Problem Set 6

ASSIGNED: February 16, 2023
DUE: **Friday, February 24, 2023** at the beginning of class.

Problem set guidelines:

1. Each solution must be your own work.
2. All problems that involve python must be completed in Jupyter Notebooks**.** Some problems may not require python and may be better completed with pen and paper.
3. Highlight your final answer when providing numerical results. Provide plots, graphs, and tables of your results when appropriate.
4. **Submission Instructions:** This quarter we will use Gradescope to collect your submissions and grade them. We will only accept a single PDF file with your compiled solutions. Please follow the instructions below to obtain a PDF file from a Jupyter Notebook:
   i. Make sure all your code runs without error, and all figures (if any) show up where intended. We will not be running your code, therefore it is essential that your solutions output and highlight your results. Please be mindful of your line length so that it fits into the PDF layout and your results are clearly shown.
   ii. Select `File->Download as->HTML (.html)`. This will download an HTML version `your_homework.html` of your notebook to your computer, typically in your `Download` folder.
   iii. Open `your_homework.html` in your web browser (just double-click on it). ***Use the `File->Print` command of your browser to produce a PDF file.*** *(Do not produce the PDF in other ways, such as "Export to PDF" or other. Alternative methods will usually produce poor page margins, format headers differently, fail to incorporate figures, and so forth.)*
   iv. **Submit your PDF file** `your_homework.pdf` to Gradescope. Do *not* submit your HTML file.
5. Problem sets are due at the beginning of the class period on the due date listed. Late problem sets will not be accepted.

Note that you can wrap lines of python code using the "\" operator to ensure that all your code is visible within the width of the page. See the following examples:

```
In [1]:
print("demonstra\
tion")
```

```
demonstration
```

```
In [2]:
def factorial(n):
    if n==1:
        return(1)
    return(n*\
        factorial(n-1))

factorial(4)
```

```
Out[2]: 24
```

```
In [3]:
factorial\
(4)
```

```
Out[3]: 24
```

Problem 1 – The computational complexity of linear algebra methods in python

When applying linear algebra methods, we are often working with 1000 x 1000 or greater sized matrices. Understanding the computational complexity of such methods is then paramount in building code that can be applied to solve these systems in finite amounts of time.

** *Note that some of these tests may take a long time (many tens to hundreds of minutes) to run. To receive full credit, you must complete each N, so make sure to reserve time to either run this overnight or as you work on something else.*

(a) Write a function that takes an integer N as input and returns (i) a N x N matrix consisting of random (non-integer) values from -10 to 10 and (ii) a vector of size N x 1 again of random (non-integer) values from -10 to 10. The random numbers should be selected from a uniform distribution.

(b) Using the Gaussian elimination method, calculate the time it takes to solve for x using A matrices and corresponding b vectors randomly generated by your function from part a for each of N = [2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000]. Plot the time for each calculation versus the size of the matrix N on a log – log plot.

(c) Perform a regression on the linear portion of the results (*i.e.*, you may not be able to regress all the data points) to determine the scaling behavior of the computational complexity ($O(N^2)$, $O(N^3)$, etc.) of these methods. Do the results you find match the theoretical expectations for Gaussian elimination? Explain why or why not.

(d) Repeat parts b and c to determine the time it takes to solve the same matrices using the python library function `numpy.linalg.solve`. Provide a plot with your measured results and regression that overlays the results for the numpy library function and that you wrote for Gaussian elimination.

Problem 2 – Decomposition methods for matrix inversion and determinant

Matrix A can be decomposed into triangular matrices L and U such that $A = LU$. For an arbitrarily sized NxN matrix:

(a) Write your own python function that accepts a matrix A of arbitrary size and returns the corresponding matrices L and U.

(b) Write a code in python to compute the inverse of the matrix using LU decomposition.

(c) Demonstrate the operation of your code on the test matrix

$$A = \begin{bmatrix} 6 & 1 & -5 & 3 \\ 2 & 4 & 5 & 6 \\ 2 & 3 & 7 & 1 \\ 1 & 2 & 0 & -2 \end{bmatrix}$$

(d) Compare your function and its results to python's `numpy.linalg.inv` library function.

(e) Using the function that you have written to perform the LU decomposition, calculate the determinant of matrix A.

Problem 3 - Methods to solve sets of linear equations

Solve the following system of equations using the indicated methods:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 - 5x_4 &= -44 \\ 2x_1 + 5x_2 + 4x_3 - x_4 &= 8 \\ x_1 - x_2 + 10x_3 + 2x_4 &= 44 \\ 3x_1 - 2x_2 + 5x_3 - 3x_4 &= -16 \end{aligned}$$

(a) The python library function `np.linalg.solve`.

(b) The Gauss-Seidel method **without** relaxation. Plot $x_n^{(k)}$ versus iteration number.

(c) The Gauss-Seidel method **with** relaxation ($\omega = 0.95$). Plot $x_n^{(k)}$ versus iteration number.

Problem 4 - Solving sets of non-linear equations

Adapted from Chapra and Canale, Problem 6.24.

Determine the positive root of the following simultaneous nonlinear equations:

$$y = x^2 + 1$$

$$y = 2\cos x$$

(a) Plot these functions on the same graph and use a graphical approach to obtain your initial guesses.

(b) Apply the fixed-point iteration method to determine a refined estimate (see Section 6.6.1 of C&C).

(c) Determine a refined estimate of the roots with the two-equation Newton-Raphson method described in Section 6.6.2 of C&C.

Problem 5 – Comparing numerical methods to solve sets of nonlinear equations

Solve this system of equations in python by implementing the following methods from scratch.

$$\begin{aligned} 3x_1x_2 + x_2 - x_3 &= 12 \\ x_1 - x_1^2x_2 + x_3 &= 12 \\ x_1 - x_2 - x_3 &= -2 \end{aligned}$$

(a) Multi-equation Newton-Raphson method (see Sections 6.6 and 9.6 of C&C).

(b) The successive substitution method (see Section 6.1 of C&C).