Midterm Project

Name: Jianghuai Liu

Net id: jliu115

Consider the 2-D heat equation about $u(x,y,t)$ with a static source distribution $r(x,y)$:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + r(x,y) \qquad (1)$$

The initial condition and boundary condition(s) are specified as:

$$u(x,y,0) = 0 \quad for\ (x,y) \in \mathbf{\Omega} \qquad (2)$$

$$u(x,y,t) = 0 \quad for\ (x,y) \in \mathbf{\Gamma_W} \qquad (3)$$

$$\hat{n} \cdot \nabla u(x,y,t) = 0 \quad for\ (x,y) \in \partial\mathbf{\Omega} \backslash \mathbf{\Gamma_W} \qquad (4)$$

Geometry Set-up:

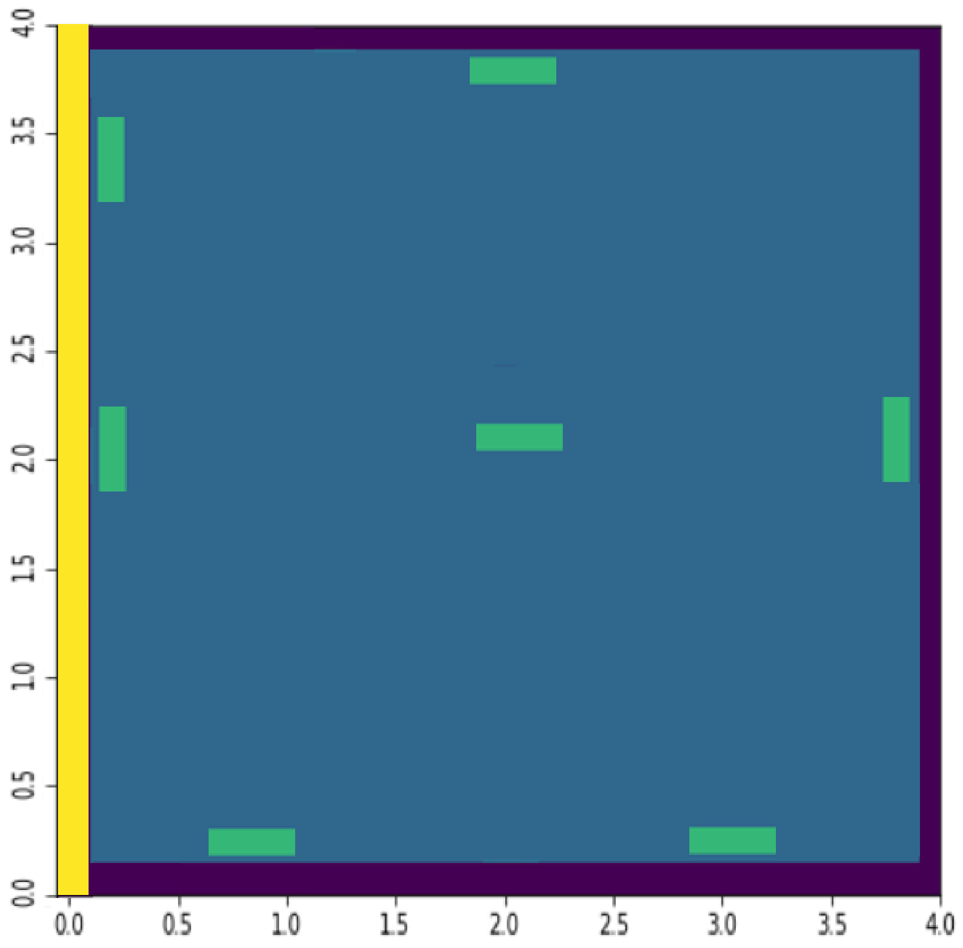

Figure 1. Geometry Setup

Figure 1 shows the geometry setup. The entire domain is set as a $4m \times 4m$ square. The yellow boundary denotes the window, on which the Dirichlet boundary condition is enforced, as expressed in (3). The deep blue boundary denotes the wall, on which the Neumann boundary condition is enforced, as expressed in (4). The green rectangles represent the heat sources, so that the distribution of $r(x, y)$ follows exactly the distribution of green rectangles. The magnitude of source distribution is illustrated in the figure below:

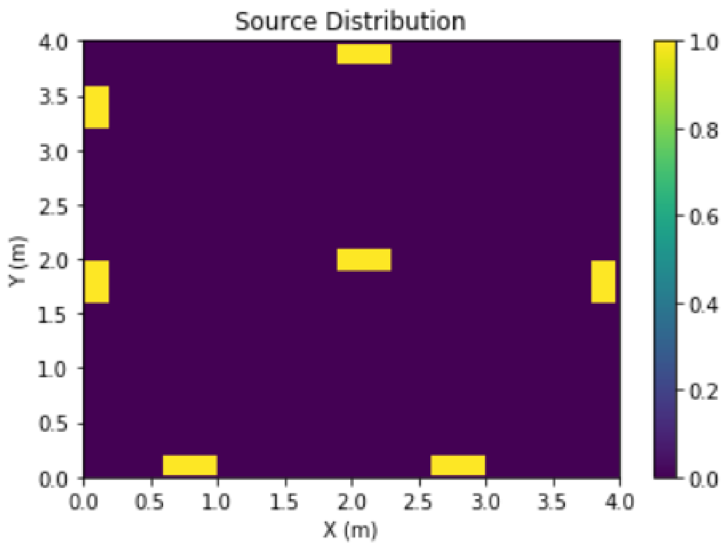

Figure 2. Source Distribution

Finite Difference Discretization:

To solve for $u(x, y, t)$ using finite difference method, we discretize the original 2-D heat equation shown in (1) into (with central difference in space):

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \theta \left[\frac{u_{i-1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i+1,j}^{n+1}}{\Delta x^2} + \frac{u_{i,j-1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j+1}^{n+1}}{\Delta y^2} + r_{i,j}^{n+1} \right]$$
$$+ (1-\theta) \left[\frac{u_{i-1,j}^{n} - 2u_{i,j}^{n} + u_{i+1,j}^{n}}{\Delta x^2} + \frac{u_{i,j-1}^{n} - 2u_{i,j}^{n} + u_{i,j+1}^{n}}{\Delta y^2} + r_{i,j}^{n} \right] \quad (5)$$

where θ denotes the implicit factor. Setting $\theta = 0$ we obtain a purely explicit scheme, while setting $\theta > 0$ we have implicit scheme. It is worthy to notice that setting $\theta = \frac{1}{2}$ represents the Crank-Nicolson scheme, which also has second-order accuracy in time, as we discovered in homework 2.

In this project, if we keep $\Delta x = \Delta y = h$, and assume a time-independent source (such that $r_{i,j}^{n+1} = r_{i,j}^{n} = r_{i,j}$ for every i, j in domain), then the discretized equation (5) can be rearranged as:

$$u_{i,j}^{n+1}(1+4\theta F)-\theta F\left(u_{i-1,j}^{n+1}+u_{i+1,j}^{n+1}+u_{i,j-1}^{n+1}+u_{i,j+1}^{n+1}\right)$$
$$=u_{i,j}^{n}(1-4(1-\theta)F)+(1-\theta)F\left(u_{i-1,j}^{n}+u_{i+1,j}^{n}+u_{i,j-1}^{n}+u_{i,j+1}^{n}\right)$$
$$+r_{i,j}\Delta t \qquad (6)$$

where $F=\frac{\Delta t}{\Delta x^2}=\frac{\Delta t}{\Delta y^2}=\frac{\Delta t}{h^2}$. Equation (6) represents a linear system looks like:

$$A\boldsymbol{u}^{n+1}=B\boldsymbol{u}^{n}+\Delta t\boldsymbol{r} \qquad (7)$$

where, if denoting the total number of grid points along each direction to be N_x and N_y respectively such that $i\in[0,N_x-1]$ and $j\in[0,N_y-1]$, $A\in\boldsymbol{R}_{N_yN_x\times N_yN_x}$ and $B\in\boldsymbol{R}_{N_yN_x\times N_yN_x}$ are the coefficient matrix for the next step solution $\boldsymbol{u}^{n+1}\in\boldsymbol{R}_{N_yN_x}$ and current step solution $\boldsymbol{u}^{n}\in\boldsymbol{R}_{N_yN_x}$, respectively.

Coefficient Matrix Build-up and Boundary Conditions Handling:

To build the full 2-D heat equation solver, we need to construct the coefficient matrix A and B, as introduced in (7), based on specific geometry setup and associated boundary conditions. For my geometry setup as illustrated in Figure 1, I set: $(i,j)_{domain}\in[1,N_x-2]\times[1,N_y-2]$, $(i,j)_{window}\in[0,0]\times[0,N_y-1]$ and $(i,j)_{wall}\in[1,N_x-1]\times[0,0]\cup[1,N_x-1]\times[N_y-1,N_y-1]\cup[N_x-1,N_x-1]\times[0,N_y-1]$. Before assigning elements within each coefficient matrix, we define the 2D to 1D index conversion function $p(i,j)$:

$$p(i,j):=jN_x+i$$

So that we could arrange the 2D distribution of solution $u(x,y,t_n)$ and source $r(x,y)$ into their 1D array:

$$u_{p(i,j)}^{n}=u_{i,j}^{n}$$

$$r_{p(i,j)}=r_{i,j}$$

On Domain:

For grid points on domain: $(i,j)_{domain}\in[1,N_x-2]\times[1,N_y-2]$, the full version of (6) holds true, so the corresponding elements of coefficient matrix are assigned as:

$$A_{p(i,j),p(i,j)}=1+4\theta F$$

$$A_{p(i,j),p(i-1,j)}=A_{p(i,j),p(i+1,j)}=A_{p(i,j),p(i,j-1)}=A_{p(i,j),p(i,j+1)}=-\theta F$$

$$B_{p(i,j),p(i,j)} = 1 - 4(1-\theta)F$$

$$B_{p(i,j),p(i-1,j)} = B_{p(i,j),p(i+1,j)} = B_{p(i,j),p(i,j-1)} = B_{p(i,j),p(i,j+1)} = (1-\theta)F$$

On Window (Dirichlet BC):

For grid points on window: $(i,j)_{window} \in [0,0] \times [0, N_y - 1]$, we enforce zero Dirichlet boundary condition, so the corresponding element of coefficient matrix is assigned as:

$$A_{p(i,j),p(i,j)} = 1$$

On Wall (Neumann BC):

For grid points on wall $(i,j)_{wall} \in [1, N_x - 1] \times [0,0] \cup [1, N_x - 1] \times [N_y - 1, N_y - 1] \cup [N_x - 1, N_x - 1] \times [0, N_y - 1]$, we enforce Neumann boundary condition. To reach second-order accuracy in space, <u>we will apply central difference to approximate the first-order derivative in Neumann boundary condition</u>. For example, at the top boundary where index $j = N_y - 1$, we have:

$$-\frac{\partial u}{\partial y} \approx -\frac{u_{i,N_y} - u_{i,N_y-2}}{2h} = 0 \rightarrow u_{i,N_y} = u_{i,N_y-2}$$

and similarly at bottom boundary where index $j = 0$, we have:

$$\frac{\partial u}{\partial y} \approx \frac{u_{i,1} - u_{i,-1}}{2h} = 0 \rightarrow u_{i,1} = u_{i,-1}$$

at right boundary where index $i = N_x - 1$, we have:

$$-\frac{\partial u}{\partial x} \approx -\frac{u_{N_x,j} - u_{N_x-2,j}}{2h} = 0 \rightarrow u_{N_x,j} = u_{N_x-2,j}$$

Based on the above treatment of Neumann boundary condition, the corresponding elements of the coefficient matrix are assigned as:

$$A_{p(i,j),p(i,j)} = 1 + 4\theta F$$

$$A_{p(N_x-1,j),p(N_x-2,j)} = A_{p(i,N_y-1),p(i,N_y-2)} = A_{p(i,0),p(i,1)} = -2\theta F$$

$$A_{p(N_x-1,j),p(N_x-1,j-1)} = A_{p(N_x-1,j),p(N_x-1,j+1)} = A_{p(i,0),p(i-1,0)} = A_{p(i,0),p(i+1,0)}$$
$$= A_{p(i,N_y-1),p(i-1,0)} = A_{p(i,N_y-1),p(i+1,0)} = -\theta F$$

$$B_{p(i,j),p(i,j)} = 1 - 4(1-\theta)F$$

$$B_{p(N_x-1,j),p(N_x-2,j)} = B_{p(i,N_y-1),p(i,N_y-2)} = B_{p(i,0),p(i,1)} = -2F(\theta - 1)$$

$$B_{p(N_x-1,j),p(N_x-1,j-1)} = B_{p(N_x-1,j),p(N_x-1,j+1)} = B_{p(i,0),p(i-1,0)} = B_{p(i,0),p(i+1,0)}$$
$$= B_{p\left(i,N_y-1\right),p(i-1,0)} = B_{p\left(i,N_y-1\right),p(i+1,0)} = -F(\theta - 1)$$

All other elements of coefficient matrix that are not mentioned are set to be 0. So we have both coefficient matrix A and B to be sparse, and it will be efficient to use sparse matrix libraries in Python to handle them.

Part 1, Explicit Solver:

By setting the implicit factor $\theta = 0$, we have an explicit solver. In this part we will verify the functionality of the code, in addition with proving the order of spatial accuracy achieved, both in the domain and on the boundaries.

Stability Analysis

For explicit solver, we have implicit factor $\theta = 0$ (which is equivalent to the forward Euler solver), and the discretized equation (6) therefore becomes:

$$u_{m,n}^{n+1} = u_{m,n}^{n}(1-4F) + F\left(u_{m-1,n}^{n} + u_{m+1,n}^{n} + u_{m,n-1}^{n} + u_{m,n+1}^{n}\right) + r_{m,n}\Delta t$$

$$= u_{m,n}^{n}\left(1-4\frac{\Delta t}{h^2}\right) + \frac{\Delta t}{h^2}\left(u_{m-1,n}^{n} + u_{m+1,n}^{n} + u_{m,n-1}^{n} + u_{m,n+1}^{n}\right)$$

$$+ r_{m,n}\Delta t \qquad (8)$$

Substituting $u_{m,n}^{n+1}$ and $u_{m,n}^{n}$ with their Fourier component $u_{m,n}^{n+1} = \tilde{u}_{n+1}e^{ikmh}e^{ilnh}$ and $u_{m,n}^{n} = \tilde{u}_{n}e^{ikmh}e^{ilnh}$ into (8), where $k = \frac{2\pi}{\lambda_x}$ and $l = \frac{2\pi}{\lambda_y}$ are the wave number in x and y direction, respectively, we obtain:

$$\frac{\tilde{u}_{n+1}}{\tilde{u}_n} = \left(1-4\frac{\Delta t}{h^2}\right) + \frac{\Delta t}{h^2}\left(e^{-ikh} + e^{ikh} + e^{-ilh} + e^{ilh}\right)$$

$$= 1 + 2\frac{\Delta t}{h^2}(cos(kh) + cos(lh) - 2) \qquad (9)$$

In the worst case when $kh = lh = \pi$, the amplification factor in (9) reaches the maximum of:

$$\left|\frac{\tilde{u}_{n+1}}{\tilde{u}_n}\right|_{max} = \left|1 - 8\frac{\Delta t}{h^2}\right| \qquad (10)$$

To ensure stability, we need to restrict the maximum amplification factor to be less than 1, which in turns produces the stability criteria:

$$\left|\frac{\tilde{u}_{n+1}}{\tilde{u}_n}\right|_{max} = \left|1 - 8\frac{\Delta t}{h^2}\right| \leq 1 \Rightarrow 8\frac{\Delta t}{h^2} \leq 2 \Rightarrow \Delta t \leq \frac{h^2}{4} \qquad (11)$$

Therefore, in the case $\Delta x = \Delta y = h$, Von Neumann stability analysis requires $\Delta t \leq \frac{h^2}{4}$ as shown in (11).

One benefit of using explicit integrator with accuracy of order higher than one is that, if keeping the same discretization size $\Delta x = \Delta y = h$ time step Δt, then at certain time moment $t > 0$ we could find that the solution from high-order accuracy integrator will be much closer to the true solution, compared with the one from 1st-order accuracy integrator. If the solution reaches steady-state as integration time increases, then after the same steps of integration, solution from high-order accuracy integrator will be closer to the true steady-state solution, compared with solution from 1st-order accuracy integrator, given the same discretization size h and Δt.

Solver Code Qualitative Verification

To check whether the solution from such a diffusion solver evolves in the expected way in the domain, it is good to run the code for few time steps under particular geometry and boundary conditions, then compare the numerical solution with an analytical solution, to see whether the solutions look alike or not (functionality or qualitative check). After this stage we could move to order of accuracy check. In this case, we assume zero source everywhere $r(x,y) = 0$, and adopt the analytical solution of $u(x,y,t)$ satisfying (1) with initial condition $u(x,y,0) = f(x,y)$, under zero Dirichlet boundary condition at all four boundaries, to be [1]:

$$u(x,y,t) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} C_{mn} sin\left(\frac{m\pi}{a}x\right) sin\left(\frac{n\pi}{b}y\right) e^{-t\left[\left(\frac{m\pi}{a}x\right)^2+\left(\frac{n\pi}{b}y\right)^2\right]} \quad (12)$$

where

$$C_{mn} = \frac{4}{ab}\int_0^a\int_0^b f(x,y) sin\left(\frac{m\pi}{a}x\right) sin\left(\frac{n\pi}{b}y\right) dydx \quad (13)$$

and a, b are the length of the boundary in x and y direction (so in this project $a = b = 4$ as shown in geometry Figure 1). For verification purpose, we set the initial condition to be:

$$u(x,y,0) = f(x,y) = \begin{cases} 1 & \text{for } (x,y) \in [1.6, 2.4] \times [1.6, 2.4] \\ 0 & \text{elsewhere} \end{cases} \quad (14)$$

Which looks like:

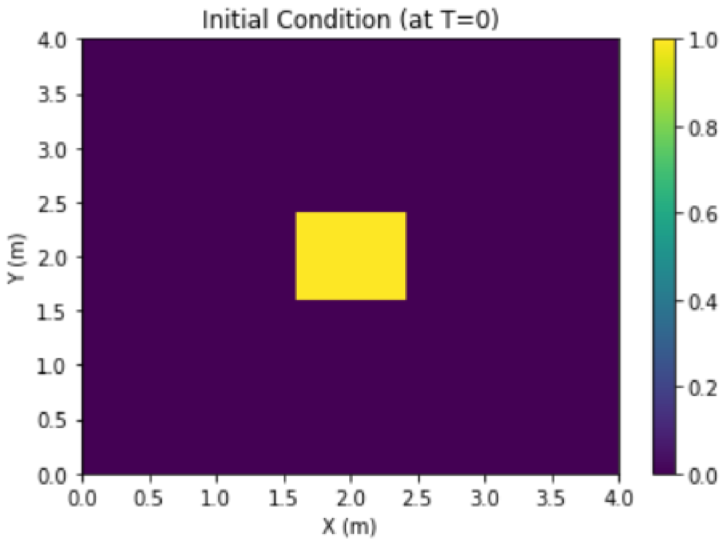

Figure 3. Initial Condition for Code Validation

Based on the domain size and the initial condition specified in (14), the constant C_{mn} becomes, as calculated from (13):

$$C_{mn} = \frac{4}{mn\pi^2}\left[cos\left(\frac{2m\pi}{5}\right) - cos\left(\frac{3m\pi}{5}\right)\right]\left[cos\left(\frac{2n\pi}{5}\right) - cos\left(\frac{3n\pi}{5}\right)\right] \quad (15)$$

The domain discretization parameters are set to be: $N_x = N_y = 201$ so that $\Delta x = \Delta y = h = 0.02m$, the time increment is set, based on stability criteria derived, as $\Delta t = \frac{h^2}{4} = 0.0001s$. We compare the numerical solution after integrating 1.1s (which corresponds to 11000 steps) with the analytical solution at 1.1s. For analytical calculation, both dummy indexes m and n are summed up to 9, since the first few terms should dominate due to $e^{-t\left[\left(\frac{m\pi}{a}x\right)^2+\left(\frac{n\pi}{b}y\right)^2\right]}$ factor as shown in (12).

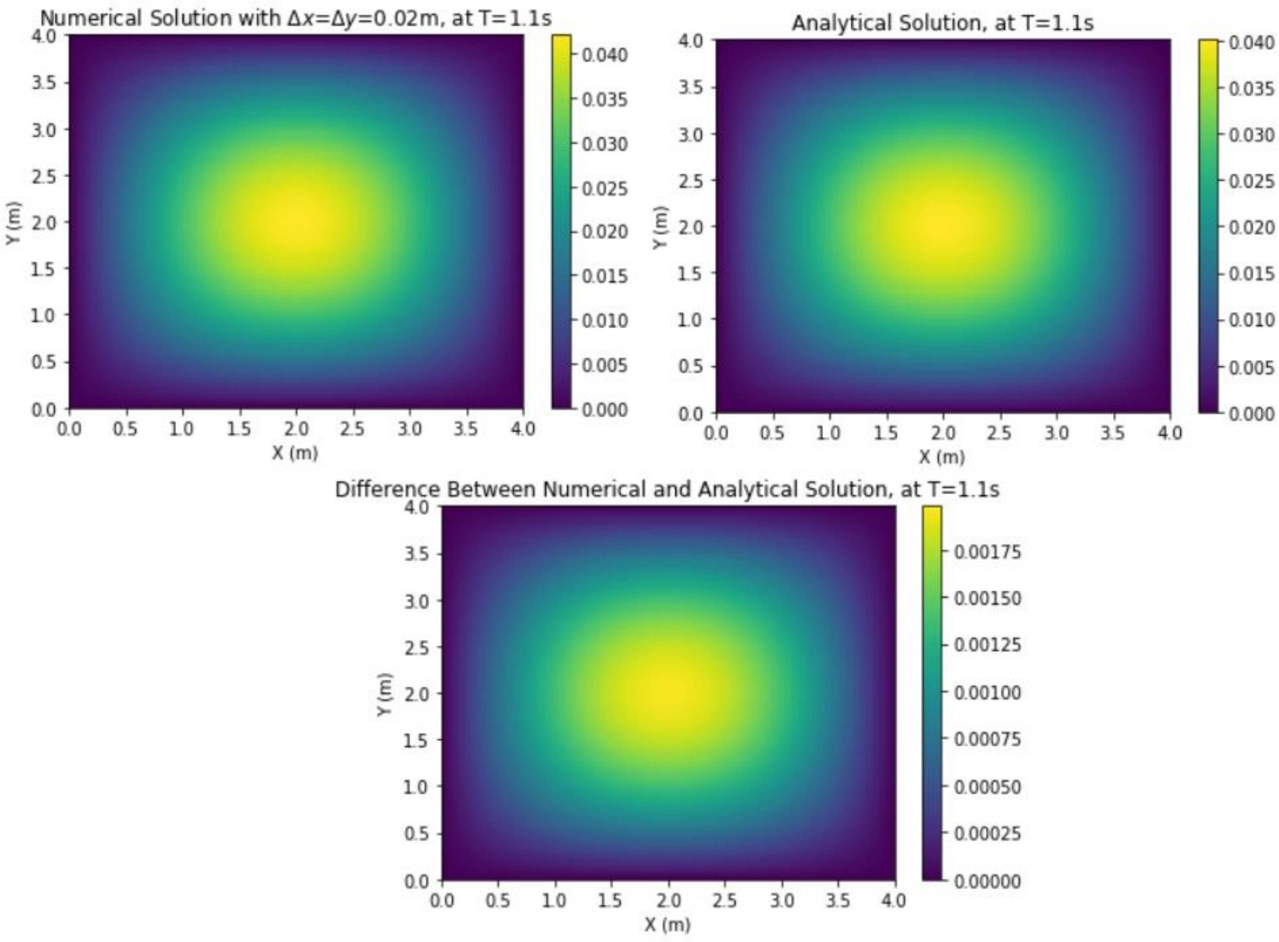

Figure 4. Comparison Between Numerical and Analytical Solution for Code Validation

From Figure 4 we can see that the numerical solution in domain generally matches the analytical solution at $t = 1.1s$, and the difference between the solutions in the domain is small. So the time-evolution of the solution obtained from the solver code generally matches the expectation.

Spatial Accuracy Verification

To verify the order of spatial accuracy achieved in domain and on boundary, we consider directly solving the Poisson's equation (and this is equivalent as solving for the steady-state solution):

$$0 = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + r(x, y) \qquad (16)$$

Numerically this is equivalent as manipulating the original linear system in (7) by setting $\boldsymbol{u}^{n+1} = \boldsymbol{u}^n$ (this makes sense physically as in the steady-state, the solution does not evolve with time everywhere anymore so that the time derivative is vanishing), so the linear system for the Poisson's equation (or the steady-state heat equation) looks like:

$$-\boldsymbol{r} = (B - A)\boldsymbol{u} := A_{ss}\boldsymbol{u} \qquad (17)$$

where matrix A and B are still the coefficient matrix shown in (7), and $B - A$ just becomes the Laplacian operator matrix, defined here as A_{ss}. The construct of Laplacian operator matrix A_{ss} will be discussed later in steady-state section. The boundary setup for this verification step is shown below:

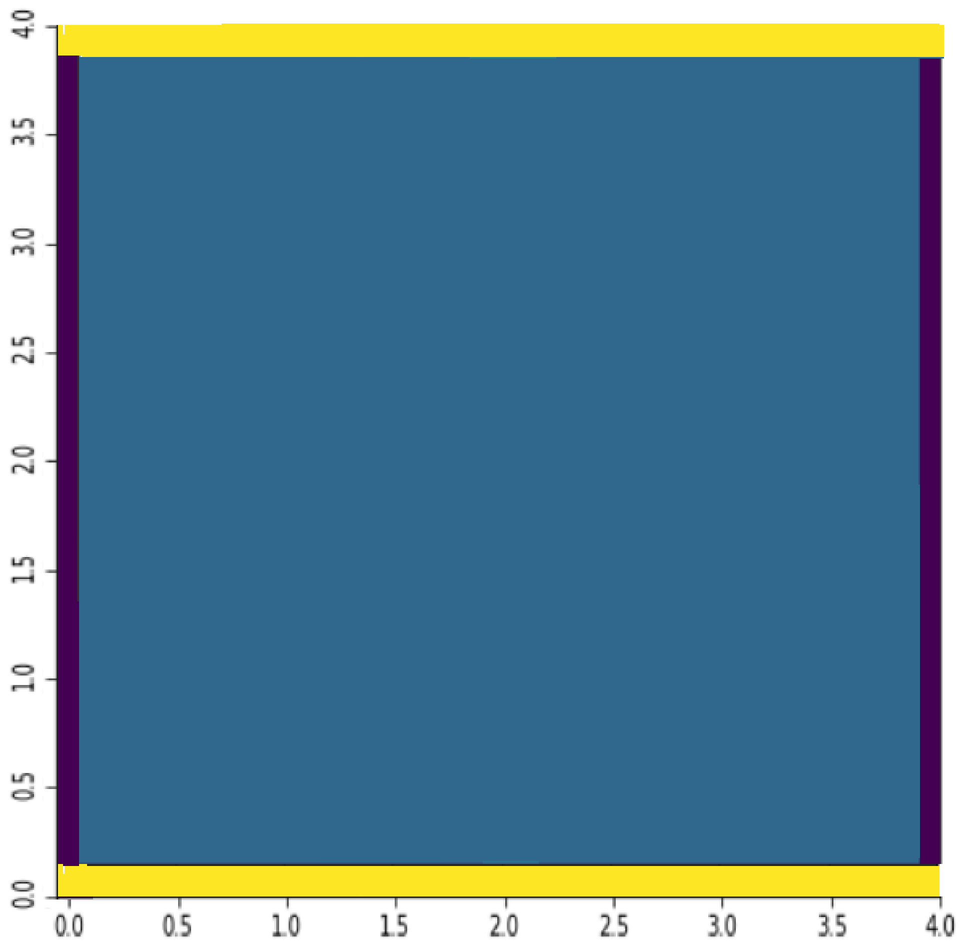

Figure 5. Boundary Setup for Spatial Accuracy Verification

which means we set Dirichlet boundary condition at top and bottom boundaries while Neumann boundary condition at left and right boundaries. In this case we use the Method of Manufactured Solution (MMS), coming up with the analytical solution that satisfies the boundary conditions as specified in Figure 5, then figuring out the required source distribution so that the Poisson's equation in (16) is satisfied. Analysis between the manufactured analytic solution and the numerical solution obtained from solving (17) will be performed to demonstrate the spatial accuracy of the solver.

Based on the boundary set-up shown in Figure 5, we could obtain an analytical solution that satisfies such boundary condition to be:

$$u(x, y) = cos\left(\frac{\pi x}{4}\right) sin\left(\frac{\pi y}{4}\right) \qquad (18)$$

Plugging (18) back to (16), we find the required source distribution set-up needs to be:

$$r(x, y) = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \ = \frac{\pi^2}{8} cos\left(\frac{\pi x}{4}\right) sin\left(\frac{\pi y}{4}\right) \qquad (19)$$

The 2D plots of analytical solution and numerical solution obtained from solving (17) with $N_x = N_y = 201$ so that $\Delta x = \Delta y = h = 0.02m$ are attached below:

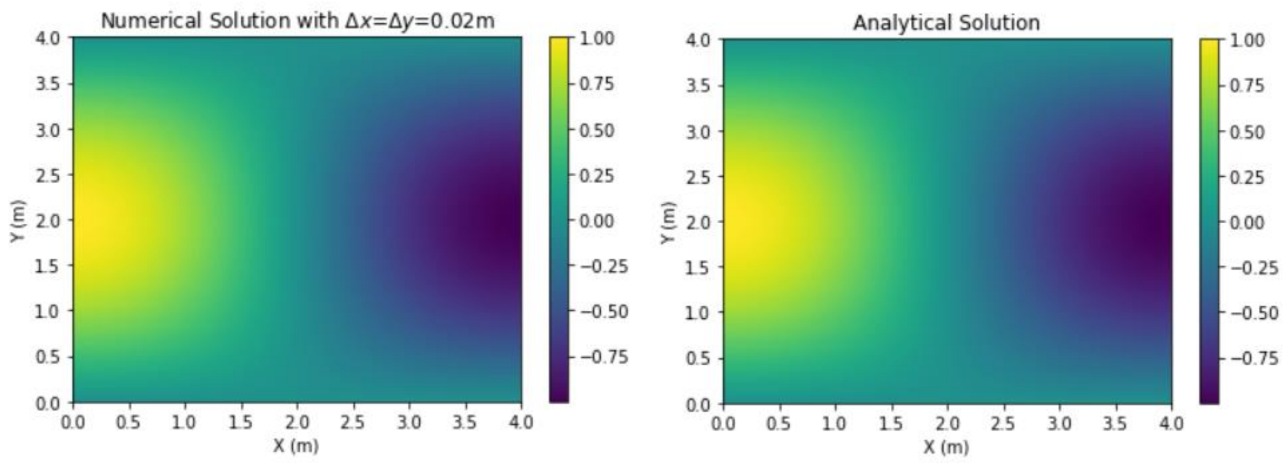

Figure 6. 2D Numerical Solution vs. Analytical Solution, for Spatial Accuracy Verification

Next we try different spatial discretization $\Delta x = \Delta y = h$ and investigate the associated maximum domain error and boundary error. In this case, the maximum domain and boundary error are calculated as:

$$Maximum\ Domain\ Error = max\left(\boldsymbol{u}_{numeric} - \boldsymbol{u}_{analytic}\right)_{domain}$$

$$Maximum\ Boundary\ Error = max\left(\boldsymbol{u}_{numeric} - \boldsymbol{u}_{analytic}\right)_{boundary}$$

We experimented several discretization sizes, and the resulting errors are summarized in the table below:

$\Delta x = \Delta y = h$	Maximum Domain Error	Maximum Dirichlet Boundary Error	Maximum Neumann Boundary Error
0.005	1.2850989014e-06	1.5774654529e-12	2.5702086562e-06
0.01	5.1402770642e-06	8.2635484633e-12	1.0280869085e-05
0.02	2.0559392968e-05	4.6026042314e-11	4.1123858897e-05
0.04	8.2210176421e-05	2.7606995032e-10	0.0001645015244

Table 1. Error Summary with Different Spatial Discretization Sizes

Finally, we make the log-log plot on Error vs. spatial discretization size h, based on the error data summarized in Table 1, for both domain error and two types of boundary error, attached below:

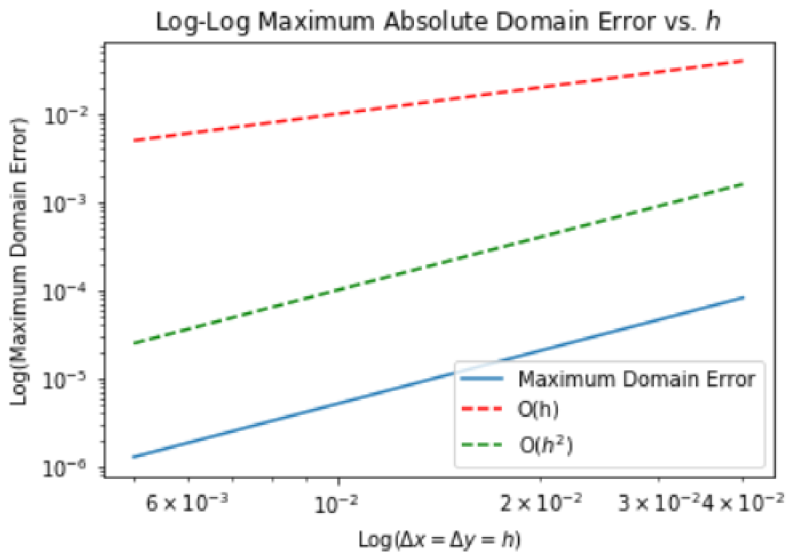

Figure 7. Log-Log Plot of Domain Error, for Spatial Accuracy Verification

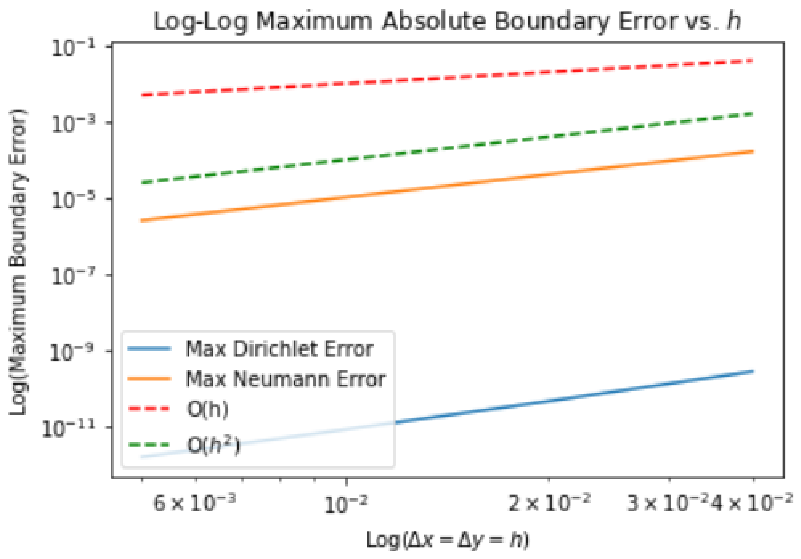

Figure 8. Log-Log Plot of Boundary Error, for Spatial Accuracy Verification

So from Figure 7 and 8, we can see that both the domain error and two types of boundary error are parallel to $O(h^2)$, which means that both the domain solution and boundary solution reaches 2nd-order accuracy in space, as we expect (recall that we use the central difference to approximate the spatial derivatives, which is 2nd-order approximation).

Explicit Integration Result

The numerical result obtained from explicit integrating up to time $T = 30 \cdot max(s_x, s_y) = 120s$, using Forward Euler (setting implicit factor $\theta = 0$), is shown in the figure below:

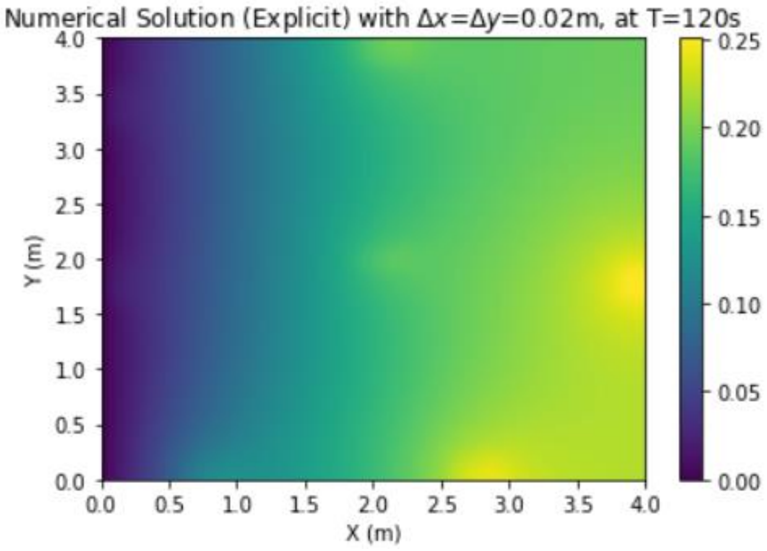

Figure 9. Numerical Solution at T=120s, with Explicit Integrator (Forward Euler)

Part 2, Implicit Solver + Steady-State Solution:

For implicit scheme, we choose the Crank-Nicolson scheme, which has 2nd-order time accuracy, by setting the implicit factor $\theta = \frac{1}{2}$.

Implicit Integration Result

The numerical result obtained from implicit integrating up to time $T = 30 \cdot max(s_x, s_y) = 120s$, using Crank-Nicolson scheme, is shown in the figure below:

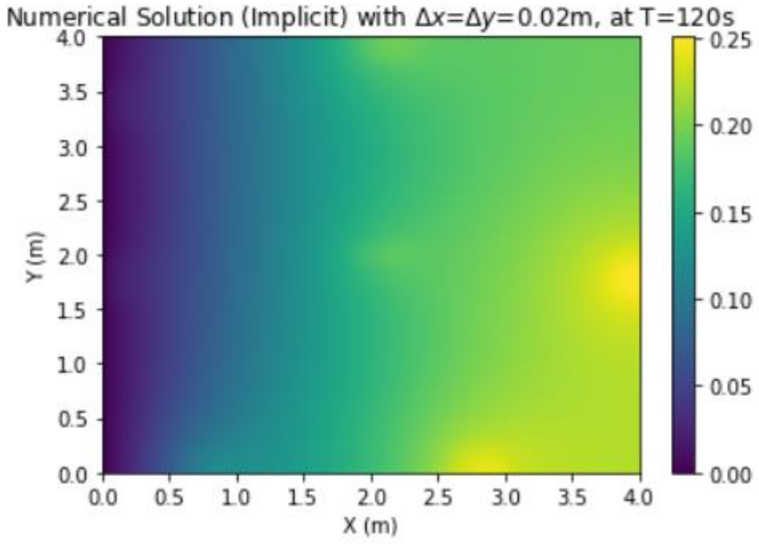

Figure 10. Numerical Solution at T=120s, with Implicit Integrator (Crank-Nicolson)

Construction of Laplacian Operator Matrix

For steady-state solution, the solution does not evolve with time everywhere anymore, so that the time derivative is vanishing, and the heat equation becomes the Poisson's equation shown in (16). If we discretize the Poisson's equation in (16) using central difference, we have (still keeping $\Delta x = \Delta y = h$):

$$-r_{i,j} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2}$$

$$= \frac{-4}{h^2} u_{i,j} + \frac{1}{h^2}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}\right) \qquad (20)$$

So the discretized equation (20) represents the steady-state linear system illustrated in (17):

$$-\boldsymbol{r} = A_{ss}\boldsymbol{u}$$

where $A_{ss} \in \boldsymbol{R}_{N_yN_x \times N_yN_x}$ denotes the Laplacian operator matrix. The construction of A_{ss}, based on the geometry setup and associated boundary conditions, is showed as follows:

Laplacian Operator Matrix on Domain

For grid points on domain: $(i,j)_{domain} \in [1, N_x - 2] \times [1, N_y - 2]$, the full version of (20) holds true, so the corresponding elements of Laplacian operator matrix are assigned as:

$$A_{ss_{p(i,j),p(i,j)}} = \frac{-4}{h^2}$$

$$A_{ss_{p(i,j),p(i-1,j)}} = A_{ss_{p(i,j),p(i+1,j)}} = A_{ss_{p(i,j),p(i,j-1)}} = A_{ss_{p(i,j),p(i,j+1)}} = \frac{1}{h^2}$$

Laplacian Operator Matrix on Window (Dirichlet BC)

For grid points on window: $(i,j)_{window} \in [0,0] \times [0, N_y - 1]$, we enforce zero Dirichlet boundary condition, so the corresponding element of Laplacian operator matrix is assigned as:

$$A_{ss_{p(i,j),p(i,j)}} = 1$$

Laplacian Operator Matrix on Wall (Neumann BC)

For grid points on wall $(i,j)_{wall} \in [1, N_x - 1] \times [0,0] \cup [1, N_x - 1] \times [N_y - 1, N_y - 1] \cup [N_x - 1, N_x - 1] \times [0, N_y - 1]$, we enforce Neumann boundary condition. Again to reach second-order accuracy in space, <u>we will apply central difference to approximate the first-order derivative in Neumann boundary condition</u>. Under this 2nd-order accuracy treatment on spatial derivative in Neumann boundary condition, the corresponding elements of Laplacian operator matrix are assigned as:

$$A_{ss_{p(i,j),p(i,j)}} = \frac{-4}{h^2}$$

$$A_{ss_{p(N_x-1,j),p(N_x-2,j)}} = A_{ss_{p(i,N_y-1),p(i,N_y-2)}} = A_{ss_{p(i,0),p(i,1)}} = \frac{2}{h^2}$$

$$A_{ss_{p(N_x-1,j),p(N_x-1,j-1)}} = A_{ss_{p(N_x-1,j),p(N_x-1,j+1)}} = A_{ss_{p(i,0),p(i-1,0)}}$$

$$= A_{ss_{p(i,0),p(i+1,0)}} = A_{ss_{p\left(i,N_y-1\right),p(i-1,0)}} = A_{ss_{p\left(i,N_y-1\right),p(i+1,0)}} = \frac{1}{h^2}$$

All other elements of Laplacian operator matrix that are not mentioned are set to be 0. So again we have A_{ss} to be large sparse matrix, which will be handled with sparse matrix library in Python.

Steady-State Solution

The steady-state solution obtained from directly solving (17), with the source distribution and geometry setup following Figure 1 and 2, is shown:

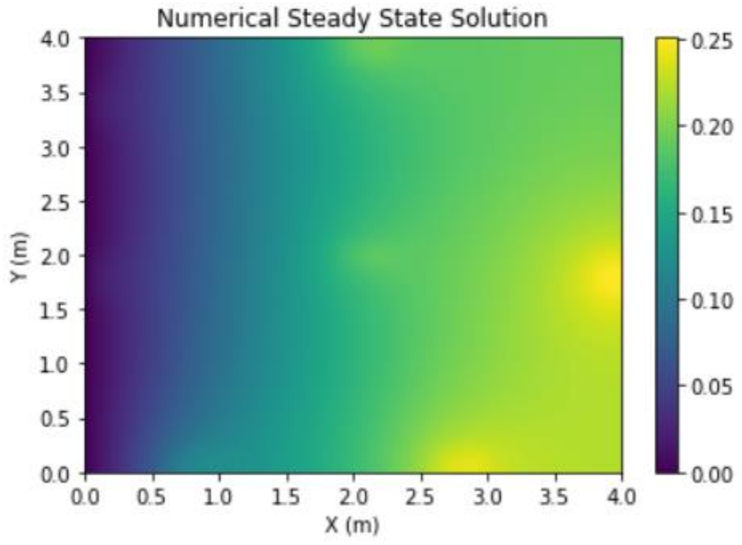

Figure 11. Numerical Steady-State Solution, from Directly Solving Poisson’s Equation

Jacobi Iteration for Steady-State Solution

Jacobi iteration method is also applied for solving the Poisson’s equation, following the iteration formula:

$$D_{ss}\boldsymbol{u}_{k+1} = -\boldsymbol{r} - R_{ss}\boldsymbol{u}_k \quad (21)$$

Where D_{ss} is the diagonal matrix of Laplacian operator matrix A_{ss}, and $R_{ss} = A_{ss} - D_{ss}$. We run 300 iterations of Jacobi iteration starting from a random initial guess, and obtain the following result:

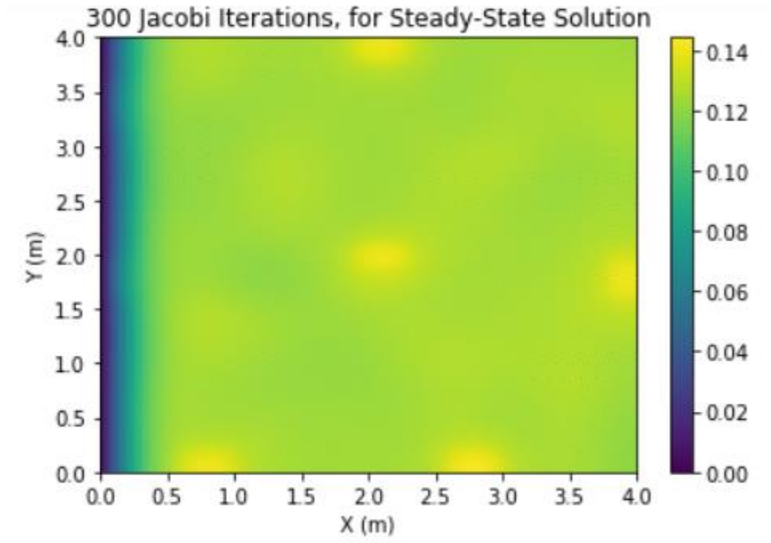

Figure 12. Steady-State Solution after 300 Jacobi Iterations

We observe that the solution from 300 Jacobi iterations, as shown in Figure 12, does not resembles the steady-state solution in Figure 11, which means that after 300 Jacobi iterations the solution has not converged to the desired solution yet. The reason is that the Jacobi iteration, as given in (21), could be considered as an explicit time integrator (with index k representing the time index). It may not be able to converge to the steady-state solution under just 300 iteration steps, as we tested in the explicit integrator part. In fact, the convergence rate is related to the eigenvalues of the Laplacian operator matrix A_{ss}, and the larger the size of A_{ss} (corresponding to larger discretization points N_x and N_y, the slower convergence rate will be (more iterations are required to converge).

Damped Jacobi Iteration for Steady-State Solution

Using the damped Jacobi iteration given as:

$$\boldsymbol{u}_{k+1} = \alpha \boldsymbol{u}_k + (1-\alpha)\widetilde{\boldsymbol{u}}_{k+1} \qquad (22)$$

where $\widetilde{\boldsymbol{u}}_{k+1}$ denotes the next-step iteration result by Jacobi, after 300 iterations (with $\alpha = 0.05$) we have:

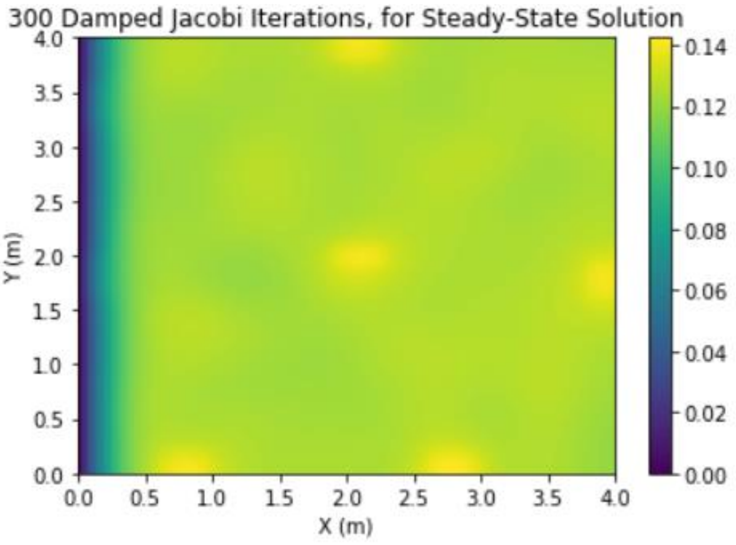

Figure 13. Steady-State Solution after 300 Damped Jacobi Iterations

From the Damped Jacobi iteration result, we see smoothing behavior on the solution. This is because the damped Jacobi iteration, as given in (22), produces weighted averaged between two consecutive iteration results from direct Jacobi iteration, thus has smoother error than direct Jacobi iteration.

Part 3, Geometric Multigrid:

In this project, we consider a multigrid V-cycle solver for the steady-state solution, based on a two-level grid coarsening by a factor of two: The finer grid has spatial step size of h with number of grid points N_h on each dimension, and the coarser grid has spatial step size of $2h$ with number of grid points N_{2h} on each dimension. So we will be solving the discretized Poisson's equation within each grid system:

$$-\boldsymbol{r}_h = A_{ss(h)}\boldsymbol{u}_h \qquad (23)$$

$$-\boldsymbol{r}_{2h} = A_{ss(2h)}\boldsymbol{u}_{2h} \qquad (24)$$

The transformation between fine grid vector and coarser grid vector follows:

$$\boldsymbol{u}_{2h} = R\boldsymbol{u}_h$$

$$\boldsymbol{r}_{2h} = R\boldsymbol{r}_h$$

$$\boldsymbol{u}_h = P\boldsymbol{u}_{2h}$$

$$\boldsymbol{r}_h = P\boldsymbol{r}_{2h}$$

where $R \in \boldsymbol{R}_{{N_{2h}}^2 \times {N_h}^2}$ and $P \in \boldsymbol{R}_{{N_h}^2 \times {N_{2h}}^2}$ are the restriction matrix and prolongation matrix, respectively. The transformation of the system matrix then follows:

$$A_{ss(2h)} = RA_{ss(h)}P \qquad (25)$$

Restriction and Prolongation

The restriction is implemented in a straightforward way, by simply picking values from the fine grid at certain grid points. So for $(i,j) \in [0, N_{2h}] \times [0, N_{2h}]$, the restriction looks like:

$$u_{i,j}^{2h} = u_{2i,2j}^{h} \qquad (26)$$

And (26) is used for constructing the restriction matrix R. For prolongation, we use bilinear interpolation from coarse to fine, which has 2nd-order accuracy in space. For $(i,j) \in [0, N_{2h}] \times [0, N_{2h}]$, the prolongation follows:

$$u_{2i,2j}^{h} = u_{i,j}^{2h} \qquad (27)$$

$$u_{2i+1,2j}^{h} = \frac{u_{i,j}^{2h} + u_{i+1,j}^{2h}}{2} \qquad (28)$$

$$u_{2i,2j+1}^{h} = \frac{u_{i,j}^{2h} + u_{i,j+1}^{2h}}{2} \qquad (29)$$

$$u_{2i+1,2j+1}^{h} = \frac{u_{i,j}^{2h} + u_{i+1,j}^{2h} + u_{i,j+1}^{2h} + u_{i+1,j+1}^{2h}}{4} \qquad (30)$$

And the combination of (27), (28), (29) and (30) is used to construct the prolongation matrix P. Notice that both restriction matrix R and prolongation matrix P are sparse matrix, so they are constructed and handled with sparse libraries in Python.

Spatial Accuracy of Restriction-Prolongation Process

There is no approximation in restriction process, as it is just picking values on fine grid at certain grid points and mapping it down to coarse grid at corresponding grid points, in this case. However, the prolongation process has approximation, as it interpolates values between grid points on the coarse grid. To investigate the spatial accuracy obtained by the restriction-prolongation process, we try to restrict-prolongate a discrete 2D function:

$$u(x,y) = cos\left(\frac{\pi x}{4}\right) sin\left(\frac{\pi y}{4}\right)$$

which is (18) showed before, and it satisfies the boundary condition set up by the geometry illustrated in Figure 1. We change the grid size of fine grid (denoted as h), perform the restriction-prolongation operation, and compare the result with the original setup $u(x,y) = cos\left(\frac{\pi x}{4}\right) sin\left(\frac{\pi y}{4}\right)$. The maximum error between the result after restriction-prolongation process and the initial setup, under different fine grid discretization size h, is summarized in the table below:

$\Delta x = \Delta y = h$	Maximum Error
0.005	1.5420939792e-05
0.01	6.1679954283e-05
0.02	0.000246658946
0.04	0.000985662336

Table 2. Restriction-Prolong Error Summary with Different Spatial Discretization Sizes

Based on the error summary shown in Table 2, we make the log-log plot on the maximum error versus fine grid discretization size, illustrated below:

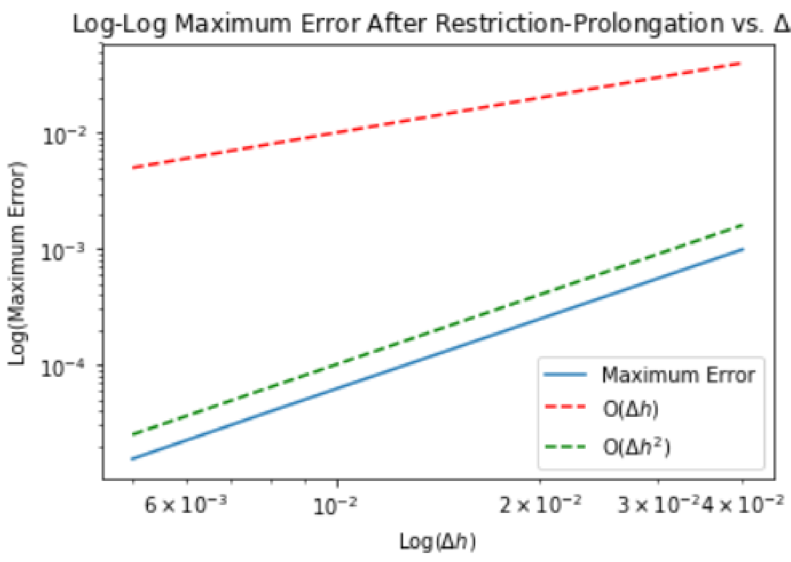

Figure 14. Lot-Lot Plot of Maximum Error after Restriction-Prolongation Process

From Figure 14, we can see that the maximum restriction-prolongation has a spatial accuracy of 2nd order. This is what we expect, because we are using bilinear interpolation for prolongation process, which is 2nd-order accuracy interpolation.

Multigrid V-Cycle

Finally, we implemented the V-cycle to solve for the steady-state solution, using the damped Jacobi smoother and restriction and prolongation matrix (at different grid levels) discussed before. The idea of the implementation of our two-level multigrid V-cycle looks like below:

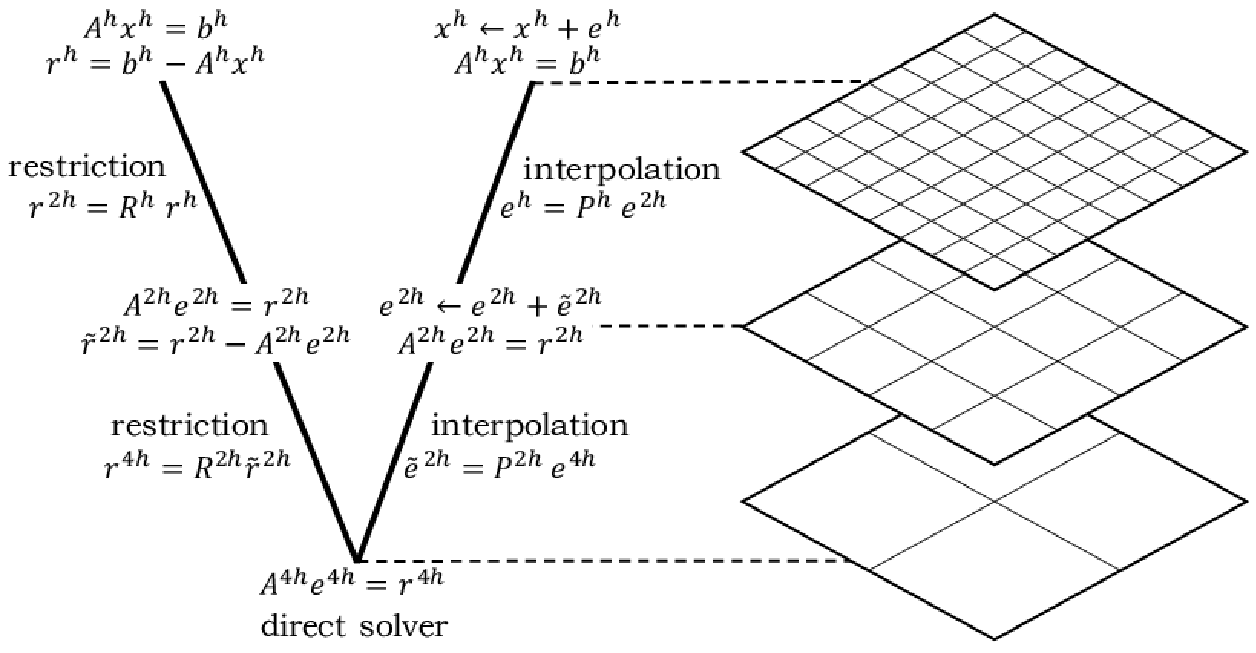

Figure 15. Illustration of Two-Level Multigrid V-Cycle [2]

except that we perform three Jacobi smoothing on $A^h x^h = b^h$ before calculating the residual r^h, and perform three Jacobi smoothing on $A^h x^h = b^h$ after correction obtained from finer grid levels. The steady-state solution obtained after 100 iterations of V-cycle is attached below:

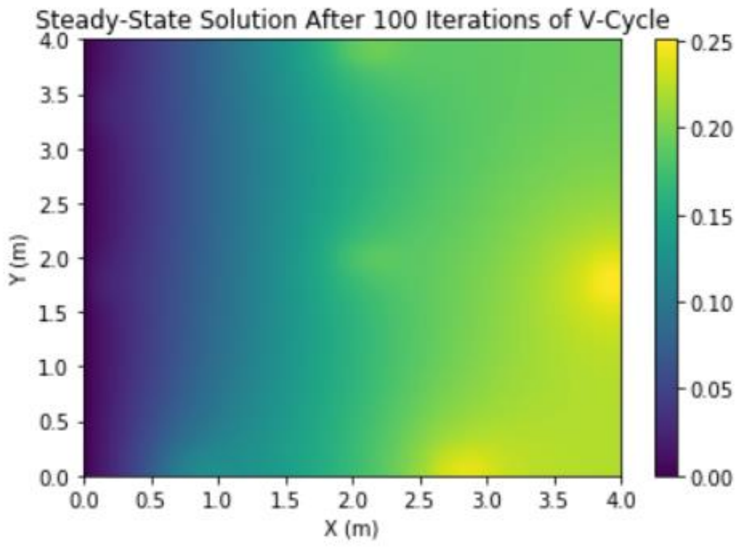

Figure 16. Steady-State Solution Obtained after 100 Iterations of V-Cycle

Finally, we record the maximum residual after each iteration of V-cycle, and make the semi-log plot of the maximum residual against the iteration count of V-cycle, attached below:

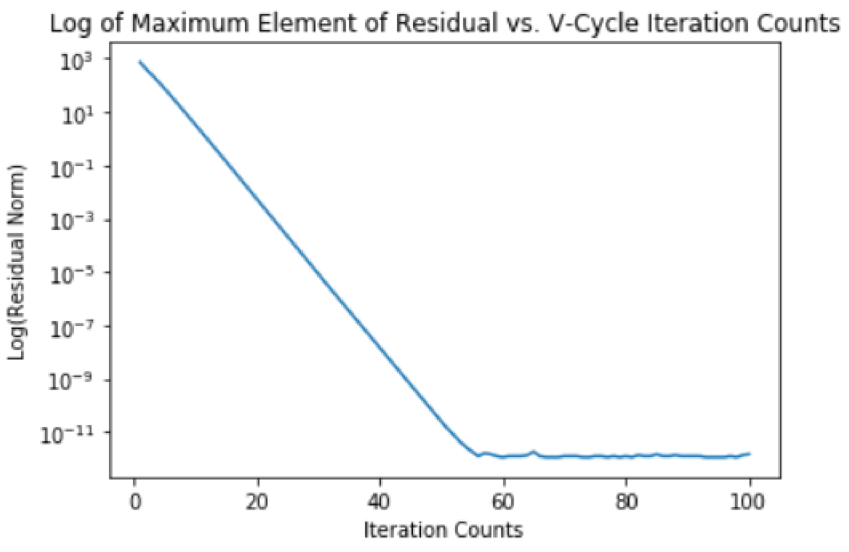

Figure 17. Semi-Log Plot of Maximum Residual vs. V-Cycle Iteration Count

So from figure 17, we observe that the maximum residual decreases rapidly around the beginning few iterations of V-cycle. After 60 iterations of V-cycle, the residual does not change much as we iterate more, and the solution converges (to the one shown in Figure 16).

Conclusion:

In this project, we constructed and tested a 2-D heat equation solver, under rectangular domain and mixed boundary conditions (Homogeneous Dirichlet and Neumann boundary conditions), using finite difference method. First, we built an explicit integrator, using Forward Euler and central spatial difference scheme, which has 2nd-order accuracy in space, imposed on both the boundaries and inner domain. Then we validated the numerical results from the code by comparing it with an analytical solution of 2-D heat equation. The spatial accuracy test has also been performed to verify that 2nd-order spatial accuracy is achieved on both inner domain and boundaries. After accuracy verification, we ran explicit integrator, constructed with Forward Euler and central spatial difference scheme (which is FTCS), to the steady-state condition.

Next, we moved to implicit solver, by constructing the associated linear systems representing the discretized heat equation. Special treatment on the coefficient matrix and Laplacian operator matrix handling the boundary conditions were introduced. Then we ran the implicit integrator, with Crank-Nicolson scheme, to the steady-state condition. After implicit solver, we directly focused on the steady-state solution. We implemented iteration (in this case, Jacobi and damped Jacobi iterations) method to solve for the steady-state solution, using the Laplacian operator matrix that handles the boundary conditions, constructed in previous part. However, we found that iteration method converges slowly if only iterating on the same grid level, which invokes us to try geometric multigrid method. In the iteration method part, we constructed Jacobi smoother using damped Jacobi iteration, which will be adopted as the iteration smoother in multigrid method.

Finally, we built a geometric multigrid solver, in which the V-cycle is implemented. In this case, we made a two-level grid coarsening by a factor of two. Transferring vectors between coarse grid and fine grid (and vice versa) requires prolongation and restriction, and the associated prolongation and restriction matrix were discussed. We showed that the restriction-prolongation process reaches 2nd-order accuracy in space (which is expected, since bilinear interpolation is adopted). After accuracy verification, we implemented V-cycle (with the damped Jacobi smoother) to solve for the steady-state solution. We found that the norm of residual in the fine grid drops drastically as we iterate the V-cycle, and the solution converges to the steady-state solution after around 60 iterations of V-cycle, at which the residual norm on the fine grid drops down to order of 10^{-11}.

Reference

[1] Ryan C. Daileda. *The Two Dimensional Heat Equation.* Trinity University, 2012. http://ramanujan.math.trinity.edu/rdaileda/teach/s12/m3357/lectures/lecture_3_6_short.pdf.

[2] Ibeid, Huda & Olson, Luke & Gropp, William. (2018). *FFT, FMM, and Multigrid on the Road to Exascale: performance challenges and opportunities*.