Homework 9

Kevin Sturm
Python 3

Test your code with examples! Below the word function refers to python function. Use numpy arrays!

Problem 1. Write a python script which generates, for odd $n \geq 5$ three different matrices $A \in \mathbf{R}^{n\times n}$. For $n = 5$ the matrices look like

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & & & & 1 \\ 1 & & & & 1 \\ 1 & & & & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad \begin{pmatrix} & & 1 & & \\ & & 1 & & \\ & & 1 & & \\ & & 1 & & \\ & & 1 & & \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ & & & & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & & & & \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

where all entries which are not shown have to be initialized with 0. For $n > 5$ the matrices look accordingly. Avoid loops! Instead, use matrix functions and matrix indexing!

Problem 2. Write a Python-script which determines the maximum $x_{\max} := \max_{1\leq i\leq n} x_i$ of a vector $x \in \mathbf{R}^n$. Further, all entries x_i with $|x_i| \geq x_{\max}$ should be replaced by $\operatorname{sign}(x_i)x_{\max}$. Avoid loops and arithmetics, and use only appropriate vector/matrix functions and indexing instead.

Problem 3. Write a C++ function f, which implements the series

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right), \quad a > 0.$$

Write a pybind11 wrapper called *wrap* and import this C++ function into Python. Compute the value of the sequence x_n by calling the C++ in Python for $n = 1, 2, 3, 4, 5$.

Problem 4. Write a Python-function `ishermitian` which tests if a given matrix $A \in \mathbf{C}^{n\times n}$ is hermitian, i.e., $A = A^H := \overline{A}^T$ resp. $a_{ij} = \overline{a_{ji}}$ for all $0 \leq i, j \leq n$. Avoid loops, und use only appropriate vector/matrix functions and indexing instead.

Problem 5.
Let $L = (\ell_{ij})_{i,j=1,\dots,n} \in \mathbf{R}^{n\times n}$ be a regular (i.e. L has full rank) and lower triangular matrix (i.e. $\ell_{ij} = 0$ for $i < j$). We write L in the block form

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}$$

with $L_{11} \in \mathbf{R}^{p\times p}$, $L_{21} \in \mathbf{R}^{q\times p}$ and $L_{22} \in \mathbf{R}^{q\times q}$,where $p + q = n$.

(a) Show that $\det(L) = \prod_{j=1}^{n} \ell_{jj}$.

(b) Show that

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1}L_{21}L_{11}^{-1} & L_{22}^{-1} \end{pmatrix}.$$

(c) Write a function `invertL`, which L^{-1} recursively calculates the inverse as described.You can test your function with the help of the function `scipy.linalg`. Avoid loops.

Problem 6. One possible algorithm for eigenvalue computations is the *Power Iteration.* It approximates (under certain assumptions) the eigenvalue $\lambda \in \mathbf{R}$ with the greatest absolute value of a symmetric matrix $A \in \mathbf{R}^{n\times n}$ as well as the corresponding eigenvector $x \in \mathbf{R}^n$. The algorithm is obtained as follows: Given a vector $x^{(0)} \in \mathbf{R}^n\backslash\{0\}$, e.g., $x^{(0)} = (1,\dots,1) \in \mathbf{R}^n$, define the sequences

$$x^{(k)} := \frac{Ax^{(k-1)}}{\|Ax^{(k-1)}\|_2} \quad \text{and} \quad \lambda_k := x^{(k)} \cdot Ax^{(k)} := \sum_{j=1}^{n} x_j^{(k)}(Ax^{(k)})_j \quad \text{for } k \in \mathbb{N},$$

where $\|y\|_2 := \left(\sum_{j=1}^{n} y_j^2\right)^{1/2}$ denotes the Euclidean norm. Then, under certain assumptions, (λ_k) converges towards λ, and $(x^{(k)})$ converges towards an eigenvector associated to λ (actually to the eigenspace).

(a) Write a function `poweriteration`, which, given a matrix A, a tolerance τ and an initial vector $x^{(0)}$, verifies whether the matrix A is symmetric. If this is not the case, then the function displays an error message and terminates. Otherwise, it computes (λ_k) and $(x^{(k)})$ until

$$\|Ax^{(k)} - \lambda_k x^{(k)}\|_2 \le \tau \quad \text{and} \quad |\lambda_{k-1} - \lambda_k| \le \begin{cases} \tau & \text{if } |\lambda_k| \le \tau, \\ \tau|\lambda_k| & \text{else,} \end{cases}$$

and returns λ_k and $x^{(k)}$. Realize the function in an efficient way, i.e., avoid unnecessary computations (especially of matrix-vector products) and storage of data.

(b) Compare `poweriteration` with the `numpy.linalg` function `eig`.

Problem 7. Let $U = (u_{ij}) \in \mathbf{C}^{n\times n}$ be an upper triangular and regular matrix, i.e., $u_{jk} = 0$ for $j > k$, such that $u_{jj} \ne 0$ for all $j = 1,\dots,n$.

(a) Show that for every $b \in \mathbf{C}^n$, there exists a unique solution $x \in \mathbf{C}^n$ of $Ux = b$.

(b) Write a function `solveU`, which, given an upper triangular matrix U as above and a vector $b \in \mathbf{C}^n$, computes the unique solution $x \in \mathbf{C}^n$ of $Ux = b$. Use only loops and arithmetics. You must not use the `np.linalg.inv` or `scipy.linalg` module to solve the linear system. However, you can use it to test your implementation.

Problem 8. Write a Python-script which displays for a given vector $x \in \mathbf{C}^N$ the trimmed vector $x' \in \mathbf{C}^{N-2}$, where entry with highest resp. lowest absolute value are cut out of x. The remaining vector should be sorted ascendingly with respect to the absolute value. In case of equality, the elements are sorted ascendingly with respect to the imaginary part. Avoid loops, and use only appropriate array functions and indexing instead.