CS 555 Project Report

Danny Paez

April 22, 2020

Part 1

For this project, we will be constructing a 2D finite-difference implementation for the initial boundary value problem on the following domain:

$$\begin{aligned} u_t &= \Delta u + r(x) & \\ u(x,0) &= 0 & (x \in \Omega) \\ u(x,t) &= 0 & (x \in \Gamma_W) \\ \hat{n} \cdot \nabla u(x,t) &= 0 & (x \in \partial\Omega \setminus \Gamma_W) \end{aligned}$$

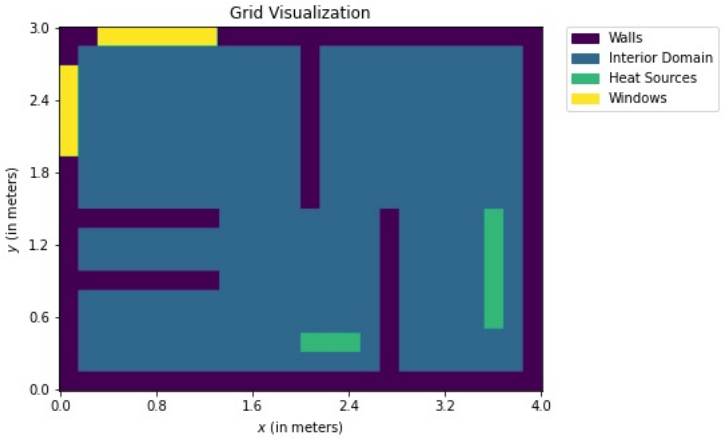

Figure 1. Room geometry on a 3×4 meter room

where Ω denotes the blue area, $\partial\Omega$ denotes the purple and yellow area, Γ_W denotes the yellow area, $r(x)$ denotes a time-constant function corresponding to heat sources (green area).

We first need to create a 2D spatial discretization of the entire domain. We pick a uniform mesh size h for both spatial directions and construct our 2D grid. Using second-order centered differences for u_{xx} and u_{yy}, we construct our Laplacian matrix:

$$\frac{\partial}{\partial t} u_{i,j} = \frac{1}{h^2}\left(u^t_{i,j+1} + u^t_{i,j-1} - 4u^t_{i,j} + u^t_{i+1,j} + u^t_{i-1,j}\right) \tag{1}$$

where subscript i denotes the y-dimension and subscript j denotes the x-dimension. From (1), we can see that our construction will run into a few issues:

- Points adjacent to walls.
- Window points.

Gridpoints that lie adjacent to walls will have their stencils touch gridpoints that are not considered to be degrees of freedom. However, we can use the Neumann boundary condition on the walls to solve this issue. For a gridpoint that hits a wall when traveling north/south, we set $u_x = 0$. For a gridpoint that hits a wall when traveling west/east, we set $u_y = 0$. We approximate both partial derivatives from a second-order centered difference:

$$u_x \approx \frac{u_{i-1,j} - u_{i+1,j}}{2h} = 0 \iff u_{i-1,j} = u_{i+1,j} \tag{2}$$

$$u_y \approx \frac{u_{i,j-1} - u_{i,j+1}}{2h} = 0 \iff u_{i,j-1} = u_{i,j+1} \tag{3}$$

Implementing (2) and (3) in (1), we obtain the following additional equations:

$$\frac{\partial}{\partial t}u_{i,j} = \frac{1}{h^2}\left(u^t_{i,j+1} + u^t_{i,j-1} - 4u^t_{i,j} + 2u^t_{i+1,j}\right) \tag{4}$$

$$\frac{\partial}{\partial t}u_{i,j} = \frac{1}{h^2}\left(u^t_{i,j+1} + u^t_{i,j-1} - 4u^t_{i,j} + 2u^t_{i-1,j}\right) \tag{5}$$

$$\frac{\partial}{\partial t}u_{i,j} = \frac{1}{h^2}\left(2u^t_{i,j+1} + u^t_{i+1,j} - 4u^t_{i,j} + u^t_{i-1,j}\right) \tag{6}$$

$$\frac{\partial}{\partial t}u_{i,j} = \frac{1}{h^2}\left(2u^t_{i,j-1} + u^t_{i+1,j} - 4u^t_{i,j} + u^t_{i-1,j}\right) \tag{7}$$

where (4) is for gridpoints that hit a wall when traveling north, (5) is for gridpoints that hit a wall when traveling south, (6) is for gridpoints that hit a wall when traveling west, and (7) is for gridpoints that hit a wall when traveling east.

For window gridpoints, we can just choose not to associate an equation with it, since it always stays at 0.

From the way we are handling these boundary conditions, I claim that we will still achieve second-order spatial accuracy. Since we are using second-order differences for u_x, and u_y, they do not degrade the accuracy of our second-order centered difference of u_{xx} and u_{yy}. Since we are effectively forcing the window gridpoints to stay at 0, it trivially becomes accurate.

To verify that our boundary condition handling achieves second-order accuracy in the ℓ^∞ norm, we will test our implementation on a few simpler problems:

$$\begin{aligned} u_t &= \Delta u \\ u(0,y,t) &= u(1,y,t) = 0 \\ u_y(0,y,t) &= u_y(1,y,t) = 0 \\ u(x,y,0) &= \sin(\pi x)\cos(\pi y) \\ (x,y) &\in [0,1]\times[0,1] \end{aligned} \tag{8}$$

where the actual solution of (8) is:

$$u(x,y,t) = \sin(\pi x)\cos(\pi y)\exp(-2t\pi^2)$$

$$\begin{aligned} u_t &= \Delta u \\ u(x,0,t) &= u(x,1,t) = 0 \\ u_x(0,y,t) &= u_x(1,y,t) = 0 \\ u(x,y,0) &= \cos(\pi x)\sin(\pi y) \\ (x,y) &\in [0,1]\times[0,1] \end{aligned} \tag{9}$$

where the actual solution of (9) is:

$$u(x,y,t) = \cos(\pi x)\sin(\pi y)\exp(-2t\pi^2)$$

(8) will test our Neumann boundary condition handling in the north/south direction and our Dirichlet boundary condition handling in the west/east direction. (9) will test our Neumann boundary condition handling in the west/east direction and our Dirichlet boundary condition handling in the north/south direction. Using our implementation on a 100×100, 200×200, 300×300, and 400×400 grid, we arrive at the following results:

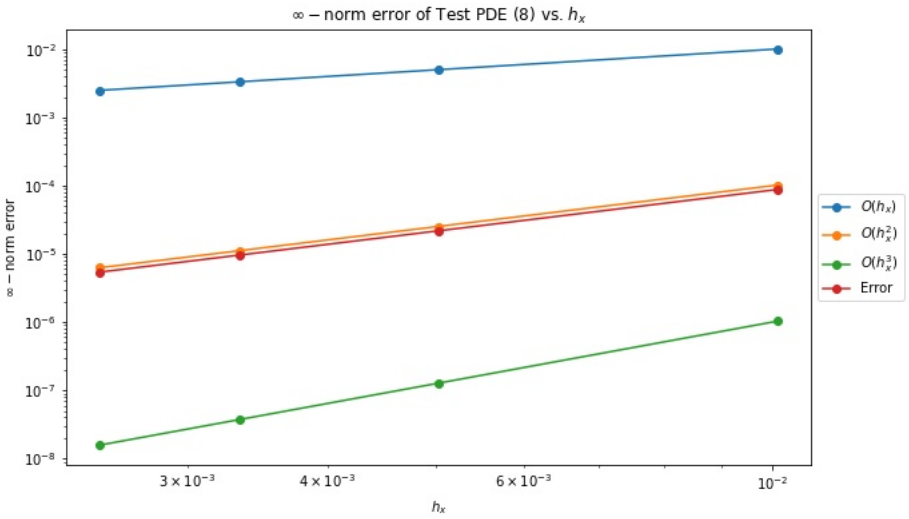

Figure 2. Error convergence plot on (8)

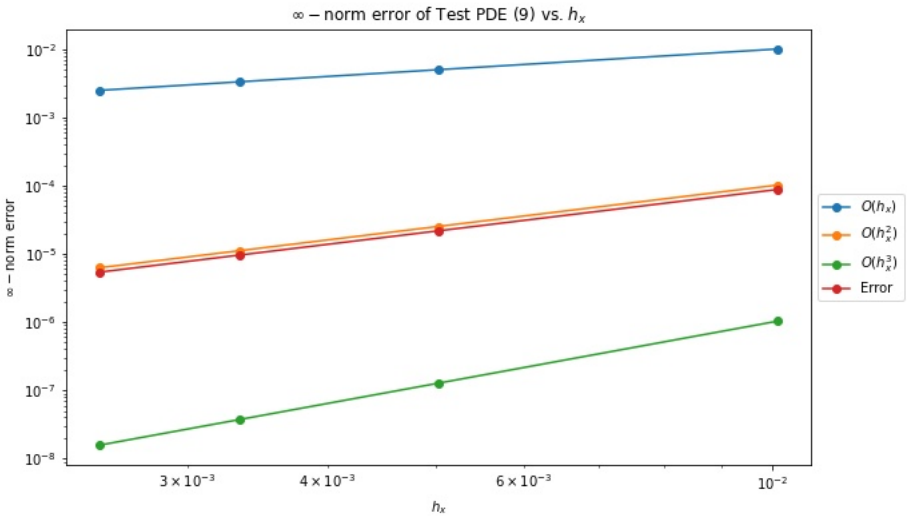

Figure 3. Error convergence plot on (9)

These plots further support my argument that our boundary condition handling achieves second-order spatial accuracy.

After we construct our Laplacian matrix, we need to decide what time integrator to use. For Part 1, we will use a Forward Euler time integrator. I claim that our time step h_t is bounded by $h^2/4$.

Proof: We will use von-Neumann stability analysis to derive the upper bound for h_t.

$$\begin{aligned}
u_t &= \Delta u \\
\frac{u_{i,j}^{t+1} - u_{i,j}^t}{h_t} &= \frac{1}{h^2} \cdot \left(u_{i-1,j}^t + u_{i+1,j}^t - 4u_{i,j}^t + u_{i,j-1}^t + u_{i,j+1}^t \right) \\
\implies u_{i,j}^{t+1} &= u_{i,j}^t + \frac{h_t}{h^2} \cdot \left(u_{i-1,j}^t + u_{i+1,j}^t - 4u_{i,j}^t + u_{i,j-1}^t + u_{i,j+1}^t \right) \\
&= u_{i,j}^t \cdot \left(1 - 4\frac{h_t}{h^2} \right) + \frac{h_t}{h^2} \cdot \left(u_{i-1,j}^t + u_{i+1,j}^t + u_{i,j-1}^t + u_{i,j+1}^t \right)
\end{aligned}$$

Taking the Fourier Transform of both sides, we get:

$$\begin{aligned}
\hat{u}_{t+1}(\varphi, \gamma) &= \hat{u}_t(\varphi, \gamma) \cdot \left(1 - 4\frac{h_t}{h^2} \right) + \hat{u}_t(\varphi, \gamma) \cdot \frac{h_t}{h^2} \cdot \left(e^{-i\varphi} + e^{i\varphi} + e^{-i\gamma} + e^{i\gamma} \right) \\
&= \hat{u}_t(\varphi, \gamma) \cdot \left(1 - 4\frac{h_t}{h^2} \right) + \hat{u}_t(\varphi, \gamma) \cdot \frac{h_t}{h^2} \cdot \left(2\cos(\varphi) + 2\cos(\gamma) \right)
\end{aligned}$$

We can see that:

$$\begin{aligned}
\hat{p}(\varphi, \gamma) &= 1 \\
\hat{q}(\varphi, \gamma) &= \left(1 - 4\frac{h_t}{h^2} \right) + \frac{h_t}{h^2} \cdot \left(2\cos(\varphi) + 2\cos(\gamma) \right) \\
\implies \hat{s}(\varphi, \gamma) = \frac{\hat{q}(\varphi, \gamma)}{\hat{p}(\varphi, \gamma)} &= \left(1 - 4\frac{h_t}{h^2} \right) + \frac{h_t}{h^2} \cdot \left(2\cos(\varphi) + 2\cos(\gamma) \right)
\end{aligned}$$

We need to satisfy $\max |\hat{s}(\varphi, \gamma)| \leq 1$. Setting $\cos(\varphi) = \cos(\varphi) = -1$ will maximize $|\hat{s}(\varphi, \gamma)|$.

$$\begin{aligned}
\max |\hat{s}(\varphi, \gamma)| &= \left| \left(1 - 4\frac{h_t}{h^2} \right) - 4\frac{h_t}{h^2} \right| \\
&= \left| 1 - 8\frac{h_t}{h^2} \right| \\
\left| 1 - 8\frac{h_t}{h^2} \right| \leq 1 &\iff -1 \leq 1 - 8\frac{h_t}{h^2} \leq 1 \\
&\implies -2 \leq -8\frac{h_t}{h^2} \leq 0 \\
&\implies 0 \leq 8\frac{h_t}{h^2} \leq 2 \\
&\implies \frac{h_t}{h^2} \leq \frac{1}{4} \\
&\implies h_t \leq \frac{h^2}{4}
\end{aligned}$$

□

Since our time step h_t is bounded by $h^2/4$, using an explicit time integrator that has an order of accuracy higher than first (say second-order) would cause our temporal accuracy to be roughly $O((h^2/4)^2) = O(h^4/16) = O(h^4)$. In this case, we will achieve fourth-order temporal accuracy while achieving second-order spatial accuracy. We may have to compute additional sparse matrix-vector multiplications in the process, but we will achieve more accuracy. Using Forward Euler, we will integrate the solution to time $T = 30 \cdot \max(s_x, s_y) = 30 \cdot \max(4, 3) = 120$ seconds. Below is the result of this computation:

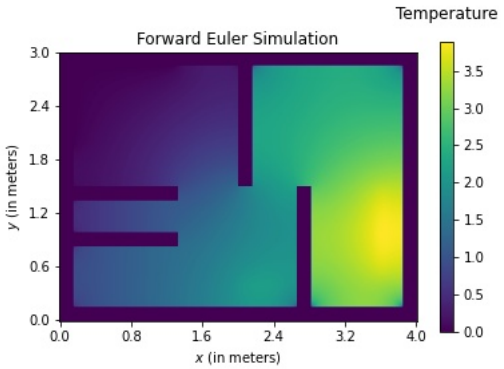

Figure 4. Forward Euler Simulation to $T = 120$ seconds.

Part 2

For this part, we will use a second-order accurate diagonally implicit time integrator. In particular, we will use Kraaijevanger and Spijker's two-stage Diagonally Implicit Runge Kutta method. Through brief experimentation, I found that $h_t \leq 5h^2/4$ yielded stable solutions. Below is a plot of the solution time-integrated to $T = 30 \cdot \max(s_x, s_y) = 30 \cdot \max(4, 3) = 120$ seconds using the mentioned time integrator:

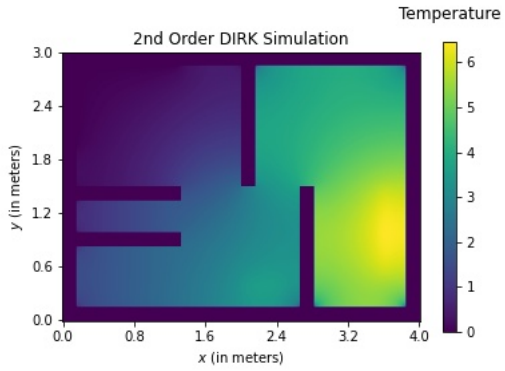

Figure 5. Second Order DIRK Simulation to $T = 120$ seconds.

From Figure 5, we can see that our solution is heading towards the steady state. Steady state occurs when $u_t = 0$. Therefore:

$$u_t = 0 \iff \Delta u + r(x) = 0 \implies -\Delta u = r(x)$$

From the above, we can directly solve for the steady state solution. Below is the result of the above computation:

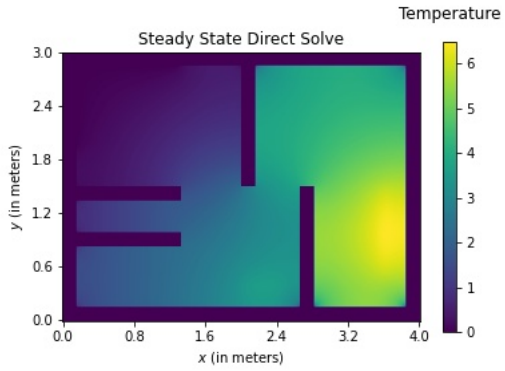

Figure 6. Direct steady state solution.

In addition, we will also compute the steady state solution via the Jacobi method. We will run the method for 300 iterations from a random vector of numbers as the initial guess. Below is the result of this computation:

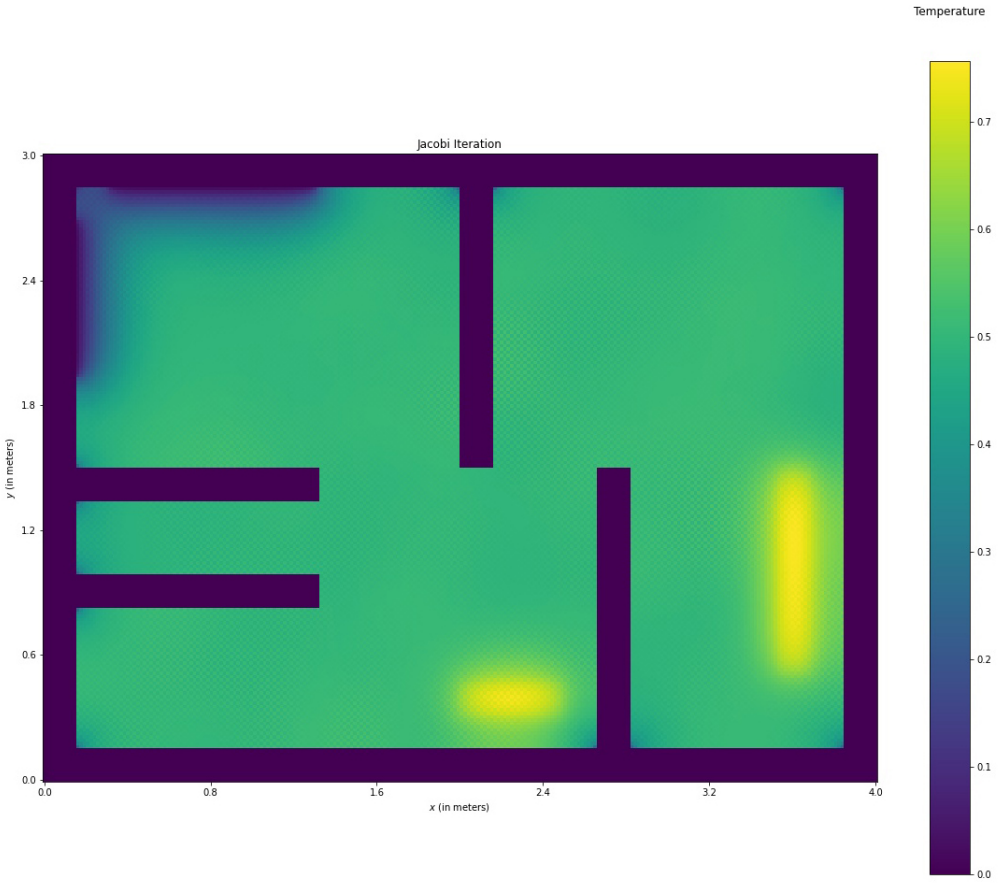

Figure 7. Steady state solution from the Jacobi method.

Looking at the Jacobi solution, the solution appears to have a 'checkerboard"-like pattern throughout the solution (especially near walls and windows). If we view Jacobi as an explicit time integrator, then we can perhaps explain the above observation. Below we plot $h_t\lambda_i$, where λ_i is an eigenvalue of our discrete Laplacian matrix:

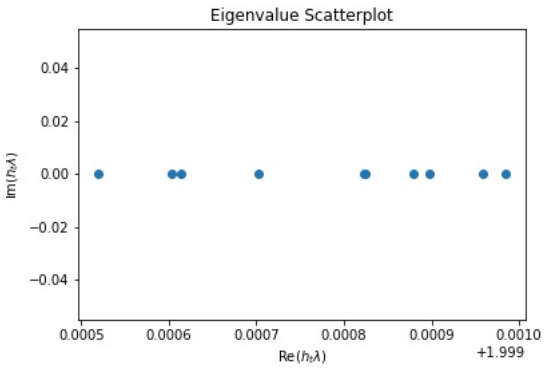

Figure 8. Scatterplot using the top 10 eigenvalues.

This plot suggests that $|\text{Re}(h_t\lambda)| \leq 2$. It would be reasonable to say that the Jacobi plot comes from those eigenvalues being so close to the upper bound (if not exact).

We will also compute the steady state solution via the damped Jacobi method. We will run the method for 300 iterations from a random vector of numbers as the initial guess. We specify a weight parameter $\alpha = 0.05$ here. Below is the result of this computation:

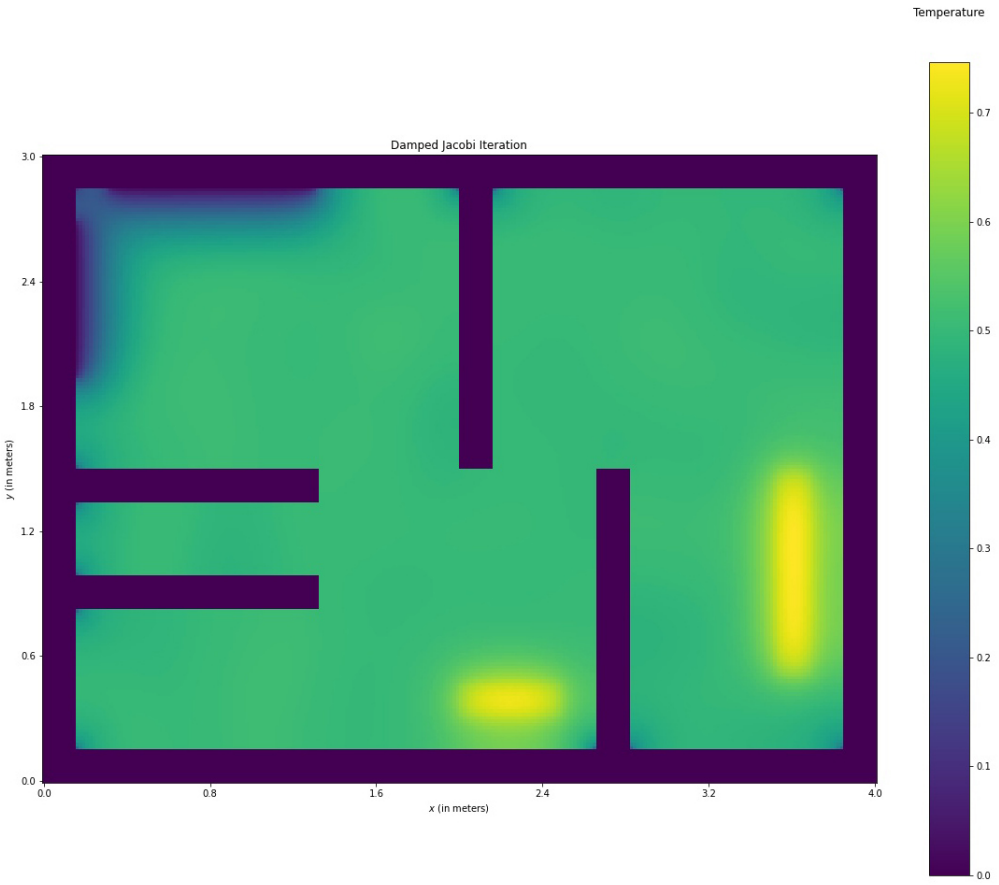

Figure 9. Steady state solution from the damped Jacobi method using $\alpha = 0.05$.

Instead of getting the "checkerboard" effect from the Jacobi plot, we can see that the steady state solution now has some sort of "smoothing" effect here. Since we are working with a discrete mesh, solving for an approximated solution will require some sort of interpolation to occur here (i.e. smoothing).

Part 3

For this part, we will consider the Geometric Multigrid method. For the restriction operator, we will simply just pick off even rows/columns of a fine mesh to compute the coarser mesh. For the prolongation operator, we will use bilinear interpolation on the coarse mesh to compute the finer mesh.

To show that our restriction/prolongation operators (along with the boundary conditions) achieves second order accuracy in the ℓ^∞-norm, we will test out these operators on test PDEs (8) and (9). Using our implementation on a 100×100, 200×200, 300×300, and 400×400 grid, we arrive at the following results:

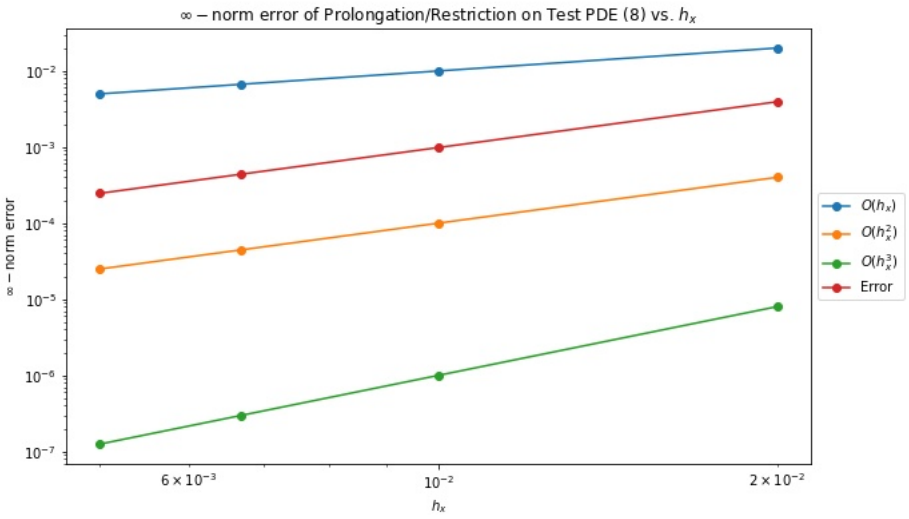

Figure 10. Error convergence plot on (8)

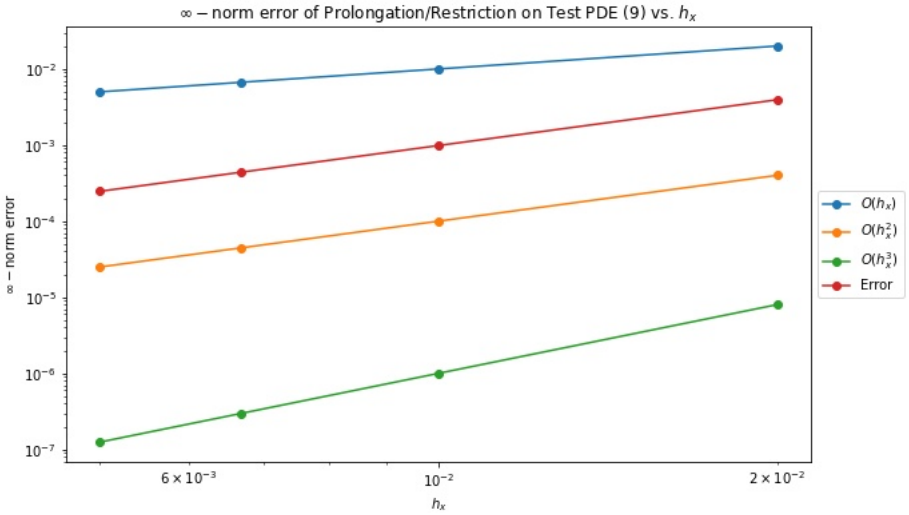

Figure 11. Error convergence plot on (9)

These plots show that our restriction/prolongation operators do in fact achieve second-order spatial accuracy.

For solving the steady-state solution, we will use a 6-Layer Multigrid V-Cycle:

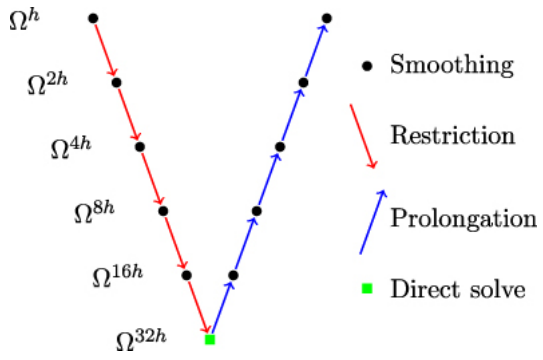

Figure 12. V-Cycle Architecture

We run the method from a random vector until the residual norm has decreased by $\sim 10^{12}$. Below is a semilogarithmic plot of the residual norm of the system against the iteration count and what the steady-state solution looks like under V-Cycle:

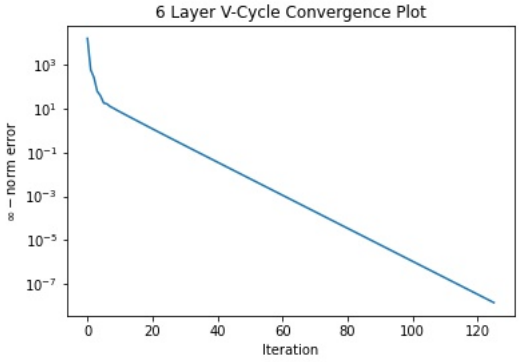

Figure 13. 6-Layer V-Cycle residual norm plot.

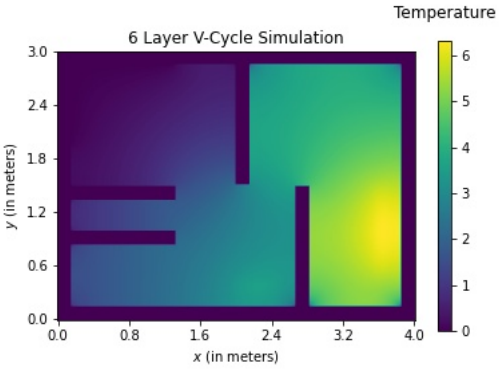

Figure 14. Steady state solution generated from 6-Layer V-Cycle.

From Figure 13, we can see that the 6-Layer V-Cycle converges quite quickly. In fact, it has linear convergence.