The University of Texas at Austin
CSE 380: TOOLS/TECHNIQS COMPUTATIONAL SCI (64340)
Model Document: Heat Equation

by Alec Carruthers, UT EID: ac74895

Governing Equations, Nomenclature, and Boundary Conditions

The steady-state heat equation with constant coefficients in two dimensions is provided below:

$$-k\nabla^2 T(x,y) = q(x,y) \tag{1}$$

where k is the thermal conductivity, T is the temperature of the material, q is a heat source, x is the horizontal location/position, and y is the vertical location/position. Within the finite difference scheme, the variables x and y will be replaced by i and j, respectively. Each combination of i and j represents a node within the finite difference mesh. In the case of one dimensional heat conduction, Eq. 1 can be reduced to:

$$-k\nabla^2 T(x) = q(x) \tag{2}$$

With a grid composed of N nodes, the 1-Dimensional grid/discretization will have bounds [0,$N-1$], i.e. a length of N grid points. In the 2-Dimensional case, the grid points are defined within a mesh of size N_x x N_y, i.e. $0 \leq x \leq N_x - 1$ and $0 \leq y \leq N_y - 1$.

Nomenclature

(i) i: Increment in the x direction

(ii) j: Increment in the y direction

(iii) T_i, $T_{i,j}$: Temperature on the i^{th} node (1D) and the Temperature on the node at (i,j) (2D)

(iv) q_i, $q_{i,j}$: Heat source on the node at location i (1D) and the heat source on the node at (i,j) (2D)

(v) k: Thermal Diffusivity

(vi) T_{left}: Temperature at the left boundary (1D)

(vii) T_{right}: Temperature at the right boundary (1D)

(viii) T_{l_n},T_{r_n},T_{t_n},T_{b_n}: Temperature of the node at the left, right, top, or bottom boundaries at the node index n (2D)

(ix) N: Total number of discretization nodes (N = N_x * N_y)

(x) n: An index/counter starting from the left most node (1D) and the bottom left most node (2D). The count starts from a value of zero and traverses right. When a boundary is hit in the 2D case, the count continues on the next row up at the left most node

(xi) N_x: Number of nodes along the x-axis (equal to N for 1D)

(xii) N_y: Number of nodes along the y-axis

(xiii) T_n: The temperature at the n^{th} node.

The Dirichlet boundary conditions define that the temperature at the boundary of the system will be known. Since the problem is steady state, the temperature at the boundaries should remain constant at the boundaries for the 1D case. The temperature at these boundaries will be denoted:

$$T(0) = T_{left}, T(N-1) = T_{right}$$

However, the 2D case allows for the temperature to have spatial variation along the x and or y axes. As such, the Dirichlet boundary conditions can be defined as follows, assuming the 2D case:

$$T(0,y) = T_l(y), T(N_x,y) = T_r(y), T(x,0) = T_b(x), T(x,N_y) = T_t(x) \tag{3}$$

The spacing between each node in both the x and y directions will be found using the following equation:

$$\Delta x = \frac{N_x}{N_x - 1}, \Delta y = \frac{N_y}{N_y - 1} \tag{4}$$

where N_x and N_y are the total number of nodes, including the boundary nodes, along the x and y axes, respectively.

List of Assumptions

The following list of assumptions will used in this analysis:

(i) In the 1D case, a domain of size N, i.e. [0,N-1]

(ii) In the 2D case, a square domain, i.e. mesh, of size N_x x N_y will be used

(iii) Temperature changes/discretizations will only be measured along the x and y axes (no temperature change along either diagonal of the node of interest)

(iv) Equal sized intervals in the 1D case and equal sized mesh spacings for the 2D case. From Eq. 4, this would mean that $N_x = N_y$ and $\Delta x = \Delta y$

(v) The governing equation can be differentiated many times allowing for Taylor Series approximations of the heat transfer to be found

(vi) The diffusivity of the material is constant across both spatial dimensions

(vii) When a stencil is incomplete because it requires a node that lies outside the domain, the missing node will be assumed to have the temperature of the closest boundary cell. This case only arises during the 4th order approximation case. Since the boundary conditions are known and will be imposed by MASA, this seemed like a valid assumption.

Numerical Method

The primary idea behin applying finite difference approximations for some derivative term in a partial differential equation is to apply the Taylor Series expansion in both the forward and reverse directions. The degree/order of the approximation required determines the total number of Taylor Series expansions that need to be taken about a particular grid point. The first Taylor Series expansion about a grid point "x" in both the forward (Eq. 5) and reverse (Eq. 6) directions are defined below:

$$f_{i+1} = f(x+\Delta x) = f(x)+\Delta x f^{'}(x)+\frac{\Delta x^2}{2!}f^{''}(x)+\frac{\Delta x^3}{3!}f^{'''}(x)+\frac{\Delta x^4}{4!}f^{(4)}(x)+\frac{\Delta x^5}{5!}f^{(5)}(x)+\frac{\Delta x^6}{6!}f^{(6)}(x)+... \tag{5}$$

$$f_{i-1} = f(x-\Delta x) = f(x)-\Delta x f^{'}(x)+\frac{\Delta x^2}{2!}f^{''}(x)-\frac{\Delta x^3}{3!}f^{'''}(x)+\frac{\Delta x^4}{4!}f^{(4)}(x)-\frac{\Delta x^5}{5!}f^{(5)}(x)+\frac{\Delta x^6}{6!}f^{(6)}(x)+... \tag{6}$$

One can see that the odd components of the above equations cancel out, resulting in just the even components.

0.1 1-D Case, 2^{nd} Order Approximation

Applying the forward and backward approximations shown in Eq. 5 and 6 to the second order derivative of temperature in the Eq. 1, one can obtain:

$$T^{''}(x) \approx \frac{T_{i-1}-2T_i+T_{i+1}}{\Delta x^2} - \frac{\Delta x^2}{12}T^{(4)}(x)+... \tag{7}$$

where $T_{i+1} and T_{i-1}$ are the temperatures at the nodes in front of and behind node i. In terms of truncation error, Eq. 7 can be described as:

$$T^{''}(x_i) \approx \frac{T_{i-1}-2T_i+T_{i+1}}{\Delta x^2} + O(\Delta x)^2 \tag{8}$$

where $O(\Delta x^2)$ defines the leading truncation error term in the 2nd order finite difference approximation of the 1D heat conduction equation. Substituting Eq. 8 into 2 and solving for the temperature at the ith node (T_i), one obtains:

$$T_i = \frac{1}{2}[T_{i+1}+T_{i-1}+\frac{q_i \Delta x^2}{k}] + O(\Delta x)^2 \tag{9}$$

0.2 1-D Case, 4^{th} Order Approximation

A similar procedure to what was done in the previous section can be applied here, but now the forward and backward Taylor Series expansions about a position x or node i need to consider the second closes nodes in both the forward and backward directions, i.e. $f(x+2\Delta x)$ and $f(x-2\Delta x)$. The resulting 4^{th} order approximation for the second derivation of temperature can be defined as follows:

$$T''(x_i) \approx \frac{-T_{i-2} + 16T_{i-1} - 30T_i + 16T_{i+1} - T_{i+2}}{12\Delta x^2} - \frac{\Delta x^4}{90} T^{(6)}(x_i) + ... \tag{10}$$

where T_{i+2} and T_{i-2} are the temperatures at nodes *i+2* and *i-2*, respectively. In terms of the leading truncation error term, Eq. 10 can be redefined as:

$$T''(x_i) \approx \frac{-T_{i-2} + 16T_{i-1} - 30T_i + 16T_{i+1} - T_{i+2}}{12\Delta x^2} + O(\Delta x)^4 \tag{11}$$

Substituting Eq. 11 into the 1-D heat equation (Eq. 2) and solving for T_i, one obtains:

$$T_i = \frac{1}{30}[-T_{i+2} + 16T_{i+1} + 16T_{i-1} - T_{i-2} + \frac{12q_i\Delta x^2}{k}] + O(\Delta x)^4 \tag{12}$$

0.3 2-D Case, 2^{nd} Order Approximation

For 2D finite difference problems, forward and backward Taylor Series expansions need to be taken about a central node in both the vertical and horizontal directions. Consequently, The second order finite difference approximation of the two dimensional, second derivative of temperature can be calculated as follows:

$$T''(x_i, y_j) = \frac{\partial^2 T}{\partial x_i^2} + \frac{\partial^2 T}{\partial y_j^2}$$

$$T''(x_i, y_j) \approx [\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + O(\Delta x^2) + \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta x^2} + O(\Delta x^2)]$$

$$T''(x_i, y_j) \approx [\frac{T_{i-1,j} + T_{i+1,j} - 4T_{i,j} + T_{i,j-1} + T_{i,j+1}}{\Delta x^2}] + O(\Delta x^2) \tag{13}$$

Since it was assumed that the 2D domain was square and had constant mesh spacing, the Δx equaled the Δy, hence why only Δx appears. Substituting the previous equation into Eq. 1, the temperature at a node with position (i,j) can be calculated as:

$$T_{i,j} = \frac{1}{4}[T_{i-1,j} + T_{i+1,j} + T_{i,j+1} + T_{i,j-1} + \frac{q_{i,j}\Delta x^2}{k}] + O(\Delta x)^2 \tag{14}$$

0.4 2-D Case, 4^{th} Order Approximation

Utilizing some of the principles outlined in the previous two sections, the fourth order approximation of the second derivative of temperature in two dimensions is defined as follows:

$$T''(x_i, y_j) \approx \frac{1}{12\Delta x^2}[-T_{i-2,j} + 16T_{i-1,j} - 60T_{i,j} + 16T_{i+1,j} - T_{i+2,j} - T_{i,j-2} + 16T_{i,j-1} + 16T_{i,j+1} - T_{i,j+2}] + O(\Delta x^4) \tag{15}$$

Substituting Eq. 15 into the two-dimensional heat equation (1) and solving for the temperature at the node positioned at (i,j), one can obtain:

$$T_{i,j} = \frac{1}{60}[-T_{i-2,j} + 16T_{i-1,j} + 16T_{i+1,j} - T_{i+2,j} - T_{i,j-2} + 16T_{i,j-1} + 16T_{i,j+1} - T_{i,j+2} + \frac{12q_{i,j}\Delta x^2}{k}] + O(\Delta x)^4 \quad (16)$$

0.5 1D and 2D Mesh Grids

It has been mentioned numerous times in the previous sections, but a node based approach will be used for this finite difference analysis.

1D Mesh/Grid

The following figure shows the discretized mesh grid for the 1D heat conduction case:

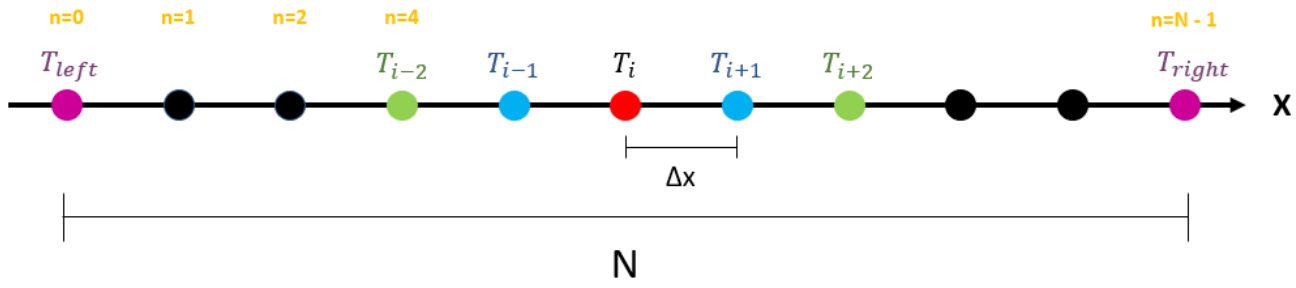

Figure 1: A sample 1D discretization mesh that will be used for iteratively solving the 1D heat conduction equation. The blue nodes pertain to both the second and fourth order second derivative approximations, while the green nodes are only used in the fourth order approximation.

2D Mesh/Grid

The mesh grid used for the 2D heat equation case is shown below:

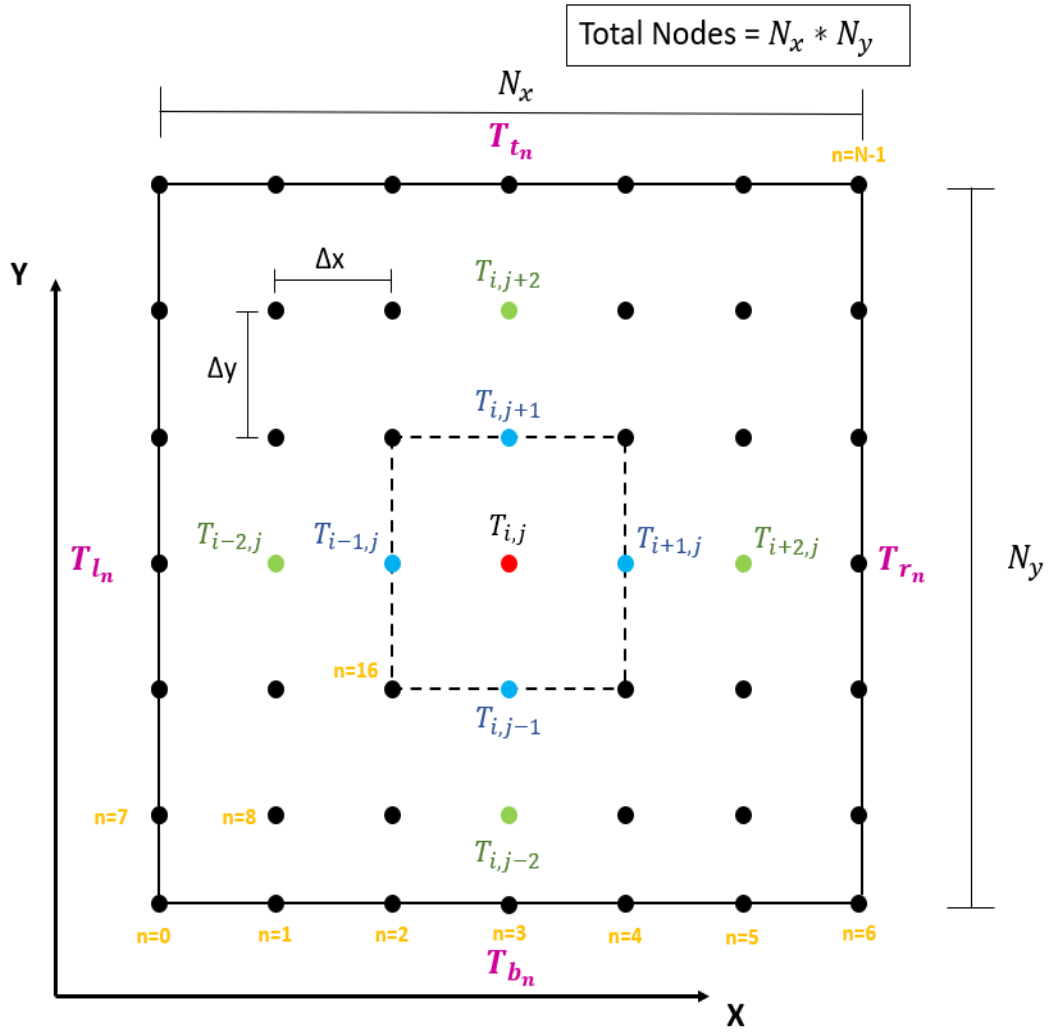

Figure 2: A sample 2D meshgrid that will be used for iteratively solving the 2D heat conduction equation. The blue nodes pertain to both the second fourth order second derivative approximations, while the green nodes are only used in the fourth approximation.

0.6 Linear System

1D, 2nd Order System:

It should be noted that for both of the 1D cases, indexing will start from the left most node (assuming the x direction is used). It is indexed using a value of zero and each node increments the index until the rightmost node is reached, corresponding to an index of N-1, where N is the total number of nodes used in the discretization mesh.

The linear system that can be used to solve the 1D, second order heat equation is shown below:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & -2 & 1 & 0 & & & & \vdots \\ 0 & 1 & -2 & 1 & 0 & & & \vdots \\ \vdots & 0 & 1 & -2 & 1 & 0 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & \ddots & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ T_{N-1} \end{bmatrix} = \begin{bmatrix} T_{left} \\ \frac{-q_1 \Delta x^2}{k} \\ \frac{-q_2 \Delta x^2}{k} \\ \vdots \\ \vdots \\ \vdots \\ \frac{-q_{N-2} \Delta x^2}{k} \\ T_{right} \end{bmatrix}$$

Using a starting index of zero, the square matrix of size (N-1) x (N-1) where N is the total number of nodes in the discretized mesh. The first and last rows contain all zeros except for the first and last elements, respectively, which are set to one. This is done to ensure that the boundary conditions are satisfied. The temperatures T_{left} and T_{right} represent the temperatures at the nodes lying on the left and right boundaries, respectively.

There are 3 non-zero terms in any interior row, where N is the total number of discretization points in the 1D mesh.

1D, 4th Order System

The one dimensional, 4th order heat equation analyzed in this project can be represented as follows:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 16 & -30 & 16 & -1 & & & & & \vdots \\ -1 & 16 & -30 & 16 & -1 & & & & \vdots \\ 0 & -1 & 16 & -30 & 16 & -1 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & -1 & 16 & -30 & 16 & -1 & 0 \\ \vdots & & & & -1 & 16 & -30 & 16 & -1 \\ \vdots & & & & & -1 & 16 & -30 & 16 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ T_{N-1} \end{bmatrix} = \begin{bmatrix} T_{left} \\ \frac{-12 q_1 \Delta x^2}{k} \\ \frac{-12 q_2 \Delta x^2}{k} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \frac{-12 q_{N-2} \Delta x^2}{k} \\ T_{right} \end{bmatrix}$$

The idea of ghost nodes to complete incomplete stencils was used in this project. In the 1-

dimensional case, the second and second to last nodes are the only nodes with an incomplete stencil, hence why the second and second to last rows of the A matrix look like they are missing a term relative to the other interior nodes. To complete the stencil, MASA is used to query for the exact temperature at the position of the ghost node. Since these ghost nodes do not exist in the domain, they cannot be represented in the A matrix. As a result, the contribution from these ghost nodes is added to the result of the matrix vector product for the nodes with an incomplete stencil. The number of non-zero terms in any interior row is either 4 or 5, where N is the total number of discretization nodes. **If a complete stencil can be formed, i.e. after the second row and before the second to last row, the number of non-zero terms in the row is 5. However along the second and second to last row, there are just 4 terms that are non-zero.** There are less terms in this these two rows based on the assumption that nodes lying outside the domain take on the value of the closest boundary node. As a result, the external node can be lumped together with the node actually on the boundary, reducing the number of non-zero points from five to four.

2D, Second Order System

Displaying a matrix for both of the 2D cases is somewhat cumbersome, so just a couple of the different regimes will be highlighted by themselves. The matrix below shows how the first few rows of the A matrix would be formatted in the 2D case. This pattern of a single one in the row with all other entries equaling zero occurs for the first and last $N_x + 1$ rows of A (Boundary conditions must be satisfied for all nodes along the top and bottom of the meshgrid. The plus one comes into play because even after passing over all of the nodes on the bottom of the meshgrid, the next node is a left boundary). Below is the format of the first three rows of A:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots \\ 0 & 1 & 0 & \cdots \\ 0 & 0 & 1 & \cdots \\ & & \vdots & \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} T_{n=0} \\ T_{n=1} \\ T_{n=2} \\ \vdots \end{bmatrix}$$

The temperature values are known at the positions being referenced above, so just the proper indexing needs to occur to retrieve them from MASA. These values are not shown as a function of q because they are the boundary temperatures. Inside the two rows of nodes representing the boundary, the A matrix will have the following form (assuming a complete stencil for visual purposes):

$$\begin{bmatrix} & & & & & \vdots & & & & & & & \\ \cdots & 0 & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & 0 & \cdots & \cdots & \cdots \\ \cdots & \cdots & 0 & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & 0 & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ T_n \\ T_{n+1} \\ T_{n+2} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{-q_n \Delta x^2}{k} \\ \frac{-q_{n+1} \Delta x^2}{k} \\ \frac{-q_{n+2} \Delta x^2}{k} \\ \vdots \end{bmatrix}$$

The horizontal dotted lines in between the coefficients a number of zeros equal to N_x nodes. These gaps arise from nodes in the stencil with positions (i,j-1) and (i,j+1). The -4 coefficient represent the central node for that iteration, i.e. (i, j), while the $(i+1, j)$ and $(i-1, j)$ nodes are represented by the one coefficients to the right and left of the central node coefficient, respectively. As a result, for the rows corresponding to nodes within the interior of the material and not set solely equal

to the boundary temperature, there are **5 non-zero terms**. The pattern shown above will occur across all of the rows that are not a boundary N_x - 2 times, i.e. it will occur at all the nodes along the row except for the two boundary nodes.

2D, Fourth Order The only change on the matrix A from the second to fourth order occurs on the interior nodes, so only that will be discussed here (see the first matrix in the previous section for what the first $N_x + 1$ rows look like). For the fourth order case, incomplete stencils will occur across all of the second outermost nodes, i.e. one node in from the boundary. The following matrix represents how the A matrix in the linear system will be set up for adjacent nodes with a complete stencil:

$$\begin{bmatrix}
 & & & & & & & & \vdots & & & & & & & & & & \\
\cdots & 0 & -1 & \cdots & 16 & \cdots & -1 & 16 & -60 & 16 & -1 & \cdots & 16 & \cdots & -1 & 0 & \cdots & \cdots & \cdots \\
\cdots & \cdots & 0 & -1 & \cdots & 16 & \cdots & -1 & 16 & -60 & 16 & -1 & \cdots & 16 & \cdots & -1 & 0 & \cdots & \cdots \\
\cdots & \cdots & \cdots & 0 & -1 & \cdots & 16 & \cdots & -1 & 16 & -60 & 16 & -1 & \cdots & 16 & \cdots & -1 & 0 & \cdots \\
 & & & & & & & & \vdots & & & & & & & & & &
\end{bmatrix}$$

It was not shown above, but incomplete stencils from missing nodes in the vertical direction arise from nodes positioned along the second and second to last rows of the mesh. Alternatively, incomplete stencils from missing nodes in the horizontal direction occur along the second and second to last columns of the mesh. Similar to what was done in the 1D, fourth order case the linear system will be solved using a standard matrix vector multiplication and solving for the appropriate temperature. Ghost nodes cannot be accounted for in the A matrix because they lie outside the domain, which means their contribution to the stencil needs to be accounted for after the matrix vector multiplication/addition. Due to the size of the A matrix, a complete representation of the linear system could not be constructed. For completeness, the A matrix will be denoted as A, which is multiplied to vector x and equated to the solution vector:

$$A \begin{bmatrix} \vdots \\ T_n \\ T_{n+1} \\ T_{n+2} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{-12q_n\Delta x^2}{k} \\ \frac{-12q_{n+1}\Delta x^2}{k} \\ \frac{-12q_{n+2}\Delta x^2}{k} \\ \vdots \end{bmatrix}$$

The double dots inbetween the -1 and 16 terms represents a distance of N_x nodes. This matrix has a pentadiagonal structure along with four minor diagonals. **As such, there are nine non-zero elements in a row representing a node with a fully defined stencil. If the node does not have a complete stencil because of one neighboring node (i-2,i+2,j-2,j+2) then the number of non-zero elements is equal to eight. With two such nodes that lie outside the domain, the number of non-zero elements reduces to seven.** The maximum number of incomplete stencils that a single node can cause is two for this case.

It should be noted that the trend shown above only occurs at node locations that are more than two nodes inside from the boundary in either direction. Within each row of the mesh containing nodes with complete stencils, the trend shown above will occur N_x - 4 times (2 boundary nodes and 2 nodes with an incomplete stencil).

Iterative Method

The Jacobi and Gauss-Seidel iterative methods will are defined in the following two sections.

Jacobi

Variables: A = $[a_{i,j}]$ (Matrix of node coefficients of size N x N), **b**: solution vector of length N (combination of boundary conditions and heat source terms), tol: stopping tolerance, x_g: Initial guess for state vector (length N), x_i: State vector defined after the first iteration (length N), M: Max iterations

count = 0 Initialize $\mathbf{x_g}$ vector while count $\leq$ M

1. for i=0; i<=N-1; i++
   (a) $x_{val} = 0$
   (b) for j=0; j<=N-1; j++
      i. if j!=i
      $$x_{val} = x_{val} - a[i,j]x_g[j]$$
   (c) x[i] = $\frac{x_{val}}{a[i,i]} + \frac{b[i]}{a[i,i]}$
2. if $\|\mathbf{x} - \mathbf{x_g}\| < tol$
   (a) return **x**
   (b) break
3. else
   (a) $count = count + 1$
   (b) $\mathbf{x_g} = \mathbf{x}$
   (c) **Back to Step 3.a with updated $\mathbf{x_g}$ vector**

Gauss-Seidel

Instead of updating the state vector at the end of each iteration, the state vector will be updated as soon as components from the next iteration are computed. Thus, two summations are needed, one for all of the indices along the row before the current index (recently updated indices) and another summation for all the terms following the node of interest.

Variables: A = $[a_{i,j}]$ (Matrix of node coefficients of size N x N), **b**: solution vector of length N (combination of boundary conditions and heat source terms), tol: stopping tolerance, x_g: Initial guess for state vector (length N), x_i: State vector defined after the first iteration (length N), M: Max iterations

count = 1
Initialize $\mathbf{x_g}$ vector
while count $\leq$ M

1. for i=0; i<=N-1; i++
   (a) Solve
   $$x_i = \frac{1}{a_{i,i}} \sum_{j=0}^{i-1} - a_{i,j} x_j + \frac{1}{a_{i,i}} \sum_{j=i+1}^{N-1} - a_{i,j} x_{g_j} + \frac{b_i}{a_{i,i}}$$
2. if $\|\mathbf{x} - \mathbf{x_g}\| < tol$
   (a) return $\mathbf{x}$
   (b) break
3. else
   (a) $count = count + 1$
   (b) $\mathbf{x_g} = \mathbf{x}$
   (c) **Back to Step 3.a with updated $\mathbf{x_g}$ vector**

Memory Requirements

Using the notation of N being the total number of nodes in either the 1D or 2D case the representation of the memory requirements can be differentiated by the iterative procedure used. Obviously the 2D cases will have a larger N value, but symbolically, the memory requirements will be the same. At this time, the following memory estimates will be made using a dense matrices, but if I deem a sparse matrix to be more suitable, I reserve the right to change to that.

Jacobi
For this case, the memory requirement can be estimated to be 8 * $N^2 + 4N$ bytes. The A matrix has N^2 parameters, each copy of the solution vector has N parameters $(2 * N)$, the b vector in the linear system has N parameters, and the initial guess has N parameters.
Gauss-Seidel
The Gauss-Seidel requires roughly 8 * $N^2 + 3N$ bytes. This method requires less memory because the solution vector is updated inplace and as new values become available as opposed to waiting until the end of the iteration.

Build Procedures

The first step in the build process is to clone the GitHub repository. From the command line, enter the following code for HTTPS download:

```
git clone https://github.com/uthpc/cse380-2020-student-aleccarruthers.git
```

Enter the *proj01* directory (all of the following commands in this section will be run from this location):

```
cd cse380-2020-student-aleccarruthers/proj01
```

The build system can now be bootstrapped using Autotools, specifically the command *autoreconf*. Within the *proj01* directory, enter:

```
autoreconf --install
```

In order to create the executable, the *proj01* directory needs to be properly configured, such that the libraries used within the source code (MASA and GRVY) can be found. To do this, the directory containing the built libraries needs to be made an environment variable (not technically, but it is easier this way). To create an environment variable specifying the path to the built libraries, enter the following command:

```
export PKGPATH=/work/00161/karl/stampede2/public/
```

With this environment variable set, the *proj01* directory can be configured. Within the *proj01* directory, enter the following command:

```
./configure --with-masa=$PKGPATH/masa-gnu7-0.50/ --with-grvy=$PKGPATH/grvy-gnu7-0.34/
```

Once the configure process is complete, enter the following line within the *proj01* directory to finish the build:

```
make
```

After the build is complete, an executable titled *heatsolver* should be available in the *src* directory.

To run a series of regression tests to check that the executable is running properly, enter the following command from the *proj01* directory:

```
make check
```

This command will run the *tests/reg_test.sh* shell script with the general results of the tests sent to standard out, i.e. the screen, and the test specific results to the *tests/reg_test.sh.log* log file. If everything runs as expected the standard out should show a 1 of 1 pass message implying that all

of the sub tests passed. If a single test fails, the general/broad results sent to standard out will display 0 of 1. For a description of which test/s failed, view the previously mentioned log file.

Input Options

The following inputs are listed within the *src/input.dat* file and used to run the *heatsolver* executable:

1. nodes: An integer value specifying the number of nodes in the domain. The list below provides the three possible cases for how the number of nodes can be set (1 case for 1-dimensional domaind and 2 cases for 2-dimensional domain)

   (a) dimension=1: The value input will be the total number of nodes in the domain

   (b) dimension=2 (case a): If the value input (val) has a perfect square root, i.e. 25 or 36, then this value will be treated as the **TOTAL** number of nodes in the mesh, i.e. a domain size of ***sqrt(val)*** x ***sqrt(val)***.

   (c) dimension=2 (case b): If the value input (val) does not have a perfect square root, i.e. 11 or 27, then this value will be treated as the **number of nodes in each dimension**. Therefore the domain will have the size ***val*** x ***val***.

2. solver: The iterative solver used to solve the heat equation. **Supported solvers: "Jacobi" or "GS"**. With or without quotes is acceptable.

3. dimension: Defines whether to solve the heat equation in a 1-dimensional or 2-dimensional domain. **Supported inputs: 1 or 2**

4. finorder: Solve the heat equation using either second order or fourth order finite differentiation. **Supported differentiation orders: 2 or 4**

5. verification: Run heatsolver in either verification or standard operating mode. **Supported modes: t (verification) or f (standard)**. NOTE: The professor mentioned that for this iteration of the project, there is really no difference between the verification and standard mode, so nothing changes by switching the type.

6. loglevel: The type of output to display. **Supported options: "standard" or "debug".** With or without quotes is acceptable.

7. max_iterations: The maximum number of iterations that can be run/attempted before exiting out of the script. This should be set as integer with no special characters.

8. length: Size of the domain in terms of spatial coordinates/distances. **Default is set to unity**. In the 1D case, the length defines the domain size in the x direction, while in the 2D case, this length defines the domain size in both the x and y directions. *Note: While set as an input parameter for future iterations of the code, only a length of one was tested, hence keep L=1 for now.*

9. frequency: The frequency used to derive the MASA source and exact temperature vectors. In the 1D case, this frequency defines the "A_x" term, while in two dimensions, the input frequency defines both the "A_x" and "B_y" terms. *Note: Similar to the length, this was set as an input parameter for future iterations of the code, but all of the tests were conducted using a value of 5, so keep frequency=5.*

10. k: The thermal conductivity used across the domain in both the 1D and 2D cases.

Verification Procedures

Since running the executable for this project is basically the same thing as running the verification procedure, i.e. computing an L2 normalized error between the numerically generated and exact solutions, the general method for running the executable will be described here. From the *proj01* directory where all of the building occurred, enter the *src* directory.From here, the heatsolver executable can be run using the following command:

heatsolver input.dat

The output of this command includes the input parameters specified in the input.dat file, the L2 normalized loss between the numeric and exact solutions, and timing metrics. One can change the model parameters by entering the input.dat file with any text editor and modifying the input parameters. Any inputs that are changed should adhere to the specifications and descriptions provided in Section 0.6.

For example, to run the executable for a Jacobi, second order finite difference case in one dimension with 40 nodes, the input file should be modified to start off like the following:

nodes = 40
solver = Jacobi
dimension = 1
finorder = 2
...

If it is desired to run the executable in debug mode where both the A matrix and the MASA source vector (including their spatial position) are printed to standard out, modify the following line in the input file to be:

loglevel = debug

To return to the standard output formatting, change the following line in the input file to be:

loglevel = standard

Verification Exercise

A total of seven unique discretization points were used in the verification/convergence rate analyses, including: 10, 20, 40, 60, 80, 100, 120. For the 1-dimensional cases, the previously listed numbers represent the total number of nodes in the grid, while in the 2-dimensional case, those numbers represent the number of nodes in both the x and y directions, i.e. a N_x x N_x grid. For each of the nodes above, the model was run with:

1. solver = Jacobi AND solver = GS
2. dimension = 1 AND dimension = 2
3. finorder = 2 AND finorder = 4 (Jacobi was only run using 2)
4. verification = t
5. loglevel = standard
6. max_iterations = 250000
7. length = 1.0
8. frequency = 5.0
9. k = 1.0

The convergence rate (the absolute value of the slope of the loss vs mesh points figure) of the Gauss-Seidel iterative solver for a 1-dimensional domain was found to be approximately 2.065 for the second order finite difference and 4.142 for the fourth order finite difference (3).

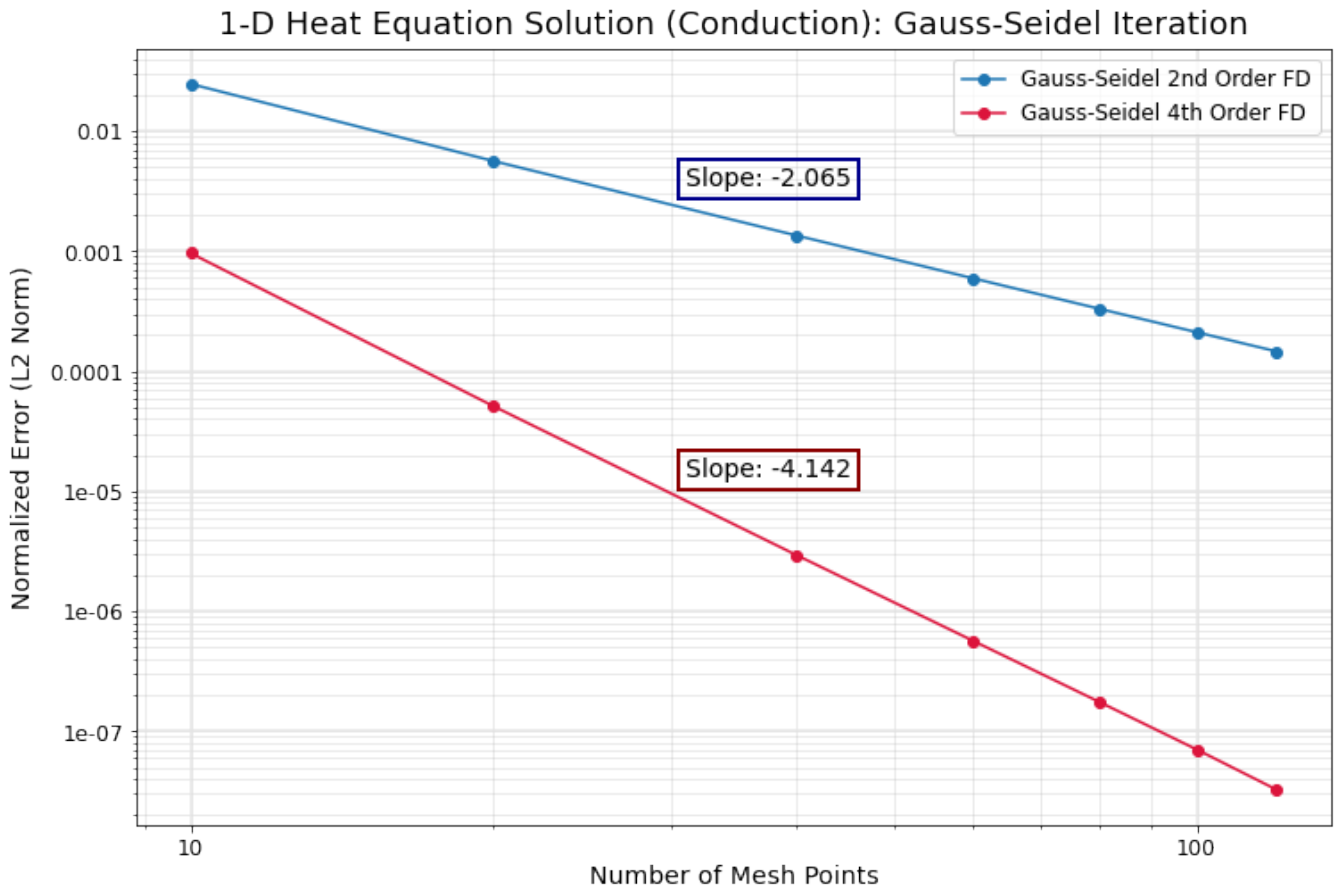

Figure 3: The normalized loss between the numeric and exact solution obtained from a Gauss-Seidel numerical solving method plotted as a function of the number of nodes in a 1-dimensional grid.

The expected convergence rate for the second and fourth order finite difference methods was two and four, respectively. Therefore, the percentage difference between the calculated and expected/theoretical value is approximately 3.25 % and 3.55 % for the second order and fourth order finite differencing methods.

The convergence rates associated with the 2-dimensional Gauss-Seidel cases follow a similar trend with the second order finite differencing method having an approximate absolute convergence rate of 2.049 and the fourth order method having a convergence rate of 4.14 (4).

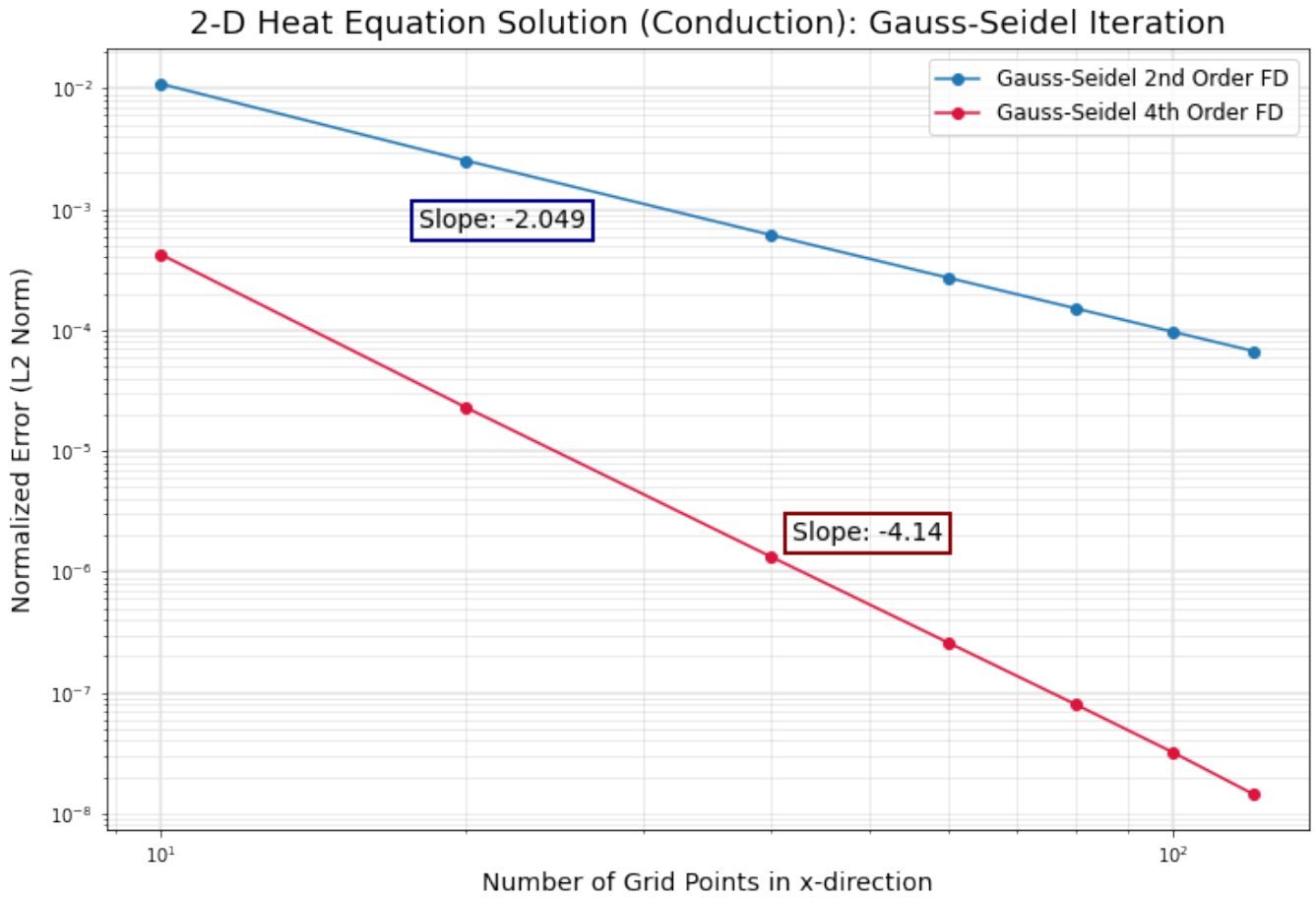

Figure 4: The normalized loss between the numeric and exact solution obtained from a Gauss-Seidel numerical solving method plotted as a function of the number of nodes in a 2-dimensional grid. Here, the x-axis represents the number of nodes along each axis, i.e. the x and y axes create a square mesh.

Similar to what was done for the 1-dimensional Gauss-Seidel case studies, the percentage difference between the numerically derived and theoretical convergence rate was found to be 2.45 % and 3.5 % for the second and fourth order finite differencing methods, respectively.

Unlike the Gauss-Seidel iterative solving technique, the Jacobi fourth order finite differencing case, at least for this specific problem, is numerically unstable for both the 1-dimensional and 2-dimensional cases. As a result, only the second order finite differencing method was considered here with the results from both the 1-D and 2-D analyses plotted on the same figure (5).

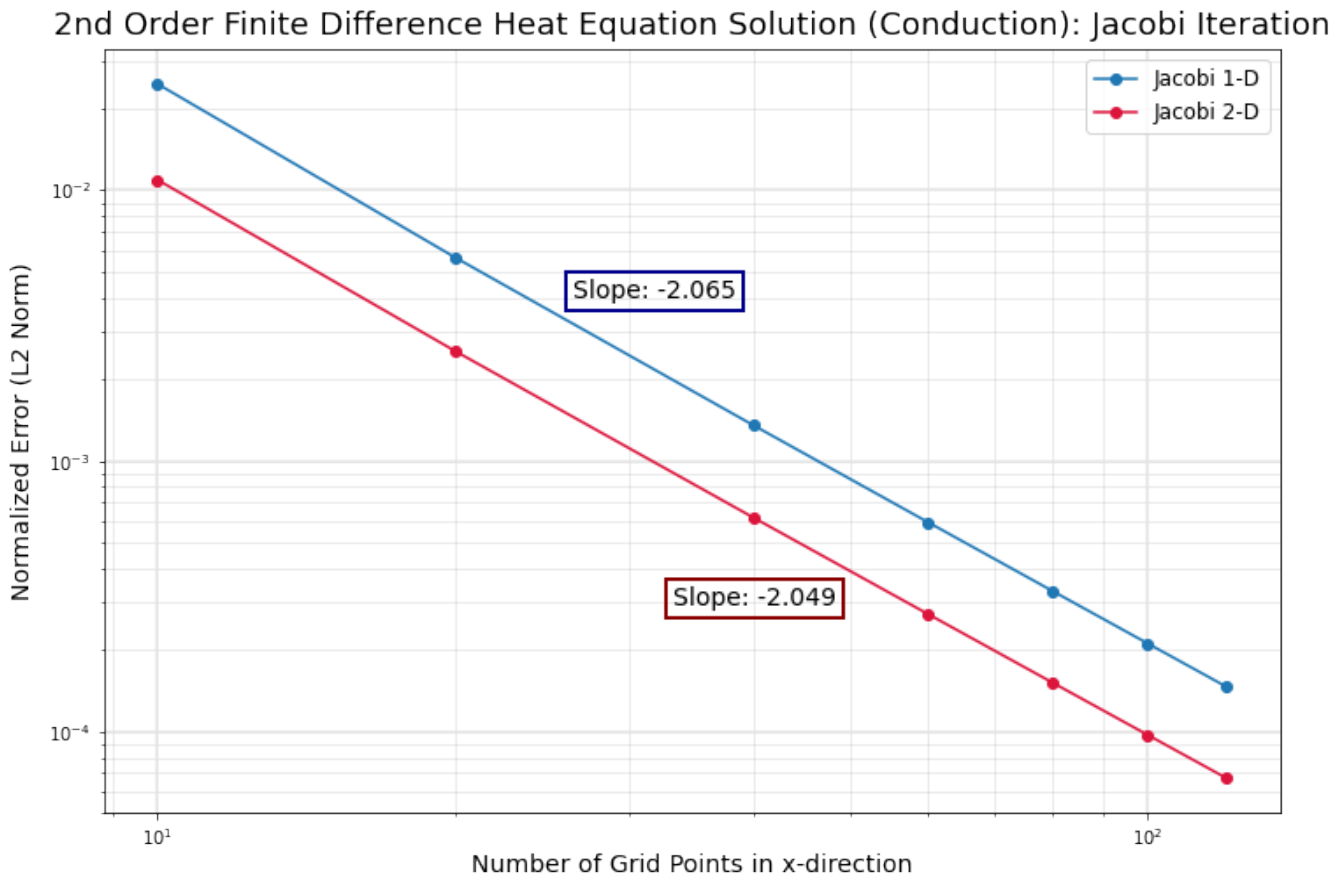

Figure 5: The normalized loss as a function of grid size (l2 norm) between the numeric and exact solutions of the 1D/2D heat conduction problem using a second order finite differencing Jacobi iterative solver.

The percent differences between the numerically derived and expected convergence rate for the 1−dimensional and 2−dimensional cases were found to be 3.25 % and 2.45 %, respectively.

Runtime Performance

The library GRVY was used for all of the runtime performance metrics, including timings for the A matrix initialization, MASA source vector initialization, MASA solution vector initialization, numerical solver, and the normalized error calculation. The following figures show the formatting/-timing results for two of the samples used within the verification exercise with their respective model inputs shown above the timing results (Figures 6 and 7).

```
Nodes: 40
Dimension: 1
Finite Difference Order: 4
Maximum Iterations: 250000
Numerical Solver: GS
Output Mode: standard
Domain Length: 1
MASA Frequency: 5
Thermal Conductivity: 1
Standard Operating Mode
Test Verify: 1

L2 Norm: 2.93932998343e-06

-----------------------------------------------------------------------------------------------
GRVY Timing - Performance Timings:                          |      Mean      Variance       Count
--> Numerical Solver         : 1.32339e-02 secs ( 80.3750 %) | [1.32339e-02  0.00000e+00          1]
--> Initialize MASA Sources  : 1.92404e-03 secs ( 11.6855 %) | [1.92404e-03  0.00000e+00          1]
--> Initialize MASA Solution : 1.24407e-03 secs (  7.5557 %) | [1.24407e-03  0.00000e+00          1]
--> Initialize A Matrix      : 3.29018e-05 secs (  0.1998 %) | [3.29018e-05  0.00000e+00          1]
--> L2 Norm                  : 1.62125e-05 secs (  0.0985 %) | [1.62125e-05  0.00000e+00          1]
--> GRVY_Unassigned          : 1.40667e-05 secs (  0.0854 %)

        Total Measured Time = 1.64652e-02 secs (100.0000 %)
-----------------------------------------------------------------------------------------------
```

Figure 6: The performance metrics for the 1-dimensional, fourth order Gauss-Seidel implementation with 40 nodes.

```
Nodes: 80
Dimension: 1
Finite Difference Order: 4
Maximum Iterations: 250000
Numerical Solver: GS
Output Mode: standard
Domain Length: 1
MASA Frequency: 5
Thermal Conductivity: 1
Standard Operating Mode
Test Verify: 1
L2 Norm: 1.75536315152e-07

-----------------------------------------------------------------------------------------------
GRVY Timing - Performance Timings:                          |      Mean      Variance       Count
--> Numerical Solver         : 2.17646e-01 secs ( 98.5407 %) | [2.17646e-01  0.00000e+00          1]
--> Initialize MASA Sources  : 1.86300e-03 secs (  0.8435 %) | [1.86300e-03  0.00000e+00          1]
--> Initialize MASA Solution : 1.26600e-03 secs (  0.5732 %) | [1.26600e-03  0.00000e+00          1]
--> Initialize A Matrix      : 5.41210e-05 secs (  0.0245 %) | [5.41210e-05  0.00000e+00          1]
--> L2 Norm                  : 5.00679e-06 secs (  0.0023 %) | [5.00679e-06  0.00000e+00          1]
--> GRVY_Unassigned          : 3.50475e-05 secs (  0.0159 %)

        Total Measured Time = 2.20869e-01 secs (100.0000 %)
-----------------------------------------------------------------------------------------------
```

Figure 7: The performance metrics for the 1-dimensional, fourth order Gauss-Seidel implementation with 80 nodes.

The following two tables display the performance metrics for the Gauss-Seidel, second order finite difference implementation using a 1-dimensional (Table 1) and 2-dimensional domain (Table 2). For brevity and formatting purposes, the column headers were abbreviated from what is shown in Figure 7. The following mappings were used: "Numerical Solver" to "Num. Solve", "Initialize MASA Sources" to "MASA src", "Initialize MASA Solution" to "MASA sol", "Initialize A Matrix" to "A Mat", and "L2 Norm" to "L2".

Table 1: Performance metrics for all of the Gauss-Seidel 1-dimensional and second order finite difference test cases used in the verification analysis.

Nodes_x	A Mat (s)	MASA src (s)	MASA sol (s)	Num. solve (s)	L2 (s)
10	0.000008	0.00176	0.00120	0.000038	9.540000e-07
20	0.000008	0.00176	0.00126	0.000243	9.540000e-07
40	0.000016	0.00183	0.00128	0.004080	1.190000e-06
60	0.000021	0.00176	0.00124	0.020400	9.540000e-07
80	0.000039	0.00184	0.00125	0.066800	9.540000e-07
100	0.000040	0.00174	0.00127	0.174000	9.540000e-07
120	0.000063	0.00194	0.00134	0.401000	1.190000e-06

Table 2: Performance metrics for all of the Gauss-Seidel 2-dimensional and second order finite difference test cases used in the verification analysis.

Nodes_x	A Mat (s)	MASA src (s)	MASA sol (s)	Num. Solve (s)	L2 (s)
10	0.000042	0.001830	0.00127	0.00178	9.540000e-07
20	0.000725	0.001770	0.00120	0.12500	2.860000e-06
40	0.010800	0.001830	0.00126	8.42000	1.000000e-05
60	0.051100	0.002090	0.00137	106.00000	2.290000e-05
80	0.171000	0.002420	0.00165	653.00000	3.000000e-05
100	0.408000	0.004670	0.00175	2260.00000	5.200000e-05
120	0.882624	0.004816	0.00208	6440.00000	7.890000e-05

One of the trends that can be noticed is that while the number of nodes is relatively low, the functions used for the initialization of the MASA source and solution vectors make up a non-negligible percentage of the overall time. However, as the number of nodes increases, the numerical solver function begins to dominate the computation time, making the time spent on all other functions nearly irrelevant. The example timings shown in Figures 6 and 7 were for just the 1-dimensional case, but it can be seen that in Tables 1 and 2 that computation time of the numerical solver is order of magnitudes larger in the 2-dimensional case than the 1-dimensional case. This is one of the downsides and limitations of using a dense A matrix. While relatively straight forward to program, the computation time and complexity does not scale well with a large number of nodes.

Conclusion

From the convergence rates shown in Section 0.6, the executable *heatsolver* seems to provide reasonably accurate results when solving the steady state heat conduction equation using accepted parameters. The most significant limitation/drawback of this executable is the lack of optimal performance as the number of nodes increases. Since a dense matrix was used to solve the system of equations, the number of parameters in the model blows up as number of nodes increases. Sparse matrix representations of the currently used A matrix are being investigated and implementing them while retaining model accuracy is one of the goals for future iterations of this executable.