Magma Cooling Simulator

User Guide

Version #1, July 2024

Contents

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 (J.kg.$^{-1}$.K^{-1}) and λ the thermal conductivity (W.m^{-1}.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` : modify an existing `.stl` geometry to generate heteregenous cavities.
- `getThermalBarycenter` : returns the coordinates of the last point reaching a given temperature, as well as the time it occurs.
- `isosurface` : creates a 3D cooling animation over time with an isosurface at a given temperature.
- `heatMapSlice` : creates a `.pdf` file containing slice views of the diffusion.
- `heatMapSliceAnimation` : creates a `.avi` animation file with a slice view of the diffusion.
- `geometryView` : creates a `.pdf` file containing views of the geometry and mesh.

The **tools/utils** folder contains utilitaries 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 :

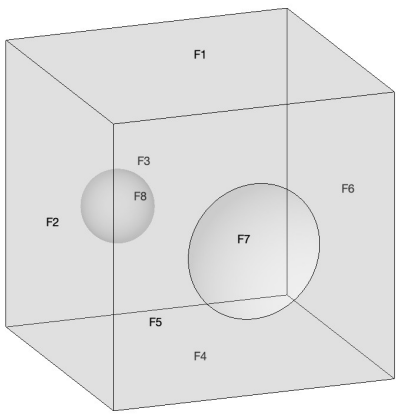
Fields	Type	Description
`structure`	`pde.DiscreteGeometry`	Geometry
`exposed_faces`	`array (double)`	ID of the faces exposed to the radiation flow
`nbr_cavities_faces`	`array (double)`	Number of faces in all the cavities

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)
lambda	string	Thermal conductivity ($Wm^{-1}K^{-1}$)

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. User must however give the number of cavities and the IDs of the faces exposed to radiation.

Example :

```
geometry.structure = tools.importGeometry('Torus.stl');
geometry.exposed_faces = 6;
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
`type`	`string`	Type of cavities; must be `'full'` or `'void'`
`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 :

```
c_geometry = tools.generateGeometry('cube',15,100,'void',0.2,1.2);
e_geometry = tools.generateGeometry('ellipsoid',[1,0.5,0.5],5,'full',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
type	string	Type of cavities; must be 'full' or 'void'
min_radius	array (double)	Minimum radius of a cavity (m)
max_radius	array (double)	Maximum radius of a cavity (m)

Examples :

```
cube = tools.importGeometry('Cube.stl');
c_geometry = tools.modifyGeometry(cube, 'cube', 2, 30, 'full', 0.1, 0.9);
ellipsoid = tools.importGeometry('Ellipsoid.stl');
e_geometry = tools.modifyGeometry(ellipsoid, 'ellipsoid',[1,0.5,0.5], 30, 'void', 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:

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

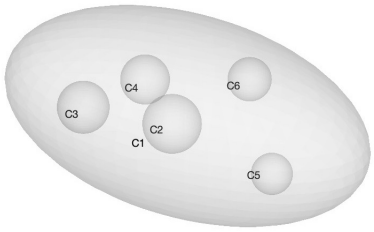
where T is the temperature, ρ is the material density, c_p is the specific heat, and λ is the thermal conductivity, among with the thermal diffusivity $\kappa = \frac{\lambda}{\rho c_p}$.

First, the ellipsoid geometry is defined :

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

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



Set properties of main material `basalt` and heterogeneous material `vaporised_basalt`.

```
basalt.rho=1000;
basalt.cp=1000;
basalt.T_0=2000;
basalt.lambda="tholeiitic";  % choose "tholeiitic", "dresser", "hebei"
vaporised_basalt.rho=1000;
vaporised_basalt.cp=1000;
vaporised_basalt.T_0=2000;
vaporised_basalt.lambda="hebei";  % choose "tholeiitic", "dresser", "hebei"
```

Set the parameters of the problem.

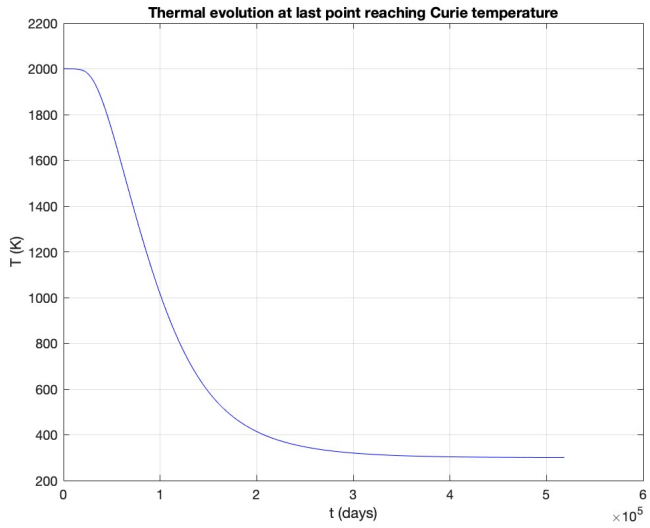
```
options.material=basalt;
options.cavities_material=vaporised_basalt;
options.latent_heat=0;          % (J/kg) Latent heat of basalt = 4e5.
                                % Set to 0 for no-latent heat
options.T_latent=1373;          % (K) Crystallization temperature of basalt
options.eps=1;                  % Emissivity
options.TCurie=858;             % (K) Curie temperature
options.T_out=300;              % (K) Outer space temperature
options.tmax=6*3600*24;         % (s) max integration time
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.

```
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`.

```
Result=tools.getThermalBarycenter(simulation, options.TCurie);
tempBar=interpolateTemperature(simulation,Result.Bar,1:numel(simulation.SolutionTimes));
figure;
plot(simulation.SolutionTimes, tempBar, '-b');
xlabel('t (days)');
ylabel('T (K)');
title('Thermal evolution at last point reaching Curie temperature');
grid on;
```



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

```
results.simulation=simulation;
results.temperature = simulation.Temperature;
results.nodes = simulation.Mesh.Nodes;
results.tlist=simulation.SolutionTimes;
results.tempBar=tempBar;
results.bar=Result.Bar;
results.time=Result.Time/(3600*24);
results.nbr_cavities=geometry.nbr_cavities;
results.volume=geometry.volume;
results.porous_volume=geometry.porous_volume;
results.porosity_fraction=(results.porous_volume/results.volume)*100;
```

Results can now be used to plot animations or figures.

```
tools.geometryView('Geometry.pdf', simulation, geometry);
tools.isosurface('Animation.avi', simulation, options.TCurie);
tools.heatMapSlice('Slice.pdf',simulation,'z',0);
tools.heatMapSliceAnimation('SliceVideo.avi',simulation,'z',0);
```

Results can also be extracted to use in PYTHON thanks to the `getPythonResults` from the `.mat` file.

5 Simulate

The function first defines the time list to solve the transient problem.

```
tlist = 0:options.dt:options.tmax;
```

The code checks whether there's any latent heat property. If so, adds a gaussian function with $\mu = 1373$, $\sigma = 5$ to the specific heat capacity value. It then reflects the latent heat release.

The temperature-dependent properties depending on the cavities type (`full` or `void`) are given. For the main material for example,

```
if options.material.lambda == "dresser"
  lambda_material = @(location, state) 0.35+0.85*exp(-1.7e-3*(state.u-273));
elseif options.material.lambda == "hebei"
  lambda_material = @(location, state) 0.59+0.23*exp(-4.3e-4*(state.u-273));
elseif options.material.lambda == "tholeiitic"
  lambda_material = @(location, state) 0.46+0.95*exp(-2.3e-3*(state.u-273));
else
  error('Unknow material type.');
end
```

Create a PDE model for transient thermal analysis.

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

Import the geometry.

```
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`.

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

Specify the thermal properties of the main material located in cell 1, and the initial temperature

```
thermalProperties(thermalModel,'Cell',1,'ThermalConductivity',lambda, ...
                  'MassDensity',options.material.rho, ...
                  'SpecificHeat',options.material.cp);
thermalIC(thermalModel,options.material.T_0,'Cell',1);
```

If the cavities are not voids but heteregenous material, set the material properties and inital temperature

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

```
thermalModel.StefanBoltzmannConstant = 5.670373E-8;
thermalBC(thermalModel,'Face',geometry.exposed_faces, ...
          'Emissivity',@(region,state) options.eps, ...
          'AmbientTemperature',options.T_out, ...
          'Vectorized','on');
```

Solve the model for the time steps in `tlist`.

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

The results are stored in a structure containing the following properties :

1. **Temperature**. Temperature values at nodes.
2. **SolutionTimes**. Same list as the tlist input to solve.
3. **XGradients**. x-component of the temperature gradient at nodes.
4. **YGradients**. y-component of the temperature gradient at nodes.
5. **ZGradients**. z-component of the temperature gradient at nodes.
6. **Mesh**. Finite element mesh, which is also a structure containing nodes coordinates, elements, maximum and minimum element size, and geometric order.

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

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

6.2 Built-in visualization tools

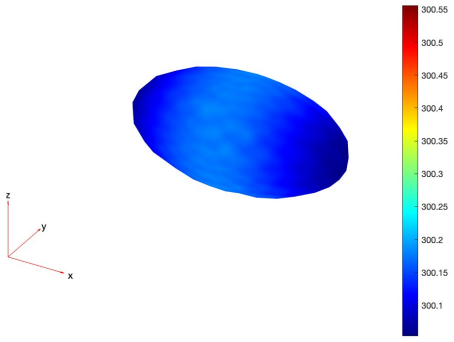
The `geometryView` function allows the user to save the following figures of the geometry in a pdf file. One can call the function in a script using:

```
tools.geometryView('Geometry.pdf', simulation, geometry);
```

It saves in a file the three following figures:

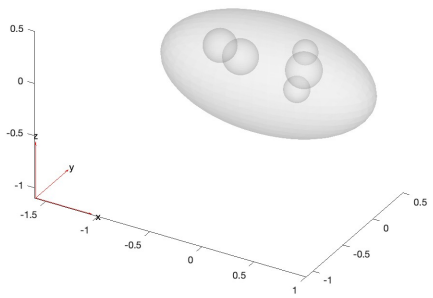
`pdeplot3D` : Plot solution or surface mesh for 3-D problem.

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



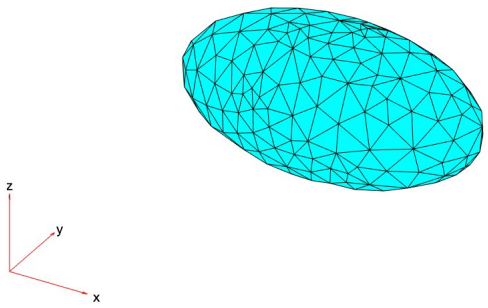
`pdegplot` : Plot PDE geometry.

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



`pdemesh` : Plot PDE mesh.

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

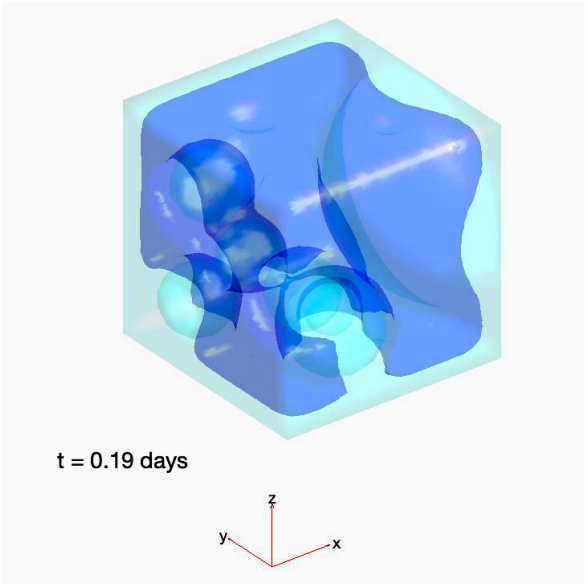


6.3 Isosurface 3D animation

One can create an animation of the thermal evolution at a given temperature. The `isosurface` 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

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



6.4 Heat map slice view

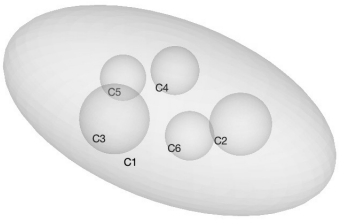
One can represent the temperature distribution inside the geometry at different time steps, inside a chosen plane. To do so, the `heatMapSlice` and `heatMapSliceAnimation` functions located in the `tools` folder are implemented. The first one outputs a subplot of the heatmap at different times, whereas 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 `Simulate`
`view_type`	`string`	Slice plane for representation; must be `'x'`, `'y'`, or `'z'`
`value`	`double`	Value of the slice view plane

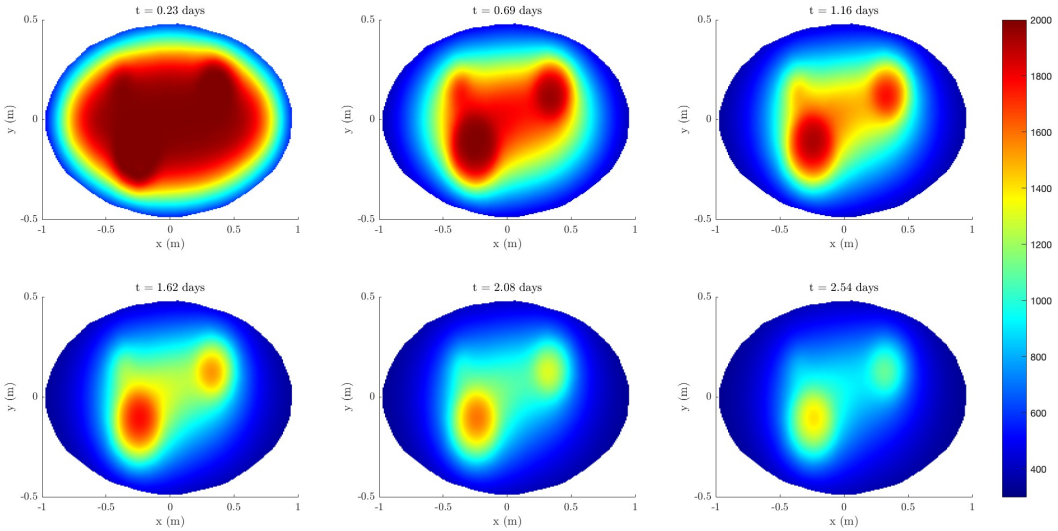
Example : One can obtain the visualization in the $z = 0$ plane by calling the function

```
tools.heatMapSlice('slice.pdf',simulation,'z',0);
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



7 Python Bridge

7.1 MATLAB Engine : `matlabengine`

By calling the matlabengine package, you can directly start a MATLAB session and call seamlessly call the model functions.
For more information and advanced use, please consult the official documentation.

7.1.1 Installation

Prerequisites

Python <= 3.11
MATLAB Desktop with the PDE Toolbox installed

```
$ pip install matlabengine
```

7.1.2 Starting the engine

One need to start or connect to a MATLAB session before doing anything.

```
eng = matlab.engine.start_matlab()
```

The MATLAB Engine should detect the appropriate version automatically but in some edge cases, it could not work. In this case, please consult the official documentation on the subject.

7.1.3 Calling Functions

Calling the model functions with the MATLAB Engine is simple : it is the exact same way as you would do in MATLAB. The only difference is that you need to add the `eng` prefix when calling them. For example :

```
geo = eng.tools.generateGeometry('ellipsoid',matlab.double([1.0,0.5,0.5]),3.0,0.1,0.3)
```

Be careful !

When calling a function that does not return one value, you need to add the `nargout` parameter to indicate how many objects will be returned.

7.1.4 Handling data types

Arrays and Integerss

PYTHON and MATLAB data types are not equivalent : therefore `matlabengine` translates them.
The majority of this model functions are expecting MATLAB '`double`' array type. But PYTHON '`int`' type is translated as a MATLAB '`int64`' type, leading to errors.
Hence, you need to be sure to pass '`float`' types to the MATLAB engine functions. PYTHON '`float`' is the only type converted as a MATLAB '`double`'

Structures

To create a MATLAB structure ('`geometry`', '`options`', '`material`'), you need to pass a PYTHON dictionnary containing fields name ('`string`') as keys and the data to transmit as values. For example:

```
basalt = {'rho': 1000.0,"cp": 100.0,"T_0": 2000.0}
```

Extract from the official documentation

Python type	MATLAB Type
float	double
complex	complex double
int	int64
float(nan)	NaN
float(inf)	inf
bool	logical
str	char
dict	structure
list	cell array
set	cell array
tuple	cell array

7.1.5 API Documentation

From the official documentation.

matlab.double

A lot of MATLAB built in functions need arrays to work properly. This function allows a python integer, a float or a list to be converted to a MATLAB 'double' array

Fields	Type	Description
struct	int, list or dict	structure to convert

eng.workspace

To make MATLAB consider variables (and save them), you need to add them to the MATLAB Workspace. In PYTHON, the MATLAB Workspace is represented as a dictionary, you can add variables to the Workspace the same way you would add a new field to a PYTHON dictionnary. For example:

```
eng.workspace['results']=results
```

eng.load

Load MATLAB variables from a given 'filepath'. Can only import '.mat' files.

Fields	Type	Description
filepath	string	file to load

eng.save

Used to save MATLAB workspace variables under a given 'filepath'. Output file can only be saved under a '.mat' extension.

Example :

```
eng.save("test.mat", "results", nargout=0)
```

Fields	Type	Description
`filepath`	`string`	file under which to save the variable
`var`	`string`	MATLAB Workspace variable name

7.2 SciPy Integration

Load MAT Files

MATLAB `.mat` files can be loaded in PYTHON using `scipy.io.loadmat`. They are then loaded as dictionnaries containing `numpy.ndarray`

Example :

```
1 results = scipy.io.loadmat("file.mat")
```