Linear systems and complexity

Due: Wednesday, February 6th, 12:30pm. Put the written part of the assignment in the drop box. Write your answers out carefully and clearly. Upload the following files to Blackboard:

- `tridag_matvec.py`,
- a script called `compare.py` that produces the plots from 2c and 2d.
- a text file called `README` explaining any other files submitted. If you submit no other files, a `README` file is not needed.

Be sure all files submitted include a comment line with your name and student number, e.g.,

```
# Jean-Jaques Rousseau
# 123456789
```

Also, be a **good programmer** and include comments with a brief description of the functionality, input and output arguments and usage of each function or script. Add some comments that explain what steps are taken. Marks will be awarded or subtracted based on the readability and transparency of your code.
If you worked together with class mates on your code, and a substantial part of the code you submit coincides with theirs, you must list their names in a comment, e.g.

```
# Written in collaboration with Shawn Shawnson and Lea Leason.
```

Failure to do so may qualify your work as plagiarism.
A discussion thread for this assignment is available on Slack. Pose your questions there before approaching the lecturer or TA.

Question 1 **10 marks**

Consider the following matrices and vector:

$$A = \begin{pmatrix} 1 & 2 & -4 \\ 2 & 4 & 1 \\ -2 & -1 & 3 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 0 & -4 \\ 2 & -4 & 1 \\ 2 & 10 & 3 \end{pmatrix} \qquad C = \begin{pmatrix} 2 & 0 \\ 50 & 1 \end{pmatrix} \qquad R = \begin{pmatrix} 2 \\ -3 \\ 1 \end{pmatrix}$$

(**a**) Compute the decomposition $PA = LU$ (by hand!). Show the intermediary steps like we did in lecture 5/6.

(**b**) Use the decomposition you found at (**a**) to solve $Ax = R$ (by hand!).

(**c**) Use Gaussian elimination to solve $Bx = R$ (by hand!). Show intermediary steps like we did in lecture 5.

(**d**) Compute the condition number of C (use Python/SciPy). If we solve $Cx = Q$, where Q is a 2-vector of which we know the entries up to 5 digits of precision, then how accurately can we compute x?

Question 2 **10 marks**

The banded, upper triangular matrix $A \in \mathbb{R}^{n\times n}$ can be written as

$$A = \begin{bmatrix} a_1 & b_1 & c_1 & & & & \\ & a_2 & b_2 & c_2 & & & \\ & & a_3 & b_3 & c_3 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & & a_{n-1} & b_{n-1} \\ & & & & & & a_n \end{bmatrix} \tag{1}$$

where

$$\vec{a} = (a_1, a_2, \ldots, a_{n-2}, a_{n-1}, a_n) \in \mathbb{R}^n,$$
$$\vec{b} = (b_1, b_2, \ldots, b_{n-2}, b_{n-1}) \in \mathbb{R}^{n-1},$$
$$\vec{c} = (c_1, c_2, \ldots, c_{n-2}) \in \mathbb{R}^{n-2}.$$

The entries of $\vec{a}$, $\vec{b}$, and $\vec{c}$ are all assumed to be nonzero.

(**a**) Write a pseudocode for computing the matrix-vector product of A with a vector x. That is, given vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$ that define A as in definition (1), and given a vector $x \in \mathbb{R}^n$, your algorithm should compute the matrix-vector product $\vec{y} = A\vec{x}$.

Your pseudocode should have the following form:

Input: vector $\vec{x} \in \mathbb{R}^n$ and vectors $\vec{a} \in \mathbb{R}^n$, $\vec{b} \in \mathbb{R}^{n-1}$ and $\vec{c} \in \mathbb{R}^{n-2}$.

⋮

Insert pseudocode here

⋮

Output: vector $\vec{y} \in \mathbb{R}^n$ such that $\vec{y} = A\vec{x}$

(**b**) Analyse the complexity of the algorithm from part (**a**). That is, determine how many flops are required to compute the product $\vec{y} = A\vec{x}$ with your algorithm. In terms of "Big-Oh" notation, what is the asymptotic behaviour of your algorithm as n increases?

(**c**) Implement your pseudo-code in as a function called `tridag_matvec.py`. Also, write a script called `compare.py` to generate a tridiagonal test matrix and a test vector for a $n = 10^k$, $k = 2, \ldots, 7$, compute the product and measure the time your code takes to complete. Produce a plot of the time taken versus n on a logarithmic scale, along with your prediction.

(**d**) Repeat the test, but now using the built-in matrix-vector product (`scipy.dot`/`scipy.matmul`) and the matrix A defined as an $n \times n$ matrix with mostly zeros. Plot the time taken in the same plot as for (**c**). Which algorithm is faster? Is the difference as great as you expected?