Overview

The pedagogical purpose of this project is to

1. give you practice implementing a numerical linear algebra algorithm, so that you better understand the algorithm through practice as well as theory, and
2. demonstrate how numerical linear algebra and multivariable calculus are useful in solving computational problems.

This project consists of two parts. The first coding part is worth 56 points, the second writing part is worth 48 points. Extra 6 points will be considered for quality of discussion.

You may work in teams of **up to three** persons. List all names of your group members in a `team_members.txt` file, an upload it on the Dropbox on T-Square of *one* of your team members by **March 29**.

Code for your project should be submitted in either Java, Python, or MATLAB (or its free version FreeMat). (Choose only one of these languages to use for the whole project—do not mix and match.) Your final deliverable should be submitted in a single `.zip` archive file. The archive file should be uploaded to the Dropbox on T-Square of *one* of your team members by **11:59 p.m.** on **April 18**. The file should contain:

- A `team_members.txt` file listing the name of each person in the project team.
- A `.doc`, `.docx`, or `.pdf` file for each part of the project you submit, containing the written component of the project.
- A `readme.txt` file for each part of the project you submit, explaining how to execute that part of the project, use your code to read the test files (in the form *.dat), and input the variables. Further, it should tell graders how to read the output results and find where are they.
- For Java or Python users, do not submit a project that has to be run on an IDE. The graders should be able to at least initialize your project on a command line interface, either Linux or Windows.

While you may use built-in or external libraries of classes and objects for matrix and linear algebra (e.g. `numpy`), you may only use built-in/library functions and methods that do the following:

- Create/instantiate/initialize vectors and matrices
- Add, subtract, and multiply vectors and matrices
- Take the dot product of two vectors
- Transpose a matrix or vector
- The inverse of a matrix

You must program any other linear algebra functionality you wish to use yourself, including (but not limited to) procedures which

- Find the LU-factorization of a matrix
- Find the QR-factorization of a matrix
- Find the solution of triangular systems with backward (forward) substitution
- Find the determinant of a matrix
- Find the trace of a matrix
- Find the eigenvalues and eigenvectors of a matrix
- Rotate, reflect, or project a vector

Using built-in or external library functions or using code copied from an external source (e.g. the Internet) for these operations will result in significant penalties on the relevant portion of the project. Moreover, submitting copied code *without proper credit due* will be considered plagiarism (see below).

Academic Honor Code Warning

You may not share code outside of your team. Code that appears improperly shared will be submitted to the Office of Student Integrity for investigation and possible sanctions. Additionally, code that appears copied from an external source and does not have a proper citation will be considered plagiarism and will also be referred to the Office of Student Integrity. **Disciplinary sanctions could include a grade of 0 on the project, an additional loss of a letter grade in the class, official notation of academic dishonesty on your transcript, and possible suspension or expulsion from the Institute.**

Definition of Norm in practice

In practice, to calculate the error , we use the maximum norm of the vector and matrix:

$$||\vec{x}||_\infty = \max_{1\leqslant i\leqslant n} |x_i|, \qquad ||A||_\infty = \max_{1\leqslant i,j\leqslant n} |A_{ij}|.$$

Example:

$$||\begin{pmatrix} 2 \\ -4 \\ 1 \end{pmatrix}||_\infty = \max_{1\leqslant i,j\leqslant n}\{|2|, |-4|, |1|\} = 4,$$

$$||\begin{pmatrix} 2 & 0 \\ -3 & 1 \end{pmatrix}||_\infty = \max_{1\leqslant i,j\leqslant n}\{|2|, |0|, |-3|, |1|\} = 3.$$

Required Codes

The maximum size of A is 20×20 in this project.

(a) (8 pts) Implement the LU decomposition method for a $n \times n$ matrix A. Name the procedure `lu_fact`. Your program should return the matrices L and U, and the error $\|LU - A\|_\infty$.

(b) (16 pts) Implement two versions of a procedure to compute a QR-factorization of a $n \times n$ matrix A. The first version should use Householder reflections. The second version should use Givens rotations. Name the procedures `qr_fact_house` and `qr_fact_givens`. In each version, your program should return the matrices Q, R, and the error $\|QR - A\|_\infty$.

(c) (8 pts) Implement the procedures named `solve_lu` to obtain the solution to a system $A\vec{x} = \vec{b}$ by the LU-factorization. The inputs to the procedure should be a $n \times (n+1)$ argument matrix $\{A|\vec{b}\}$. The outputs should be a $n \times 1$ vector solution $\vec{x}$ and the error $\|A\vec{x} - \vec{b}\|_\infty$. IMPORTANT: the use of an inverse matrix function should be avoided, your program should use backward or forward substitution; using an inverse matrix defects the purpose of these methods (Why?).

Also, name `solve_qr_house` and `solve_qr_givens` for $QR-$factorizations.

(d) (16 pts) Implement a procedure named `jacobi_iter` that uses the Jacobi iterative method to approximate the solution to the system $A\vec{x} = \vec{b}$. The inputs to the procedure should be

- a $n \times (n+1)$ argument matrix $\{A|\vec{b}\}$ (the algorithm should work for $n \geq 2$).
- a $n \times 1$ vector $\vec{u}_0$ of n floating-point real numbers that serves as the initial guess for the solution of $A\vec{x} = \vec{b}$.
- a tolerance parameter ε (a positive floating-point real number) that determines when the approximation is close enough, that is, iteration process stops when $\|\vec{x}_n - \vec{x}_{n-1}\|_\infty \leq \varepsilon$.
- a positive integer M giving the maximum number of times to iterate the method before quitting.

The outputs should be

- the approximate solution $\vec{x}_N$, the number of iterations N, and the error $\|A\vec{x}_N - \vec{b}\|_\infty$.
- If the procedure iterates M times and has not attained an answer with sufficient accuracy, the output should instead be a value representing a failure (for example, `None` in Python). Outputs should also include its iteration number M and the error $\|A\vec{x}_M - \vec{b}\|_\infty$.

Also, name `gs_iter` for Gauss-Seidel iterative method.

(e) (8 pts) Implement a procedure named `power_method` that uses the power method to approximate an eigenvalue and associated eigenvector of an $n \times n$ matrix. The inputs to the procedure should be

- a $n \times n$ matrix A with floating-point real numbers as entries (the algorithm should work for $n \geq 2$).
- a $n \times 1$ vector $\vec{u}_0$ of n floating-point real numbers that serves as the initial guess for an eigenvector of A.
- a $n \times 1$ vector $\vec{w}$ of n floating-point real numbers that serves as auxiliary vector.
- a tolerance parameter ε (a positive floating-point real number) that determines when the approximation is close enough, that is, iteration process stops when $\|\vec{u}_n - \vec{u}_{n-1}\|_\infty \leq \varepsilon$.
- a positive integer M giving the maximum number of times to iterate the power method before quitting.

The outputs should be

- the approximate largest eigenvalue λ_N in view of absolute value, the corresponding eigenvector $\vec{u}_N$, and the iteration number N.
- If the procedure iterates M times and has not attained an answer with sufficient accuracy, the output should instead be a value representing a failure (for example, `None` in Python). Outputs should also include its iteration number M.

Application problems and the writing reports

1 The Hilbert Matrix

(16 pts) The Hilbert matrix is a square matrix with entries being the unit fractions $H_{ij} = \frac{1}{i+j-1}$. For instance, the 4×4 Hilbert matrix is

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}.$$

The objective of this part of the project is to solve the linear system $H\vec{x} = \vec{b}$, for different dimensions n, with different solution methods. This exercise will lead you to a comparison of some of the numerical methods covered in class methods.

For each $n = 2, 3, \ldots, 20$, consider the $n \times n$ Hilbert matrix H. Let $\vec{b} = 0.1^{n/3}$, or $\vec{b} = \begin{bmatrix} 0.1^{n/3} \\ 0.1^{n/3} \\ \vdots \\ 0.1^{n/3} \end{bmatrix}$.

Your job is to

(a) For each $n = 2, 3, \ldots, 20$, solve the system $H\vec{x} = \vec{b}$, where $\vec{b} = 0.1^{n/3}$. For each n, using `solve_lu`, `solve_qr_house`, and `solve_qr_givens` to obtain the solution, your program should output the solution $\vec{x}_{sol}$, and the error $\|LU - H\|_\infty$, $\|QR - H\|_\infty$, and $\|H\vec{x}_{sol} - \vec{b}\|_\infty$. The output should be easily readable on the screen or a text file.

(b) Summarize your findings by plotting the errors obtained as a function of n, for each of the methods. The plot can be done using your own code, Excel, or any graphing program. The plots should be included in the written component.

(c) Answer the following questions in the associated written component for this part of the project:

(i) Why is it justified to use the LU or $QR-$factorizations as opposed of calculating an inverse martix?

(ii) What is the benefit of using LU or $QR-$factorizations in this way? (Your answer should consider the benefit in terms of conditioning error.)

---

2 Convergence of the iterative methods

(16 pts) In this part, you will use both Jacobi and Gauss-Seidel method to obtain the approximate solution to a system $A\vec{x} = \vec{b}$.

I. 3×3 symmetric matrix $A = \begin{pmatrix} 1 & 1/3 & 1/9 \\ 1/3 & 1 & 1/3 \\ 1/9 & 1/3 & 1 \end{pmatrix}$ and 3×1 vector $\vec{b} = \begin{pmatrix} 0.9 \\ 0.1 \\ 0.3 \end{pmatrix}$.

In GaussSeidel method, you may use this formula for the inverse of lower triangle $S = \begin{pmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{pmatrix}$, the inverse S^{-1} is $\begin{pmatrix} 1/a & 0 & 0 \\ -b/ac & 1/c & 0 \\ (-cd+be)/acf & -e/cf & 1/f \end{pmatrix}$ where $a, c, f \neq 0$.

Your job is to

(a) Randomly generate at least one hundred 3×1 initial vectors $\vec{x}_0$. The entries of the vectors should be floating-point real numbers uniformly distributed in the interval $[-10, 10]$. For each $\vec{x}_0$

- Record $\vec{x}_0$.
- Use both `jacobi_iter` and `gs_iter` to find the approximation solution $\vec{x}_N$ of $A\vec{x} = \vec{b}$ within a tolerance of 5 decimal places ($\|\vec{x}_n - \vec{x}_{n-1}\|_\infty \leq \varepsilon = 0.00005$).

- Use a maximum of $M = 100$ iterations before quitting in failure.
- Record $\vec{x}_N$ and number of iterations N in both methods.

(b) Average these 100 $\vec{x}_N$ from (a) to get an approximation solution $\vec{x}_{approx}$. Compute the error of this approximation to the exact solution $\vec{x}_{exact} = \begin{pmatrix} 39/40 \\ -13/40 \\ 12/40 \end{pmatrix}$, that is $\|\vec{x}_{approx} - \vec{x}_{exact}\|_\infty$, for both iterative methods.

(c) For each $\vec{x}_0$, evaluate the ratio of iteration steps N between Jacobi and Gauss-Seidel methods, that is $N_{Jacobi}/N_{Gauss-Seidel}$, then average these 100 ratios.

(d) Using your results from (a), plot two colored scatterplots in the same figure. The x-axis is the initial error $\|\vec{x}_0 - \vec{x}_{exact}\|_\infty$, and y-axis is the iterations N associated with this initial data $\vec{x}_0$. Use black scatters for Jacobi results, and blue for Gauss-Seidel results.

(e) In the written component for this part, interpret your results in part (b),(c) and graphs from part (d). Especially explain why the graphs look the way they do. (Your answer should consider the spectral radius of the matrix $S^{-1}T$, see lecture notes for the definition of S, T.)

II. Both Jacobi and Gauss-Seidel methods do not work for this 2×2 symmetric matrix $A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$. Let's set $\vec{b} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$, then the exact solution $\vec{x}_{exact} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. The sequence obtained from the method converges only when we luckily pick the exact solution + a vector from null space of $S^{-1}T$ as the initial vectors $\vec{x}_0$.

Your job is to

(f) Pick a 2×1 initial vectors $\vec{x}_0$ other than $\begin{pmatrix} * \\ 1 \end{pmatrix}$, where $*$ is any real number, for example, $\vec{x}_0 = \vec{0}$. Set $\varepsilon = 0.00005$. For each $M = 2, 3, \ldots, 10$, record its error $\|A\vec{x}_M - \vec{b}\|_\infty$ in both `jacobi_iter` and `gs_iter` to obtain 9 successive errors $\|A\vec{x}_2 - \vec{b}\|_\infty, \|A\vec{x}_3 - \vec{b}\|_\infty, \ldots, \|A\vec{x}_{10} - \vec{b}\|_\infty$.

(g) Using your results from (f), plot two colored scatterplots in the same figure. The x-axis is the integer M, and y-axis is the error $\|A\vec{x}_M - \vec{b}\|_\infty$. Use black scatters for Jacobi results, and blue for Gauss-Seidel results.

(h) In the written component, interpret your results from the graphs in (g). Especially explain why the method doesn't work. (Your answer should consider the spectral radius of the matrix $S^{-1}T$, see lecture notes for the definition of S, T.)

---

3 Convergence of the Power Method

(16 pts) I. In this part you'll investigate the convergence of the power method for calculating eigenvalues and eigenvectors of randomly generated 2×2 matrices. You may use this formula for finding the inverse of A, $A^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

Definition. The trace of A is the sum of the diagonal entries, that is

$$\text{trace}(A) = a_{11} + a_{22}$$

, by theorem, $\text{trace}(A) = \lambda_1 + \lambda_2$. Also, by theorem, the determinant of $A = \lambda_1 * \lambda_2$.

Your job is to

(a) Generate at least two hundred 2×2 matrices. The entries of the matrices should be floating-point real numbers uniformly distributed in the interval $[-2, 2]$. For each matrix A, find its inverse A^{-1}. (In the unlikely event A^{-1} doesn't exist, throw out A, generate a new random 2×2 matrix, and restart with this newly generated matrix.)

- Use `power_method` to find the largest eigenvalue of A in view of absolute value within an accuracy of 5 decimal places ($\varepsilon = 0.00005$). Use a maximum of $M = 100$ iterations before quitting in failure. Pick $\vec{u}_0 = \vec{w} =< 1, 0 >^t$, in the case of $< 1, 0 >^t$ doesn't work, try $< 0, 1 >^t$ or $< 1, 1 >^t$.
- Use `power_method` on A^{-1} to find the smallest eigenvalue of A in view of absolute value within an accuracy of 5 decimal places ($\varepsilon = 0.00005$). Use a maximum of $M = 100$ iterations before quitting in failure. Pick $\vec{u}_0 = \vec{w} =< 1, 0 >^t$, in the case of $< 1, 0 >^t$ doesn't work, try $< 0, 1 >^t$ or $< 1, 1 >^t$.

  Notice that the $\lambda_{max}(A^{-1}) = \frac{1}{\lambda_{min}(A)}$, make sure you got the correct eigenvalue the problem asked.
- Record the trace and determinant of A, and also record the number of iterations needed for the runs of `power_method` on A and on A^{-1}.

(b) Using your results from (a), plot two color-coded scatterplots. The x-axis of the plot should be the determinant, the y-axis should be the trace. Plot the determinant and trace of each matrix from (a) as a point, and color the point based on the number of iterations needed in `power_method` for that matrix. The second scatterplot should do the same, except coloring the point based on the number of iterations needed in `power_method` for the inverse of each matrix.

(c) In the written component for this part, interpret your graphs from part (b). Especially explain why the graphs look the way they do.

II. In this part you'll investigate how to compute the middle eigenvalue of a 3×3 matrix numerically. $A = \begin{pmatrix} -2 & 1 & 2 \\ 0 & 2 & 3 \\ 2 & 1 & -2 \end{pmatrix}$.

The largest and smallest eigenvalues in view of absolute value are -4 and -1, which can be computed as what we did in part I. Pretend we don't know 3 is the other eigenvalue.

Your job is to

(d) Use `power_method` on $(A-pI)^{-1}$ to find the the middle eigenvalue in view of absolute value. Choose two different p values as $p_1 = -(\frac{1+4}{2})$ and $p_2 = \frac{1+4}{2}$. Set $\varepsilon = 0.00005$ and maximum $M = 100$, $\vec{u}_0 = \vec{w} =< 1, 0, 0 >^t$.

(e) In the written component, interpret your results from part (d). Especially explain why the largest eigenvalue of $(A-p_2I)^{-1}$ converges to $(3-p_2)^{-1}$ but the largest eigenvalue of $(A-p_1I)^{-1}$ does not converge to $(3-p_1)^{-1}$.

---

testing data: vector or matrix with floating-point numbers.

1_H.dat : 4 by 4 matrix; 1_b.dat : 4 by 1 vector; 1_Hb.dat : 4 by 5 argument matrix consist of H and b in the form $\{H|b\}$.
2_A.dat : 3 by 3 matrix; 2_b.dat : 3 by 1 vector; 2_Ab.dat : 3 by 4 argument matrix consist of A and b in the form $\{A|b\}$.
3_A.dat : 3 by 3 matrix.