CS/MATH 375, Fall 2024 — HOMEWORK # 7
Due : Friday, Oct. 11th at 10:00pm on Canvas

Instructions

- **Report:** In general, your report needs to read coherently. That is, start off by answering question 1. Fully answer the question, and provide all the information needed to understand your answer. If Matlab code or output is part of the question, include that code or output (e.g., screenshot) alongside your narrative answer. If discussion is required for a question, include that. Overall, your report is your narrative explanation of what was done, your answers to the specific questions, and how you arrived at your answers. *Your report should include your Matlab scripts, code output, and any figures.*
- **What to hand in:** Submission must be one **single PDF** document, containing your entire report, submitted on Canvas.
- **Partners:** You are allowed to (even encouraged) to **work in pairs**. If you work with a partner, only one member of the group should need to submit a report. On Canvas, both partners should join a group (numbered 1 through 15). Then either member can upload the report for the entire group. Groups of more than 2 students are not allowed.
- **Typesetting:** If you write your answers by hand, then make sure that your handwriting is readable. Otherwise, I cannot grade it.
- **Plots:** All plots/figures in the report must be generated in Matlab or Python and not hand drawn (unless otherwise specified in the homework question).

  In general, make sure to (1) title figures, (2) label both axes, (3) make the curves nice and thick to be easily readable, and (4) include a legend for the plotted data sets. The font-size of all text in your figures must be large and easily readable.

Reading: Read chapter 2.5 in the book.

Iterative Solvers for the Poisson Equation

The Poisson equation, modeling a large number of physical phenomena (electro-statics, fluid flow, steady state temperature), is

$$\begin{aligned} -\nabla \cdot \nabla u(\mathbf{x}) &= f(\mathbf{x}) && \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= g(\mathbf{x}) && \mathbf{x} \in \partial\Omega, \end{aligned}$$

where $u(\mathbf{x})$ is the solution, $\Omega \subset \mathbb{R}^d$, $\partial\Omega$ is the boundary of Ω, and g is a function specifying the boundary conditions. For this homework, we will use the built-in Matlab function `gallery` to generate matrices associated with an approximation of the Poisson equation in 2D. To create a matrix, define P to be a positive integer, e.g.,

```
>> P = 10;
```

and then type

```
>> A = gallery('poisson',P);
```

Note that A is a sparse, banded, SPD, $n \times n$ matrix where $n = P^2$ (P denotes resolution in each spatial dimension). For the right hand side, we will simply use a vector of ones:

```
>> b = ones(P^2,1);
```

For the rest of the problems in this homework, use this matrix A and this vector $\vec{b}$ when testing your iterative solver methods.

"Vector" and "component'-wise" versions of the methods

In class, we saw that it is possible to write the Jacobi method multiple equivalent ways. One of them (which we'll use in this homework) is

$$\vec{x}^{(k)} = D^{-1}(C_L + C_U)\vec{x}^{(k-1)} + D^{-1}\vec{b}.$$

We will refer to this as the "vector update" version of Jacobi. We then went on to show that the "component-wise update" version of the Jacobi algorithm given by

$$x_i^{(k)} = -\sum_{j=1, j\neq i}^{n} \left(\frac{a_{ij}}{a_{ii}}\right) x_j^{(k-1)} + \frac{b_i}{a_{ii}}$$

is equivalent.
Similarly, we will call

$$x^{(k)} = (D - C_L)^{-1}C_U x^{(k-1)} + (D - C_L)^{-1}b,$$

the "vector update" version of Gauss-Seidel. The "component-wise update" is given by

$$x_i^{(k)} = -\sum_{j=1}^{i-1} \left(\frac{a_{ij}}{a_{ii}}\right) x_j^{(k)} - \sum_{j=i+1}^{n} \left(\frac{a_{ij}}{a_{ii}}\right) x_j^{(k-1)} + \frac{b_i}{a_{ii}}.$$

---

1. *Jacobi Algorithms*

(a) Write a Matlab function, `function x = my_jacobi(A,b, tot_it)`, which takes as input the matrix A, the vector $\vec{b}$, number of iterations `tot_it` and outputs the Jacobi solution after `tot_it` iterations. For this part, code up the component-wise version of the Jacobi method.

(b) Now create another function, `function x = my_vector_jacobi(A,b, tot_it)`, which also takes as input the matrix A, the vector $\vec{b}$, number of iterations `tot_it` and outputs the Jacobi solution after `tot_it` iterations. For this part, code up the vector-update version of the Jacobi method. Go ahead and "backslash" for D^{-1}. Verify that you get the same residuals and errors using the component-wise and vector-update versions with $P = 20$ and 100 iterations.

Hint: Use the `diag` function in Matlab to extract the main diagonal of the matrix: `diag(diag(A))`. You can also use `triu` to get the upper triangular portion and `tril` to get the lower triangular portion of A (with diagonals included).

(c) Which version of the Jacobi algorithm is faster? Compare the time it takes to perform 100 iterations with $P = 20$. What do you think is causing the speed difference?

(d) Now, we will investigate the error in the Jacobi solution by varying the size of the linear systems, and keeping the number of Jacobi iterations fixed at 100. We compare the error in the Jacobi solution with the "true solution" obtained from the Matlab "backslash" operator. Make a table showing how the 2-norm in the relative error in the Jacobi solution,

$$\frac{\|x_{\text{true}} - x\|_2}{\|x_{\text{true}}\|_2},$$

varies with increasing size of the linear system. Use $P = 10, 20, 40, 80, 160$ which corresponds to $n = 100, 400, 1600, 6400, 25600$. Use whichever version (component-wise or vector-update) of Jacobi is faster. Does the relative error increase or decrease as the system size increases?

2. *Comparing Algorithms*

(a) Create a function, `function x = my_gauss_seidel(A,b, tot_it)`, which also takes as input the matrix A, the vector $\vec{b}$, number of iterations `tot_it` and outputs the Gauss-Seidel solution after `tot_it` iterations. You may choose either the component-wise or the vector update version.

(b) Now investigate the performance of these two (Jacobi and GS) methods. Compute the error in the solutions obtained from Jacobi and GS as the number of iterations is varied for a fixed system size (use $P = 160$). Make a **single** table showing the 2-norm in the relative error as the number of iterations is

varied. Use `tot_it = 50, 100, 200, 400, 800, 1600`. This table will have three columns: Num Iterations, Error Jacobi, and Error GS. Comment on the performance of the methods in reducing the relative error.