Modeling Document

- **Governing Equations**

$$-k\nabla^2 T(x,y) = q(x,y) \text{ in } \Omega \tag{1}$$
$$T(x,y) = T_{masa}(x,y) \text{ on } \partial\Omega$$

where,

Ω is a 2D bounded domain
$\partial\Omega$ is the boundary of the domain
T is the material's temperature field
q is the heat source term
k is the thermal conductivity

We have the Dirichlet boundary conditions.

- **Assumptions**
  - The thermal conductivity is assumed to be constant.
  - We assume a square domain $\Omega = \{\ (x,y) : x \in [0,L], y \in [0,L]\ \}$ for the 2D case. For 1D, it will obviously be a line $\Omega = \{\ (x,y) : x \in [0,L]\ \}$
  - Dirichlet boundary condition is assumed at the boundaries
  - For the fourth order scheme, we assume that the values at the points adjacent to the boundary points are known from the `MASA` solution. This is to reduce the cumbersome effort required to come up with different schemes at the boundary.
  - For the 2D case we assume symmetrical discretization i.e. the number of grid points in both directions are the same and $\Delta x = \Delta y = h$.

- **Nomenclature for discretization** Our numerical methods are all node based (as will be reiterated later).
  - 1D We have $(N+1)$ points $\{x_0, x_1, x_2, \ldots x_N\}$ in the x-direction with $x_i = i\Delta x$, where $\Delta x = L/N$
  - 2D We have $(N+1)$ points $\{x_0, x_1, x_2, \ldots x_N\}$ in the x-direction with $x_i = i\Delta x$, where $\Delta x = L/N = h$ and $(N+1)$ points $\{y_0, y_1, y_2, \ldots y_N\}$ in the y-direction with $y_j = j\Delta y$, where $\Delta y = L/N = h$. Hence, we have a $(N+1)\times(N+1)$ grid.
  - i is always associated with the indexing in x-direction and j is always associated with the indexing in y-direction.

- $T(x_i, y_j)$ is given the shorthand notation $T(i, j)$ and $q(x_i, y_j)$ is given the shorthand notation $q(i, j)$
- The composite index for 2D grid is given by $k = i + (N + 1)j$.

- **Numerical Method**

  Our numerical methods are all node based (as will be reiterated later).

  - **2^{nd} order finite difference approximation**

    Definition

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T(x + \Delta x) - 2T(x) + T(x - \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

    Discretized heat equation

    1. 1D

$$\begin{cases} -k\left(\frac{T(i+1)-2T(i)+T(i-1)}{\Delta x^2}\right) + \mathcal{O}(\Delta x^2) = q(i), \ \ i \in \{1, 2, 3, \ldots, N-1\} \\ T(0) = T_{masa}(0) \ \text{ and } \ T(N) = T_{masa}(L) \end{cases} \tag{2}$$

    2. 2D

$$\begin{cases} -k\left(\frac{T(i+1,j)-2T(i,j)+T(i-1,j)}{h^2}\right) - k\left(\frac{T(i,j+1)-2T(i,j)+T(i,j-1)}{h^2}\right) \\ +\mathcal{O}(h^2) = q(i, j), \ \ i \in \{1, 2, 3, \ldots, N-1\} \ \ j \in \{1, 2, 3, \ldots, N-1\} \\ \\ T(i, j) = T_{masa}(x_i, y_j) \ \text{ for } \ i \in \{0, N\}, \ j \in \{0, 1, 2, \ldots, N\} \\ T(i, j) = T_{masa}(x_i, y_j) \ \text{ for } \ i \in \{0, 1, 2, \ldots, N\}, \ j \in \{0, N\} \end{cases} \tag{3}$$

  - **4^{th} order finite difference approximation**

    Definition

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{-T(x - 2\Delta x) + 16T(x - \Delta x) - 30T(x) + 16T(x + \Delta x) - T(x + 2\Delta x)}{12\Delta x^2} + \mathcal{O}(\Delta x^4)$$

    Discretized heat equation

    We have Dirichlet boundary conditions at both the boundary points and adjacent to boundary points.

    1. 1D

$$\begin{cases} -k\left(\frac{-T(i-2)+16T(i-1)-30T(i)+16T(i+1)-T(i+2)}{12\Delta x^2}\right) + \mathcal{O}(\Delta x^4) = q(i), \ \ i \in \{2, 3, \ldots, N-2\} \\ T(i) = T_{masa}(x_i), \ \ i \in \{0, 1, N-1, N\} \end{cases} \tag{4}$$

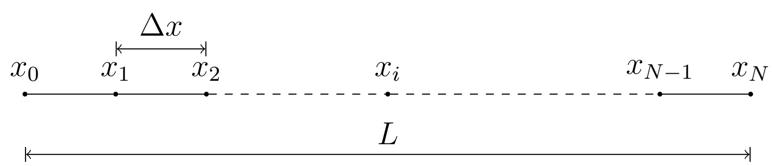
2. 2D

$$\begin{cases} -k\left(\frac{-T(i-2,j)+16T(i-1,j)-30T(i,j)+16T(i+1,j)-T(i+2,j)}{12h^2}\right) \\ -k\left(\frac{-T(i,j-2)+16T(i,j-1)-30T(i,j)+16T(i,j+1)-T(i,j+2)}{12h^2}\right) + \mathcal{O}(h^4) = q(i,j), \\ \;i \;\in \{2,\ldots,N-2\}, j \;\; \in \{2,\ldots,N-2\} \\ \\ T(i,j) = T_{masa}(x_i,y_j) \;\text{ for }\; i \;\in \{0,1,N-1,N\},\; j \in \{0,1,2,\ldots,N\} \\ T(i,j) = T_{masa}(x_i,y_j) \;\text{ for }\; i \;\in \{0,1,2,\ldots,N\},\; j \in \{0,1,N-1,N\} \end{cases} \tag{5}$$

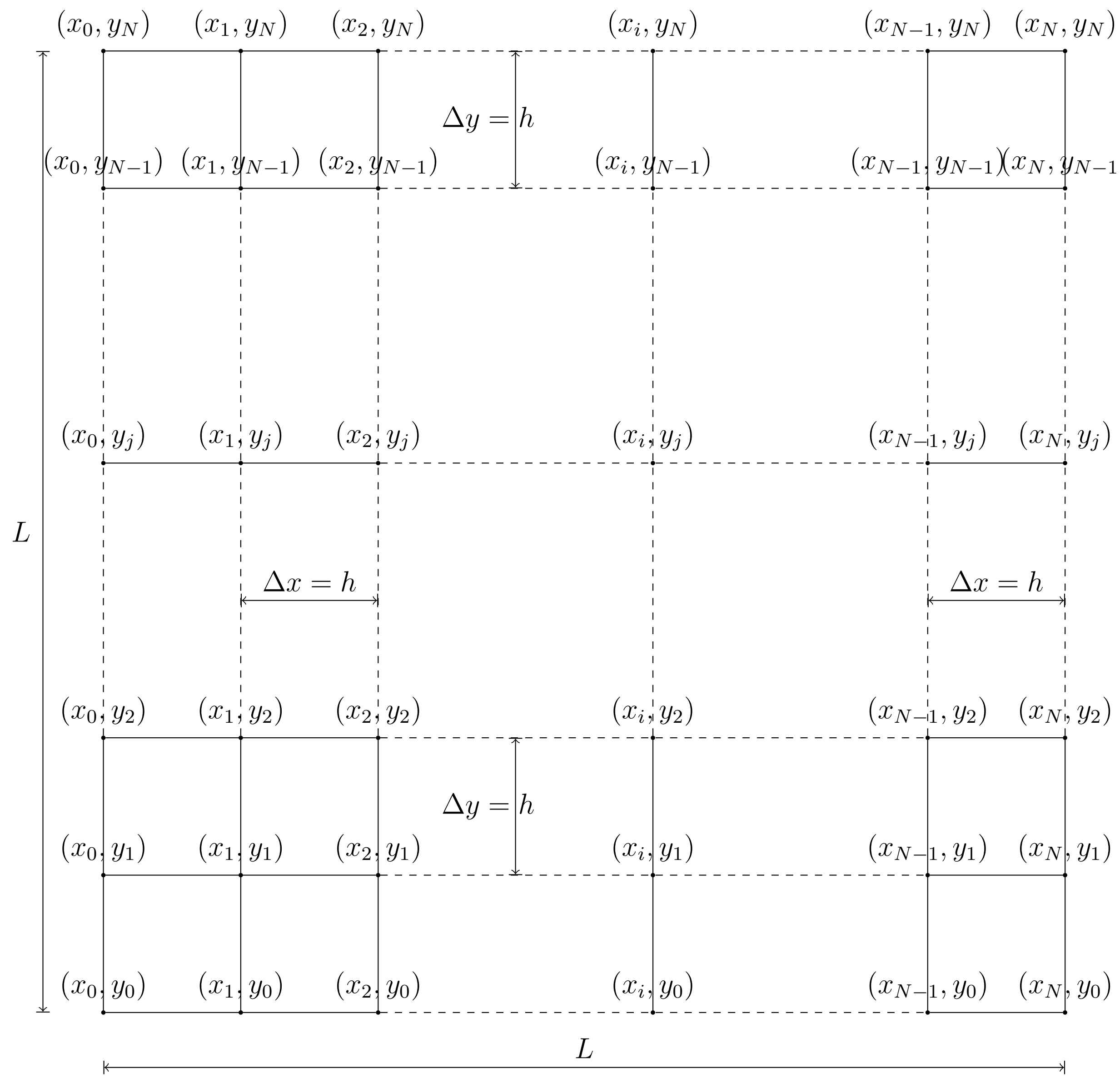
- **Mesh diagrams**

  Our schemes are node based.

  1. 1D



  2. 2D



- **Linear system of Equations**
  - 2^{nd} **order finite difference approximation**
    1. 1D

       We first define the following vectors

$$\mathbf{q} = \left[T_{masa}(0), \frac{\Delta x^2}{k}q(1), \ldots, \frac{\Delta x^2}{k}q(N-1), T_{masa}(L)\right]^T$$
$$\mathbf{T} = [T(0), \ldots, T(N)]^T$$

We now define a $(N+1)\times(N+1)$ tridiagonal matrix $\mathbf{A}$ such that,

$$\mathbf{A} = \begin{bmatrix} 1 & & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & & 1 \end{bmatrix}$$

Now (2) can be written as,

$$\mathbf{AT} = \mathbf{q}$$

The first and last rows are different because they account for boundary conditions. The sparsity pattern of A can be given by,

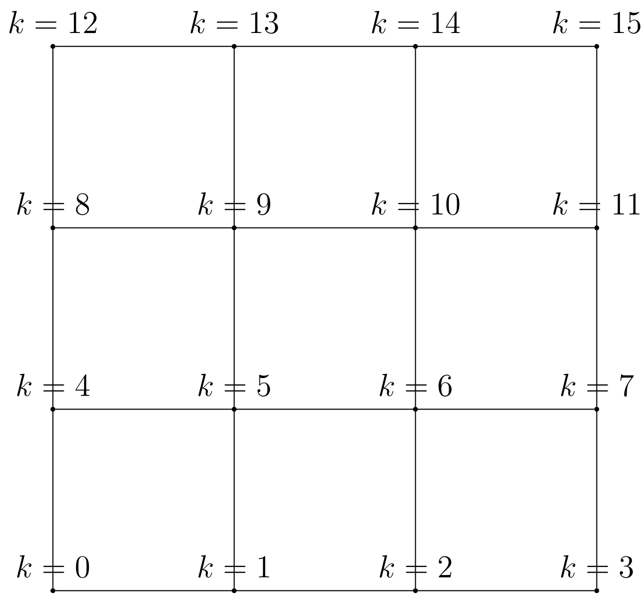
$$\mathbf{A} = \begin{bmatrix} \times & & & & & & & \\ \times & \times & \times & & & & & \\ & \times & \times & \times & & & & \\ & & \times & \times & \times & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \times & \times & \times & \\ & & & & & \times & \times & \times \\ & & & & & & & \times \end{bmatrix}$$

`Number of non-zero elements on an interior row of the matrix = 3`

2. 2D
   The elements will be numbered in the following fashion

$$k = i + (N+1)j,\ i \in [0,\ldots,N],\ j \in [0,\ldots,N]$$

Here is an illustration for a 4×4 grid i.e. $N = 3$,



So we can imagine how the matrix is going to look: the terms of second derivative in x-direction will be adjacent to each other, but they terms of the second derivative in y-direction will be offset by N points. This will become clear in the visual representation below. We first define the following $(N+1)^2 \times 1$ vectors

$$\mathbf{q}(k) = \begin{cases} \frac{h^2}{k} q(i,j) \ \text{ if } \ i \in \{1,2,\ldots,N-1\},\ j \in \{1,2,\ldots,J-1\} \\ T_{masa}(x_i, y_j) \ \text{ otherwise i.e. at boundaries} \end{cases}$$

$$\mathbf{T} = [T(0), T(1), \ldots, T((N+1)^2)]^T$$

We now define $(N+1)^2 \times (N+1)^2$ matrices $\mathbf{A_x}$ (interior x-direction derivatives), $\mathbf{A_y}$ (interior y-direction derivatives) and $\mathbf{A_b}$ (dirichlet nodes) such that,

$$\mathbf{A_x} = \begin{cases} \mathbf{A_x}(i,i) = \begin{cases} 2 \ \text{ if } i \text{ corresponds to an interior point} \\ 0 \ \text{ otherwise} \end{cases} \\ \mathbf{A_x}(i,i-1) = \begin{cases} -1 \ \text{ if } i \text{ corresponds to an interior point} \\ 0 \ \text{ otherwise} \end{cases} \\ \mathbf{A_x}(i,i+1) = \begin{cases} -1 \ \text{ if } i \text{ corresponds to an interior point} \\ 0 \ \text{ otherwise} \end{cases} \end{cases}$$

$$
\mathbf{A_y} = \begin{cases}
\mathbf{A_y}(j,j) = \begin{cases} 2 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\
\mathbf{A_y}(j,j-N) = \begin{cases} -1 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\
\mathbf{A_y}(j,j+N) = \begin{cases} -1 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases}
\end{cases}
$$

$$
\mathbf{A_b} = \begin{cases}
\mathbf{A_b}(i,i) = \begin{cases} 1 & \text{if } i \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases}
\end{cases}
$$

Now, the net matrix is $\mathbf{A} = \mathbf{A_x} + \mathbf{A_y} + \mathbf{A_b}$. The sparsity pattern of $\mathbf{A}$ is given by, (illustration for a 5×5 grid)

$$
\mathbf{A} = \begin{bmatrix}
\times&&&& & &&&& & &&&& & &&&& & &&&& \\
&\times&&& & &&&& & &&&& & &&&& & &&&& \\
&&\times&& & &&&& & &&&& & &&&& & &&&& \\
&&&\times& & &&&& & &&&& & &&&& & &&&& \\
&&&&\times & &&&& & &&&& & &&&& & &&&& \\
&&&& & \times&&&& & &&&& & &&&& & &&&& \\
\times&&&& & \times&\times&\times&& & &&\times&& & &&&& & &&&& \\
&\times&&& & &\times&\times&\times& & &&&\times& & &&&& & &&&& \\
&&\times&& & &&\times&\times&\times & &&&&\times & &&&& & &&&& \\
&&&& & &&&&\times & &&&& & &&&& & &&&& \\
&&&& & &&&& & \times&&&& & &&&& & &&&& \\
&&&& & \times&&&& & \times&\times&\times&& & &&\times&& & &&&& \\
&&&& & &\times&&& & &\times&\times&\times& & &&&\times& & &&&& \\
&&&& & &&\times&& & &&\times&\times&\times & &&&&\times & &&&& \\
&&&& & &&&& & &&&&\times & &&&& & &&&& \\
&&&& & &&&& & &&&& & \times&&&& & &&&& \\
&&&& & &&&& & \times&&&& & \times&\times&\times&& & &&\times&& \\
&&&& & &&&& & &\times&&& & &\times&\times&\times& & &&&\times& \\
&&&& & &&&& & &&\times&& & &&\times&\times&\times & &&&&\times \\
&&&& & &&&& & &&&& & &&&&\times & &&&& \\
&&&& & &&&& & &&&& & &&&& & \times&&&& \\
&&&& & &&&& & &&&& & &&&& & &\times&&& \\
&&&& & &&&& & &&&& & &&&& & &&\times&& \\
&&&& & &&&& & &&&& & &&&& & &&&\times& \\
&&&& & &&&& & &&&& & &&&& & &&&&\times
\end{bmatrix}
$$

`Number of non-zero elements on an interior row of the matrix =` 5.

The repeating interior block of $\mathbf{A}$ is given by,

$$
\mathbf{A} = \begin{bmatrix}
 & & 1 & & & & & & \\
-1 & \dots & -1 & 4 & -1 & \dots & -1 & & \\
 & -1 & \dots & -1 & 4 & -1 & \dots & -1 & \\
 & & -1 & \dots & -1 & 4 & -1 & \dots & -1 \\
 & & & & & & 1 & &
\end{bmatrix}
$$

Now (3) can be written as,

$$
\mathbf{AT} = \mathbf{q}
$$

– $4^{\mathbf{th}}$ **order finite difference approximation**

1. 1D

   We first define the following vectors

$$\mathbf{q} = \left[T_{masa}(0), T_{masa}(x_1), \frac{12\Delta x^2}{k} q(3) \ldots, \frac{12\Delta x^2}{k} q(N-2), T_{masa}(x_{N-1}), T_{masa}(L)\right]^T$$
$$\mathbf{T} = [T(0), \ldots, T(N)]^T$$

   We now define a $(N+1)\times(N+1)$ pentadiagonal matrix $\mathbf{A}$ such that (blank elements are 0),

$$\mathbf{A} = \begin{bmatrix} 1 & & & & & & & & & \\ & 1 & & & & & & & & \\ 1 & -16 & 30 & -16 & 1 & & & & & \\ & 1 & -16 & 30 & -16 & 1 & & & & \\ & & 1 & -16 & 30 & -16 & 1 & & & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & & 1 & -16 & 30 & -16 & 1 & \\ & & & & & 1 & -16 & 30 & -16 & 1 \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{bmatrix}$$

   Now (4) can be written as,

$$\mathbf{AT} = \mathbf{q}$$

   Note that the first and last two rows in $\mathbf{A}$ looks different because we accounted for the Dirichlet conditions. The sparsity pattern of the matrix is given by,

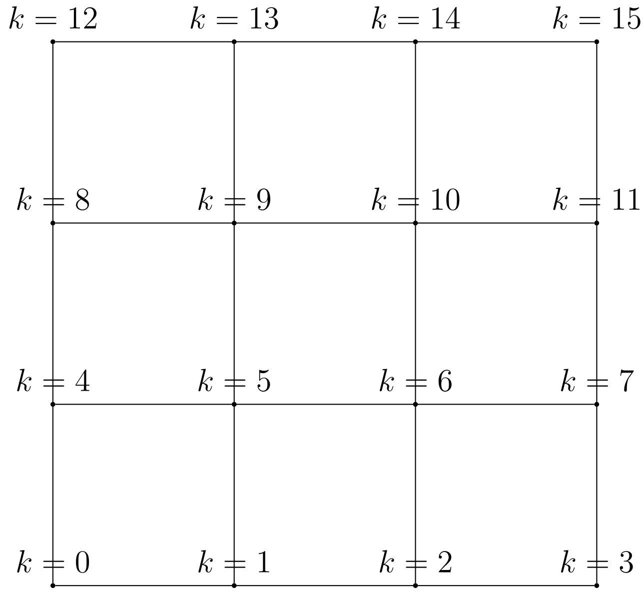
$$\mathbf{A} = \begin{bmatrix} \times & & & & & & & & & \\ & \times & & & & & & & & \\ \times & \times & \times & \times & \times & & & & & \\ & \times & \times & \times & \times & \times & & & & \\ & & \times & \times & \times & \times & \times & & & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & & \times & \times & \times & \times & \times & \\ & & & & & \times & \times & \times & \times & \times \\ & & & & & & & & \times & \\ & & & & & & & & & \times \end{bmatrix}$$

   `Number of non-zero elements on an interior row of the matrix =` 5.

2. 2D The elements will be numbered in the following fashion

$$k = i + (N+1)j,\ i \in [0,\ldots,N],\ j \in [0,\ldots,N]$$

Here is an illustration for a 4×4 grid i.e. $N = 3$,



So we can imagine how the matrix is going to look: the terms of second derivative in x-direction will be adjacent to each other, but they terms of the second derivative in y-direction will be offset by N points. This will become clear in the visual representation below. We first define the following $(N+1)^2 \times 1$ vectors

$$\mathbf{q}(k) = \begin{cases} \frac{12h^2}{k} q(i,j) & \text{if } i \in \{1, 2, \ldots, N-1\},\ j \in \{1, 2, \ldots, N-1\} \\ T_{masa}(x_i, y_j) & \text{otherwise} \end{cases}$$

$$\mathbf{T} = [T(0), T(1), \ldots, T((N+1)^2)]^T$$

We now define $(N+1)^2 \times (N+1)^2$ matrices $\mathbf{A_x}$ (interior x-direction derivatives), $\mathbf{A_y}$ (interior y-direction derivatives) and $\mathbf{A_b}$ (boundary and close to boundary elements) such that,

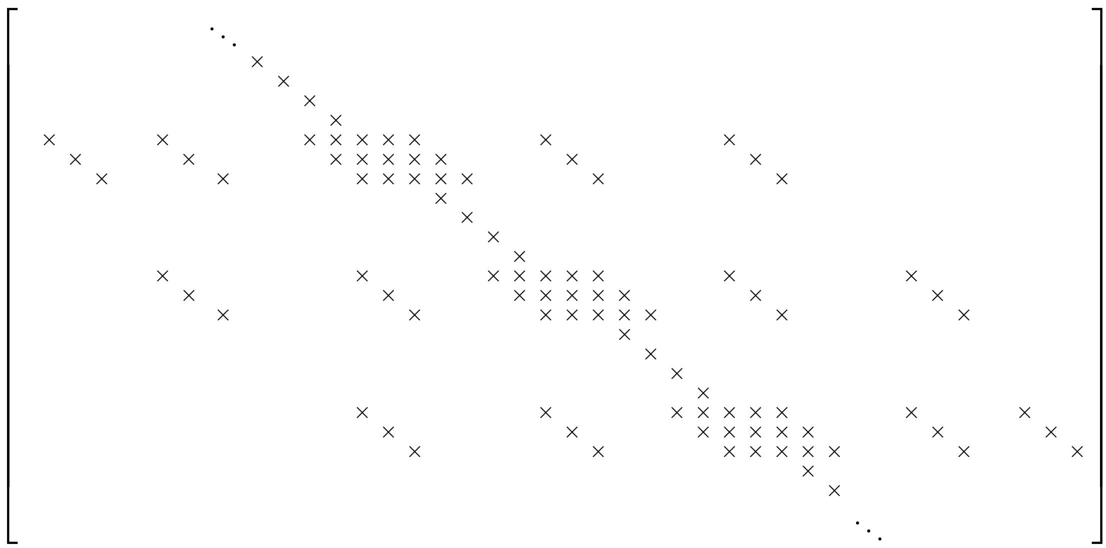
$$\mathbf{A_x} = \begin{cases} \mathbf{A_x}(i,i) = \begin{cases} 30 & \text{if } i \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_x}(i,i-1) = \begin{cases} -16 & \text{if } i \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_x}(i,i+1) = \begin{cases} -16 & \text{if } i \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_x}(i,i+2) = \begin{cases} 1 & \text{if } i \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_x}(i,i-2) = \begin{cases} 1 & \text{if } i \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

$$\mathbf{A_y} = \begin{cases} \mathbf{A_y}(j,j) = \begin{cases} 30 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_y}(j,j-N) = \begin{cases} -16 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_y}(j,j+N) = \begin{cases} -16 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_y}(j,j+2N) = \begin{cases} 1 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{A_y}(j,j-2N) = \begin{cases} 1 & \text{if } j \text{ corresponds to an interior point} \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

$$\mathbf{A_b} = \begin{cases} \mathbf{A_b}(j,j) = \begin{cases} 1 & \text{if } j \text{ is a boundary point or an adjacent to boundary point} \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

Now, the net matrix is $\mathbf{A} = \mathbf{A_x} + \mathbf{A_y} + \mathbf{A_b}$. The sparsity pattern of $\mathbf{A}$ is given by, (illustration for a 7×7 grid, which means the interior matrix is 3×3)

$\mathbf{A} =$



`Number of non-zero elements on an interior row of the matrix = 9.`

The repeating interior block of $\mathbf{A}$ is given by,

$$
\mathbf{A} = \begin{bmatrix}
 & & & & 1 & & & & & & & & & & \\
 & & & & & 1 & & & & & & & & & \\
1 & \dots & -16 & \dots & 1 & -16 & 60 & -16 & 1 & \dots & -16 & \dots & 1 & & \\
 & 1 & \dots & -16 & \dots & 1 & -16 & 60 & -16 & 1 & \dots & -16 & \dots & 1 & \\
 & & 1 & \dots & -16 & \dots & 1 & -16 & 60 & -16 & 1 & \dots & -16 & \dots & 1 \\
 & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & 1 & & & &
\end{bmatrix}
$$

Now (5) can be written as,

$$\mathbf{AT} = \mathbf{q}$$

- **Iterative solvers**

  Here is the pseudo-code for dense matrix solvers (the actual implementation might be for sparse matrices or not, this is not decided yet).

  1. **Jacobi**

```
subroutine jacobi (A,q, TOL, max_iter)
!!! Find T = inv(A)*q using Jacobi iteration

K = size(q)
iters = 0 !!! Number of iterations
error = 0 !!! Compare to tolerance
T(1:K) = 0 !!! Initialize entire T array to zero

do while (iters <= max_iter)

    T_old(:) = T(:)

    do i = 1,...,K
```

```
        !!! We use only the old values and none of the ↩
            updated values for the update.

        T(i) =  1/A(i,i) * (q(i)-Σ_{j≠i}^{K} A(i,j)T_old(j))
    end do

    error = ||T_old−T|| / ||T||

    iters = iters + 1

    if (error <= TOL)
        break
    end if


end do

return T
end subroutine
```

2. **Gauss-Seidel**

```
subroutine gauss_seidel (A,q, TOL, max_iter)
!!! Find T = inv(A)*q using Gauss Seidel iteration

K = size(q)
iters = 0 !!! Number of iterations
error = 0 !!! Compare to tolerance
T(1:K) = 0 !!! Initialize entire T array to zero

do while (iters <= max_iter)

    T_old(:) = T(:)

    do i = 1,...,K
        !!! T_old is not used at all, we use older values↩
             for >i and new values for <i.

        T(i) =  1/A(i,i) * (q(i)-Σ_{j≠i}^{K} A(i,j)T(j))
    end do

    error = ||T_old−T|| / ||T||
    iters = iters + 1
    if (error <= TOL)
        break
    end if



end do

return T
end subroutine
```

- **Estimate of memory requirements**

  We assume that our matrices are stored as normal, dense matrices (assuming that our iterative solvers have generic, dense implementations).

  – Second order approximation

    * 1D

Variable	Dimension	Memory
T & T_old	2(N+1)	$16\times$(N+1)
q	N+1	$8\times$(N+1)
A	(N+1)$\times$(N+1)	$8\times$(N+1)^2
N	1	4
L	1	8
Δx	1	8
k	1	8
	Total	8(N+1)^2+24(N+1)+28

    * 2D

Variable	Dimension	Memory
T & T_old	2(N+1)(N+1)	$16\times$(N+1)^2
q	(N+1)(N+1)	$8\times$(N+1)^2
A	(N+1)(N+1)$\times$(N+1)(N+1)	$8\times$(N+1)^4
N	1	4
L	1	8
Δx	1	8
Δy	1	8
k	1	8
	Total	8(N+1)^4+24(N+1)^2+36

  – Fourth order approximation

    * 1D

Variable	Dimension	Memory
T & T_old	2(N+1)	16×(N+1)
q	N+1	8×(N+1)
A	(N+1)×(N+1)	8×(N+1)^2
N	1	4
L	1	8
Δx	1	8
k	1	8
	Total	8(N+1)^2+24(N+1)+28

* 2D

Variable	Dimension	Memory
T & T_old	2(N+1)(N+1)	16×(N+1)^2
q	(N+1)(N+1)	8×(N+1)^2
A	(N+1)(N+1)×(N+1)(N+1)	8×(N+1)^4
N	1	4
L	1	8
Δx	1	8
Δy	1	8
k	1	8
	Total	8(N+1)^4+24(N+1)^2+36

- **Build Procedures and description of files**

  - Various `C++` files

    `main.cpp` - The driver routine, also has defensive checks. It parses the input file, feeds it to various functions and depending on mode, prints out to `std::out`. It also stores the result to `output.log`.

    `q_assemble.cpp` - Assembles `q` vector based on specifications in the input file

    `T_exact_assemble.cpp` - Assembles the `MASA` solution of the temperature field based on specifications in the input file

    `matrix_assemble.cpp` - Assembles `A` matrix based on specifications in the input file

    `print.cpp` - Contains various functions to print stuff, this is just so that `main.cpp` remains relatively uncluttered.

    `global_variables.h` - Defined objects of various `GRVY` classes as `extern` variables. It is imported in all other cpp files so that the same object is used in all files.

    `solvers.cpp` - Contains the solvers and function to choose solvers based on input file specifications

- Running the primary code

  You will find `Makefile.am` in all subdirectories and `connfigure.ac` in `proj01`.

  These are the steps to run the primary code (assuming you are in the `proj01` subdirectory). All of the code is in the `proj01/src` subdir. The shell script should work but if doesn't - change the permissions using `chmod 777 build_autotools.sh`.

```
$ ./build_autotools.sh ### Runs autoreconf and ./configure with ↩
    appropriate options to link to MASA and GRVY
$ make ### Creates an executable in proj01/src subdir named ↩
    heat_solve
$ cd src/
$ ./heat_solve
```

- Running regression tests

  You can find these tests in the `proj01/tests` subdir in `test.sh`.

```
$ ./build_autotools.sh ### Runs autoreconf and ./configure with ↩
    appropriate options to link to MASA and GRVY
$ export  PATH=/work/00161/karl/stampede2/public/bats/bin/:$PATH ### ↩
    Add bats to path to run regression tests
$ make check
```

- Running the various post-processing scripts

  The shell script should work but if they don't - change the permissions using `chmod 777 shell_script_name`.

  `mesh_size_change.sh` is a script to run all configurations you want in a for loop. All you have to do is change the array variables inside. It uses `plot_convergence.script` to create convergence plots. It stores outputs of time and error in `output_*.dat`.

  `curve_1D_plotter.script` and `surface_2D_plotter.script` create 1D curves (`plot.png`) and 2D surfaces (`surface.png`) for the temperature fields that get stored in `output.log` or `output_100x100.log`.

- Various output files

  `reference_sol_*.log` contain reference solutions created using `mesh_size_change.sh` for regression testing using `TOL = 1e-17` and `MAX_ITERS= 1000000`.

  `convergence_*.png` contains images of convergence analysis, created by `mesh_size_change.sh`

  `output.log` contains the position vs Temperature data, created by `main.cpp`. It gets updated every time the executable is run.

  `output_100x100.log` is a snapshot solution of a 100x100 grid with gauss-seidel 4th order to plot surface plots.

  `output_*.dat` is created by `mesh_size_change.sh` annd contains data on n, L2 error and time taken.

  `plot.png` has the 1D temperature vs x plot

  `surface.png` has the surface plot for temperature over a 2D domain.

- **Input Options**

  Here is an example `input.dat` file.

```
verification = 1              #Comparison with MASA solution? 1 for yes; no otherwise; Always keep as 1. Otherwise ↩
    a couple of regression tests might fail.
mode = debug                  #To enable debug mode, use 'debug', anything else is normal mode.

[grid]

length          = 1.0         # Length of domain in each direction
dimension = 1                 # dimension of domain
grid_points = 200             # Number of points in one direction


[solver]

thermal_conductivity    = 1.0               # Thermal conductivity k_0
solver_name = jacobi                        # Use either jacobi or gauss
order = 2                                   # Order of accuracy of stencil, use 2 or 4
error_TOL       = 0.00000000000000001       # Tolerance
max_iters       = 1000000                   # Maximum number of iterations
```

  - Modes

    The debug mode can be activated by using `mode = debug`. Anything else is assumed to mean not in debug mode. It gives out a verbose output, an example can be found below.

    There is also a verification mode which is recommended to always be turned on, since post-processing scripts grep for certain strings to aggregate the data for convergence plots.

  - Grid options

    The options are domain length `length`, dimension `dimension` (which can be 1D or 2D) and number of grid points in one direction `grid_points`.

  - Solver options

    The solver options are essentially specifications of physical parameters such as `thermal_conductivity` and other specifications such as error tolerance `error_TOL`, order of accuracy desired `order`, name of the solver `solver_name` and maximum iterations allowed `max_iters`. DO NOT change the `error_TOL = 1e-17` and `max_iters = 1000000` if you want to compare to reference solutions.

- **Verification procedures** By default, the verification mode is always on. You can tweak that in `input.dat` if you don't want to keep it on.

  It is recommended to always be turned on, since post-processing scripts grep for certain strings to aggregate the data for convergence plots.

  Here is a sample output for verification mode on , debug mode off, 1D gauss 2nd order solution -

```
--> verification_mode = 1
--> mode          = no_debug
--> n             = 10
--> dimension     = 1
```

```
--> length        = 1.000000
--> order         = 2
--> error_TOL     = 1.0000000000000001e-17
--> thermal_conductivity = 1.000000
--> max_iters     = 1000000
--> solver        = gauss

VERIFICATION MODE -
L2 norm of error for n 10    is    0.048363774712789243

------------------------------------------------------------------------------------------------------------------
GRVY Performance timing - Performance Timings:                         |     Mean         Variance        Count
--> q_order2_dim1                    : 1.72210e-03 secs ( 30.1486 %) | [1.72210e-03  0.00000e+00           1]
--> main                             : 1.38593e-03 secs ( 24.2633 %) | [1.38593e-03  0.00000e+00           1]
--> write_results_output_file        : 1.26815e-03 secs ( 22.2014 %) | [1.26815e-03  0.00000e+00           1]
--> T_exact_order2_dim1              : 1.14799e-03 secs ( 20.0977 %) | [1.14799e-03  0.00000e+00           1]
--> l2_norm                          : 8.03471e-05 secs (  1.4066 %) | [2.94312e-07  2.31745e-13         273]
--> gauss                            : 5.81741e-05 secs (  1.0184 %) | [5.81741e-05  0.00000e+00           1]
--> print_verification_mode          : 2.09808e-05 secs (  0.3673 %) | [2.09808e-05  0.00000e+00           1]
--> assemble_A                       : 5.96046e-06 secs (  0.1043 %) | [5.96046e-06  0.00000e+00           1]
--> print_matrix_A                   : 1.90735e-06 secs (  0.0334 %) | [1.90735e-06  0.00000e+00           1]
--> solve                            : 9.53674e-07 secs (  0.0167 %) | [9.53674e-07  0.00000e+00           1]
--> assemble_q                       : 0.00000e+00 secs (  0.0000 %) | [0.00000e+00  0.00000e+00           1]
--> GRVY_Unassigned                  : 1.57356e-05 secs (  0.2755 %)

                  Total Measured Time = 5.71203e-03 secs ( 99.9332 %)
------------------------------------------------------------------------------------------------------------------
```

- **Debug mode output** With debug mode on, the output looks as -

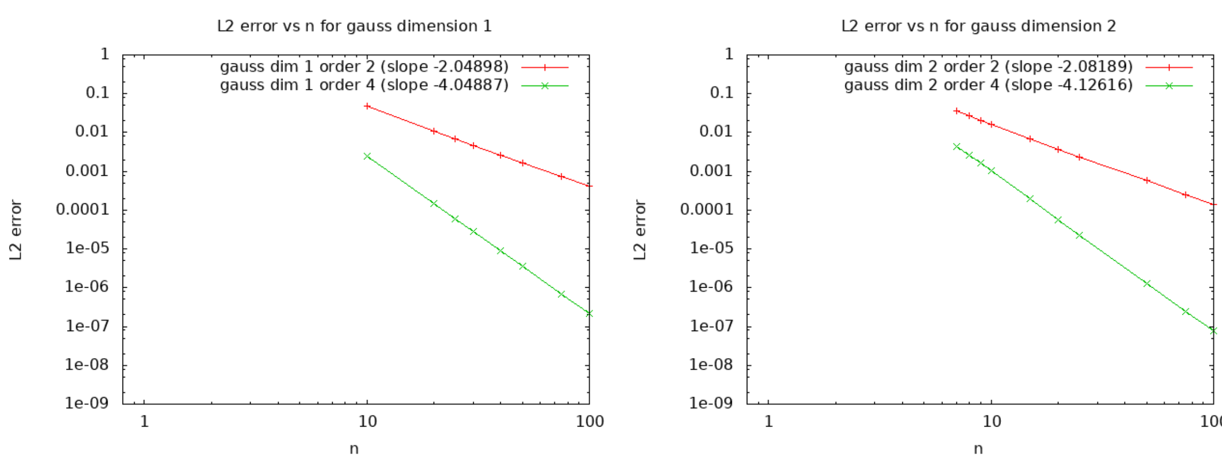
```
--> verification_mode = 1
--> mode          = debug
 Registering user-supplied default value for grid/grid_points
--> n             = 10
 Registering user-supplied default value for grid/dimension
--> dimension     = 1
 Registering user-supplied default value for grid/length
--> length        = 1.000000
 Registering user-supplied default value for solver/order
--> order         = 2
 Registering user-supplied default value for solver/error_TOL
--> error_TOL     = 1.0000000000000001e-17
 Registering user-supplied default value for solver/thermal_conductivity
--> thermal_conductivity = 1.000000
 Registering user-supplied default value for solver/max_iters
--> max_iters     = 1000000
--> solver        = gauss

VERIFICATION MODE -
L2 norm of error for n 10    is    0.048363774712789243

DEBUG MODE - printing A in MATLAB compatible form
A = [1  0  0  0  0  0  0  0  0  0  ;
-1  2  -1  0  0  0  0  0  0  0  ;
0  -1  2  -1  0  0  0  0  0  0  ;
0  0  -1  2  -1  0  0  0  0  0  ;
0  0  0  -1  2  -1  0  0  0  0  ;
0  0  0  0  -1  2  -1  0  0  0  ;
0  0  0  0  0  -1  2  -1  0  0  ;
0  0  0  0  0  0  -1  2  -1  0  ;
0  0  0  0  0  0  0  -1  2  -1  ;
0  0  0  0  0  0  0  0  0  1  ;
]

DEBUG MODE - printing q in MATLAB compatible form
q = [1
0.37336077072775858
0.084634015730502873
-0.24369393582936677
-0.45799478645826142
-0.45799478645826142
-0.24369393582936708
0.084634015730502651
0.37336077072775847
1
]

DEBUG MODE - Compare vector values
q               T_exact         T_computed
1.000000        1.000000        1.000000
```

```
0.373361        0.766044        0.756306
0.084634        0.173648        0.139251
−0.243694       −0.500000       −0.562437
−0.457995       −0.939693       −1.020432
−0.457995       −0.939693       −1.020432
−0.243694       −0.500000       −0.562437
0.084634        0.173648        0.139251
0.373361        0.766044        0.756306
1.000000        1.000000        1.000000

--------------------------------------------------------------------------------------------------------------------------------
GRVY Performance timing − Performance Timings:                                |    Mean          Variance         Count
−−> q_order2_dim1                      : 1.86706e−03 secs ( 27.8743 %) | [1.86706e−03  0.00000e+00          1]
−−> main                               : 1.71304e−03 secs ( 25.5749 %) | [1.71304e−03  0.00000e+00          1]
−−> write_results_output_file          : 1.44100e−03 secs ( 21.5135 %) | [1.44100e−03  0.00000e+00          1]
−−> T_exact_order2_dim1                : 1.21903e−03 secs ( 18.1996 %) | [1.21903e−03  0.00000e+00          1]
−−> print_matrix_A                     : 1.03951e−04 secs (  1.5519 %) | [1.03951e−04  0.00000e+00          1]
−−> l2_norm                            : 9.27448e−05 secs (  1.3846 %) | [3.39725e−07  2.79576e−13        273]
−−> print_compare_q_Texact_Tcomputed   : 7.39098e−05 secs (  1.1034 %) | [7.39098e−05  0.00000e+00          1]
−−> print_vector_q                     : 7.20024e−05 secs (  1.0750 %) | [7.20024e−05  0.00000e+00          1]
−−> gauss                              : 5.57899e−05 secs (  0.8329 %) | [5.57899e−05  0.00000e+00          1]
−−> print_verification_mode            : 3.19481e−05 secs (  0.4770 %) | [3.19481e−05  0.00000e+00          1]
−−> assemble_A                         : 8.10623e−06 secs (  0.1210 %) | [8.10623e−06  0.00000e+00          1]
−−> solve                              : 1.90735e−06 secs (  0.0285 %) | [1.90735e−06  0.00000e+00          1]
−−> matrix_order2_dim1                 : 9.53674e−07 secs (  0.0142 %) | [9.53674e−07  0.00000e+00          1]
−−> assemble_q                         : 0.00000e+00 secs (  0.0000 %) | [0.00000e+00  0.00000e+00          1]
−−> GRVY_Unassigned                    : 1.57356e−05 secs (  0.2349 %)

                 Total Measured Time = 6.69813e−03 secs ( 99.9858 %)
--------------------------------------------------------------------------------------------------------------------------------
```

- **Verification Exercise**

  - Gauss solver

    These are the grid convergence plots for the gauss-seidel solver. The expected slopes were -2 and -4 for 2nd and 4th order respectively. We have slopes pretty close to these expected values.



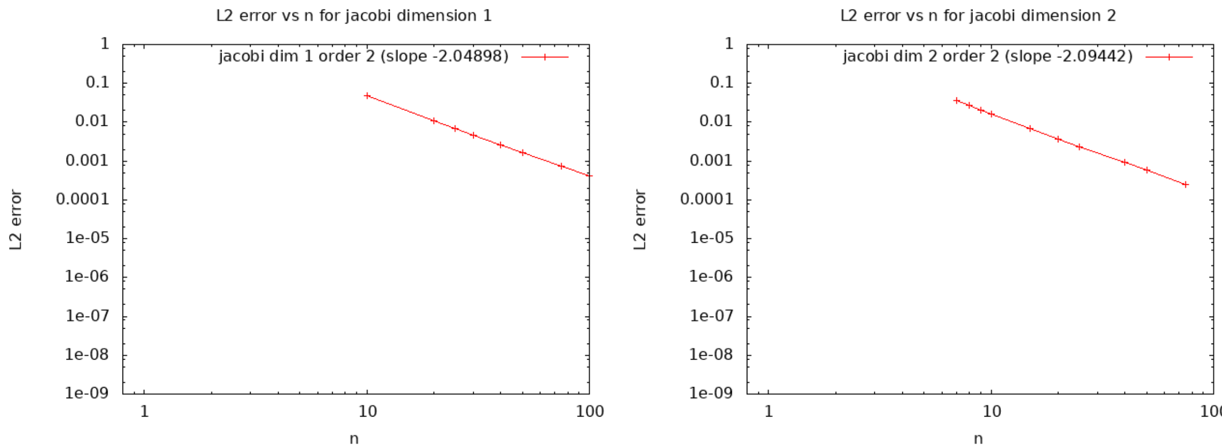
(a) 1D

(b) 2D

Figure 1: Convergence analysis of gauss-seidel solver

  - Jacobi solver

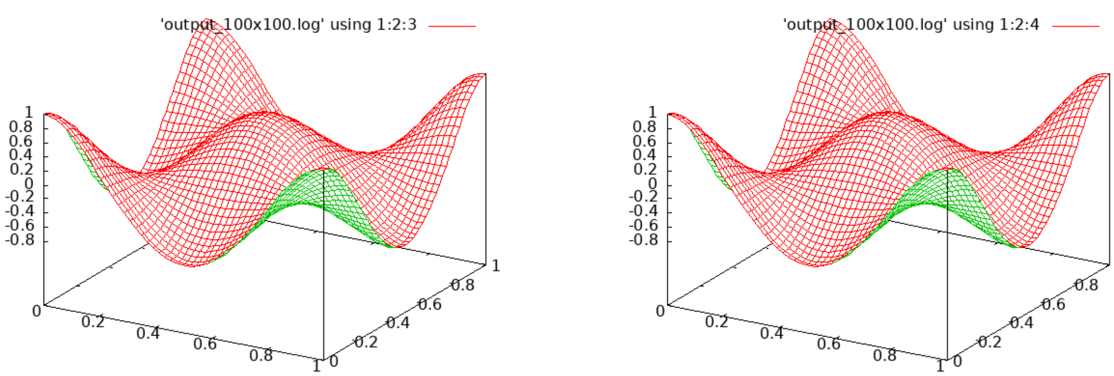
These are the grid convergence plots for the jacobi solver. We don't consider jacobi with 4th order solver since it is not stable. The expected slope was -2 for 2nd order. We have slopes pretty close to this expected value.



(a) 1D (b) 2D

Figure 2: Convergence analysis of jacobi solver

Just for completeness, here we have two surface plots for a 4th order gauss 2D simulation on a `100x100` grid. Both plots look essentially similar.



(a) T_exact (b) T_computed Gauss 4th order 100x100

Figure 3: Surface plots

- **Runtime performance**

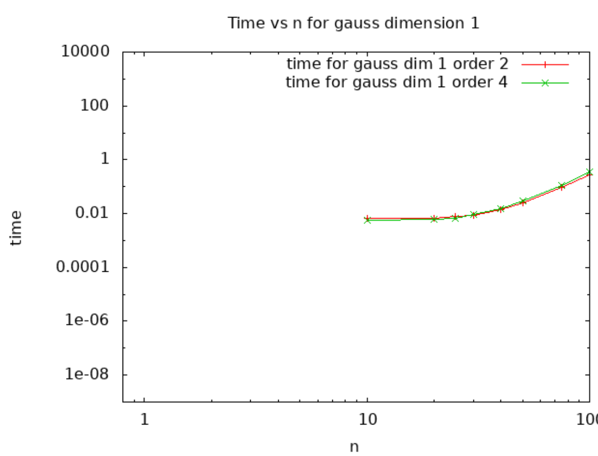
  A sample of time output for the given input configuration is shown below.

```
—> verification_mode = 1
—> mode             = no_debug
```

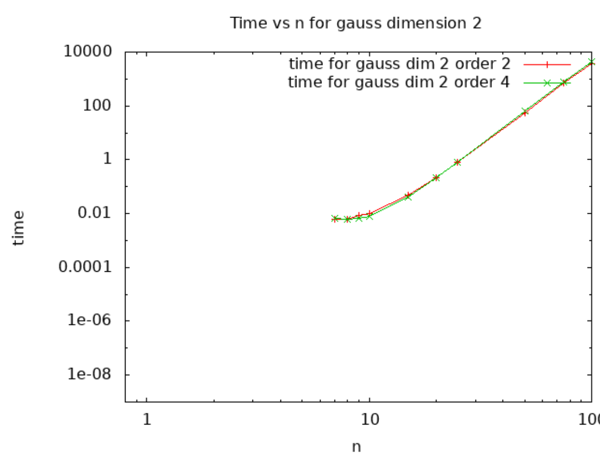
```
---> n              = 20
---> dimension      = 2
---> length         = 1.000000
---> order          = 4
---> error_TOL      = 1.0000000000000001e-17
---> thermal_conductivity = 1.000000
---> max_iters      = 1000000
---> solver         = gauss

VERIFICATION MODE --
L2 norm of error for n 20    is    5.7410182758840052e-05

--------------------------------------------------------------------------------------------------------------
GRVY Performance timing - Performance Timings:                       |      Mean       Variance       Count
---> gauss                            : 2.02494e-01 secs ( 94.6842 %) | [2.02494e-01  0.00000e+00          1]
---> write_results_output_file        : 4.57191e-03 secs (  2.1378 %) | [4.57191e-03  0.00000e+00          1]
---> q_order4_dim2                    : 1.84488e-03 secs (  0.8626 %) | [1.84488e-03  0.00000e+00          1]
---> main                             : 1.34730e-03 secs (  0.6300 %) | [1.34730e-03  0.00000e+00          1]
---> T_exact_order4_dim2              : 1.22809e-03 secs (  0.5742 %) | [1.22809e-03  0.00000e+00          1]
---> l2_norm                          : 9.53197e-04 secs (  0.4457 %) | [7.93670e-07  2.06032e-13       1201]
---> matrix_order4_dim2               : 7.10964e-04 secs (  0.3324 %) | [7.10964e-04  0.00000e+00          1]
---> print_matrix_A                   : 6.26087e-04 secs (  0.2928 %) | [6.26087e-04  0.00000e+00          1]
---> print_verification_mode          : 2.81334e-05 secs (  0.0132 %) | [2.81334e-05  0.00000e+00          1]
---> assemble_A                       : 5.00679e-06 secs (  0.0023 %) | [5.00679e-06  0.00000e+00          1]
---> print_vector_q                   : 3.09944e-06 secs (  0.0014 %) | [3.09944e-06  0.00000e+00          1]
---> print_compare_q_Texact_Tcomputed : 2.86102e-06 secs (  0.0013 %) | [2.86102e-06  0.00000e+00          1]
---> assemble_T_exact                 : 2.14577e-06 secs (  0.0010 %) | [2.14577e-06  0.00000e+00          1]
---> solve                            : 1.90735e-06 secs (  0.0009 %) | [1.90735e-06  0.00000e+00          1]
---> assemble_q                       : 9.53674e-07 secs (  0.0004 %) | [9.53674e-07  0.00000e+00          1]
---> GRVY_Unassigned                  : 4.19617e-05 secs (  0.0196 %)

                Total Measured Time = 2.13863e-01 secs (100.0000 %)
--------------------------------------------------------------------------------------------------------------
```

The total time for various cases for various solver configurations is plotted and can be found below.

- Gauss solver



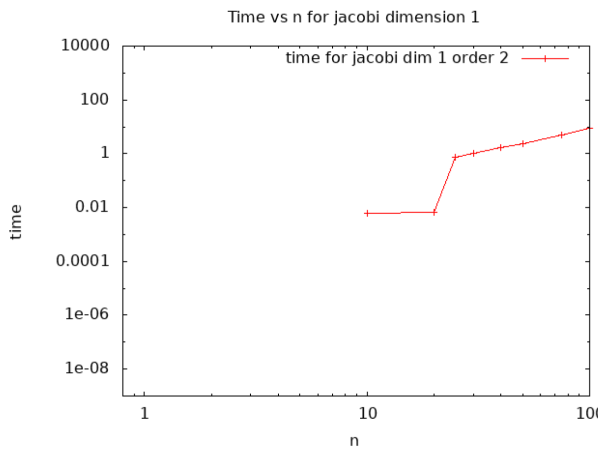
(a) 1D



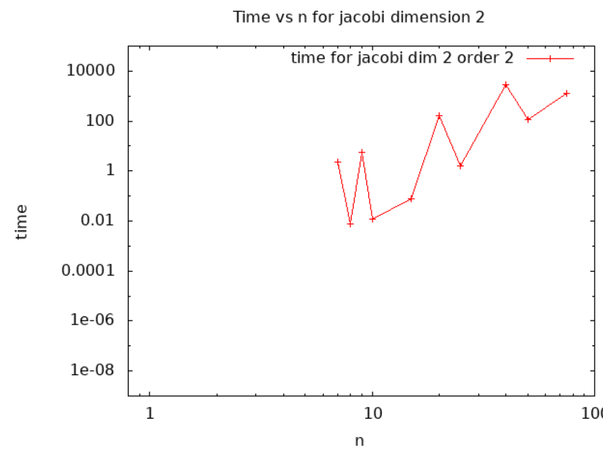
(b) 2D

Figure 4: Runtime analysis of gauss solver

- Jacobi



(a) 1D



(b) 2D

Figure 5: Runtime analysis of jacobi solver

- **Regression testing**

  `make check` runs the regression tests, which are stored in `proj01/tests/test.sh`. The sample output of `make check` is given below. (the colour is red due to LaTeXbut the test has actually passed). It shows only 1 test but technically the `test.sh` script contains 5 tests with the 4th test actually containing 4 tests and the 5th containing 2 tests. This makes a total of 9 tests.

```
Making check in src
make[1]: Entering directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/src'
make[1]: Nothing to be done for `check'.
make[1]: Leaving directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/src'
Making check in tests
make[1]: Entering directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
make  check-TESTS
make[2]: Entering directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
make[3]: Entering directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
PASS: test.sh
make[4]: Entering directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
make[4]: Nothing to be done for `all'.
make[4]: Leaving directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
============================================================================
Testsuite summary for FULL-PACKAGE-NAME VERSION
============================================================================
# TOTAL: 1
# PASS:  1
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[3]: Leaving directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
make[2]: Leaving directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
make[1]: Leaving directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01/tests'
make[1]: Entering directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01'
make[1]: Leaving directory `/home1/07665/shrey911/temp/cse380-2020-student-Shreyas911/proj01'
```

  Alternatively, you can run `proj01/tests/test.sh` directly, which gives the following output to `std::out`-

```
$ ./test.sh
```

```
✓ verify that the code is compiling
✓ verify that the verification mode runs fine
✓ verify that the debug mode runs fine
✓ verify all gauss solver outputs match reference outputs
✓ verify all jacobi solver outputs match reference outputs
```