University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 338 - Numerical Methods, Spring 2018

Assignment 2

Due date: Tuesday March 6, 11:59pm.
No late assignments will be accepted.

As in all work in this course, 20% of your grade is for quality of presentation, including the use of good English, properly commented and easy-to-understand programs, and clear proofs. In general, short, simple answers are worth more than long, complicated ones. Unless stated otherwise, all answers should be justified. The TA has a limited amount of time to devote to each assignment, so what you hand in should be legible (either typed or *neatly* hand-written), well-organized and easy to evaluate. (An employer would demand no less.) All computer problems are to be done in Python with the NumPy and SciPy libraries.

Hand in five files: the source code of all your Python programs, a pdf file of figures generated by the progams, a text file of all printed output, a pdf file of answers to all the non-programming questions (scanned hand-writing is fine, but *not* photographs), and a scanned, signed copy of the cover sheet at the end of the assignment. Be sure to indicate clearly which question(s) each program and piece of output refers to. All the Python code (functions and script) for a given question should appear in one location in your source file, along with a comment giving the question number. All material in all files should appear in order; *i.e.*, material for Question 1 before Question 2 before Question 3, etc. It should be easy for the TA to identify the material for each question. In particular, all figures should be titled, and all printed output should be identified with the Question number. The five files should be submitted electronically as described on the course web page. In addition, if we run your source file, it should not produce any errors, it should produce all the output that you hand in (figures and print outs), and it should be clear which question each piece of output refers to. *Output that is not labled with the Question number will not be graded.*

Style: Use the solutions to Assignment 1 as a guide/model for how to present your solutions to Assignment 2.

I don't know policy: If you do not know the answer to a question (or part), and you write "I don't know", you will receive 20% of the marks of that question (or part). If you just leave a question blank with no such statement, you get 0 marks for that question.

More questions will be added shortly.

Tips on Scientific Programming in Python: If you haven't already done so, please read the NumPy tutorial on the course web page. Be sure to read about slicing and indexing Numpy arrays. For example, if `A` is a matrix, then `A[:,5]` returns the 5th column, and `A[7,[3,6,8]]` returns the 3rd, 6th and 8th elements in row 7. Similarly, if `v` is a vector, then the statement `A[6,:]=v` copies `v` into the 6th row of `A`. Note that if `A` and `B` are two-dimensional numpy arrays, then `A*B` performs *element-wise* multiplication, *not* matrix multiplication. To perform matrix multiplication, you should use `numpy.matmul(A,B)`. Whenever possible, *do not use loops*, which are very slow in Python. In particular, avoid iterating over the elements of a large vector or matrix. Instead, use numpy's vector and matrix operations, which are much faster and can be executed in parallel. For example, if `A` is a matrix and `v` is a column vector, the `A+v` will add `v` to every column of `A`. Likewise for rows and row vectors. Also, the functions `sum` and `mean` in `numpy` are useful for summing or averaging over all or part of an array. Many NumPy functions that are defined for single numbers can be passed lists, vectors and matrices instead. For example, $f([x_1, x_2, ..., x_n])$ returns the list $[f(x_1), f(x_2), ..., f(x_n)]$. The same is true for many user-defined functions. The term `numpy.inf` represents infinity. It results from dividing by 0 in numpy. It can also result from overflow (*i.e.*, from numbers that are too large to represent in the computer, like 10^{1000}). The term `numpy.nan` stands for "not a number", and it results from doing 0/0, inf/inf or inf-inf in numpy. For generating and labelling plots, the following SciPy functions in `matplotlib.pyplot` are needed: `plot`, `xlabel`, `ylabel`, `title`, `suptitle` and `figure`. You can use Google to conveniently look up SciPy functions. e.g., you can google "numpy matmul" and "pyplot suptitle".

1. (16 points total) *LU factorization.*

   (a) (10 points) Write down the LU factorization of the following matrix:

   $$A = \begin{pmatrix} 1 & 3 & -1 \\ 3 & 7 & -2 \\ 2 & 12 & -2 \end{pmatrix}$$

   Do not use pivoting. You do not need to show the entire derivation, but do show each of the elimination matrices M_k and L_k used in the derivation.

   (b) (6 points) Use the LU factorizarion to solve the equations $Ax_1 = b_1$ and $Ax_2 = b_2$, where $b_1 = (-2, -2, -10)^T$ and $b_2 = (4, 6, 20)^T$. You do not need to show the entire derivations, but do show the results of solving $Ly_i = b_i$ and $Ux_i = y_i$.

2. (27 points total) Suppose A is a symmetric positive-definite matrix, and let LL^T be its Cholesky factorization.

   (a) (5 points) Derive formulas for computing the first column of L.

(b) (5 points) Prove that for $i > 1$,

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2} \tag{1}$$

(c) (5 points) Prove that for $i > 1$ and all j,

$$L_{ji} = \frac{A_{ji} - \sum_{k=1}^{i-1} L_{jk} L_{ik}}{L_{ii}} \tag{2}$$

(d) (5 points) Use the above results to write an efficient algorithm (in pseudo code) to compute the lower triangle of L. Argue that your algorithm is well-defined. That is, at each point in the computation, it only uses values that have already been computed.

(e) (7 points) If A is $n \times n$, prove that your algorithm performs $n^3/6 + O(n^2)$ floating-point multiplications (where a division counts as one multiplication). You may assume that a square root does a fixed number of multiplications (independent of n). Also $\sum_{i=1}^{n} i^k = n^{k+1}/(k+1) + O(n^k)$.

3. (10 points total) Which of the following matrices, A, are orthogonal? Justify your answers. (2 points each)

(a) $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$

(b) $\frac{1}{5}\begin{pmatrix} 4 & 0 \\ 0 & -5 \\ 3 & 0 \end{pmatrix}$

(c) $\frac{1}{\sqrt{2}}\begin{pmatrix} -1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}$

(d) $\frac{1}{\sqrt{13}}\begin{pmatrix} 2 & 3 \\ 3 & -2 \end{pmatrix}$

(e) $\frac{1}{\sqrt{12}}\begin{pmatrix} -2 & 1 \\ 2 & 3 \\ -2 & 2 \end{pmatrix}$

4. (8 points total) Consider the following linear least-squares problem:

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 3 & 4 \\ 0 & 0 & 5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

(a) (2 points) If we denote this problem $Ax \approx b$, what is the minimum value of $\|Ax - b\|$? Show your work.

(b) (6 points) What is the solution vector, x, for this problem? Show your work.

5. (20 points total) Suppose A is a non-singular matrix and u and v are column vectors. Show that the inverse of $A - uv^T$ can be easily computed from the inverse of A. In particular, do the following:

   (a) (7 points) Prove that

   $$(A - uv^T)^{-1} = A^{-1} + \frac{A^{-1}uv^TA^{-1}}{1 - v^TA^{-1}u}$$

   (b) (3 points) Write an efficient algorithm (in pseudo code) to evaluate the right-hand side of the above formula. Assume you are given A^{-1}.

   (c) (5 points) If A is $n \times n$, how many floating-point multiplications does your algorithm make? Justify your answer.

   (d) (5 points) A rank-1 matrix has the form xy^T, where x and y are column vectors. Suppose A and B are non-singular matrices. Show that $A - B$ is rank-1 if and only if $A^{-1} - B^{-1}$ is rank 1.

6. (15 points total) *Curve Fitting.*

   In this problem you will write a Python program to fit a curve to data using linear least-squares. First, download the file `data2.pickle.zip` from the course web site. Uncompress it (if your browser did not automatically do it). Start the Python interpreter and import the `pickle` module. You can then read the file `data2.pickle` with the following command:

   ```
   with open('data2.pickle','rb') as f:
       data = pickle.load(f)
   ```

   The variable `data` will now contain a 2×50 Numpy array. Each column of the array represents a 2-dimensional point, (t_i, y_i). The data is illustrated in the scatter plot in Figure 1 below. Your job is to fit a trigonometric polynomial of degree 4 to this data. Specifically, you are to estimate the coefficients $x = (x_0, x_1, ..., x_4)$ in the following function:

   $$y(t) = x_0 + x_1 \sin t + x_2 \sin 2t + x_3 \sin 3t + x_4 \sin 4t$$

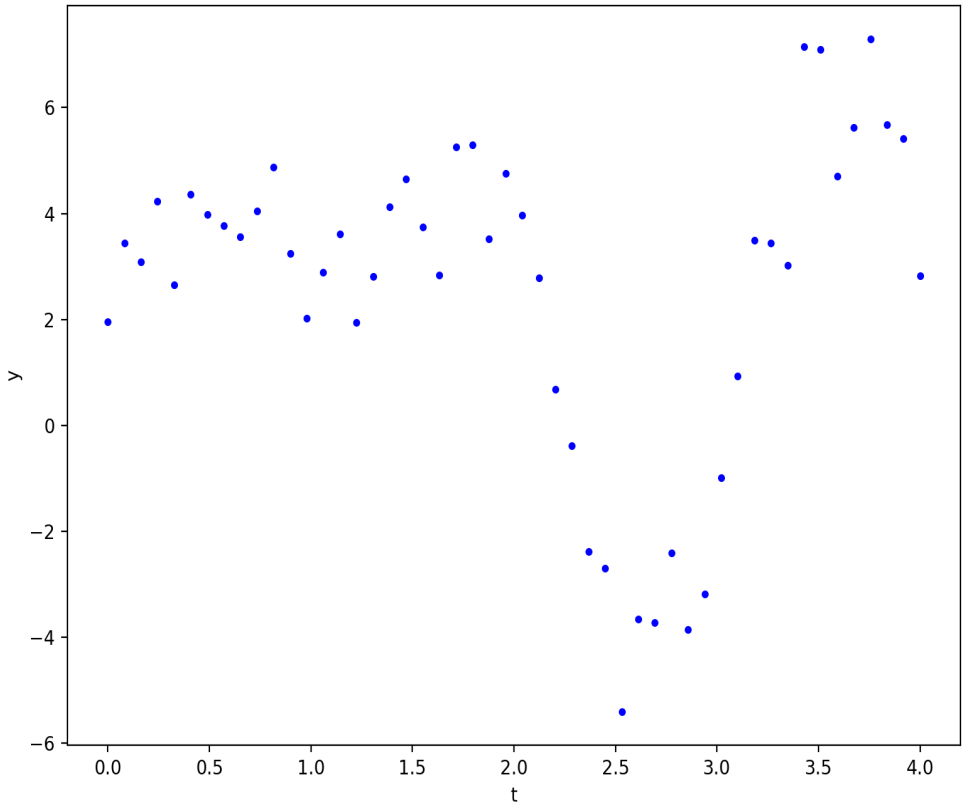
   You should formulate this as a least-squares problem, $min_x \|Ax - b\|^2$, and solve for x. Specifically, write a Python program that reads the data file and does the following:

   (a) (5 points) Constructs the matrix A and the vector b from the data.

   (b) (2 points) Uses the function `lstsq` in `numpy.linalg` to solve the least-squares problem. (Note that `lstsq` returns several values.)

   (c) (1 point) Prints out the estimated values of the coefficients x_i. (Hand in the printed values.)

Figure 1: Data for curve fitting



(d) (4 points) Plots the fitted polynomial (in green) using 1000 equally spaced values of t between 0 and 4. Plots the 50 data points (in blue) on top of the polynomial. Label the axes, and title the figure, "Question 6. Data and fitted polynomial". (Hand in the plot.)

(e) (2 points) Computes and prints the error, $\|Ax - b\|$.

If you have done everything correctly, the error should be less than 6.5.

(1 point) What are the dimensions of matrix A? How many entries does it have?

7. (19 points total) *Augmented Systems.*

Repeat Question 6, but instead of using the function `lstsq` to solve the least squares problem, use the Augmented System method. Specifically, write a Python program that does the following:

(a) (5 points) Constructs the augmented system.

(b) (4 points) Solves the system using LU factorization, as in Assignment 1.

(c) (1 point) Prints the estimated values of the coefficients x_i. (Hand in the printed values.)

(d) (4 points) Plots the fitted polynomial and the data points, as in Question 6(d). Title the figure, "Question 7. Data and fitted polynomial". (Hand in the plot.)

(e) (4 points) Prints the error, $\|Ax - b\|$, without explicitly computing $Ax - b$ (see page 118 in the text).

You should find that the answers above are very similar (if not identical) to those in Question 6.

(1 point) What are the dimensions of the matrix in the augmented system? How many entries does it have?

8. (30 points total) *Lossy Compression.*

In this question, you will use linear least squares to compress MNIST digits. Recall that each MNIST digit is a 784-dimensional vector. In general, suppose x is a N-dimensional vector, and we wish to compress it to an M-dimensional vector, z, where $M < N$. We cannot do this for all vectors without losing some information. (Hence the term *lossy compression.*) The goal is to minimize the information loss, so that when we reconstruct x from z, the result is as similar to x as possible. One way to do the reconstruction is with a matrix. That is, given a $N \times M$ matrix, A, we let $\hat{x} = Az$ be the reconstruction of x. To compress x, we find the z that minimizes the reconstruction error, $\|x - Az\|$. If we use the L_2 norm to measure error, then we can find z using linear least squares.

In this question, you will use a random matrix for A. (In the next assignment, you will do something much more sophisticated and effective.) In the questions below, as in all questions, minimize the use of iteration in Python. In particular, do not iterate over the elements of a vector or matrix.

(a) (5 points) Choose an MNIST image, x. Use a random matrix, A, to compress x to a vector of dimension 700 as described above. Use the function `lstsq` to solve the least squares problem. Display x as an image. Display its reconstruction, $\hat{x}$, as an image. Display both images side-by-side in a single figure. Give each image and the entire figure appropriate titles. The reconstruction should look slightly noisy, due to loss of information during compression. You will find the functions `subplot`, `title` and `suptitle` in `matplotlib.pyplot` useful. Also, compute and print out the reconstruction error, $\|\hat{x} - x\|$.

(b) (5 points) Repeat part (a) using QR factorization to solve the least squares problem, instead of `lstsq`. In particular, use the functions `qr` and `solve_triangular` in `scipy.linalg`, and use `qr` in "full" mode (the default). (See slide 27 in Chapter 3.) Compute and print out the reconstruction error, $\|\hat{x} - x\|$, but *without* explicitly computing the difference $\hat{x} - x$ or using the reconstruction $\hat{x}$. (See slide 26 in Chapter 3.) Use the same image, x, and the same matrix, A, as in part (a), so you should get exactly the same results.

(c) (5 points) Use Python to *randomly* pick 16 MNIST images. Using QR factorization, compress each of them to a 600-dimensional vector. Using `subplot`, display the 16 reconstructed images in a single figure arranged in a 4x4 grid. Title the figure, "Question 8(c): some compressed and reconstructed images." You may find the function `shuffle` in `numpy.random` useful.

(d) (5 points) In this question, you will compress a MNIST image by varying amounts, and display the reconstructed images. Choose an MNIST image (different from part (a)). Using QR factorization, compress the image to a M-dimensional vector, for M = 50, 150, 250, ..., 750. This will give you eight different compressed versions of the image, $z_1, z_2, ..., z_8$, from which you should produce eight reconstructed versions of the image, $\hat{x}_1, \hat{x}_2, ..., \hat{x}_8$.

Display the original image and the eight reconstructions in a single figure arranged in a 3×3 grid. Place the original image in the top-left corner, and arrange the reconstructions in order of increasing compression from left to right and top to bottom (so M=50 is in the bottom-right corner.) Give each image and the entire figure appropriate titles. If you have done everything correctly, the reconstructions should look noisier when there is more compression (i.e., when M is low).

(e) (5 points) Using QR factorization, compress all the MNIST training images to 600-dimensional vectors. Compute and print out the root mean squared (RMS) reconstruction error:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\|\hat{x}_i - x_i\|^2}$$

where x_i is the i^{th} image, $\hat{x}_i$ is its reconstruction, and N is the total number of images (55,000 in this case). Compute the RMS error *without* explicitly computing any of the reconstructions, $\hat{x}_i$.

(f) (5 points) Randomly pick 1000 MNIST training images. Using QR factorization, compute the RMS reconstruction error for the images when they are compressed to M-dimensional vectors. Do this for M = 4, 14, 24, 34, ..., 784. Plot the RMS error vs. M. Label the figure and axes of the plot appropriately. As in part (e), compute the RMS errors *without* computing any reconstructions, $\hat{x}_i$. (Note that your compression algorithm should work when M=784, which results in no compression and should give a reconstruction error of 0.)

University of Toronto Mississauga
CSC 338 - Numerical Methods

Cover sheet for Assignment 2

Complete this page and hand it in with your assignment.

Name: _____
(Underline your last name)

Student number: _____

I declare that the solutions to Assignment 2 that I have handed in are solely my own work, and they are in accordance with the University of Toronto Code of on Academic Matters.

Signature: _____