Computational Methods in Simulation hand-in week 4

KXS806

1 FINITE ELEMENT METHOD

FEM is a method to solve some partial differential equations, its general idea is to split a system into several more minor elements, the number of these elements is finite so it's called the finite element method.

There are five steps in doing FEM for a given problem:

1.1 Write volume integral

We need to multiply the PDE with some trial function v(x) and then we integrate the whole formula over the domain. This step will introduce a system of algebraic equations that can be solved to obtain the approximate solution to the original PDE.

1.2 Integration by Parts

In this step, we simplify the formula we got in the last step using integration by parts. This step is to simplify the calculation because it can transform the second-order derivative into the first-order. So we make the y(x) from strong form to weak form so we slack the requirement to the function y(x).

1.3 Make an Approximation

Rather than find the real function y we can find a good approximation which we can write as

$$y(x) \approx \tilde{y}(x) = \sum_{i=1}^{n} N_i(x)\hat{y}_i$$

As computers can only deal with discrete values, we need to transform the continuous function into a set of discrete values to enable it to do the simulation. We can do this step using matrix multiply and get results very fast.

We can write it more compactly as the following formula:

$$\tilde{y}(x) = \sum_{i=1}^{n} N_i(x)\hat{y}_i = \underbrace{\begin{bmatrix} N_1(x) & \cdots & N_n(x) \end{bmatrix}}_{\mathrm{N}(x)} \underbrace{\begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix}}_{\hat{y}} = \mathbf{N}\hat{y}$$

shape function: shape function is used to approximate some shape within a finite element mesh with elements. They can help to describe the approximated point's coordinates. In some circumstances, the simulation is bad because the chosen shape function cannot describe the shape. E.g. we are simulating a curve but choose a linear shape function, then it will predict the shape inside the element is linear and make some error. Then we can either change another shape function that can describe the curve or use a finer grid and add the number of elements we use to provide a better simulation.

1.4 Choose a Trial Function

In this step, we need to define what our trial function looks like. The purpose of choosing a trial function is to construct an approximation that satisfies the essential properties of the model we want to simulate. Also, we need to consider if the function is simple to calculate. After replacing the v(x) with the chosen trial function we need to do some mathematical cleanup and get something similar to $K\hat{y} = f$. Now we have a simple linear system to solve.

1.5 Compute a Solution

As we already get the expression for K and f we need to calculate for $\hat{y}$. But first, we need to add boundary conditions. We have the linear system as follows now:

$$\begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} & \cdots & \mathbf{K}_{1,n-1} & \mathbf{K}_{1n} \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2,n-1} & \mathbf{K}_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{K}_{n-1,1} & \mathbf{K}_{n-1,2} & \cdots & \mathbf{K}_{n-1,n-1} & \mathbf{K}_{n-1,n} \\ \mathbf{K}_{n1} & \mathbf{K}_{n2} & \cdots & \mathbf{K}_{n,n-1} & \mathbf{K}_{nn} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{y}}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n-1} \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n-1} \\ \mathbf{f}_n \end{bmatrix}$$

Then we need to add the boundary described in the question to the right-hand side and change all points on the boundary to 1 and all points in that line to 0 so that line will result in the boundary. here as the example is in 1D there are only two boundary points so we only add two lines and get the following matrix:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2,n-1} & \mathbf{K}_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{K}_{n-1,1} & \mathbf{K}_{n-1,2} & \cdots & \mathbf{K}_{n-1,n-1} & \mathbf{K}_{n-1,n} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n-1} \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} a \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n-1} \\ b \end{bmatrix}$$

Then as in the code, we want a square matrix to do the calculation we need to move the known parts to right-hand side so we get the formula as:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & \mathbf{K}_{22} & \cdots & \mathbf{K}_{2,n-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \mathbf{K}_{n-1,2} & \cdots & \mathbf{K}_{n-1,n-1} & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{n-1} \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} a \\ \mathbf{f}_2 - \mathbf{K}_{21}a - \mathbf{K}_{2n}b \\ \vdots \\ \mathbf{f}_{n-1} - \mathbf{K}_{n-1,1}a - \mathbf{K}_{n-1,n}b \\ b \end{bmatrix}$$

Then we are good to go and just give the computer these two matrics and solve the system, then we can get a list of $\hat{y}$ and we can plot them to see the result of the simulation.

1.6 choose of grid size

One of the most important affect reasons for the simulation effect is the choice of element size. If we make the element very small then the effect of simulation will be improved, but it also requires a huge time to compute them, so we can make a none linear grid spacing.

When using linear grid spacing the distance between nodes within each element is uniform, it's easy to implement and very efficient for uniformly shaped elements like squares or triangles.

When using a non-linear grid spacing the spacing between nodes within each element varies, by using it we can get an improved accuracy when simulating a shape with localized features or gradients. Also, it can help to make the resource allocation more efficient, it focuses on computational effort in the region with the highly changed area and use less computation resource in the smooth region.

So, the choice between these two grid spacing depends on the problem, if the problem contains complex geometries or regions of rapid variations we can use a non-linear grid spacing to capture all feature more efficiently. However, it's more complex to implement and requires more effort in generating the mesh. Changing the grid spacing from linear to non-linear can affect the accuracy and efficiency, the choice should consider both the problem and the desired level of accuracy and computational efficiency.

2 PRACTIAL PART

2.1 The 1D Poisson Problem with Linear FEM

In this section, we first compute the K_e array and use that to assemble the global K array.

First we just ignore boundary conditions and simply just fill in values and we get the fill pattern and eigenvalues as follows:

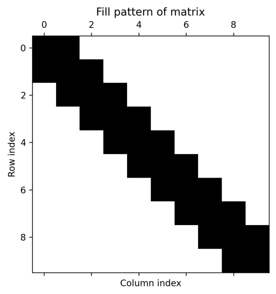

Fig. 1. Fill pattern without boundary condition

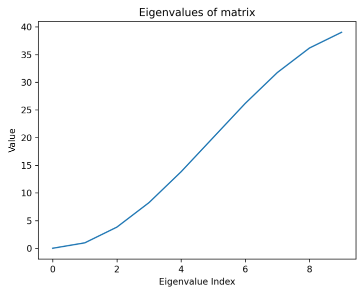

Fig. 2. eigenvalue without boundary condition

We can observe there's a 0 eigenvalue in the eigenvalue plot which means the formula is unsolvable now, so we need to add boundary conditions to limit it in a specific area. As it's a 1D example there are only two boundary points, we set their value to be 1 and 2 and modify the K and f matrix and solve the formula.

Then we plot the solution as the follow plot:

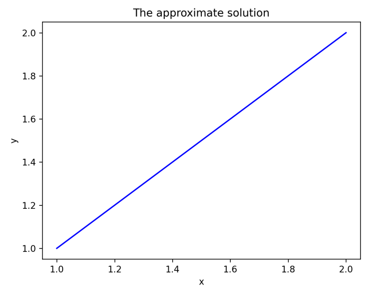

Fig. 3. solution plot

2.2 The 2D Poisson Problem with linear FEM

Similar in 1D example we first compute the K_e matrix and use it to assemble the K matrix, then we get the fill pattern and eigenvalues as follows:

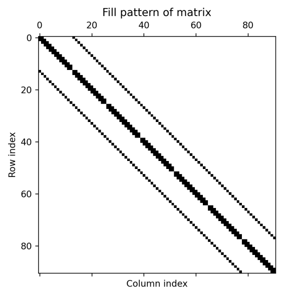

Fig. 4. fill pattern without boundary condition

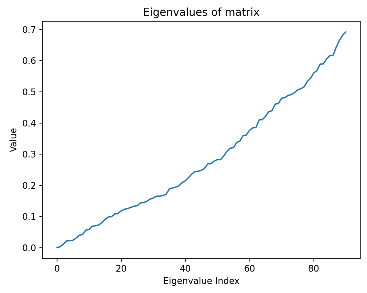

Fig. 5. Eigenvalue

Then we can add the boundary conditions, as it's a plane we make all left points have the value of 3 and all right points have a value of 2, so we get a slope. We plot the solution both in 2D and 3D as follows:

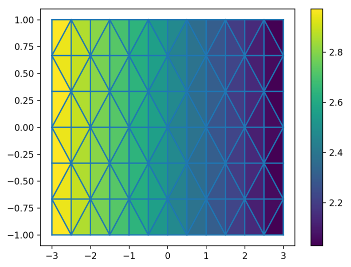

Fig. 6. 2D visulize

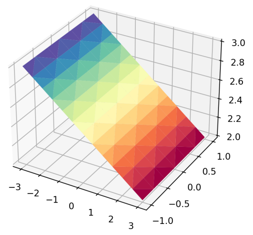

Fig. 7. 3D visulize

2.3 experiment

For the experiment part we changed the border's value as a quadratic equation and also remove the y points in Ke array and get a plot as follows:

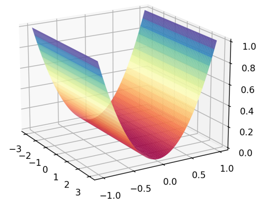

Fig. 8. new plot

Then we tried to change the size of grid, then we calculate the error by subtracting the real simulated value with the estimated value on each node and sum them up, and we get the error plot as follow:

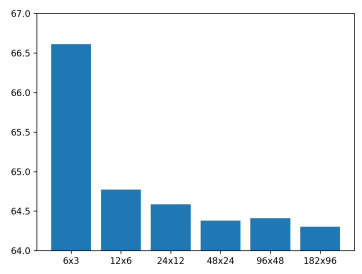

Fig. 9. error plot

From the plot, we can see as the grid size become smaller the error becomes lower.