340W25 PA1

**Due date:** Check the schedule table or eclass

**Submission:** A single **zip file** must be submitted through eClass. Use filename in the format: **pa1-[firstname]-[lastname].zip**.

**Total:** 100 (50 from code submission + 50 from lab demo)

**Submission details:**

You are expected to submit a **Matlab function/script for each portion of the PA   
(programming assignment)**; make sure your functions and demo script are easy to both demo and understand.

Your code should run on any machine without needing to be modified by the examining TA. You can test this by unzipping your solution to a new folder and running it from the new location.

In addition to the code, you should also submit written documentation as a **pdf file** to address the non-programming questions that appear throughout the PA description (Q2D this time).

Finally, we intend to browse your code before the demo. So, for consistency, we ask that you name your files something along the lines of **pa1\_1A.m, pa1\_1B.m,...,** so we can easily identify the question you are answering. For files that contain a specific function, like Q2A, the file name should match the name of the function, for example, **elimMat.m**. The written documentation should be in a file called **writeup.pdf.**

**Demo details:**

During the demo, you will demonstrate your solution, explain your code and strategy, and answer questions from the examining TA, who will ensure that you did the work yourself and that you understand the material relevant to the assignment. The questions can be considered a small, informal oral quiz where the examining TA draws a few questions from a prepared list. They will test your knowledge of the assignment and related theories. The target meantime for a demo is about 5min (+/-2), so make sure your functions and demo script are prepared to be easy to both demo and understand.

# Q1: Intro to Matlab (10)

### Objectives

The objective of the first question is to acquire basic Matlab literacy, in particular, familiarity with vector and matrix operations.

### Deliverables

1. Vectorize [this script](https://drive.google.com/file/d/1BLN6x6BPha2u6oiykoZBnjU6dVpdgjfl/view?usp=drive_link) and compare the running time of both implementations. Write your comparison as matlab comments (5)
2. Fill in [this script](https://drive.google.com/file/d/1BAnWPD_8cDb5iv7WyMisXKJ4QPofrXgQ/view?usp=drive_link) details to perform the required [image](https://drive.google.com/file/d/1BFyXWWwmFIOuSIU8F0Bafa5Z8A1HWxRk/view?usp=drive_link) modifications as below. (5)  
   You **should not** use the imcrop function of Matlab. Display all images in one window like the below figure.



# Q2: LU factorization (30)

### Objectives

1. To practically implement the routines involved in LU factorization and thereby get hands-on knowledge of the numerics of factorization.
2. To write numerical routines in Matlab, particularly to access sub-parts of vectors and matrices.
3. To write clean, correct vectorized code.

### Deliverables

The goal is to compose a linear equation solver from the following parts:

1. Write a function [M\_k, L\_k] = elimMat(A, k) that, given matrix A and index k, computes the elimination matrix M\_k and L\_k as specified on page 67 of the textbook. You can assume the matrix A is non-singular. (5)
2. Write a function [L, U] = myLU(A) that reuses elimMat to give an LU factorization of A. (5)
3. Write afunction x = backSubst(U, y, k) that performs back-substitution to solve Ux = y either recursively or using one for loop. (10)  
   A function for forward-substitution solving Ly = b recursively is provided [here](https://drive.google.com/file/d/1BPRMbkg1CpOTszIWYmG8m8gMCHGjgfLZ/view?usp=drive_link).
4. In your written documentation, explain how to use your programs to solve Ax = b using methods described on page 68 of the textbook (5):
   1. Factorization: LU = A
   2. fwdSubst to solve for y in Ly = b
   3. backSubst to solve for x in Ux = y.
5. Test your program with, e.g., example 2.13 and one test of your own design. Write an m-file script to execute the test. (5)

### Coding Style to Follow

Show how to use matlab matrix operations to write the above in a clear, compact form with vectorization and a minimum number of for loops (e.g., copying the for-loop-based algorithms 2.2 and 2.3 from the book will not yield good marks). Write your code in a readable, commented, and suitably indented fashion.

### 

### Vectorization Requirements

The following table gives the restrictions on the number of loops (for & while) for the deliverable functions:

| Function | **Maximum** number of loops |
| --- | --- |
| elimMat(A, k) | 0 |
| myLU(A) | 1 |
| backSubst(U, y, k) | 1 |

### Hints

First, understand and experiment with the supplied algorithm for forward-substitution alone. Modify it to do back-substitution. Then write the elimMat and myLU routines. myLU can be written recursively much the same way as the fwdSubst, or with one "for" loop. You need no loop in computing the elimination matrix and, at most, one loop (or recursion) in computing the LU, and forward and back-substitutions. Achieving this depends mostly on skills in linear algebra to do operations on vectors and matrices as a whole instead of element by element and on understanding Matlab's matrix elements addressing as described in the first part of the tutorial. This way, each function (elim matrix, lu, and substitutions) can be written in just 5-10 lines.

# Q3: Approximation (35)

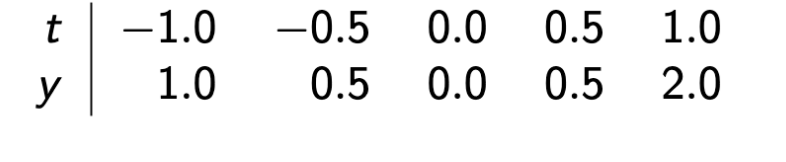
### 

### Objectives

In this question, our objective is to use matlab’s built-in function to perform QR factorization and SVD. Both of them will be applied in the context of solving least square problems related to fitting a function that approximates a given dataset. **For each question, print the output vector or matrix such as Q, R, and x.**

### Deliverable

Our given dataset contains five points as used in the slides:



Let’s approximate this dataset using a quadratic polynomial function. Form the corresponding least-squares problem **Ax** ≅ **b**.

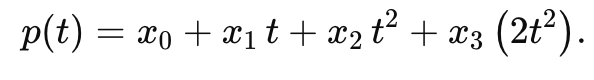
Solve it using QR factorization:

1. Use Matlab’s built-in qr function to perform **reduced** QR factorization of **A**.
2. Find the best-approximating coefficients **x** using **Q** and **R** matrices from the previous step and backSubst from the previous question.
3. Plot the data points and the quadratic function based on coefficients **x** in the same figure.

Solve it using SVD:

1. Use Matlab’s built-in svd function to perform **reduced** singular-value decomposition of **A**.
2. Find the best-approximating coefficients **x** using matrices from the previous step and backSubst from the previous question. **Is it efficient to use backSubst here?**
3. Plot the data points and the quadratic function based on coefficients **x** in the same figure.

Now, let’s say we want to approximate this dataset using the following quadratic polynomial function with four unknowns:



Form the corresponding least-squares problem **Ax** ≅ **b**. Note that A will be 5 by 4.

Solve it using QR factorization:

1. Use Matlab’s built-in qr function to perform **reduced** QR factorization of **A**.
2. Can you find the best-approximating coefficients **x** using **Q** and **R** matrices from the previous step and backSubst from the previous question? What difficulty do you face?

Solve it using SVD:

1. Use Matlab’s built-in svd function to perform **reduced** singular-value decomposition of **A**.
2. Find the best-approximating **minimum-norm** coefficients **x** using matrices from the previous step. Take appropriate steps to modify backSubst so that it can calculate the pseudo-inverse of the diagonal matrix. Make a new copy of backSubst to do that.
3. Plot the data points and the quadratic function based on coefficients **x** in the same figure.

Now, let’s interpolate the same dataset. Form the linear system **Ax** = **b**.

Solve it using QR factorization:

1. Use Matlab’s built-in qr function to perform reduced QR factorization of **A**.
2. Find the best-approximating coefficients **x** using matrices from the previous step and backSubst from the previous question.
3. Plot the data points and the interpolant based on coefficients **x** in the same figure.

Solve it using SVD:

1. Use Matlab’s built-in svd function to perform singular-value decomposition of **A**.
2. Find the best-approximating coefficients **x** using matrices obtained from the previous step and backSubst from the previous question.

Note: Do not use inv or any other means to invert a matrix directly. Do not use \ operator or built-in functions for calculating pseudo-inverse such as pinv.

### 

### Q4: Interpolation (25)

### Objectives

For this question, you will be implementing polynomial interpolation using the Monomial, Lagrange, and Newton basis functions.

### Deliverables

Starter code is provided for you to fill in. Complete the required tasks in [**PA1\_4.m**](https://drive.google.com/file/d/1L47ppcXQ0tuLJ2wkoJy54O-yBVWnzbFV/view?usp=drive_link), which are labelled with **TODO (X marks)**.

### Generalization Requirements

You may not hard code any constants specific to the example problem. Try changing the constants given in **coeffs** and **x\_data** to make sure your solution still works. No marks will be given for a task if your solution does not work for other polynomials.

For full marks, your code must work for polynomials of any degree. (Hint: **degree** is provided as a variable and calculated automatically). You can assume you will have as many unique data points as needed for the system to have a unique solution.

For 50% of the indicated marks, your code must work for any degree 3 polynomial with 4 data points provided. You may find it easier to start with this solution and generalize to any degree polynomial after.

### Symbolic Math

This assignment makes use of MatLab’s Symbolic Math library. If you do not have it installed, you will get an error message with a clickable link that will allow you to install it. Make sure to ask us if you have any issues with this.