CDS: Numerical Methods - Assignment week 4

Solutions to the exercise have to be handed in via Brightspace in form of one or several executable Python scripts (*.py) which run without any errors. The deadline for the submission is **Thursday Mar. 5, 13:30**. Feel free to use the Science Gitlab repository to submit your solutions.

1 Eigenvalues and Eigenvectors

In the following you will implement your own eigenvalue / eigenvector calculation routines based on the inverse power method and the iterated QR decomposition.

(a) Inverse Power Method: We start by implementing the inverse power method to calculate the eigenvector corresponding to an eigenvalue which is closest to a given parameter σ. In detail, you should implement a Python function `vec, n = inversePower(A, sigma, eps)` which takes as input the $n \times n$ square matrix A, the parameter σ, as well as some accuracy ε and which returns the eigenvector $\mathbf{v}$ (to the eigenvalue which is closets to σ) and the number of needed iteration steps. To do so, implement the following algorithm.

Start with setting up the needed input:

$$B = (A - \sigma \mathbf{1})^{-1} \tag{1}$$
$$\mathbf{b}^{(0)} = (1, 1, 1, ...) \tag{2}$$

where $\mathbf{b}_0$ is a vector with n entries. Afterwards repeat and increase $k = 1, 2, 3, \dots$ until the error e is smaller than ε:

$$\mathbf{b}^{(k)} = B \cdot \mathbf{b}^{(k-1)} \tag{3}$$
$$\mathbf{b}^{(k)} = \frac{\mathbf{b}^{(k)}}{|\mathbf{b}^{(k)}|} \tag{4}$$
$$e = \sqrt{\sum_{i=0}^{n} \left(|b_i^{(k-1)}| - |b_i^{(k)}|\right)^2} \tag{5}$$

Return the last $\mathbf{b}^{(k)}$ as the eigenvector `vec` and the number of needed iteration k as `n`. Test your routine by calculating all eigenvectors for the matrix

$$A = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 3 \end{pmatrix}.$$

Compare your results to the ones from `numpy.linalg.eig()`.

(b) Next you will need to implement the tri-diagonalization scheme following Householder. To this end implement a Python function `T = tridiagonalize(A)` which takes a symmetric matrix A as input and returns a tridiagonal matrix T of the same dimension. Therefore, your algorithm should execute the following steps:

Let k run $k = 0, 1, 2, \ldots, n-1$ and repeat:

$$q = \sqrt{\sum_{j=k+1}^{n} (A_{j,k})^2} \tag{6}$$

$$\alpha = -\operatorname{sgn}(A_{k+1,k}) \cdot q \tag{7}$$

$$r = \sqrt{\frac{\alpha^2 - A_{k+1,k} \cdot \alpha}{2}} \tag{8}$$

$$\mathbf{v} = \mathbf{0} \qquad \text{... vector of dimension } n \tag{9}$$

$$v_{k+1} = \frac{A_{k+1,k} - \alpha}{2r} \tag{10}$$

$$v_{k+j} = \frac{A_{k+j,k}}{2r} \quad \text{for } j = 2, 3, \ldots, n \tag{11}$$

$$P = \mathbf{1} - 2\mathbf{v}\mathbf{v}^T \tag{12}$$

$$A = P \cdot A \cdot P \tag{13}$$

At the end return A as T. Hint: Use `np.outer()` to calculate the *matrix* $\mathbf{v}\mathbf{v}^T$ as needed in the definition of the Housholder transformation matrix P. Apply your routine to the matrix A defined above as well as to a few random, but symmetric matrices of different dimension n.

(c) Implement the QR decomposition based diagonalization routine for tri-diagonal matrices T in Python as a function `d = QREig(T, eps)`, which takes a tri-diagonal matrix T and some accuracy ε as input and returns all eigenvalues as a vector $\mathbf{d}$. By making use of the QR decomposition as implemented in nummpy (`numpy.linalg.qr()`) the algorithm is very simple and reads:

Repeat until the error e is smaller than ε:

$$T = Q \cdot R \qquad \text{... do this decomposition with the help of Numpy!} \tag{14}$$

$$T = R \cdot Q \tag{15}$$

$$e = |\mathbf{d_1}| \tag{16}$$

where $\mathbf{d_1}$ is the first sub-diagonal of T at each iteration step. Afterwards return the main-diagonal of A as $\mathbf{d}$. Test your routine for the case of the matrix A defined above. To this end you need to tri-diagonalize it first.

(d) With the help of `d = QREig(T, eps)` you can now calculate all eigenvalues and with the help of `vec, n = inversePower(A, sigma, eps)` you can calculate all corresponding eigenvectors by setting σ to approximately the eigenvalues saved in $\mathbf{d}$ (you should add some small random noise to σ in order to avoid singularity issues in the inversion needed for the inverse power method). Apply this combination to calculate all eigenvalues and eigenvectors of A defined above.

(e) Optional: Test your eigenvalue / eigenvector algorithm for other, larger random (but symmetric) matrices.