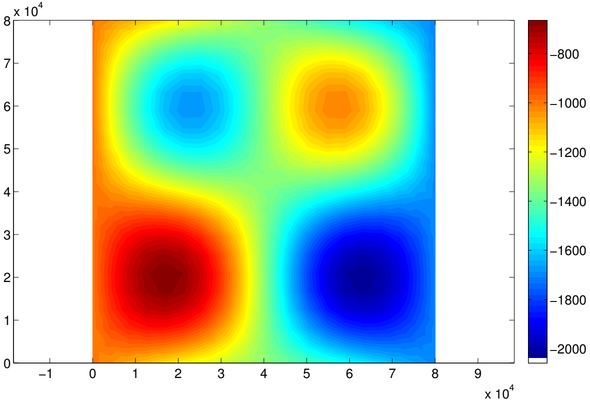
- thickness:

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

Parameterization

Use `ISMIPA.par` file to parameterize the model:

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



Parameterization

Use `ISMIPA.par` file to parameterize the model:

Friction: frozen bed
$\rightarrow$ Does not matter (use default value in `ISMIPA.par` file)

Rheology:

- Ice-flow parameter: $A = 10^{16}$ Pa^n a$^-1$
- Exponent in Glen's flow law: $n = 3$

Parameterization

Use `ISMIPA.par` file to parameterize the model:

Friction: frozen bed
$\rightarrow$ Does not matter (use default value in `ISMIPA.par` file)

Rheology:

- Ice-flow parameter: $B = \frac{1}{A^{1/n}}$ in Pa s$^{1/n}$

```
md.materials.rheology_B= ...
... 6.8067*10^7*ones(md.mesh.numberofvertices,1);
```

- Exponent in Glen's flow law:

```
md.materials.rheology_n= ...
... 3*ones(md.mesh.numberofelements,1);
```

Parameterization

Use `ISMIPA.par` file to parameterize the model:

Friction: frozen bed
$\rightarrow$ Does not matter (use default value in `ISMIPA.par` file)

Rheology:

- Ice-flow parameter: $B = \frac{1}{A^{1/n}}$ in Pa s$^{1/n}$

```
md.materials.rheology_B= ...
... 6.8067*10^7*ones(md.mesh.numberofvertices,1);
```

- Exponent in Glen's flow law:

```
md.materials.rheology_n= ...
... 3*ones(md.mesh.numberofelements,1);
```

Boundary condition:

- Ice Sheet default boundary conditions

Parameterization

Use `ISMIPA.par` file to parameterize the model:

Friction: frozen bed
$\rightarrow$ Does not matter (use default value in `ISMIPA.par` file)

Rheology:

- Ice-flow parameter: $B = \frac{1}{A^{1/n}}$ in Pa s$^{1/n}$

```
md.materials.rheology_B= ...
... 6.8067*10^7*ones(md.mesh.numberofvertices,1);
```

- Exponent in Glen's flow law:

```
md.materials.rheology_n= ...
... 3*ones(md.mesh.numberofelements,1);
```

Boundary condition:

- Ice Sheet default boundary conditions

```
md=SetIceSheetBC(md);
```

Extrusion

Use `extrude.m` to extrude the model:

```
>> help extrude
 EXTRUDE - vertically extrude a 2d mesh

     vertically extrude a 2d mesh and create corresponding 3d mesh.
     The vertical distribution can:
     - follow a polynomial law
     - follow two polynomial laws, one for the lower part and one for the upper part
     of the mesh
     - be discribed by a list of coefficients (between 0 and 1)

     Usage:
        md=extrude(md,numlayers,extrusionexponent);
        md=extrude(md,numlayers,lowerexponent,upperexponent);
        md=extrude(md,listofcoefficients);

     Example:
        md=extrude(md,8,3);
        md=extrude(md,8,3,2);
        md=extrude(md,[0 0.2 0.5 0.7 0.9 0.95 1]);

     See also:
     MODELEXTRACT, COLLAPSE
```

Extrusion

Use `extrude.m` to extrude the model:

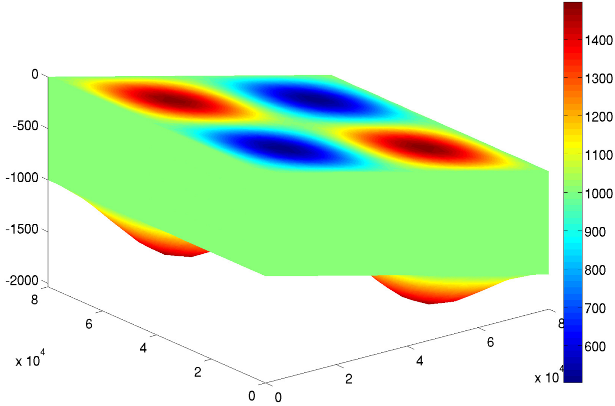
```
>> help extrude
 EXTRUDE - vertically extrude a 2d mesh

     vertically extrude a 2d mesh and create corresponding 3d mesh.
     The vertical distribution can:
     - follow a polynomial law
     - follow two polynomial laws, one for the lower part and one for the upper part
     of the mesh
     - be discribed by a list of coefficients (between 0 and 1)

     Usage:
        md=extrude(md,numlayers,extrusionexponent);
        md=extrude(md,numlayers,lowerexponent,upperexponent);
        md=extrude(md,listofcoefficients);

     Example:
        md=extrude(md,8,3);
        md=extrude(md,8,3,2);
        md=extrude(md,[0 0.2 0.5 0.7 0.9 0.95 1]);

     See also:
     MODELEXTRACT, COLLAPSE
```

```
md=extrude(md,5,1);
```

Extrusion

Use `extrude.m` to extrude the model:

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



Flow equation

Use `setflowequation.m` to specify the approximation :

```
>> help setflowequation
 SETELEMENTSTYPE - associate a solution type to each element

   This routine works like plotmodel: it works with an even number of inputs
   'hutter','macayeal','pattyn','stokes' and 'fill' are the possible options
   that must be followed by the corresponding exp file or flags list
   It can either be a domain file (argus type, .exp extension), or an array of element flags.
   If user wants every element outside the domain to be
   setflowequationd, add '¬' to the name of the domain file (ex: '¬Pattyn.exp');
   an empty string '' will be considered as an empty domain
   a string 'all' will be considered as the entire domain
   You can specify the type of coupling, 'penalties' or 'tiling', to use with the input ...
          'coupling'

   Usage:
      md=setflowequation(md,varargin)

   Example:
      md=setflowequation(md,'pattyn','Pattyn.exp','macayeal',md.mask.elementonfloatingice,'fill'
      md=setflowequation(md,'pattyn','Pattyn.exp',fill','hutter','coupling','tiling');
```

Flow equation

Use `setflowequation.m` to specify the approximation :

```
>> help setflowequation
 SETELEMENTSTYPE - associate a solution type to each element

   This routine works like plotmodel: it works with an even number of inputs
   'hutter','macayeal','pattyn','stokes' and 'fill' are the possible options
   that must be followed by the corresponding exp file or flags list
   It can either be a domain file (argus type, .exp extension), or an array of element flags.
   If user wants every element outside the domain to be
   setflowequationd, add '¬' to the name of the domain file (ex: '¬Pattyn.exp');
   an empty string '' will be considered as an empty domain
   a string 'all' will be considered as the entire domain
   You can specify the type of coupling, 'penalties' or 'tiling', to use with the input ...
          'coupling'

   Usage:
      md=setflowequation(md,varargin)

   Example:
      md=setflowequation(md,'pattyn','Pattyn.exp','macayeal',md.mask.elementonfloatingice,'fill'
      md=setflowequation(md,'pattyn','Pattyn.exp',fill','hutter','coupling','tiling');
```

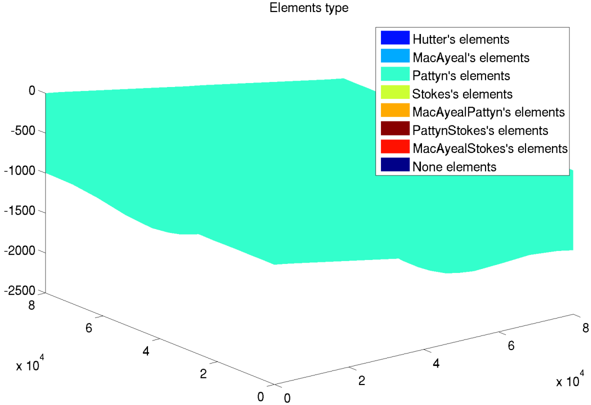
```
md=setflowequation(md,'pattyn','all');
```

Flow equation

Use `setflowequation.m` to specify the approximation :

```
md=setflowequation(md,'pattyn','all');
```

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



Boundary conditions

Ice frozen on the bedrock: zero velocity

```
%Create dirichlet on the bed only
md.diagnostic.spcvx=NaN*ones(md.mesh.numberofvertices,1);
md.diagnostic.spcvy=NaN*ones(md.mesh.numberofvertices,1);
md.diagnostic.spcvz=NaN*ones(md.mesh.numberofvertices,1);
pos=find(md.mesh.vertexonbed);
md.diagnostic.spcvx(pos)=0;
md.diagnostic.spcvy(pos)=0;
md.diagnostic.spcvz(pos)=0;
```

Boundary conditions

Periodic boundary conditions: `md.diagnostic.vertex_pairing`

```
%Create MPCs to have periodic boundary conditions
posx=find(md.mesh.x==0);
posx2=find(md.mesh.x==max(md.mesh.x));
posy=find(md.mesh.y==0 & md.mesh.x≠0 & ...
      md.mesh.x≠max(md.mesh.x)); %Don't take the same grids twice
posy2=find(md.mesh.y==max(md.mesh.y) & md.mesh.x≠0 & ...
      md.mesh.x≠max(md.mesh.x));
md.diagnostic.vertex_pairing=[posx,posx2;posy,posy2];
```

Solve model

Set cluster and print convergence while running:

```
%Set cluster and print messages
md.cluster=generic('name',oshostname(),'np',2);
md.verbose=verbose('convergence',true,'solution',true);
```

Solve model

Set cluster and print convergence while running:

```
%Set cluster and print messages
md.cluster=generic('name',oshostname(),'np',2);
md.verbose=verbose('convergence',true,'solution',true);
```

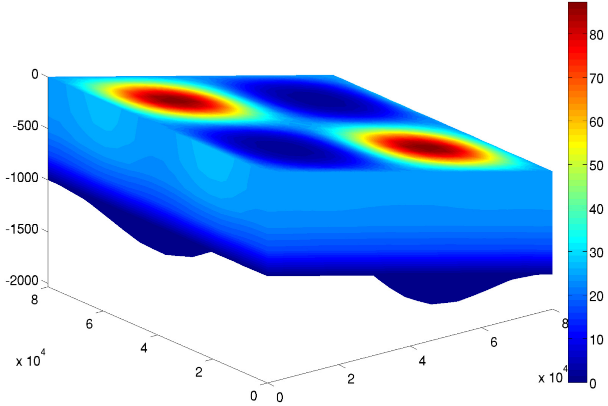
```
md=solve(md,DiagnosticSolutionEnum);
```

Results

```
plotmodel(md,'data',md.results.DiagnosticSolution.Vx)
```

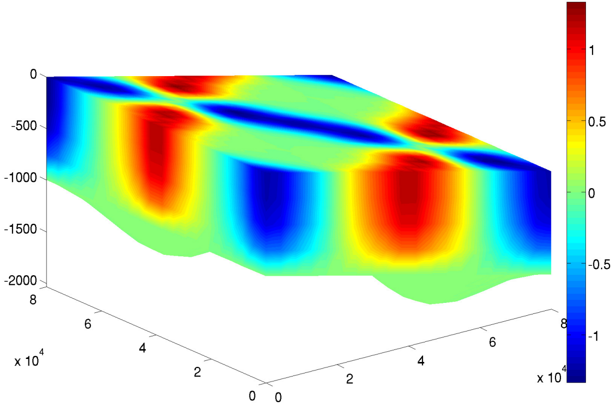
Results

```
plotmodel(md,'data',md.results.DiagnosticSolution.Vx)
```



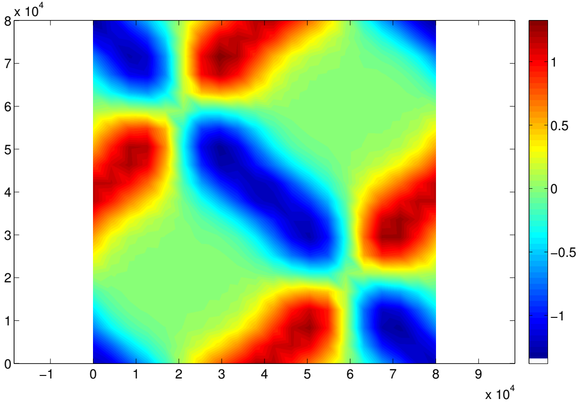
Results

```
plotmodel(md,'data',md.results.DiagnosticSolution.Vx)
```



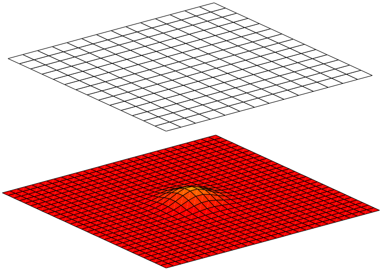
Results

```
plotmodel(md,'data',md.results.DiagnosticSolution.Vy,'layer',md.mesh
```



Test F

Square ice sheet flowing over a sinusoidal bed



- Sinusoidal bedrock but flat surface
- Ice frozen or sliding on the bed
- Periodic boundary conditions
- Transient model until steady-state

Transient

```
>> md.transient

ans =

   transient solution parameters:
      isprognostic       : 1     -- indicates if a prognostic solution is used in the transient
      isthermal          : 0     -- indicates if a thermal solution is used in the transient
      isdiagnostic       : 1     -- indicates if a diagnostic solution is used in the transient
      isgroundingline    : 0     -- indicates if a groundingline migration is used in the transient
      requested_outputs  : N/A   -- list of additional outputs requested
```

```
>> md.prognostic

ans =

   Prognostic solution parameters:
      spcthickness         : (2000x1)   -- thickness constraints (NaN means no constraint)
      hydrostatic_adjustment : 'Absolute' -- adjustment of ice shelves surface and bed elevations: ...
              'Incremental' or 'Absolute'
      stabilization        : 1          -- 0->no, 1->artificial_diffusivity, 3->discontinuous Galerkin

   Penalty options:
      penalty_factor       : 3          -- offset used by penalties: penalty = Kmax*10^offset
      vertex_pairing       : (200x2)    -- pairs of vertices that are penalized
```

```
>> md.timestepping

ans =

   timestepping parameters:
      time_step            : 4          -- length of time steps [yrs]
      final_time           : 80         -- final time to stop the simulation [yrs]
      time_adapt           : 0          -- use cfl condition to define time step ? (0 or 1)
      cfl_coefficient      : 0.5        -- coefficient applied to cfl condition
```

Parameterization file

```
disp('      creating geometry');
L=100000;
alpha=3*pi/180;
md.geometry.surface=-md.mesh.x*tan(alpha);
md.geometry.bed=md.geometry.surface-1000+100*exp(-((md.mesh.x-L/2).^2+(md.mesh.y-L/2).^2)/(10000^2));
md.geometry.thickness=md.geometry.surface-md.geometry.bed;

disp('      creating drag');
md.friction.coefficient=sqrt(365.25*24*3600/(1000*2.140373*10^-7))*ones(md.mesh.numberofvertices,1);
md.friction.p=ones(md.mesh.numberofelements,1);
md.friction.q=zeros(md.mesh.numberofelements,1);

disp('      creating flow law paramter');
md.materials.rheology_B=(1/(2.140373*10^-7/(365.25*24*3600)))*ones(md.mesh.numberofvertices,1);
md.materials.rheology_n=1*ones(md.mesh.numberofelements,1);

disp('      boundary conditions for diagnostic model');
md=SetIceSheetBC(md);

%Field 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);
```

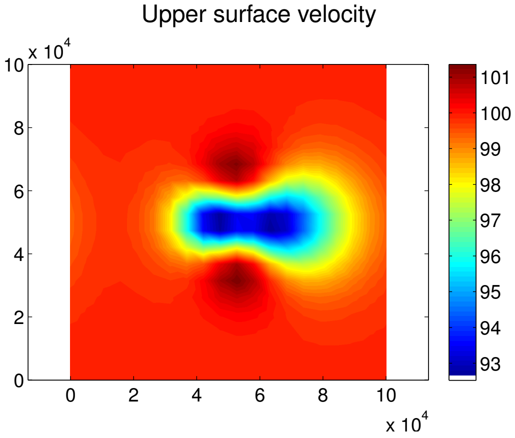
Runme file

```
%Create model
md=model;

%Create mesh
L=100000; %in m
nx=20; %numberof nodes in x direction
ny=20;
md=squaremesh(md,L,L,nx,ny);

md=setmask(md,'',''); %ice sheet test
md=parameterize(md,'./ISMIPF.par');
md=extrude(md,5,1);
md=setflowequation(md,'pattyn','all');

%Boundary conditions
md.diagnostic.spcvx(:)=NaN;
md.diagnostic.spcvy(:)=NaN;
md.diagnostic.spcvz(:)=NaN;
%Create dirichlet on the bed if no slip
pos=find(md.mesh.vertexonbed);
md.diagnostic.spcvx(pos)=0;
md.diagnostic.spcvy(pos)=0;
md.diagnostic.spcvz(pos)=0;

%Create MPCs to have periodic boundary conditions
posx=find(md.mesh.x==0);
posx2=find(md.mesh.x==max(md.mesh.x));
posy=find(md.mesh.y==0); %Don't take the same grids two times
posy2=find(md.mesh.y==max(md.mesh.y));
md.diagnostic.vertex_pairing=[posx,posx2;posy,posy2];
md.prognostic.vertex_pairing=[posx,posx2;posy,posy2];

%Transient parameters
md.timestepping.time_step=4;
md.timestepping.final_time=4*20;
md.transient.isthermal=0;

%Compute the diagnostic
md.cluster=generic('name',oshostname(),'np',2);
md.verbose=verbose('convergence',true,'solution',true);
md=solve(md,TransientSolutionEnum);
```

Results

Results after 20 iterations of 4 years:

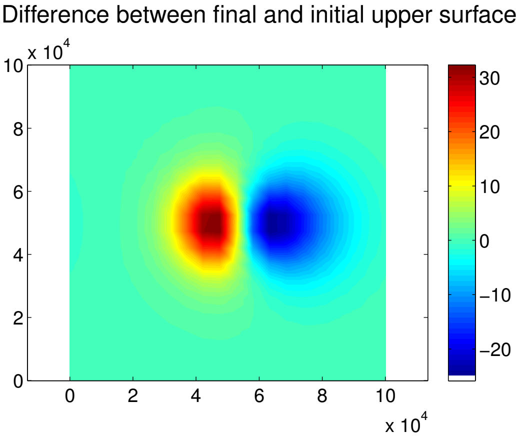
```
1    >> plotmodel(md,'data',md.results.Transient(20).Vel,'layer',md.mesh.numberoflayers)
```



Results

Results after 20 iterations of 4 years:

```
>> plotmodel(md,'data',md.results.Transient(20).Surface-md.geometry.surface,'layer',md.mesh.numberoflayers)
```



Results

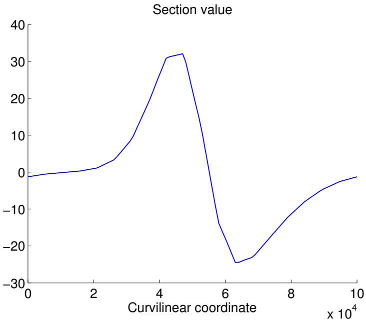
Results after 20 iterations of 4 years:

```
>> plotmodel(md,'data',md.results.Transient(20).Surface-md.geometry.surface,'layer',md.mesh.numberoflayers,...
   ...'sectionvalue','Profile.exp')
```

Define profile in `Profile.exp`

```

## Name:icefront
## Icon:0
# Points Count  Value
2 1.
# X pos Y pos
0 50000
```

Difference between final and initial surface on a section




Thanks!