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 real-world computational problems.

This project consists of three parts. The first part is worth 60 points, the second part is worth 25 points, and the third part is worth 15 points. Extra points will be considered for presentation and quality of discussion for up to 20 points.

You may work in teams of **up to three** persons. Code for your project should be submitted in either Java, Python, or MATLAB (ot 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 **March 31**. The file should contain:

- A `team_members.txt` file listing the name of each person in the project team.
- All source files for each part of the project you submit.
- A `.doc`, `.docx`, or `.pdf` file for each part of the project you submit, containing the written component of the project.
- A `readme` file for each part of the project you submit, explaining how to execute that part of the project.
- 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 and subtract vectors and matrices
- Multiply a scalar with a vector or matrix
- Take the dot product of two vectors
- Transpose a matrix or vector

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

- Find the multiplication of two matrices
- 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.**

1 The Hilbert Matrix

The Hilbert matrix is a square matrix with entries being the unit fractions

$$H_{ij} = \frac{1}{i+j-1}$$

(`http://en.wikipedia.org/wiki/Hilbert_matrix`.)

For instance, the 5×5 Hilbert matrix is

$$H = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}.$$

The objective of this part of the project is to solve the linear system $H\overline{\mathbf{x}} = \overline{\mathbf{b}}$, 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.

To calculate the error in this part, use the maximum norm:

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

Your job is to

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

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

(c) Implement the procedures to obtain the solution to a system $A\overline{\mathbf{x}} = \overline{\mathbf{b}}$, for an $n \times n$ matrix A and an $n \times 1$ vector $\overline{\mathbf{b}}$, using the LU and QR-factorizations. Name your programs `solve_lu_b`, `solve_qr_b`. IMPORTANT: the use of an inverse matrix function should be avoided, your program should use backward or forward substitution; use of an inverse matrix defeats the purpose of these methods (Why?).

(d) For each $n = 2, 3, \ldots, 20$, solve the system $H\overline{\mathbf{x}} = \overline{\mathbf{b}}$, where H is the Hilbert matrix H of $n \times n$ and $\overline{\mathbf{b}} = 0.1^{n/3}(1, 1, \ldots, 1)^t$. using the LU and QR-factorizations and solution methods described in (a), (b) and (c). For each n, your program should output the solution $\overline{\mathbf{x}}_{sol}$, and the errors $||LU - H||_\infty$ or $||QR - H||_\infty$, and $||H\overline{\mathbf{x}}_{sol} - \overline{\mathbf{b}}||$. The output should be easily readable on the screen or a text file.

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

(f) 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 matrix?

(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.)

Added note: A test case for your code. For $n = 4$

$$H = \begin{pmatrix} 1. & 0.5 & 0.333333 & 0.25 \\ 0.5 & 0.333333 & 0.25 & 0.2 \\ 0.333333 & 0.25 & 0.2 & 0.166667 \\ 0.25 & 0.2 & 0.166667 & 0.142857 \end{pmatrix}, \qquad \overline{\mathbf{b}} = \begin{pmatrix} 0.0464159 \\ 0.0464159 \\ 0.0464159 \\ 0.0464159 \end{pmatrix}.$$

LU:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0.333333 & 1. & 1 & 0 \\ 0.25 & 0.9 & 1.5 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1. & 0.5 & 0.333333 & 0.25 \\ 0. & 0.0833333 & 0.0833333 & 0.075 \\ 0. & 0. & 0.00555556 & 0.00833333 \\ 0. & 0. & 0. & 0.000357143 \end{pmatrix},$$

$\overline{\mathbf{x}}_{sol} = (-0.185664, 2.78495, -8.35486, 6.49822)^t$.

QR with Householder reflections:

$$Q = \begin{pmatrix} 0.838116 & -0.522648 & 0.153973 & 0.0263067 \\ 0.419058 & 0.441713 & -0.727754 & -0.31568 \\ 0.279372 & 0.528821 & 0.139506 & 0.7892 \\ 0.209529 & 0.502072 & 0.653609 & -0.526134 \end{pmatrix},$$

$$R = \begin{pmatrix} 1.19315 & 0.670493 & 0.474933 & 0.369835 \\ 0 & 0.118533 & 0.125655 & 0.117542 \\ 0 & 0 & 0.00622177 & 0.00956609 \\ 0 & 0 & 0 & -0.000187905 \end{pmatrix},$$

$\overline{\mathbf{x}}_{sol} = (-0.185664, 2.78495, -8.35486, 6.49822)^t$.

QR with Givens rotations:

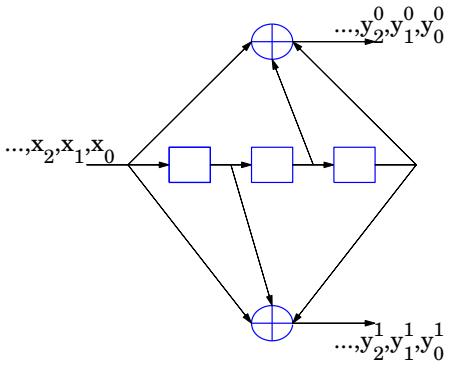
$$Q = \begin{pmatrix} 0.838116 & -0.522648 & 0.153973 & -0.0263067 \\ 0.419058 & 0.441713 & -0.727754 & 0.31568 \\ 0.279372 & 0.528821 & 0.139506 & -0.7892 \\ 0.209529 & 0.502072 & 0.653609 & 0.526134 \end{pmatrix},$$

$$R = \begin{pmatrix} 1.19315 & 0.670493 & 0.474933 & 0.369835 \\ 0. & 0.118533 & 0.125655 & 0.117542 \\ 0. & 0. & 0.00622177 & 0.00956609 \\ 0. & 0. & 0. & 0.000187905 \end{pmatrix},$$

$\overline{\mathbf{x}}_{sol} = (-0.185664, 2.78495, -8.35486, 6.49822)^t$.

2 Convolutional codes

Consider the rate-1/2 linear convolutional code in the figure:



(Error control systems, S.B. Wicker, p. 265).

A binary data stream $\overline{\mathbf{x}} = (x_0, x_1, x_2, \dots)$ is fed into a shift-register circuit consisting of a series of memory elements. With each successive input, the values of memory elements are tapped off and added according to the pattern in the figure, creating a pair of output coded data streams $\overline{\mathbf{y}}^0 = (y_0^0, y_1^0, y_2^0, \dots)$ and $\overline{\mathbf{y}}^1 = (y_0^1, y_1^1, y_2^1, \dots)$. These output streams can be multiplexed to create a single coded data stream $\overline{\mathbf{y}} = (y_0^0, y_0^1, y_1^0, y_1^1, y_2^0, y_2^1, \dots)$. $\overline{\mathbf{y}}$ is the convolutional code word.

Each element in the output stream $\overline{\mathbf{y}}$ is the linear combination of the elements in the input stream $\overline{\mathbf{x}}$. For the rate-1/2 code in the figure, an arbitrary coordinate y_j^0 in the output stream y_j^0 can be represented as

$$y_j^0 = x_j + x_{j-2} + x_{j-3}.$$

For instance, when we encode the information sequence $\overline{\mathbf{x}} = (1, 0, 1, 1, 0)$, we obtain the following coded output sequences

$$\overline{\mathbf{y}}^0 = (1, 0, 0, 0, 1, 0, 1, 0)$$
$$\overline{\mathbf{y}}^1 = (1, 1, 1, 1, 1, 1, 1, 0).$$

(Note that the input $\overline{\mathbf{x}}$, is equivalent to $(1, 0, 1, 1, 0, 0, 0, 0)$.) The convolutional code word corresponding to $\overline{\mathbf{x}} = (1, 0, 1, 1, 0)$ is

$$y = (11, 01, 01, 01, 11, 01, 11, 00).$$

The objective of this part of the project is to encode any input stream $\overline{\mathbf{x}}$, and to be able to decode an output $\overline{\mathbf{y}}$, knowing that it was obtained from the rate-1/2 convolutional code in the question.

Your job is:

- Reformulate the *encoding* problem as a linear transformation between the input space of $\overline{\mathbf{x}}$ streams and the output space of $\overline{\mathbf{y}}$ streams. In other words, write $\overline{\mathbf{y}}^0 = A^0\overline{\mathbf{x}}$, and $\overline{\mathbf{y}}^1 = A^1\overline{\mathbf{x}}$, where you need to determine A^0 and A^1. Your program should generate random binary input streams $\overline{\mathbf{x}}$ of a given length n, and produce the output stream y.

- Reformulate the *decoding* problem as the solution of the linear system $A^i\overline{\mathbf{x}} = \overline{\mathbf{y}}^i$ $(i = 0, 1)$. To find the solution, implement the iterative methods to solve linear systems:

  1. Jacobi iteration,
  2. Gauss-Seidel iteration.

Your programs should be named `jacobi`, `gauss_seidel`. In each case, the input is a matrix A, $n \times n$; a vector $\overline{\mathbf{y}}$, $n \times 1$; an initial guess vector $\overline{\mathbf{x}}_0$; an error tolerance number tol.

Your programs should find the stream $\overline{\mathbf{x}}$ given $\overline{\mathbf{y}}$ of any length $2m$, with an error tolerance tol of 10^{-8}. The programs should report the number of iterations required $kmax$ to achive the error tolerance $||\overline{\mathbf{x}}_n - \overline{\mathbf{x}}_{n-1}|| < tol$, or report that the method doesn't converge after a fixed maximum number of iterations.

For a test case, encode the stream $\overline{\mathbf{x}} = (1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0)$ to get $\overline{\mathbf{y}} = (11, 01, 01, 01, 00, 00, 10, 01, 00, 00, 01, 11, 11, 01, 10, 11, 00)$, and then decode $\overline{\mathbf{y}}$ to obtain $\overline{\mathbf{x}}$.

For the written component of this part, compare the results of the two methods above, and discuss the number of iterations required to obtain the desired precision. Is the length of the initial stream n important? Does n have an effect on the number of iterations required to achieve the error tolerance?

Note: You should produce results that are binary streams. One way to implement binary addition: 1+1=0, 0+0=0, 1+0=1, 0+1=1, is the use of the mod 2 (modulus 2) operator.

3 Urban population dynamics

We are interested in forecasting the population of a city. The model we use is based on a Leslie matrix, it describes the growth of populations (and their projected age distribution). It is used in ecology to model changes in a population over a period of time.

In this application of the Leslie Matrix, the population of a city is divided in age groups of 10 years: age groups of 0-9,10-19,11-20, etc., so we'll use 9 groups. At each time step, the population is represented by a vector with an element for each age classes where each element indicates the number of individuals currently in that class.

For 4 groups, the Leslie Matrix model is of the form:

$$\begin{pmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{pmatrix}(k+1) = \begin{pmatrix} f_1 & f_2 & f_3 & f_4 \\ s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \end{pmatrix} \begin{pmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{pmatrix}(k),$$

(`http://en.wikipedia.org/wiki/Leslie_matrix`) where n_x is the number of individuals in age class x, s_x is the fraction of individual that survives from age class x to age class $x+1$, and f_x is the fecundity, the per capita average number of female offsprings reaching n_1 born from mother of age class x.

The survival fractions would then show the fraction of "newborns" (0-9) who survive to age 10, the fraction of 10 to 19 year olds who survive to 20 etc.

The Leslie Matrix Model can be written as

$$\overline{\mathbf{n}}(k+1) = A\overline{\mathbf{n}}(k), \quad k = 0, 1, 2, \ldots \quad \text{and } \overline{\mathbf{n}}(0) \text{ given.}$$

This is, the population in 2000 was $\overline{\mathbf{n}}(0)$. In 2010, the population will be $\overline{\mathbf{x}}(1) = A\overline{\mathbf{x}}(0)$, in 2020 it will be $\overline{\mathbf{x}}(2) = A\overline{\mathbf{x}}(1)$, etc. The solution is found iteratively as

$$\overline{\mathbf{x}}(k) = A^k\overline{\mathbf{x}}(0).$$

Suppose the city in question has a population divided in 9 age groups: 0-9,10-19,...,70-79,80+. Suppose that the population distribution of the city in question in 2000 is

$$\overline{\mathbf{x}}(0) = (2.1, 1.9, 1.8, 2.1, 2.0, 1.7, 1.2, 0.9, 0.5)(\times 10^5).$$

and that the Leslie matrix, A, for this model appears as

$$A = \begin{pmatrix} 0 & 1.2 & 1.1 & .9 & .1 & 0 & 0 & 0 & 0 \\ .7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .85 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .88 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .77 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .40 & 0 \end{pmatrix}$$

For the analysis of this part, you will need to implement the Power method to find eigenvalues and eigenvectors. Name your program `power_method`. The input for your program will be an $n \times n$ matrix A, an error tolerance tol, and an initial approximation vector $\overline{\mathbf{u}}_0$. The output should be the approximated eigenvalue λ and approximated eigenvector $\overline{\mathbf{v}}$ and the number of iterations needed to achieve the desired error tolerance. Or, the program should report that the method doesn't converge after a fixed number of iterations.

For this part, your job is:

1. Interpret the data in the matrix, and discuss the social factors that influence those numbers.

2. What will the population distribution be in 2010? 2020? 2030? 2040? 2050? Calculate also the total population in those years, and by what fraction the total population changed each year.

3. Use the power method to calculate the largest eigenvalue of the Leslie matrix A. The iteration of the power method should stop when you get 8 digits of accuracy. What does this tell you? Will the population go to zero, become stable, or be unstable in the long run? Discuss carefully and provide the mathematical arguments for your conclusion. You might want to investigate the convergence of $||A^k||$.

4. Suppose we are able to decrease the birth rate of the second age group by half in 2020. What are the predictions for 2030, 2040 and 2050? Calculate again the largest eigenvalue of A (to 8 digits of accuracy) with your program and discuss its meaning regarding the population in the long run.