Ice Sheet System model
User interface

Eric Larour\textsuperscript{1}, Eric Rignot\textsuperscript{1,3}, Mathieu Morlighem\textsuperscript{1,2}, Hélène Seroussi\textsuperscript{1,2} Chris Borstad\textsuperscript{1}, Feras Habbal\textsuperscript{1,3}, Daria Halkides\textsuperscript{1,4}, Behnaz Khakbaz\textsuperscript{1}, John Schiermeier\textsuperscript{1}, Nicole Schlegel\textsuperscript{1}

\textsuperscript{1}Jet Propulsion Laboratory - California Institute of Technology
\textsuperscript{2}Laboratoire MSSMat, École Centrale Paris, France
\textsuperscript{3}University of California, Irvine
\textsuperscript{4}Joint Institute for Regional Earth System Science & Engineering, UCLA
Startup file

- **startup.m** must be present when matlab is launched
- Automatically executed by matlab at “start up” to load all ISSM tools

```matlab
1 % Recover ISSM_TIER and USERNAME
2 ISSM_TIER=getenv('ISSM_TIER');
3 USERNAME =getenv('USER');
4 if (isempty(ISSM_TIER)),
5     error('issmdir error message: 'ISSM_TIER' environment variable is empty! You should define ISSM_TIER in your .cshrc or .bashrc!');
6 end
7
8 % Now add all issm code paths necessary to run issm smoothly.
9 % We capture the error output, so that we can warn the user to update
10 % the variable ISSM_TIER in this file, in case it is not correctly setup.
11
12 % ISSM path
13 addpath([ISSM_TIER 'src/m/utils']); % loads recursivepath
14 addpath([ISSM_TIER 'doc']);
15 addpath([ISSM_TIER 'bin']);
16 addpath(recursivepath([ISSM_TIER 'src/m']));
17 addpath(recursivepath([ISSM_TIER 'externalpackages/scotch']));
18 addpath(recursivepath([ISSM_TIER 'externalpackages/canos']));
19 addpath(recursivepath([ISSM_TIER 'externalpackages/kml']));
20 addpath(recursivepath([ISSM_TIER 'externalpackages/googleearthtoolbox']));
21 addpath(recursivepath([ISSM_TIER 'externalpackages/export_fig']));
```
Matlab Model class

>> md=model
md =

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

- Model properties are sorted by type:
  - mesh properties in `md.mesh`
  - material properties in `md.material`
  - ...

- Each model field is itself a matlab object with fields

- Default parameters are provided (physical constants, etc)

```matlab
>> md.materials
ans =
    Materials:
    rho_ice : 917   -- ice density [kg/m^3]
    rho_water : 1023   -- water density [kg/m^3]
    mu_water : 0.001787   -- water viscosity [N s/m^2]
    heatcapacity : 2093   -- heat capacity [J/kg/K]
    thermalconductivity : 2.4   -- ice thermal conductivity [W/m/K]
    meltingpoint : 273.15   -- melting point of ice at latm in K
    latenheat : 334000   -- latent heat of fusion [J/m^3]
    beta : 9.8e-08   -- rate of change of melting point with pressure [K/Pa]
    mixed_layer_capacity : 3974   -- mixed layer capacity [W/kg/K]
    thermal_exchange_vel... : 0.0001   -- thermal exchange velocity [m/s]
    rheology_B : N/A   -- flow law parameter [Pa/s^(1/n)]
    rheology_n : N/A   -- Glen's flow law exponent
    rheology_law : 'Paterson'   -- law for the temperature dependance of the ...
    rheology: 'None', 'Paterson' or 'Arrhenius'
```
Saving and loading a model

- \( md \) is a placeholder for all parameters, constants, geometrical properties, etc.
- No additional file required, everything is in ONE matlab variable.
- Use matlab's `save` command to save a model in binary format
- Use `loadmodel` to reload a model from a binary file

→ makes sure that old models are loaded correctly

```matlab
1 >> md=model;
2 >> md.miscellaneous.name
3 ans =
4   
5 >> md.miscellaneous.name='test';
6 >> md.miscellaneous.name
7 ans =
8   test
9 >> save myfirstmodel md
10 >> loadmodel md;
11 >> md.miscellaneous.name
12 ans =
13   test
14 >> md.miscellaneous.name
15 ans =
16   test
```
Display model fields

- `plotmodel` can be used to display any model field/result
- Arguments (pairs of options):
  1. model
  2. 'data'
  3. name of model field
  4. name of option1
  5. option1 value
  6. name of option2
  7. option2 value
  8. ...

```matlab
1   >> plotmodel(md,'data',md.mesh.x);
2   >> plotmodel(md,'data',md.mesh.x,'colorbar',0,'caxis',[0 500000]);
3   >> plotmodel(md,'data','mesh');
4   >> plotmodel(md,'data','BC');
5   >> plotmodel(md,'data','BC','data',md.mesh.x,'data','mesh');
```

- More info:
  - `plotdoc`
**EXP format**

- We rely on Argus’ format for geometrical files
- Argus files have an \( \text{exp} \) extension
- Example for a square domain:

```plaintext
1  ## Name:domainoutline
2  ## Icon:0
3  # Points Count  Value
4  5 1.
5  # X pos Y pos
6  0  0
7  100000  0
8  100000  100000
9  0  100000
10 0  0
```

- IssM has tools to read/write/modify \( \text{exp} \) files
- This format is used for:
  - domain outline (mesh boundary)
  - floating ice extension
  - flag elements or vertices inside a profile
Mesh generation with triangle

- `triangle` is used to generate simple uniform unstructured meshes
- Arguments:
  1. model
  2. name (string) of the domain outline
  3. element mean size

```plaintext
1  md=model;
2  md=triangle(md,'Square.exp',10000);
3  plotmodel(md,'data','mesh');
```
Parameterization file

- The parameter file sets up all required fields of a model:
  - Geometrical properties (Upper surface elevation, thickness,...)
  - Boundary conditions (friction, front,...)
  - Solver settings
  - ...

- Example for a square ice shelf:

```plaintext
hmin=300;
hmax=1000;
ymin=min(md.mesh.y);
ymax=max(md.mesh.y);
md.geometry.thickness=hmax+(hmin-hmax)*(md.mesh.y-ymin)/(ymax-ymin);
md.geometry.bed=md.materials.rho_ice/md.materials.rho_water*md.geometry.thickness;
md.geometry.surface=md.geometry.bed+md.geometry.thickness;
pos=find(md.mask.vertexonfloatingice);
md.friction.coefficient=200*ones(md.mesh.numberofvertices,1);
md.friction.coefficient(pos)=0;
md.friction.p=ones(md.mesh.numberofelements,1);
md.friction.q=ones(md.mesh.numberofelements,1);
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.vel=zeros(md.mesh.numberofvertices,1);
md.materials.rheology_B=paterson((273-20)*ones(md.mesh.numberofvertices,1));
md.materials.rheology_n=3*ones(md.mesh.numberofelements,1);
md=SetIceShelfBC(md,'Front.exp');
```
Interpolation modules

- **Interpolation from a regular grid to a mesh:** `InterpFromGridToMesh`

  ```java
  md.initialization.temperature=InterpFromGridToMesh(x,y,temp,md.mesh.x,md.mesh.y,250);
  ```

- **Interpolation from between two meshes:** `InterpFromMeshToMesh2d`

  ```java
  md.initialization.temperature=InterpFromMeshToMesh2d(index,x,y,temp,md.mesh.x,md.mesh.y);
  ```
SeaRISE example

```matlab
% Greenland parameters
md.missellaneous.name = 'SeaRISEGreenland';

modellpath = [ismdirl '/projects/ModelData/SeaRISE/Greenland5km_v1.2'];
thicknesspath = [modellpath 'thk.mat'];
surfacepath = [modellpath 'surf.mat'];
bedrockpath = [modellpath 'topg.mat'];
vxpath = [modellpath 'surfvelx.mat'];
vypath = [modellpath 'surffely.mat'];
temperaturepath = [modellpath 'surftemp.mat'];
precippath = [modellpath 'smb.mat'];

% Some hardcoded parameters for this deck

disp('reading geometry');
md.geometry.thickness = InterpFromFile(md.mesh.x, md.mesh.y, thicknesspath, 0);
pos0 = find(md.geometry.thickness < 1);
md.geometry.thickness(pos0) = 1;
md.geometry.surface = InterpFromFile(md.mesh.x, md.mesh.y, surfacepath, 0);
md.geometry.bedrock = InterpFromFile(md.mesh.x, md.mesh.y, bedrockpath, 0);
md.geometry.bathymetry = InterpFromFile(md.mesh.x, md.mesh.y, bathymetry);  
md.geometry.bed = md.geometry.surface - md.geometry.thickness;

% reading velocities 

disp('reading velocities');
md.inversion.vx_obs = InterpFromFile(md.mesh.x, md.mesh.y, vxpath, 0);
md.inversion.vy_obs = InterpFromFile(md.mesh.x, md.mesh.y, vypath, 0);
md.inversion.vx_obs = sqrt(md.inversion.vx_obs.^2 + md.inversion.vy_obs.^2);
md.initialization.vx = md.inversion.vx_obs;
md.initialization.vy = md.inversion.vy_obs;
md.initialization.vz = md.mesh.numberofvertices, 1;
md.initialization.vel = md.inversion.vel_obs;

% creating friction parameters

disp('creating friction parameters');
md.fricition.p = ones(md.mesh.numberofelements, 1);
md.fricition.q = ones(md.mesh.numberofelements, 1);
md.fricition.coefficient = 30 * ones(md.mesh.numberofvertices, 1);
min_drag_coeff = 50; background_drag_coeff = 200;
md.fricition.coefficient = background_drag_coeff * ones(md.mesh.numberofelements, 1);
md.fricition.coefficient = md.inversion.vel_obs > 30;
md.fricition.coefficient = background_drag_coeff + (min_drag_coeff - background_drag_coeff) / maxvel * md.inversion.vel_obs;

% loading temperature

disp('loading temperature');
md.initialization.temperature = InterpFromFile(md.mesh.x, md.mesh.y, temperaturepath, 0) + 273.15;
md.initialization.pressure = md.materials.rho_ice * md.constants.g * (md.geometry.surface - md.mesh.z);
...
Mask

- `setmask` is used to generate areas where ice is grounded and floating
- Arguments:
  1. model
  2. floating ice domain
  3. grounded ice inside the floating ice
- Ice considered grounded by default
- Input files in Argus format
- Examples
  ```
  md=setmask(md,"","") \rightarrow all grounded
  md=setmask(md,\'all\',") \rightarrow all floating
  md=setmask(md,\'IceShelves.exp\',") \rightarrow grounded with some floating parts
  md=setmask(md,\'all\',\'Islands.exp\') \rightarrow floating with some grounded parts
  ```
- To display the mask:
  ```
  >> plotmodel(md,'data',md.mask.elementonfloatingice)
  ```
Flow equation

`setflowequation` is used to generate the approximation used to compute the velocity

- **Arguments:**
  - 1. model
  - 2. approximation names
  - 3. approximation domains

- Domains can be Argus files or array of element flags
- **Approximation available**
  - stokes (Full-Stokes model)
  - pattyn (Higher-order model)
  - macayeal (Shallow Shelf Approximation)
  - hutter (Shallow Ice Approximation)

- Model coupling possible (see tomorrow’s presentation)
Flow equation

`setflowequation` is used to generate the approximation used to compute the velocity

- Examples

```matlab
1  md=setflowequation(md,'hutter','all')
2  md=setflowequation(md,'stokes','all')
3  md=setflowequation(md,'macayeal',md.mask.elementonfloatingice,'pattyn',md.mask.elementongrounded
   ice)
4  md=setflowequation(md,'macayeal','IceShelves.exp','fill','pattyn')
```

- To display the type of approximation:

```matlab
1  >> plotmodel(md,'data','elements_type')
```
# Diagnostic parameters

Most diagnostic parameters can be found in `md.diagnostic`

```matlab
>> md.diagnostic
ans =

    Diagnostic solution parameters:

    Convergence criteria:
        restol   : 0.0001  -- mechanical equilibrium residue convergence criterion
        reltol   : 0.01   -- velocity relative convergence criterion, NaN -> not applied
        abstol   : 10     -- velocity absolute convergence criterion, NaN -> not applied
        maxiter  : 100    -- maximum number of nonlinear iterations
        viscosity_overshoot : 0  -- over-shooting constant new=new+C*(new-old)

    boundary conditions:
        spcvx   : N/A   -- x-axis velocity constraint (NaN means no constraint)
        spcwy   : N/A   -- y-axis velocity constraint (NaN means no constraint)
        spcvz   : N/A   -- z-axis velocity constraint (NaN means no constraint)
        icefront   : N/A   -- segments on ice front list (last column 0-> Air, 1-> Water, 2->Ice)

    Rift options:
        rift_penalty_threshold : 0  -- threshold for instability of mechanical constraints
        rift_penalty_lock      : 10 -- number of iterations before rift penalties are locked

    Penalty options:
        jpenalty_factor   : 3  -- offset used by penalties: penalty = Kmax*10^offset
        jvertex_pairing   : N/A -- pairs of vertices that are penalized

    Other:
        shelf_dampening   : 0  -- use dampening for floating ice ? Only for Stokes model
        stokesreconditioning   : 100000000000000  -- multiplier for incompressibility equation. Only for Stokes model
        referential         : N/A  -- local referential
        requested_outputs   : N/A  -- additional outputs requested
```

Launch diagnostic solution with:

```matlab
>> md=solve(md,DiagnosticSolutionEnum)
```
Boundary conditions

- \( \text{md} = \text{SetIceSheetBC}(\text{md}) \)
  - Dirichlet BC for all nodes on boundary

- \( \text{md} = \text{SetIceShelf}(\text{md}, '\text{Front.exp}') \)
  - Neumann BC for all nodes on boundary in 'Front.exp'
  - Dirichlet BC for all other nodes on boundary

- \( \text{md} = \text{SetMarineIceSheefBC}(\text{md}) \)
  - Dirichlet BC for all nodes on grounded boundary
  - Neumann BC for all nodes on floating boundary
Prognostic parameters

Most prognostic parameters can be found in `md.prognostic`

```plaintext
>> md.prognostic
ans =

    Prognostic solution parameters:
        spcthickness : N/A    -- 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 : N/A        -- pairs of vertices that are penalized
```
Prognostic parameters

Use `md.timestepping` to change the time step:

```matlab
>> md.timestepping
ans =

timestepping parameters:
  time_step : 0.5  -- length of time steps [yrs]
  final_time : 5   -- 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
```

Launch prognostic solution with:

```matlab
>> md=solve(md,PrognosticSolutionEnum)
```
Transient solution

Most transient parameters can be found in `md.transient`

```matlab
>> md.transient
ans =

        transient solution parameters:
        isprognostic : 1  -- indicates if a prognostic solution is used in the ...
                        transient
        isthermal    : 1  -- 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
```

Transient solutions in 2D combine:
- diagnostic
- prognostic
- grounding line

→ Some of these components can be deactivated
Transient solution

**Use** `md.timestepping` **to change the time step:**

```matlab
>> md.timestepping
ans =

timestepping parameters:
    time_step    : 0.5  -- length of time steps [yrs]
    final_time   : 5    -- 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
```

Launch transient solution with:

```matlab
>> md=solve(md,TransientSolutionEnum)
```
Extrusion

- extrude is used to extrude the 2d mesh into a 3d mesh
- Arguments:
  1. model
  2. number of layers
  3. lower extrusion exponent
  4. upper extrusion exponent (optional)
- Examples

```
1. md=extrude(md, 8, 1)
2. md=extrude(md, 10, 1.5)
3. md=extrude(md, 10, 1.5, 1.5)
```

- To display the mesh:

```
1. plotmodel(md, 'data', 'mesh');
```
Thermal solution

Most thermal parameters can be found in `md.thermal`

```matlab
ans =
  Thermal solution parameters:
  spctemperature : N/A -- temperature constraints (NaN means no ...
    constraint)
  stabilization   : 1   -- 0->no, 1->artificial_diffusivity, 2->SUPG
  maxiter        : 100  -- maximum number of non linear iterations
  penalty_lock   : 0   -- stabilize unstable thermal constraints ...
  that keep zigzagging after n iteration (default is 0, no stabilization)
  penalty_threshold : 0  -- threshold to declare convergence of ...
    thermal solution (default is 0)
```

Thermal solution in 3D only to compute

- thermal steady state
- thermal transient

→ Controlled by `md.timestepping.timestep`

Launch thermal solution with:

```matlab
>> md=solve(md, ThermalSolutionEnum)
```
Transient solution

Most transient parameters can be found in `md.transient`

```matlab
>> md.transient
ans =

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

Transient solutions in 3D combine:

- thermal
- diagnostic
- prognostic
- grounding line

→ Some of these components can be deactivated
Thanks!