Linear systems and complexity

Due: Tuesday, March 22, 11:59pm.

Push your assignment solutions to the appropriate GitHub Classroom repository. Submit the following for this assignment:

- A PDF file, typeset in LaTeX, with answers to questions 1(a),(f) and 2(a),(b),(e).
- Python functions called `VanderBuild.py` for question 1(b), `LUPsolve.py` for question 1(d), and `Tridiag_Matvec.py` for question 2(c). Use the templates provided in the assignment repository.
- Python scripts called `VanderAnalysis.py` for question 1(e), and `MatvecAnalysis.py` for question 2(d). Use the templates provided in the assignment repository.
- Three plots, 1 each for questions 1(c), 1(e), and 2(d).

Question 1 — 30 marks

The *Vandermonde matrix* arises in interpolation problems, which we will be covering soon in class. One form of the Vandermonde matrix has elements:

$$V_{ij} = x_{i-1}^{j-1} \quad \text{for } 1 \leq i \leq N+1 \text{ and } 1 \leq j \leq N+1, \tag{1}$$

where $x_i = i/N$, for $i = 0, \ldots, N$, (which are equally spaced points between 0 and 1). For $N = 4$, the matrix looks like:

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & x_0^4 \\ 1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 \\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4 \\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4 \end{bmatrix}$$

with $(x_0, x_1, x_2, x_3, x_4) = (0, 1/4, 1/2, 3/4, 1)$.

It turns out that as N gets large, this matrix becomes very poorly conditioned, and therefore the solutions of linear systems involving it become inaccurate. In this question we explore this further.

(a) Write a psuedo-code for building the matrix V, for a given N and with elements given in equation (1), and compute the computational complexity of your algorithm, i.e. for a given N, determine the number of FLOPS it takes to build the matrix. In terms of "Big-Oh" notation, what is the asymptotic behaviour of your algorithm as N increases? Try to make your algorithm as efficient as possible. **Three bonus marks** if you make your algorithm $O(N^2)$. Submit your answer type-set using LaTeX.

Your pseudocode should have the following form:

Input: integer N

$\vdots$

Insert pseudocode here

$\vdots$

Output: V, an $N+1 \times N+1$ array of floats

(b) Implement your pseudo-code in a function called `VanderBuild.py` which inputs N, and returns the $N+1 \times N+1$ numpy array containing the Vandermonde matrix V.

(c) Write a script that computes the condition number of V, for $N = 3, \ldots, 20$. Plot your results on a semilog plot (condition number on the y axis with a log-scale).

(d) Write a Python function `LUPsolve.py` which inputs an $N+1 \times N+1$ numpy array A and a $N+1 \times 1$ numpy array $\mathbf{b}$, and computes the decomposition $PA = LU$ (i.e. with pivoting), and uses it to solve the linear system

$$A\mathbf{x} = \mathbf{b}$$

The inputs of your function should be A and $\mathbf{b}$, and the output of your function should be $\mathbf{x}$, L, U, and P, in that order.

(e) This part involves solving systems of equations with the matrix V. Define a vector $\mathbf{c}$ equal to the last column of V. Write a Python script called `VanderAnalysis.py` that calls your function `LUPsolve.py` to solve the linear system

$$V\mathbf{x} = \mathbf{c}$$

The exact solution of this system of equations is $\mathbf{x} = \mathbf{e}_{N+1}$ (check, without solving the system explicitly, that this indeed must be the case). Compute the relative error and the maximal relative error (according to the theoretical upper bound discussed in Lecture 7) for all $N = 15, \ldots, 25$, and plot them together on a semilogarithmic scale.

(f) Up to what matrix size does the numerically obtained solution have any meaning? Explain. What is the condition number of V and what is the relative residual for this value of N?

Question 2 **30 marks**

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

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

meaning that $A_{i,j} = 0$ if $j < i-1$ or $j > i+1$. Such banded matrices with band-width 1 are referred to as *tridiagonal*. The numbers a_i $(2 \leq i \leq n)$, b_i $(1 \leq i \leq n)$, and c_i $(1 \leq i \leq n-1)$ are all assumed to be nonzero.

(a) Write a pseudocode for computing the matrix-vector product of the tridiagonal matrix A with a vector $\mathbf{x}$. That is, given arrays $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ that define A as in definition (2), and given a vector $\mathbf{x} \in \mathbb{R}^n$, your algorithm should compute the matrix-vector product $\mathbf{y} = A\mathbf{x}$. Submit your answer type-set using LaTeX. Hint: To get an idea of how the algorithm should work, create a tridiagonal matrix and vector $\mathbf{x}$ for $n = 4$ or 5, and compute the product by hand. Do not include your by-hand computation in your assignment submission.

Your pseudocode should have the following form:

Input: arrays of n floats $\mathbf{x}$, $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$.

$\vdots$

Insert pseudocode here

$\vdots$

Output: array of n floats y such that $\mathbf{y} = A\mathbf{x}$

(b) Analyse the complexity of the algorithm from part (a). That is, determine how many flops are required to compute the product $\mathbf{y} = A\mathbf{x}$ 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 a function called `Tridiag_Matvec.py`.

(d) Write a script called `MatvecAnalysis.py` that generates a tridiagonal test matrix A of the form (2) and a test vector $\mathbf{x}$ for $n = 10^k,\ k = 3, \ldots, 6$, computes the product $A\mathbf{x}$, and measures the time your code takes to complete. Also, try for $n = 10^k,\ k = 3, \ldots, 7$, or 8. You can use the numpy random number generator `numpy.random.rand(n)` to create your A and $\mathbf{x}$. Produce a log-log plot of the time taken versus n (i.e. both axes on a logarithmic scale).

(e) Is the plot what is expected based on your analysis of the complexity of the algorithm? In particular, what is the slope of the graph on the log-log scale? How would the plot look different if you had used the usual matrix-vector product functions (e.g. `matmul`, `@`)? Explain.