Project 2

Mason Schechter

Dec 11 2020

1 Introduction

Here, we will be solving the steady-state heat equation:

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

where k is the thermal conductivity, T is the material temperature, and q is a heat source term. The expanded form of (1) in 1 dimension is:

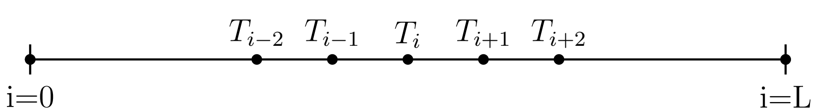
$$-k\frac{\delta^2 T(x)}{\delta x^2} = q(x) \tag{2}$$

The expanded form of (1) in 2 dimensions is:

$$-k(\frac{\delta^2 T(x)}{\delta x^2} + \frac{\delta^2 T(y)}{\delta y^2}) = q(x,y) \tag{3}$$

To solve the steady-state heat equation in 1 and 2 dimensions, we will first need to approximate the value of the second derivative terms in these equations. To do this, we will use 2nd- and 4th-order central finite-difference approximations. The resulting linear system from these numerical methods will be solved using two iterative solution mechanisms: Jacobi and Gauss-Seidel.

2 1 Dimension Approximations



Here, we are going to approximate the steady-state heat equation in 1 dimension. In the above diagram, T_i is the temperature at the point $x = i$, where $x \in [0, L]$. In this node-based scheme, we will be using constant mesh spacing of N points, such that the distance between a point, T_i, and the previous point, T_{i-1} or $T(x - \Delta x)$, or the next point, T_{i+1} or $T(x + \Delta x)$, is the same distance $\Delta x = L/N$.

We are assuming:

1. The mesh spacing is constant
2. Dirichlet boundary conditions are applied for any incomplete stencil
3. The values at the boundaries will be provided through manufactured solutions.
4. $T(x)$ is smooth and continuous for all $x \in [0, L]$
5. The values T_i and q_i are not changing over time

The 2nd-order approximation for the second derivative in (2) is:

$$\frac{\delta^2 T(x)}{\delta x^2} \approx \frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2} \tag{4}$$

Substituting this approximation into (2) gives us:

$$T_{i-1} - 2T_i + T_{i+1} \approx -\frac{q_i \Delta x^2}{k} \tag{5}$$

where the truncation error of this approximation is on the order of Δx^2. The resulting linear system looks like this:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & \dots & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ \vdots \\ \vdots \\ T_{L-1} \\ T_L \end{bmatrix} = \begin{bmatrix} -\frac{q_0 \Delta x^2}{k} \\ -\frac{q_1 \Delta x^2}{k} \\ -\frac{q_1 \Delta x^2}{k} \\ \vdots \\ -\frac{q_{L-1} \Delta x^2}{k} \\ -\frac{q_L \Delta x^2}{k} \end{bmatrix}$$

Each interior (non-boundary) row contains 3 non-zero entries and $N - 3$ zeros.

The 4th-order approximation for the second derivative in (2) is:

$$\frac{\delta^2 T(x)}{\delta x^2} \approx \frac{-T_{i-2} + 16T_{i-1} - 30T_i + 16T_{i+1} - T_{i+2}}{12\Delta x^2} \tag{6}$$

Substituting this approximation into (2) gives us:

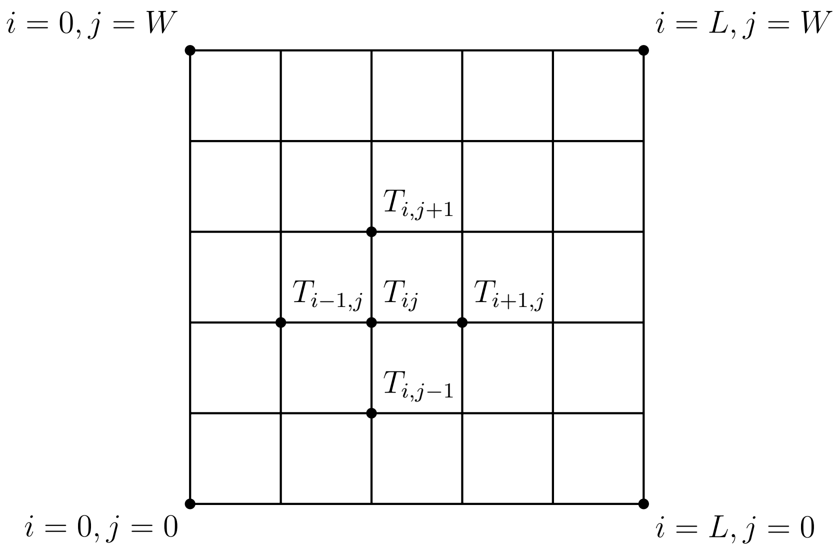
$$-T_{i-2} + 16T_{i-1} - 30T_i + 16T_{i+1} - T_{i+2} \approx -\frac{12 q_i \Delta x^2}{k} \tag{7}$$

where the truncation error of this approximation is on the order of Δx^4. The resulting linear system looks like this:

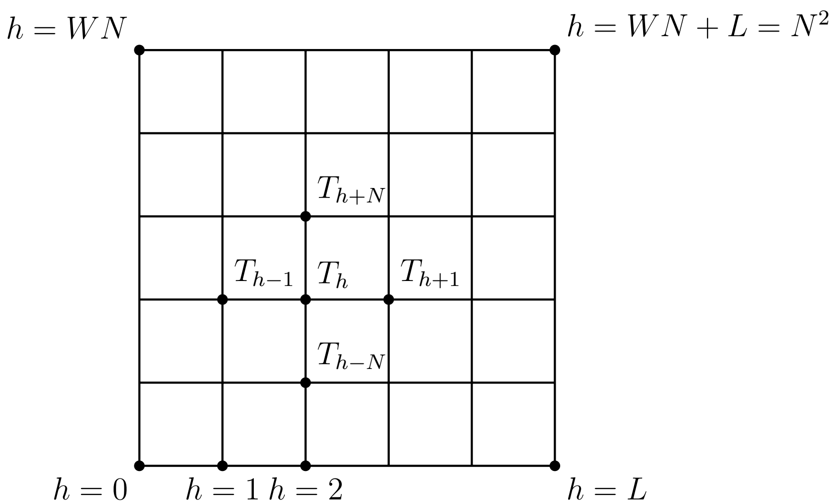
$$
\begin{bmatrix}
1 & 0 & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\
0 & 1 & 0 & 0 & \dots & \dots & \dots & \dots & 0 \\
-1 & 16 & -30 & 16 & -1 & 0 & \dots & \dots & 0 \\
0 & -1 & 16 & -30 & 16 & -1 & 0 & \dots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\
0 & \dots & \dots & 0 & -1 & 16 & -30 & 16 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
T_0 \\ T_1 \\ T_2 \\ \vdots \\ \vdots \\ \vdots \\ T_{L-2} \\ T_{L-1} \\ T_L
\end{bmatrix}
=
\begin{bmatrix}
-\frac{12q_0\Delta x^2}{k} \\ -\frac{12q_1\Delta x^2}{k} \\ -\frac{12q_2\Delta x^2}{k} \\ \vdots \\ \vdots \\ \vdots \\ -\frac{12q_{L-2}\Delta x^2}{k} \\ -\frac{12q_{L-1}\Delta x^2}{k} \\ -\frac{12q_L\Delta x^2}{k}
\end{bmatrix}
$$

Each interior row has 5 non-zero entries and $N-5$ zeros.

3 2 Dimension Approximations



Here, we are going to approximate the steady-state heat equation in 2 dimensions. In the above diagram of our node-based scheme, T_{ij} is the temperature at the point $x=i, y=j$ or $T(x=i, y=j)$, where $x \in [0, L]$ and $y \in [0, W]$. We are assuming a square domain, such that $L=W$. We will be using the same number of discretization points, N, for each dimension, such that $\Delta x = \Delta y = \Delta n$. However, we will need to make a mapping from this 2-D coordinate system to a 1-D vectored system. We will use the equation $h = i + j(N_x) = i + j(N_y)$ where N_x and N_y are the number of discretization points in the x- and y-directions, respectively. Again, they are the same value for the following solutions. Our new mesh grid will look like this:



We are assuming:

1. The mesh spacing is constant, such that $N_x = N_y$, therefore $\Delta x = \Delta y = \Delta n$
2. The domain is square, such that $L = W$
3. Dirichlet boundary conditions are applied for any incomplete stencil
4. The values at the boundaries will be provided through manufactured solutions.
5. $T(x, y)$ is smooth and continuous for all $x \in [0, L]$ and all $y \in [0, W]$
6. The values T_{ij} and q_{ij} are not changing over time

The 2nd-order approximation for the second derivative terms in (3) is:

$$\frac{\delta^2 T(x)}{\delta x^2} \approx \frac{T_{h-1} - 2T_h + T_{h+1}}{\Delta x^2} \tag{8}$$

$$\frac{\delta^2 T(y)}{\delta y^2} \approx \frac{T_{h-N} - 2T_h + T_{h+N}}{\Delta y^2} \tag{9}$$

Substituting this approximation into (3) gives us:

$$T_{h-N} + T_{h-1} - 4T_h + T_{h+1} + T_{h+N} \approx -\frac{q_h \Delta n^2}{k} \tag{10}$$

where the truncation error of this approximation is on the order of Δn^2. The resulting linear system looks like this:

$$\begin{bmatrix} \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \dots & Nzeros & 1 & -4 & 1 & Nzeros & 1 & 0 & \dots & \dots \\ \dots & 1 & Nzeros & 1 & -4 & 1 & Nzeros & 1 & 0 & \dots \\ \dots & 0 & 1 & Nzeros & 1 & -4 & 1 & Nzeros & 1 & 0 \\ \dots & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \dots & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \dots & 0 & \dots & \dots & \dots & \dots & 0 & 1 & 0 & 0 \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vdots \\ T_{h-1} \\ T_h \\ T_{h+1} \\ \vdots \\ \vdots \\ \vdots \\ T_{N^2-2} \\ T_{N^2-1} \\ T_{N^2} \end{bmatrix} = \begin{bmatrix} -\frac{q_0\Delta n^2}{k} \\ -\frac{q_1\Delta n^2}{k} \\ -\frac{q_2\Delta n^2}{k} \\ \vdots \\ \vdots \\ \vdots \\ -\frac{q_{N^2-2}\Delta n^2}{k} \\ -\frac{q_{N^2-1}\Delta n^2}{k} \\ -\frac{q_{N^2}\Delta n^2}{k} \end{bmatrix}$$

Each interior row contains 5 non-zero entries and N-5 zeros.

The 4th-order approximation for the second derivative terms in (3) is:

$$\frac{\delta^2 T(x)}{\delta x^2} \approx \frac{-T_{h-2} + 16T_{h-1} - 30T_h + 16T_{h+1} - T_{h+2}}{12\Delta n^2} \tag{11}$$

$$\frac{\delta^2 T(y)}{\delta y^2} \approx \frac{-T_{h-2N} + 16T_{h-N} - 30T_h + 16T_{h+N} - T_{h+2N}}{12\Delta n^2} \tag{12}$$

Substituting this approximation into (3) gives us:

$$-T_{h-2N}+16T_{h-N}-T_{h-2}+16T_{h-1}-60T_h+16T_{h+1}-T_{h+2}+16T_{h+N}-T_{h+2N} \approx -\frac{12q_h\Delta n^2}{k} \tag{13}$$

where the truncation error of this approximation is on the order of Δn^4. The resulting coefficient matrix looks like this:

$$A = \begin{bmatrix} \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \dots & Nzeros & -1 & 16 & -60 & 16 & -1 & Nzeros & 16 & Nzeros \\ \dots & 16 & Nzeros & -1 & 16 & -60 & 16 & -1 & Nzeros & \dots \\ \dots & Nzeros & 16 & Nzeros & -1 & 16 & -60 & 16 & -1 & Nzeros \\ \dots & -1 & Nzeros & 16 & Nzeros & -1 & 16 & -60 & 16 & -1 \\ \dots & 0 & -1 & Nzeros & 16 & Nzeros & -1 & 16 & -60 & 16 \\ \dots & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Each interior row has 9 non-zero entries. The resulting linear system looks like this:

$$
A \begin{bmatrix} \vdots \\ T_{h-1} \\ T_h \\ T_{h+1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ T_{N^2-1} \\ T_{N^2} \end{bmatrix} = \begin{bmatrix} -\frac{12q_0\Delta n^2}{k} \\ -\frac{12q_1\Delta n^2}{k} \\ -\frac{12q_2\Delta n^2}{k} \\ \vdots \\ \vdots \\ \vdots \\ -\frac{12q_{N^2-2}\Delta n^2}{k} \\ -\frac{12q_{N^2-1}\Delta n^2}{k} \\ -\frac{12q_{N^2}\Delta n^2}{k} \end{bmatrix}
$$

4 Jacobi Method

Initialize two, 1 x N^2 T vectors, T_{old} and T_{new};
Set each value in T_{old} to 0;
while *currentError* $>$ *errorThreshold* **do**
    for *each h from* 0 *to* N^2 **do**
        solve for T_h with the values from T_{old} for the other components;
        store result T_h in T_{new};
    end
    currentError = L2norm(T_{old}, T_{new}) ;
    $T_{old} = T_{new}$
end

Estimated memory needed for 1D: $8N^2$ bytes for coefficient matrix, $16N$ bytes for temp vectors, $8N$ bytes for RHS vector.
Estimated memory needed for 2D: $8N^4$ bytes for coefficient matrix, $16N^2$ bytes for temp vectors, $8N^2$ bytes for RHS vector.

5 Gauss-Seidel Method

Estimated memory needed for 1D: $8N^2$ bytes for coefficient matrix, $16N$ bytes for temp vectors, $8N$ bytes for RHS vector.
Estimated memory needed for 2D: $8N^4$ bytes for coefficient matrix, $16N^2$ bytes for temp vectors, $8N^2$ bytes for RHS vector.

Initialize two, $1xN^2$ T vectors, T_{old} and T_{new};
Set each value in T_{old} to 0;
$T_{new} = T_{old}$;
while *currentError* $>$ *errorThreshold* **do**
    for *each h from* 0 *to* N^2 **do**
        solve for T_h with the values from T_{new} for the other components;
        store result T_h in T_{new};
    end
    currentError = L2norm(T_{old}, T_{new}) ;
    $T_{old} = T_{new}$;
end

6 Build Instructions

To build the executable, you will need compiled versions of MASA and GRVY libs. For this project, I used the pre-compiled versions available in work/00161/karl/stampede2/public. For regression testing, you will need a compiled version of BATS. For this project, I used the pre-compiled version available in work/00161/karl/stampede2/public/bats/bin

Configure:

$ autoreconf -f -i
$ module load hdf5
$ export PKGPATH=/work/00161/karl/stampede2/public
$ ./configure −−with-masa=$PKGPATH/masa-gnu7-0.50
−−with-grvy=$PKGPATH/grvy-gnu7-0.34 −−with-hdf5=$TACC_HDF5_DIR

Make:

$ make

Building test directory: (ensure hd5f is loaded)

$ export PATH=/work/00161/karl/stampede2/public/bats/bin/:$PATH
$ make check

Enabling code coverage:

First, swap the default compiler to a compiler provided by Dr. Schulz
$ export CLASSPATH=/work/00161/karl/stampede2/public
$ export MODULEPATH=$CLASSPATH/ohpc/pub/modulefiles/:$MODULEPATH
$ module swap intel gnu7

Then, re-configure using the '–enable-coverage' flag

$ ./configure −−with-masa=$PKGPATH/masa-gnu7-0.50
−−with-grvy=$PKGPATH/grvy-gnu7-0.34 −−with-hdf5=$TACC_HDF5_DIR
–enable-coverage

$ make check
$ make coverage

Building with Petsc:

$module load petsc
$ ./configure −−with-masa=$PKGPATH/masa-gnu7-0.50
−−with-grvy=$PKGPATH/grvy-gnu7-0.34 −−with-hdf5=$TACC_HDF5_DIR
$ make

7 Input Options

The input file is called input.dat, found in the /src directory.
DISCRETIZATION_POINTS = the number of points per direction
SOLVER_TYPE = which iterative solver, Jacobi, Gauss-Seidel or Petsc
ORDER = the order of accuracy, 2nd or 4th
DIMENSIONS = dimensions of the initial problem, 1 or 2
MAX_ITERATIONS = maximum iterations for the solvers
ERROR_THRESHOLD = minimum error between iterations

A_X = coefficient for the Sin(x) term in manufactured solution
B_Y = coefficient for the Cos(y) term in manufactured solution
K_0 = thermal conductivity for manufactured solution

X_MIN = left boundary in 1D and 2D space
X_MAX = right boundary in 1D and 2D space
Y_MIN = lower boundary in 2D space
Y_MAX = upper boundary in 2D space

OUTPUT_MODE = verbosity of stdout, INFO or DEBUG
RUN_MODE = whether or not to check numerical solution vs the analytical solution, VERIFICATION or NONE

8 Verification Output and Analysis

To run in verification mode, make sure RUN_MODE is set to 'VERIFICATION'. The only difference between 'VERIFICATION' and 'NONE' is stdout will include an L2 norm value in 'VERIFICATION' mode. Both modes will output the computed numerical solution.

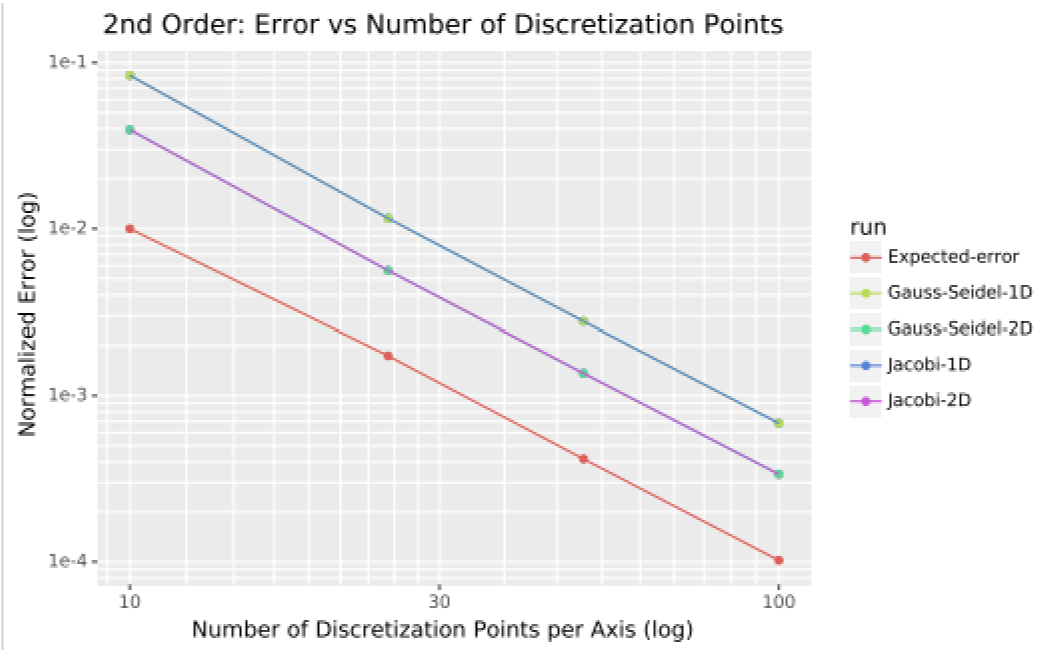
Example standard out in verification mode:

```
# DISCRETIZATION_POINTS=10
# SOLVER_TYPE=Gauss-Seidel
# ORDER=4
# DIMENSIONS=1
# MAX_ITERATIONS=250000
# ERROR_THRESHOLD=1e-9
# A_X=10.0
# B_Y=10.0
# K_0=1.0
# X_MIN=0
# X_MAX=1
# Y_MIN=0
# Y_MAX=1
# OUTPUT_MODE=INFO
# RUN_MODE=VERIFICATION
```

```
1.000000000000000 0.443666021702229 -0.619244941901932 -0.997510743829360 -0.266699501681184 0.7632265
```

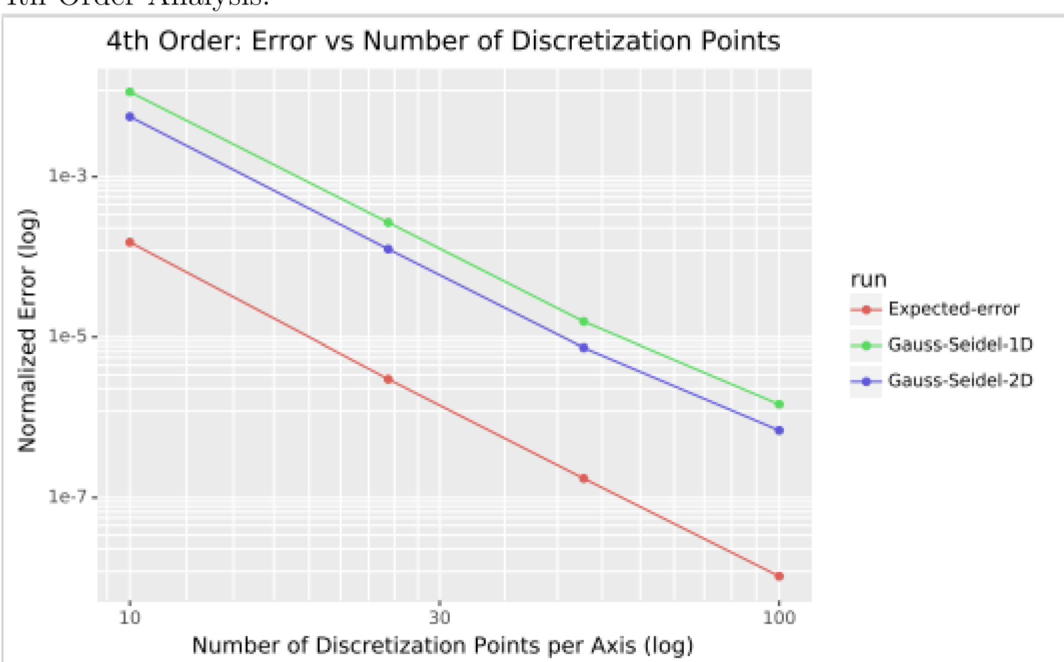
```
This solution took 25694 iterations

L2 Norm of numerical vs analytical solutions: 0.000683564618811
-----------------------------------------------------------------------------------------------------
My Timer - Performance Timings:                                     |      Mean      Variance      Co
--> Iterative solution               : 2.26582e-01 secs ( 96.9563 %) | [2.26582e-01  0.00000e+00
--> Initializing analytical solution : 1.67990e-03 secs (  0.7188 %) | [1.67990e-03  0.00000e+00
--> Initializing linear system       : 5.31673e-05 secs (  0.0228 %) | [5.31673e-05  0.00000e+00
--> GRVY_Unassigned                  : 5.37992e-03 secs (  2.3021 %)

                 Total Measured Time = 2.33695e-01 secs (100.0000 %)
-----------------------------------------------------------------------------------------------------
```

2nd Order Analysis:



Expected slope: -2.0
Gauss-Seidel-1D slope: -2.09
Gauss-Seidel-2D slope: -2.07
Jacobi-1D slope: -2.09
Jacobi-2D slope: -2.07

4th Order Analysis:



Expected slope: -4.0

Gauss-Seidel-1D slope: -3.90
Gauss-Seidel-2D slope: -3.91

9 Runtime Analysis

	Jacobi		Gauss-Seidel			
Dimension	1	2	1		2	
Order	2	2	2	4	2	4
	N = 10		N = 10			
Initialize analytical	1.90282e-03 secs	1.70708e-03 secs	1.71018e-03 secs	3.33500e-03 secs	1.72305e-03 secs	1.76120e-03 secs
Initialize linear system	1.50204e-05 secs	5.50747e-05 secs	1.28746e-05 secs	5.04971e-04 secs	6.79493e-05 secs	6.10352e-05 secs
Iterations	5.81741e-05 secs	1.58820e-02 secs	3.40939e-05 secs	3.19481e-05 secs	9.35793e-04 secs	9.33886e-04 secs
	N = 25		N = 25			
Initialize analytical	1.86992e-03 secs	1.66988e-03 secs	1.72400e-03 secs	1.77193e-03 secs	1.72782e-03 secs	1.81985e-03 secs
Initialize linear system	3.09944e-05 secs	5.56207e-03 secs	1.59740e-05 secs	1.59740e-05 secs	1.82891e-03 secs	1.79815e-03 secs
Iterations	1.11794e-03 secs	3.13859e-01 secs	4.83036e-04 secs	5.64098e-04 secs	1.11904e-01 secs	2.45112e-01 secs
	N = 50		N = 50			
Initialize analytical	1.79195e-03 secs	9.61614e-03 secs	2.05398e-03 secs	1.78599e-03 secs	2.07400e-03 secs	1.76787e-03 secs
Initialize linear system	2.78950e-05 secs	8.15530e-02 secs	2.90871e-05 secs	2.90871e-05 secs	3.00291e-02 secs	2.60570e-02 secs
Iterations	2.30331e-02 secs	2.66319e+01 secs	8.73995e-03 secs	7.24006e-03 secs	1.04849e+01 secs	8.76134e+00 secs
	N = 100		N = 100			
Initialize analytical	1.79410e-03 secs	1.73593e-03 secs	2.09618e-03 secs	1.74308e-03 secs	2.03991e-03 secs	2.08306e-03 secs
Initialize linear system	8.48770e-05 secs	4.10881e-01 secs	7.29561e-05 secs	7.60555e-05 secs	4.54892e-01 secs	4.75551e-01 secs
Iterations	3.72655e-01 secs	1.10364e+03 sec	1.16501e-01 sec	9.68540e-02 secs	5.60175e+02 secs	5.09548e+02 secs

Petsc runtime, 1D, 2nd order, 100 disc points:

```
---------------------------------------------------------------------------------------
My Timer - Performance Timings:                                        |      Mea
--> Iterative solution                : 2.60749e-02 secs (  7.5634 %) | [2.60749
--> Initializing analytical solution : 1.23610e-02 secs (  3.5855 %) | [1.23610
--> Initializing linear system        : 2.19393e-03 secs (  0.6364 %) | [2.19393
--> GRVY_Unassigned                   : 3.04123e-01 secs ( 88.2148 %)

              Total Measured Time = 3.44753e-01 secs (100.0000 %)
---------------------------------------------------------------------------------------
```

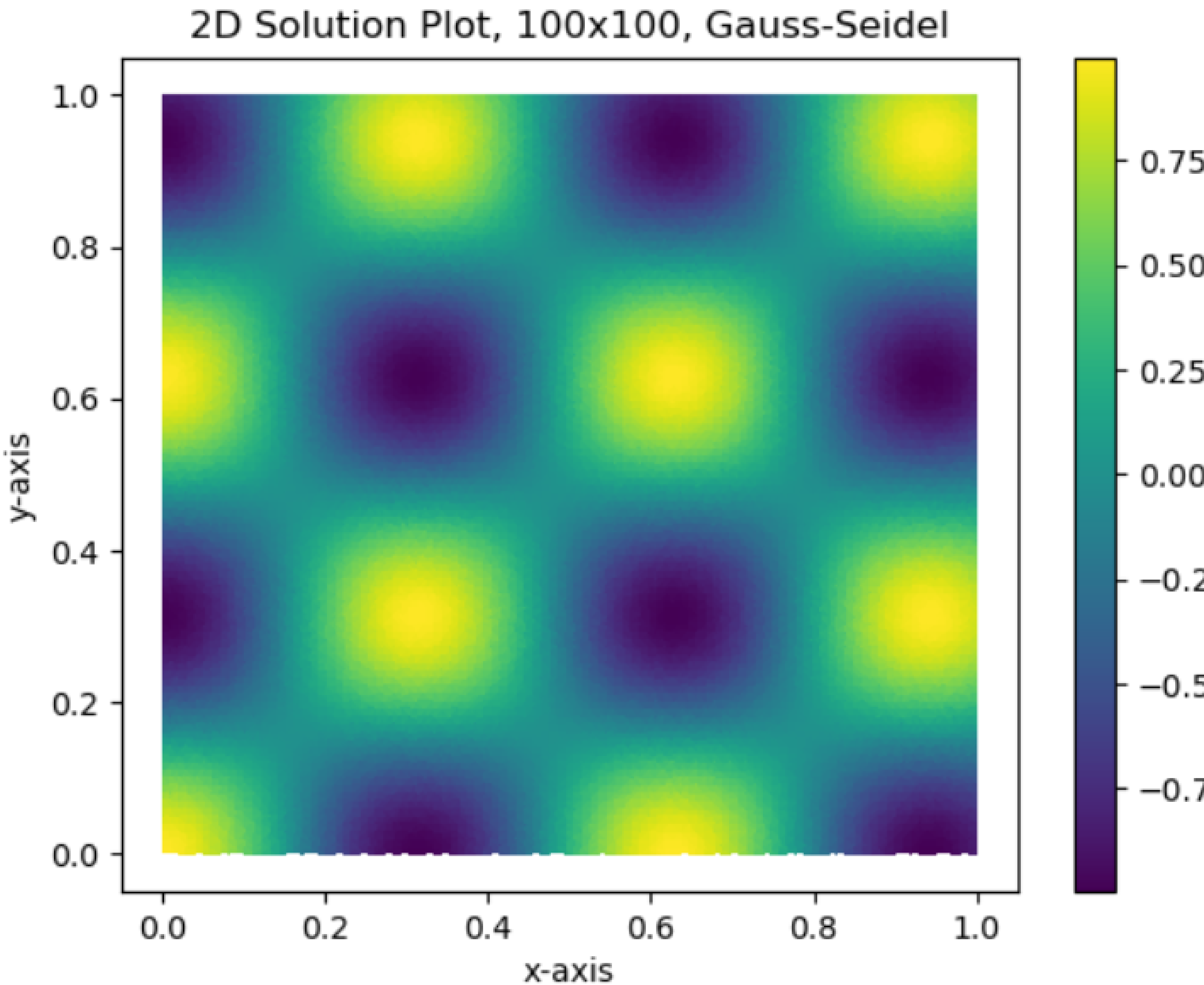
Petsc runtime, 2D, 2nd order, 100x100 disc points:

```
---------------------------------------------------------------------------------------
My Timer - Performance Timings:                                        |      Mear
--> Iterative solution                : 9.22044e-01 secs ( 73.3975 %) | [9.22044
--> Initializing linear system        : 1.36311e-02 secs (  1.0851 %) | [1.36311
--> Initializing analytical solution : 1.26801e-02 secs (  1.0094 %) | [1.26801
--> GRVY_Unassigned                   : 3.07879e-01 secs ( 24.5081 %)

              Total Measured Time = 1.25623e+00 secs (100.0000 %)
---------------------------------------------------------------------------------------
```

We can see that, in 1D, there is not an appreciable speed difference between my

Jacobi/Gauss-Seidel implementations and the Petsc GMRES solver. However, Petsc GMRES is 2-3 orders of magnitude faster than either of my implementations.

10 Solution Figure



This figure was made with the script included in /src, called 'hdf5_plot.py'. The packages used are h5py, numpy, and matplotlib. The input file is the 'sol.h5' file produced by our heat_equation_solver binary.

11 Code Coverage

LCOV - code coverage report

				Hit	Total	Coverage
Current view:	top level					
Test:	Project 1		**Lines:**	566	584	96.9 %
Date:	2020-12-10 20:04:07		**Functions:**	28	28	100.0 %

Directory	Line Coverage			Functions	
test		96.9 %	566 / 584	100.0 %	28 / 28

Generated by: LCOV version 1.14