1.An Explicit Solver

In this project, we will write a solver for the heat equation in a complex two dimensional domain.

$$u_t = \Delta u + r(x) = \partial_x^2 u + \partial_y^2 u + r(x, y) \tag{1}$$

where $r(x)$ is a time constant heat sources function. The initial and boundary conditions are as follows.

$$u(x, 0) = 0 \qquad (x \in \Omega) \tag{2}$$
$$u(x, t) = 0 \qquad (x \in \Gamma_W) \tag{3}$$
$$\hat{n} \cdot \nabla u(x, t) = 0 \qquad (x \in \partial\Omega \backslash \Gamma_W) \tag{4}$$

The scenario is that we will simulate the heat flow in a room. In this scenario, Ω is the room, Γ_W is window and $\partial\Omega\backslash\Gamma_W$ is wall. The specification of boundary conditions says that the outside air is at zero degrees and the wall is taken as an insulator where no heat exchange can occur. The initial temperature is taken as zero and the heater in the room is modeled as heat source. The domain is shown in Fig.1. The thickness of the wall is 0.2. There are three heaters with dimension 0.4×0.4 at $(1.6, 1.6)$, $(1.2, 3.9)$, and $(3.4, 3.9)$.

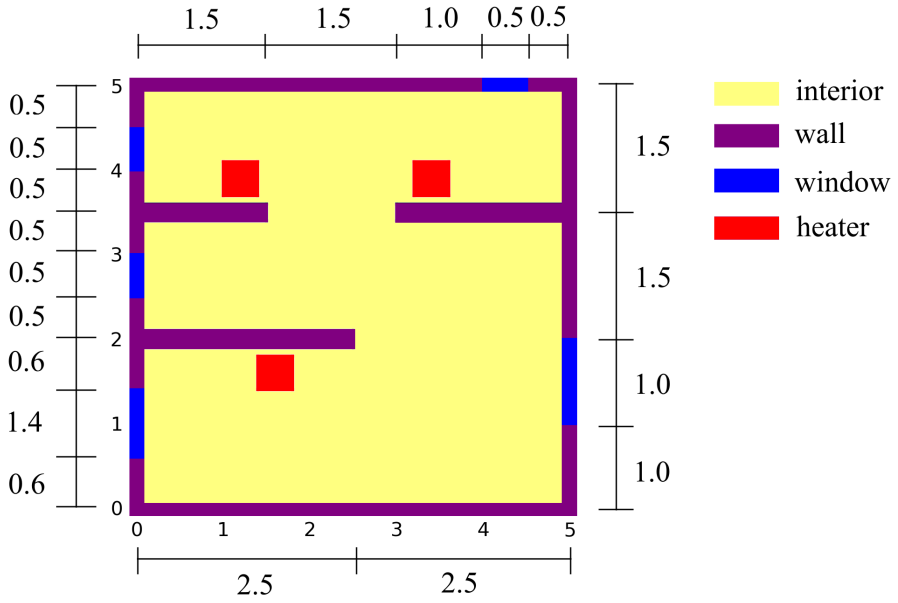

Figure 1: Room plan

1.1. Finite Difference Approximation

In this section, a strategy to solve the problem in 1D using finite difference will be investigated. Finite difference will be use to discretize the space. The treatment of Dirichlet and Neumann boundary conditions will be discussed and the verification with a steady state test case will be shown. The steady state equation will be considered so we can focus on the spatial discretization.

$$\Delta u + r(x) = \partial_x^2 u + r(x) = 0 \tag{5}$$

For this section, consider the domain $\Omega = [0, 1]$.

1.1.1 Second Order Finite Difference

Second order central difference will be used to approximate first and second order derivative. Consider a one dimensional grid points $\{x_i\}_{i=0}^{i=n}$ with spacing h_x, the second order finite difference approximation are.

$$u_i' = \frac{u_{i+1} - u_{i-1}}{2h_x} \tag{6}$$

$$u_i'' = \frac{u_{i+1} - 2u_i + u_{i-1}}{h_x^2} \tag{7}$$

Where $u_i \approx u(x_i)$. Using second order finite difference, (5) can be written as follows.

$$\begin{bmatrix} -\frac{2}{h_x^2} & \frac{1}{h_x^2} & 0 & ... & & \\ \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & 0 & ... & \\ 0 & \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} \\ & & & & \frac{1}{h_x^2} & -\frac{2}{h_x^2} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = - \begin{bmatrix} r(x_0) \\ r(x_1) \\ r(x_2) \\ \vdots \\ r(x_{n-1}) \\ r(x_n) \end{bmatrix}$$

1.1.2 Dirichlet Boundary Conditions

To impose Dirichlet boundary conditions for u at x_i, we just need to change the i^{th} row to match the equation $u_i = g(x_i)$ where $g(x)$ is the Dirichlet boundary condition. Consider a homogeneous Dirichlet boundary condition $g(1) = 0$. Imposing the boundary condition, the matrix equation becomes.

$$\begin{bmatrix} -\frac{2}{h_x^2} & \frac{1}{h_x^2} & 0 & ... & & \\ \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & 0 & ... & \\ 0 & \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = - \begin{bmatrix} r(x_0) \\ r(x_1) \\ r(x_2) \\ \vdots \\ r(x_{n-1}) \\ 0 \end{bmatrix}$$

Since the last equation is trivial, only the other equations will be solved.

$$\begin{bmatrix} -\frac{2}{h_x^2} & \frac{1}{h_x^2} & 0 & ... & \\ \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & 0 & \\ 0 & \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & \\ & & \ddots & \ddots & \ddots \\ & & & \frac{1}{h_x^2} & -\frac{2}{h_x^2} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \end{bmatrix} = - \begin{bmatrix} r(x_0) \\ r(x_1) \\ r(x_2) \\ \vdots \\ r(x_{n-1}) \end{bmatrix}$$

Because there are no approximation used, this treatment for Dirichlet boundary condition is expected to maintain the second order accuracy of the centered difference that is used in the interior of the domain.

1.1.3 Neumann Boundary Conditions

To maintain second-order accuracy, we will use a new technique known as using "ghost points". For this technique, we assume $u(x)$ extends smoothly outside of the domain so we can make the usual second order centered approximation to the first derivative (eq.(6)). If $u'(x_i) = \alpha$, we have

$$u_i' = \alpha = \frac{u_{i+1} - u_{i-1}}{2h_x}$$

We can use this to solve the value of the extension (ghost nodes) in the positive direction.

$$u_{i-1} = u_{i+1} - 2h_x\alpha$$

and in the negative direction

$$u_{i+1} = u_{i-1} + 2h_x\alpha$$

We can substitute this into (7). For the calculation using ghost node in the positive direction,

$$u_i'' = \frac{u_{i+1} - 2u_i + (u_{i+1} - 2h_x\alpha)}{h_x^2} = \frac{2u_{i+1} - 2u_i}{h_x^2} - \frac{2\alpha}{h_x} \tag{8}$$

and in the negative direction

$$u_i'' = \frac{(u_{i-1} + 2h_x\alpha) - 2u_i + u_{i-1}}{h_x^2} = \frac{2u_{i-1} - 2u_i}{h_x^2} + \frac{2\alpha}{h_x} \tag{9}$$

This results was derived using second order finite difference for first and second derivative, so it is expected that this treatment for Neumann boundary conditions will preserve the second order accuracy of centered difference that is used in the interior of the domain.

Now, consider the previous problem with Neumann boundary condition $u'(0) = 0$. This will translate to

$$u_0'' = \frac{2u_1 - 2u_0}{h_x^2}$$

Then, combined with the Dirichlet boundary condition from Section 2.2, the system of equation becomes

$$\begin{bmatrix} -\frac{2}{h_x^2} & \frac{2}{h_x^2} & 0 & \cdots & \\ \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & 0 & \\ 0 & \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} & \\ & & \ddots & \ddots & \ddots \\ & & & \frac{1}{h_x^2} & -\frac{2}{h_x^2} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \end{bmatrix} = - \begin{bmatrix} r(x_0) \\ r(x_1) \\ r(x_2) \\ \vdots \\ r(x_{n-1}) \end{bmatrix}$$

1.1.4 Verification: 1D

In this section we will verify whether the scheme from previous sections has second order accuracy. Consider the steady state temperature distribution that satisfies boundary condition in the previous chapters ($u_x(0) = 0$ and $u(1) = 0$).

$$u(x) = 20(1 - x^4)$$

Notice that this satisfy the boundary condition.

$$u_x(0) = -80(0)^3 = 0$$

Substitute this to (5) to get the corresponding heat source.

$$r(x) = -u_{xx} = 240x^2$$

Several different mesh size h were used to solve the problem. Let u_i be the finite difference approximate solution and $u(x)$ be the true solution. The l_∞ error for each number of grid is computed as follows.

$$e = \max_i |u_i - u(x_i)|$$

The results is shown in Fig.2. This shows that the implemented 1D finite difference scheme (together with the way we handle boundary condition) has second order accuracy.

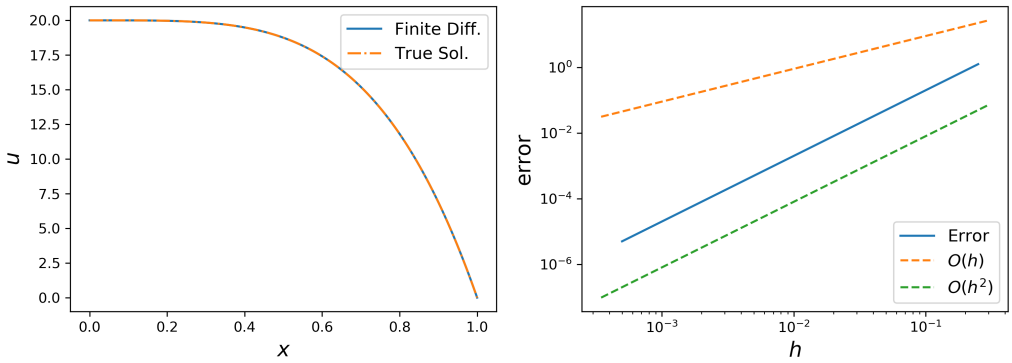

Figure 2: Steady state solution and l_∞ error for the 1D test problem

1.1.5 Verification: 2D

Applying centered difference in two dimension,

$$\begin{aligned}\Delta u(x_i, y_j) &= \partial_x^2 u(x_i, y_j) + \partial_y^2 u(x_i, y_j) \\ &\approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2}\end{aligned}$$

Where i and j are enumeration of grid points in the x and y direction. The same strategy as before will be used to treat Dirichlet and Neumann boundary conditions. Consider the following test case where the domain $\Omega = [0, 1] \times [0, 1]$.

$$\begin{aligned}\Delta u(x, y) + r(x, y) &= 0 \\ u(x, 0) = u(0, y) &= 0 \\ u_x(1, y) = u_y(x, 1) &= 0\end{aligned} \tag{10}$$

Consider the following steady state temperature distribution.

$$u(x, y) = 20(4x - x^4)(4y - y^4)$$

Notice that this satisfy the boundary condition. Substitute this to the (10) to get the corresponding heat source.

$$r(x, y) = 240\left((x^2)(4y - y^4) + (4x - x^4)(y^2)\right)$$

Several different mesh size h were used to solve the problem. The results is shown in Fig.3. This shows that the implemented 2D finite difference scheme (together with the way we handle boundary condition) has second order accuracy.

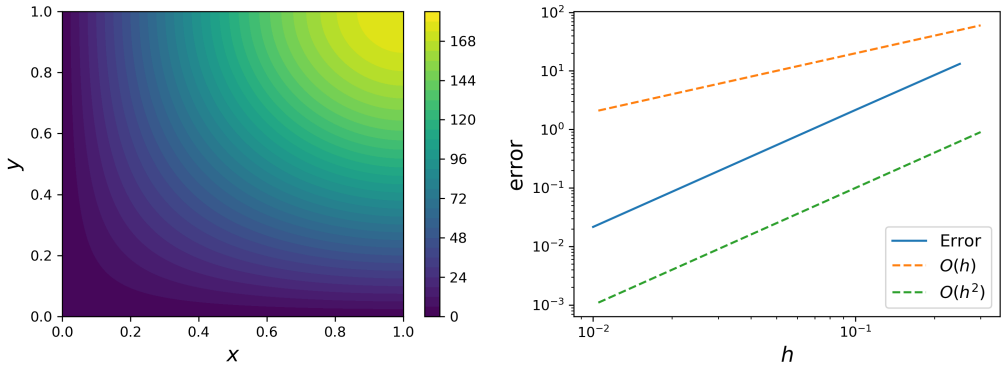

Figure 3: Steady state solution and l_∞ error for the 2D test problem

1.2 Time Integrator

1.2.1 Forward Euler: von Neumann Stability Analysis

In this section, the stability restriction on time step for Forward Euler will be derived. Recall Forward Euler scheme for homogeneous equation $u_t = \Delta u$ in one dimension.

$$u_{k,l+1} = u_{k,l} + h_t \frac{u_{k+1,l} - 2u_{k,l} + u_{k-1,l}}{h_x^2}$$

Where h_t is time step size and h_x is the grid size. Periodic boundary conditions is assumed. We will use von Neumann stability analysis to establish conditions under which this scheme is stable. First, rearrange the equation.

$$u_{k,l+1} = \lambda u_{k+1,l} + (1 - 2\lambda)u_{k,l} + \lambda u_{k-1,l}$$

Where $\lambda = h_t/h_x^2$. This scheme can be written as

$$P_h \mathbf{u}_l + 1 = Q_h \mathbf{u}_l$$

where $\mathbf{u}_l$ is the vector of the approximation at each grid point at time l. Elements of the vector are ordered such that adjacent element represent adjacent grid points.

$$P_h = I, \qquad Q_h = \text{tridiag}\,(\lambda, 1 - 2\lambda, \lambda)$$

Recall that P_h and Q_h are Toeplitz operator. The corresponding Toeplitz vector, p_k and q_k are given below.

$$p_k = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases} \qquad q_k = \begin{cases} \lambda & \text{if } k = -1 \\ 1 - 2\lambda & \text{if } k = 0 \\ \lambda & \text{if } k = 1 \end{cases}$$

Apply discrete Fourier transform.

$$\begin{aligned}
\hat{\mathbf{p}}(\varphi) &= \sum_k p_k e^{-i\varphi k} = e^{-i\varphi(0)} = 1 \\
\hat{\mathbf{q}}(\varphi) &= \sum_k q_k e^{-i\varphi k} \\
&= \lambda e^{-i\varphi(-1)} + (1-2\lambda)\, e^{-i\varphi(0)} + \lambda e^{-i\varphi(1)} \\
&= \lambda \left(e^{-i\varphi} + e^{i\varphi}\right) + (1-2\lambda) \\
&= \lambda \cos(\varphi) + (1-2\lambda)
\end{aligned}$$

Check the first condition of *von Neumann* stability. For some constant c which is independent of h_x and h_t,

$$\max_\varphi \left| \frac{1}{\hat{\mathbf{p}}(\varphi)} \right| = \max_\varphi \left| \frac{1}{1} \right| = 1 \leq c$$

Check the second condition of *von Neumann* stability.

$$\begin{aligned}
\max_\varphi |s(\varphi)| &= \max_\varphi \left| \frac{\hat{\mathbf{q}}(\varphi)}{\hat{\mathbf{p}}(\varphi)} \right| \\
&= \max_\varphi \left| \frac{\lambda \cos(\varphi) + (1-2\lambda)]}{1} \right| \\
&= \max_\varphi |\lambda \cos(\varphi) + (1-2\lambda)|
\end{aligned}$$

We use the fact that $\max_\varphi |s(\varphi)| = \max_\varphi |s(\varphi)|^2$

$$\max_\varphi |s(\varphi)|^2 = \max_\varphi \left(\lambda^2 \cos^2(\varphi) + 2\lambda(1-2\lambda)\cos(\varphi) + (1-2\lambda)^2\right)$$

To find the maximum, differentiate the expression inside the bracket with respect to φ and find the zeros.

$$-2\lambda^2 \cos(\varphi)\sin(\varphi) - 2\lambda(1-2\lambda)\sin(\varphi) = 0$$

The zeros of this equation is when $\sin(\varphi) = 0$. When this happens, we can have $\cos(\varphi) = 1$ or $\cos(\varphi) = -1$. First, we check the case when $\cos(\varphi) = 1$.

$$\begin{aligned}
\max_\varphi |s(\varphi)|^2 &\leq 1 \\
\lambda^2 + 2\lambda - 4\lambda^2 + 1 - 4\lambda + 4\lambda^2 &\leq 1 \\
\lambda^2 - 2\lambda &\leq 0 \\
\lambda(\lambda - 2) &\leq 0
\end{aligned}$$

Since we are using positive step both in time and space, $\lambda > 0$. To satisfy this condition,

$$\lambda \leq 2$$

Next, we consider the case when $\cos(\varphi) = -1$.

$$\begin{aligned} \max_{\varphi} |s(\varphi)|^2 &\leq 1 \\ \lambda^2 - 2\lambda + 4\lambda^2 + 1 - 4\lambda + 4\lambda^2 &\leq 1 \\ 9\lambda^2 - 6\lambda &\leq 0 \\ 3\lambda(3\lambda - 2) &\leq 0 \end{aligned}$$

To satisfy this condition,

$$\lambda \leq \frac{2}{3}$$

Using the smaller bound, we find that

$$h_t \leq \frac{2}{3} h_x^2$$

Notice that this is just for one dimension, we have the same scheme in the other dimension.

$$h_t \leq \frac{2}{3} h_y^2$$

Summing these two,

$$2h_t \leq \frac{2}{3} h_x^2 + \frac{2}{3} h_y^2$$

The final constrain in h_t for the scheme to be stable is

$$h_t \leq \frac{1}{3} \left(h_x^2 + h_y^2 \right) \tag{11}$$

1.2.2 Second Order Runge-Kutta

The implemented central difference is second order accurate. However, the accuracy of the solution of time depended problem also depends on accuracy of the time integrator. Forward Euler which is discussed in the previous section is only first order accurate. If we keep the proportion between time and space discretization $h_t/h_x^2 =$ constant, the local error will decrease with order $O(h_x^2)$. However, the global error that accumulate as we march through time is one order lower, $O(h_x)$. Because of this, even thought the space discretization is second order accurate, the combined scheme is only first order accurate. We can resolve this by setting h_t to be proportional to h_x^3, but this may significantly increase the computational cost. A better way to resolve this issue is to use a second order scheme like second order Runge-Kutta.

$$\begin{aligned} \frac{\partial u}{\partial t} &= f(t, u) \\ k_1 &= h_t f(t_l, u_l) \\ k_2 &= h_t f(t_l + h_t, u_l + k_1) \\ u_{l+1} &= u_l + \frac{1}{2} (k_1 + k_2) \end{aligned}$$

In our context,

$$f(u,t) = \Delta u + r(x)$$

Now we look at the time dependent version of the previous problem. The domain is $\Omega = [0,1]$.

$$\begin{aligned} \Delta u(x,y,t) + r(x,y,t) &= u_t \\ u(x,y,0) &= 0 \\ u(x,0,t) = u(0,y,t) &= 0 \\ u_x(1,y,t) = u_y(x,1,t) &= 0 \end{aligned} \tag{12}$$

Consider the following solution

$$u = 20\left(1 - e^{-t}\right)(4x - x^4)(4y - y^4)$$

Substitute this to (12) to get the corresponding heat source.

$$r(x,y,t) = 240\left(1 - e^{-t}\right)\left((x^2)(4y - y^4) + (4x - x^4)(y^2)\right) + 20e^{-t}(4x - x^4)(4y - y^4)$$

Several different mesh size h were used to solve the problem. Time step size $h_t = h^2/4$ will be used. The results is shown in Fig.4. This shows that the implemented 2D finite difference scheme has second order accuracy.

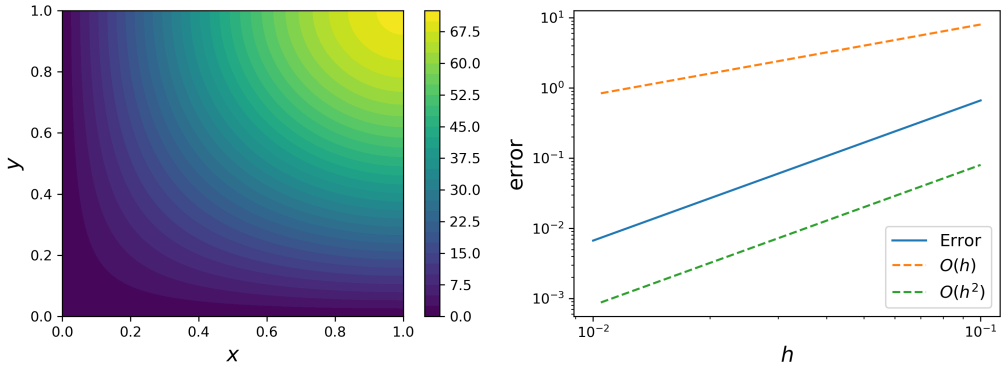

Figure 4: Transient solution at $t = 0.5$ and l_∞ error for the 2D test problem

1.3 Heat Flow in a Room

Finally we have all tools we need to simulate heat flow in a room. The heat flow will be modeled as a PDE (1) with initial and boundary conditions (2), (3), (4). The domain together with location of walls, windows, and heaters is given in Fig.1. Let S^{heat} be a set of grid points inside the heater. The heat source is defined as follow.

$$r(x,y) = \begin{cases} 120 & x \in S^{heat} \\ 0 & \text{otherwise} \end{cases}$$

Grid size and time step size are $h = 0.025$, $h_t = (h^2)/4 = 0.00015625$. The results are shown in Fig.5-8. The room plan overlay is also shown together with the temperature profile. Black is walls, blue is windows, and red is heaters.

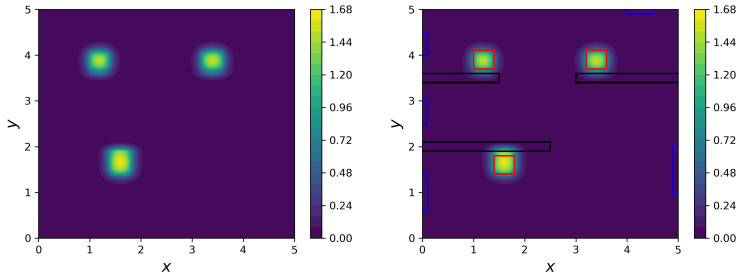

Figure 5: Temperature profile at $t = 0.015625$, with and without room plan overlay

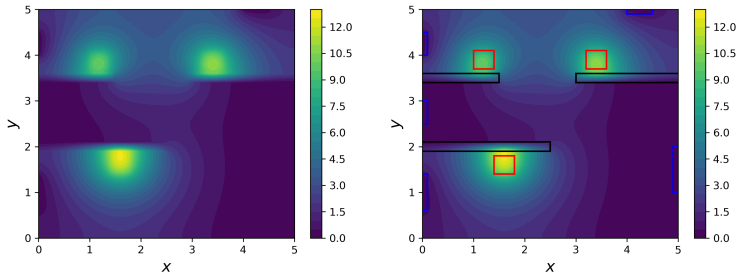

Figure 6: Temperature profile at $t = 0.937499$, with and without room plan overlay

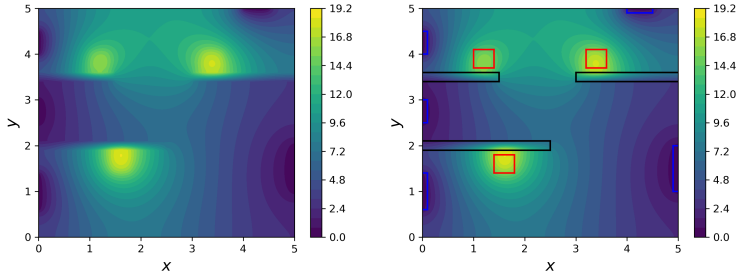

Figure 7: Temperature profile at $t = 4.687499$, with and without room plan overlay

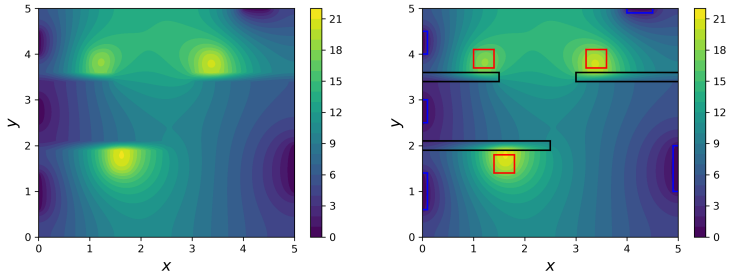

Figure 8: Temperature profile at $t = 31.99968$, with and without room plan overlay

2. Implicit Solver, Steady-State Solution

2.1 Diagonally Implicit Runge-Kutta Methods

In this section we will look into an implicit method. The advantage of using implicit method is that it is unconditionally stable. This gives us more freedom in choosing time step size because we do not have to worry about stability. The only restriction that we have in choosing the time step size comes from accuracy consideration. We will implement 2nd order, symplectic Diagonally Implicit Runge Kutta (DIRK) method form Qin and Zhang.

$$\begin{aligned}
\frac{\partial u}{\partial t} &= f(t,u) \\
k_1 &= f\left(t_l + \frac{1}{4}h_t, u_l + \frac{1}{4}h_t k_1\right) \\
k_2 &= f\left(t_l + \frac{3}{4}h_t, u_l + \frac{1}{2}h_t k_1 + \frac{1}{4}h_t k_2\right) \\
u_{l+1} &= u_l + \frac{h_t}{2}\left(k_1 + k_2\right)
\end{aligned}$$

For the heat equation,

$$f(t,u) = \Delta u + r$$

Notice k_1 and k_2 are defined implicitly. Let I be the identity operator and L_h be the finite difference operator for Δ. Derive the equation to solve for k_1.

$$\begin{aligned}
k_1 &= \Delta u_l + \frac{1}{4}h_t \Delta k_1 + r \\
\left(I - \frac{1}{4}h_t L_h\right) k_1 &= \left(L_h u_l + r\right)
\end{aligned}$$

and for k_2.

$$\begin{aligned}
k_2 &= \Delta u_l + \frac{1}{2}h_t \Delta k_1 + \frac{1}{4}h_t \Delta k_2 + r \\
\left(I - \frac{1}{4}h_t L_h\right) k_2 &= \left(L_h u_l + \frac{1}{2}h_t \Delta k_1 + r\right)
\end{aligned}$$

The same problem as in Section 1.3 will be solved with the implicit method. Grid size and time step size are $h = 0.025$, $h_t = (h^2) * 25 = 0.015625$. Notice that the time step h_t is 100 times larger that the time step that was used for explicit method. For simulating the heat flow until $t = 32$, the computation time is four times faster compared to the explicit method. The results has good agreement with the results from explicit method. They are shown in Fig.9-12.

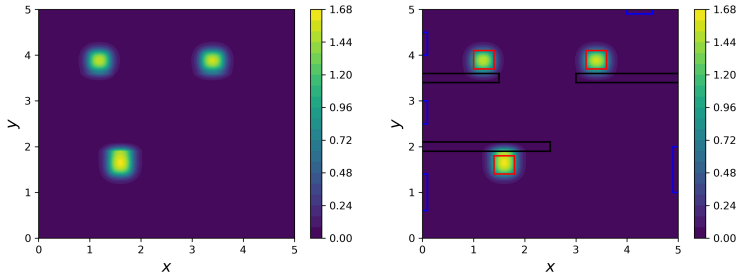

Figure 9: Temperature profile at $t = 0.015625$, with and without room plan overlay

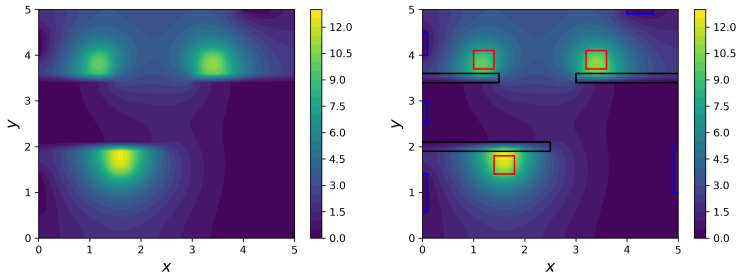

Figure 10: Temperature profile at $t = 0.937499$, with and without room plan overlay

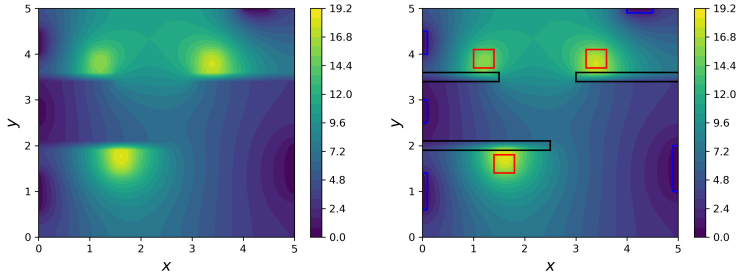

Figure 11: Temperature profile at $t = 4.687499$, with and without room plan overlay

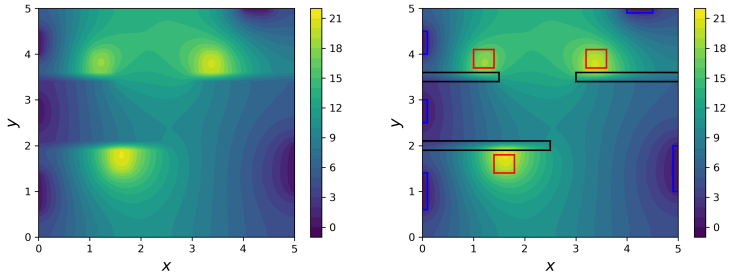

Figure 12: Temperature profile at $t = 31.99968$, with and without room plan overlay

2.2 Steady State Solution

When we reach steady state, the solution does not change in time. This means $u_t = 0$. So, we can solve the steady state directly by solving the poisson equation

$$\Delta u + r = 0$$

Solving the same setup as Section 1.3 and 2.1, with $u_t = 0$ gives the steady state solution for heat flow in a room problem. The result is shown in Fig.13. This has a good match with Fig.8 and Fig.12. This means both of the previously performed transient simulations have reached steady state.

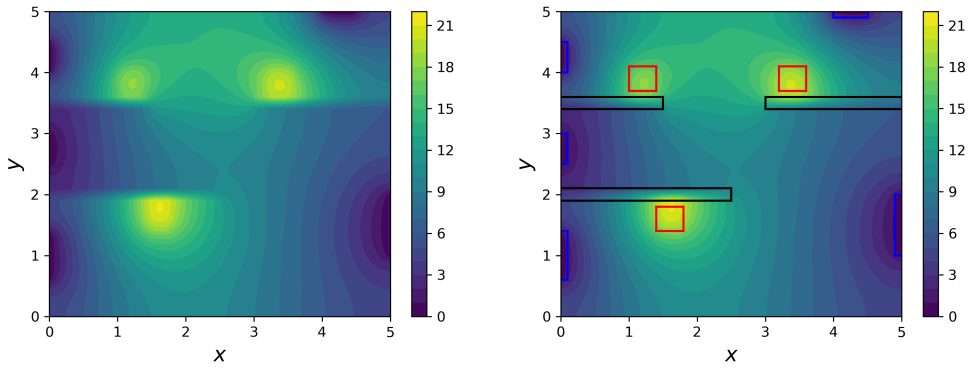

Figure 13: Steady state solution for heat flow in a room problem

2.3 Jacobi method

In this section we will implement Jacobi method to solve system of linear equations $Ax = b$ and use it to solve the steady state of heat flow in a room problem. Consider the discrete poisson equation

$$L_h u = -r$$

Where L_h is the finite difference operator that approximate the laplace operator. L_h can be decomposed into a diagonal component D, a lower triangular part L, and an upper triangular part U such that

$$L_h = D + L + U$$

Jacobi is an iterative method. Starting with an initial guess, the next approximation $u^{(k+1)}$ is obtained by the following formula.

$$u^{(k+1)} = D^{-1}\left(-r - (L+U)u^{(k)}\right)$$

This can also be written as

$$u^{(k+1)} = Tu^{(k)} + C$$

where

$$T = -D^{-1}(L+U)$$
$$C = -D^{-1}r$$

The iteration will be stopped once the residual is less than the chosen tolerance $\varepsilon = 10^{-8}$. The residual is computed as follows.

$$Res = \Delta u + r \approx L_h u + r \tag{13}$$

A random initial guess for 200 $\times$ 200 grid points will be used to start the iteration. We will show the solution and error evolution at and up to iteration 300 step. Then, we will run the simulation until we actually reached steady state. The results can be seen in Fig.14 and Fig.15.

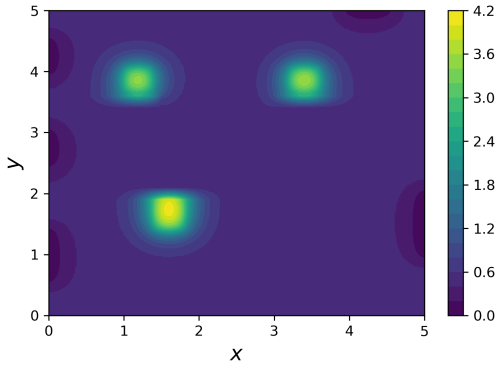

Figure 14: Solution at step 300 with Jacobi method

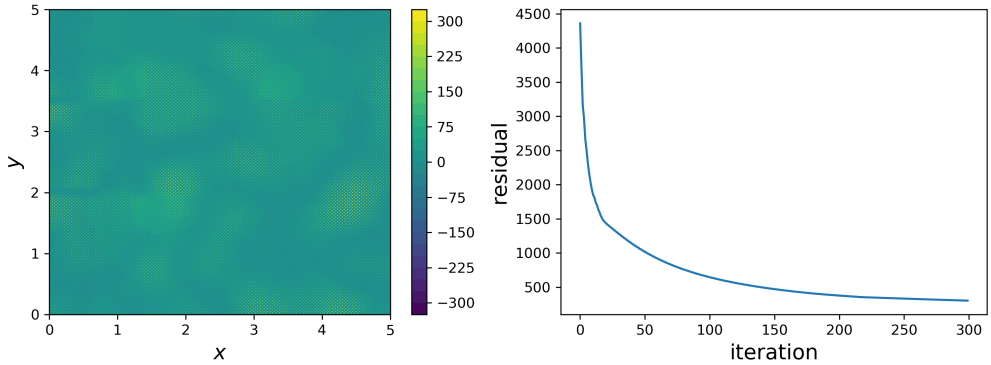

Figure 15: Residual distribution and evolution at and up to step 300 with Jacobi method

Looking at Fig.14, this looks like the early transient state of the heat flow. If we observe the evolution equation for Jacobi iteration, $u^{(k+1)} = Tu^{(k)} + C$, it does resemble an explicit time integrator. Recall that for Forward Euler, we have convergence criteria that the eigenvalue of the operator has to be inside the convergence region for the scheme to converge. This is analogous to Jacobi method, for convergence, the spectral radius $\rho\left(D^{-1}(L+U)\right) < 1$. This condition is satisfied if the operator is strictly or irreducibly diagonally dominant. It is said (ref:wikipedia) that Jacobi does not converge for every symmetric positive definite matrix. Our operator $(-\Delta)$ is symmetric and positive definite but for some reason in this case we got convergence. This probably is because of the shape of the domain, wall, and the way we treat Neumann boundary on the wall make the operator becomes not symmetric and positive definite anymore.

2.4 Damped Jacobi method

Let $\tilde{u}^{(k+1)}$ be the iterate computed by Jacobi at iteration $k+1$. For Damped Jacobi method, the iterative formula is

$$u^{(k+1)} = \alpha u^{(k)} + (1-\alpha)\tilde{u}^{(k+1)}$$

For an appropriately chosen small α. We will do the same thing as before, show the solution and error evolution at and up to iteration 300 step then run the simulation until we actually reached steady state. For this problem, we choose $\alpha = 0.05$. The results can be seen in Fig.16 and Fig.17.

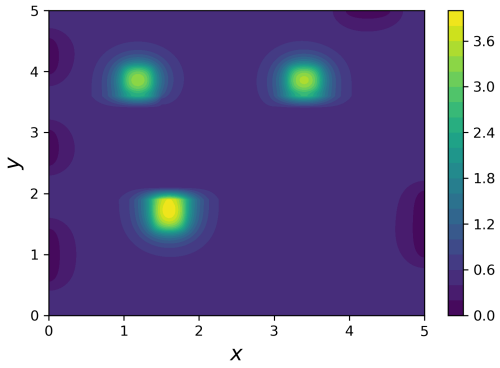

Figure 16: Solution at step 300 with damped Jacobi method

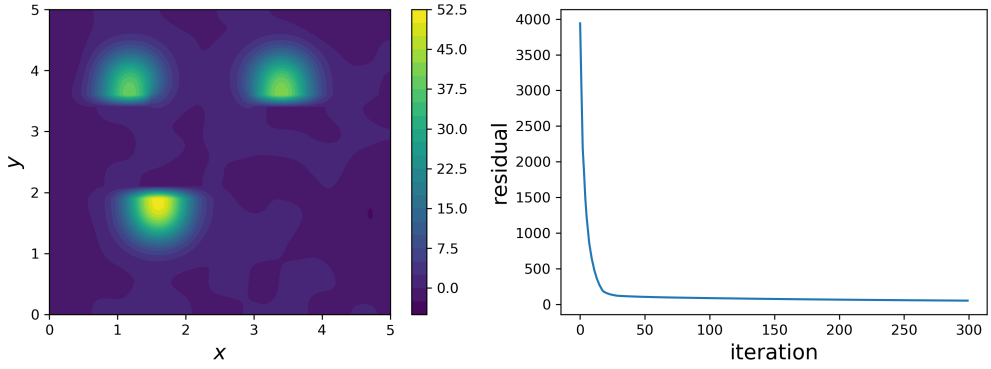

Figure 17: Residual distribution and evolution at and up to step 300 with damped Jacobi method

Comparing Fig.15 and Fig.17, we can see that the residual distribution for damped Jacobi is "smoothen". The high residual also concentrate at the heat source where we have high gradient. This can be explained by decomposing the error $e^{(i)}$ into eigenvectors of poisson equation and determining the convergence factors for these eigenmodes. This way, it can be shown that the damped Jacobi reduce the high frequency part of the error more quickly compared to the Jacobi method. In contrast, the low frequency part decrease slower than the Jacobi method. However, since we are looking at the first 300 steps, the higher frequency error is still visible (see Fig.15). As a results, the l_∞ error norm for the damped Jacobi is smaller and the error distribution appear to be more smooth compared to Jacobi method.

3. Geometric Multigrid

3.1 Restriction and Prolongation

In this section, "restriction" and "prolongation" operators will be constructed. They are operator that connect levels of meshes (with "restriction" going from fine to coarse, and prolongation the other way around). Both strategy will be tested against the same problem in section 1.1.5.

3.1.1 Restriction

Restriction operator allow us to go from finer mesh to coarser mesh. To do restriction, we simply picks values off the fine mesh at appropriate points (shown as green dot in Fig.18).

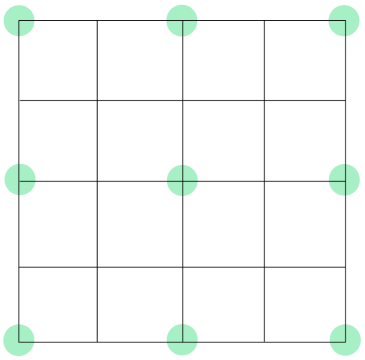

Figure 18: Grid points and value taken from it to go to coarser level

This strategy will be tested. First, solve the same problem as in section 1.1.5 and restrict the solution. Next, we compute the residual of the restricted solution is computed using eq.(13). The results are shown in Fig.19.

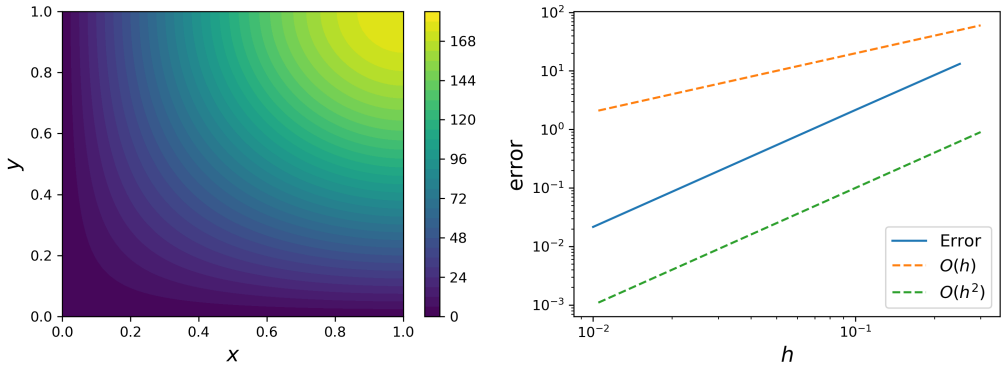

Figure 19: Restricted solution and its convergence rate

3.1.2 Prolongation

Prolongation operator allows us to project the solution go from coarser mesh to finer mesh. First, we recall the poisson equation for steady state of heat flow and the second order central difference

discretization on it evaluated at a point.

$$0 = \Delta u(x_i, y_j) + r(x_i, y_j)$$
$$0 = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} + r(x_i, y_j)$$

Solving this for $u_{i,j}$,

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

We first apply this equation in the diagonal direction. In Fig.20, we compute the blue node from value from green nodes. Next, we can compute the value of the orange nodes using the same equation.

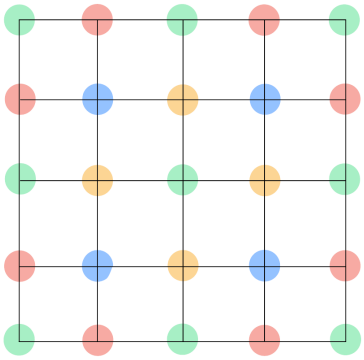

Figure 20: Scheme for prolongation: from green nodes, compute values at blue nodes, then orange nodes, then red nodes

At this points, we are ready to compute the value at the Dirichlet and Neumann boundary (red nodes on the Fig.20). The same treatment as in section 1.1.3 will be used for Neumann boundary conditions. We solve eq.(8) and eq.(9) by taking $u_i'' + r = 0$ and $\alpha = 0$ (assuming homogeneous Neumann BC). Applying this in both direction, we get the following.

$$u_{i,j} = u_{i+1,j} + r(x_i, y_j) \qquad \text{if ghost node is on the left}$$
$$u_{i,j} = u_{i-1,j} + r(x_i, y_j) \qquad \text{if ghost node is on the right}$$
$$u_{i,j} = u_{i,j+1} + r(x_i, y_j) \qquad \text{if ghost node is on the bottom}$$
$$u_{i,j} = u_{i,j-1} + r(x_i, y_j) \qquad \text{if ghost node is on the top}$$

Last, we ensure that we are imposing Dirichlet boundary condition by making the solution at corresponding nodes equal to zero. Now we are ready to test this strategy. To come up with function that solves the poisson equation, we solve the same problem in Section 1.1.5. Then we prolongate the solution and check the residual with eq.(13). The result is shown in Fig.21.

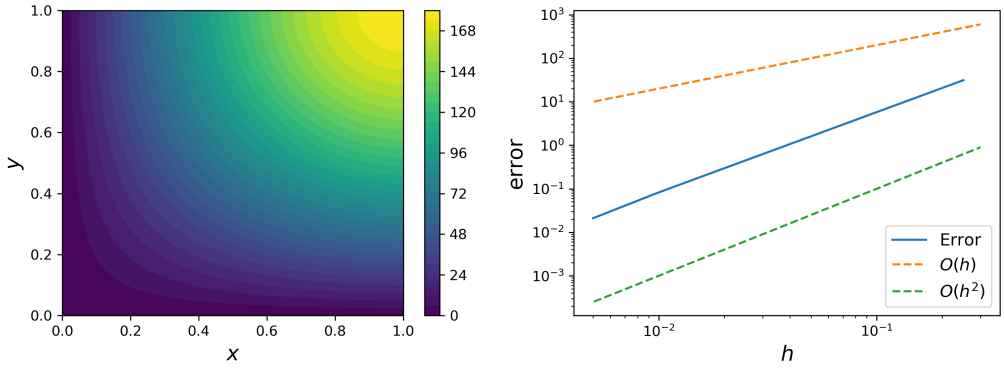

Figure 21: Prolongated solution and its convergence rate

3.2 Multigrid: V-Cycle

Using the restriction and prolongation operator, we implement a multigrid solver. We will use three level V-Cycle (two coarsening of the underlying mesh). Multigrid solver algorithm for two level V-Cycle is given below. To get three level, we simply apply the same algorithm when we solve the coarser mesh. Note that for this scheme, $r(i, j)$ will not be added when we prolongate the solution because we are adding the solution for residual to the original solution.

- Perform three or so Jacobi smoothing on $A_h v_h = f_h$
- Compute the residual $r_h = f_h - A_h v_h$
- Compute the restricted residual r_{2h} from r_h
- Obtain a coarse-grid correction c_{2h} approximating the solution of $A_{2h} c_{2h} = r_{2h}$
- Prolongate the coarse-grid correction c_{2h} to obtain c_h
- Correct v_h with c_{2h}: Let $w_h = v_h + c_h$
- Perform three more rounds of Jacobi smoothing on $A_h v_h = f_h$ and return w_h

We will solve the heat flow in a room problem using this scheme. We run the scheme on a family of grids ($n = 50$, $n = 100$, $n = 200$) until the residual norm has decreased by at least a factor of 10^{12}. The solution is shown in Fig.23 and the revolution of residual is shown in Fig.22

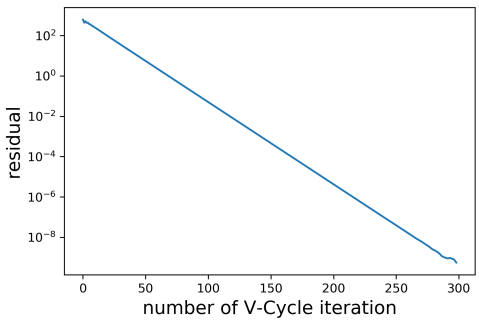

Figure 22: Residual evolution for the multigrid method

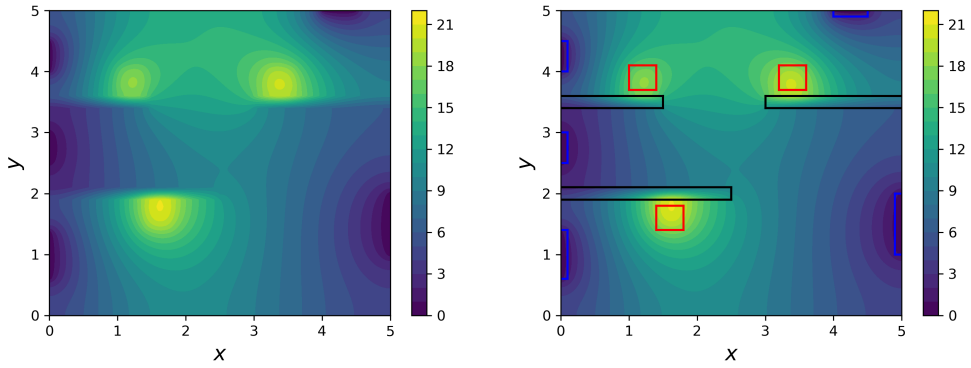

Figure 23: Solution of heat flow in a room problem with multigrid method

A. How to Reproduce Results

On the uploaded code, starting at line 825, there are several commented function call. To run the simulation and recreate figures in the report, just un-comment the corresponding function and run the code.