# CSC338. Assignment 1

Due Date: Monday, February 8, 10pm

### What to Hand In

Please hand in 2 files:

- Python File containing all your code, named `a1.py`.
- PDF file named `a1_written.pdf` containing your solutions to the written parts of the assignment. Your solution can be hand-written, but must be legible. Graders may deduct marks for illegible or poorly presented solutions.

If you are using Jupyter Notebook to complete the work, your notebook can be exported as a .py file (File -> Download As -> Python). Your code will be auto-graded using Python 3.6, so please make sure that your code runs. There will be a 10% penalty if you need a remark due to small issues that renders your code untestable.

**Make sure to remove or comment out all matplotlib or other expensive code before submitting your code! Expensive code can render your code untestable, and you will incur the 10% penalty for remark.**

Submit the assignment on **MarkUs** by 10pm on the due date. See the syllabus for the course policy regarding late assignments. All assignments must be done individually.

```
import math
import numpy as np
```

## Question 1.

You will have enough information to solve this question after the week 1 lecture.

### Part (a) – 3pt

What is the approximate absolute and relative errors in approximating $\pi$ by the following values?

1. 3
2. 3.14
3. 3.1416

You may assume that the "true" value of $\pi$ is represented by `math.pi`. Save the results in the variables `q1_abs`, `q1_rel`, `q2_abs`, `q2_rel`, and `q3_abs`, `q3_rel` below.

```
q1_abs = None
q1_rel = None
q2_abs = None
q2_rel = None
q3_abs = None
q3_rel = None
```

### Part (b) – 2pt

$\pi$ can be represented as the infinite series $4\sum_{i=0}^{\infty}\frac{(-1)^i}{1+2i}$. Say you want to approximate $\pi$ using only the first 100 terms of the sum, what kind of error does this introduce? Explain your reasoning.

Include your answer in your PDF file.

### Part (c) – 3pt

Consider the function $f(x) = \frac{x - \tan(x)}{x^3}$, which is also implemented and plotted below. What is $f(0.00000001)$? Given that $f$ is continuous when $x \in (-\frac{\pi}{2}, 0) \cup (0, \frac{\pi}{2})$, what should $f(0.00000001)$ be? Why does Python compute such inaccurate value of $f(0.00000001)$?

Include your answer in your PDF file.

```python
def f(x):
    return (x - math.tan(x))/math.pow(x, 3)

def plot_f():
    import matplotlib.pyplot as plt
    xs = [x for x in np.arange(-1.25, 1.25, 0.025) if abs(x) > 0.025]
    ys = [f(x) for x in xs]
    plt.plot(xs, ys, 'bo')
    # plt.show() # uncomment this if you are not using a jupyter notebook

# plot_f() # please comment this out before submitting, or your code might be untestable
```

### Part (d) – 2pt

Define a Python function `f2(x)` to computes accurate values of $f(x) = \frac{x - \tan(x)}{x^3}$ for both small positive values of $x$. The relative error should be no more than 1% for those values of $x$.

```python
def f2(x):
    return None
```

### Part (e) – 4pt

The exponential function is given by the infinite series

$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$

Compute the absolute forward and backwards error if we approximate the exponential function by the first four terms of the series for elements of the array `xs` below.

Save your solution in the array `q3_forward` and `q3_backward`, so that `q3_forward[i]` and `q3_backward[i]` are the absolute forward and backward errors corresponding to the input `xs[i]`.

You may find the functions `math.exp` and `math.log` helpful. You may assume that the true value of $e^x$ can be computed using the function `math.exp`.

```python
xs = [-0.5, 0.1, 0.7, 3.0]

q3_forward  = [None, None, None, None]
q3_backward = [None, None, None, None]
```

### Part (f) – 3pt

Consider the function $f(x) = \frac{\sin(x)}{x}$, it is continous everywhere except at $x = 0$. Find the condition number $K_f(x)$ of $f$ as a function of $x$. Around what $x$ values would you consider evaluating $f(x)$ to be highly sensitive? The funtion $\frac{\sin(x)}{x}$ is very important in Physics, Signal Processing, and Fourier Analysis, for further reading, refer to the Wikipedia article on the sinc function.

Include your answer in your PDF file.

## Question 2.

You will have enough information to solve this question after the week 2 lecture.

In this question, you will be implementing a normalized floating point system $F(\beta = 4, p = 6, L = -4, U = 4)$.

```
# Constants we will use in our code.
BASE = 4
PRECISION = 6
L = -4
U = +4
```

We'll represent a floating-point number in this system as a 3-tuple `(sign, mantissa, exponent)`, with `sign` being either `1` or `-1`, `mantissa` being a list of length $p$, and `exponent` being a value between $L$ and $U$.

```
example_float_num = (1, [2, 1, 2, 3, 3, 3], -2)
```

### Part (a) – 2pt

Write a function `to_python` that takes a floating-point number tuple and returns a Python floating-point representation of its value.

```
def to_python(float_num):
    """
    Return a Python floating point representation of this float_num.

    >>> float_val = (-1, [2,2,0,3,0,0], 2)
    >>> to_python(float_num)
    -40.75
    """

    # your code goes here
    return None
```

### Part (b) – 4pt

Write a function `add_floats` that takes two floating-point number tuple **of the same sign** and returns a Python floating-point representation of their sum. You should throw a `ValueError` if the addition is unsuccessful.

```
def add_floats(float_num1, float_num2):
    """
    Return a Python floating point representation of this float_num.

    >>> add((1, [2,0,0,0,0,0], 0), (1, [1,0,0,0,0,0], 0))
    (1, [3,0,0,0,0,0], 0)
    """
    # your code goes here
    return None
```

### Part (c) – 2pt

Is our implementation of floating point addition associative? If yes, prove it. If not, provide a counter-example.

Include your answer in your PDF file.

### Part (d) – 3pt

Create a variable `all_floats` that contains a list of all normalized floating point numbers in our floating point system.

In your writeup, explain what is the value of `len(all_floats)`, and why you believe the value to be correct.

```
all_floats = []
```

### Part (e) – 2pt

The decimal number 518.6875 is not representable in our floating point system exactly. Round 518.6875 to the nearest machine number in our floating point system using chopping. Save the floating point tuple obtained using chopping in the variable `chop`.

```
chop = None
```

## Question 3.

You will have enough information to solve this question after the week 2 lecture.

### Part (a) – 3pt

We want to try and find the roots of the polynomial $f(x) = 2^{-11}x^2 + 2^{10}x + 2^{-11}$, let's label these roots as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$. Does catastrophic cancellation occur when computing $x_1$? How about $x_2$? Explain your reasoning.

Include your answer in your PDF file.

### Part (b) – 2pt

For the polynomial in part (f), let's use an alternative formula to compute the roots. Let $x_1 = \frac{2c}{-b-\sqrt{b^2-4ac}}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$. Save the result of finding $x_1$ using the formula in part(f) in the variable `old_root`, save the result of finding $x_1$ using the new formula in the variable `new_root`, and save the absolute error in finding this root in the variable `abs_err` (assuming our new formula for finding $x_1$ is the true value).

```
old_root = None
new_root = None
abs_err = None
```

### Part (c) – 1pt

Consider the functionn $z(n)$ below:

```
def z(n):
    a = pow(4.0, n) + 18.0
    b = (pow(4.0, n) + 9.0) + 9.0
    return a-b
```

Use Python to find the only positive integer for which the value of `z(n)` is non-zero and save the result in the variable `nz` below.

```
nz = None
```

**Part (d) – 4pt**

Explain why `zn` from part (h) makes `z(zn)` non-zero. 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 in the addition.

## Question 4.

You will have enough information to solve this question after the week 3 lecture.

The use of any functions in `np.linalg` is strictly forbidden for this question.

We will write the following functions together during the tutorials.

```
def eliminate(A, b, k):
    """Eliminate leading coefficients in the rows below the k-th row of A
    in the system np.matmul(A, x) == b. The elimination is done in place."""
    n = A.shape[0]
    for i in range(k + 1, n):
        m = A[i, k] / A[k, k]
        A[i, :] = A[i, :] - m * A[k, :]
        b[i] = b[i] - m * b[k]

def gauss_elimination(A, b):
    """Convert the system np.matmul(A, x) == b into the equivalent system np.matmul(U, x) == c
    via Gaussian elimination. The elimination is done in place."""
    if A.shape[0] == 1:
        return
    eliminate(A, b, 0)
    gauss_elimination(A[1:, 1:], b[1:])

def back_substitution(A, b):
    """Return a vector x with np.matmul(A, x) == b, where
        * A is an nxn numpy matrix that is upper-triangular and non-singular
        * b is an nx1 numpy vector
    """
    n = A.shape[0]
    x = np.zeros_like(b, dtype=np.float)
    for i in range(n-1, -1, -1):
        s = 0
        for j in range(n-1, i, -1):
            s += A[i,j] * x[j]
        x[i] = (b[i] - s) / A[i,i]
    return x

def solve(A, b):
    """Return a vector x with np.matmul(A, x) == b, where
        * A is an nxn numpy matrix that is non-singular
        * b is an nx1 numpy vector
    """
    gauss_elimination(A, b)
    return back_substitution(A, b)
```

**Part (a) – 3pt**

Classify each of the following matrices `A1`, `A2`, and `A3` as well-conditioned or ill-conditioned. You should do this question by hand instead of writing a function to compute condition numbers.

Save the classifications in the Python list called `conditioning`. Each item of the array should be either the string "well" or "ill".

```
A1 = np.array([[1, 2],
               [2, 1]])
A2 = np.array([[1, 2],
               [2, 4.001]])
A3 = np.array([[3, 7],
               [0.1, 59472]])

conditioning = []
```

### Part (b) – 4pt

Solve the following system $A\vec{x} = \vec{b}$ using Gaussian elimination **by hand**. Include your answer and all your steps in your PDF file.

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 7 & 9 \\ 4 & 11 & 17 \end{bmatrix}, \vec{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

### Part (c) – 3pt

Write a function `forward_substitution` that solves the lower-triangular system $A\vec{x} = \vec{b}$.

Hint: This function should be very similar to the `back_substitution` function.

```
def forward_substitution(A, b):
    """Return a vector x with np.matmul(A, x) == b, where
        * A is an nxn numpy matrix that is lower-triangular and non-singular
        * b is a nx1 numpy vector

    >>> A = np.array([[2., 0.],
                      [1., -2.]])
    >>> b = np.array([1., 2.])
    >>> forward_substitution(A, b)
    array([ 0.5 , -0.75])
    """
    # TODO: complete this function
```

### Part (d) – 4pt

Write a function `elementary_elimination_matrix` that returns the $k$-th elementary elimination matrix ($M_k$ in your notes).

You may assume that `A[i, j] == 0 for i > j, j < k - 1`. (the subtraction in $k - 1$ is because indicies begin at 0 in Python).

```
def elementary_elimination_matrix(A, k):
    """Return the elements below the k-th diagonal of the
    k-th elementary elimination matrix.

    (Do not use partial pivoting, since we haven't
    introduced the idea yet.)

    Precondition: A is a non-singular nxn numpy matrix
```

```
                A[i,j] = 0 for i > j, j < k - 1

    As always, these examples are for your understanding only.
    The actual Python output might differ slightly.

    >>> A = np.array([[2., 0., 1.],
                      [1., 1., 0.],
                      [2., 1., 2.]])
    >>> elementary_elimination_matrix(A, 1)
    np.array([[1., 0., 0.],
              [-0.5, 1., 0.],
              [-1., 0., 1.]])
    >>> A = np.array([[2., 0., 1.],
                      [0., 1., 0.],
                      [0., 1., 2.]])
    >>> elementary_elimination_matrix(A, 2)
    np.array([[1., 0., 0.],
              [0., 1., 0.],
              [0., -1., 1.]])
    """
    # TODO: complete this function
```

**Part (e) – 3pt**

Write a function `lu_factorize` that factors a matrix $A$ into its upper and lower triangular components. Use the function `elementary_elimination_matrix` as a helper.

```
def lu_factorize(A):
    """Return two matrices L and U, where
            * L is lower triangular
            * U is upper triangular
            * and np.matmul(L, U) == A
    >>> A = np.array([[2., 0., 1.],
                      [1., 1., 0.],
                      [2., 1., 2.]])
    >>> L, U = lu_factorize(A)
    >>> L
    array([[1. , 0. , 0. ],
           [0.5, 1. , 0. ],
           [1. , 1. , 1. ]])
    >>> U
    array([[ 2. ,  0. ,  1. ],
           [ 0. ,  1. , -0.5],
           [ 0. ,  0. ,  1.5]])
    """
    # TODO: complete this function
```

**Part (f) – 3pt**

Prove that the triangle inequality holds for the Frobenius norm. That is, $||A+B||_F \le ||A||_F + ||B||_F$ for all square matrices $A$ and $B$ of the same size. The Frobenius norm is defined as $||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{i,j}|^2}$ for an $n \times n$ matrix $A$.