CSC338. Homework 4

Due Date: Monday, June 8, 11:59pm

What to Hand In

Please hand in 3 files:

- Python File containing all your code, named `hw4.py`.
- PDF file named `hw4_written.pdf` containing your solutions to the written parts of the assignment.
- TeX file named `hw4_written.tex`

Your code will be auto-graded using Python 3.8, so please make sure that your code runs. There will be a 20% 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 homework!

Submit the assignment on **MarkUs** by 11:59pm 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

For this question, we will again start from code from tutorial 3.

```
# Code from tutorial 3

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

def gauss_elimination(A, b):
    """Return a vector x with np.matmul(A, x) == b using
    the Gauss Elimination algorithm, without partial pivoting."""
    for k in range(A.shape[0] - 1):
```

```
        eliminate(A, b, k)
    x = backward_substitution(A, b)
    return x
```

Part (a) – 1 pt

Solve the system $Ax = b$ for the values of A and b below using the function `gauss_elimination` from last time. Save the solution you obtain in the variable `soln_nopivot`.

```
e = pow(2, -100)
A = np.array([[e, 1],
              [1, 1]])
b = np.array([1 + e, 2])

soln_nopivot = None
```

Part (b) – 2 pt

Write a helper function `partial_pivot` that performs partial pivoting on `A` at column `k`, so that the function `gauss_elimination_partial_pivot` performs Gauss Elimination with Partial Pivoting.

```
def partial_pivot(A, b, k):
    """Perform partial pivoting for column k. That is, swap row k
    with row j > k so that the new element at A[k,k] is the largest
    amongst all other values in column k below the diagonal.

    This function should modify A and b in place.
    """
    # TODO

def gauss_elimination_partial_pivot(A, b):
    """Return a vector x with np.matmul(A, x) == b using
    the Gauss Elimination algorithm, with partial pivoting."""
    for k in range(A.shape[0] - 1):
        partial_pivot(A, b, k)
        eliminate(A, b, k)
    x = backward_substitution(A, b)
    return x
```

Part (c) – 1 pt

Solve the system $Ax = b$ for the values of A and b below using `gauss_elimination_partial_pivot`. Save the solution you obtain in the variable `soln_pivot`.

```
e = pow(2, -100)
A = np.array([[e, 1],
              [1, 1]])
b = np.array([1 + e, 2])

soln_pivot = None
```

Part (d) – 2 pt

Do your answers in parts (a) and (d) match? If not, which is the correct answer? Include your explanation in the PDF write-up.

Question 2

Part (a) – 3 pt

Consider the following matrices `M1`, `M2`, and `M3`. Compute each of their L_1, L_2, and L_∞ norms. Save the results in the variables below.

For the L_2 norm you may find the function `np.linalg.norm` helpful. You can compute the L_1 and L_∞ norms either by hand or write a function.

```
M1 = np.array([[3., 0.],
               [-4., 2.]])

M2 = np.array([[2., -2., 0.3],
               [0.5, 1., 0.9],
               [-4., -2., 5]])

M3 = np.array([[0.2, -0.2],
               [1.0, 0.2]])

# fill in these answers
M1_l_1 = 0
M1_l_2 = 0
M1_l_infty = 0
M2_l_1 = 0
M2_l_2 = 0
M2_l_infty = 0
M3_l_1 = 0
M3_l_2 = 0
M3_l_infty = 0
```

Part (b) – 4 pt

Show that an induced matrix norm $||\cdot||$ has the property

$$||A + B|| \leq ||A|| + ||B||$$

Part (c) – 4 pt

Is it true that for a vector, $||v||_\infty \leq ||v||_1$? What about for a matrix: is it true that for a matrix, $||M||_\infty \leq ||M||_1$? Include your solution and justification in your pdf writeup.

Question 3

Part (a) [3 pt]

Write a function `matrix_condition_number` that computes the condition number of a 2×2 matrix. Use the

$$L_1$$

matrix norm.

```
def matrix_condition_number(M):
    """
    Returns the condition number of the 2x2 matrix M.
    Use the $L_1$ matrix norm.
```

```
    Precondition: M.shape == [2, 2]
                  M is non-singular

    >>> matrix_condition_number(np.array([[1., 0.], [0., 1.]]))
    1
    """
```

Part (b) [2 pt]

Classify each of the following matrices `A1`, `A2`, `A3` and `A4`, as well-conditioned or ill-conditioned.

You may do this question either by hand, or by using the function above.

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

```python
A1 = np.array([[3, 0],
               [0, pow(3, -30)]])
A2 = np.array([[pow(3, 20), 0],
               [0,          pow(5, 50)]])
A3 = np.array([[pow(4, -40), 0],
               [0,           pow(2, -80)]])
A4 = np.array([[pow(5, -17), pow(5, -16)],
               [pow(5, -18), pow(5, -17)]])

# whether A1, A2, A3, A4 is "well" or "ill" conditioned
conditioning = [None, None, None, None]
```

Part (c) [2 pt]

It should be immediate obvious that the matrix

$$\begin{bmatrix} 100 & 2 \\ 201 & 4 \end{bmatrix}$$

is ill-conditioned. Explain why. Include your answer in your pdf write-up.

Part (d) [2 pt]

Suppose that A and B are two $n \times n$ matrices, and both are well-conditioned. Is $A(B^{-1})$ also well-conditioned? Why or why not? Include your answer and justifications in your pdf write-up. Be specific.

Part (e) [4 pt]

Describe an efficient algorithm to compute $d^T B^T A^{-1} B d$

Where:

- A is an invertible $n \times n$ matrix,
- B is an $n \times n$ matrix, and
- d is an $n \times 1$ vectors

Be clear and specific. Include your strategy in your pdf write-up.