Resolution of a 2D Poisson problem

Numerical analysis - University of Luxembourg

Project 2

Introduction

The Poisson equation appears in many physical applications: electrostatics, fluid mechanics, gravitational field, heat conduction, etc. Let us consider the Poisson problem in two space dimensions x and y. It takes the form

$$-\Delta u = f(x,y), \quad (x,y) \in \Omega, \quad \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{1}$$

where $u = u(x,y)$ is the function we are seeking, $f = f(x,y)$ is a source term, and Ω is the computational domain. In this project you will design a Poisson solver over a square domain

$$\Omega = \{(x,y),\ 0 \leq x \leq 1, 0 \leq y \leq 1\}, \tag{2}$$

and analyze its numerical properties. We propose to discretize the Poisson equation by a classical finite difference scheme, and to investigate the numerical performance of various linear system solvers. In the first part you will focus on *finite difference discretization* in two-dimensions and the verification of the code. In the second part you will test different direct and iterative linear system solvers, and try to understand their performance.

You should write a small project report that shows your results, and summarize your findings. For the evaluation I will pay attention to the quality of the interpretations, how you apply the algorithms and the quality of the numerical illustrations. It does not need to be long: I will prefer a short summary that shows that you have understood and analyzed the methods.

1 Finite difference discretization

In this part we propose to i) find a discrete version of the Poisson problem, ii) implement and validate the solver.

1.1 Designing the solver

You may follow the steps below

1. Generate a rectangular grid of $(N_x - 1) \times (N_y - 1)$ points, which will represent the discretization of the interior domain of Ω. Each point (x_i, y_j) of the grid has an index $i \in \{1, \cdots, N_x - 1\}$ and $j \in \{1, \cdots, N_y - 1\}$. You can use the `numpy.meshgrid` function.

2. Recall the finite difference formula for the second order derivative in 1D and its approximation order. Apply the formula to the x and y directions.

3. Combine the two partial derivatives and write an explicit form of the discrete Laplacian in terms of the grid points. Use the notation shorthand $u_{ij} = u(x_i, y_j)$ for the unknowns. What simplification do you obtain with an uniform grid ?

4. Write the discrete Laplacian in a matrix form, and express the discrete Poisson equation as a linear system $\mathbb{A}\mathbf{u} = \mathbf{f}$ of size $(N_x - 1) \times (N_y - 1)$. To do so you need to order the unknowns to get a global unknown vector. For example, you can use the column ordering $\mathbf{u} = \mathbf{u}_j,\ j = \{1, N_y - 1\}$, where each $\mathbf{u}_j$ is a vector of size $(N_x - 1)$ such that

$$\mathbf{u}_j = (u_{0j}, u_{1j}, \cdots, u_{N_x-1,j})^T$$

Up to now we have not considered the boundary points. In any PDE solver we need to specify *boundary conditions*. Let us assume we use a homogeneous Dirichlet boundary condition everywhere, that is

$$u(x, 1) = u(x, 0) = u(1, y) = u(0, y) = 0, \quad (x, y) \in \partial\Omega.$$

Show that the boundary condition can be incorporated into the linear system as

$$\mathbb{A}\mathbf{u} = \mathbf{b} + \mathbf{f},$$

where you will specify the vector $\mathbf{b}$, of size $(N_y - 1)^2$. How would you extend the procedure for a non-zero boundary condition ?

1.2 Validation of the implementation

To validate the implementation we need to define an exact solution. To do so let us take

$$u_{\text{ex}}(x, y) = \sin^2(\pi x) \sin^2(\pi y)$$

Find by hand the corresponding right-hand-side $f(x, y)$ for this solution. Is the exact solution consistent with the boundary conditions ?

Given a number of grid points (start with a small number), call a linear system solver and compute the relative error in the maximum norm in the computational domain Ω. Show a convergence plot in log-log scale with respect to the step size. What is the expected convergence rate ? Do you observe any difficulty in terms of computational time ?

2 Solving the linear system

In this part we will analyze various ways to solve the linear system. We will compare the numerical cost of some direct and iterative methods.

2.1 Direct methods

In the first section we used a dense representation of the matrix, meaning that all the entries were stored. Implement instead a **sparse** representation of the matrix that stores only the non-zeros entries. In `Python` you can use the `scipy.sparse` library (look up at the documentation: `scipy.sparse.spdiags`, `scipy.sparse.kron`, `scipy.sparse.eye`, ...)

- Solve again the system using a sparse solver, and compare the computational time with the direct approach (i.e. solving the problem with a dense representation of the matrix). You may write the discrete Laplacian as

  $$\Delta_h = \mathbb{I} \otimes \mathbb{T} + \mathbb{T} \otimes \mathbb{I},$$

  where $\otimes$ is the Kronecker product, and $\mathbb{T}$ a matrix to be identified.

- Explain why it is advantageous to use a sparse matrix representation. As a justification, you may count the number of operations when doing a Gaussian elimination of a tridiagonal matrix.

2.2 Iterative methods

We shall now investigate the performance of some iterative methods. Before doing so you may find theoretically the eigenvalues of the discrete Laplacian in 2D by following these steps

- If $\mathbb{T}$ is of size N, show that its eigenvalues are $\lambda_i = 2 - 2\cos(\frac{\pi i}{N+1})$. (*Hint: find a recurrence formula for the eigenvalues and look again at the definition of Chebychev polynomials*)
- Use the Kronecker product form of Δ_h to deduce the eigenvalues $\lambda_{i,j}$ of the 2D discrete Laplacian (*Hint: look up on what does the Kronecker product to eigenvalues*)

We now turn into the implementation of iterative methods.

1. Implement the Jacobi and Gauss-Seidel methods, and validate their implementation thanks to a small test that you will choose.
2. Do you expect these methods to converge for the discrete Poisson problem ? Explain your answer by giving linear algebra arguments and by writing down the convergence radius. How will the convergence be affected when increasing the number of grid points ?
3. Explain how to measure the cost of an iterative solver (number of iterations **and** cost per iteration).
4. Implement the SOR method. Plot the number of iterations to reach a given residual (that you fix) as a function of the relaxation parameter ω. Is there an optimal parameter ? If yes, can you predict this value theoretically ?
5. Plot the decrease of the residual for different methods (Jacobi, Gauss-Seidel, SOR) and conclude on the computational performance of these iterative methods.

3 Extensions to the solver

You can continue the development of your solver by choosing one of the following extensions

1. Add a diffusion term to your solver such that the equation becomes

$$-\Delta u + \alpha u = f,$$

   where α is a real parameter. Solve again your problem with direct and iterative methods. What is going on for iterative methods when $\alpha < 0$? Explain the behaviour by an analysis of the eigenvalues.
2. Implement a higher order finite difference formula.
3. Implement the conjugate gradient iterative method, and compare its efficiency with the other iterative methods.
4. Implement non-homogeneous Dirichlet boundary conditions and/or Neumann boundary conditions thanks to the 2nd order centered difference scheme. Validate your solver by checking the convergence rate thanks to an exact solution that you will define.