General instructions

- You have 2 hours to complete the examination. When applicable, people with special facilities have 2h20 minutes in total.
- The exam is "closed book", meaning that you can only make use of the material given to you.
- You are supposed to write the codes in the Python programming language, but no points are subtracted for syntax errors if the code is correct otherwise.
- **Do not use while statements.**
- **Do not use global variables.**
- **Elementary operations:** any mathematical operation on a scalar value such as multiplications, additions, comparisons (like max), absolute value, square root, power. Do not count accessing array elements in complexity calculations, or the definition of variables. You can neglect the operations that do not depend on the input size.
- Note that you could also try to compute the complexity of the algorithms without the coding part, in case you get stuck in the code. You need in any case to write down step by step the complexity calculations. The coefficient multiplying the "n-term" is also expected in your answer.
- In order to get the full points per sub-question, all implementations have to be performed at the lowest possible computational complexity. For instance, if the result of an operation is needed in several parts of an algorithm, that operation has to be performed only once (and the result re-used) to get the full points. If the same result is computed more than once, only half of the points will be given in those code lines.
- **From NumPy, you are allowed to use only these methods:** `shape`, `zeros`, `dot`, `sqrt`, `matmult`, `@`.
- **Do not use SciPy.**
- The grade is computed with the formula: points/totalpoints $\times\, 9 + 1$.
- Keep the names of the variables as stated in the question, if applicable.
- Choose descriptive and concise names for your own functions.
- **If you do not follow these instructions you will not receive any points in the respective question.**

Identities

$$\sum_{i=1}^{n} i = n\frac{n+1}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}, \quad \sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a} \quad (a \neq 1)$$

Exercise 1 (3 points)

Consider the linearly independent vectors $\vec{v}_1, \ldots, \vec{v}_n \in \mathbb{R}^n$. An orthogonal set of vectors $\vec{u}_1, \ldots \vec{u}_n$ can be constructed by the algorithm:

$$\begin{aligned}
\vec{u}_1 &= \vec{v}_1 \\
\vec{u}_2 &= \vec{v}_2 - \frac{\langle \vec{v}_2, \vec{u}_1 \rangle}{\langle \vec{u}_1, \vec{u}_1 \rangle} \vec{u}_1 \\
\vec{u}_3 &= \vec{v}_3 - \frac{\langle \vec{v}_3, \vec{u}_2 \rangle}{\langle \vec{u}_2, \vec{u}_2 \rangle} \vec{u}_2 - \frac{\langle \vec{v}_3, \vec{u}_1 \rangle}{\langle \vec{u}_1, \vec{u}_1 \rangle} \vec{u}_1 \\
\vec{u}_4 &= \vec{v}_4 - \ldots
\end{aligned}$$

with $\langle \cdot, \cdot \rangle$ the standard $\mathbb{R}^n$ scalar product. The general formula for $\vec{u}_k$, with $k = 1, \ldots, n$, in terms of $\vec{v}_1, \ldots, \vec{v}_k$ has the form:

$$\begin{aligned} &\text{for } k = 1: \qquad \vec{u}_1 = \vec{v}_1, \\ &\text{for } k > 1: \qquad \vec{u}_k = \vec{v}_k - \sum_{j=1}^{k-1} \frac{\langle \vec{v}_k, \vec{u}_j \rangle}{\langle \vec{u}_j, \vec{u}_j \rangle} \vec{u}_j. \end{aligned} \tag{1}$$

(a) **(2 pts)** Write a Python function that implements the algorithm (1). It should receive the vectors to orthogonalize $\vec{v}_1, \ldots, \vec{v}_n$ as a list of 1D NumPy arrays (each NumPy array corresponding to a vector), and return the orthogonalized set $\vec{u}_1, \ldots, \vec{u}_n$ as a list with the 1D NumPy arrays.

(b) **(1 pts)** Give the complexity of the algorithm in terms of n. Specify the constants in front of each power of n. Note: the complexity of `numpy.dot(a, b)` corresponds to that of the standard scalar product.

Solution:

For each step k, with $k = 1, \ldots, n$

- **(0.25 pts)** Dot products: $2n(k-1)$
- **(0.25 pts)** One scalar-vector multiplication: $n(k-1)$
- **(0.25 pts)** Subtraction of k vectors of dimension n: $n(k-1)$

Summing complexities for $k = 1, \ldots, n$ **(0.25 pts)**:

$$4n \sum_{k=1}^{n} (k-1) = 4n \sum_{k=1}^{n} k - 4n^2 = 4n \frac{n(n+1)}{2} - 4n^2 = 2n^3 - 2n^2$$

Exercise 2 (6 points)

Consider a matrix $A \in \mathbb{R}^{n \times n}$, vector $\vec{b} \in \mathbb{R}^n$ and the following iterative procedure, with given starting vector $\vec{x}^{(0)} \in \mathbb{R}^n$:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \alpha_k \left(\vec{b} - A\vec{x}^{(k)} \right), \quad \alpha^{(k)} = \frac{\left(A \left(\vec{b} - A\vec{x}^{(k)} \right) \right)^\intercal \left(\vec{b} - A\vec{x}^{(k)} \right)}{\left\| A \left(\vec{b} - A\vec{x}^{(k)} \right) \right\|^2} \tag{2}$$

(c) **(0.5 pts)** Write a function called `residual` that receives a matrix A and vectors $\vec{b}$ and $\vec{c}$ and returns $\vec{b} - A\vec{c}$.

(d) **(0.5 pts)** Write a function called `alpha` that receives a matrix A and the output of `residual` and returns $\alpha^{(k)}$. The function must make use of the function written in question (c).

(e) **(2 pts)** Write a function called `iterations` that implements the iterative procedure (2). The function receives: a matrix A, vectors $\vec{b}$ and $\vec{x}^{(0)}$, values for a relative tolerance $\epsilon > 0$ and a maximum number of iterations $N \in \mathbb{N}$. The stopping criterium of the iteration is the condition $\|\vec{b} - A\vec{x}^{(k)}\| < \epsilon \|\vec{b}\|$ or reaching the maximum number of iterations. The function returns the last computed $\vec{x}^{(k)}$ and the number of iterations. Assume given the previously defined functions from questions (c) and (d).

(f) **(1 pt)** Compute the total number of operations involved in the previous question (e) in terms of the vector dimension n. Assume that the maximum number of N iterations was reached. Also consider the operations carried out by the functions implemented in questions (c) and (d). Specify

the constants multiplying each power of n. Note: the complexity of `numpy.dot(a, b)` corresponds to that of the standard scalar product.

Solution:

- **(0.2 pts)** Computation the norm of $\vec{b}$: $n + n = 2n$ ops (only once, outside the loop)
- **(0.2 pts)** Computation of the residual $\vec{b} - A\vec{c}$: $2n^2 + n$ ops (multiplications and additions/subtractions).
- **(0.1 pts)** Computation of the relative error: $2n$ ops (dot product).
- **(0.2 pts)** Computing α: $2n \cdot n + 2n + 2n = 2n^2 + 4n$ ops ($2n^2$ for $A\vec{r}$, $2n$ for each dot product).
- **(0.2 pts)** Update solution: $2n$ ops (n multiplications and n additions)
- **(0.1 pts)** Adding all operations, for each iteration: $N(4n^2 + 9n) + 2n$

(g) **(1.5 pts)** Assume now that A is a tridiagonal matrix. Explain which adaptions to the answers to the previous sub-questions (c), (d), (e) should be performed to minimize the computational complexity for that particular case. Calculate the resulting complexity. Let all your adapted functions still receive the same input, in particular the matrix A. Note that you may need to create (an) additional function(s).

(h) **(0.5 pts)** Please discuss if it is possible to speed up the algorithm by considering that the matrix A is not just tridiagonal but also symmetric. Justify by indicating which specific operations can be omitted in the algorithms/code written in question (g).

Solution: The only place where symmetry could be used is in the matrix vector multiplication steps. However, it is not possible to save operations by using the symmetry. Answers concerning memory saving (since upper and lower diagonals are the same) do not give points, since that is not what is was asked.