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

Assignment 1

Due date: Tuesday Feb 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 four 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.*

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. 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. (7 points total) Assume a floating point number system with base $\beta = 10$, $p = 4$ significant digits, and an exponent range of ± 15. The system uses chopping. What is the result of each of the following floating-point arithmetic operations (1 point each):

   (a) $15 + 10^{-2}$

   (b) $1.234 + 10^{-5}$

   (c) $13 + 10^4$

   (d) $157 + 10^5$

   (e) $1.234 \times 10^3 + 5.678 \times 10^{-3}$

   (f) $10^5 \times 10^{12}$

   (g) $10^{-6}/10^{12}$

2. (6 points total) What are the approximate absolute and relative errors in approximating π^2 by each of the quantities below (2 points each). Show your work.

   (a) 9.8

   (b) 9.869

(c) $(22/7)^2$

3. (9 points total) Consider the problem of evaluating the function $f(x) = x^3 - x^2 - 2x$. Suppose there is a small error, h, in the value of x.

   (a) (2 points) Estimate the absolute error in $f(x)$?

   (b) (2 points) Estimate the relative error in $f(x)$?

   (c) (1 point) Are these errors examples of computational error or propagated data error?

   (d) (2 points) Estimate the condition number for this problem. Simplify your answer.

   (e) (2 points) For what values of the argument x is this problem highly sensitive?

4. (12 points total) The sine function is given by the infinite series

$$\sin(x) \;=\; x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \;=\; \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

   (a) (6 points) Using Python, compute the (absolute) forward and backward errors if we approximate the sine function by the first two terms in the series, i.e., by $f(x) = x - x^3/3!$, for $x = 0.1$, 0.5, 1.0, 2.0, 3.0 and 4.0? You may find the NumPy functions `sin` and `arcsin` useful. (Recall that $\sin^{-1} = \arcsin$). Why do $x = 3.0$ and 4.0 not produce a backward error? Hand in your Python code and output.

   (b) (1 point) Is the forward error an example of computational error or propagated data error?

   (c) (5 points) Using Python, plot $f(x)$ v.s. x for 1000 values of x uniformly spaced between -3 and 3. Plot $\sin(x)$ on the same axes. Use red for $f(x)$ and blue for $\sin(x)$. Title the figure, "Question 4(c): f(x) and sin(x)". Do *not* use iteration. (See the tips at the start of this assignment.) You may find the NumPy function `linspace` useful. Hand in your Python program and the plot.

5. (24 points total) This question builds on the previous one. In particular, you will approximate the function $\sin(x)$ by the following function, for some value of N:

$$f(x) \;=\; \sum_{n=0}^{N} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

More specifically, you will use the following algorithm:

```
Algorithm 1:
    t = x
    y = x
    negXsq = -x**2
    for n from 1 to N
        t = t*negXsq/[2n(2n+1)]
        y = y+t
```

Hand in your Python programs and plots in the questions below.

(a) (10 points) Using mathematical induction, give a detailed proof that the final value of `y` in Algorithm 1 is equal to $f(x)$ (assuming infinite-precision arithmetic).

(b) (2 points) Implement Algorithm 1 in Python.

In the remainder of this question, your programs should *not* use iteration. (See the tips at the start of this assignment.)

(c) (2 points) Using your implementation of Algorithm 1 with `N=6`, plot $f(x)$ v.s. x for 1000 values of x uniformly spaced between -7.0 and 7.0. Make a similar plot with `N=11` for x between -11.0 and 11.0. The centre of each curve should look like $\sin(x)$, but one end of the curve should rise dramatically, and the other end should drop dramatically. Title the two figures, "Question 5(c): f(x), N=6" and, "Question 5(c): f(x), N=11", respectively. You may find the function `numpy.linspace` useful.

(d) (3 points) Consider the dramatic rise and drop at the ends of the curves in part (c). Is this an example of rounding error or truncation error (as described in Section 1.2.4 of Heath)? Explain your answer.

(e) (2 points) Using your implementation of Algorithm 1 with `N=100`, plot $f(x)$ v.s. x for 1000 values of x uniformly spaced between 0.0 and 30.0. Do the same thing for the NumPy function `sin(x)`. The two curves should look the same. Title the two figures, "Question 5(e): f(x), N=100" and, "Question 5(e): sin(x)", respectively.

(f) (1 point) Using your implementation of Algorithm 1 with `N=100`, plot $f(x)$ v.s. x for 1000 values of x uniformly spaced between 30.0 and 40.0. The left end of the curve should look like $\sin(x)$, but right end should be noisy. Title the figure, "Question 5(f): f(x), N=100".

(g) (1 point) Using your implementation of Algorithm 1 with `N=100`, plot $f(x)$ v.s. x for 1000 values of x uniformly spaced between 40.0 and 50.0. The right end of the curve should be so noisy that the vertical axis extends from at least $-30,000$ to at least $+30,000$. Title the figure, "Question 5(g): f(x), N=100".

(h) (3 points) Why are the curves in parts (f) and (g) noisy? Is the noise due to rounding error or truncation error (as described in Section 1.2.4 of Heath)? Explain your answer.

6. (10 points total) Consider the two mathematically equivalent expressions: $4x^2 - 1$ and $(2x-1)(2x+1)$. For what values of x is there a substantial difference in the accuracy of the two expressions? In this case, which expression is more accurate? Give a clear and mathematically precise explanation. Hint: consider cancellation error.

7. (15 points) Consider the expression $(2.0^n + 10.0) - ((2.0^n + 5.0) + 5.0)$. Using Python, find all positive integers n for which the value of this expression is non-zero. Using the

rounding rules described in Section 1.3.4 of Heath, give a *detailed* explanation of why the expression takes on these particular non-zero values. Why is the expression zero for all other values of n? You may assume that Python uses IEEE Double Precision for floating-point arithmetic (*i.e.*, round to even in base 2 with a mantissa of 53 bits). Hint: look at the rounding error produced by each of the three additions.

8. (8 points total) Classify each of the following matrices, A, as well-conditioned or ill-conditioned and justify your answers. (2 points each)

   (a) $\begin{pmatrix} 3 & 0 \\ 0 & 3^{-30} \end{pmatrix}$

   (b) $\begin{pmatrix} 3^{20} & 0 \\ 0 & 5^{30} \end{pmatrix}$

   (c) $\begin{pmatrix} 4^{40} & 0 \\ 0 & 2^{80} \end{pmatrix}$

   (d) $\begin{pmatrix} 5^{-17} & 5^{-16} \\ 5^{-18} & 5^{-17} \end{pmatrix}$

9. (17 points total) This simple warm-up question is meant to illustrate the vast difference in execution speed between iteration in Python (which is slow) and matrix operations in Numpy (which are fast). *To receive any marks for this question, you must do part (c).*

   (a) (6 points) Write a Python function `kindofsquare(A)` that computes the matrix product AA^T, where A is a matrix and A^T is its transpose. Recall that if $C = AB$ is a product of two matrices, then the ij^{th} element of `C` is given by

   $$C_{ij} = \sum_k A_{ik} B_{kj}$$

   Your function will need to use loops. It should not use any NumPy operations other than `shape` (to determine the dimensions of A) and `zeros` (to initialize the product matrix).

   (b) (6 points) Write a Python function `compare(I,K)` to measure execution speed. Specifically, the function should do the following:

   - Use the function `randn` in `numpy.random` to create a random matrix, A, of size $I \times K$.
   - Use your function `kindofsquare` to compute AA^T. Call the result $B1$
   - Use the function `time.time` to measure the execution time of `kindofsquare`.
   - Print out the execution time.
   - Use the `numpy` functions `matmul` and `transpose` to compute AA^T. Call the result $B2$.

- Use the function `time.time` to measure the execution time of computing AA^T.
- Print out the execution time.
- Compute and print out the magnitude of the difference between $B1$ and $B2$:

$$\sum_{ij} |B1_{ij} - B2_{ij}|$$

  Do *not* use iteration to compute this magnitude. Instead, use only NumPy operations. You may find the functions `sum` and `abs` in `numpy` useful.

If your function `kindofsquare` is working correctly, the last step should produce a very small number (much less than 1). You should also find that `kindofsquare` is much slower than using NumPy operations (*thousands* of times slower).

(c) (5 points) Run `compare(1000,10)`, `compare(1000,100)` and `compare(1000,1000)`, and hand in the printed results. (Depending on the speed of your computer, the last one may take several minutes to execute.) Be sure it is clear which measurement each printed value refers to. In each case, how many floating-point multiplications does `kindofsquare` perform?

Because loops and iteration in Python are so slow, your programs in the rest of this assignment should avoid using them. Instead, you should use matrix and vector operations in NumPy wherever possible.

In the rest of this assignment and the remaining assignments of this course, you will build a simple machine-learning program to process hand-written digits. In this question, you will begin by familiarizing yourself with the data and performing elementary matrix operations on it.

Download the MNIST data file (`tfMnist.pickle.zip`) from the course web page. Uncompress it (if your browser did not do so automatically). Start the Python interpreter and import the `pickle` module. You can then read the file with the following command:

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

(`'rb'` opens the file for reading in binary.) The variable `data` is an object containing three data sets, called the training, validation and test sets. For now, we will only use the training set. In particular, `data.train.images` is a NumPy array with 55,000 rows and 784 columns. Each row represents an image of a handwritten digit.

If you have trouble openning the file, then download (and uncompress) the alternate file `npMnist.npy.zip` instead. Start the python interpreter and import `numpy`. You can then read the file with the following command:

```
data = numpy.load('npMnist.npy')
```

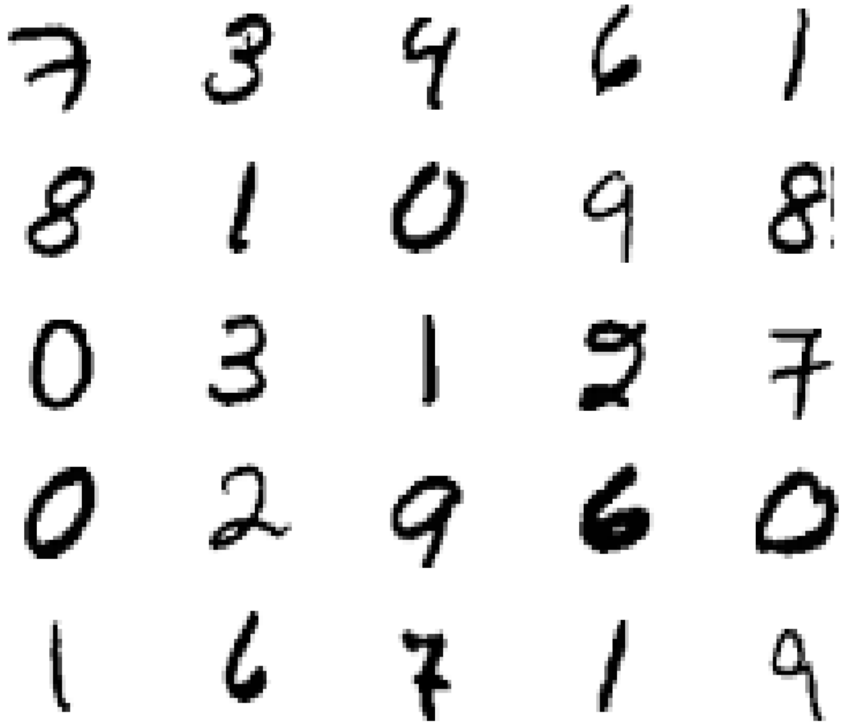

Figure 1: Some MNIST hand-written digits

The variable `data` will now contain a single NumPy array of dimensions 55000×784 with all the MNIST training images.

Although each MNIST digit is stored as a row vector with 784 components, it actually represents an array of pixels with 28 rows and 28 columns ($784 = 28 \times 28$). To view a digit, you must first convert it to a 28×28 array using the function `numpy.reshape`. You can check the dimensions of a NumPy array by using the function `numpy.shape`. (Alternatively, given a NumPy array, `A`, you can use `A.shape` and `A.reshape`.) To display a 2-dimensional array as an image, you can use the function `imshow` in `matplotlib.pyplot`. To see an image in black-and-white, add the keyword argument `cmap='Greys'` to `imshow`. To remove the smoothing and see the 784 pixels clearly, add the keyword argument `interpolation='nearest'`. Try displaying a few digits. (Do not hand them in.) Figure 1 shows the first 25 digits in the file.

10. (13 points total) Permutation Matrices.

(a) (5 points) Let `perm = [3,2,0,1]`. `perm` is a permutation of the list `[0,1,2,3]`. If `A` is a 4×4 matrix, then the NumPy expression `A[:,perm]` permutes the columns of `A`. That is, it rearranges them so that column 3 becomes column 0, column 2 becomes column 1, etc. Likewise, the expression `A[perm,:]` permutes the rows of `A`. Use this idea to construct a 4×4 permutation matrix, P, starting from the identity matrix. Print P. Also print PP^T. The latter result should be the identity matrix, thus showing that $P^T = P^{-1}$. You may find the functions `identity` and `transpose` in `numpy` useful.

(b) (8 points) Create a random permutation matrix P of dimension 784×784. Choose a MNIST digit, x, and display it. Let $y = Px$, and display it. You should find that the digit has become scrambled. Finally, let $z = P^T y$ and display it. You should find that the original digit has been restored, again demonstrating that $P^T = P^{-1}$. Title the three figures, "Question 10(b): an MNIST digit", "Question 10(b): permuted digit", and "Question 10(b): restored digit", respectively. Use the function `axis` in `matplotlib.pyplot` to turn off the axes in your figures. You may find the function `permutation` in `numpy.random` useful.

11. (18 points total) In this question, let x be the MNIST digit that you chose in Question 10(b). Title each figure generated below with the question number, e.g., "Question 11(b)".

(a) (3 points) Using the function `rand` in `numpy.random` construct a random 784×784 matrix, A. Compute and print out the condition number of A. With very high probability, it will be greater than 10^4. (If it is not, then generate a new matrix A.) Let $y = Ax$ and display y. It should look like random noise. You may find the function `cond` in `numpy.linalg` useful.

(b) (2 points) Compute A^{-1} and let $z = A^{-1}y$. Display z. It should look exactly like x. You may find the function `inv` in `numpy.linalg` useful.

(c) (5 points) Solve $Az = y$ using LU factorization and foward and back substitution. Display z. It should look exactly like x. You may find the functions `lu` and `solve_triangular` in `scipy.linalg` useful. Using partial pivoting, `lu` decomposes a matrix into triangular matrices, L and U, and a permutation matrix, P.

(d) (1 point) Let $B = AA^T$. Compute and print out the condition number of B. You should find that $cond(B) = cond(A)^2$.

(e) (1 point) Let $C = BB^T$. Compute and print out the condition number of C. You should find that $cond(C) = cond(B)^2$.

(f) (1 point) Let $y = Cx$. Display y. It should look like random noise.

(g) (1 point) Let $z = C^{-1}y$. Display z. It should also look like random noise.

(h) (4 points) Why are the values of z in parts (b) and (g) different? In particular, why is the value in (b) right, while the value in (g) is wrong?

12. (29 points total) In this question, you will implement a Python function `spock(A,B,C,x)` that returns $A^{-1}(B + C^{-1})(I + 3B)x$. Here, A, B and C are square matrices, and x is a column vector. I is the identity matrix. You will implement the function in two different ways and compare them on the MNIST data.

    Implement your `spock` function so that you can pass it a matrix, `X`, instead of a single vector, `x`. In this case, the `spock` function should be applied to each column of the matrix and return a matrix of the results. If you implement `spock` for a single vector, `x`, using NumPy operations, it will probably work properly for a matrix, `X`, as well. No iteration is needed.

    Do not use any iteration in your programs. *To receive any marks for this question, you must do parts (d) and (e).*

    (a) (4 points) Implement the `spock` function by computing matrix inverses (as in Question 11(b)). When given a vector, `x` (not a matrix), your implementation should not do any matrix-matrix multiplications (which are very slow). It may, however, do matrix-vector multiplications (which are much faster). Call this implementation `spock1`.

    (b) (10 points) Implement the `spock` function without computing any matrix inverses. Instead, solve linear equations by doing LU decomposition and forward and backward substitution (as in Question 11(c)). Again, when given a vector, `x`, your implementation should not do any matrix-matrix multiplications (just matrix-vector multiplications). Call this implementation `spock2`.

    (c) (10 points) How many floating-point multiplications do your implementations in (a) and (b) do when given a vector, `x`? How many do they do when given a matrix, `X`? Justify your answers. Do not count any multiplications used to do permutation. You may assume that inverting a $n \times n$ matrix requires n^3 multiplications. (If your implementations are correct, then `spock2` will do fewer mutliplications than `spock1` when the matrices are large.)

    (d) (3 points) Implement a Python function `compareSpocks(A,B,C,X)`, where `X` is a matrix of column vectors. Your compare function should return

    $$\|\texttt{spock1(A, B, C, X)} - \texttt{spock2(A, B, C, X)}\|/N$$

    where N is the number of columns in `X`. Here, $\|\cdot\|$ denotes a kind of matrix norm. In partricular, if Y is a matrix, then $\|Y\| = \sum_{ij} |Y_{ij}|$.

    (e) (2 points) Create three random matrices, A, B, C, of dimensions 784×784, and use them and your function `compareSpocks` to compare the results of `spock1` and `spock2` on the MNIST training data. Print the result. If you have done everything correctly, the result should be a very small number (much smaller than 1).

University of Toronto Mississauga
CSC 338 - Numerical Methods

Cover sheet for Assignment 1

Complete this page and hand it in with your assignment.

Name: ______________________________
(Underline your last name)

Student number: ______________________________

I declare that the solutions to Assignment 1 that I have handed in are solely my own work, and they are in accordance with the University of Toronto Code of Behavior on Academic Matters.

Signature: ______________________________