Zeyi Xu *Jan 12, 2024*

226B Project 1

0.1 Problem setting and introduction

This is a MATLAB implementation of finite difference methods to Poisson problems. We consider the following 2D-Poisson problem on the rectangle area $D = (0,1) \times (0,2)$ with Dirichlet boundary conditions

$$\begin{cases} -\Delta u = 2\pi^2 \sin(\pi x)\sin(\pi y) \text{ in } D, \\ u = 0 \text{ on } \partial D. \end{cases} \tag{1}$$

This problem admits a smooth exact solution

$$u(x,y) = \sin(\pi x)\sin(\pi y). \tag{2}$$

In this project, multiple ways are explored to solve the problem. Following sections are a step-by-step implementation. All the codes are available at `https://github.com/ZeyiXuu/FDMethods_2DPoisson/`.

0.2 Discretization of problem

We first create a 2D uniform grid on the rectangle D. Use meshgrid function in MATLAB to generate a uniform grid with size h (see grid_eval.m). Then plot any function defined on D using surf function (see grid_eval.m). Take $h = 0.01$ and $f(x,y) = 2\pi^2 \sin(\pi x)\sin(\pi y)$ as an example, we shall get the following figure (Figure 1).

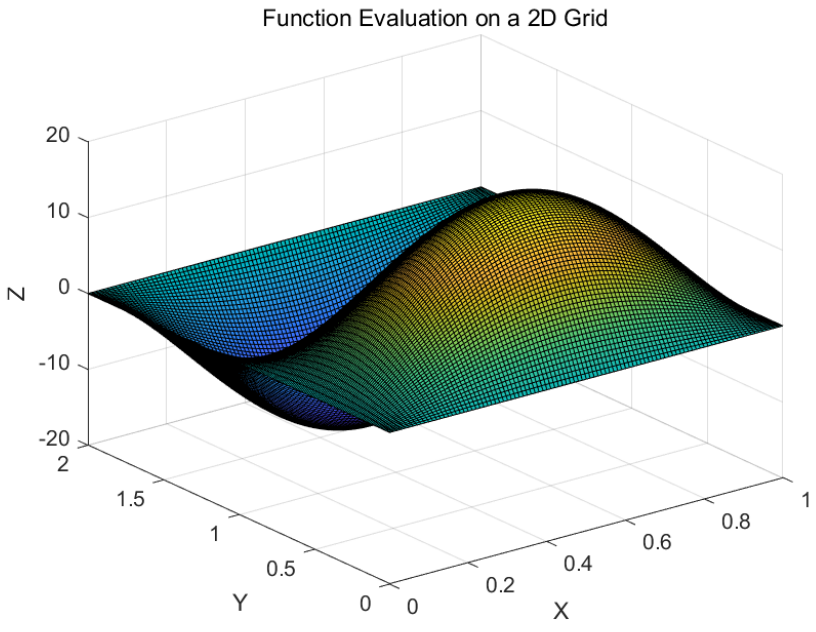

Figure 1: Example of a grid function

0.3 Three ways to implement matrix-vector product

From the discretization, the problem is reduced to solving the following linear system

$$Au = f, \tag{3}$$

where u and f are discretizations of the solution and right-hand-side function respectively, and A the tensor corresponding to the operator $-\Delta$. If we re-index so that u and f become vectors, A is then a big matrix. Then we have 3 ways to compute Au, given any u.

The first way is to view A as a big matrix, as stated above. Re-index u in a first-column-then-row manner, and do the same thing to f. Thus the matrix A is a block tri-diagonal matrix

$$A = \frac{1}{h^2}\begin{pmatrix} T & -I & & & \\ -I & T & -I & & \\ & -I & T & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & T \end{pmatrix}, \tag{4}$$

where

$$T = \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix}$$

is a tri-diagonal matrix. Detailed code please find ProductBigMatrix.m.

The second way is to view A as a transform to each entry of u. This is a tensor-free way, and it works since we do not need matrix A itself when solving linear systems, the mapping $A: \ u \mapsto Au$ is enough. Actually, the map is given element-wise by

$$(Au)_{i,j} = \frac{4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2}. \tag{5}$$

We implement this way using for loops, detailed code in ProductMatrixFree.m.

The third way is through tensor product structure. We view the 5-point stencil in 2D case as composition of 3-point stencils in 1D cases. More specifically,

$$Au = uT_x + T_y u, \tag{6}$$

where T_x and T_y are of the form

$$\begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}$$

and has compatible sizes. For detailed code please see Product1Dmatrix.m.

Experiment on these 3 ways using the same input u, we find that though they all yield the same result, as the problem size grows, the second and the third ways maintain similar speeds, while the first way is much slower. Thus, we choose the second way (matrix-free way) as the default way to do multiplication in the following chapters.

Next, we verify the truncation error of the problem. Take the exact solution $u(x, y) = \sin(\pi x)\sin(\pi y)$ as the input, and evaluate the infinite norm of the residue $Au - f$. Ideally, we would expect it to get closer to 0 as h decreases. In the experiment, we take $h = 1/2, 1/4, \cdots, 1/2^{10}$, in each case, we compute the norm of the residue. Then, taking logarithm and we get the plot below (figure 2).

The plot is almost a straight line. From the slope, we get the order of truncation error is 1.

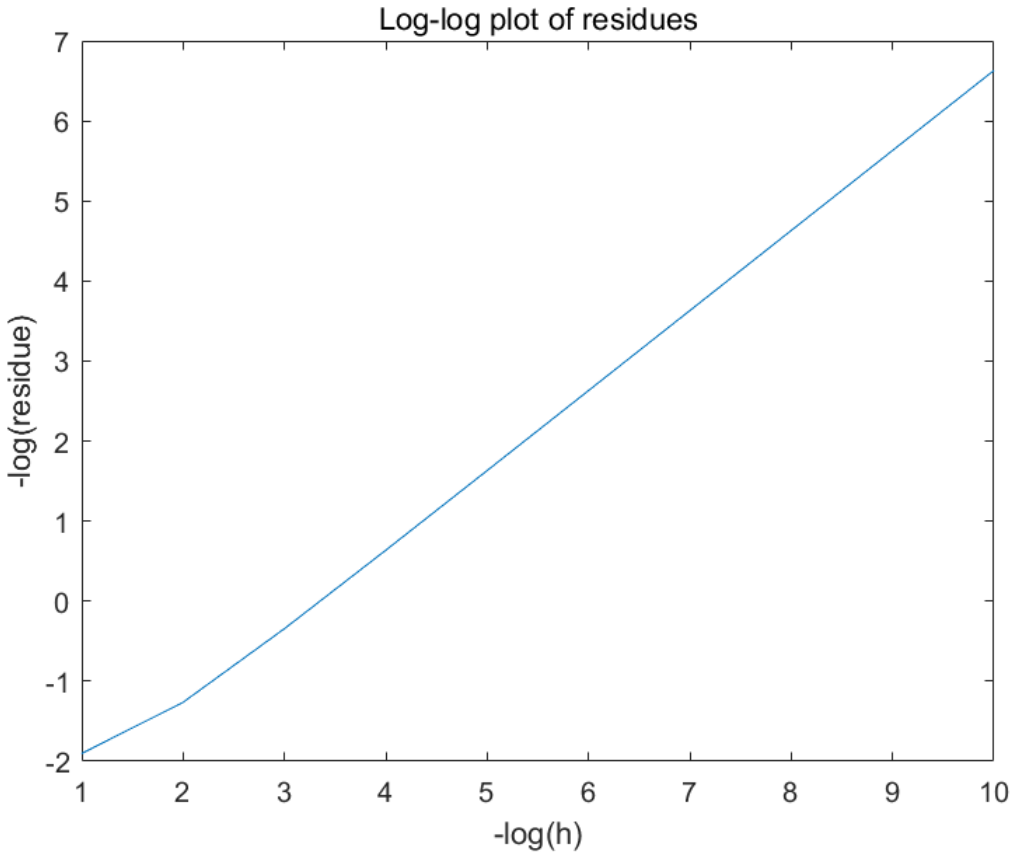

Figure 2: Log-log plot of residue with grid size

0.4 Boundary conditions

In the problem that we consider, the boundary condition is Dirichlet with zero on the boundary. But it turns out that this approach applies similarly to other problems.

If the problem is Dirichlet with general boundary values, we evaluate the function values at grid points on the boundary. Then if (i, j) is on the boundary, we compute $(Au)_{ij}$ using the matrix-free way and move it to the right hand side. Then the vector u after subtraction is again zero on the boundary.

If the problem is Neumann, on the boundary it holds

$$\frac{\partial u}{\partial \mathbf{n}} = g,$$

discretize the left hand side and g. Then we get the map on the boundary grids. Move the values hg_{ij} to right hand side, then we get the extended linear system

$$\tilde{A}\tilde{u} = \tilde{f}.$$

Now $\tilde{u}$ includes all the boundary values.

0.5 Two linear system solvers

The most direct way to solve the linear system is $u = A^{-1}f$. This method is notoriously slow as the dimension of A is $O(1/h^2)$. What's worse, it will be inaccurate if A is ill-conditioned. So we mainly consider the following two methods as surrogate ways.

The first way is Gauss-Seidel methods. See GS.m for a single iteration of Gauss-Seidel, and GS_iteration.m for a complete G-S method.

The second way is congugate gradient methods. Use the MATLAB built-in function pcg to solve the linear system, see pcg_solver.m for detailed code.

In the experiments of two methods, we both set the max iteration $N = 1000$, and tolerance $tol = 0.1h^2$. take $h = 0.1, 0.01$, and we verify that the two methods yield the same result.

0.6 Convergence

Finally, we want to verify that the numerical result yielded by above solvers converges to the true solution as $h \to 0$. To this end, we take $h = 1/8, 1/16, 1/32, 1/64$. We solve the problem with each grid size h, and get the numerical solution say uh. We then compute the corresponding infinite norm of the error $uh - uI$, where uI is the grid version of the true solution. Then we get the data below: From the

h	error norm
1/8	0.0651
1/16	0.0327
1/32	0.0164
1/64	0.0082

table it is seen that as grid size h decreases to $h/2$, the error norm also decreases approximately half of itself, showing that the order of the error is 1.