



Magma Cooling Simulator

User Guide

Contents

1	Project	3
1.1	Description	3
1.2	Model	3
2	Code Structure	4
2.1	Hierarchy	4
2.2	Variables	4
2.2.1	Geometry	4
2.2.2	Material	4
2.2.3	Options	5
2.2.4	Output	5
3	Geometry.	6
3.1	Import a geometry	6
3.2	Generate a new geometry	6
3.3	Modify an existing geometry	6
4	Heat Transfer Problem Example	8
5	Simulate	10
6	Results Analysis	11
6.1	interpolateTemperature	11
6.2	Built-in visualization tools	11
6.2.1	pdeplot3D	11
6.2.2	pdegplot	11
6.2.3	pdemesh	12
6.3	Isosurface 3D animation	12
6.4	Heat map slice view	13

1 Project

1.1 Description

This project simulates the natural cooling of magma-composed particles of different geometries. The original purpose of this code is to observe the cooling of the leftovers resulting from a potential collision between Mars and a protoplanet four billions years ago. In this context, the particles simulated are part of the accretion disk which ended up forming Phobos and Deimos (*P. Rosenblatt et al., Accretion of Phobos and Deimos in an extended debris disc stirred by transient moons. Nat. Geosci. 9, 581–583 (2016)*).

The model focuses on the resolution of the heat transfer equation with the finite element method using MATLAB's Partial Differential Equation Toolbox as its main support.

A PYTHON bridge permitting to use the model without leaving PYTHON is made available. It relies on the MATLAB engine API for PYTHON.

1.2 Model

Once a geometry and a material are defined, the modeling of the heat transfer problem is as follows. The initial temperature is set at $T_0 = 2000$ K, with and outside temperature equals to $T_{out} = 300$ K. For the diffusion matter, we consider a temperature-dependent thermal diffusivity $\kappa = \frac{\lambda}{\rho c_p}$ where ρ is the density (kg.m^{-3}), c_p the specific heat capacity ($\text{J.kg}^{-1}.\text{K}^{-1}$) and λ the thermal conductivity ($\text{W.m}^{-1}.\text{K}^{-1}$). Radiation is the predominant mode of heat transfer on the particle's surface.

Other properties can also be defined to improve the model such as latent heat L , or the material's heterogeneity inside the object.

2 Code Structure

2.1 Hierarchy

The **tools** folder contains the tools used to generate or import geometries and those used to exploit the simulation results :

- `generateGeometry` : generates a geometry when given a shape (*cube*, *ellipsoid*), size, and parameters to create material spherical heterogeneity (`nbr_cavities`, `min_radius`, `max_radius`).
- `modifyGeometry` : ?
- `createAnimation` : creates a 3D cooling animation over time with an isosurface at a given temperature.
- `getThermalBarycenter` : returns the coordinates of the last point reaching a given temperature, as well as the time it occurs.

The **tools/utils** folder contains utilities useful to the **tools** folder functions :

- `digCube` : randomly creates spherical heterogeneity inside a cube.
- `digEllipsoid` : randomly creates spherical heterogeneity inside an ellipsoid.
- `spheresIntersect` : checks whether two spheres intersect.

2.2 Variables

The code mostly uses structure arrays to store geometry, material, options, and output data.

2.2.1 Geometry

A geometry structure is composed of the following mandatory fields :

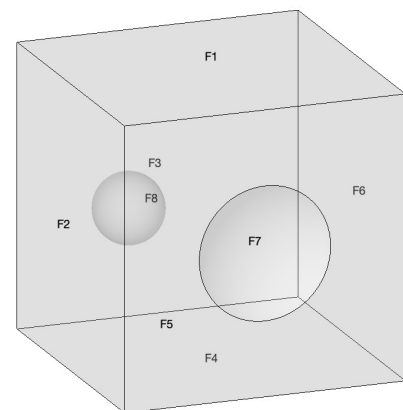
Fields	Type	Description
<code>structure</code>	<code>pde.DiscreteGeometry</code>	Geometry
<code>exposed_faces</code>	array (double)	ID of the faces exposed to the radiation flow
<code>nbr_cavities_faces</code>	array (double)	Number of faces in all the cavities

MATLAB counts the faces of the geometry in the following order :

1. planar external faces
2. curved/modified external faces
3. faces inside the cavities

The following heterogeneous cube example is given. Here are the corresponding parameters :

```
exposed_faces = [1:7]
nbr_cavities_faces = 1
```



2.2.2 Material

A material structure is composed of the following mandatory fields :

Fields	Type	Description
rho	array (double)	Density of the object ($kg \cdot m^3$)
cp	array (double)	Specific heat ($J \cdot kg^{-1} \cdot K^{-1}$)
T_0	array (double)	Initial temperature of the object (K)

2.2.3 Options

An options structure is composed of the following mandatory fields :

Fields	Type	Description
material	material	The properties of the material composing the main geometry
cavities_material	material	The properties of the material composing the cavities
T_out	array (double)	Temperature outside the object (K)
eps	array (double)	Emissivity of the object (no dimension)
dt	array (double)	Simulation time step (s)
tmax	array (double)	Duration of the simulation (s)

2.2.4 Output

Fields	Type	Description
Temperature	array (double)	Temperature values at nodes
SolutionTimes	array (double)	Solution times
XGradients	array (double)	x-component of the temperature gradient at nodes
YGradients	array (double)	y-component of the temperature gradient at nodes
ZGradients	array (double)	z-component of the temperature gradient at nodes
Mesh	FEMesh	Finite element mesh

3 Geometry

Several ways to handle geometries are implemented. One can import a pre-existing geometry from a file, generate cubic and ellipsoidal geometries from scratch, or modify existing geometries.

3.1 Import a geometry

One can import a `geometry.structure` from a STL or a STEP file using the `importGeometry` MATLAB built-in function.

Example :

```
1 geometry.structure = tools.importGeometry('Torus.stl');
2 geometry.exposed_faces = 6;
3 geometry.nbr_cavities = 5;
```

3.2 Generate a new geometry

One can generate new cubic or ellipsoidal geometries. The function `generateGeometry` returns a `geometry` structure, with the following arguments :

Fields	Type	Description
shape	string	'cube' or 'ellipsoid'
unit	array (double)	semi-axes of the ellipsoid or edge of the cube (m)
nbr_cavities	array (double)	Number of cavities that must be digged
min_radius	array (double)	Minimum radius of a cavity (m)
max_radius	array (double)	Maximum radius of a cavity (m)

The radius of the spherical heterogenous cavities is randomized between `min_radius` and `max_radius`. Set `nbr_cavities` to 0 for homogeneous geometry.

Examples :

```
1 c_geometry = tools.generateGeometry('cube',15,100,0.2,1.2);
2 e_geometry = tools.generateGeometry('ellipsoid',[1,0.5,0.5],5,0.05,0.11);
```

3.3 Modify an existing geometry

One can modify a pre-existing entirely filled geometry to insert cavities or heterogeneity using the `modifyGeometry` function, that takes the following parameters :

Fields	Type	Description
previous_geometry	pde.DiscreteGeometry	Geometry to modify
shape	string	'cube' or 'ellipsoid'
unit	array (double)	semi-axes of the ellipsoid or edge of the cube (m)
nbr_cavities	array (double)	Number of cavities that must be digged
min_radius	array (double)	Minimum radius of a cavity (m)
max_radius	array (double)	Maximum radius of a cavity (m)

Examples :

```
1 cube = tools.importGeometry('Cube.stl');
2 c_geometry = tools.modifyGeometry(cube, 'cube', 2, 30, 0.1, 0.9);

1 ellipsoid = tools.importGeometry('Ellipsoid.stl');
2 e_geometry = tools.modifyGeometry(ellipsoid, 'ellipsoid', [1,0.5,0.5], 30, 0.05, 0.11);
```

4 Heat Transfer Problem Example

This example shows how to solve the problem using the Finite Element Method with a temperature-dependent thermal diffusivity. The considered geometry is an heterogeneous ellipsoid. The partial differential equation for transient heat transfer is:

$$\rho c_p \frac{\partial T}{\partial t} = \nabla(\lambda \nabla T)$$

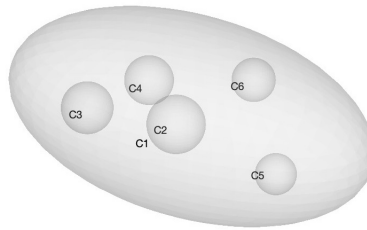
where T is the temperature, ρ is the material density, c_p is the specific heat, and λ is the thermal conductivity, among with the thermal diffusivity $\kappa = \frac{\lambda}{\rho c_p}$.

First, the ellipsoid geometry is defined :

```
1 nbr_cavities=5;
2 geometry=tools.generateGeometry('ellipsoid',[1,0.5,0.5], nbr_cavities, 0.1, 0.3);
3 geometry.exposed_faces=1;
```

Plot the geometry with cell labels displayed. These labels will be used to define heterogeneous material properties in the Simulate function.

```
1 figure;
2 pdegplot(geometry.structure, 'FaceAlpha', 0.2, 'CellLabels', 'on');
3 % delete axis
4 delete(findobj(gca,'type','Text'));
5 delete(findobj(gca,'type','Quiver'));
6 axis off;
```



Set properties of main material basalt and heterogeneous material vaporised_basalt.

```
1 basalt.rho=1000;
2 basalt.cp=1000;
3 basalt.T_0=2000;
4 vaporised_basalt.rho=1000;
5 vaporised_basalt.cp=1000;
6 vaporised_basalt.T_0=2000;
```

Set the parameters of the problem.

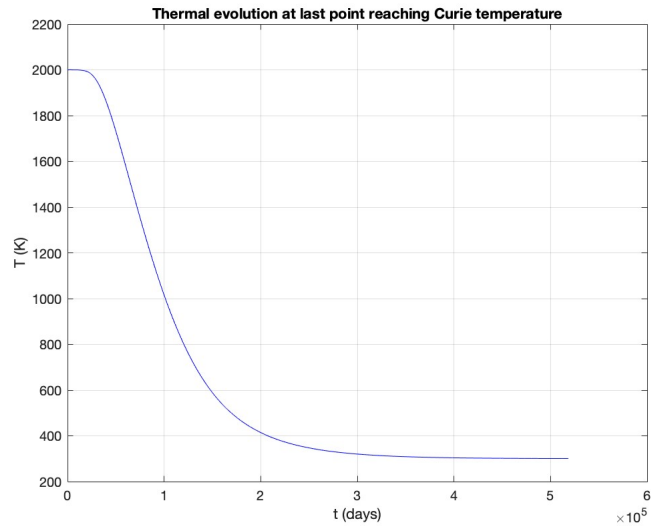
```
1 options.material=basalt;
2 options.cavities_material=vaporised_basalt;
3 options.eps=1; % Emissivity
4 options.TCurie=858; % (K) Curie temperature
5 options.T_out=300; % (K) Outer space temperature
6 options.tmax=6*3600*24; % (s) max integration time
7 options.dt=200; % (s) time-step
```

Call the simulate function. A description of the implementation of the function can be found in the next section.

```
1 simulation=Simulate(geometry, options);
```

One can get the coordinates and time when the whole geometry reached the Curie temperature, to plot the thermal evolution over time at that node, using the built-in PDE function interpolateTemperature.

```
1 [Bar, Time]=tools.getThermalBarycenter(simulation, options.TCurie);
2 tempBar=interpolateTemperature(simulation, Bar, 1:numel(simulation.SolutionTimes));
3 figure;
4 plot(simulation.SolutionTimes, tempBar, '-b');
5 xlabel('t (days)');
6 ylabel('T (K)');
7 title('Thermal evolution at last point reaching Curie temperature');
8 grid on;
```

All the results of the simulation can be saved in a structure and in a file

```

1 results.simulation=simulation;
2 results.tlist=simulation.SolutionTimes;
3 results.tempBar=tempBar;
4 results.bar=Bar;
5 results.nbr_cavities=geometry.nbr_cavities;
6 results.time=Time/(3600*24);
7 results.volume=geometry.volume;
8 results.porous_volume=geometry.porous_volume;
9 results.porosity_fraction=(results.porous_volume/results.volume)*100;
10 save('Example.mat','results');
```

Results can now be used to plot animations or figures.

5 Simulate

The function first defines the time list to solve the transient problem as well as the the temperature-dependent properties :

```
1 tlist = 0:options.dt:options.tmax;
2 lambda = @(location,state) (0.46+0.95*exp(-2.3e-3*state.u));
```

Create a PDE model for transient thermal analysis.

```
1 thermalModel = createpde('thermal','transient');
```

Import the geometry.

```
1 thermalModel.Geometry=geometry.structure;
```

Generate a mesh for the geometry stored in the model object. Parameter Hmax is the maximum mesh edge length, and geometric order must be specified as linear or quadratic.

```
1 generateMesh(thermalModel,'Hmax',0.2,'GeometricOrder','quadratic');
```

Specify the thermal properties of the main material located in cell 1.

```
1 thermalProperties(thermalModel,'Cell',1,'ThermalConductivity',lambda, ...
2                 'MassDensity',options.material.rho, ...
3                 'SpecificHeat',options.material.cp);
```

Specify the thermal properties of the heterogeneous material located in other cells.

```
1 thermalProperties(thermalModel,'Cell',2:geometry.nbr_cavities+1, ...
2                 'ThermalConductivity',lambda, ...
3                 'MassDensity',options.cavities_material.rho, ...
4                 'SpecificHeat',options.cavities_material.cp);
```

Specify the initial temperature in both materials.

```
1 thermalIC(thermalModel,options.material.T_0,'Cell',1);
2 thermalIC(thermalModel,options.cavities_material.T_0,'Cell',2:geometry.nbr_cavities+1);
```

Define the radiation boundary condition on the geometry surface of the model.

```
1 thermalModel.StefanBoltzmannConstant = 5.670373E-8;
2 thermalBC(thermalModel,'Face',1, ...
3           'Emissivity',@(region,state) options.eps, ...
4           'AmbientTemperature',options.T_out, ...
5           'Vectorized','on');
```

Solve the model for the time steps in tlist.

```
1 Results = solve(thermalModel,tlist);
```

6 Results Analysis

6.1 interpolateTemperature

The built-in function allows to interpolate temperature in thermal result at arbitrary spatial locations.

Fields	Type	Description
thermalResults	TransientThermalResults	results of the simulation returned by Simulate
querypoints	double	coordinates specified as a vector

Example : interpolates temperature at the origin of geometry

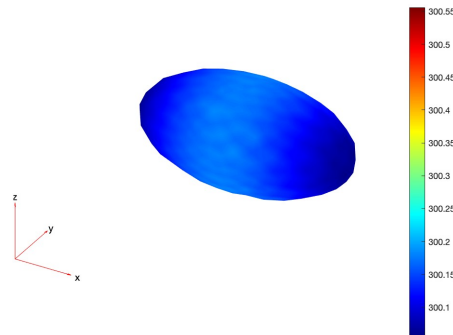
```
1 simulation=Simulate(geometry, options);
2 tempOrigin=interpolateTemperature(simulation,[0;0;0],1:numel(simulation.SolutionTimes));
```

6.2 Built-in visualization tools

6.2.1 pdeplot3D

Plot solution or surface mesh for 3-D problem.

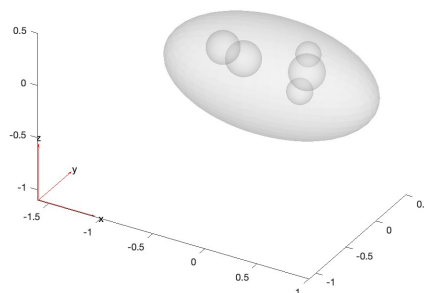
```
1 simulation=Simulate(geometry, options);
2 figure;
3 pdeplot3D(simulation.Mesh,ColorMapData=simulation.Temperature(:,end));
```



6.2.2 pdegplot

Plot PDE geometry.

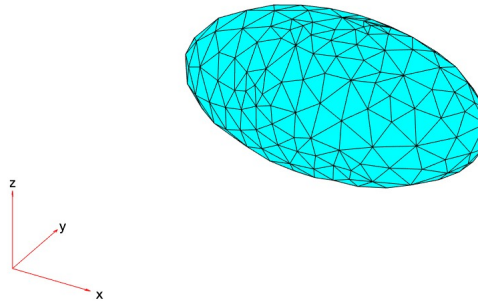
```
1 figure;
2 pdegplot(geometry.structure, "FaceAlpha",0.2);
```



6.2.3 pdemesh

Plot PDE mesh

```
1 simulation=Simulate(geometry, options);  
2 figure;  
3 pdemesh(simulation.Mesh);
```

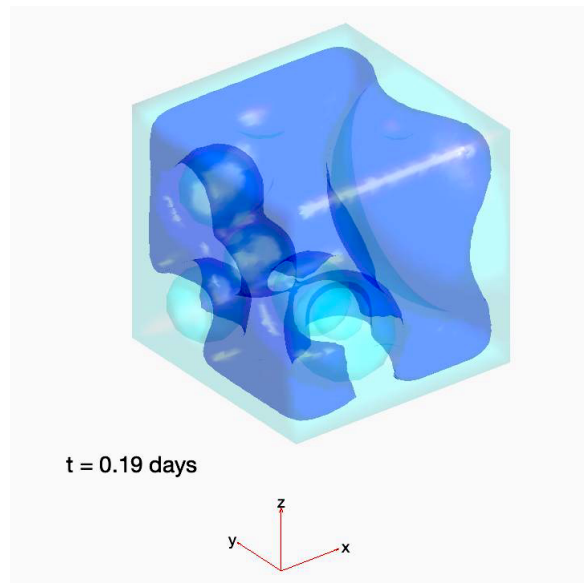


6.3 Isosurface 3D animation

One can create an animation of the thermal evolution at a given temperature. The `createAnimation` function hence export the animation in a video file. Parameters are the video filename, the result structure of the simulation, and a temperature to render the isosurface.

Example : Animation of a cube with voided cavities at Curie temperature

```
1 tools.createAnimation('Animation.avi', simulation, options.TCurie);
```



6.4 Heat map slice view

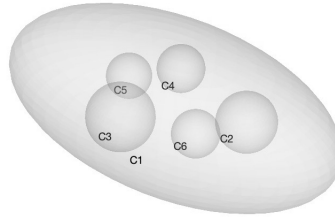
One can represent the temperature distribution inside the geometry at different time steps, inside a chosen plane. To do so, the `heatMapSlice` and `heatMapSliceAnimation` functions located in the `tools` folder are implemented. The first one outputs a subplot of the heatmap at different times, whereas the second outputs an animation under a video file. They both take the following parameters

Fields	Type	Description
filename	string	Filename under which to save the figure or animation (.pdf or .avi)
thermalResults	TransientThermalResults	Results of the simulation returned by <code>Simulate</code>
view_type	string	Slice plane for representation; must be 'x', 'y', or 'z'
value	double	Value of the slice view plane

Example : One can obtain the visualization in the $z = 0$ plane by calling the function

```
1 tools.heatMapSlice('slice.pdf',simulation,'z',0);
2 tools.heatMapSliceAnimation('sliceAnimation.avi',simulation,'z',0);
```

Considering the following geometry with low thermal diffusivity in heterogeneous material :



With the `heatMapSlice` function, one can get

