GUIDE TO USING THE INPUT FILE

The input file **input.in** serves as an interface for the user to define and configure the simulation parameters, select appropriate boundary and initial condition and save output of the simulation run. Few important points need to be kept in mind while preparing the infile:

1. Each input parameter line should end with semicolon; This separates one parameter from the other.
2. The infile is agnostic to spaces between key and value, i.e. key = value is same as key=value.
3. The key nomenclature is case insensitive, I.e.FILL_CUBE and Fill_Cube is identical.
4. Any line starting with # is not read. It can be used to write comments.
5. Boundary key should be placed before Filling key. Only after the variables in the Boundary key are read, they are identified as field variables and memory allocation is done in the code.

We discuss each infile keys in detail now.

1. Dimension

   DIM key denotes the number of dimensions of the simulation. A value of 2 would run a 2-D simulation while 3 denotes a 3-D simulation. Takes in integer values as **DIM = 2;**

2. Simulation size

   Num_X, Num_Y and **Num_Z** are the simulation domain sizes in the x,y and z directions respectively. They take an integer value as an input. Num_Z only needs to be defined if DIM=3. Even if it is wrongly initialized in the infile, it would be selected as 1 by default. Example assignment,

   ##Mesh size in different directions##
   Num_X = 400;
   Num_Y = 100;

3. Grid spacing

   dx, dy, dz denotes the discretized mesh spacing in the three directions. dz works similar to Num_Z. Takes in float as

   #Grid spacing#
   dx = 0.03;
   dy = 0.03;

4. Temporal parameters

   a) **dt** : denotes the temporal discretization step.

   b) **total_steps** : denotes the total number of time steps for the simulation run.

   c) **timebreak** : denotes the number of time steps after which the output will be written to files . Say, if total_steps = 1000 and timebreak =100, output files will be written every 100 steps(100,200,300 and so on). The information about the same will be displayed on your screen (terminal). Example:

##Time spacing##
dt = 1e-5;
##Total simulation timesteps##
total_steps = 50000;
##Time interval after which results are stored##
timebreak = 10000;

5. Phase-field model parameters : These are the parameters that correspond to the phase-field model of thermal dendrites by Kobayashi and follows the same notation.
    a) **tau**: interface relaxation parameter
    b) **epsilon**: interface width
    c) **delta**: anisotropy strength
    d) **j** : symmetry of the anisotropy
    e) **theta_0**: angle of rotation
    f) **a** : strength of thermal noise
    g) **alpha** : well tilting constant
    h) **gamma** : driving force coupling constant
    i) **K** : latent heat
    j) **T_e** : Equilibrium temperature

    The values can be assigned as,

##Phase-field model parameters##
epsilon = 0.01;
tau = 0.0003;
K = 1.4;
delta = 0.0;
j = 4;
theta_0 = 0.0;
alpha = 0.9;
gamma = 10.0;
a = 0.01;
T_e = 1.0;

6. Boundary conditions :

The boundary conditions are invoked via the **boundary** key. The key will take either 5 or 7 comma separated values depending on whether DIM has been initialized to 2 or 3 respectively. The first entity after = is the variable name followed by comma separated type of boundary condition. For instance,

boundary = phi, NOFLUX, NOFLUX, NOFLUX, NOFLUX;

implies isolated or Neumann boundary condition have been applied to top, bottom, left, right (in that order always) faces respectively. For 3-D simulations additional two boundary conditions would have to be provided for front and back faces. In its current form the code supports two types of

boundary conditions NOFLUX and PERIODIC.

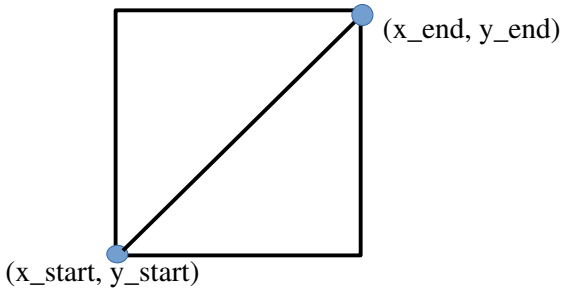
7. Initial condition:

The initial condition for the field variables can be assigned by the Fill_type key. Current implementation supports three filling options.

a) **Fill_Cube**: the key takes 6 or 8 comma separated values depending upon 2-D or 3-D simulation. For instance,

Fill_Cube = phi, 1.0, 0, 50, 0, 50;

The first value after = is the variable name. The second entry is filling value i.e. what value do you want to assign to the variable.The next 4 co-ordinates denote the spatial points (in order of x_start, x_end,y_ start,y_end) that define the boundaries of a square or rectangle as shown in fig. below. For 3-D two more arguments denoting z_start and z_end needs to be added .



The value is assigned only in the interior of the cube. The region outside the cube is filled by a value of 1.0 – filling value, which in this case is 0.

b) **Fill_Sphere**: works similar to Fill_Cube but used to fill a circle or a spherical initial domain. 5 arguments denoting name of variable, filling value, radius,x_center,y_center. For instance,

Fill_Sphere = phi, 1.0, 10, 50, 50;

Additional argument of z_center will be needed for running code in 3 dimensions. The value is assigned only in the interior of the circle. The region outside the circle is filled by a value of 1.0 – filling value, which in this case is 0.

c) **Fill_Constant**: To be used in cases where the entire simulation is to be assigned a constant value. Takes two arguments irrespective of the dimensions. First argument is the variable name followed by a comma separated constant filling value. For instance,

Fill_Constant = temp, 0.0;

8. **Respawn** :

Often the simulation needs to be restarted from a previously saved point. (The simulation may have ended pre-maturely). In such cases two infile keys needs to be invoked

a) RESPAWN :

The flag needs to be set to 1 to invoke respawn strategy as **RESPAWN = 1**;

b) restart_time

the last saved integer timestep file value needs to specified as **restart_time = 50000**;

With these two keys defined, the simulation will skip over the filling commands and assign the field variable values from previously saved output files. To respawn a simulation all field variables need to have been saved.

9. **File writing options**:

The code allows the user to save the output of their simulation results in file formats .vtk and .csv. The keys can be activated by a setting the flag to 1 as either **WRITE_TO_VTK = 1**; or **WRITE_TO_CSV = 1**; An output folder is automatically created on running the simulation. The resulting output can then be visualized using open source packages such as Paraview or Matplotlib.