Linear systems and complexity

Due: Friday, February 12th, 17:00. Push the following files to your GitHub Classroom repository:

- A PDF file, typeset in LaTeX, with answers to question 1 and 2(d).
- A Python function called `banded_matvec.py` for question 2(c). Use the template provided in the assignment repository.
- A Python script called `test_matvec.py` for question 2(d). Use the template provided in the assignment repository.

Question 1 **10 marks**

Consider the following matrices and vector:

$$A = \begin{pmatrix} 1 & -2 & -4 \\ 2 & 4 & 1 \\ -3 & -6 & 3 \end{pmatrix} \qquad C = \begin{pmatrix} 2 & 0 \\ 50 & 1 \end{pmatrix} \qquad R = \begin{pmatrix} 2 \\ -3 \\ 1 \end{pmatrix}$$

(**a**) Compute the decomposition $PA = LU$. Show the intermediary result for L, U and P after the permutation and Gauss elimination for each column.

(**b**) Use the decomposition you found at (**a**) to solve $Ax = R$. Show the solution to the intermediary linear system with the matrix L.

(**c**) 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 **20 marks**

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

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

meaning that $A_{i,j} = 0$ if $j < i-1$ or $j > i+1$. The numbers a_i $(2 \le i \le n)$, b_i $(1 \le i \le n)$, and c_i $(1 \le i \le n-1)$ 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 arrays a, b, and 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: array of n floats x and arrays a, b and c.

$\vdots$

Insert pseudocode here

$\vdots$

Output: array of n floats y 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 `banded_matvec.py`. Also, write a script called `test_matvec.py` to generate a banded test matrix of the form (1) 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**) Add to the script of part (**c**) a comparison to the built-in matrix-vector product (`scipy.dot`/`scipy.matmul`). Define the matrix A as a $n \times n$ 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?