AMATH 301
Homework 4
Due: Friday, February 1, 2019

In this homework assignment you will use three different iterative methods to solve the same linear system. The intention is that you recognize the similarities between the methods and re-use code where appropriate. You will see the Jacobi and Gauss-Seidel iterative methods in the video lectures and in class. The successive over-relaxation method (SOR) is not covered in class, but is very similar to the other two methods.

Problem Statement: Ax=b

Consider the system $A\vec{x} = \vec{b}$, where $A \in \mathbb{R}^{N\times N}$ is a square matrix with -2's on the main diagonal, 1's directly above and below the diagonal, and 0's everywhere else. For $N = 5$, the matrix is given below,

$$\begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

The vector $\vec{b} \in \mathbb{R}^N$ has elements of the form

$$b_k = \cos(5t_k) + \frac{1}{2}\sin(7t_k)$$

where $\vec{t} \in \mathbb{R}^N$ is a column vector of linearly spaced points between 0 and π.

Problem 1 (Scorelator).

Learning goal: Use the `diag` *command to build a large matrix.*

Build the matrix A and vectors $\vec{b}$ and $\vec{t}$ in MATLAB for $N = 30$. Use the `diag` command to accomplish this; `diag` is a versatile command that can be used in several ways, and you will have to use it several times to build A.

Compute the 'true solution' $\vec{x}$ to the system $A\vec{x} = \vec{b}$ for comparison in later problems.

Save A to the file **A1.dat**, the vector $\vec{b}$ to the file **A2.dat**, and the true solution $\vec{x}$ to **A3.dat**.

Jacobi Iteration

Jacobi iteration can be written succinctly as starting with an initial guess x_0, and calculating iterates according to the formula

$$\vec{x}_{k+1} = -D^{-1}T\vec{x}_k + D^{-1}\vec{b}.$$

Here $A = D + T$, with D a diagonal matrix whose entries match the diagonal entries of A, and T containing all remaining entries of A.

We can write the Jacobi iteration even more succinctly by defining

$$M = -D^{-1}T \qquad\qquad g = D^{-1}\vec{b},$$

so that

$$\vec{x}_{k+1} = M\vec{x}_k + g.$$

Problem 2 (Scorelator).

Learning goal: Use the `diag`*, or* `triu` *and* `tril` *commands. Learning goal: Implement the Jacobi iterative method for solving $Ax = b$.*

Define the matrices D and T in MATLAB. Look up and use the `diag` command to define D. Use D, T, and b to define M and g for this problem.

Implement Jacobi iteration to solve the problem $A\vec{x} = \vec{b}$. Use an initial guess of all zeros and iterate for 400 iterations. Save the 2-norm of the error of each iterate, $\|\vec{x}_{\text{true}} - \vec{x}_{\text{jacobi}}\|_2$, to a save vector.

Note: In this homework I am not counting the initial guess as an iterate.

Save the final iterate to file as **A4.dat**. Save the errors of the iterates to file as **A5.dat**.

Problem 3 (Scorelator).

Learning goal: Use the `eig` *command to obtain the eigenvalues of a matrix.*

Use the `eig` command (and any other commands you need) to obtain $\lambda_{\max}$: the eigenvalue of the Jacobi iteration matrix M with the largest absolute value. Save this eigenvalue to the file **A6.dat**.

Problem 4 (Writeup).

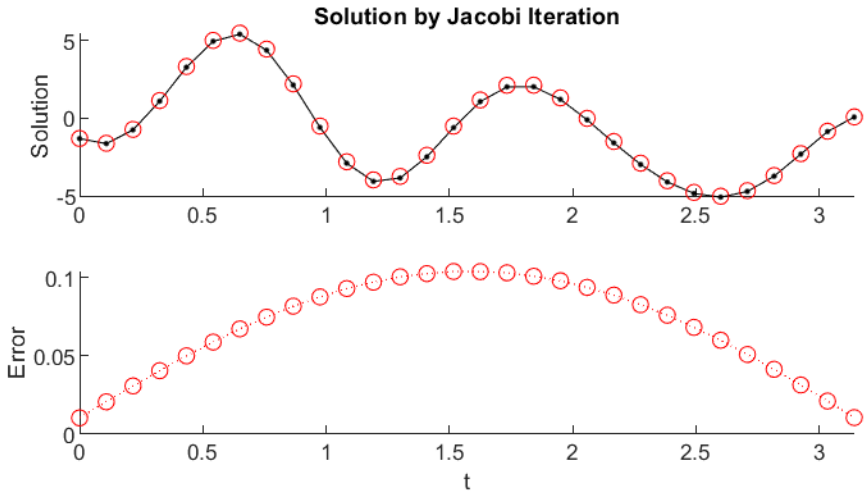
Learning goal: Create a figure that compares a true and approximate solution, and shows the error of the approximation.

Create a figure showing the solution you obtained through Jacobi iteration along with the true solution, as well as the error of the Jacobi solution.

Your figure should consist of two plots, made with the `subplot` command, to split the figure into an upper and lower axes. The upper window should show the true solution in black with small dots connected by solid lines. The Jacobi solution should be plotted as red circles. Both solution should be plotted against the t vector from Problem 1. Plot the error in the lower axes as red circles connected by dotted lines. Set the limits of horizontal the axes to span the interval $[0, \pi]$.

Label the axes and title the upper axes.

Save your figure to file as `solution_jacobi.png`. Your result should look like the one below.



Problem 5 (Writeup).

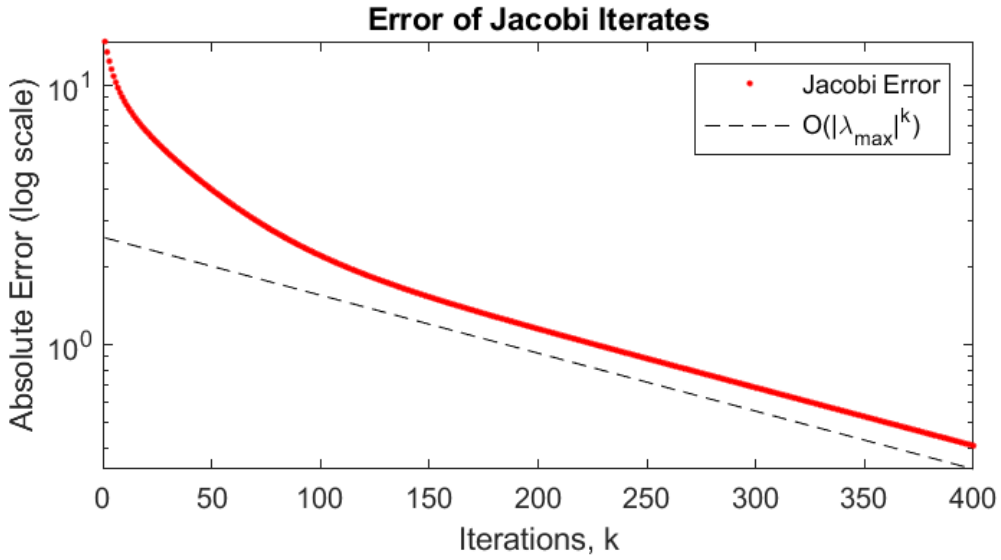
Learning goal: Predict the rate of convergence of an iterative method using eigenvalue analysis.

Learning goal: Create a figure that shows how quickly an iterative method converges.

Create a figure showing the error of the Jacobi iterates. Use the vector of errors you created in Problem 2, and plot the errors as red dots.

Add a plot showing the decaying trend $|\lambda_{\max}|^k$ against iterate number k, where $\lambda_{\max}$ is the largest-magnitude eigenvalue you calculated in Problem 3. You should see that after several hundred iterations, the error begins to decrease at a rate that matches $|\lambda_{\max}|$. Add appropriate titles and labels to the axes.

Save your figure to file as `errors_jacobi.png`. Your plot should look like the one below.



Gauss-Seidel Iteration

Learning goal: Relate and compare Gauss-Seidel and Jacobi iterative methods.

Gauss-Seidel is similar to Jacobi but instead of splitting the matrix $A = D + T$, we split it as

$$A = S + T,$$

where S is the lower-triangular part (including the main diagonal) of A, and T is the remaining part. The iteration is then

$$\vec{x}_{k+1} = -S^{-1}T\vec{x}_k + S^{-1}\vec{b},$$

or

$$\vec{x}_{k+1} = M\vec{x}_k + g, \qquad M = -S^{-1}T \qquad g = S^{-1}b.$$

Problem 6 (Scorelator).

Repeat the procedures from Problem 2–3 but now use Gauss-Seidel iteration instead of Jacobi iteration. Save the solution you obtain from 400 iterations of Gauss-Seidel to the file **A7.dat**. Save the vector of iterate errors to the file **A8.dat**.

Save the largest-magnitude eigenvalue $\lambda_{\max}$ of the Gauss-Seidel iteration matrix M to the file **A9.dat**.

Problem 7 (Writeup).

Create a figure like the one from Problem 4 that showcases the solution obtained with Gauss-Seidel iteration. Use the color green instead of red when plotting Gauss-Seidel iterations.

Save your figure to file as `solution_GS.png`

Problem 8 (Writeup).

Create a figure like the one from Problem 5 showing the error of the iterates from the Gauss-Seidel iteration you performed.

Save your figure to file as `errors_GS.png`.

Successive Over-relaxation (SOR)

Learning goal: Implement a new method that strongly resembles examples a known example.

Successive Over-relaxation (SOR) is similar to Gauss-Seidel. We split the matrix

$$A = L + D + U,$$

where D is the diagonal part of A, L is the lower-triangular part (excluding the main diagonal) of A, and U remaining upper-triangular part (again excluding the main diagonal). The iteration is then

$$\vec{x}_{k+1} = (D + \omega L)^{-1} \left(\omega \vec{b} - (\omega U + (\omega - 1)D)\vec{x}_k \right)$$

where ω is a parameter that we can choose. In practice, we *tune* the parameter to obtain the best rate of convergence.

The iteration can be written as

$$\vec{x}_{k+1} = M\vec{x}_k + g, \qquad M = -(D + \omega L)^{-1}(\omega U + (\omega - 1)D) \qquad g = (D + \omega L)^{-1}(\omega b).$$

Problem 9 (Scorelator).

Repeat the procedures from Problem 2–3 but now use Successive Over-relaxation with $\omega = 1.5$ instead of Jacobi iteration. Save the solution you obtain from 400 iterations to the file **A10.dat**. Save the vector of iterate errors to the file **A11.dat**.

Save the largest-magnitude eigenvalue $\lambda_{\max}$ of the SOR iteration matrix M to the file **A12.dat**.

Problem 10 (Writeup).

Create a figure like the one from Problem 4 that showcases the solution obtained with Gauss-Seidel iteration. Use the color green instead of red when plotting Gauss-Seidel iterations. In The title of the plot, state the value of the parameter ω so that the reader knows what parameter values were used.

Save your figure to file as `solution_SOR.png`

Problem 11 (Writeup).

Create a figure like the one from Problem 5 showing the error of the iterates from the Gauss-Seidel iteration you performed. In The title of the plot, state the value of the parameter ω so that the reader knows what parameter values were used.

Save your figure to file as `errors_SOR.png`.

Problem 12 (Scorelator).

Learning goal: Predict the performance of an iterative method based on eigenvalue analysis.

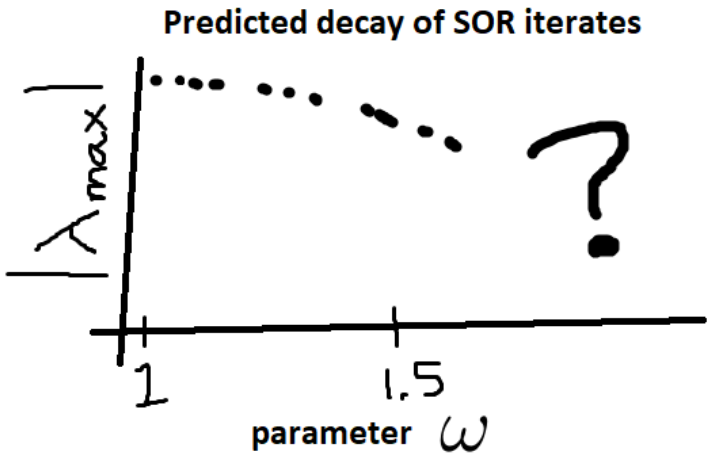
Successive over-relaxation with $\omega = 1.5$ produces better results than Jacobi or Gauss-Seidel, but it is possible that a different value of ω would lead to even better convergence. Use a loop to find the values of the largest-magnitude eigenvalue $\lambda_{\max}$ of the SOR iteration matrix for values of $\omega = 1.0, 1.01, 1.02, \ldots, 1.9$ (91 total values). Save these values into a vector (of length 91), and save this vector to file as **A13.dat**.

Problem 13 (Writeup).

Learning goal: Illustrate how the rate of convergence of a method varies as a parameter is tuned.

Plot the values of $|\lambda_{\max}|$ versus ω that you found in the previous problem. Label your axes and include a title.

Save your figure to file as `SOR_rates.png`. Your plot should look something like the following.



Problem 14 (Scorelator).

Learning goal: Tune a method parameter to achieve the best performance.

Repeat the procedure from Problem 2–3 one more time, using SOR with ω set equal to the best value that you found in the previous problem. The 'best' value of ω should produce the *smallest* $|\lambda_{\max}|$ eigenvalue.

Save the solution you obtain from 400 iterations to the file **A14.dat**. Save the vector of iterate errors to the file **A15.dat**.

Save the largest-magnitude eigenvalue $\lambda_{\max}$ of the SOR iteration matrix M to the file **A16.dat**.

Problem 13 (Writeup).

Create a figure like the one from Problem 5 showing the error of the iterates from the SOR method with the optimal value of ω that you found.

Save your figure to file as `errors_SOR_optimal.png`.