Programming Lab #1

Oct 28/29, 2024

Material:

- `basics.ipynb`
- `implementation_efficienty.ipynb`
- `gram_schmidt.ipynb`

Task 1.1 *Numpy and Matplotlib Basics*

(a) Using the method `numpy.random.normal`, generate a vector $x \in \mathbb{R}^{100}$ and a matrix (array) $A \in \mathbb{R}^{150\times 100}$ with standard-normal distributed entries. Visualize the entries of x with the methods `matplotlib.pyplot.plot` (as a graph) and `matplotlib.pyplot.hist` (as a histogram over $n = 30$ bins).

(b) Generate $x \in \mathbb{C}^{100}$ and $A \in \mathbb{C}^{150\times 100}$ such that real and imaginary parts of each entry are standard-normally distributed. Compute the matrix-vector product $y = Ax$.

(c) Generate $A \in \mathbb{C}^{n\times n}$ with standard-normally distributed entries. Compute the eigenvalues with `numpy.linalg.eig` for several instances of A for $n \in \{10, 100, 500\}$ and plot them using the method `matplotlib.pyplot.scatter`. Do you notice anything recurring regarding the eigenvalue distribution?

Whenever not otherwise specified, we will always use vectors and matrices with components that are standard-normal distributed.

Task 1.2 *Efficiency of Implementation*

(a) For $u, v, w \in \mathbb{R}^{10.000}$, compute the product $u \cdot v^T \cdot w$ in both orders of associativity, i.e., $(u \cdot v^T) \cdot w$ and $u \cdot (v^T \cdot w)$. Measure execution time for both computations with the method `time.time` from the `time` library. Validate that both computations yield the same result by computing the relative errors between them.

(b) Let $n \in \mathbb{N}$. For indices $1 \le i < j \le n$ and $\theta \in [0, 2\pi)$, a Givens rotation matrix is defined as

$$G_{i,j}(\theta) = \begin{bmatrix} I_{i-1} & & & & 0 \\ & c & 0 & -s & \\ & 0 & I_{j-i-2} & 0 & \\ & s & 0 & c & \\ 0 & & & & I_{n-j+1} \end{bmatrix},$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. Implement the effective left action $G_{i,j}(\theta) \cdot A$ of a Givens rotation on a matrix $A \in \mathbb{R}^{n\times n}$ and validate the result by explicit matrix multiplication. Measure execution time for $n \in \{100, 1000, 10.000\}$ and $\theta = 0.1$.

Task 1.3 *Gram-Schmidt Orthogonalization*

We explore the differences between the classical and the modified Gram-Schmidt algorithm. Given linearly independent vectors $v_1, \dots, v_k \in \mathbb{C}^n$, both algorithms compute orthonormal vectors $q_1, \dots, q_k \in \mathbb{C}^n$ satisfying

$$\operatorname{span}\{v_1, \dots, v_j\} = \operatorname{span}\{q_1, \dots, q_j\}, \quad j = 1, \dots, k.$$

Classical Gram-Schmidt algorithm:

for $j = 1, \ldots, k$ **do**
  $\widehat{q}_j = v_j$
  for $i = 1, \ldots, j-1$ **do**
    $\widehat{q}_j = \widehat{q}_j - (v_j, q_i)q_i$
  end for
  $q_j = \frac{\widehat{q}_j}{\|\widehat{q}_j\|_2}$
end for

Modified Gram-Schmidt algorithm:

for $j = 1, \ldots, k$ **do**
  $\widehat{q}_j = v_j$
  for $i = 1, \ldots, j-1$ **do**
    $\widehat{q}_j = \widehat{q}_j - (\widehat{q}_j, q_i)q_i$
  end for
  $q_j = \frac{\widehat{q}_j}{\|\widehat{q}_j\|_2}$
end for

The classical and the modified Gram-Schmidt algorithms are mathematically equivalent, i.e., they compute the same orthonormal vectors in *exact* arithmetic.

Implement both algorithms. The quantity $F(A) = \left\| I_n - A^H A \right\|_F$ (you can also take any other norm) measures how far A is from having orthonormal columns. Write a function that computes this quantity and perform the following experiment: For each $j = 1, 2, \ldots, 10$, generate $V \in \mathbb{C}^{1000 \times 100j}$, run both algorithms on V and compare the errors. Discuss the results.
Plot the error as a function of j (or the matrix size) to quickly get an impression of the qualitative behaviour. Then take a look at the precise numbers. A semilog plot shows the order of magnitude (much better than a normal plot). This is particularly useful for plotting errors.