Computational Methods in Simulation hand-in week 1

KXS806

1 FINITE DIFFERENCE METHODS THEORY

The main idea of Finite Difference Methods is to replace derivatives with finite difference approximation so that we can easily compute simulations in computers with linear functions. FDM is a discretization method.

FDMs are one of the most widely used ways to numerical solutions of PDEs, to do this it needs to use Taylor series expansions to build the formulas, but during this, there will be some discretization error generated, so this is an approximation method.

When we want to do a simulation of a problem. We usually start from a given PDE and use the Taylor series to replace derivatives with finite difference approximations using a cook-book of rules. There are 3 commonly used approximations:

- forward difference approximation

  As we want to compute $\frac{\partial f_i}{\partial x}$, we need to use a 1st order Taylor expansion: $f(x_{i+1}) = f(x_i) + \frac{\partial f_i}{\partial x}\Delta x + o(\Delta x)$. Using above expansion we can get the approximation as

  $$\frac{\partial f_i}{\partial x} \approx \frac{f_{i+1} - f_i}{\Delta x}$$

- backward difference approximation

  We can re-write the Taylor expansion as$f(x_{i-1}) = f(x_i) - \frac{\partial f_i}{\partial x}\Delta x + o(\Delta x)$. Using above expansion we can get the approximation as

  $$\frac{\partial f_i}{\partial x} \approx \frac{f_i - f_i - 1}{\Delta x}$$

- central difference approximation

  We can add the forward and backward difference approximations and get the central difference approximation as

  $$\frac{\partial f_i}{\partial x} \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x}$$

2 2D TOY EXAMPLE

First, we have a function f(x,y) and a number k $\kappa > 0 \in \mathbb{R}$. let u(x,y) : $\mathbb{R}^2 \mapsto \mathbb{R}$ and let the PDE to be $\nabla^2 u - \kappa^2 u = f(x, y)$.Here we call a PDE defines what happens in the domain for a governing equation.

Then we want to calculate $\nabla^2 u$ by using 2dn order CD and the term u is sample value at node(i,j). Then we have

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

And then we can rewrite it to be

$$u_{i,j} = \frac{f_{i,j} - c_{i-1,j}u_{i-1,j} - c_{i,j-1}u_{i,j-1} - c_{i,j+1}u_{i,j+1} - c_{i+1,j}u_{i+1,j}}{c_{i,j}}$$

This is called an **update formula**for the unknown $u_{i,j}$. c's are coefficients and f are right-hand side terms.

By seeing the update formula we find whatever we want to calculate always depends on five neighbor values, this defines a **stencil** and can be graphically illustrated as fig1:

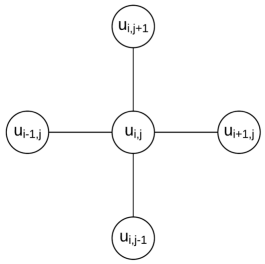

Fig. 1. stencil

With this stencil, we can update all nodes in the domain or called **compitational mesh**. The update stencils means that $\forall i, j$ we have a linear equation $u_{i,j} = g(u_{i,j})$where g is an affine function. This is a fixed point problem and we can use iteration to solve this problem. Assuming $u^0_{i,i} = 0$, then we get

$$u^{k+1}_{i,i} = g(u^k_{i,i})$$

continuing updating until the difference between $u^{k+1}_{i,i}$ and $u^k_{i,i}$ is small enough.

We can update inner nodes in domain by above update scheme, but we cannot compute central difference approximations on the border of our domain. For most of the examples we will only consider two different boundary conditions which are:

$$u(x, y) = k_d, \forall x, y \in \Gamma$$
$$\frac{\partial u}{\partial x} = k_n, \forall x, y \in \Gamma$$

Here Γ is the set of points on the boundary. There are two main techniques for dealing with boundary conditions: Elimination of unknown variables and adding ghost variables. I will use adding ghost variables here.First we need to add extra nodes to our domain and the results are as follows:

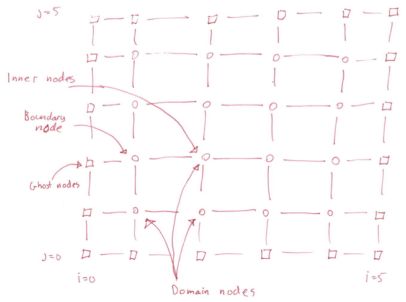

Fig. 2. after adding ghost nodes

Then we need to update those ghost nodes. As we have $\frac{\partial u}{\partial x} = 0$ on the boundary, we can use central difference approximation and get $\frac{\partial u_{\frac{1}{2},j}}{\partial x} = \frac{u_{1,j} - u_{0,j}}{\Delta x} = 0$. Then we can know the update formula for the ghost node $u_{o,j}$ is $u_{o,j} = u_{1,j}$.

In order to calculate updates more quickly we can use the matrix form of all the equations and concatenate those into a matrix containing all coefficients A with $\mathbf{Au} = \begin{bmatrix} \mathbf{A}_{DD} & \mathbf{A}_{DG} \end{bmatrix} \begin{bmatrix} \mathbf{u}_D \\ \mathbf{u}_G \end{bmatrix} = \mathbf{f}$.

3 PRACTICAL CODING

Here we will simulate the 2D toy example with f = x + y and k = 2. First, we draw the computational grid and domain as fig 3. Here the size of the domain is 6*6 and dx is set to be 1/4. As there's 20 ghost nodes on the outer border, there are 16 domain nodes in the domain area.

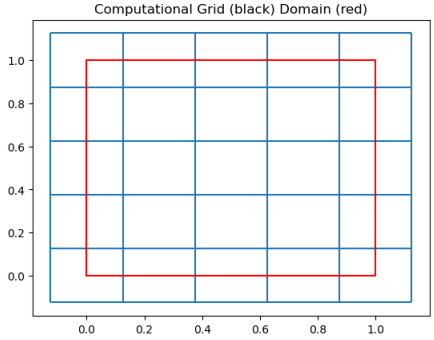

Fig. 3. computational grid

Then as we know

$$\nabla^2 u - k^2 u \approx \frac{1}{\Delta x^2} u_{i-1,j} + \frac{1}{\Delta y^2} u_{i,j-1} + (-k^2 - \frac{2}{\Delta x^2} - \frac{2}{\Delta y^2}) u_{i,j} + \frac{1}{\Delta y^2} u_{i,j+1} + \frac{1}{\Delta x^2} u_{i+1,j}$$

First, we used central difference approximation to compute matrix assemble without boundary conditions, we defined the coefficient matrix A and the right-hand side vector B. Then we plot the fill pattern of A as fig 4

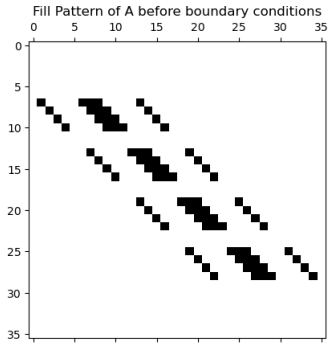

Fig. 4. Matrix A

Then we need to add stencils of the ghost nodes, for those nodes on the left and right boundary conditions we use the boundary condition $\frac{\partial u}{\partial x} = k_n, \forall x, y \in \Gamma$ and let $k_n = 0$ and we get

$$\left(\frac{\partial u}{\partial x}\right)_{\frac{1}{2},j} \approx u_{1,j} - u_{0,j} = 0 \quad (1)$$

$$\left(\frac{\partial u}{\partial x}\right)_{4+\frac{1}{2},j} \approx u_{5,j} - u_{4,j} = 0 \quad (2)$$

so we have $u_{0,j} = u_{1,j}$ and $u_{5,j} = u_{4,j}$. Then we modify the matrix A and get the result as fig5 and 6.

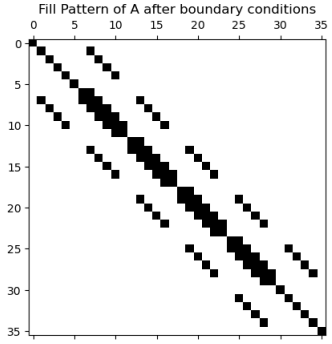

Fig. 5. Fill Pattern of A

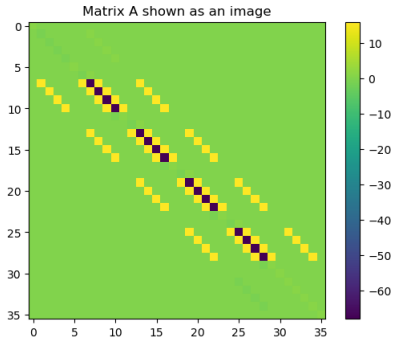

Fig. 6. image A

Finally, we can visualize the solution in 2D index space as fig7

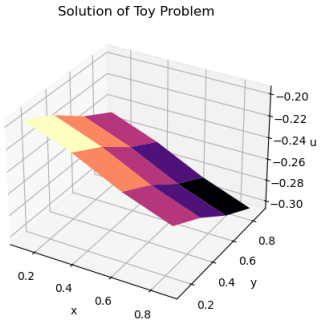

Fig. 7. result

Then I did some experiments. First I changed f into f = x*y and get the following result:

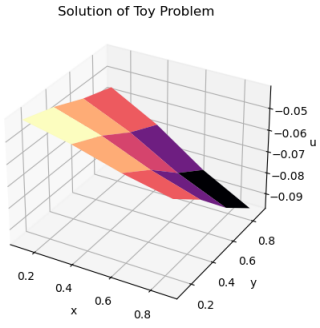

Fig. 8. result of f=x*y

We can see the shape of the result have been changed but the overall height had not been changed.

Then I changed k from 2 to 10 and get the following result:

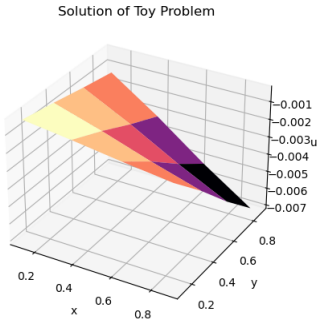

Fig. 9. result of f=x*y and k=10

We can see after changing the value of k the shape doesn't change but the height of the image has been changed.