Numerical Solution of Steady-State Heat Equation

Jiaqi Li

November 2018

1 Governing Equations

In this document, we attempt to solve the steady-state heat equation in one- and two- dimensions. In 1D, we have

$$-kT''(x) = q(x), \quad x \in \Omega \tag{1}$$

In 2D, we have

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

where k is the thermal conductivity, T is temperature, and q is a given heat source term. To simplify the problem, we make the following assumptions:

- k is constant throughout the domain.
- $\Omega = (0, l)$ in 1D and $\Omega = (0, l) \times (0, l)$ in 2D.
- Dirichlet boundary condition:

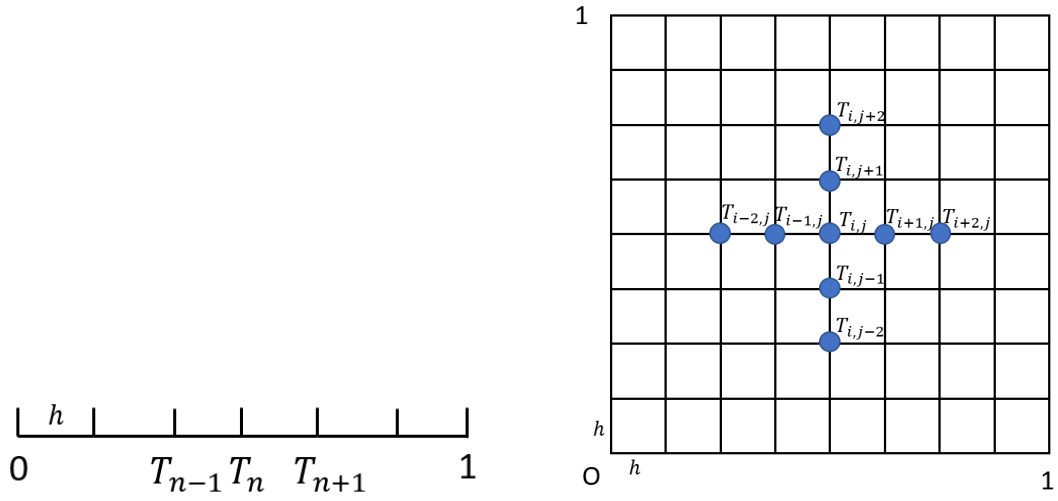
$$T|_{\partial\Omega} = g \tag{3}$$

  where $\partial\Omega$ denotes the boundary of domain Ω, and g is a given function defined on $\partial\Omega$.

From PDE theory we know that the combination of equation (1)/(2) and (3) forms a well-posed problem provided that q and g are smooth.

2 Discretization Schemes

Central difference is used to discretize the second-order derivative in the heat equation.



(a) 1D mesh

(b) 2D mesh

Figure 1: Representative figures of 1D and 2D mesh

2.1 1D case

In 1D case, we partition the domain into uniform intervals, as sketched in Figure 1(a). Assume $0 = x_0 < x_1 < ... < x_m = l$, and $x_n - x_{n-1} = h,\ n = 1, 2, ..., m$. The second-order approximation to the second-order derivative is

$$T''(x_n) = \frac{1}{h^2}\left(T_{n-1} - 2T_n + T_{n+1}\right) + O(h^2)$$

where $T_n = T(x_n)$. Then the discrete approximation to equation (1) is

$$\frac{1}{h^2}\left(T_{n-1} - 2T_n + T_{n+1}\right) + O(h^2) = -\frac{q_n}{k}$$

where $q_n = q(x_n)$. In order to get an approximate solution, we neglect the second-order term, and obtain

$$T_{n-1} - 2T_n + T_{n+1} = -\frac{q_n h^2}{k}, \quad n = 1, 2, ..., m-1 \tag{4}$$

The boundary condition (3) implies

$$T_0 = g(0), \quad T_m = g(1) \tag{5}$$

Equations (4) and (5) form a solvable linear system.

The fourth-order scheme is

$$T''(x_n) = \frac{1}{h^2}\left(-\frac{1}{12}T_{n-2} + \frac{4}{3}T_{n-1} - \frac{5}{2}T_n + \frac{4}{3}T_{n+1} - \frac{1}{12}T_{n+2}\right) + O(h^4)$$

The corresponding discrete approximation to equation (1) is

$$\frac{1}{h^2}\left(-\frac{1}{12}T_{n-2}+\frac{4}{3}T_{n-1}-\frac{5}{2}T_n+\frac{4}{3}T_{n+1}-\frac{1}{12}T_{n+2}\right)+O(h^4)=-\frac{q_n}{k}$$

For practical calculation, we neglect the fourth-order term, obtaining

$$-\frac{1}{12}T_{n-2}+\frac{4}{3}T_{n-1}-\frac{5}{2}T_n+\frac{4}{3}T_{n+1}-\frac{1}{12}T_{n+2}=-\frac{q_n h^2}{k},\quad n=2,3,...,m-2 \tag{6}$$

The boundary condition (5) still applies. However, we need two more equations to close the system. A simple choice is to use second-order difference at the nodes adjacent to the boundary.

$$T_{n-1}-2T_n+T_{n+1}=-\frac{q_n h^2}{k},\quad n=1,m-1 \tag{7}$$

Equations (6) together with (5) and (7) form a solvable linear system.

2.2 2D case

In 2D case, we partition the domain uniformly in both x and y direction, as illustrated in Figure 1(b). Let $0=x_0<x_1<...<x_m=l$, and $0=y_0<y_1<...<y_n=l$. Let Δx and Δy be the mesh size in x and y direction, respectively. The second-order approximation to derivatives in the heat equation (2) is

$$\begin{aligned}\nabla^2 T(x_i,y_j)&=\left(\frac{\partial}{\partial x^2}+\frac{\partial}{\partial y^2}\right)T|_{(x_i,y_j)}\\&=\frac{1}{\Delta x^2}(T_{i-1,j}-2T_{i,j}+T_{i+1,j})+\frac{1}{\Delta y^2}(T_{i,j-1}-2T_{i,j}+T_{i,j+1})+O(\Delta x^2)+O(\Delta y^2)\end{aligned}$$

The approximation to the equation is

$$\frac{1}{\Delta x^2}(T_{i-1,j}-2T_{i,j}+T_{i+1,j})+\frac{1}{\Delta y^2}(T_{i,j-1}-2T_{i,j}+T_{i,j+1})+O(\Delta x^2)+O(\Delta y^2)=-\frac{q_{i,j}}{k}$$

where $q_{i,j}=q(x_i,y_j)$. When $m=n$, and therefore $\Delta x=\Delta y=h$, the above equation can be simplified as

$$T_{i,j-1}+T_{i-1,j}-4T_{i,j}+T_{i+1,j}+T_{i,j+1}=-\frac{q_{i,j}h^2}{k},\quad i,j=1,2,...,n-1 \tag{8}$$

where we have omitted the second-order error $O(h^2)$. The boundary condition (3) implies

$$T_{0,j}=g(0,y_j),\quad T_{n,j}=g(1,y_j),\quad T_{i,0}=g(x_i,0),\quad T_{i,n}=g(x_i,1) \tag{9}$$

for $i=1,2,...,n-1$ and $j=0,1,...,n$. Equations (8) and (9) form a closed linear system.

The fourth-order approximation to derivatives is

$$\begin{aligned}\nabla^2 T(x_i,y_j)=&\frac{1}{\Delta x^2}\left(-\frac{1}{12}T_{i-2,j}+\frac{4}{3}T_{i-1,j}-\frac{5}{2}T_{i,j}+\frac{4}{3}T_{i+1,j}-\frac{1}{12}T_{i+2,j}\right)+O(\Delta x^4)\\&+\frac{1}{\Delta y^2}\left(-\frac{1}{12}T_{i,j-2}+\frac{4}{3}T_{i,j-1}-\frac{5}{2}T_{i,j}+\frac{4}{3}T_{i,j+1}-\frac{1}{12}T_{i,j+2}\right)+O(\Delta y^4)\end{aligned}$$

The approximation to the heat equation (2) is

$$\frac{1}{\Delta x^2}\left(-\frac{1}{12}T_{i-2,j}+\frac{4}{3}T_{i-1,j}-\frac{5}{2}T_{i,j}+\frac{4}{3}T_{i+1,j}-\frac{1}{12}T_{i+2,j}\right)+O(\Delta x^4)$$
$$+\frac{1}{\Delta y^2}\left(-\frac{1}{12}T_{i,j-2}+\frac{4}{3}T_{i,j-1}-\frac{5}{2}T_{i,j}+\frac{4}{3}T_{i,j+1}-\frac{1}{12}T_{i,j+2}\right)+O(\Delta y^4)=-\frac{q_{i,j}}{k}$$

Assume $m=n, \Delta x=\Delta y=h$. Then the above equation can be simplified:

$$-\frac{1}{12}T_{i,j-2}+\frac{4}{3}T_{i,j-1}-\frac{1}{12}T_{i-2,j}+\frac{4}{3}T_{i-1,j}-5T_{i,j}$$
$$+\frac{4}{3}T_{i+1,j}-\frac{1}{12}T_{i+2,j}+\frac{4}{3}T_{i,j+1}-\frac{1}{12}T_{i,j+2}=-\frac{q_{i,j}h^2}{k} \quad (10)$$

for $i,j=2,3,...,n-2$, where we have omitted the fourth-order error $O(h^4)$. The boundary condition (9) still holds, but we need $(4n-8)$ more equations to close the system. A simple choice is to use second-order difference at the nodes adjacent to the boundary:

$$T_{i,j-1}+T_{i-1,j}-4T_{i,j}+T_{i+1,j}+T_{i,j+1}=-\frac{q_{i,j}h^2}{k} \quad (11)$$

where $1\le i,j\le n-1$ and at least one of i,j equals 1 or $n-1$. Equations (10) together with (9) and (11) form a closed linear system.

3 Implementation of Numerical Methods

3.1 Resulting linear system

3.1.1 1D case

Let

$$\boldsymbol{u}=\begin{bmatrix}T_0 & T_1 & \dots & T_m\end{bmatrix}^T$$

be a column vector of unknowns. The second-order finite difference scheme (4), (5) gives the following tridiagonal linear system

$$\begin{bmatrix}1 & & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & & 1\end{bmatrix}\begin{bmatrix}T_0\\T_1\\T_2\\\vdots\\T_{m-1}\\T_m\end{bmatrix}=\begin{bmatrix}g(0)\\-q_1h^2/k\\-q_2h^2/k\\\vdots\\-q_{m-1}h^2/k\\g(1)\end{bmatrix}$$

where zeros in the matrix are omitted. There are three non-zero entries on an interior row of the matrix.

The fourth-order difference scheme (6), (5), and (7) results in the following linear system

$$\begin{bmatrix} 1 & & & & & & \\ 1 & -2 & 1 & & & & \\ -\frac{1}{12} & \frac{4}{3} & -\frac{5}{2} & \frac{4}{3} & -\frac{1}{12} & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & -\frac{1}{12} & \frac{4}{3} & -\frac{5}{2} & \frac{4}{3} & -\frac{1}{12} \\ & & & & 1 & -2 & 1 \\ & & & & & & 1 \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ T_{m-2} \\ T_{m-1} \\ T_m \end{bmatrix} = \begin{bmatrix} g(0) \\ -q_1 h^2/k \\ -q_2 h^2/k \\ \vdots \\ -q_{m-2} h^2/k \\ -q_{m-1} h^2/k \\ g(1) \end{bmatrix}$$

There are five non-zero entries on an interior row of the matrix.

3.1.2 2D case

Assume $m = n$. Let

$$\boldsymbol{\alpha}_j = \begin{bmatrix} T_{0,j} & T_{1,j} & \dots & T_{n,j} \end{bmatrix}^T, \quad j = 0, 1, \dots, n$$

The second-order difference scheme (8), (9) gives

$$\boldsymbol{C}\boldsymbol{\alpha}_{j-1} + \boldsymbol{B}\boldsymbol{\alpha}_j + \boldsymbol{C}\boldsymbol{\alpha}_{j+1} = \boldsymbol{f}_j, \quad j = 1, 2, \dots, n-1 \tag{12}$$

where

$$\boldsymbol{B} = \begin{bmatrix} 1 & & & & \\ 1 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 1 \\ & & & & 1 \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 0 \end{bmatrix}$$

are $(n+1) \times (n+1)$ matrices, and

$$\boldsymbol{f}_j = \begin{bmatrix} T_{0,j} & -q_{1,j} h^2/k & \dots & -q_{n-1,j} h^2/k & T_{n,j} \end{bmatrix}^T$$

is column vector with length $(n+1)$. Rewriting equation (12) using block matrices, we obtain

$$\begin{bmatrix} \boldsymbol{I} & & & & \\ \boldsymbol{C} & \boldsymbol{B} & \boldsymbol{C} & & \\ & \ddots & \ddots & \ddots & \\ & & \boldsymbol{C} & \boldsymbol{B} & \boldsymbol{C} \\ & & & & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_0 \\ \boldsymbol{\alpha}_1 \\ \vdots \\ \boldsymbol{\alpha}_{n-1} \\ \boldsymbol{\alpha}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_0 \\ \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{n-1} \\ \boldsymbol{\alpha}_n \end{bmatrix}$$

Note that $\boldsymbol{\alpha}_0$ and $\boldsymbol{\alpha}_n$ on the right hand side is given by boundary condition (9). There are five non-zero entries on an interior row of the matrix.

The second-order scheme (10), (9), (11) results in the following equations for $\boldsymbol{\alpha}_j$:

$$\boldsymbol{F}\boldsymbol{\alpha}_{j-2} + \boldsymbol{E}\boldsymbol{\alpha}_{j-1} + \boldsymbol{D}\boldsymbol{\alpha}_j + \boldsymbol{E}\boldsymbol{\alpha}_{j+1} + \boldsymbol{F}\boldsymbol{\alpha}_{j+2} = \boldsymbol{f}_j, \quad j = 2, 3, ..., n-2$$

where

$$\boldsymbol{D} = \begin{bmatrix} 1 & & & & & & \\ 1 & -4 & 1 & & & & \\ -\frac{1}{12} & \frac{4}{3} & -5 & \frac{4}{3} & -\frac{1}{12} & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & -\frac{1}{12} & \frac{4}{3} & -5 & \frac{4}{3} & -\frac{1}{12} \\ & & & & 1 & -4 & 1 \\ & & & & & & 1 \end{bmatrix}$$

$$\boldsymbol{E} = \begin{bmatrix} 0 & & & & & & \\ & 1 & & & & & \\ & & \frac{4}{3} & & & & \\ & & & \ddots & & & \\ & & & & \frac{4}{3} & & \\ & & & & & 1 & \\ & & & & & & 0 \end{bmatrix}, \boldsymbol{F} = \begin{bmatrix} 0 & & & & & & \\ & 0 & & & & & \\ & & -\frac{1}{12} & & & & \\ & & & \ddots & & & \\ & & & & -\frac{1}{12} & & \\ & & & & & 0 & \\ & & & & & & 0 \end{bmatrix},$$

are $(n+1) \times (n+1)$ matrices. Then the global system is

$$\begin{bmatrix} \boldsymbol{I} & & & & & & \\ \boldsymbol{C} & \boldsymbol{B} & \boldsymbol{C} & & & & \\ \boldsymbol{F} & \boldsymbol{E} & \boldsymbol{D} & \boldsymbol{E} & \boldsymbol{F} & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & \boldsymbol{F} & \boldsymbol{E} & \boldsymbol{D} & \boldsymbol{E} & \boldsymbol{F} \\ & & & & \boldsymbol{C} & \boldsymbol{B} & \boldsymbol{C} \\ & & & & & & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_0 \\ \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \vdots \\ \boldsymbol{\alpha}_{n-2} \\ \boldsymbol{\alpha}_{n-1} \\ \boldsymbol{\alpha}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_0 \\ \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \vdots \\ \boldsymbol{f}_{n-2} \\ \boldsymbol{f}_{n-1} \\ \boldsymbol{\alpha}_n \end{bmatrix}$$

There are nine non-zero entries on an interior row of the matrix.

3.2 Iterative solutions for the resulting linear system

Once the linear system is formed, we utilize iterative methods to solve the system. Suppose the system is

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where $\boldsymbol{A}$ is an $n \times n$ matrix.

3.2.1 Jacobi method

Algorithm 1 Jacobi method

$\boldsymbol{x}^{(0)} \leftarrow 0$

for $k = 0, 1, \ldots$ **do**

$$x_i^{(k+1)} \leftarrow \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, ..., n$$

  if $||\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}|| < tol$ **then**

    exit

  end if

end for

3.2.2 Gauss-Seidel method

Algorithm 2 Gauss-Seidel method

$\boldsymbol{x}^{(0)} \leftarrow 0$

for $k = 0, 1, \ldots$ **do**

$$x_i^{(k+1)} \leftarrow \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, ..., n$$

  if $||\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}|| < tol$ **then**

    exit

  end if

end for

3.3 Required memory

Denote the total degrees of freedom by N, and let n be the number of grid points in one direction. Then $N = n$ in 1D, and $N = n^2$ in 2D. We need to store two discrete temperature fields (to calculate the difference between two iterations) and one source term field, which take up $3 \times 8N = 24N$ Bytes using double precision.

Besides, we need to store the non-zero entries in the matrix A and their positions. In our code, we store a sparse matrix A as its N rows and each row stores the position of non-zero entries (an integer array) and their values (array of double precision real numbers). Suppose the number of non-zero entries on each row is r, then each row would take up $r \times (4N + 8N) = 12rN$ Bytes.

From Section 3.1 we know that in 1D, $r = 3$ for second-order difference and $r = 5$ for fourth-order difference; in 2D, $r = 5$ for second-order difference and $r = 9$ for fourth-order difference.

Summing up the required memory for vectors and the matrix, we obtain Table 1.

	2nd order	4th order
1D	$60n$	$84n$
2D	$84n^2$	$132n^2$

Table 1: Required memory in different cases

4 Program Usage

This section describes how to build the program, input options, and verification procedures.

4.1 Build procedures

We use an autotools-based build system. With the configure script generated by autotools, we can generate Makefiles with the following commands (on Stampede2):

```
$ autoreconf --install
$ module load gnu7 boost hdf5
$ export PKGPATH=/work/00161/karl/stampede2/public/
$ ./configure FC=gfortran --with-masa=$PKGPATH/masa-gnu7-0.50 \
    --with-grvy=$PKGPATH/grvy-gnu7-0.34 --with-hdf5=$TACC_HDF5_DIR
```

Alternatively, the user can source the shell script "configure.sh", which does exactly the same thing. After successful invocation of "configure", we can proceed with the command "make" and build the code. Then a binary executable named "heateq" should be found in src/ directory.

4.2 Input options

The name of input file can be specified as a command line argument; if no argument is provided, the default name "input.dat" is assumed. We utilize the GRVY library for input parsing. An example of input file is provided below.

```
# input.dat: an input file example
dimen = 2               # dimension of the problem (1 for 1D, 2 for 2D)
side_Length = 1.0       # side length of domain
num_Mesh = 32           # number of meshes in one direction
```

```
order = 4               # order of discretization (2 or 4)
solver_Flag = 2         # solver type (1 - Jacobi, 2 - Gauss-Seidel, 3 - GMRES)

verification_Flag = 1   # flag for verification mode (1 - on, 0 - off)
debug_Flag = 1          # flag for debug mode (0 - off, 1 - standard, 2 - verbose)

k_0 = 1.0               # thermal conductivity

eps = 1.0e-12           # iterative solver tolerance
max_Iter = 250000       # max solver iterations
print_Iter = 1000       # print error every print_iter iterations
output_File = 'sol.dat' # name of output file
```

With those comments marked with "#", the meaning of various input options is clear.

4.3 Verification procedures

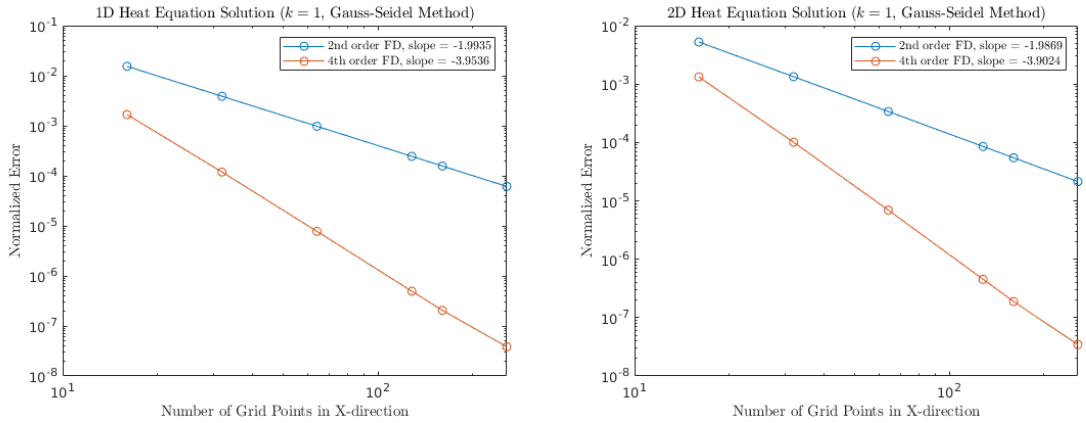
In our program, we use a manufactured solution for the steady-state heat equation available within the MASA library. Simply set verification_Flag to 1 in the input file to enable the verification mode. The program will call MASA to get the source term $q(x, y)$, and then solve the heat equation as described before. Finally, we compare the difference between our numerical solution and the exact solution from MASA, and output their normalized L^2 difference. An example of the stdout is shown below:

```
L2 error between numerical solution and MASA:
  7.767339199626751E-006
```

5 Program Verification and Performance

5.1 Verification results

In both 1D and 2D, we perform a uniform mesh refinement study: $n = 16, 32, 64, 128, 160, 256$. In Figure 2 we present the results for Gauss-Seidel method. From Figure 2(a), we see that in 1D, the slope for second-order scheme is -1.9935, and the slope for fourth-order scheme is -3.9536. From Figure 2(b), we see that in 2D, the slope for second-order scheme is -1.9869, and the slope for fourth-order scheme is -3.9024. They are all very close to the theoretical value of -2 and -4, confirming the correctness of our program.



(a) 1D problem

(b) 2D problem

Figure 2: Convergence plot for 1D and 2D problem

5.2 Runtime performance

GRVY library is utilized to measure runtime performance. In Figure 3, we present the result for 1D problem, where num_Mesh = 256, and we use second-order finite difference with Gauss-Seidel method to solve the linear system. From the result we see that the iterative solver (which updates the solution vector) takes up most of the time (89% here), and other part of Solve_System takes a large part (10%) too. In each iteration, besides calling functions to update the solution vector, the subroutine Solve_System computes the difference between two iterations, and decides whether to exit loop based on the difference.

```
-------------------------------------------------------------------------------------------------------------
Heat Equation Solver - Performance Timings:                          |      Mean      Variance       Count
--> Solve_System/Gauss-Seidel Iteration : 4.11932e+00 secs ( 89.3818 %) | [2.85210e-05  9.07937e-11     144431]
--> Solve_System                        : 4.67736e-01 secs ( 10.1490 %) | [4.67736e-01  0.00000e+00          1]
--> Initialize                          : 3.81494e-03 secs (  0.0828 %) | [3.81494e-03  0.00000e+00          1]
--> Postprocess                         : 2.90084e-03 secs (  0.0629 %) | [2.90084e-03  0.00000e+00          1]
--> Read_Input                          : 2.33889e-03 secs (  0.0507 %) | [2.33889e-03  0.00000e+00          1]
--> GRVY_Unassigned                     : 1.25675e-02 secs (  0.2727 %)

                  Total Measured Time = 4.60868e+00 secs (100.0000 %)
-------------------------------------------------------------------------------------------------------------
```

Figure 3: Runtime performance measurements

5.3 Code coverage

We use the gcov/lcov tool for code coverage. A suite of regression tests are run, which includes different cases: 1D and 2D, second- and fourth-order, Jacobi and Gauss-Seidel method. Figure 4 shows our result (before linking to PETSc and HDF5). To reproduce the result, configure with additional flag "--enable-coverage", and then run

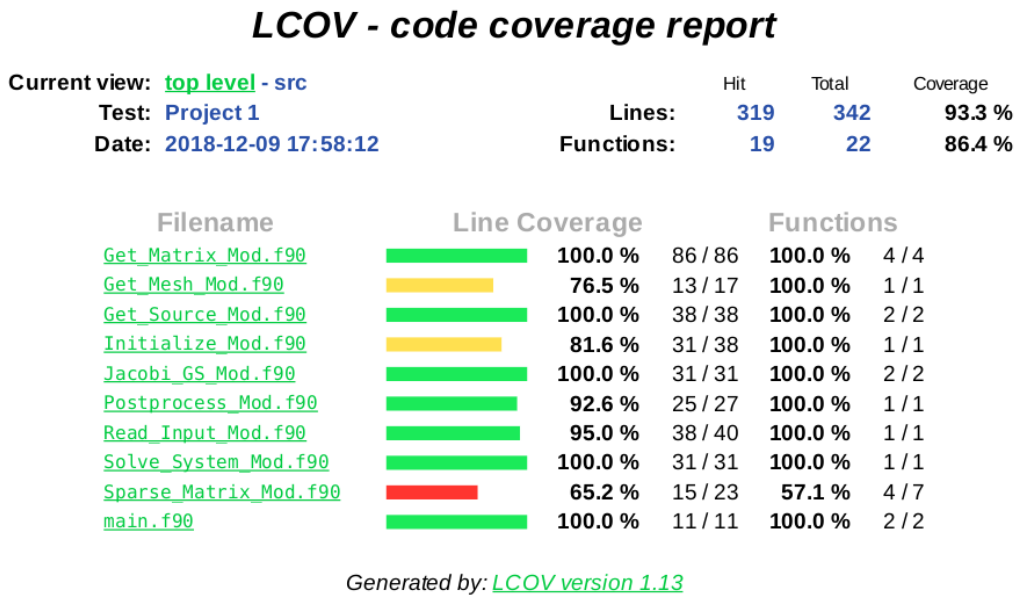


Figure 4: Code coverage report

```
$ make clean
$ make
$ make coverage
```

In Figure 4, note that the source file Sparse_Matrix_Mod.f90 has only 65.2% line coverage. Further examination reveals that there is a subroutine for outputing sparse matrix that is never called in our regression tests. This is fine, since the sparse matrix is only printed when we run the program in verbose output mode (debug_Flag = 2).

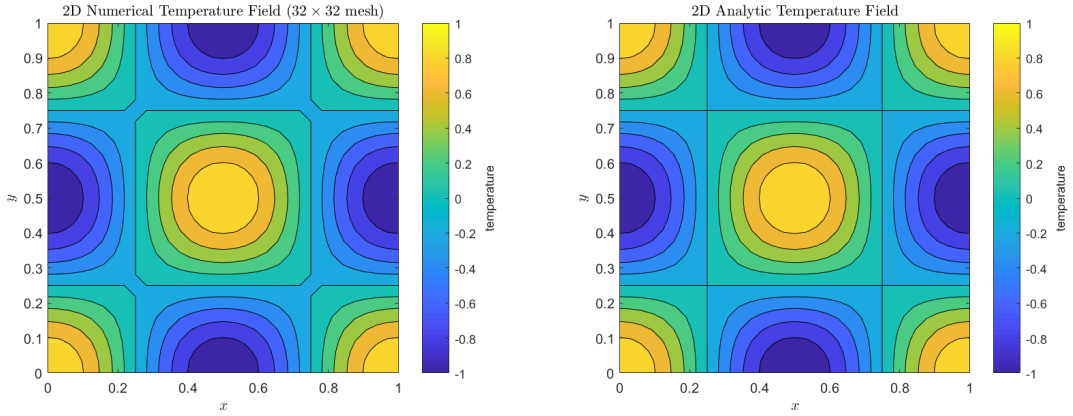
5.4 2D solution plot

As described in Section 4.3, we use the MASA library to get the source term, and solve for the temperature field. Figure 5(a) shows the result for a 32×32 mesh, obtained with fourth-order finite difference and Gauss-Seidel method. Figure 5(b) presents the corresponding analytic temperature field. The exact temperature field $T(x, y)$ is given by

$$T(x, y) = \cos(2\pi x) \cos(2\pi y)$$

It can be seen that our numerical result is very close to the analytic one.

Tools to produce these 2D plots: We use MATLAB's plotting utilities. First, read the data using function "h5read". Then plot with function "contourf". The labels and title are set with MATLAB functions as well.



(a) Numerical temperature field

(b) Analytic temperature field

Figure 5: 2D solution plot

6 PETSc Implementation

In addition to the internal Jacobi and Gauss-Seidel solvers, our application include (optional) support to use PETSc's GMRES solver. To enable PETSc solver, configure with an additional option "--with-petsc=$PETSC_DIR", and make sure to load module "gcc" and "petsc/3.9-uni". For convenience, the configure lines are reproduced below:

```
$ module load boost hdf5
$ module load gcc
$ module load petsc/3.9-uni
$ export PKGPATH=/work/00161/karl/stampede2/public/
$ ./configure FC=mpif90 FCFLAGS='-g -O0 -Wall' --with-masa=$PKGPATH/masa-gnu7-0.50 \
    --with-grvy=$PKGPATH/grvy-gnu7-0.34 --with-hdf5=$TACC_HDF5_DIR \
    --with-petsc=$PETSC_DIR
```

Then run "make" to build our program. The PETSc solver's tolerances can be set via command line:

```
$ ./heateq input.dat -ksp_atol <abtol> -ksp_rtol <rtol> -ksp_max_it <maxits>
```

where "abtol" is the absolute tolerance, "rtol" is the relative tolerance, and "maxits" is maximum iterations. The default values are 1×10^{-50}, 1×10^{-7}, 10000, respectively.

6.1 Verification of PETSc solver

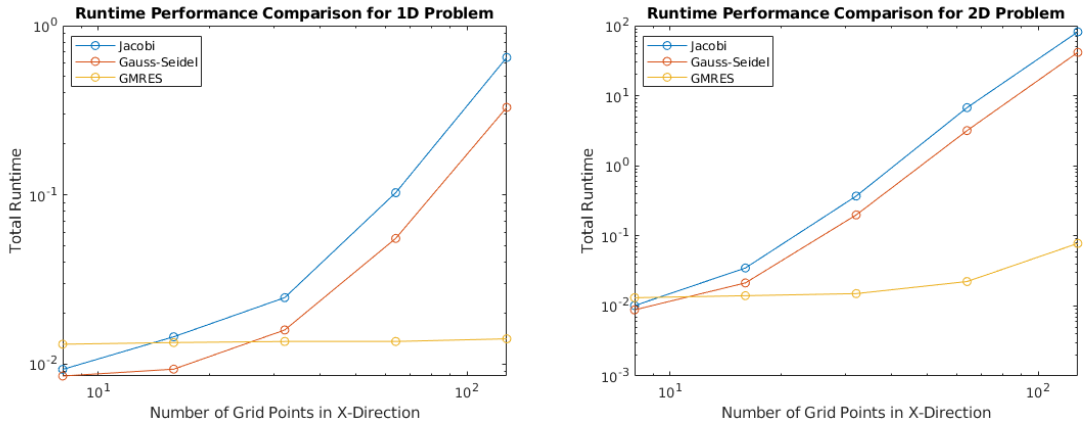
We redo the uniform mesh refinement study as before, and compare the PETSc result with our internal solvers. Table 2 shows the normalized error compared to MASA solution for 1D problem, second-order scheme. It can be seen that the results from PETSc and Gauss-Seidel are very close (the same in first four digits). Therefore the solution using PETSc also follows the same $O(h^2)$ convergence rate. Hence our implementation of PETSc is trustworthy.

num_Mesh	16	32	64	128	256
Gauss-Seidel	1.539E-2	3.882E-3	9.766E-4	2.450E-4	6.136E-4
PETSc	1.539E-2	3.882E-3	9.766E-4	2.450E-4	6.136E-5

Table 2: Normalized error compared to MASA for 1D problem, second-order scheme

6.2 Comparison of runtime performance

We compare the runtime performance of GMRES method from PETSc with Jacobi and Gauss-Seidel methods written by ourselves. As shown in Figure 6, GMRES is slower than Jacobi and Gauss-Seidel method when the number of grid points is small, which is due to the overhead of preparing PETSc objects; However, it scales much better to larger-size problems.



(a) 1D problem

(b) 2D problem

Figure 6: Comparison of total runtime for second-order scheme