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

Assignment 3

Due date: Tuesday April 3, 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 Assignments 1 and 2 as a guide/model for how to present your solutions to Assignment 3.

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.

No more questions will be added.

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. (13 points total) We wish to annihilate all but the first entry of the vector $a = (3, 1, 2, 4)^T$ by using a Householder transformation, $H = I - 2vv^T/v^Tv$. Write Python programs to do each of the following tasks:

   (a) (2 points) Without computing H or v, compute and print the value of Ha.

   (b) (2 points) Compute and print the value of v.

   (c) (2 points) Compute and print H.

   (d) (4 points) Verify your answer to part (a) by computing Ha in two different ways. (*i*) Using the value of H computed in part (c) and carrying out the matrix-vector multiplication. (*ii*) Using v instead of H and not doing any matrix-vector multiplications. Print the value of Ha in both cases. If you do not get exactly the same answer as in part (a), explain why not. You may find the function `numpy.eye` useful. Parts (*i*) and (*ii*) should take only one line of Python code each.

   (e) (3 points) Compute HA where A is a random 4×6 matrix. Do this in two ways. (*i*) Using the value of H computed in part (c) and carrying out the matrix-matrix

multiplication. Call the result B. (*ii*) Not doing any matrix-matrix multiplications, though you may do matrix-vector multiplications. Call the result C. Print B and C. They should be very similar (if not identical). Compute and print $\sum_{ij}(B_{ij} - C_{ij})^2$. It should be very close to 0. Do *not* use iteration in any part of this question.

2. (6 points total) Write out the statement for updating the iterate x_k using Newton's method for solving each of the equations below. (2 points each):

   (a) $x^3 - 5x^2 + 8x - 4 = 0$

   (b) $x\cos(20x) = x^3$

   (c) $e^{-2x} + e^x = x + 4$

3. (9 points) For each equation in Question (2), use Python to plot the left-hand side of the equation as a function of x (in blue). On the same axes, plot the the right-hand side as a function of x (in orange). Choose the range of x so that all solutions to the equation are clearly visible. How many solutions does each equation have? Label the three figures appropriately and hand them in. (3 points each)

4. (10 points) For the equation in Question 2(c), implement Newton's method in Python and find all the values of x that satisfy the equation. At each iteration of Newton's method, print out the values of x, $f(x)$, and Δx. Here, Δx is the change in the value of x between iterations, and Newton's method is trying to solve the equation $f(x) = 0$. Iteration should terminate when $|f(x)|$ and $|\Delta x|$ are both less than 10^{-10}.

5. (10 points total) We wish to solve the following equation:

$$2x^2 - x^3 = 1$$

   (a) (2 points) What is the (absolute) condition number for this problem (as a function of x)?

   (b) (8 points total) Show that solving this problem is equivalent to finding the fixed points of the functions, g_i, below. In each case, determine whether fixed-point iteration converges at $x = 1$ and if convergence is linear or quadratic. If it is linear, determine the convergence constant, C.

      i. (4 points) $g_1(x) = 1/x + (x^3 - 1)/2x$

      ii. (4 points) $g_2(x) = \sqrt{(1 + x^3)/2}$

6. (19 points) We wish to use Newton's method to solve the following system of equations:

$$\begin{aligned} y_0 &= x_0^2 + x_0x_1 + x_1^2 - 3x_0 - x_1 - 3 = 0 \\ y_1 &= 2x_0^2 - x_0x_1 - x_1^2 + x_0 + 2x_1 - 1 = 0 \end{aligned}$$

   (a) (4 points) Derive the Jacobian matrix for this problem.

(b) (10 points) Write a Python program that uses Newton's method to solve the equations. Use $(0, 0)$ as the initial value of the vector $x = (x_0, x_1)$, perform ten iterations, and use matrix inversion when computing s, the change in x. At each iteration, print out the norm of s and the norm of the vector $y = (y_0, y_1)$. If you have done everything correctly, both norms should be less than 10^{-16} when iteration terminates. Print out the initial and final values of x.

(c) (5 points) Repeat part (b) using LU factorization instead of matrix inversion when computing s, as in Assignment 1. If you have done everything correctly, you should get the same value for x as in part (b).

7. (15 points total) For each matrix below, determine whether it is positive definite, negative definite or indefinite. Hints: if A is a symmetric matrix, then (i) A is positive definite iff it has a Cholesky factorization; (ii) A is negative definite iff $-A$ is positive definite; and (iii) A is indefinite iff it is neither positive or negative definite. Show your work. (5 points each)

(a) $\begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$

(b) $\begin{pmatrix} -3 & 2 \\ 2 & -3 \end{pmatrix}$

(c) $\begin{pmatrix} -3 & 4 \\ 4 & -3 \end{pmatrix}$

8. (27 points total) Consider the function $y = 2x_0^4 + 3x_1^4 - x_0^2x_1^2 - x_0^2 - 4x_1^2 + 7$.

(a) (2 points) Derive the gradient.

(b) (4 points) Derive the Hessian. (It should be symmetric.)

(c) (10 points) Write a Python program that uses Newton's method to find a local minimum of the function. Use $(1, 1)$ as the initial value of the vector $x = (x_0, x_1)$, perform ten iterations, and use matrix inversion when computing s, the change in x. At each iteration, print out the norm of s and the value of y. If you have done everything correctly, then when iteration terminates, the norm of s should be less than 10^{-16} and the gradient should be very close to $(0, 0)$. Print out the initial and final values of x.

(d) (5 points) The Hessian is guaranteed to be symmetric. Also, close to a minimum, it is positive definite. It is therefore amenable to Cholesky factorization. Repeat part (b) using Cholesky factorization instead of matrix inversion when computing s. If you have done everything correctly, you should get the same value for x as in part (b). Use the Cholesky factorization function in `numpy.linalg`.

(e) (3 points) Without running your programs, find three other local minima of y. Hint: notice that the value of y does not depend on the signs of x_0 and x_1.

(f) (3 points) Test your program in part (d) by finding these three other minima. Hand in the print outs for each of the three runs.

9. (33 points total) *Lossy Compression.*

This question builds on Question 8 in Assignment 2. Again, the goal is to use linear least squares to compress vectors, and to use this to compress MNIST images. Now, however, you will be much cleverer in choosing the compression matrix, A. Instead of choosing the matrix randomly, as in Assignment 2, you will use linear least squares repeatedly in order to find the matrix that compresses the MNIST images as much as possible. That is, you will write a program that learns the best possible compression matrix.

Recall that to compress an M-dimensional vector, x, to a K-dimensional vector, z, we find the z that minimizes $\|x - Az\|$, where A is an $M \times K$ matrix. The reconstruction of x from z is $\hat{x} = Az$. In assignment 2, you computed the root mean squared (RMS) reconstruction error for the entire MNIST training set:

$$RMS \;=\; \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|x_i - Az_i\|^2}$$

where x_i is the i^{th} MNIST image, z_i is its compressed version, and N is the total number of images (55,000 in this case). In this question, you will find the matrix A that minimizes the RMS error. Equivalently, you can minimize the total squared error:

$$TSE \;=\; \sum_{i=1}^{N} \|x_i - Az_i\|^2$$

Of course, this is a chicken-and-egg problem: you need the compressed vectors, z_i, to compute A; but you need A in order to compress the vectors. To solve this problem, you will initialize A randomly, and then iteratively improve it. Here is an outline of your algorithm:

```
A = a random matrix
do repeatedly
    (1) use A to compress the MNIST images
    (2) use the compressed images to find a better A
```

You will do this in such a way that the total squared error decreases at each step. Since this error is bounded below by 0, the procedure will converge.

In Assignment 2, you learned how do do step (1). The question now is how to do step (2). Interestingly, this is also a least-squares problem. To see this, it is helpful to use matrix notation. Let X be a matrix in which each column is a MNIST training image, and let Z be the corresponding matrix of compressed images (so column i of Z is the compressed version of column i of X). Finally, let A be the current compression matrix. It is not hard to show the following:

$$TSE \;=\; \|X - AZ\|_F^2 \qquad (1)$$

Here $\|\cdot\|_F$ is the Frobenius norm of a matrix. That is, $\|M\|_F^2 = \sum_{ij} M_{ij}^2$. We can now treat compression as a matrix-factorization problem. That is, we want to factorize matrix X into A and Z so that $X \approx AZ$, and more specifically, so that $\|X - AZ\|_F$ is as small as possible. This can be done iteratively using the algorithm above, using linear least squares to implement steps (1) and (2).

Step (1): Given matrices X and A, use linear least squares to find the matrix Z that minimizes TSE. You can do this with a single call to the Python function `lstsq` in `numpy.linalg`. This is what you did in Assignment 2.

Step (2). It is easy to show that $\|M\|_F = \|M^T\|_F$. Thus,

$$TSE \;=\; \|(X - AZ)^T\|_F^2 \;=\; \|X^T - Z^T A^T\|_F^2$$

You can therefore do the reverse of Step (1): given matrices X^T and Z^T, use linear least squares to find the matrix A^T that minimizes TSE. Again, you can do this with a single call to `lstsq`.

What to do:

(a) (5 points) Prove Equation (1).

(b) (2 points) Prove that $\|M\|_F = \|M^T\|_F$ for any matrix, M.

(c) (7 points) Suppose X is a $M \times N$ matrix. Write a Python python function `matfact(X,K)` that factorizes X into a $M \times K$ matrix, A, and a $K \times N$ matrix, Z, as described above. You should do this as follows, where $MSE = TSE/MN$ is the mean squared error per pixel per image:

- Initialize A to be a random $M \times K$ matrix.
- Use linear least squares to compute Z from X and A, by minimizing TSE, as described above. Print MSE.
- Use linear least squares to compute A from X and Z, by minimizing TSE, as described above. Print MSE.
- Repeat the previous two steps ten times.
- Return A and Z

Your program should have exactly one loop. If you have done everything correctly, the value of MSE should always decrease.

(d) (2 points) Use your function `matfact(X,K)` to compute the best matrix, A, for compressing MNIST images to vectors of dimension $K = 100$. The matrix `X` should consist of all the MNIST training images. If you have done everything correctly, the final mean squared error (MSE) should be below 0.0058.

(e) (3 points) Using A and Z, reconstruct all the MNIST training images from their compressed versions. You can do this in a single line of Python code. Display 16 of the reconstructed images. Display them in a single figure in a 4×4 grid.

(f) (4 points) The matrix A returned by `matfact` is much better at compressing MNIST images than the method you used in Assignment 2, which used a random compression matrix. To see this, generate a random matrix for compressing MNIST images to 100-dimensional vectors. Call this matrix `Arand`. Use it to compress and reconstruct all the MNIST training images (1 or 2 lines of Python code). Display 16 of these images in a single figure in a 4×4 grid. If you have done everything correctly, the reconstructions in part (e) should look much better than these.

(g) (3 points) In the MNIST images, all pixels values are between 0 and 1. However, the reconstructions may have values outside this range. To see this, plot a histogram of all the pixel values of your reconstructed images from part (e). The histogram should contain $784 \times 55,000 = 43,120,000$ values. You can do this in a single line of Python code using the function `hist` in `matplotlib.pyplot`. Generate a histogram with 100 bins. If you have done everything correctly, the histogram should have a high peak near 0 (from the white background in all the images). It should also have a significant number of negative values and values greater than 1.

(h) (2 points) Clean up your reconstructed images from part (e) by replacing all the negative values by 0, and replacing all values greater than 1 by 1. You can do this in a single line of Python code with the `numpy` functions `maximum` and `minimum`. Display 16 of the cleaned-up images in a single figure in a 4×4 grid. If you have done everything correctly, they should look much less noisy than before clean up.

(i) (5 points) The compression matrix, A, returned by `matfact` in part (d) was computed from the MNIST training images. However, it can be used to compress any set of similar images. To see this, use A to compress and then reconstruct the MNIST *test* images. You should *not* use the function `matfact` in any part of this question, since this will change matrix A. The idea is to use a compression matrix that was trained on one set of images to compress a different set of images. Compression and reconstruction can be done in one line of Python code each. Compute and print out the MSE of the reconstructed test images. Display 16 of the reconstructed images in a single figure in a 4×4 grid. Clean up the reconstructed images and display 16 of them in a single figure in a 4×4 grid.

To obtain the MNIST test data, load the MNIST pickle file from Assignment 1:

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

Recall that `data.train.images` is a NumPy array containing the MNIST training data. Likewise, `data.test.images` is a Numpy array of dimensions 10000×784 containing the test data.

If you have trouble loading the file, then download (and uncompress) the alternate file `npMnistTest.npy.zip` from the course web page. Start the python interpreter and import `numpy`. You can then read the file with the following command:

```
Xtest = numpy.load('npMnistTest.npy')
```

The variable `Xtest` will now contain the test data.

University of Toronto Mississauga
CSC 338 - Numerical Methods

Cover sheet for Assignment 3

Complete this page and hand it in with your assignment.

Name: _____
(Underline your last name)

Student number: _____

I declare that the solutions to Assignment 3 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: _____