Numerical methods
Computer project Final

Gleb Zhdanko

March 2025

General Information

The computations and visualizations were performed on a MacBook Pro (13-inch, 2020, Two Thunderbolt 3 Ports) equipped with a 1.4 GHz Quad-Core Intel Core i5 processor.

I used Python 3.12.4. The following Python libraries were utilized in the implementation:

- `numpy`: for numerical computations,
- `scipy`: for solving linear systems and other numerical methods,
- `matplotlib`: for creating high-quality visualizations.

I am using notation for last elements like in python: u_{-1} - means last element, u_{-2} - second element from the end and so on.

Problem 1

Implement the following iterative methods:

1. the point S.O.R. method with $\omega = 2/(1 + \pi h)$, and
2. the conjugate gradient method

in **matrix-free** (inplace) **operator** form to solve 2-D Poisson's equation

$$u_{xx} + u_{yy} = -2\cos(x)\sin(y)$$

on a unit square. The boundary conditions and the exact solution are given by the formula

$$u_{ex} = \cos(x)\sin(y).$$

Use the standard five-point difference scheme with $h = \Delta x = \Delta y = 0.1, 0.05, 0.025$, and 0.01. The initial iteration should be zero in the interior of the square. Stop the iterations when the changes in the solution as measured in the norm

$$||\mathbf{u}^{n+1} - \mathbf{u}^n||_2 = \left(\Delta x \Delta y \sum_{j,k} (u_{j,k}^{n+1} - u_{j,k}^n)^2 \right)^{1/2}$$

is less then 10^{-8}.

(a) For each method demonstrate the second order accuracy of the finite difference approximation by plotting the L_∞ error $||\mathbf{u}^n - \mathbf{u}_{ex}||_\infty = \max_{j,k} |u_{j,k}^n - u_e x(x_j, y_k)|$ of the converged solutions as a function of mesh spacing h.

(b) For each resolution plot the L_∞ error $||\mathbf{u}^n - \mathbf{u}_{ex}||_\infty = \max_{j,k} |u_{j,k}^n - u_{ex}(x_j, y_k)$ as a function of iterations n. Comment on the accuracy of the schemes, the efficiency of the methods, and the influence of the resolution on the number of iterations.

Solution:

0.1 SOR

The implementation of SOR for Gauss-Seidel is straight-forward. I have used code from previous project and added weighted factor ω. Since our BC is Dirichlet boundary condition, we can set them once, and then don't update in the main loop, since our code only update interior points. Next, we set a stop condition determined by the problem statement.

Figure 1: SOR Method

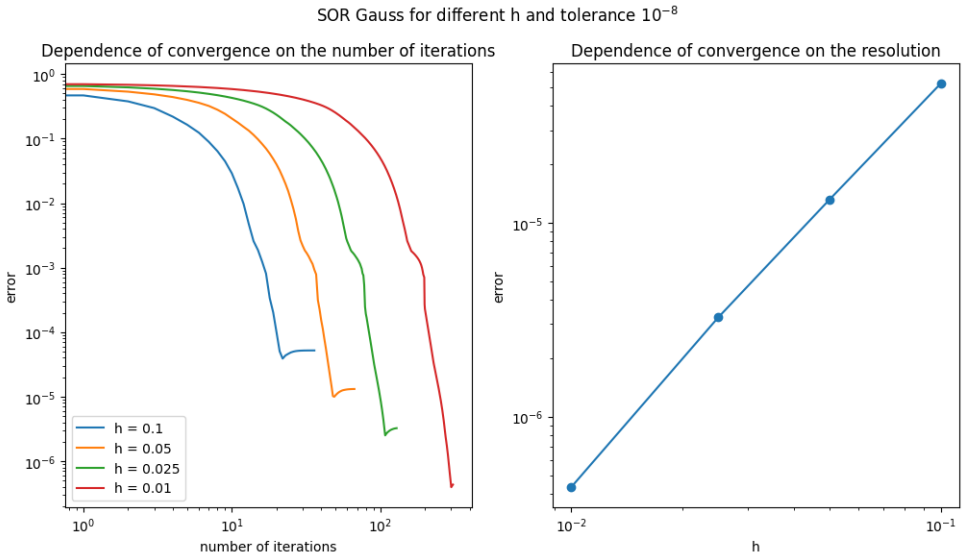
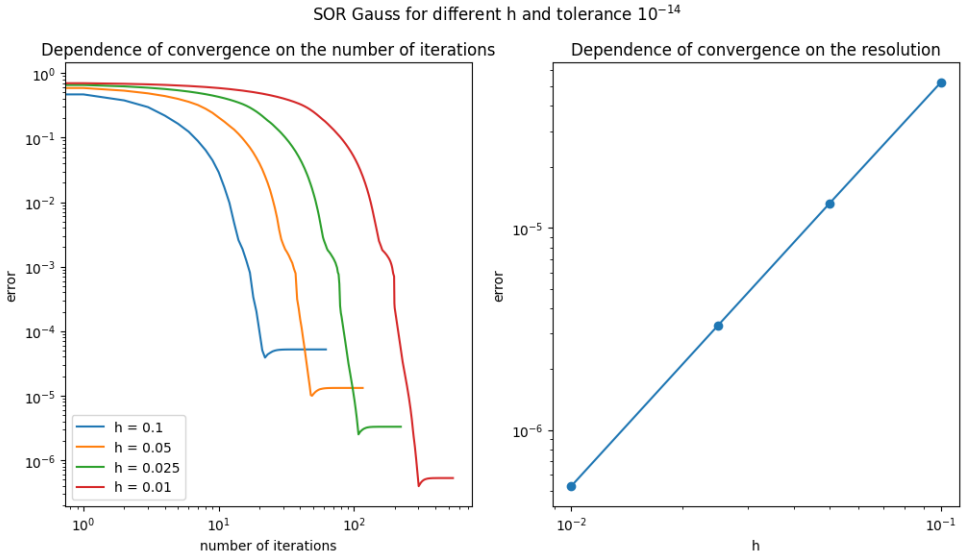


Figure 2: SOR Method (with smaller tolerance)



I have considered SOR method for higher tolerance, just to show that it goes on plateau after some iterations. Although, the difference between old and new solution become much smaller, the error of the numerical solution doesn't change much.

Here is the number of steps and order of accuracy for SOR with tolerance 10^{-8}:

- Order of convergence: 2.077973114016815
- Number of iterations for h = 0.1: 37
- Number of iterations for h = 0.05: 68
- Number of iterations for h = 0.025: 130
- Number of iterations for h = 0.01: 310

Here is the number of steps and order of accuracy for SOR with tolerance 10^{-14}:

- Order of convergence: 1.9945752696186692
- Number of iterations for h = 0.1: 63
- Number of iterations for h = 0.05: 117
- Number of iterations for h = 0.025: 223
- Number of iterations for h = 0.01: 540

I think this is due to the accuracy of our method. The accuracy of 5 point rule is $O(h^2)$. And no matter how small we make difference between new and old solution, the error couldn't decrease below some threshold.

Each time we decrease h by factor of 2, the number of iteration to "converge" doubled.

Order of convergence matches the theoretical order.

0.2 Conjugate gradient method

Remark: I took for granted that matrix A is symmetric positive definite.

The algorithm is following:

1. Let $p = 0$, $r^{(0)} = f - Av^{(0)}$ and $q^{(0)} = r^{(0)}$. For $p = 0, 1, \ldots,$ compute the vectors $v^{(p)}, r^{(p)}$, and $q(p)$ from:
2. $v^{(p+1)} = v^{(p)} + \alpha^{(p)} q^{(p)}$, where $\alpha^{(p)} = (r^{(p)})^T r^{(p)} / (q^{(p)})^T A q^{(p)}$,
3. $r^{(p+1)} = r^{(p)} - \alpha^{(p)} A q^{(p)}$,
4. $q^{(p+1)} = r^{(p+1)} + \beta^{(p)} q^{(p)}$, where $\beta^{(p)} = (r^{(p+1)})^T r^{(p+1)} / (r^{(p)})^T r^{(p)}$.

To implement conjugate gradient method, we need to understand our matrix A and how it acts on the vectors. Since we are using 5 points rule and dimensions of vectors (v, q) are the same, we can believe that action of A is similar for v and q vector. For numerical solution we know how our matrix A acts:

$$Av_{i,j} = \frac{v_{i+1,j} - 4v_{i,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1}}{h^2},$$

where v is a vector but with double symbols. For ease of implementation it's clear that we can keep v as a matrix, not vector. The issue with vector q and especially with calculating $q^T Aq$. First, $q^T Aq$ - is a scalar, moreover, it is just dot product between two vectors: q and Aq. Therefore, we can consider q as a matrix, and action of A on q will be the same as for v. $r^T r$ - is also scalar, and it's just a sum of all components of r squared, therefore, r can also be considered as matrix.

Now, using Pyhton syntax sugar the implementation of conjugate gradient method is straightforward and almost the same as algorithms provided earlier.

Similarly to SOR, Conjugate method goes on plateau. Number of steps and order of accuracy for Conjugate Gradient Method with tolerance 10^{-8}:

- Order of convergence: 1.954778737700221
- Number of iterations for h = 0.1: 28, complete convergence in 121
- Number of iterations for h = 0.05: 58, complete convergence in 441
- Number of iterations for h = 0.025: 112, complete convergence in 1681
- Number of iterations for h = 0.01: 275, complete convergence in 10201

Number of steps and order of accuracy for Conjugate Gradient Method with tolerance 10^{-16}:

- Order of convergence: 1.9945751706548134
- Number of iterations for h = 0.1: 40, complete convergence in 121
- Number of iterations for h = 0.05: 86, complete convergence in 441
- Number of iterations for h = 0.025: 174, complete convergence in 1681

Figure 3: Conjugate Gradient Method

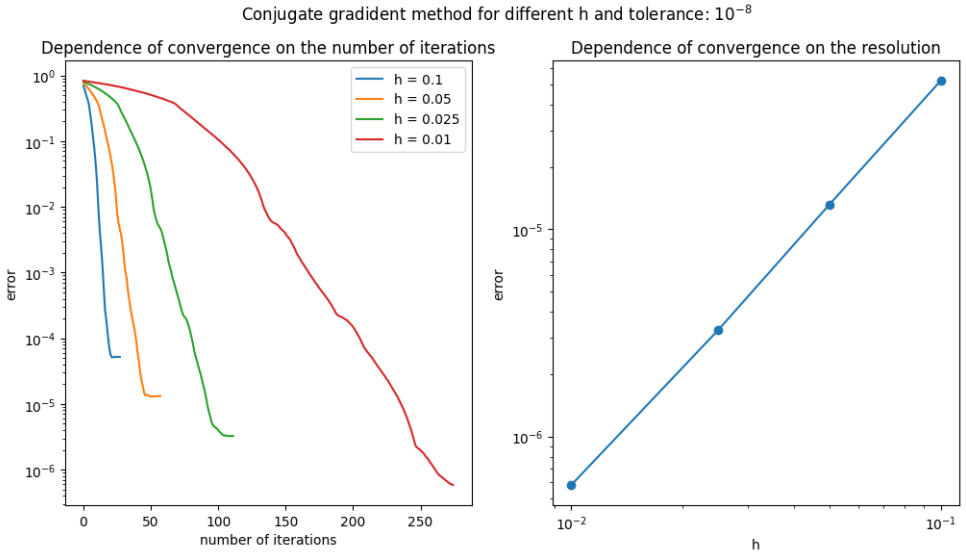
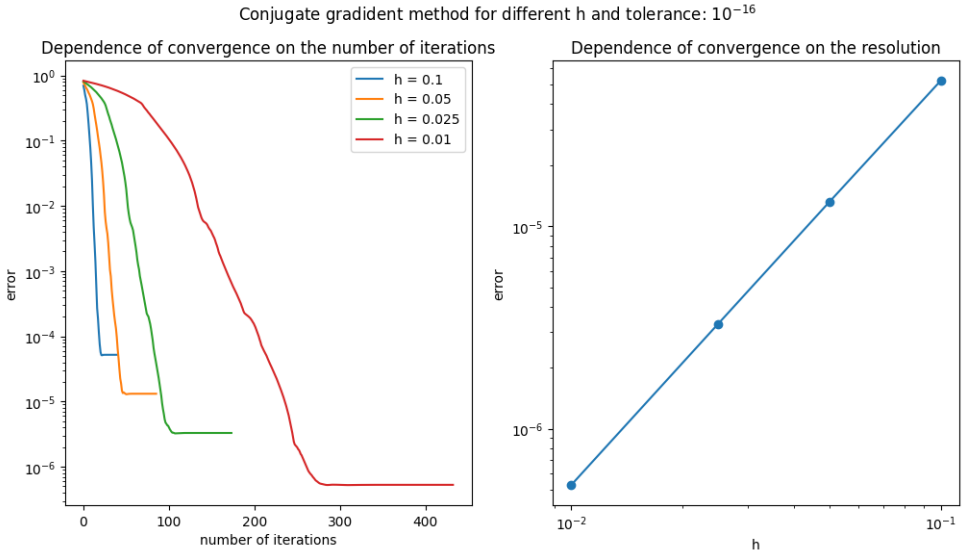


Figure 4: Conjugate Gradient Method (with smaller tolerance)



- Number of iterations for h = 0.01: 433, complete convergence in 10201

Similarly to SOR, the number of iterations to stop the algorithm is doubled each time we decrease h by factor of 2.

In general, Conjugate gradient method is faster than SOR.

Thank you very much for the course, it was very interesting to learn!