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 (worth 60 points) is required, and you may do either the second or third part (each of which is worth 40 points). If you submit all three parts, you can earn up to 20 points of extra-credit.

You may work in teams of **up to three** persons. Code for your project should be submitted in either Java, Python, or MATLAB. (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 **November 24**. 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.

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 inverse of a matrix
- Find the QR-factorization of a matrix

- 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 Gauss-Newton Method

This part of the project is **required**. For this part, you will implement the Gauss-Newton method for finding the best curve approximation to a set of data.

Given a curve $f_{a,b,c}(x)$ to approximate, the Gauss-Newton algorithm works as follows:

1. The inputs are a list of n pairs of floating point numbers of the form

   $$[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)],$$

   a triple of floating point numbers (a_0, b_0, c_0), and a number of iterations N.

2. Initialize $\boldsymbol{\beta}$ to be a vector with 3 coordinates: $\begin{bmatrix} a_0 \\ b_0 \\ c_0 \end{bmatrix}$. (The components of $\boldsymbol{\beta}$ are referred to as $\beta_1, \beta_2, \beta_3$.)

3. Initialize $\mathbf{r}$, the vector of *residuals*, to be a vector with n coordinates so that the i^{th} coordinate is $r_i = y_i - f_{\beta_1,\beta_2,\beta_3}(x_i)$.

4. Initialize $\mathbf{J}$, the *Jacobian* of $\mathbf{r}$, to be a $3 \times n$ matrix so that the ij-entry is

   $$\mathbf{J}_{ij} = \frac{\partial r_i(\boldsymbol{\beta})}{\partial \beta_j}.$$

5. Do the following procedure N times:
   (a) Set $\boldsymbol{\beta} = \boldsymbol{\beta} - (\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \mathbf{r}$.
   (b) For $i = 1, ..., n$, set $r_i = y_i - f_{\beta_1,\beta_2,\beta_3}(x_i)$.
   (c) For $i = 1, ..., n$ and $j = 1, 2, 3$, set
   $$\mathbf{J}_{ij} = \frac{\partial r_i(\boldsymbol{\beta})}{\partial \beta_j}.$$
6. Output $\boldsymbol{\beta}$.

You will write four executable programs (one for each of the curves listed below) that, when run, do the following:

1. prompt the user for a text file containing a list of points,
2. prompt the user for initial guesses for the parameters a, b, and c,
3. prompt the user for a number of iterations to run the Gauss-Newton algorithm, and
4. output the parameters giving the best approximation for the appropriate curve matching the given points.

The curves to be approximated will be the following.

- Quadratic: $f_{a,b,c}(x) = ax^2 + bx + c$
- Exponential: $f_{a,b,c}(x) = ae^{bx} + c$
- Logarithmic: $f_{a,b,c}(x) = a \log(x + b) + c$
- Rational: $f_{a,b,c}(x) = \frac{ax}{x+b} + c$

Name the programs `gn_qua`, `gn_exp`, `gn_log`, and `gn_rat`.

As written, the algorithm is ill-suited for numerical computations. Your job is to

(a) Implement two versions of a procedure to compute a QR-factorization of an $m \times n$ matrix A, where $m \geq n$. The first version should use Householder reflections. The second version should use Givens rotations. Name the procedures `qr_fact_househ` and `qr_fact_givens`.

(b) Implement a modified version of the Gauss-Newton algorithm. Line 5(a) in the above algorithm should be split into two separate lines:

- Set Q and R equal to the output of the QR-factorization of $\mathbf{J}$.
- Set $\boldsymbol{\beta} = \boldsymbol{\beta} - R^{-1} Q^\top \mathbf{r}$.

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

(i) Why is it justified to modify the algorithm to set $\boldsymbol{\beta}$ to $\boldsymbol{\beta} - R^{-1}Q^{\top}\mathbf{r}$?

(ii) What is the benefit of modifying the algorithm in this way? (Your answer should consider the benefit in terms of conditioning error.)

You should do **one** of the next two parts of the project. If you do both, specify which one should be graded for extra credit.

2 Convergence of the Power Method

In this part you'll investigate the convergence of the power method for calculating eigenvalues and eigenvectors of randomly generated 2×2 matrices.

Your job is to

(a) 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

- an $n \times n$ matrix A with floating-point real numbers as entries (the algorithm should work for $n \geq 2$),
- a vector $\mathbf{v}$ of n floating-point real numbers that serves as the initial guess for an eigenvector of A,
- a tolerance parameter ε (a positive floating-point real number) that determines when the approximation is close enough, and
- a positive integer N giving the maximum number of times to iterate the power method before quitting.

The outputs should be the approximate eigenvalue and eigenvector obtained by the power method as well as the number of iterations needed to obtain that approximation. If the procedure iterates N 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).

(b) Write a program that generates at least 1000 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,

- Compute 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 absolute value within an accuracy of 5 decimal places ($\varepsilon = 0.00005$). Use a maximum of $N = 100$ iterations before quitting in failure.

- Use `power_method` on A^{-1} to find the smallest eigenvalue of A in absolute value within an accuracy of 5 decimal places. Use a maximum of $N = 100$ iterations before quitting in failure.
- 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}.

(c) Using your results from (b), 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 (b) 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.

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

3 Animation in Two Dimensions

In this project, using linear transformations, you will create a movie consisting of 121 frames (numbered from 0 to 120) which can be shown one every 1/24 second. The idea is to create a sequence of frames such that consecutive pairs of frames differ slightly from one another. You should at least provide the data corresponding to each frame and show pictures of sample frames, but if you are able to provide a movie, that would be nice.

The plot should consist of three line-drawn letters (for example, 'L', 'U', 'Z'—though feel free to choose any three letters you want). Each letter should be drawn as a polygon with line segments. For instance, 'L' can be drawn as the polygon whose vertices are

$$(0,0), (4,0), (4,2), (2,2), (2,6), (0,6).$$

Let the x-direction refer to the axis horizontal relative to the screen, the y-direction refer to the axis vertical relative to the screen, and the z-direction refer to the axis perpendicular to the screen.

During the movie:

- The first letter should rotate three times about the line in the z-direction through the center of the letter.
- The second letter should rotate twice about the line in the y-direction through the center of the letter.
- The third letter should rotate five times about the line in the x-direction through the center of the letter.

[Hint: A rotation of the letter orthogonal to the frame can be achieved by scaling the letter.]

For the written component of this part, describe the linear and nonlinear transformations you needed for this animation.