Assignment 2

Due Date: February 4, 8:59pm

Please see the guidelines at https://www.cs.toronto.edu/~lczhang/338/homework.html

What to Hand In

Please hand in 2 files:

- Python File containing all your code, named `csc338_a2.py`. 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.
- PDF file named `csc338_a2_written.pdf` containing your solutions to the written parts of the assignment, which includes Q2(e,f), Q3(c,d), Q4(b), Q5(e), and Q6. Your solution can be hand-written, but must be legible. Graders may deduct marks for illegible or poorly presented solutions.

Submit the assignment on **MarkUs** by 9pm on the due date. See the syllabus for the course policy regarding late assignments. All assignments must be done individually.

Question 1. Diagonal Linear Systems [3 pt]

Write a function `solve_diagonal(A, b)` that takes a diagonal numpy matrix `A` of size $n \times n$, and a numpy array `b` of size n, and returns the solution `x` to the problem `Ax=b`.

Hint: You may find the numpy function `np.diag` helpful.

Question 2. Gauss Elimination [17 pt]

Part (a) [1pt]

Use the code below, which we write in Tutorial 3, to solve the below system. Save your resulting numpy array in the variable `q2_x`

Part (b) [3 pt]

For the values of `A` and `b` below, show the updated value of `A` and `b` after each elimination step when solving the linear system $Ax = b$.

Save those values in arrays `q2_A` and `q2_b` so that `q2_A[0]` shows the elimination of the first column below the diagonal, `q2_A[1]` shows the elimination of the second column below the diagonal, and so on. Similarly, `q2_b[i]` should show the corresponding value of `b` the same number of eliminations.

Hint: You can do this question by hand, or write a different version of the `gauss_elimination` function. You may also find the function `np.copy` helpful.

Part (c) [6 pt]

Write a helper function `partial_pivot` that performs partial pivoting on `A` at column `k`. Your helper function will be called from the function `gauss_elimination_partial_pivot`.

Part (d) [3 pt]

Solve the system $Ax = b$ for the values of A and b below using `gauss_elimination` and then `gauss_elimination_partial_pivot`.

Save the solution you obtain from `gauss_elimination` in the variable `q2d_nopivot`. Save the solution you obtain from `gauss_elimination_partial_pivot` in the variable `q2d_pivot`.

Which is the correct answer? Save the correct answer in the variable `q2d_correct`.

Note: Make sure your code accounts for the fact that both function `gauss_elimination_partial_pivot` and `gauss_elimination` modify their parameters.

Part (e) [2 pt]

Why do we obtain different results from the two algorithms? Include your answer in your pdf writeup. Be specific.

Part (f) [2 pt]

Is the matrix A from part (d) well-conditioned or ill-conditioned? Include your answer and justifications in your pdf writeup. Be specific.

Question 3. Matrix and Vector Norms [15 pt]

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.

Part (b) [4 pt]

Show that $||A||_1 = max_j \sum_{i=1}^{m} |a_{ij}|$. Include your solution in your pdf writeup.

Part (c) [4 pt]

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

Part (d) [4 pt]

Assume x and y are 2-vectors. Is it possible to have $||x||_1 > ||y||_1$ but $||x||_2 < ||y||_2$?

If so, save an example of such vector in the variables `q5_x` and `q5_y`. Otherwise, set both variables to the Python value `None`.

Question 4. Condition Numbers [8 pt]

Part (a) [4 pt]

Classify each of the following matrices `A1`, `A2`, `A3` and `A4`, 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 a Python array called `conditioning`. Each item of the array should be either the string "well" or the string "ill".

Part (c) [4 pt]

Suppose that A is well-conditioned. Is A^2 also well-conditioned? Why or why not? Include your answer and justifications in your pdf writeup. Be specific.

Question 5. LU Factorization [27 pt]

Part (a) [5 pt]

Write a function `forward_substitution` that solves the lower-triangular system $Ax = b$.

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

Part (b) [5 pt]

Write a function `elementary_elimination_matrix` that returns the elements below the k-th diagonal in the k-th elementary elimination matrix. These values corresponds to the values of [-m_{k+1}, ... - m_n] from your notes.

Since Python indices begin at zero, the first elementary elimination matrix is for $k = 0$.

Do not perform any pivoting.

You may assume that `A[i,j] = 0 for i > j, j < k`

Part (c) [10 pt]

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.

Part (d) [3 pt]

Write a function `solve_lu` that solves a linear system Ax=b by * factoring $A = LU$ (using the `lu_factorize` function) * solving $Ly = b$ using forward substitution (using the `forward_substitution` function) * solving $Ux = y$ using backward substitution (using the `backward_substitution` function)

Part (e) [4 pt]

Prove that the matrix

```
A = [[0 1]
     [1 0]]
```

has no LU facotirzation. No lower triangular matrix L and upper triangular matrix U such that $A = LU$.

Include your proof in your pdf writeup.

Question 6. Application [10 pt]

Part (a) [5 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 writeup.

Part (b) [5 pt]

Write a function `invert_matrix` that takes an $n \times n$ matrix `A`, and computes its inverse by solving n systems of linear equations of the form $Ax = b$.

You can use any code that you wrote in this assignment. You **will** be graded for efficiency.

Include your code in **both** your `.py` file **and** your pdf writeup.