**Numerical Methods in Heat and Mass Transfer**
**Final Project**
**May 1st, 2015**

**Group 5**

# MODELING OF SOLID-LIQUID PHASE CHANGE USING THE ENTHALPY-POROSITY METHOD

**Yash Ganatra**
ME Department

**Galen Jackson**
ME Department

## ABSTRACT

A numerical model was developed for solid-liquid phase change with the inclusion of convection. Algorithms for the continuity, conservation of linear momentum, and conservation of energy were developed using the finite volume method. The conservation of linear momentum and continuity equation were solved for using the SIMPLE algorithm and was validated against a driven lid problem. The conservation of energy equation was solved using an upwinding differencing scheme and was validated against a pure conduction case. These two algorithms were combined together and solved iteratively to calculate the temperature distribution and melting front of a solidifying phase change material. Results were validated against the literature and the effects of the morphology constant were investigated.

## INTRODUCTION

Thermal management is often a limiting factor in the development of electronic devices. As electronics get smaller and more powerful, new approaches to keeping these devices cool have to be investigated in order to manage the ever increasing thermal demand. One of these new approaches is in the form of thermal energy storage, specifically in passive heating application using a phase change material (PCM). These materials are capable of passively keeping an electronic system at a constant temperature using the PCM's isothermal phase change from solid to liquid, or vice versa. This phase change involves energy storage via latent heat, which can be enthalpy storage on the order of several magnitudes higher than sensible heat storage, the energy stored by a material changing temperature.

Recent designs have incorporated phase change materials into already existing cooling technologies. One such approach is to place phase change materials into heat sinks, as shown in Fig. 1 [1-3]. This approach replaces air with a PCM as the working fluid cooling the device and allows for a combination of latent heat and natural convection to absorb energy.

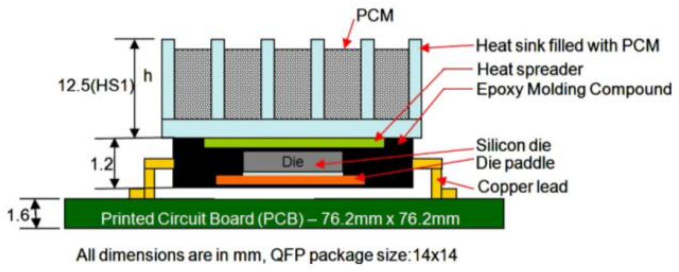

PCM
Heat sink filled with PCM
Heat spreader
Epoxy Molding Compound
12.5(HS1)
h
1.2
Silicon die
Die
Die paddle
Copper lead
1.6
Printed Circuit Board (PCB) – 76.2mm x 76.2mm
All dimensions are in mm, QFP package size:14x14

**Fig. 1. Heat sink using a PCM as the passive cooling medium [1].**

Typical domain geometries investigated for PCMs are rectangular in nature. Specifically, a common numerical approach to solving the problem is to solve a square geometry where one side is kept at a constant temperature that is higher than the melting temperature, $T_h$, and the opposite end at a temperature lower than the melting temperature of the PCM, $T_c$. To simplify the problem, the other sides of the rectangular geometry are set as adiabatic and gravity is set in the direction parallel to the temperature gradient [4-6].

The phase change process can vary greatly depending on the type of material. For pure metals and other materials, the phase change region is isothermal and a constant melting temperature can be used. However, most phase change composite materials, such as alloys, have a temperature range at which they change phases. Typically, one of two approaches is used to model the phase change region. One method is to treat the latent heat as a source term that is changing with time [6,7]. This method is ideal for the isothermal phase change case, since the source term accounts for the enthalpy change over the phase change process. The other approach, called the enthalpy or apparent heat capacity method [4], approximates the phase change region using an effective specific heat capacity. This method assumes that the enthalpy profile during the phase change is linear; however, curve fits can be used instead if the profile is found to be nonlinear [5].

In order to further understand the effect natural convection has on the phase change, a model was produced using the process provided by Voller et al. [6] to calculate the solidification of a liquid under the numerical conditions shown in Fig. 2. Specific focus was given to the methodology behind the calculation in order to further understand the numerical techniques used in typical phase change models.

## NOMENCLATURE

| | | | |
|---|---|---|---|
| $a$ | Coefficients in discretized conservation equations | $x, y$ | Coordinate directions |
| $A$ | Porosity function | Greek symbols | |
| $C$ | Morphological constant | $\alpha$ | Thermal diffusivity |
| $c, C_P$ | Specific heat | $\beta$ | Thermal coefficient of expansion |
| $F$ | Liquid fraction | $\in$ | Half-mushy range |
| $g$ | Gravity | $\mu$ | viscosity |
| $H$ | Total enthalpy | $\rho$ | density |
| $h$ | Sensible enthalpy | $\xi$ | perturbation |
| $\Delta H$ | Latent heat | Subscripts | |
| $k$ | Conductivity | $app$ | apparent |
| $p$ | Pressure | $m$ | Melting point |
| $S_b$ | Momentum source | $P$ | Node point |
| $S_h$ | Enthalpy source term | $W$ | West neighboring node |
| $T$ | Temperature | $E$ | East neighboring node |
| $t$ | Time | $N$ | North neighboring node |
| $\vec{V}$ | Velocity ($u$,$v$) | $S$ | South neighboring node |
| | | Other symbols | |
| | | $()^0$ | Old value |
| | | $[\![A, B]\!]$ | Maximum value of A,B |
| | | $()_n$ | $n$th iterative value |

## METHODOLOGY

### *Governing equations*

In order to calculate the convection phase change process, the governing equations for continuity, conservation of linear momentum, and conservation of energy equations need to be determined. Using the Navier-Stokes equations and the diffusion equation, the equations governing the process are

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0 \quad (1)$$

$$\frac{\partial(\rho \mathbf{u})}{\partial t} + \nabla \cdot \left(\rho \vec{\mathbf{V}} u\right) = -\nabla p + \nabla \cdot (\mu \nabla u) + A\vec{u} + S_b \quad (2)$$

$$\frac{\partial(\rho c_p T)}{\partial t} + \nabla \cdot \left(\rho c_p \vec{\mathbf{V}} T\right) = \nabla \cdot (k \nabla T) + S_h \quad (3)$$

respectively. For the velocity in the direction of gravity, $S_b$ is the momentum source term caused by buoyancy defined as

$$S_b = \rho g \beta (T - T_{ref}) \quad (4)$$

where $T_{ref}$ is the melting temperature of the PCM and $A$ is a correction term for the solid regions of the PCM defined as

$$A = -C(1 - F). \quad (5)$$

This $A$ term is the “porosity” section of the enthalpy-porosity method [8] because it produces a smooth transition between the solid and liquid regions of the PCM by treating the melting region as a porous structure. In Eq. 5, $C$ is a constant based on the morphology of the melting region and controls the degree to which the convection field penetrates into the melting region. It is important to be selective regarding which value of $C$ is used because it can have a significant effect on the melting front profile created.

In order to simplify the governing equations, the Boussinesq approximation was used. This approximation assumed that the density term only changes in the gravity source term, removing the transient response calculation of the density in the continuity equation. The following sections describe the numerical approaches to modeling the conservation of linear momentum and conservation of energy equations.

### *Conservation of linear momentum*

The conservation of linear momentum was modeled using the SIMPLE algorithm for a finite volume method developed by Patankar [9]. This method uses a staggered mesh discretization, shown in Fig. 2.

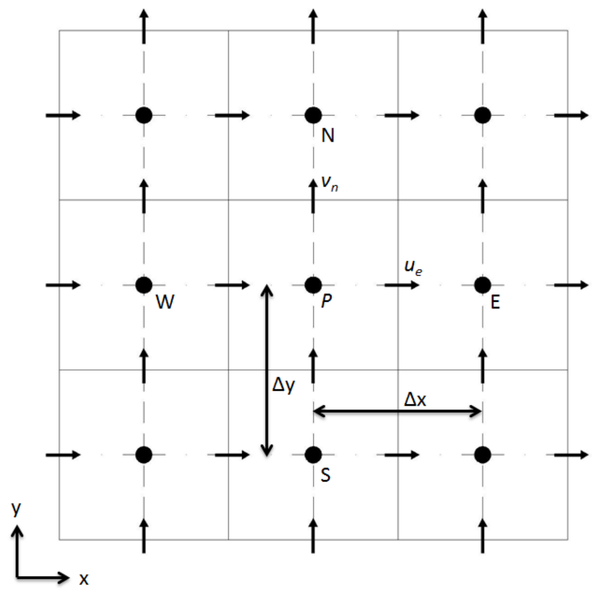


**Fig. 2. Discretization of the domain for numerical approach with staggered velocity mesh.**

By creating a staggered mesh where the pressure and velocities are stored at different points, an effect called checkerboarding, where the calculated pressure gradient creates an alternating velocity profile, is prevented. The SIMPLE algorithm calculates the $u$ and $v$ momentum equations using a decomposition of the terms

$$u = u^* + u'$$

$$v = v^* + v' \quad (6)$$
$$p = p^* + p'$$

where the starred terms are the solutions to the discretized momentum equations and the prime terms are correction terms. Applying conservation principles to the momentum equation creates

$$a_e u_e^* = \sum_{nb} a_{nb} u_{nb}^* + b_e + \Delta y(p_P^* - p_E^*)$$
$$a_n v_n^* = \sum_{nb} a_{nb} v_{nb}^* + b_n + \Delta x(p_P^* - p_N^*) \quad (7)$$

where

$$a_w = \frac{\mu \Delta y}{\Delta x} + \frac{F_w^*}{2}, \quad a_{ee} = \frac{\mu \Delta y}{\Delta x} - \frac{F_w^*}{2}$$
$$a_s = \frac{\mu \Delta y}{\Delta x} + \frac{F_s^*}{2}, \quad a_n = \frac{\mu \Delta y}{\Delta x} - \frac{F_n^*}{2}$$
$$a_p = \sum_{nb} a_{nb} - A\Delta x \Delta y + (F_{ee}^* - F_w^* + F_n^* - F_s^*)$$
$$b_e = 0, \quad b_n = \rho g \beta (T_n - T_{ref})$$
$$F_w^* = \rho u_w^* \Delta y$$
$$F_{ee}^* = \rho u_w^* \Delta y$$
$$F_{nn}^* = \rho v_{nn}^* \Delta x$$
$$F_s^* = \rho v_s^* \Delta x$$

Including a simplification for the prime terms finally solves for the actual velocities

$$u_e = u_e^* + d_e(p_P' - p_E')$$
$$v_n = v_n^* + d_n(p_P' - p_N') \quad (8)$$
$$d_e = \frac{\Delta y}{a_e}, \quad d_n = \frac{\Delta x}{a_n}$$

where the prime pressures are solved for using the continuity equation, generating

$$a_p p_P' = \sum_{nb} p_{nb}' + b \quad (9)$$

where

$$a_E = \rho_e d_e \Delta y$$
$$a_W = \rho_w d_w \Delta y$$
$$a_N = \rho_n d_n \Delta x$$
$$a_S = \rho_s d_s \Delta x$$
$$a_P = \sum_{nb} a_{nb}$$
$$b = F_w^* - F_e^* + F_s^* - F_n^*$$

Equations 6-9 are solved iteratively using a line-by-line TDMA solver for each equation until the pressure prime term is satisfactory close to zero, demonstrating that continuity is satisfied.

*Conservation of energy*

A fixed grid approach is used to solve phase change problems 11. Hence, the boundary condition at the interface is not explicitly needed to track the melt front. Representing appropriate mass and heat transfer conditions near the vicinity of the phase change region poses a challenge. This section describes two ways to account for phase change in fixed grid methods–volumetric source terms and the apparent heat capacity method.

### 1. Source Term based formulation

A step change in enthalpy is shown in Fig. 3, which depicts an enthalpy-temperature variation for an isothermal phase change.

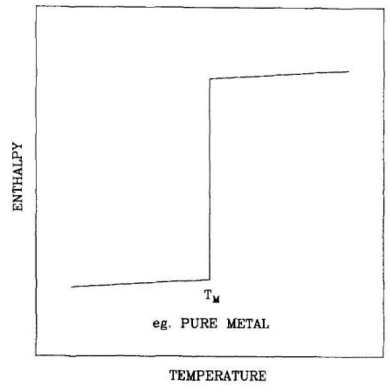


**Fig. 3. Variation of enthalpy with temperature for isothermal phase change**

The energy equation (10) in terms of total enthalpy, H, is shown

$$\frac{\partial \rho H}{\partial t} + \nabla(\rho \boldsymbol{u} H) = \operatorname{div}(k \nabla T) \quad (10)$$

The total enthalpy can be written as the sum of sensible latent heat and latent heat as shown (11),

$$H = h + \Delta H = C_p T + \Delta H \quad (11)$$

Substituting (11) in (10), gives the energy equation (12) with sensible latent heat, $h$, as its primary variable,

$$\frac{\partial \rho h}{\partial t} + \nabla(\rho \boldsymbol{u} h) = \operatorname{div}(\alpha \nabla h) - \frac{\partial(\rho \Delta H)}{\partial t} - \nabla(\rho \boldsymbol{u} \Delta H) \quad (12)$$

The effect of latent heat, $\Delta H$, which accounts for phase change, can be accounted for as a volumetric source (heat generation / heat sink) term (12),

$$S_h = \frac{\partial(\rho \Delta H)}{\partial t} + \nabla(\rho \boldsymbol{u} \Delta H) \quad (13)$$

The source term consists of a transient part, which accounts for evolution or absorption of latent heat, and a convective term, which denotes the velocity of the melt. For isothermal phase change, since the velocity at the solid-liquid interface is zero, the convective term is neglected.

$$S_h = \frac{\partial(\rho \Delta H)}{\partial t} \quad (14)$$

Using $h = C_p T$ to solve for temperature as the primary variable yields Eqn. (15),

$$\frac{\partial(\rho C_p T)}{\partial t} + \nabla(\rho \boldsymbol{u} C_p T) = \operatorname{div}(k \nabla T) - \frac{\partial(\rho \Delta H)}{\partial t} \quad (15)$$

### 2. Apparent Heat Capacity method

Phase change is accounted for in this process by altering the specific heat of the substance during phase transition as shown in Fig. 4. [14],[12][13]

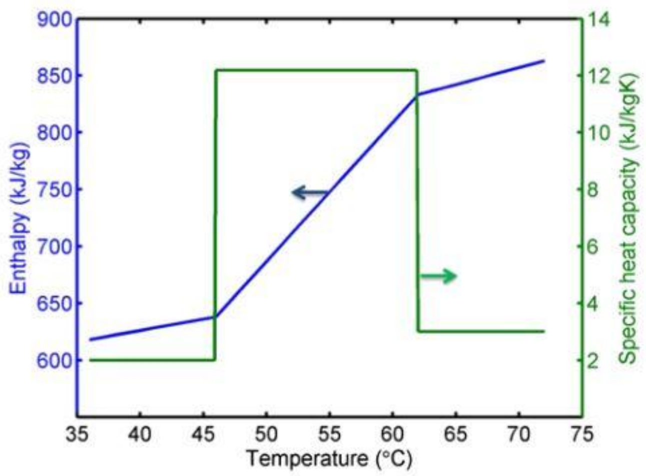


**Fig. 4. Variation of enthalpy and specific heat with temperature [15]**

If the latent heat is released uniformly, apparent specific heat capacity is defined as shown in Eqn. (16), [14]

$$C_{app} = \begin{cases} C_{m-\xi} & T < T_{m-\xi} \\ C_{in} & T_{m-\delta} < T < T_{m+\xi} \\ C_{m+\xi} & T > T_{m+\xi} \end{cases} \tag{16}$$

where $T_m$ is the melting temperature, $\xi$ is the artificial perturbation applied to an isothermal phase change (0.001 °C in this case), and $C_{in}$, heat capacity during phase transition, is defined as

$$C_{in} = \frac{\left\{\int_{T_{m-\xi}}^{T_{m+\xi}} C(T)dt + \Delta H\right\}}{T_{m+\xi} - T_s} \tag{17}$$

Note that if the phase change is not isothermal, $T_m$ becomes the mean melting temperature and $\xi$ becomes the melting temperature range. The energy equation with temperature as the primary variable simplifies to (18),

$$\frac{\partial(\rho C_{app}T)}{\partial t} + \nabla(\rho \boldsymbol{u} C_{app}T) = \text{div}(k\nabla T) \tag{18}$$

For isothermal phase change, an artificial phase change temperature range has to be added to avoid making Eqn. (17) undefined, which accounts for a narrow band representing the mushy zone–the intermediate zone between the solid and liquid phases.

The melt fraction is calculated as (),

$$f = \begin{cases} 0 & T < T_{m-\xi} \\ \dfrac{T - T_{m-\xi}}{T_{m+\xi} - T_{m-\xi}} & T_{m-\xi} < T < T_{m+\xi} \\ 1 & T > T_{m+\xi} \end{cases} \quad ()$$

*Discretization*

Following control volume based formulation as shown in Patankar and Murthy [16], Eqns. (30) and (18) can be written in the general form of conservation equation as shown (20),

$$\nabla \cdot \boldsymbol{J} = S \tag{20}$$

$$\boldsymbol{J} = (\boldsymbol{\rho}\vec{\boldsymbol{V}}\boldsymbol{\varphi}) - \nabla\, \tau\nabla\emptyset \tag{21}$$

where

$$\vec{\boldsymbol{V}} = \boldsymbol{u}\ \hat{\imath} + v\,\hat{\jmath}$$

Using the Divergence theorem, Eqn. (20) can be discretized as,

$$\sum_{\boldsymbol{f}} (\vec{\boldsymbol{J}_f} \cdot \vec{A_f}) = \overline{S}\Delta V \tag{22}$$

Where $\overline{S}$ represents the linearized source terms as shown,

$$\overline{S} = S_c + S_P\emptyset_P \tag{23}$$

The discretized energy equation using fully implicit time stepping is written as,

$$a_P\,\emptyset_P = \sum a_{nb}\,\emptyset_{nb} + a_P^0\emptyset_P^0 + d \tag{24}$$

Using Upwind differencing scheme, the coefficients are computed as

$$\begin{cases} a_{nb} = D_f + [\![-F_f, 0]\!] \\ a_P = \left(\sum a_{nb}\right) - S_P\Delta V + \sum_f F_f \\ b = a_P^0\emptyset_P^0 + d \\ d = S_c\Delta V \end{cases} \tag{25}$$

The diffusion term is calculated using,

$$D_f = \frac{\tau A_f}{\delta} \tag{26}$$

Where $\delta$ denotes the distance between the line joining two adjacent cell centroids or the shortest distance between the cell centroid and the corresponding face if the cell is a boundary cell. The convective term is calculated using,

$$F_f = \rho\,\vec{\boldsymbol{V}_f} \cdot \vec{A_f} \tag{27}$$

### 1. Source Term based formulation

The discretized coefficients for source term based formulation are shown in Table .

**Table : Discretized coefficients for Source Term based formulation**

| $\emptyset$ | $\boldsymbol{\rho}$ | $\boldsymbol{\tau}$ | $\boldsymbol{a_P^0}$ | $\boldsymbol{S_c}$ | $\boldsymbol{S_p}$ |
|---|---|---|---|---|---|
| **T** | $\boldsymbol{\rho C}$ | k | $\boldsymbol{\rho C}\dfrac{\Delta V}{\Delta t}$ | $\rho\dfrac{(\Delta H^0 - \Delta H)}{\Delta t}$ | 0 |

Enthalpy update at each iteration is shown (28) [11]

$$[H_p]_{n+1} = [H_p]_n + \frac{a_p}{a_p^0}\,c\lambda\,([T_p]_n - T_{m)} \tag{28}$$

To prevent overshoot or undershoot of nodal latent heat, if more than one computational cell undergoes phase change simultaneously, nodal latent heat values are bound as (29),

$$[H_p]_{n+1} = \begin{cases} set\ [H_p]_{n+1} = L & if [H_p]_{n+1} > L \\ set\ [H_p]_{n+1} = 0 & if [H_p]_{n+1} < 0 \end{cases} \quad (29)$$

The nodal latent heat values are updated according to the difference in nodal temperatures computed from the energy equation (15) and the phase change temperature. In case of freezing, nodal latent heat values would be positive due to latent heat evolution. Thus, the source term in Eqn. (14) would act as a heat source until freezing has been completed, after which there would be no heat generation. To account for the absorption of latent heat during melting, the energy equation source term from Eqn. (15) would be negative and act as a heat sink until melting is complete. Since the corrections in nodal latent heat values are driven by temperature differences, at convergence the temperature of all cells undergoing phase change would be the phase change temperature, $T_m$, for isothermal phase change.

Meanwhile, $\frac{a_p}{a_p^0}$ accounts for the influence of time-step and grid size and is referred to as a "normalizing factor" . For a refined grid, greater cells would undergo phase change at a given instant due to a higher $a_P$ value. The increase in time-step size would result in more cells undergoing phase change, which is reflected in a decrease of $a_P^0$.

**2. Apparent heat capacity formulation**

Table  shows the discretized coefficients for the apparent heat capacity method. The primary advantage of this approach is that due to the absence of source terms, the problem can be solved using a standard conduction code with non-linear specific heat. The treatment of non-linear material properties has been described by Patankar  . Further details regarding implementation have been given in Refs. [11],[14].

**Table : Discretized coefficients for apparent heat capacity based formulation**

| $\emptyset$ | $\rho$ | $\tau$ | $a_P^0$ | $S$ |
|---|---|---|---|---|
| **T** | $\rho C_{app}$ | k | $\rho C_{app} \frac{\Delta V}{\Delta t}$ | 0 |

Convection Diffusion Phase change

The coupled momentum and energy equations are solved using a control volume based discretization procedure according to the flow chart shown in Fig. 5. A fully implicit time discretization procedure and upwind differencing scheme is used to evaluate combined convection and diffusion coefficients. The momentum and continuity equations are solved using the SIMPLE algorithm   to obtain the velocity fields. The energy equations are solved using the apparent heat capacity formulation, since the enthalpy updating scheme at under-relaxation factor, $\lambda = 0.01$  caused divergence due to temperature oscillations. Decreasing $\lambda$ further leads to a rapid increase in computation time.

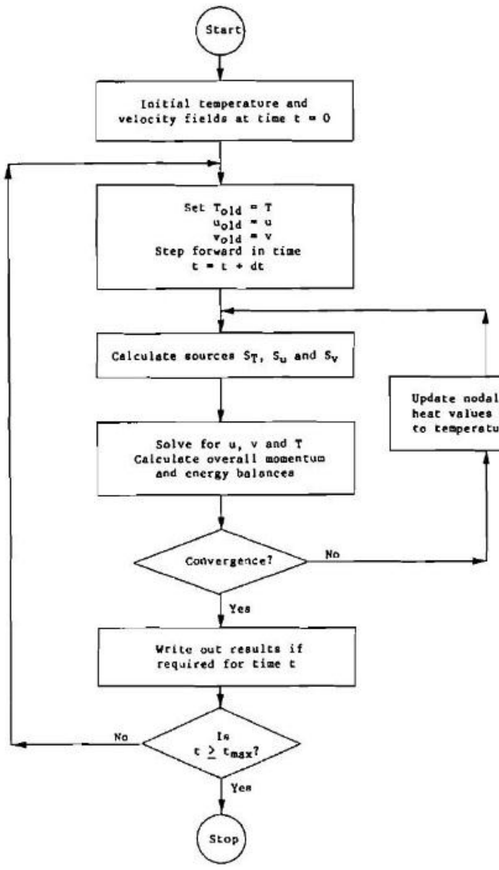

Start
Initial temperature and velocity fields at time t = 0
Set T_old = T
u_old = u
v_old = v
Step forward in time
t = t + dt
Calculate sources S_T, S_u and S_v
Update nodal latent heat values according to temperature
Solve for u, v and T
Calculate overall momentum and energy balances
Convergence?
No
Yes
Write out results if required for time t
No
Is t ≥ t_max?
Yes
Stop

**Fig. 5 Flow chart of computational algorithm [8].**

To reduce computational time, a maximum of 100 iterations per time step are performed for momentum, pressure and energy equations. Table 3 and Table4 show the solver convergence criteria and under-relaxation factors respectively.

**Table 3: Solver convergence criteria for convection-diffusion problem**

| | |
|---|---|
| Momentum | 1e-4 |
| Pressure | 1e-3 |
| Energy | 1e-4 |
| Unsteady energy balance | 1e-3 |

**Table 4: Under-relaxation factors for convection-diffusion problem**

| | |
|---|---|
| Momentum | 0.4 |
| Pressure | 0.7 |
| Energy | 0.9 |

## VALIDATION

In order to determine the validity of the numerical methods, the results from the SIMPLE algorithm and upwind differencing schemes were compared to results from the literature.

*SIMPLE algorithm validation*

The SIMPLE algorithm was used to solve a driven lid problem with a dimensionless square geometry for each side, shown in Fig. 6. A 60 x 60 mesh was used to solve the problem.

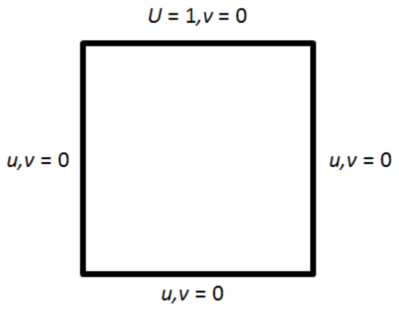


**Fig. 6. Domain for lid driven velocity problem.**

The results of this calculation were compared directly to Ghia et al. [10] for the vertical centerline of the *u* velocity and the horizontal centerline of the *v* velocity. The results of this calculation are shown in Fig. 7.

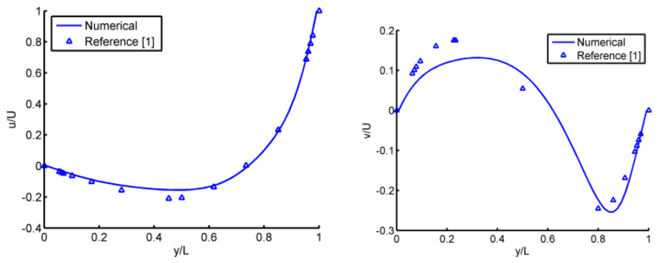


**Figure 7. Comparison of current numerical study to results from Ghia et al. [10] for *u* velocity (left) and *v* velocity (right).**

For the *u* and *v* velocities, the current model under-predicted the actual results by up to 35% error. While this value is high, the overall trend of the results was correct and a significant portion of the results were within 5% error. The large error in the center of the domain was possibly due to the mesh not being fine enough. However, results for a smaller grid mesh often diverged. Another cause for the large error was possibly due to the codes inability to converge to a low pressure prime term. Figure 8 shows the convergence per number of iterations observed in solving the system.

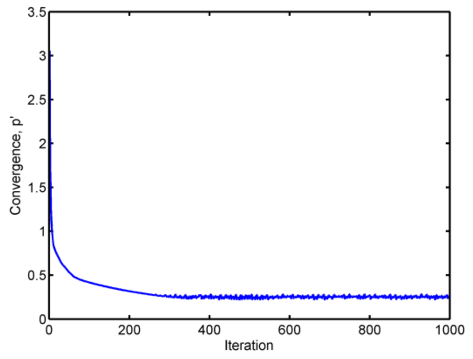


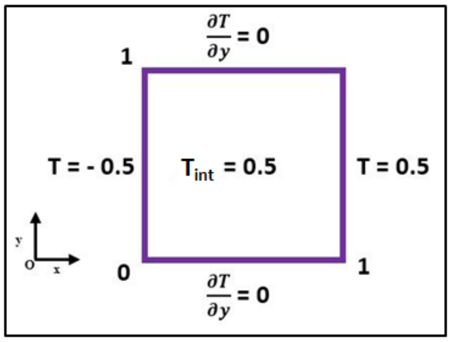
**Fig. 8. Convergence versus increasing iteration calculation for the SIMPLE algorithm.**

As can be seen in Fig. 8, the convergence began to oscillate between values when it reached *p'* = 0.37, which was much larger than the set tolerance of 0.001. This tolerance was within the range of difference between the current results and the actual results, and was a significant contributor to the error of the results. Another potential contributor to the error was the fact that the boundary conditions were forced to their values. Originally, the calculations used the momentum balance to determine the boundary conditions. However, this often produced large errors in the predicted values, specifically with the *u* velocity, whose boundary condition was being predicted as 0.617 instead of 1. Since the induced error in forcing the boundary conditions was lower than the error produced by the momentum balance, a forced value of 1 was set to the top *u* velocity terms. Although the error was significant, the results of the SIMPLE algorithm were deemed accurate enough to be used in the overall calculation.

*Energy equation validation*

The enthalpy updating scheme (enthalpy-porosity method) is validated against a well-studied benchmark problem shown in Fig. 9 [17],[18]. The problem considers conduction only phase change in a square cavity. The boundary and initial conditions are shown in Fig. 9. The material properties are listed in

Table 5.



$\frac{\partial T}{\partial y} = 0$
1
T = - 0.5
$T_{int}$ = 0.5
T = 0.5
y
o
x
0
1
$\frac{\partial T}{\partial y} = 0$

**Fig. 9. Domain for diffusion only phase change**

**Table 5: Material properties for validation case for diffusion only phase change**

| *Property* | *Magnitude* |
|---|---|
| $k$ | 1 |
| $\rho$ | 1 |
| c | 1 |
| $\Delta H$ | 5 |
| $T_{freeze}$ | 0 |
| $T_{init}$ | -0.5 |

Table  shows the solver convergence criteria. Unsteady energy balance is computed at each time step as shown in Eqn. (30) . The resulting set of algebraic (discretized) equations is

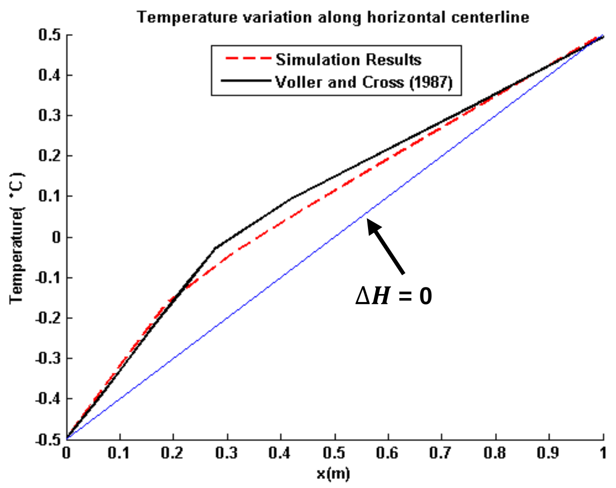


**Fig. 10. Comparison of temperature variation along horizontal centerline for 10 x 10 mesh with simulation results of Cross *et* at t=500s.**

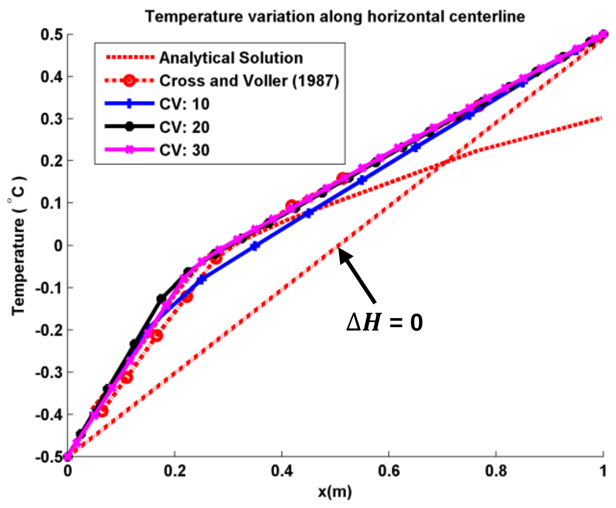


**Fig. 11. Comparison of temperature variation along horizontal centerline for different grid sizes with analytical solution and simulation results of Cross et al at t=500s.**

solved using Line by Line Tridiagonal Matrix Algorithm (TDMA) solver.

$$Q_{error} = Q_{x=0} + Q_{abs} + Q_{x=L} \tag{30}$$

$$Q_{x=0} = \Delta t \int_0^Y k \frac{\partial T}{\partial x} dy$$

$$Q_{x=L} = -\Delta t \int_0^Y k \frac{\partial T}{\partial x} dy$$

$$Q_{abs} = \rho \oiint c(T - T^o) + (\Delta H - \Delta H^0)\, dx\, dy$$

**Table 6: Solver convergence criteria for diffusion only problem**

| | |
|---|---|
| Solver tolerance | 1E-6 |
| Unsteady energy balance | 1E-5 |

A time step of 5s is used and the end time for the simulation is 500s. The computational time is 2s. From Fig. 10 and Fig. 11 it can be seen that the results from the enthalpy updating scheme agree well with the results by Cross *et al*.

## RESULTS

The results are validated against Cross *et al.*. The morphological constant, $C = 200$. Table compares the location of the solidification front at t=100s. From Table 7, it can be seen that the computed results are very close to the results reported, with a 5.6% error.

Table Comparison of solidification front location at t =100s

| Computed | Cross *et al.* |
|---|---|
| 0.183 | 0.194 |

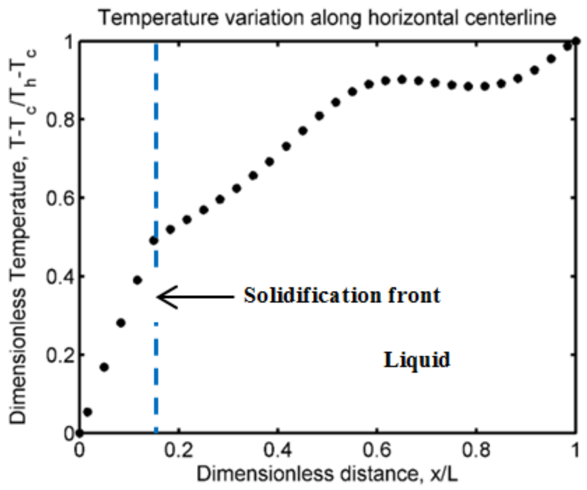


**Fig. . Plot of non-dimensional temperature along horizontal centerline with non-dimensional length.**

Figure 7 shows a clear difference in the temperature profile observed from solid to liquid. In the solid region, the temperature change is linear until the solidification front is reached, where an abrupt change in the temperature gradient occurs. This section is no longer linear, with the buoyancy of the liquid section mixing the surrounding temperature. There is also a local maximum around x/L = 0.6, which corresponds to the maximum velocity observed in the PCM. The *u* and *v* velocity profiles are shown in Fig. 8.

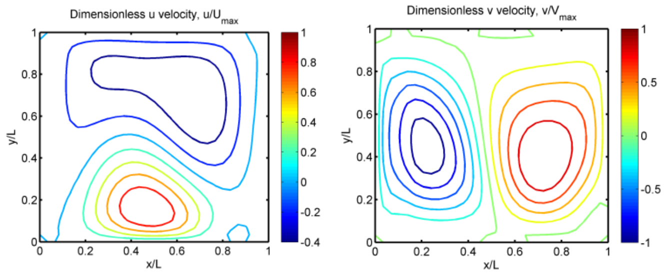


**Fig. 8. Velocity contours of the *u* component (left) and *v* component (right).**

From Fig. 8, it can be observed that the peak *u* velocity occurs at 0.7, which is slightly after the local maximum observed in Fig. 7. However, if the *u* and *v* components are combined together, then the peak temperature is near that point. In order to understand the temperature profile further, Fig. 9 provides the temperature contour for the domain.

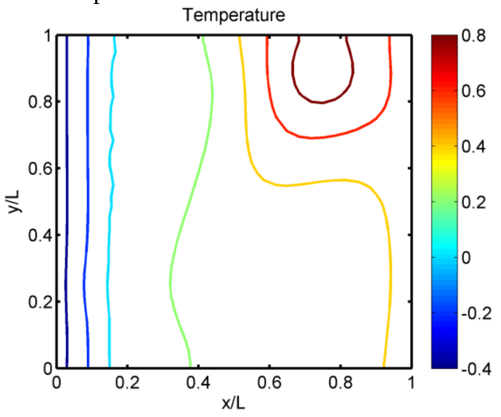


**Fig. 9. Temperature contour (in °C) of the calculated domain.**

While the melting front matches the literature results very closely, observation of the temperature contour of the domain displays that the numerical calculation was not perfect. Near the top right corner of the domain, the calculated temperature reaches 0.8 °C, which was higher than the boundary temperature of 0.5 °C. While it is not clear why this numerical artifact occurs in the calculation, there are several factors that are potential causes. The most likely cause was the low solving tolerance used to calculate the answer. As is mentioned in the SIMPLE algorithm validation, the SIMPLE algorithm does not always converge to a desired tolerance. Because of this, limiters were set in all the SIMPLE algorithm calculations to accept and record the tolerance level calculated and then the algorithm was allowed to continue. These convergence values were on the order of $10^{-2}$, which was around the same magnitude of the calculated velocity values. Since the convergence was high, the continuity equation was only satisfied up to that accuracy and thus the convection term in certain sections of the domain did not conserve energy. Because of this imbalance, the temperature at certain sections was able to increase past the maximum temperature.

Further problems were shown clearly with the existence of the *v* component of velocity inside the solid region of the foam. From Fig. 8, the *v* velocity is almost at its maximum negative value where there should be no velocity due to the existence of the solid region. This issue was a result of the morphological constant used. At its current value, the morphological constant did not raise the *A* source term to a high enough value to negate the velocity profile created in the calculation. Although the validation case used a morphological value of 200, the velocity observed in the solid region was able to be completely removed by increasing the constant significantly. Figure 10 shows the *v* velocity component contour with a morphological value of 20000.

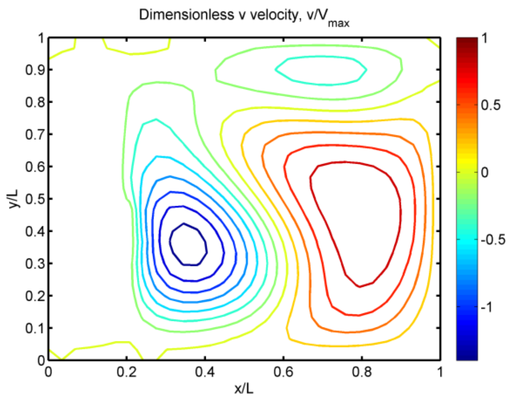


**Fig. 10. *V* velocity distribution over the domain with *C* = 20000.**

Figure 10 shows a drastic change in the velocity profile. With the increased constant, the velocity is no longer propagating into the solid region and is no longer symmetric. Furthermore, observation of the temperature contour shows that the artifact observed in Fig. 9 reduces from 0.8 °C to 0.55 °C. Increasing the morphological constant causes the calculation to become more realistic, which displays how important this particular parameter is. This clear delineation between the solid and liquid regions can be seen clearly in Fig. 11, which shows the velocity streamline superimposed over the temperature profile. It can also be observed from Fig. 11 that the buoyancy forces cause a vortex near the top-center of the domain. This vortex appears to have a significant effect on the temperature profile, as the temperature gradient becomes more significant as it approaches the vortex.

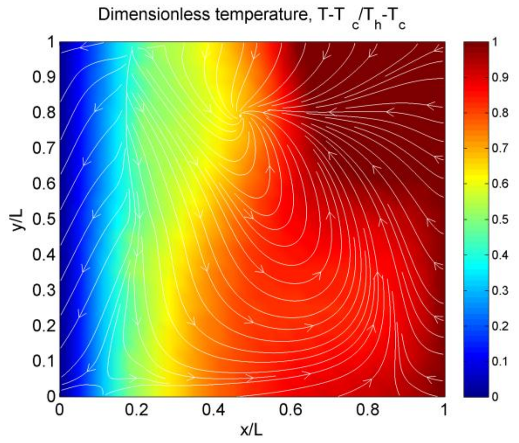


**Fig. 11. *V* velocity distribution over the domain with *C* = 20000.**

Finally, the morphological constant's effect on the velocity in the solid region was investigated for various *C* values at x/L=0.15, shown in Fig. 12.

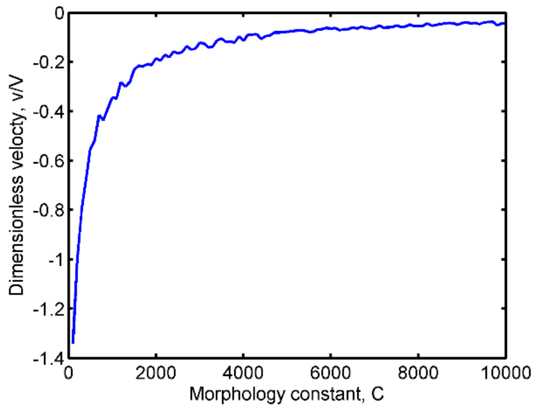


**Fig. 12. Morphology constant versus the *v* component of velocity at x/L=0.15.**

Figure 12 displays an exponential decay of the velocity observed in the solid region with increasing *C*. After a morphology constant of 4000, the trend slows down and the decrease in *v* velocity becomes asymptotic. However, *C*=200 is in the region with the largest change in *v*, supporting the fact that the morphology constant needs to be set to a larger value.

## CONCLUSION

The solidification process of a PCM was successfully modeled and validated against using the literature. While the predicted results were close, the existence of numerical artifacts increased the uncertainty that the model was accurately measuring the process. It was claimed that the inaccuracies were primarily due to the low tolerance of the SIMPLE algorithm as well as the value of the morphological constant used. An investigation was performed on the morphological constant and it was discovered that if the value of the constant was not high enough the velocity was not properly constrained in the solid region of the PCM.

Further progress needs to be done with regards to the SIMPLE algorithm code to improve the tolerance produced. Improvement to the SIMPLE code will improve results throughout the rest of the model, since the errors are the largest with that code. This is proven further by the SIMPLE algorithm's inability to reduce below a certain tolerance as well as its inability to converge when the mesh is increased.

## REFERENCES

[1] Kandasamy, R., Wang, X., and Mujumdar, A., "Transient cooling of electronics using phase change material (PCM)-based heat sinks," *Applied Thermal Engineering*, **28**, (2007): 1047-1057.

[2] Yang, Y., Wang, Y., "Numerical simulation of three-dimensional transient cooling application on a portable electronic device using phase change material," *International Journal of Thermal Sciences*, **51**, (2012): 155-162.

[3] Kandasamy, R., Wang, X., and Mujumdar, A., "Transient cooling of electronics using phase change material (PCM)-based heat sinks," *Applied Thermal Engineering*, **28**, (2008): 1047-1058.

[4] Voller, V. R., Cross, M., and Markatos, N. C., "An enthalpy method for convection/diffusion phase change," *International Journal for Numerical Methods in Engineering*, **24**, (1987): 271-284.

[5] Swaminathan, C. R., and Voller, V. R., "A general enthalpy method for modeling solidification processes," *Metallurgical transactions B*, **23B**, (1992): 651-664.

[6] Swaminathan, C. R., Voller, V. R., "On the enthalpy method," *International Journal for Numerical Methods Heat Fluid Flow*, **3**, (1993): 233-244.

[7] Nayak, K. C., Saha, S. K., Srinivasan, K., and Dutta, P., "A numerical model for heat sinks with phase change materials and thermal conductivity enhancers," *International Journal of Heat Mass Transfer*, **49**, (2006): 1833-1844.

[8] Brent, A. D., Voller, V. R., and Reid, K. J., "Enthalpy-porosity technique for modeling convection-diffusion phase change: Application to the melting of a pure metal," *Numerical Heat Transfer*, **13**, (1988): 297-218.

[9] Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, Hemisphere, Washington, 1980.

[10] Ghia, U., Ghia, K. N., Shin, C. T., "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *Journal of Computational Physics*, **48**, (1982) 387-411.

[11] Voller, Vaughan R., and C. Prakash. "A fixed grid numerical modelling methodology for convection-diffusion mushy region phase-change problems." *International Journal of Heat and Mass Transfer* **30,** (1987): 1709-1719.

[12] Voller, V. R. "An overview of numerical methods for solving phase change problems." *Advances in numerical heat transfer* **1**, (1997): 341-380.

[13] Swaminathan, C. R., and V. R. Voller. "Towards a general numerical scheme for solidification systems." *International Journal of Heat and Mass Transfer* **40**, (1997): 2859-2868.

[14] Hu, Henry, and Stavros A. Argyropoulos. "Mathematical modelling of solidification and melting: a review." *Modelling and Simulation in Materials Science and Engineering* **4**, (1996): 371.

[15] Jackson, Galen R., et al. "Modeling Thermal Storage in Wax-Impregnated Foams with a Pore-Scale Submodel." *Journal of Thermophysics and Heat Transfer* (2015): 1-8.

[16] Murthy, J. Y., 2002, Unpublished Course Notes, Numerical methods in heat, mass and momentum transfer

[17] K. Morgan, 'A numerical analysis of freezing and melting with convection', *Comp. Meth. App. Eng.*, **28**, (1981): 275-284

[18] Gartling, D. K. *Finite element analysis of convective heat transfer problems with change of phase*. No. SAND-77-2037C; CONF-780712-2. Sandia Labs., Albuquerque, N. Mex.(USA), 1978.

## ANNEX

## MATLAB CODE

```
clc;close all;clear all
%Meshgrid in (x,y) form
%Solver 2D diffusion only phase change with initial temperature Ti
%Aim is to determine melt front propogation
%---------------------<< ENTHALPY-POROSITY METHOD >>----------------------%
% Solving for ENTHALPY as independent variable (H)
% Primary Variable ->  H = h (sensible) + delta_H (latent)
% with H ; S_H = 0 ?????
% phi -> h = C*T
%convectivr term coeff -> none
%transient term coef -> rho ;
%diffusive term coef -> k
% source term -> d(rho*delta_H)/dt

%rectangular cavity with prescribed temperatures on sides
Bout=200:200:10000;
for bit=1:length(Bout)
%% ----------------------Geometric and Material Props---------------------%
%%
[l,w,CVx,CVy,rho,rho_ref,beta,k,T_melt,Cp,mu,latent_heat,g,Pr] = material_prop();
T_ref = T_melt;
%% ------------- Mesh Generate ; Arrays for Post Processing---------------%
%%
[x_increment,y_increment,xface,yface,xx,yy,xp,yp,x,y] = mesh_gen(CVx,CVy,l,w);
% Can this be done later???????
%% --------------TIME STEPPING AND BOUNDARY CONDITION DETAILS-------------%
%%
[end_time,delta_t,timesteps,b_conditions,zone,T_initial] = time_bc_details(CVx,CVy);
% CHECK FOR T_REF AND HANGING T's
%% -------------------------INITIAL CONDITIONS---------------------------%
%%
 [delta_H,delta_H_old,lfrac,initial_state,initial_Temperature,...
    uint,vint,p] = initial_conditions(CVx,CVy,latent_heat,b_conditions,T_initial);
%%
%intializing T_old which will carry
%forward previous values with initial temperature
T_old = initial_Temperature(2:CVy+1,2:CVx+1);

%% -----------------------CONVERGENCE CRITERIA----------------------------%
%%
[energy_TOL,urelax_energy,my_lambda,TOL_energy_imbalance] = solver_tolerance();

%% ------------------------------TIME LOOP--------------------------------%%
count = 0;
tic
u=uint;
v=vint;
uold=uint;
vold=vint;
Temperature=initial_Temperature;
% -------------------------TIME LOOP STARTS------------------------------- %
for time = 0:delta_t:end_time
    fprintf('\n\nTime=%f\n',time);
    count=count+1;
    uold=u;
    vold=v;
    %% ------------------------SOURCE TERMS-------------------------------%
    %%
    %-----------------------------ENERGY EQUATION-------------------------%

    %%
    % ADD MOMENTUM HERE
    %% Energy Solver call
    %%
    T_currentiter=T_old;
    T_current=T_currentiter;
    T_current(1,1)=T_currentiter(1,1);
    tol=1e-3;
    Titer=0;
    conv=1;
   while conv>tol
    T_currentiter=T_current;
       Titer=Titer+1;

    [ap0,b_source,lfrac] = source_term_energy_cp(xface,yface,CVx,CVy,delta_H_old,delta_H,rho,Cp,delta_t,T_currentiter);
    delta_H_old = delta_H;

    [u,v,p]=ProjectSimpleProgram(u,v,p,T_current,lfrac,delta_t,uold,vold,Bout(bit));
    [iter,iter_uns_energy,ap,an,as,aw,ae,ab,b,...
    F_e,F_w,F_n,F_s,D_e,D_w,D_n,D_s,D_b,u,v,...
    Temperature,T_current,delta_H,delta_H_old,q] = energy_eqn_solver(initial_Temperature,T_old,delta_H_old,...
    b_source,ap0,time,end_time,u,v,lfrac);
    Tb_left = b_conditions(1,1); Tb_right = b_conditions(1,2);
    qb_top = b_conditions(1,3); qb_bottom = b_conditions(1,4);
%% Calculating temperatures at faces where flux is prescribed
%%
del_bn = abs(yy(1,1) - yface(1,1));
del_bs = abs(yface(CVy,1) - yy(CVy,1));
Temperature(1,:) = (qb_top+ ((k./del_bn).*Temperature(2,:)))./(k./del_bn);
Temperature(CVy+2,:) = (qb_bottom + ((k./del_bs).*Temperature(CVy+1,:)))./(k./del_bs);

    %% Momentum solver
        conv=max(max(abs(T_current-T_currentiter)));
    fprintf('Temperature: iteration=%d  Convergence=%f\n',Titer,conv);

if Titer==5
    break
else
    continue
end


   end
       if max(max(Temperature))>Tb_right+1;
        fprintf('Temperature too hot: Time:%f',time);
        break
    end
    if min(min(Temperature))<Tb_left-.05;
        fprintf('Temperature too cold: Time:%f',time);
        break
    end
     figure(1)
     contour(v);colorbar;
%     figure(2)
%     contour(Temperature);colorbar;
    %conv=max(max(abs(T_current-T_currentiter)));
    %fprintf('Temperature: iteration=%d  Convergence=%f\n',Titer,conv);
    %% ---------------------Set old values------------------------------------
    %%
    T_old = T_current;
    initial_Temperature = Temperature; %???? CHECK ????

    %% -------------------------------PLOT----------------------------------%%
    %%
     %%
    %-----Liquid Fraction-----%
    %lfrac = delta_H./latent_heat;
    %figure(1)
    %    hold on
    %    plot(time,lfrac(floor(CVy/2),1),'k*')
    %    hold off
    %M_1(count) = movies_filled_temp(count,Temperature,u,v);
    %M_2(count) =movies_lfrac(u,v,count);
end
% ------------------------TIME LOOP ENDS----------------------------------%
%%
%movie2avi (M_1,'TemperatureProfile.avi','fps',5,'quality',100);
%movie2avi (M_2,'Meltfraction.avi','fps',5,'quality',100);
toc
%%
%%-------------------------POST PROCESSING------------------------------%%

%%
%% Extract Boundary Conditions
%%

vout(bit,:)=v(floor(CVy/2),:);
% figure(2)
% plot(x,(Temperature(floor(CVy/2),:)+0.5)/(.5+.5),'k*')
% ydata = Temperature(floor(CVy/2),:);
% % plot_data = [x ydata];
% % csvwrite('plot_data_10.dat',plot_data);
% title('Temperature variation along horizontal centerline'),xlabel('Dimensionless distance, x/L'),ylabel('Dimensionless Temperature, T-T_c/T_h-
T_c')
% legend(['CV: ',num2str(CVx) ' Time step size: ',num2str(delta_t)])
% figure(3)
% pcolor(xp,flipud(yp),lfrac),shading interp,
% title(' Spatial Variation of Liquid fraction with time'),xlabel('x'),ylabel('y'),colorbar
% %legend(['CV:' num2str(CVx)],['time step size:' num2str(delta_t)])
% figure(4)
% pcolor(x,flipud(y),Temperature),shading interp,
% title('Temperature'),xlabel('x'),ylabel('y'),colorbar
% %legend(['CV:' '\n Time step size:' num2str(CVx),num2str(delta_t)])
% figure(5)
% contour(x,flipud(y),Temperature),
% title('Temperature'),xlabel('x/L'),ylabel('y/L'),colorbar
end
plot(Bout,vout(:,2)/max(max(v)))

function [l,w,CVx,CVy,rho,rho_ref,beta,k,T_melt,Cp,mu,latent_heat,g,Pr] = material_prop()
% All Geometric and Material Properties listed here
%   Detailed explanation goes here
%% Cavity Dimensions and control volumes
%%
l = 1; %length in m
w = 1; %width in m
CVx = [20];
CVy = [20];
%%
rho = 1; %density in kg/m^3
rho_ref = 1;
beta = 3e-3; %Volumetric thermal expansion coefficient
k = .001; %in W/m C
T_melt = 0; %Phase change temperature in C
Cp = 1; %specific heat in J/kg*K
mu = 1; %Dynamic Viscosity in kg/(m.s)
latent_heat = 5; %latent heat in J/kg
%%
g = 1000; % m/s^2
Pr = 2.16e-2; %Prandtl Number
end


function [delta_H,delta_H_old,lfrac,initial_state,initial_Temperature,...
    u,v,p] = initial_conditions(CVx,CVy,latent_heat,b_conditions,T_initial)
%% Initialize nodal latent heats -> indicates initial state (solid/liquid)
%%
%%--------------------> 0 - solid ; latent_heat - liquid <-----------------
initial_state = latent_heat;
%%
%Set nodal latent heat values according to initial state
```

```
delta_H_old = ones([CVy,CVx]).*initial_state;
delta_H = delta_H_old; %set initial value for t=0
if(initial_state == latent_heat)
    lfrac = ones([CVy,CVx]);
else
    lfrac = zeros([CVy,CVx]);
end

%% Extract Boundary Conditions
%%
Tb_left = b_conditions(1,1); Tb_right = b_conditions(1,2);
qb_top = b_conditions(1,3); qb_bottom = b_conditions(1,4);
%%  Dummy rows, columns added to show boundary temperatures------%
%%
initial_Temperature= ones([CVy+2,CVx+2]).*T_initial;
initial_Temperature(:,1) = Tb_left;
initial_Temperature(:,CVx+2) = Tb_right;

%% MOMENTUM EQUATIONS
%% %% Velocities
%%
u = ones([CVy,CVx+1])*1e-4;v = ones([CVy+1,CVx])*1e-4;p = ones([CVy,CVx])*0;
u(:,CVx+1)=0;u(:,1)=0;u(CVy,:)=0;u(1,:)=0;
v(CVy+1,:)=0;v(1,:)=0;v(:,CVx)=0;v(:,1)=0;
end


function [end_time,delta_t,timesteps,b_conditions,zone,T_initial] = time_bc_details(CVx,CVy)
%Time Step Details + BC + Initial Conditions
%% ---------------------------TIME STEPPING DETAILS----------------------%
%%
end_time = 100; %time in s
delta_t = 10; %---dt---
timesteps = end_time/delta_t; % No.of Time-steps -- NOTE--
%% -----------------------------BOUNDARY CONDITIONS-----------------------%
%% ENERGY EQUATION
%%
T_initial = 0.5;
Tb_left = -0.5; %in degC
Tb_right = 0.5;
qb_top = 0; %in W/m^2
qb_bottom = 0;
b_conditions = [Tb_left Tb_right qb_top qb_bottom]; %Array of BC's
%% MOMETUM EQUATION
%%

%% ----Identify interior,boundary, edge nodes and label corresponding CV--%
%%
[zone] = create_zones(CVx,CVy);
end


function [x_increment,y_increment,xface,yface,xx,yy,xp,yp,x,y] = mesh_gen(CVx,CVy,l,w)
x_increment = l./CVx;
y_increment= w./CVy;
[xface,yface] = meshgrid(0:x_increment:l,0:y_increment:w);%check
for i = 1:(size(xface,2)-1) %Generate cell centroids in x
    xp(i) = (xface(1,i)+xface(1,i+1))./2;
end
for i = 1:(size(yface,1)-1)%Generate cell centroids in y
    yp(i) = (yface(i,1)+yface(i+1,1))./2;
end
[xx,yy] = meshgrid(xp,yp); %Generate 2d gridpoints
x = zeros([1,(CVx+2)]);
y = zeros([(CVy+2),1]);
x(1,2:(CVx+1)) = xp;
y(2:(CVy+1),1) = yp;
x(1,1) = 0.0;
x(1,CVx+2) = l;
y(1,1) = 0;
```

```
y(CVy+2,1) = w;


end


function [ustar,vstar,p]=ProjectSimpleProgram(u,v,p,T,lfrac,dt,uold,vold,B)
%% Calls the simple algorithm for a natural convection process
% Nx=Nodes in the x direction
% Ny=Nodes in the y direction
% lfrac=liquid fraction

%% Input constants

[l,w,CVx,CVy,rho,rho_ref,betta,~,T_melt,~,mu,latent_heat,g,~] = material_prop;
[dx,dy,~,~,~,~,~,~,~,y] = mesh_gen(CVx,CVy,l,w);

alpha=0.7;
Nx=CVx;
Ny=CVy;
%% Find T face values
Tavg=zeros(Ny-1,Nx);
ppnew=zeros(Ny,Nx);
ustar=zeros(Ny,Nx+1);
vstar=zeros(Ny+1,Nx);
Au=zeros(Ny,Nx+1);
Av=zeros(Ny+1,Nx);
% for i=1:Nx
% Tavg(Ny+1,i)=T(Ny,i);
% Tavg(1,i)=T(1,i);
% end


epsu=zeros(Ny,Nx+1);
epsv=zeros(Ny+1,Nx);
% lfrac=delta_H/latent_heat;
% if lfrac<0
%     lfrac=0;
% elseif lfrac>1
%     lfrac=1;
% end
%
%
 for i=1:Nx
    for j=2:Ny
        Tavg(j-1,i)=(T(j,i)+T(j-1,i))/2;
        epsv(j-1,i)=lfrac(j-1,i);
        epsu(j-1,i)=lfrac(j-1,i);
%     end
    end
 end
  for i=1:Nx
    for j=1:Ny
         epsv(j,i)=lfrac(j,i);
        epsu(j,i)=lfrac(j,i);
            end
 end
epsu(:,Nx+1)=lfrac(:,Nx);
epsv(Ny+1,:)=lfrac(Ny,:);
%% Input initial guess matrix
for i=1:Nx
    for j=1:Ny
        ppnew(j,i)=1;
    end
end
for i=1:Nx+1
    for j=1:Ny
        ustar(j,i)=u(j,i);
    end
end
```

```
for i=1:Nx
    for j=1:Ny+1
        vstar(j,i)=v(j,i);
    end
end
% uold=ustar;
% vold=vstar;
%
% ppnew(Ny,:)=0;
ppnew(Ny,Nx)=0;
iter=0;
%% Setup SIMPLE while loop
bP=ones(Ny,Nx);
while max(max(abs(ppnew)))>0.001
    u=ustar;
    v=vstar;
    iter=iter+1;
    pp=ppnew;
%% Setup a value matrix
awu=zeros(Ny,Nx-1);aeu=awu;anu=awu;asu=awu;bu=awu;apu0=awu;
awv=zeros(Ny-1,Nx);aev=awv;anv=awv;asv=awv;bv=awv;apv0=awv;

for i=1:Nx+1
    for j=1:Ny
        awu(j,i)=mu*dy/dx;
        aeu(j,i)=mu*dy/dx;
        anu(j,i)=mu*dx/dy;
        asu(j,i)=mu*dx/dy;
%         apu0(j,i)=rho*dy*dx/dt;
        apu0(j,i)=rho*dx*dy/dt;
        Au(j,i)=-B*(1-epsu(j,i))*dx*dy;
        %Au(j,i)=0;
    end
end
for i=1:Nx-1
    for j=2:Ny-1
        bu(j,i+1)=dy*(p(j,i)-p(j,i+1));
    end
end
bu(:,1)=0;
bu(:,Nx+1)=0;
for i=1:Nx
    for j=1:Ny+1
        awv(j,i)=mu*dy/dx;
        aev(j,i)=mu*dy/dx;
        anv(j,i)=mu*dx/dy;
        asv(j,i)=mu*dx/dy;
%         apv0(j,i)=rho*dy*dx/dt;
        apv0(j,i)=rho*dx*dy/dt;
        Av(j,i)=-B*(1-epsv(j,i))*dx*dy;
        %Av(j,i)=0;
    end
end
bv=zeros(Ny+1,Nx);
for i=2:Nx-1
    for j=1:Ny-1
        bv(j+1,i)=dx*dy*rho_ref*betta*g*(Tavg(j,i)-T_melt)+dx*(p(j,i)-p(j+1,i));
    end
end
% bv(1,:)=0;
% bv(Ny+1,:)=0;
%% Setup boundary conditions for COLM


apu=(anu+asu+awu+aeu-Au*dx*dy+apu0);
apv=(anv+asv+awv+aev-Av+apv0);

[ustar,apup]=lblTDMA(apu,aeu,awu,asu,anu,bu,Au,u,uold,apu0,rho,mu,dx,dy,1);
[vstar,apvp]=lblTDMA(apv,aev,awv,asv,anv,bv,Av,v,vold,apv0,rho,mu,dx,dy,2);
% figure(1)
% contour(ustar)
% figure(2)
% contour(vstar)
%% Setup pressure correction from COM

aE=zeros(Ny,Nx);aN=aE;aS=aE;aW=aE;app0=zeros(Ny,Nx);
anu=ones(Ny,Nx+1)*mu*dx/dy;
anv=ones(Ny+1,Nx)*mu*dx/dy;


for i=1:Nx
    for j=1:Ny
        aE(j,i)=rho*dy/apup(j,i+1);
        aW(j,i)=rho*dy/apup(j,i);
        aN(j,i)=rho*dx/apvp(j+1,i);
        aS(j,i)=rho*dx/apvp(j,i);
        bP(j,i)=rho*(dy*(ustar(j,i)-ustar(j,i+1))+dx*(vstar(j,i)-vstar(j+1,i)));
    end
end

aP=aE+aW+aN+aS;
aE(Ny,Nx)=0;aW(Ny,Nx)=0;aN(Ny,Nx)=0;aS(Ny,Nx)=0;
aP(Ny,Nx)=1;
bP(Ny,Nx)=0;
ppnew=lblTDMApressure(aP,aE,aW,aS,aN,bP,pp,app0);
%ppnew(Ny,Nx)=0;
p=p+alpha*ppnew;

if iter==1000
    break
else
    continue
end



end
conv=max(max(abs(ppnew)));
fprintf('Velocity: iteration=%d  Convergence=%f\n',iter,conv);

function [xnew,ap]=lblTDMA(ap,ae,aw,as,an,b,A,x0,xold,ap0,rho,mu,dx,dy,s)
%% Function lblTDMA
%  Use TDMA in a line by line form to solve a 2D matrix with QUICK scheme
NJ=length(ap(1,:));
NI=length(ap(:,1));
fn=zeros(NI,NJ);fs=fn;fe=fn;fw=fn;
%% Set 1d terms
if s==1;
    Ap1d=zeros(1,NJ);
    Ae1d=zeros(1,NJ);
    Aw1d=zeros(1,NJ);
    B1d=zeros(1,NJ);
else
    Ap1d=zeros(1,NI);
    An1d=zeros(1,NI);
    As1d=zeros(1,NI);
    B1d=zeros(1,NI);
end
x=x0;
xnew=x;
%xnew(2,2)=x(2,2)+.1;
alpha=0.4; % Underrelaxation factor


% TDMA solver: x direction sweep
tol=1e-4;
conv=1;
iter=0;
while conv>tol
```

```
    %% Calculate Flux terms
    % Calculate in and out fluxes
    iter=iter+1;
    x=xnew;
    if s==1;
        for j=2:NJ-1
            for i=1:NI
            fe(i,j)=rho*x(i,j)*dy;
            fw(i,j)=rho*x(i,j-1)*dy;
            end
        end
    elseif s>1;
        for i=2:NI-1
            for j=1:NJ
            fn(i,j)=rho*x(i,j)*dx;
            fs(i,j)=rho*x(i-1,j)*dx;
            end
        end
    end
%% Setup new A values
for i=1:NI
    for j=1:NJ
        ae(i,j)=mu*dy/dx-fe(i,j)/2;
        aw(i,j)=mu*dy/dx+fw(i,j)/2;
        an(i,j)=mu*dx/dy-fn(i,j)/2;
        as(i,j)=mu*dx/dy+fs(i,j)/2;
    end
end

%% Setup 0 boundary conditions

for i=1:NI
    for j=1:NJ
    ae(i,NJ)=0;
    ae(i,1)=0;
    ae(NI,j)=0;
    ae(1,j)=0;

    aw(i,NJ)=0;
    aw(i,1)=0;
    aw(NI,j)=0;
    aw(1,j)=0;

    as(i,NJ)=0;
    as(i,1)=0;
    as(NI,j)=0;
    as(1,j)=0;

    an(i,NJ)=0;
    an(i,1)=0;
    an(NI,j)=0;
    an(1,j)=0;
    end
end

ap=(ae+aw+an+as+(fe-fw+fn-fs)-A);
ap(:,NJ)=1;
ap(:,1)=1;
ap(1,:)=1;
ap(NI,:)=1;


    if s==1

    for i=1:NI
        for j=1:NJ
            Ap1d(j)=ap(i,j);
            Ae1d(j)=ae(i,j);
            Aw1d(j)=aw(i,j);
            B1d(j)=b(i,j)+ap0(i,j)*xold(i,j);
            if i>1
                B1d(j)=B1d(j)+an(i,j)*x(i-1,j);
            end
            if i<NI
                B1d(j)=B1d(j)+as(i,j)*x(i+1,j);
            end
        end
        x1d=TDMASIMPLE(Ap1d,Ae1d,Aw1d,B1d);
        for k=1:NJ
            xnew(i,k)=x1d(k);
        end
    end
    else
    for j=1:NJ
        for i=1:NI
            Ap1d(i)=ap(i,j);
            An1d(i)=an(i,j);
            As1d(i)=as(i,j);
            B1d(i)=b(i,j)+ap0(i,j)*xold(i,j);
            if j>1
                B1d(i)=B1d(i)+aw(i,j)*x(i,j-1);
            end
            if j<NJ
                B1d(i)=B1d(i)+ae(i,j)*x(i,j+1);
            end
        end
        x1d=TDMASIMPLE(Ap1d,An1d,As1d,B1d);
        for k=1:NI
            xnew(k,j)=x1d(k);
        end
    end

    end
    xnew=x+alpha*(xnew-x);
    conv=max(max(abs(xnew-x)));
    if iter>300
        fprintf('Convergence not reached in velocity before iteration 300, actual convergence=%f\n',conv);
        break
    end
end

function x=TDMASIMPLE(Ap,ae,aw,B)

%% Function TDMA
%  Solve a triagonal matrix using the TDMA method
%  Program by: Galen Jackson
%aw=[-2 -1 -1 -1];ae=[-1 -1 -1 -2];ap=-aw-ae;b=[500*2 0 0 500*2];
%Ap=ap;
Ae=-ae;
Aw=-aw;
%B=b;
N=length(Ap);
for i=2:N
    r=Aw(i)/Ap(i-1);
    Ap(i)=Ap(i)-r*Ae(i-1);
    B(i)=B(i)-r*B(i-1);
end

x(N)=B(N)/Ap(N);

for i=(N-1):-1:1
    x(i)=(B(i)-Ae(i)*x(i+1))/Ap(i);
end

function xnew=lblTDMApressure(ap,ae,aw,as,an,b,x0,ap0)
%% Function lblTDMA
%  Use TDMA in a line by line form to solve a 2D matrix with QUICK scheme
NJ=length(ap(1,:));
NI=length(ap(:,1));
Ap1d=zeros(1,NJ);
```

```
Ae1d=zeros(1,NJ);
Aw1d=zeros(1,NJ);
B1d=zeros(1,NJ);
x=x0;
xnew=x;
% TDMA solver: x direction sweep
tol=1e-3;
iter=0;
conv=1;
while conv>tol
    iter=iter+1;
    x=xnew;
    for i=1:NI
        for j=1:NJ
            Ap1d(j)=ap(i,j);
            Ae1d(j)=ae(i,j);
            Aw1d(j)=aw(i,j);
            B1d(j)=b(i,j)+ap0(i,j)*x0(i,j);
            if i>1
                B1d(j)=B1d(j)+an(i,j)*x(i-1,j);
            end
            if i<NI
                B1d(j)=B1d(j)+as(i,j)*x(i+1,j);
            end
        end
        x1d=TDMASIMPLE(Ap1d,Ae1d,Aw1d,B1d);
        for k=1:NJ
            xnew(i,k)=x1d(k);
        end
    end
    conv=max(max(abs(xnew-x)));
    if iter>300
        fprintf('Convergence not reached in pressure term before iteration 300, actual convergence=%f\n',conv);
        break
    end
end


function [ap0,b_source,lfrac] = source_term_energy_cp(xface,yface,CVx,CVy,delta_H_old,delta_H,rho,Cp,dt,T_input)
% Source term for energy equation
[~,~,~,~,rho,~,~,~,T_melt,Cps,~,latent_heat,~,~] = material_prop();
T_melt1=T_melt-0.01;
T_melt2=T_melt+0.01;
lfrac=zeros([CVy,CVx]);
ap0 = zeros([CVy,CVx]);b_source = zeros([CVy,CVx]);
for i=1:CVy
    for j = 1:CVx
        delta_y = abs(yface(i+1,j)-yface(i,j));
        delta_x = abs(xface(i,j+1)-xface(i,j));
        if T_input(i,j)<T_melt1
            Cp=Cps;
            lfrac(i,j)=0;
        elseif T_input(i,j)<T_melt2
            Cp=Cps+latent_heat/(T_melt2-T_melt1);
            lfrac(i,j)=(T_input(i,j)-T_melt1)/(T_melt2-T_melt1);
        else
           Cp=Cps;
           lfrac(i,j)=1;
        end
        ap0(i,j) = rho.*Cp.*delta_x.*delta_y./dt; %Note Volume of cell
        %% Note divide by Cp -> check
        %
        b_source(i,j) = 0;
    end
end
end

function [iter,iter_uns_energy,ap,an,as,aw,ae,ab,b,...
    F_e,F_w,F_n,F_s,D_e,D_w,D_n,D_s,D_b,u,v,...
    Temperature,T_current,delta_H,delta_H_old,q] =
energy_eqn_solver(initial_Temperature,T_old,delta_H_old,b_source,ap0,time,end_time,u,v,lfrac)
% Check T_old , u, v
% Dont update values from initial conditions
%%
%-----------------------SOLVE ENERGY EQUATION----------------------------%

%% -------------------Extract Material Props------------------------------%
%%
[l,w,CVx,CVy,rho,~,~,k,T_melt,Cp,~,latent_heat,~,~] = material_prop();
%% -----------------------Extract grid data-------------------------------%
%%
[x_increment,y_increment,xface,yface,xx,yy,~,~,~,~] = mesh_gen(CVx,CVy,l,w);
%% ----------------------Extract Time steps data -------------------------%
[~,delta_t,~,b_conditions,~] = time_bc_details(CVx,CVy);
%% ----------------------Extract Tolerance/Convergence Criteria-----------%
[energy_TOL,urelax_energy,my_lambda,TOL_energy_imbalance] = solver_tolerance();
%% -----------------Compute Under-rleaxation for enthalpy update----------%
if(time <= end_time/20)
    lambda = my_lambda(1);
elseif(time > end_time/20 && time < end_time/10);
    lambda = my_lambda(2);
elseif(time >=end_time/10 && time <end_time/3);
    lambda = my_lambda(3);
else
    lambda = my_lambda(4);
end
%% ---------------------------DISCRETIZE-----------------------------%%
%%
% for i=CVy
%     for j=CVx
%         if lfrac(i,j)<1
%             Cpeff=0;
%         else
%             Cpeff=Cp;
%         end
%     end
% end
[ap,an,as,aw,ae,ab,b,...
    F_e,F_w,F_n,F_s,D_e,D_w,D_n,D_s,D_b,u,v] =...
    discretize_ept_uds_vel(b_conditions,T_old,b_source,ap0,...
    CVx,CVy,xface,yface,xx,yy,k,rho*Cp,u,v);
%% ----------------------------CALL SOLVER------------------------------%%
%%
time;
%Temperature is a (CVy+2,CVx+2) sized array
[Temperature,delta_H,iter] = solve_energy(initial_Temperature,urelax_energy,...
    ap,an,as,ae,aw,b,ap0,delta_H_old,CVx,CVy,energy_TOL,Cp,lambda,...
    T_melt,latent_heat);
iter;

T_current = Temperature(2:CVy+1,2:CVx+1);
%% -------------------Computing unsteady energy balance------------------%%
%%
[q,iter_uns_energy] = uns_energy_conv(Temperature,T_old,delta_H,delta_H_old,...
    CVx,CVy,TOL_energy_imbalance,x_increment,y_increment,delta_t,k,rho,Cp);
end


function [q] = energy_imbalance(Temperature,T_old,delta_H,delta_H_old,CVx,CVy,delta_x,delta_y,dt,k,rho,Cp)
% Child function to check unsteady energy balance
del_x = delta_x/2;
q_left = k.*dt.*delta_y.*(Temperature(2:CVy+1,2)-Temperature(2:CVy+1,1))./(del_x);
q_right = -k.*dt.*delta_y.*(Temperature(2:CVy+1,CVy+2)- Temperature(2:CVy+1,CVy+1))./(del_x);
q_abs_1 = rho*delta_x*delta_y*Cp.*(Temperature(2:CVy+1,3:CVx) - T_old(:,2:CVx-1));
q_abs_2 = delta_H(:,2:CVx-1) - delta_H_old(:,2:CVx-1);
q_abs = q_abs_1+q_abs_2;
q = [q_left q_abs(:,:) q_right];
end
```

```
function [delta_H] = enthalpy_update_iter(u,delta_H_old,ap,ap0,...
    Cp,lambda,T_ref,latent_heat,my_CVx,my_CVy)
%b_source = zeros([CVy CVx]);
delta_H = zeros([my_CVy,my_CVx]);
%% ---------------------------Updating Enthalpy------------------------%%%
%%
for i=1:my_CVy
    for j = 1:my_CVx
%         if(bcondition(1) > T_ref && bcondition(2) > T_ref && T_initial > T_ref)
%             enthalpy_update_term =0; % Heating liquid
%         elseif (bcondition(1) < T_ref && bcondition(2) < T_ref && T_initial < T_ref)
%             enthalpy_update_term =0; %Cooling solid
%         else
%             enthalpy_update_term = ap(i,j).*Cp.*lambda./ap0(i,j);
%         end
        enthalpy_update_term = ap(i,j).*Cp.*lambda./ap0(i,j);
        delta_H(i,j) = delta_H_old(i,j) + enthalpy_update_term.*(u(i,j) - T_ref);
        %%
        %%Setting bounds
        %%
        if(delta_H(i,j) > latent_heat)
            delta_H(i,j) = latent_heat;
        end
        if(delta_H(i,j)<0)
            delta_H(i,j) = 0;
        end
        %%  ------Update end-------%%%
        %%
        %b_source(i,j) = ap0(i,j).*(delta_H_old(i,j)-delta_H(i,j)); %%%% CHECK SIGNS %%%
    end
end
end
```

```
function [u,delta_H,iter] = solve_energy(guess_value,urelax,...
    my_ap,my_an,my_as,my_ae,my_aw,my_b,my_ap0,delta_H_old,my_CVx,my_CVy,my_TOL,...
    Cp,lambda,T_ref,latent_heat)
%% Call from main_energy side
%%
%  [Temperature,delta_H,iter] = solve_energy(initial_Temperature,urelax,...
%     ap,an,as,ae,aw,b,ap0,delta_H_old,CVx,CVy,TOL,Cp,lambda,T_ref,latent_heat);
%%
%Call solvers for energy equation and update latent heats, under-relax
%equations
%  Line by Line TDMA code inside the while loop to capture every iteration
delta_H=zeros([my_CVy,my_CVx]);
error = 1;
iter =0;
u = guess_value;
while error> my_TOL
    iter = iter + 1;
    uold = u;
    % Solver Sweeps along rows and columns and solve using TDMA
    [u] = solver_sweeps(u,my_ap,my_an,my_as,my_ae,my_aw,my_b,my_CVx,my_CVy);
    %Update Nodal Latent heat at each iteration
    % u is a (CVy+2,CVx+2) sized array
    u_update = u(2:my_CVy+1,2:my_CVx+1);
    %[delta_H] = enthalpy_update_iter(u_update,delta_H_old,my_ap,my_ap0,Cp,lambda,T_ref,...
    %    latent_heat,my_CVx,my_CVy);
    error= max(max(abs((uold-u)./u)));
    % hold on
    % subplot(3,2,4),plot(iter,error,'k*'),title('Iter vs error'),xlabel('iter'),ylabel('error')
    %Under-relax
    u = uold + urelax.*(u-uold);
end
end
```

```
function [ap,an,as,aw,ae,ab,b,...
    F_e,F_w,F_n,F_s,D_e,D_w,D_n,D_s,D_b,u,v] =...
    discretize_ept_uds_vel(b_condition,T_old,b_source,ap0,...
    CVx,CVy,xface,yface,xx,yy,k,rho,u,v)
%% DIFFUSION/CONVECTION and ISOTHERMAL PHASE CHANGE ONLY with UPWINDING
%%
%Discretization scheme for Neumann BC on top and bottom with source terms according to
%Fully Implicit time stepping
%Flux in positive X,Y direction -> Cartesian
%% ------------------------INITIALZING----------------------------------%
%%
%% Convective terms
F_e = zeros([CVy,CVx]);F_w = zeros([CVy,CVx]);F_n = zeros([CVy,CVx]);F_s = zeros([CVy,CVx]);
f_correction = zeros([CVy,CVx]);
%F_b = zeros([CVy,CVx]); % Geometric BC
%% Diffusive Terms
%%
D_e = zeros([CVy,CVx]);D_w = zeros([CVy,CVx]);D_n = zeros([CVy,CVx]);D_s = zeros([CVy,CVx]);
D_b = zeros([CVy,CVx]);
%% a terms
%%
ap = zeros([CVy,CVx]);aw = zeros([CVy,CVx]);ae = zeros([CVy,CVx]);
an = zeros([CVy,CVx]);as = zeros([CVy,CVx]);ab =  zeros([CVy,CVx]);
a_sum = zeros([CVy,CVx]);
b = zeros([CVy,CVx]);
%% Extract Boundary Conditions
%%
Tb_left = b_condition(1,1); Tb_right = b_condition(1,2);
qb_top = b_condition(1,3); qb_bottom = b_condition(1,4);
%% Velocities
%%
%u = zeros([CVy,CVx+1]);v = zeros([CVy+1,CVx]);
%u(2:CVy-1,2:CVx) = 0;
%v(2:CVy,2:CVx-1) = 0;
%u = @(x)(0);%(power(x,2)+1); %calculate "u" velocity (x) component
%v = @(y)(0);%(power(y,2)+2); %calculate "v" velocity (y) component
%%
%%%%------CHECK SIGNS FOR FLUX ; SINCE AREA VECTORS ARE ALWAYS OUTWARD AND
%%%%FLUX IS IN POSITIVE X, Y (Cartesian)
%Geometric BC all
%%
%% ------------------------------INTERIOR-------------------------------%
%%
for i=2:(CVy-1)
    for j=2:(CVx-1)
        delta_y = abs(yface(i+1,j)-yface(i,j));
        delta_x = abs(xface(i,j+1)-xface(i,j));
        del_w = abs(xx(i,j)-xx(i,j-1)); %Note that j-1 = west
        del_e = abs(xx(i,j+1)-xx(i,j)); %Note that j+1 = east
        del_n = abs(yy(i,j)-yy(i-1,j)); %Note that i-1 = north
        del_s = abs(yy(i+1,j)-yy(i,j)); %Note that i+1 = south
        %% Convective terms
        %%
        F_e(i,j) = rho.*u(i,j+1).*delta_y;
        F_w(i,j) = rho.*u(i,j).*delta_y;
        F_n(i,j) = rho.*v(i,j).*delta_x;
        F_s(i,j) = rho.*v(i+1,j).*delta_x;
        f_correction(i,j) = +F_e(i,j)-F_w(i,j)+F_n(i,j)-F_s(i,j);
        %% Diffusive Terms
        %%
        D_e(i,j) = k.*delta_y./del_e;
        D_w(i,j) = k.*delta_y./del_w;
        D_n(i,j) = k.*delta_x./del_n;
        D_s(i,j) = k.*delta_x./del_s;
        %% a = F + D - max out
        %%
        aw(i,j) = max(F_w(i,j),0) + D_w(i,j);
        ae(i,j) = max(-F_e(i,j),0) + D_e(i,j);
        an(i,j) = max(-F_n(i,j),0) + D_n(i,j);
        as(i,j) = max(F_s(i,j),0) + D_s(i,j);
        a_sum(i,j) = aw(i,j)+ae(i,j)+an(i,j)+as(i,j); %Sum of nearest neighbours
        ap(i,j) = a_sum(i,j) + ap0(i,j) + f_correction(i,j);
        %% b term
        %%
        b(i,j) = (ap0(i,j).*T_old(i,j))+ b_source(i,j);
    end
end
%% ---------------------------------TOP----------------------------------%
%%
for j=2:CVx-1
    i=1;
    delta_y = abs(yface(i+1,j)-yface(i,j));
    delta_x = abs(xface(i,j+1)-xface(i,j));
    del_w = abs(xx(i,j)-xx(i,j-1));
    del_e = abs(xx(i,j+1)-xx(i,j));
    del_s = abs(yy(i+1,j)-yy(i,j));
    %% Convective Terms
    %%
    F_e(i,j) = rho.*u(i,j+1).*delta_y;
    F_w(i,j) = rho.*u(i,j).*delta_y;
    %F_b(i,j) = rho*v(yy(i,j)+(delta_y./2))*delta_x;
    F_s(i,j) = rho.*v(i+1,j).*delta_x;
    f_correction(i,j) = +F_e(i,j)-F_w(i,j)-F_s(i,j);%+F_b(i,j) % Geometric BC
    %% Diffusive Terms
    %%
    D_e(i,j) = k.*delta_y./del_e;
    D_w(i,j) = k.*delta_y./del_w;
    D_s(i,j) = k.*delta_x./del_s;
    %% a terms - F + D %% Max out
    %%
    aw(i,j) = max(F_w(i,j),0) + D_w(i,j);
    ae(i,j) = max(-F_e(i,j),0) + D_e(i,j);
    as(i,j) = max(F_s(i,j),0) + D_s(i,j);
    a_sum(i,j) = aw(i,j)+ae(i,j)+as(i,j); %w-e-b-s
    ap(i,j) = a_sum(i,j)+ap0(i,j)+f_correction(i,j);
    %% b term
    %%
    b(i,j) = (ap0(i,j).*T_old(i,j)) - (qb_top.*delta_x) + b_source(i,j); %note "-" in b_flux
end
%% -------------------------------BOTTOM----------------------------------%
%%
for j=2:CVx-1
    for i=CVy
        delta_y = abs(yface(i+1,j)-yface(i,j));
        delta_x = abs(xface(i,j+1)-xface(i,j));
        del_w = abs(xx(i,j)-xx(i,j-1));
        del_e = abs(xx(i,j+1)-xx(i,j));
        del_n = abs(yy(i,j)-yy(i-1,j));%Note that i-1 = north
        %% Convective Terms
        %%
        F_e(i,j) = rho.*u(i,j+1).*delta_y;
        F_w(i,j) = rho.*u(i,j).*delta_y;
        F_n(i,j) = rho.*v(i,j).*delta_x;
        %F_b(i,j) = rho*v(yy(i,j)-(delta_y./2))*delta_x;
        f_correction(i,j) = +F_e(i,j)-F_w(i,j)+F_n(i,j);%-F_b(i,j) % Geometric BC
        %% Diffusive Terms
        %%
        D_w(i,j) = k.*delta_y./del_w;
        D_e(i,j) = k.*delta_y./del_e;
        D_n(i,j) = k.*delta_x./del_n;
        %% a terms - F + D %% Max out
        %%
        aw(i,j) = max(F_w(i,j),0) + D_w(i,j);
        ae(i,j) = max(-F_e(i,j),0) + D_e(i,j);
        an(i,j) = max(-F_n(i,j),0) + D_n(i,j);
        a_sum(i,j) = aw(i,j)+ae(i,j)+an(i,j);%w-e-b-n
        ap(i,j) = a_sum(i,j)+ap0(i,j)+f_correction(i,j);
        %% b term
        %%
        b(i,j) = (ap0(i,j).*T_old(i,j)) + (qb_bottom.*delta_x) + b_source(i,j);
```

```
  end
end
%% ----------------------------LEFT---------------------------------------%
%%
for i=2:CVy-1
  for j=1
    delta_y = abs(yface(i+1,j)-yface(i,j));
    delta_x = abs(xface(i,j+1)-xface(i,j));
    del_e = abs(xx(i,j+1)-xx(i,j));
    del_n = abs(yy(i,j)-yy(i-1,j));%Note that i-1 = north
    del_s = abs(yy(i+1,j)-yy(i,j));
    del_b = abs(xx(i,j) - xface(i,j));
    %% Convective terms
    %%
    F_e(i,j) = rho.*u(i,j+1).*delta_y;
    F_n(i,j) = rho.*v(i,j).*delta_x;
    F_s(i,j) = rho.*v(i+1,j).*delta_x;
    %F_b(i,j) = rho*u(xx(i,j)-(delta_x./2))*delta_y;
    f_correction(i,j) = +F_e(i,j)+F_n(i,j)-F_s(i,j);%-F_b(i,j) %Geometric BC
  %% Diffusive Terms
  %%
    D_e(i,j) = k.*delta_y./del_e;
    D_n(i,j) = k.*delta_x./del_n;
    D_s(i,j) = k.*delta_x./del_s;
    D_b(i,j) = k.*delta_y./del_b;
    %% a terms - F+D - max out
    %%
    ae(i,j) = max(-F_e(i,j),0) + D_e(i,j);
    an(i,j) = max(-F_n(i,j),0) + D_n(i,j);
    as(i,j) = max(F_s(i,j),0) + D_s(i,j);
    ab(i,j) = D_b(i,j); %Geometric BC
    a_sum(i,j) = as(i,j)+ae(i,j)+an(i,j)+ab(i,j);%b-e-n-s
    ap(i,j) = a_sum(i,j)+ap0(i,j)+ f_correction(i,j);
    %% b terms
    %%
    b(i,j) = (ap0(i,j).*T_old(i,j))+ (ab(i,j).*Tb_left) + b_source(i,j); %Note Last 2 terms ->Check
  end
end
%% -------------------------------RIGHT-----------------------------------%
%%
for i=2:CVy-1
  for j=CVx
    delta_y = abs(yface(i+1,j)-yface(i,j));
    delta_x = abs(xface(i,j+1)-xface(i,j));
    del_w = abs(xx(i,j)-xx(i,j-1));
    del_n = abs(yy(i,j)-yy(i-1,j));
    del_s = abs(yy(i+1,j)-yy(i,j));
    del_b = abs(xface(i,j+1)-xx(i,j));
    %% Convective terms
    %%
    %F_b(i,j) = rho.*u(xx(i,j)+(delta_x./2)).*delta_y;
    F_w(i,j) = rho*u(i,j).*delta_y;
    F_n(i,j) = rho*v(i,j).*delta_x;
    F_s(i,j) = rho*v(i+1,j)*delta_x;
    f_correction(i,j) = -F_w(i,j)+F_n(i,j)-F_s(i,j);%F_b(i,j) %Geometric BC
    %% Diffusive terms
    %%
    D_w(i,j) = k.*delta_y./del_w;
    D_n(i,j) = k.*delta_x./del_n;
    D_s(i,j) = k.*delta_x./del_s;
    D_b(i,j) = k.*delta_y./del_b;
    %% a terms
    %%
    aw(i,j) = max(F_w(i,j),0) + D_w(i,j);
    an(i,j) = max(-F_n(i,j),0) + D_n(i,j);
    as(i,j) = max(F_s(i,j),0) + D_s(i,j);
    ab(i,j) = D_b(i,j); %Geometric BC
    a_sum(i,j) = as(i,j)+aw(i,j)+an(i,j)+ab(i,j);%w-b-n-s
    ap(i,j) = a_sum(i,j)+ap0(i,j)+f_correction(i,j);
    %% b terms
```

```
        %%
        b(i,j) = (ap0(i,j).*T_old(i,j))+(ab(i,j).*Tb_right)+ b_source(i,j);
    end
end
%%
 %%---------------------CORNER NODES DISCRETIZATION----------------------%%

%% ----------------------------LEFT TOP-----------------------------------
%%
for i=1
    for j=1
        delta_y = abs(yface(i+1,j)-yface(i,j));
        delta_x = abs(xface(i,j+1)-xface(i,j));
        del_s = abs(yy(i+1,j)-yy(i,j));
        del_e = abs(xx(i,j+1)-xx(i,j));
        del_b_w = abs(xx(i,j) - xface(i,j));
        %% Convective Terms
        %%
        F_e(i,j) = rho.*u(i,j+1).*delta_y;
        F_s(i,j) = rho*v(i+1,j)*delta_x;
        f_correction(i,j) = F_e(i,j)-F_s(i,j);
        %% Diffusive Terms
        %%
        D_s(i,j) = k.*delta_x./del_s;%No west
        D_e(i,j) = k.*delta_y./del_e; %No north
        Db_w = k.*delta_y./del_b_w; %No west
        D_b(i,j) = Db_w;
        %% a terms - max out
        %%
        ae(i,j) = max(-F_e(i,j),0) + D_e(i,j);
        as(i,j) = max(F_s(i,j),0) + D_s(i,j);
        ab(i,j) = D_b(i,j);% Geometric BC
        a_sum(i,j) = as(i,j)+ae(i,j)+ab(i,j);%b-e-b-s
        ap(i,j) =  a_sum(i,j)+ap0(i,j)+f_correction(i,j);
        %% b terms
        %%
        b_left_top = Db_w.*Tb_left - qb_top.*delta_x; %->Check
        b(i,j) = (ap0(i,j).*T_old(i,j))+ b_left_top + b_source(i,j);
    end
end
%% -----------------------------RIGHT TOP---------------------------------
%%
for i=1
    for j=CVx
        delta_y = abs(yface(i+1,j)-yface(i,j));
        delta_x = abs(xface(i,j+1)-xface(i,j));
        del_s = abs(yy(i+1,j)-yy(i,j));%No north
        del_w = abs(xx(i,j)-xx(i,j-1));%No east-adj
        del_b_e = abs(xface(i,j+1)-xx(i,j));
        %% Convective Terms
        %%
        F_w(i,j) = rho*u(i,j)*delta_y;
        F_s(i,j) = rho*v(i+1,j)*delta_x;
        f_correction(i,j) = -F_w(i,j)-F_s(i,j);
        %% Diffusive Terms
        %%
        D_s(i,j) = k.*delta_x./del_s;
        D_w(i,j) = k.*delta_y./del_w;
        Db_e = k.*delta_y./del_b_e; %No east-adj
        D_b(i,j) = Db_e;
        %% a Terms - max out
        %%
        aw(i,j) = max(F_w(i,j),0) + D_s(i,j);
        as(i,j) = max(F_s(i,j),0) + D_w(i,j);
        ab(i,j) = D_b(i,j); %Geometric BC
        a_sum(i,j) = as(i,j)+aw(i,j)+ab(i,j);%w-b-b-s ->Check
        ap(i,j) = a_sum(i,j)+ap0(i,j)+f_correction(i,j);
        %% b terms
        %%
        b_right_top =  (Db_e.*Tb_right)- qb_top.*delta_x;
        b(i,j) = (ap0(i,j).*T_old(i,j)) + b_right_top + b_source(i,j);
    end
end
%% ----------------------------LEFT BOTTOM-------------------------------
%%
for i=CVy
    for j=1
        delta_y = abs(yface(i+1,j)-yface(i,j));
        delta_x = abs(xface(i,j+1)-xface(i,j));
        del_n = abs(yy(i,j)-yy(i-1,j));
        del_e = abs(xx(i,j+1)-xx(i,j));
        del_b_w = abs(xx(i,j) - xface(i,j));
        %% Convective Terms
        %%
        F_e(i,j) = rho.*u(i,j+1).*delta_y;
        F_n(i,j) = rho*v(i,j)*delta_x;
        f_correction(i,j) = F_e(i,j)+F_n(i,j);
        %% Diffusive Terms
        %%
        D_n(i,j) = k.*delta_x./del_n;%No south-adj
        D_e(i,j) = k.*delta_y./del_e;%No west
        Db_w = k.*delta_y./del_b_w; %No west
        D_b(i,j) = Db_w; %Geometric BC
        %% a terms - max out
        %%
        ae(i,j) = max(-F_e(i,j),0) + D_e(i,j);
        an(i,j) = max(-F_n(i,j),0) + D_n(i,j);
        ab(i,j) = D_b(i,j); %Geometric BC
        a_sum(i,j) = an(i,j)+ae(i,j)+ab(i,j);%b-n-e-b
        ap(i,j) =  a_sum(i,j)+ap0(i,j)+f_correction(i,j);
        %% b terms
        %%
        b_left_bottom = Db_w.*Tb_left +qb_bottom.*delta_x;
        b(i,j) = (ap0(i,j).*T_old(i,j)) + b_left_bottom + b_source(i,j);
    end
end
%% -------------------------RIGHT BOTTOM--------------------------------
%%
for i=CVy
    for j=CVx
        delta_y = abs(yface(i+1,j)-yface(i,j));
        delta_x = abs(xface(i,j+1)-xface(i,j));
        del_n = abs(yy(i,j)-yy(i-1,j));
        del_w = abs(xx(i,j)-xx(i,j-1));
        del_b_e = abs(xface(i,j+1)-xx(i,j));
        %% Convective Terms
        %%
        F_w(i,j) = rho*u(i,j)*delta_y;
        F_n(i,j) = rho*v(i,j)*delta_x;
        f_correction(i,j) = -F_w(i,j)+F_n(i,j);
        %% Diffusive terms
        %%
        D_n(i,j) = k.*delta_x./del_n;%No south-adj
        D_w(i,j) = k.*delta_y./del_w;%No east-adj
        Db_e = k.*delta_y./del_b_e; %No east-adj
        D_b(i,j) = Db_e;
        %% a terms - F+D - max out
        %%
        aw(i,j) = max(F_w(i,j),0) + D_w(i,j);
        an(i,j) = max(-F_n(i,j),0) + D_n(i,j);
        ab(i,j) = D_b(i,j); %Geometric BC
        a_sum(i,j) = an(i,j)+aw(i,j)+ab(i,j);%b-n-e-b
        ap(i,j) = a_sum(i,j)+ap0(i,j)+f_correction(i,j);
        %% b terms
        %%
        b_right_bottom = (Db_e.*Tb_right) + qb_bottom.*delta_x;
        b(i,j) = (ap0(i,j).*T_old(i,j))+ b_right_bottom + b_source(i,j); %Note Last 2 terms
    end
end
end
```

```
function [ zone ] = create_zones(CVx,CVy)
zone = zeros([CVy,CVx]);
zone(2:(CVy-1),2:(CVx-1)) = 1; %interior nodes
zone(1,2:(CVx-1)) = 2; %top boundary
zone(CVy,2:(CVx-1)) = 3; %bottom boundary
zone(2:(CVy-1),1) = 4; %left boundary
zone(2:(CVy-1),CVx) = 5; %right boundary
zone(1,1) = 6; %Left top
zone(1,CVx) = 7; %Right top
zone(CVy,1) = 8; %Left bottom
zone(CVy,CVx) =9 ; %Right bottom
end

function [u] = solver_sweeps(u,my_ap,my_an,my_as,my_ae,my_aw,my_b,my_CVx,my_CVy)
%Do sweeps here
% --------column sweeps from left to right-------------
for j = 2: my_CVx+1
   u(2:my_CVy+1,j) = tdma(my_ap(:,j-1),my_as(:,j-1),my_an(:,j-1),(my_aw(:,j-1).*u(2:my_CVy+1,j-1))+my_b(:,j-1)+(my_ae(:,j-
1).*u(2:my_CVy+1,j+1)),my_CVy);
end
%-------------row sweeps from i = 1 to CVy ; top to bottom-----------------

for i = 2:my_CVy+1
 u(i,2:my_CVx+1) = tdma(my_ap(i-1,:),my_ae(i-1,:),my_aw(i-1,:),(my_an(i-1,:).*u(i-1,2:my_CVx+1))+my_b(i-1,:)+ (my_as(i-
1,:).*u(i+1,2:my_CVx+1)),my_CVx);
end

% <<<<---------Reverse sweeps------------------>>>>>

% --------column sweeps from right to left-------------
for j = my_CVx+1:-1:2
u(2:my_CVy+1,j) = tdma(my_ap(:,j-1),my_as(:,j-1),my_an(:,j-1),(my_aw(:,j-1).*u(2:my_CVy+1,j-1))+my_b(:,j-1)+(my_ae(:,j-
1).*u(2:my_CVy+1,j+1)),my_CVy);
end

% ------------row sweeps from bottom to top--------------

for i = my_CVy+1:-1:2
 u(i,2:my_CVx+1) = tdma(my_ap(i-1,:),my_ae(i-1,:),my_aw(i-1,:),(my_an(i-1,:).*u(i-1,2:my_CVx+1))+my_b(i-1,:)+ (my_as(i-
1,:).*u(i+1,2:my_CVx+1)),my_CVx);
end

end


function [energy_TOL,urelax_energy,my_lambda,TOL_energy_imbalance] = solver_tolerance()
%% Energy Equation
energy_TOL = 1e-4; % Solver tolerance
urelax_energy = 0.9; %Under-relaxation for energy equation
my_lambda = [0.01 0.01 0.01 0.01]; % Under-relaxation factor for enthalpy update
TOL_energy_imbalance = 1e-3; % Tolerance for unsteady energy balance
end


function u = tdma(a,b,c,d,N)
%TDMA
% a(i) = b(i)T(i+1)+c(i)T(i-1)+d(i)
% a denoted diag index = 0 , b=1 , c=-1, d = RHS , N= gridpoints
% T(i) = P(i)T(i+1)+Q(i) -> Recursion used

%Start with P(1),Q(1)
P = zeros([1,N]); % row vector
Q = zeros([1,N]); % row vector
u = zeros([1,N]); % row vector
P(1) = b(1)./a(1);
Q(1) = d(1)./a(1);

for i = 2:N
    P(i) = b(i)./(a(i) - (c(i).*P(i-1)));
    Q(i) = (d(i)+(c(i).*Q(i-1)))./(a(i) - (c(i).*P(i-1)));
end
u(N) = Q(N);
%Back substitution
for i = N-1:-1:1 %Check
    u(i) = (P(i).*u(i+1))+Q(i);
end
% fprintf('P')
% disp(P);
% fprintf('Q')
% disp(Q);
% fprintf('\n')
end

function [q,iter_uns_energy] = uns_energy_conv(Temperature,T_old,delta_H,delta_H_old,...
    CVx,CVy,TOL_energy_imbalance,x_increment,y_increment,delta_t,k,rho,Cp)
% Parent function to check unsteady energy balance
uns_energy_error = 1;
    q = zeros([CVy,CVx]);
    iter_uns_energy =0;
    while uns_energy_error > TOL_energy_imbalance
        iter_uns_energy = iter_uns_energy +1;
        q_old = q;
        [q] = energy_imbalance(Temperature,T_old,delta_H,delta_H_old,CVx,CVy,x_increment,y_increment,...
            delta_t,k,rho,Cp);
        uns_energy_error= max(max(abs((q_old-q)./q)));
    end


end

function [u,iter]= linebylinetdma_underrelaxation(guess_value,urelax,...
    my_ap,my_an,my_as,my_ae,my_aw,my_b,my_CVx,my_CVy,my_TOL )
%LINE BY LINE TDMA
%Solving by line by line TDMA
error = 1;
iter =0;
u = guess_value;
while error> my_TOL
    iter = iter +1;
    uold = u;
% --------column sweeps from left to right-------------
for j = 2: my_CVx+1
   u(2:my_CVy+1,j) = tdma(my_ap(:,j-1),my_as(:,j-1),my_an(:,j-1),(my_aw(:,j-1).*u(2:my_CVy+1,j-1))+my_b(:,j-1)+(my_ae(:,j-
1).*u(2:my_CVy+1,j+1)),my_CVy);
end
%-------------row sweeps from i = 1 to CVy ; top to bottom-----------------

for i = 2:my_CVy+1
 u(i,2:my_CVx+1) = tdma(my_ap(i-1,:),my_ae(i-1,:),my_aw(i-1,:),(my_an(i-1,:).*u(i-1,2:my_CVx+1))+my_b(i-1,:)+ (my_as(i-
1,:).*u(i+1,2:my_CVx+1)),my_CVx);
end
% <<<<---------Reverse sweeps------------------>>>>>

% --------column sweeps from right to left-------------
for j = my_CVx+1:-1:2
u(2:my_CVy+1,j) = tdma(my_ap(:,j-1),my_as(:,j-1),my_an(:,j-1),(my_aw(:,j-1).*u(2:my_CVy+1,j-1))+my_b(:,j-1)+(my_ae(:,j-
1).*u(2:my_CVy+1,j+1)),my_CVy);
end
% ------------row sweeps from bottom to top--------------
for i = my_CVy+1:-1:2
 u(i,2:my_CVx+1) = tdma(my_ap(i-1,:),my_ae(i-1,:),my_aw(i-1,:),(my_an(i-1,:).*u(i-1,2:my_CVx+1))+my_b(i-1,:)+ (my_as(i-
1,:).*u(i+1,2:my_CVx+1)),my_CVx);
end
error= max(max(abs((uold-u)./u)));
% hold on
% subplot(3,2,4),plot(iter,error,'k*'),title('Iter vs error'),xlabel('iter'),ylabel('error')
u = uold + urelax.*(u-uold);
end
end
```