CHEN 2450
HOMEWORK 4

ITERATIVE LINEAR SOLVERS

Submit all your homework using Jupyter notebooks. You are expected to include proper text and discussion for each and every problem including appropriate headings and formatting. You will lose points if your reports are not readable or just include code with a few print out statements. No exceptions.

Problem 1 (30 pts)

For this problem, submit a report with your work clearly shown. The report should be a typeset Jupyter Notebook (ipynb) with appropriate markup, headings, and equations.

Consider the following system of equations:

$$\begin{aligned} x+2y+z &= 1 \\ y-6z &= 2 \\ 10x-y+4z &= 6 \end{aligned}$$

with an initial guess of $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.

1. (5 pts) Write these in matrix form directly from the ordering given above and perform ***two*** iterations of the Jacobi method (by hand).
2. (5 pts) Rewrite these in a form that is diagonally-dominant and then show two iterations of the Jacobi method (by hand).
3. (10 pts) For the rewritten (diagonally-dominant) system, show two iterations of the Gauss-Seidel method (by hand).
4. (10 pts) Prepare a table that shows the L_2 norm for of the residual error, $\|b-Ax\|_2$, after the second iteration for each of the previous parts (1-3) of the problem and comment on your findings.

Problem 2 (70 pts)

Consider the steady-state heat transfer inside a metal rod held at fixed temperatures at both ends and immersed in air at a temperature of T_∞. The equation for the temperature distribution $T(x)$ in the rod is

$$\frac{\mathrm{d}^2T}{\mathrm{d}x^2} = h(T-T_\infty) \tag{1}$$

where h is a heat transfer coefficient (m^{-2}) that represents how fast heat is being added and/or removed by the air. The rod is fixed at both ends with fixed temperatures $T(x=0)=T_0$ and $T(x=L)=T_L$ at the left and right boundaries, respectively. The exact solution for this heat transfer problem is given by

$$T(x) = C_1 e^{-x\sqrt{h}} + C_2 e^{x\sqrt{h}} + T_\infty \tag{2}$$

with

$$C_1 = \frac{1}{e^{2L\sqrt{h}}-1}\left[T_0 - T_\infty + (T_0 - T_\infty)(e^{2L\sqrt{h}}-1) + (T_\infty - T_L)e^{L\sqrt{h}}\right] \tag{3}$$

$$C_2 = \frac{T_\infty - T_0 + (T_L - T_\infty)e^{L\sqrt{h}}}{e^{2L\sqrt{h}}-1} \tag{4}$$

Later in class we will learn how to derive numerical approximations for this problem. One numerical approximation results in the following system of linear equations

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & & 0 \\ 1 & -(2+h\Delta x^2) & 1 & 0 & \cdots & & 0 \\ 0 & 1 & -(2+h\Delta x^2) & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & 0 & 1 & -(2+h\Delta x^2) & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 & -(2+h\Delta x^2) & 1 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} T_0 \\ T_2 \\ T_3 \\ \vdots \\ T_{n-2} \\ T_{n-1} \\ T_{n-1} \end{pmatrix} = \begin{pmatrix} T_o \\ -h\Delta x^2 T_\infty \\ -h\Delta x^2 T_\infty \\ \vdots \\ -h\Delta x^2 T_\infty \\ -h\Delta x^2 T_\infty \\ T_L \end{pmatrix}. \tag{5}$$

where, as usual, Δx is the spacing between grid points. Recall that for n points there are $n-1$ intervals so that $\Delta x = L/(n-1)$. The purpose of this problem is to analyze how the Jacobi and Gauss-Siedel algorithms behave when solving this system of equations. Using $n = 100$ points (*i.e.*, 100 equations) $L = 1$, $T_0 = 500$, $T_L = 600$, $T_\infty = 300$, and $h = 100.$, answer the questions below. As usual, report all your results and code in your Jupyter notebook.

1. (40 pts) In this part you will compare the numerical solution to the analytical solution.

   (a) (10 pts) Write a function in Python that implements the exact solution given by (2). Then, use that function to plot the analytical solution on the interval $x \in [0,1]$. Use 100 points for the x-axis.

   Hint: The function should look like: exact_temperature (x). Use linspace to generate the points x (e.g. x = numpy.linspace (0,1,100) . This will generate 100 equally-spaced points starting from 0 to 1).

   (b) (10 pts) Solve the system of equations in (5) using the Jacobi and Gauss-Seidel methods, using a tolerance of $\varepsilon = 0.5$ based on the absolute error of $\|Ax-b\|_2$.

(c) (10 pts) Plot your numerical solutions along with the analytical solution, and report the number of iterations required for each method to achieve the prescribed error tolerance. Use an initial guess of $T_i = 500$, $i = 1 \ldots n$.

(d) (10 pts) Comment on the adequacy of this choice for ε. In this case, is it more convenient to choose convergence criteria based on $\|Ax - b\|$ or based on $\left\|T^{k+1} - T^k\right\|$, where k is the iteration count?

2. (30 pts) In this part, we will look at how the error varies as we take more iterations with the Jacobi and Gauss-Seidel methods as well as how the number of iterations changes between both algorithms for a given tolerance.

(a) (15 pts) Plot the error norm $\|Ax - b\|_2$ as a function of the number of iterations that you take with the Gauss-Seidel and Jacobi methods, again using $T_i = 500$ as an initial guess for temperature and $n = 100$ points (i.e. 100 equations). Use a log-log plot and show results over a range from 10-10^3 iterations. Comment on what you observe.

(b) (15 pts) Plot the number of iterations vs tolerance that you take with the Gauss-Siedel and Jacobi methods, again using $T_i = 500$ as an initial guess for temperature and $n = 100$ points (i.e. 100 equations). Use a log-log plot and show results over a range from $10^{-3} - 0.1$ error tolerance. Comment on what you observe.

Hints:

- You will probably want to test this at low iteration counts before going up to 10^3 iterations, which may take a few minutes to run.
- For question 2.(a) if you use the Jacobi and Gauss-Siedel functions that we developed in class to help with this, you need to use a very small tolerance (*e.g.*, 10^{-99}) so that you never converge and it will just take the specified number of iterations.
- Conversly, for question 2.(b), if you use the function we developed in class, make sure you set a very large number of iterations for MaxIter to make sure you stop iterating at the required tolerance.
- You will have access to Jacobi and Gauss-Siedel functions in a Python file on Canvas. You can copy the code into your jupyter notebooks and call those functions. Alternatively, place the Python file in the directory of your jupyter nobook and use the usual module import commands: from iterativesolvers import jacobi, gauss_siedel.