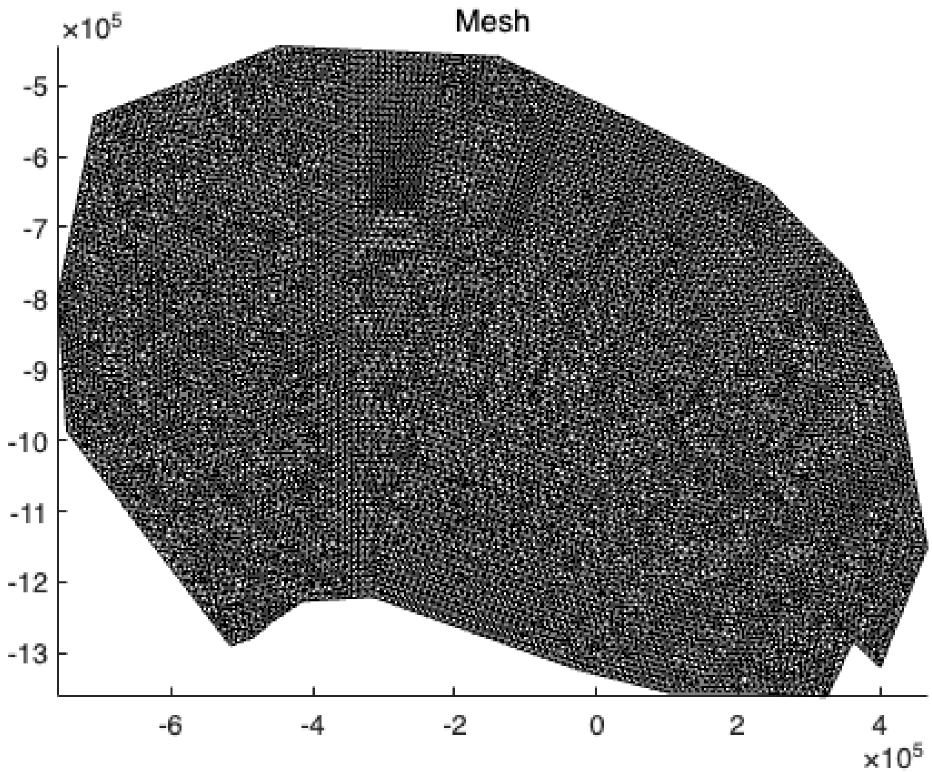
```
%%%%%%%%% Step 1: Create mesh %%%%%%%%%
% First, we need to create a mesh of the domain and refine it to capture
the observed ice flow velocity. As you have seen in the mesh tutorial, we
% first create a uniform mesh, interpolate the surface velocity and use it
as a metric to refine our initial mesh. Most of the code is provided to
% you. You only need to find the observation error so that the mesh has
about 5,000 elements .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Mesh parameters
domain = 'Ross_Ice_shelf.exp' % may be just need to change the domain
anywhere in Antarctica
```

```
domain =
'Ross_Ice_shelf.exp'
```

```
hinit=10000;    % element size for the initial mesh
hmax=30000;        % maximum element size of the final mesh
hmin=4000;        % minimum element size of the final mesh
gradation=1.7;    % maximum size ratio between two neighboring elements
err= 8;            % maximum error between interpolated and control field

% Generate an initial uniform mesh (resolution = hinit m)
cd '/Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Pig'
md=bamg(model,'domain',domain,'hmax',hinit);
```

```
At least one contour was not correctly oriented and has been re-oriented
Construction of a mesh from a given geometry
```

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



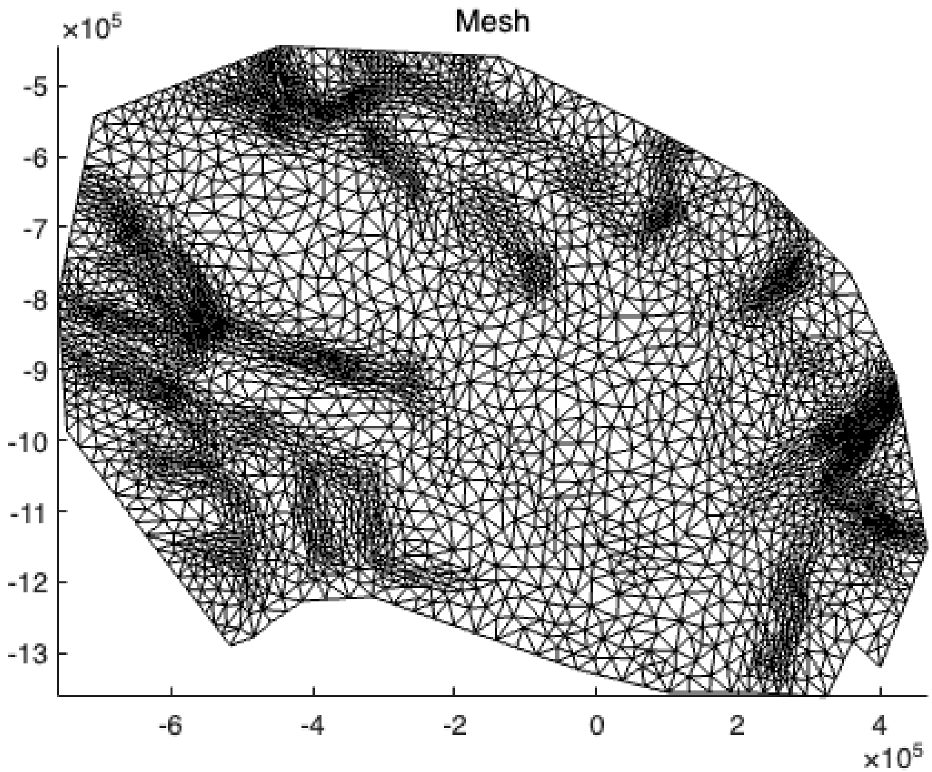

```
% Interpolate velocity so that it can be used as a metric
% interpMouginotAnt2019
[vx_obs, vy_obs]= interpMouginotAnt2019 (md.mesh.x,md.mesh.y,...
    '/Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/
antarctic_ice_vel_phase_map_v01.nc');
```

```
   -- Mouginot 2019: loading velocities
   -- Mouginot 2019: interpolating
```

```
vel_obs=sqrt(vx_obs.^2+vy_obs.^2);

% Adapt the mesh to minimize error in velocity interpolation
md=bamg(md,'hmax',hmax,'hmin',hmin,'gradation',gradation,'field',vel_obs,'er
r',err);
```

```
Anisotropic mesh adaptation
   new number of triangles = 6975
```

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




```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2: Parameterize model
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% We need to provide all the model parameters to intialize the model
including:
% - set the ice mask (using a level-set)
% - interpolate the geometry from BedMachine (bed, surface, base,
thickness, grounding line)
% - interpolate observed velocities to prepare the inversion
% - initialize friction parameters and rheology parameters (Budd sliding
law )
% - set boundary conditions
% - set flow equation
% - choose a model name
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Step-2.1 : set the ice mask (using a level-set)
% -----------------------------------------

% read the msak from the Bedmachine dataset
mask = interpBedmachineAntarctica(md.mesh.x,md.mesh.y,'mask','linear',...
    '/Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/
BedMachineAntarctica-v3.nc');
```

```
-- BedMachine Antarctica version: /Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/BedMachi
-- BedMachine Antarctica: loading mask
-- BedMachine Antarctica: interpolating mask
   -- Interpolation method: linear
```

```
% initialize the mesh of ice mask == same as the number of vertices
md.mask.ice_levelset = ones(md.mesh.numberofvertices,1);
md.mask.ice_levelset(mask>1.5) = -1;                                   % the
ice mask should be >1.5 in bedmachine
md.mask.ice_levelset = reinitializelevelset(md,md.mask.ice_levelset); %
reinitialize level set


% Step-2.2: interpolate the geometry from BedMachine (bed, surface, base,
thickness, grounding line)
%
----------------------------------------------------------------------------
--------------------

% Surface and Bed information from BedMachine
md.geometry.surface =
interpBedmachineAntarctica(md.mesh.x,md.mesh.y,'surface','linear',...
    '/Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/
BedMachineAntarctica-v3.nc');
```

```
-- BedMachine Antarctica version: /Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/BedMachi
-- BedMachine Antarctica: loading surface
-- BedMachine Antarctica: interpolating surface
   -- Interpolation method: linear
```

```
md.geometry.bed =
interpBedmachineAntarctica(md.mesh.x,md.mesh.y,'bed','linear',...
    '/Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/
BedMachineAntarctica-v3.nc');
```

```
-- BedMachine Antarctica version: /Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/BedMachi
-- BedMachine Antarctica: loading bed
-- BedMachine Antarctica: interpolating bed
   -- Interpolation method: linear
```

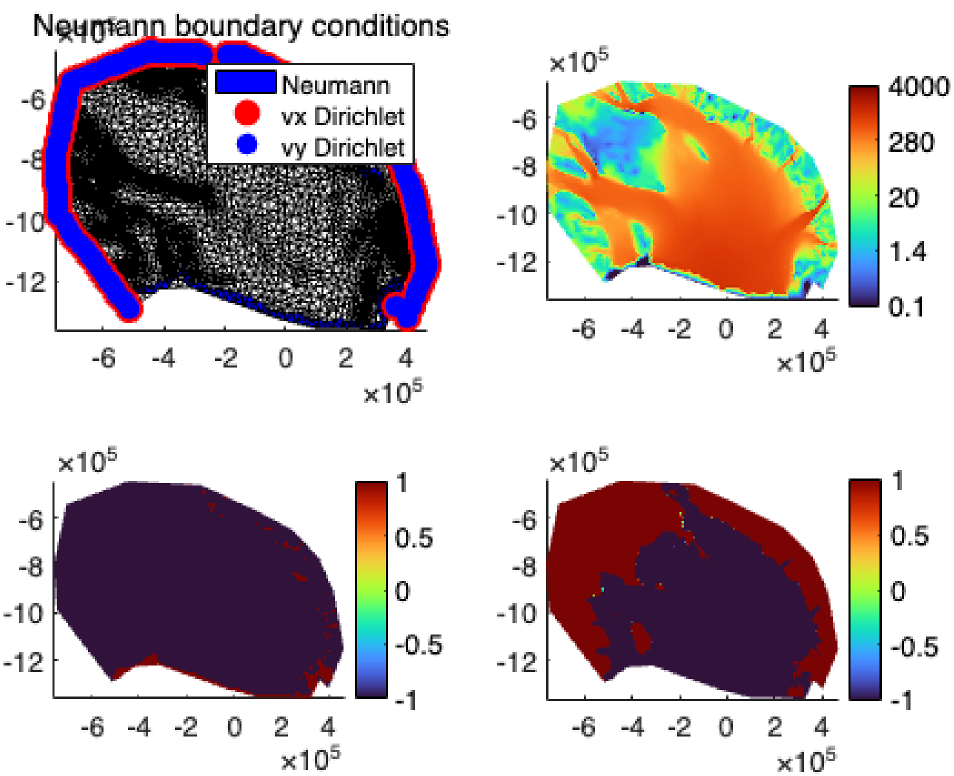
```
md.geometry.base = md.geometry.bed; % ice-shelf base same as the bed

% calculation for using hydrostatic equilibrium for the ice shelf
rho_water = 1028.9;
di = md.materials.rho_ice/rho_water;
pos = find(-di/(1-di)*md.geometry.surface > md.geometry.bed);
md.geometry.base(pos) = di/(di-1)*md.geometry.surface(pos);
md.geometry.thickness = md.geometry.surface - md.geometry.base;
md.mask.ocean_levelset = md.geometry.thickness+md.geometry.bed/di;
```

```
% Step-2.3 : interpolate observed velocities to prepare the inversion
% --------------------------------------------------------------

% Observed velocity
% interpolate observed velocity from the velocity netcdf file
[md.inversion.vx_obs, md.inversion.vy_obs] = interpMouginotAnt2019
(md.mesh.x,md.mesh.y,...
    '/Users/rishi/Desktop/ISSM-macOS-Silicon-MATLAB/examples/Data/
antarctic_ice_vel_phase_map_v01.nc');
```

```
-- Mouginot 2019: loading velocities
-- Mouginot 2019: interpolating
```

```
md.inversion.vx_obs(isnan(md.inversion.vx_obs)) = 0; % make all the nan
values to be zero
md.inversion.vy_obs(isnan(md.inversion.vy_obs)) = 0;
md.inversion.vel_obs = sqrt(md.inversion.vx_obs.^2 +
md.inversion.vy_obs.^2); % compute velocity from vx and vy


% Step-2.4 : initialize friction parameters and rheology parameters
% --------------------------------------------------------------

% initialize Sliding parameters (uniform 200 by default)
md.friction.coefficient = 200 * ones(md.mesh.numberofvertices, 1);
md.friction.p = ones(md.mesh.numberofelements, 1);
md.friction.q = ones(md.mesh.numberofelements, 1);

% Initialize Rheology parameters, assuming ice is at -10°C
md.materials.rheology_B = cuffey(273.15-10) *
ones(md.mesh.numberofvertices, 1);
md.materials.rheology_n = 3 * ones(md.mesh.numberofelements, 1);

% Other parameters (just ignore)
md.stressbalance.referential = NaN(md.mesh.numberofvertices,6);
md.stressbalance.loadingforce = zeros(md.mesh.numberofvertices,3);

% Step-2.5 : Set Boundary conditions
% ----------------------------------

% Set Boundary conditions: constrain inflow velocities only
md.stressbalance.spcvx = NaN(md.mesh.numberofvertices,1);
md.stressbalance.spcvy = NaN(md.mesh.numberofvertices,1);
md.stressbalance.spcvz = NaN(md.mesh.numberofvertices,1);
pos = find(md.mesh.vertexonboundary & md.mask.ocean_levelset>0);
md.stressbalance.spcvx(pos) = md.inversion.vx_obs(pos);
md.stressbalance.spcvy(pos) = md.inversion.vy_obs(pos);

% Step-2.6 : Set flowequation as SSA
```

```
% -----------------------------------
md = setflowequation(md,'SSA','all');

% Model name
md.miscellaneous.name = 'Ross_Inversion';
save ./Models/Ross_Parameterized md;

% Plot some parameters
plotmodel(md,'data','BC', ... % boundary conditions
    'data',md.inversion.vel_obs,'log#2',10,'caxis#2',[0.1 4000],... %
observed velocity
    'data',md.mask.ice_levelset,'caxis#3',[-1 1], ... % masked ice and no
ice
    'data',md.mask.ocean_levelset,'caxis#4',[-1 1]) % masked ice and ocean
```




```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 3: Infer ice shelf rheology
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Our initial guess for the ice rheology prefactor (B) needs to be refined
in order to match the observed velocities. As you have seen in the
% inversion tutorial, we need to:
%
% Set-up the inversion parameters
% Extract the ice shelf area so that we only perform the inversion on a
small section of the domain
```

```
% Invert for B on the ice shelf
% Plot the results
% Update the "bigger" model with the rheology prefactor from the extracted
model

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

% We are going to use the sum of 3 cost functions:
% 101, which is our regular least squares fit
% 103, the difference in log scale
% 501, Tikhonov regularization, which penalizes strong gradients in B

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


% Control general
md.inversion= m1qn3inversion(md.inversion);
md.inversion.iscontrol = 1; % controlled experiments
md.inversion.maxsteps = 80; % maximum steps
md.inversion.maxiter = 80;  % maximum interation
md.inversion.dxmin = 0.1;
md.inversion.gttol = 1.0e-6;

% Cost functions
md.inversion.cost_functions = [101 103 502];
md.inversion.cost_functions_coefficients= ones(md.mesh.numberofvertices,3);
md.inversion.cost_functions_coefficients(:,1) = 1;
md.inversion.cost_functions_coefficients(:,2) = 0.001;
md.inversion.cost_functions_coefficients(:,3) = 1e-18;
pos=find(md.inversion.vx_obs==0 | md.mask.ice_levelset>0);
md.inversion.cost_functions_coefficients(pos,1:2)=0;

% Controls
md.inversion.control_parameters={'MaterialsRheologyBbar'};
md.inversion.min_parameters = md.materials.rheology_B;
md.inversion.max_parameters = md.materials.rheology_B;
pos = find(md.mask.ocean_levelset<0);
md.inversion.min_parameters(pos) = cuffey(273);
md.inversion.max_parameters(pos) = cuffey(200);


% Solve
md.verbose=verbose('control',true);
md.cluster=generic('name',oshostname,'np',2);
mds = extract(md,md.mask.ocean_levelset<0); % only for the ice shelf
```

```
NOTE: using observed velocities to create constraints along new boundary
```

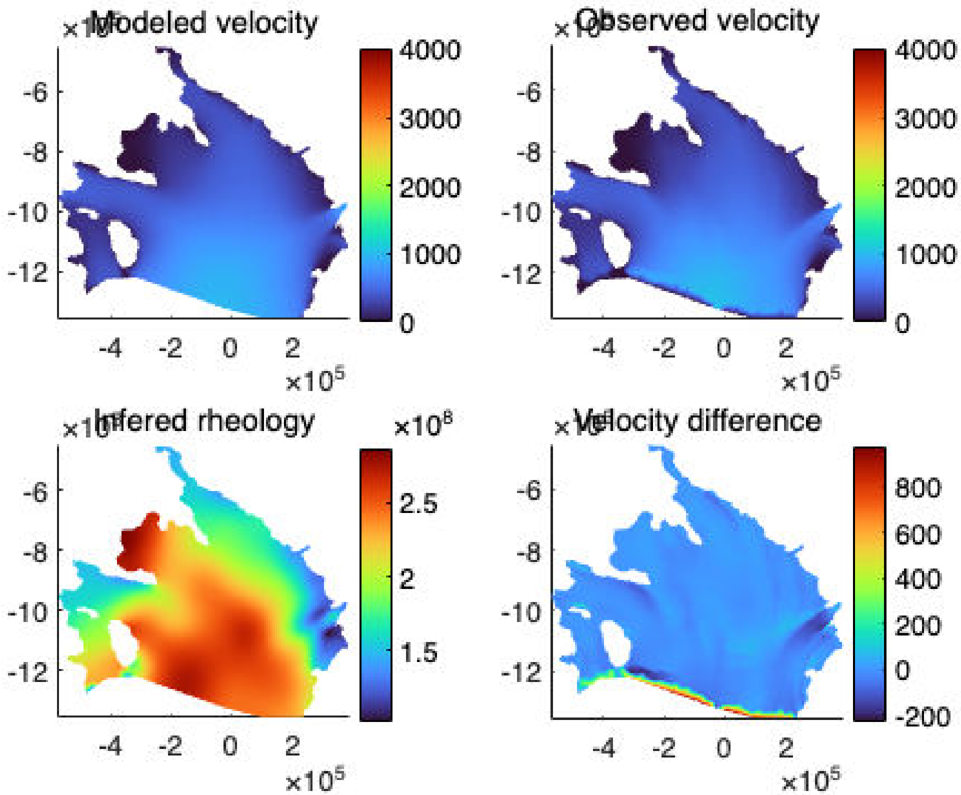
```
mds=solve(mds,'Stressbalance'); % solving stress balance only for ice shelf
```

```
launching solution sequence

Ice-sheet and Sea-level System Model (ISSM) version  4.24
(website: http://issm.jpl.nasa.gov forum: https://issm.ess.uci.edu/forum/)

call computational core:
   Initialize M1QN3 parameters
   Computing initial solution
```

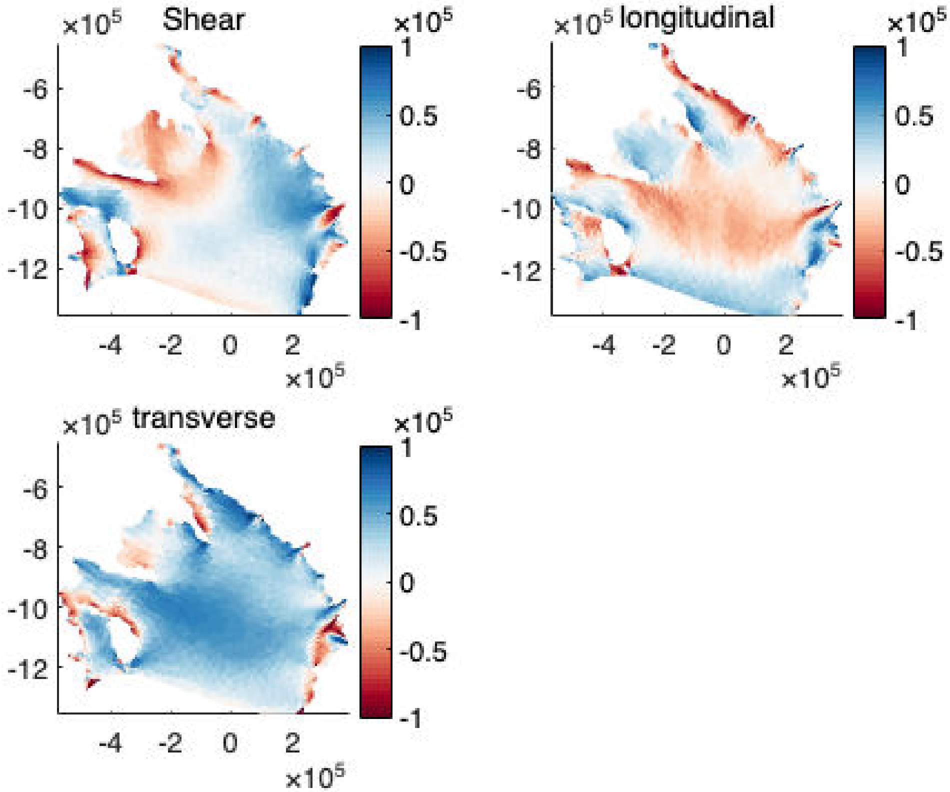
Iter	Cost function		Grad. norm	List of contributions		
1	f(x)=	102.28	8.98e-08	101.6	0.7181	2.175e-31
2	f(x)=	2.4804	2.95e-09	1.625	0.1104	0.7448
3	f(x)=	2.2279	2.56e-09	1.529	0.1074	0.5917
4	f(x)=	1.2407	7.11e-10	1.036	0.08482	0.1194
5	f(x)=	1.1276	6.99e-10	0.9483	0.07652	0.1028
6	f(x)=	1.0128	5.15e-10	0.8595	0.06692	0.08635
7	f(x)=	0.92638	8.72e-10	0.7749	0.05452	0.09692
8	f(x)=	0.88949	3.7e-10	0.7643	0.05284	0.07235
9	f(x)=	0.87954	2.59e-10	0.7613	0.05271	0.06556
10	f(x)=	0.85502	1.91e-10	0.747	0.05072	0.0573
11	f(x)=	0.83688	2.26e-10	0.7341	0.0484	0.05436
12	f(x)=	0.82243	4.32e-10	0.7183	0.04476	0.05932
13	f(x)=	0.8133	1.44e-10	0.7177	0.0446	0.05102
14	f(x)=	0.81029	1.24e-10	0.717	0.04449	0.0488
15	f(x)=	0.80317	1.6e-10	0.7132	0.04351	0.04644
16	f(x)=	0.7988	3.66e-10	0.7078	0.04162	0.04939
17	f(x)=	0.7942	1.08e-10	0.7075	0.04158	0.04514
18	f(x)=	0.79231	7.34e-11	0.7063	0.04124	0.04476
19	f(x)=	0.78941	8.58e-11	0.7041	0.0405	0.04481
20	f(x)=	0.7861	8.45e-11	0.7017	0.03974	0.04469
21	f(x)=	0.78551	3.16e-10	0.6979	0.03852	0.04912
22	f(x)=	0.78262	5.04e-11	0.6989	0.03888	0.04488
23	f(x)=	0.78244	3.99e-11	0.699	0.03895	0.04452
24	f(x)=	0.78149	4.47e-11	0.6985	0.03892	0.04403
25	f(x)=	0.78065	5.96e-11	0.698	0.03873	0.04389
26	f(x)=	0.77981	5.95e-11	0.6973	0.03833	0.04416
27	f(x)=	0.77948	2.86e-11	0.6973	0.03828	0.04387
28	f(x)=	0.77934	2.83e-11	0.6974	0.03823	0.04366
29	f(x)=	0.77918	3.69e-11	0.6977	0.03812	0.04337
30	f(x)=	0.77915	5.76e-11	0.6982	0.03795	0.04296
31	f(x)=	0.77899	2.33e-11	0.6983	0.03792	0.04277
32	f(x)=	0.77887	1.94e-11	0.6983	0.03787	0.04273
33	f(x)=	0.77867	2.23e-11	0.6982	0.03777	0.04271
34	f(x)=	0.77838	5.63e-11	0.698	0.03755	0.04278
35	f(x)=	0.77819	1.74e-11	0.698	0.0375	0.04268
36	f(x)=	0.7781	1.18e-11	0.698	0.03746	0.04264
37	f(x)=	0.77802	1.51e-11	0.698	0.03741	0.04257
38	f(x)=	0.77793	1.86e-11	0.6981	0.03737	0.04247
39	f(x)=	0.77788	3.18e-11	0.6982	0.03731	0.04234
40	f(x)=	0.77781	9.54e-12	0.6982	0.03731	0.04229
41	f(x)=	0.77779	9.3e-12	0.6982	0.03733	0.04226
42	f(x)=	0.77777	1.14e-11	0.6982	0.03735	0.0422
43	f(x)=	0.77779	2.8e-11	0.6982	0.03735	0.0422
44	f(x)=	0.7778	1.13e-11	0.6982	0.03736	0.04219
45	f(x)=	0.7778	1.13e-11	0.6982	0.03736	0.04219
46	f(x)=	0.7778	1.11e-11	0.6982	0.03736	0.04219
47	f(x)=	0.7778	1.07e-11	0.6982	0.03736	0.0422
48	f(x)=	0.7778	1.06e-11	0.6982	0.03736	0.0422
49	f(x)=	0.7778	1.07e-11	0.6982	0.03736	0.0422
50	f(x)=	0.7778	1.09e-11	0.6982	0.03736	0.0422
51	f(x)=	0.77781	1.1e-11	0.6982	0.03736	0.0422
52	f(x)=	0.77781	1.11e-11	0.6982	0.03736	0.0422

```
  53  f(x)=   0.77781   1.12e-11     0.6982   0.03736    0.0422
  54  f(x)=   0.77781   1.12e-11     0.6982   0.03736    0.0422
  55  f(x)=   0.77781   1.13e-11     0.6982   0.03736    0.0422
  56  f(x)=   0.77781   1.13e-11     0.6982   0.03736    0.0422
  57  f(x)=   0.77781   1.13e-11     0.6982   0.03736    0.0422
  58  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  59  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  60  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  61  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  62  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  63  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  64  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  65  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  66  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  67  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  68  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  69  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  70  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  71  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  72  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  73  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  74  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  75  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  76  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  77  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  78  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  79  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422
  80  f(x)=   0.77781   1.14e-11     0.6982   0.03736    0.0422

   Maximum number of function calls exceeded
   preparing final solution
write lock file:

   FemModel initialization elapsed time:   0.010003
   Total Core solution elapsed time:       9.59799
   Linear solver elapsed time:             7.34151 (76%)

   Total elapsed time: 0 hrs 0 min 9 sec
```

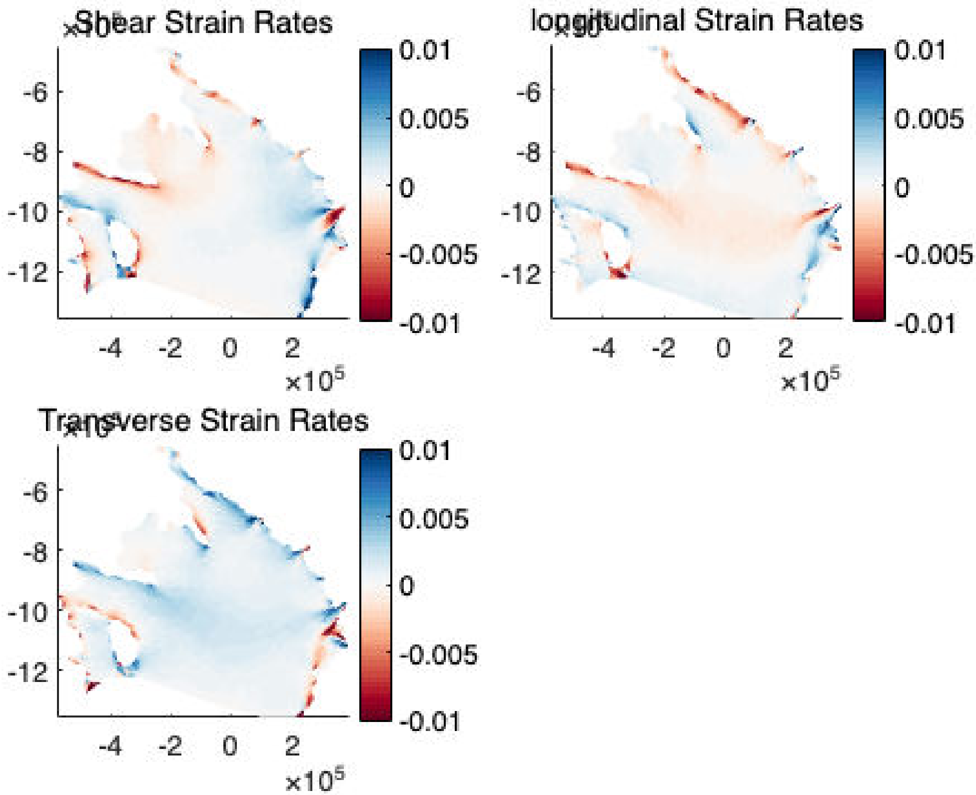
```
% Plot the results of the inversion

plotmodel(mds,...
'data',mds.results.StressbalanceSolution.Vel,'title','Modeled velocity','colormap','turbo',...
'data',mds.inversion.vel_obs,'title','Observed velocity','colormap','turbo',...
'data',mds.results.StressbalanceSolution.MaterialsRheologyBbar,'title','Infered rheology',...
'data',mds.results.StressbalanceSolution.Vel-mds.inversion.vel_obs,'title','Velocity difference',...
'caxis#1',[0 4000],'caxis#2',[0 4000])
```



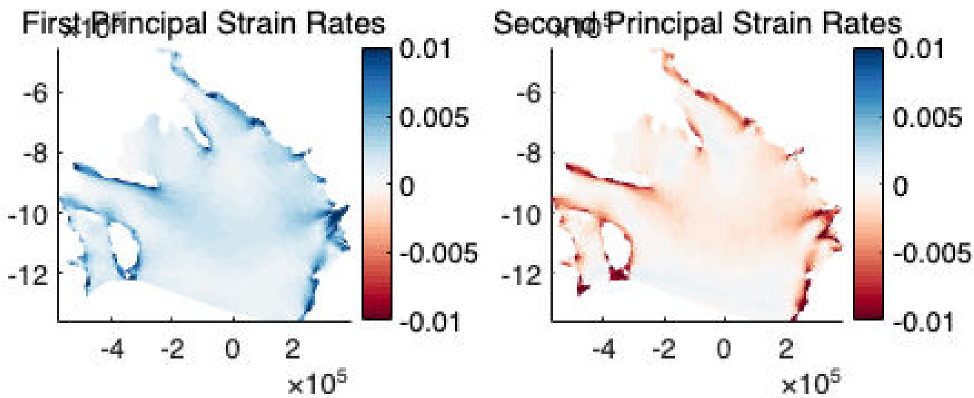

```
%Get velocities from solutions
vx = mds.results.StressbalanceSolution.Vx;
vy = mds.results.StressbalanceSolution.Vy;
vel = mds.results.StressbalanceSolution.Vel;
mds = mechanicalproperties(mds, vx, vy); % will get the output in
mds.results
mds.results.deviatoricstress;

% plotting the Deviatoric Stresses
plotmodel(mds,'data',mds.results.deviatoricstress.xy,'title','Shear',...
'data',mds.results.deviatoricstress.xx,'title','longitudinal',...
'data',mds.results.deviatoricstress.yy,'title','transverse','caxis#all',
[-1e5 1e5]);
colormap(brewermap(50, 'RdBu'));
```




```
% plotting the Strain Rates
plotmodel(mds,'data',mds.results.strainrate.xy,'title','Shear Strain
Rates',...
'data',mds.results.strainrate.xx,'title','longitudinal Strain Rates',...
'data',mds.results.strainrate.yy,'title','Transverse Strain Rates',...
'caxis#all',[-0.01 0.01]);
colormap(brewermap(50, 'RdBu'));
```

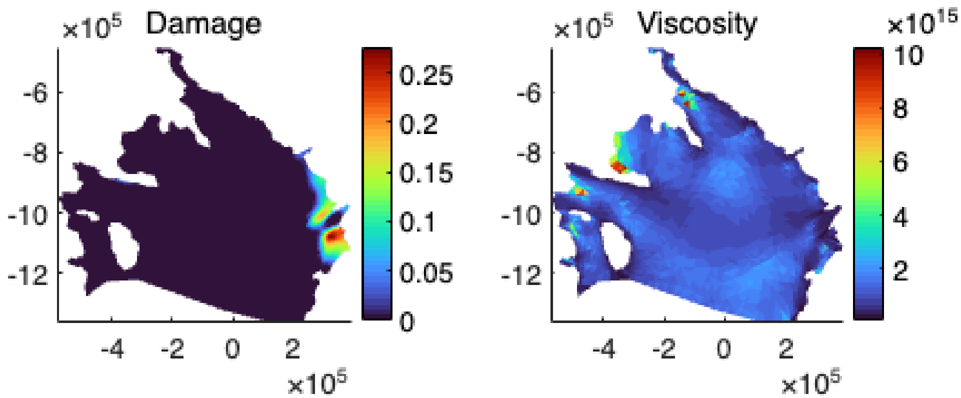



```
% plotting the Pricnipal Strain Rates
plotmodel(mds,'data',mds.results.strainrate.principalvalue1,'title','First
Principal Strain Rates','caxis#1',[-0.01 0.01],...
'data',mds.results.strainrate.principalvalue2,'title','Second Principal
Strain Rates','caxis#2',[-0.01 0.01])
colormap(brewermap(50,'RdBu'))
```

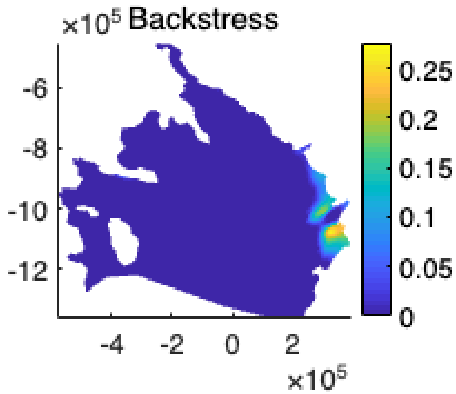



```
% Estimating Damage parameter from the inversion
mds.results.StressbalanceSolution.D = damagefrominversion(mds);
plotmodel(mds,'data',mds.results.StressbalanceSolution.D,'title','Damage',...
    'data',mds.results.viscosity,'title','Viscosity');
```

```
data provided is a struct with the following fields:
   1: nu
```



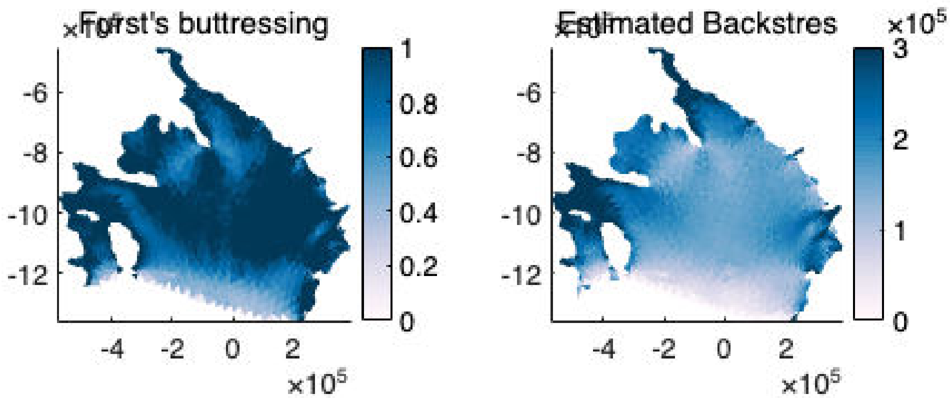

```
% Compute Backstress using Borstad's method
mds.results.StressbalanceSolution.Backstress = backstressfrominversion(mds);

plotmodel(mds,'data',mds.results.StressbalanceSolution.D,'title','Backstress
','colormap','parula');
```




```
%find elements that are floating ice
pos_floating = find(min(md.mask.ocean_levelset(md.mesh.elements),[],2)<0 &
min(md.mask.ice_levelset(md.mesh.elements),[],2)<0);

%Recover stresses
vx = mds.results.StressbalanceSolution.Vx;
vy = mds.results.StressbalanceSolution.Vy;
vel = mds.results.StressbalanceSolution.Vel;
md = mechanicalproperties(mds, vx, vy);

%Constants
g = md.constants.g;
rho_i = mds.materials.rho_ice;
rho_w = mds.materials.rho_water;

%Element averaged thickness
H = mean(mds.geometry.thickness(md.mesh.elements),2);
depth = mean(mds.geometry.base(md.mesh.elements),2)-
mean(md.geometry.bed(mds.mesh.elements),2);
vx = mean(vx(mds.mesh.elements),2);
vy = mean(vy(mds.mesh.elements),2);
vel = mean(vel(mds.mesh.elements),2);

%Allocate Kn
Kn = NaN(length(mds.mesh.elements),1);
```

```
buttress = NaN(mds.mesh.numberofelements,1);

for el=pos_floating'

    %Build stress tensor
    tau_xx = mds.results.deviatoricstress.xx(el);
    tau_yy = mds.results.deviatoricstress.yy(el);
    tau_xy = mds.results.deviatoricstress.xy(el);
    R = [2*tau_xx+tau_yy   tau_xy;tau_xy   2*tau_yy+tau_xx];

    %Decide on outward pointing unit vector
    if 0 %"Flow buttressing" - the supplement
        vector = [vx(el); vy(el)];
    else %"Max buttressing" - the main text
        vector = md.results.deviatoricstress.principalaxis2(el,:)';
    end
    n = vector./sqrt(vector(1)^2+vector(2)^2);

    %Buildsigma_nn and then Kn
    N = n'*R*n;
    N0 = 0.5*g*rho_i.*(1-rho_i./rho_w).*H(el);
    Kn(el) = 1 - N/N0;    % Furst Buttressing Number
    Backstress_Furst(el) = N0 - N; % Estimated Backstress (Pa)
end
```

```
plotmodel(mds,'data',Kn,'title', 'Furst''s buttressing','caxis#1',[0 1], ...
              'data',Backstress_Furst,'title', 'Estimated
Backstres','caxis#2',[0 3e5]);
colormap(brewermap(50,'PuBu'))
```




```
%save ./Models/Ross_Inversion_Bedmachine mds;
```