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 syntactic errors are allowed as far as the written algorithm can be well understood.
- **Do not use while statements.**
- **Do not use global variables in functions.**
- The grade is computed with the formula: points/16 $\times$ 9 + 1.
- Keep the names of the variables and functions as stated in the question.
- **If you do not follow these instructions you will not receive any points in the respective question.**

Identities

$$\sum_{i=1}^{n} i = n\frac{n+1}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

Questions

Any square matrix $A \in \mathbb{R}^{n\times n}$ can be factorized by means of LUP factorization (LU with pivoting). In the special case where A is symmetric and positive-definite[1], it admits the so-called Cholesky factorization,

$$A = LL^\top,$$

where L is a lower triangular matrix and $L^\top$ its transpose.

The non-zero entries $\ell_{i,j}$ of L can be computed with, for $j = 1, \ldots, n$:

$$\begin{aligned} \ell_{j,j} &= \sqrt{a_{j,j} - \sum_{k=1}^{j-1} \ell_{j,k}^2}, \\ \ell_{i,j} &= \frac{1}{\ell_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k}\ell_{j,k} \right) \quad \text{for } i = j+1, \ldots, n. \end{aligned} \tag{1}$$

($a_{i,j}$ refers to the entry of A in row i and column j.)

1. (1 point) Check the requirement that A must be symmetric. Write a python function with the name `is_symmetric` that receives as an argument the square matrix A in the form of a 2D `numpy` array and returns `True` if A is symmetric and `False` otherwise.

Solution: Double for loop:

[1] A positive-definite matrix has the property $x^\top Ax > 0 \;\; \forall x \in \mathbb{R}^n$. This is not relevant here.

```python
# -0.5 points for wrong function arguments/return (this is given)
def is_symmetric(A):
    n = A.shape[0]
    for i in range(0, n - 1):           # 0.5 points for double for loop,
        for j in range(i + 1, n):       #       0 if while is used
            if not A[i, j] == A[j, i]:  # 0.5 points
                return False
    return True
```

Simplest solution:

```python
def is_symmetric(A):
    return np.all(A == A.T)     # 1 point
```

2. (5 points) Write a python function `cholesky(A)` that receives a dense matrix A in the form of a 2D `numpy` array and computes and returns its Cholesky decomposition L (also a 2D `numpy` array). You need to implement the algorithm yourself, do not use `numpy`'s or `scipy`'s (or others) implementation of the Cholesky factorization. Other `numpy` functions are allowed.

   Hint: be careful when translating the limits of the sums to python loops!

Solution: Column by column (row by row also possible, adapt the rewarded points flexibly to whatever implementation is chosen (e.g., using slicing)):

```python
# -1 point for incorrect function definition (args, returns) since this is given
def chol(A):
    """Naive implementation without slicing."""
    L = np.zeros_like(A)                # 0.5 initialize L
    n = L.shape[0]

    for j in range(n):                  # 0.5 points for outer loop

        # computation of diagonal L[j, j]:       (1.5 points -->)
        s = 0
        for k in range(j):              # 0.5 for loop w/ correct limit
            s += L[j, k] ** 2           # 0.5 sum of squares of correct entries
        h = A[j, j] - s

        # solution for question 3):     1 point
        #                               subtract 0.5 if wrong condition,
        #                               wrong location, or not exiting function
        #                               with a message as requested (min 0 pts)
        if h <= 0:
            raise Exception("A is not positive definite!")

        L[j, j] = np.sqrt(h)            # 0.5 points together with def. of h

        # computation of L[i, j]:     (2.5 points -->)
        for i in range(j, n):           # 0.5 for loop /w correct limit
            s = 0
            for k in range(j):          # 0.5 for loop /w correct limit
                s += L[i, k] * L[j, k]                  # 0.5 correct sum
            L[i, j] =  1 / L[j, j] * (A[i, j] - s)      # 1 correct formula
    return L
```

3. (1 point) Checking if a matrix is positive-definite is more complicated than testing symmetry. In practice, the Cholesky factorization can be used to check if a matrix is positive-definite: from the existence of the Cholesky factorization, which is given if the term under the square root in (1) is strictly positive, follows that A is positive-definite.

   Add a check to your function from question 2.) that detects if A is not positive-definite. In this case, print a message and exit the function/abort the factorization procedure. Write only the additional or modified lines of code and indicate where they should be added to the function.

> **Solution:** See solution of 2.)

4. (5 points) Now consider the case where A is a symmetric positive-definite tridiagonal matrix

$$A = \begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & b_{n-1} & a_n \end{pmatrix}$$

   and stored in arrays of the diagonal entries $\mathbf{a} := (a_1, \ldots, a_n)$ and the off-diagonal entries $\mathbf{b} := (b_1, \ldots, b_{n-1})$.

   Write a new function `cholesky_tridiag(a, b)` that implements the Cholesky factorization adapted specifically for the sparse structure of A. L is now a bidiagonal matrix with non-zero entries only in the main diagonal and in the lower diagonal. The function returns L in terms of two arrays, for instance $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, containing the main diagonal entries and the off-diagonal entries, respectively.

   Note: the matrices A and L must be expressed solely by the arrays containing the main diagonal and off-diagonal entries; do not construct dense matrices!

   Hint: the algorithm for tridiagonal matrices can be derived from Algorithm (1) by simply adapting the sums to the sparsity pattern. Alternatively, an algorithm for α_i, β_j can be deduced from the sparse product $LL^\top$:

$$A = \begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & b_{n-1} & a_n \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha_1 & & & \\ \beta_1 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_n \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \beta_2 & \\ & & \ddots & \ddots \\ & & & \alpha_n & \beta_{n-1} \end{pmatrix}}_{L^\top}$$

$$= \begin{pmatrix} \alpha_1^2 & \alpha_1\beta_1 & & & \\ \alpha_1\beta_1 & \beta_1^2 + \alpha_2^2 & \alpha_2\beta_2 & & \\ & \alpha_2\beta_2 & \beta_2^2 + \alpha_3^2 & \ddots & \\ & & \ddots & \ddots & \alpha_{n-1}\beta_{n-1} \\ & & & \alpha_{n-1}\beta_{n-1} & \beta_{n-1}^2 + \alpha_n^2 \end{pmatrix}$$

> **Solution:** If a dense matrix is constructed despite the warnings: 0 points.
>
> Example implementation:

```
# -1 point for incorrect function definition (args, returns) since this is given
def chol_tridiag(a, b):
    n = a.size  # len(a)
    alpha = np.zeros(n)          # 0.5 initialization with correct size
    beta = np.zeros(n - 1)
```

```
    alpha[0] = np.sqrt(a[0])    # 0.5 first element

    # computation of alpha, beta (4 points ->)
    for i in range(1, n):       # 1 for correct loop (only 1 loop necessary)
        beta[i - 1] = b[i - 1] / alpha[i - 1]          # 1.5 for beta
        alpha[i] = np.sqrt(a[i] - beta[i - 1] ** 2)    # 1.5 for alpha
    return alpha, beta
```

5. Analyze the time complexity of both Cholesky factorization algorithms.

   Consider as elementary operations the square root, addition, subtraction, multiplication, division with equal fixed cost. The time complexity is given in Big-O notation (including the constant factor of the leading order term).

   (a) (3 points) Determine the number of operations and the time complexity of Algorithm (1) for dense matrices. Compare your findings to the complexity of LU factorization ($O(\frac{2}{3}n^3)$). Which algorithm is more time efficient?

   (b) (1 point) Determine the number of operations and the time complexity of your derived Cholesky factorization for sparse tridiagonal matrices from question 4.).

Solution:

(a)
- (1.5 points): At each iteration $j = 1, \dots, n$ we have for $\ell_{j,j}$ in Algo (1):
  - $j-1$ multiplications
  - $j-1$ additions/subtractions
  - 1 square root

  and for $\ell_{i,j}$, for $i = j+1, \dots, n$:
  - j multiplications (or $j-1$ dividing the term in parenthesis directly by $\ell_{j,j}$)
  - $j-1$ additions/subtractions
  - 1 division

  Thus, the number of operations is:

$$f(n) = \sum_{j=1}^{n}(2j-1) + \sum_{j=1}^{n}\sum_{i=j+1}^{n} 2j = n^2 + \sum_{j=1}^{n}(n-j)2j = n^2 + \sum_{j=1}^{n}\left(2nj - 2j^2\right)$$
$$= n^2 + n^2(n+1) - 2\left(\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}\right) = \frac{1}{3}n^3 + n^2 - \frac{1}{3}n.$$

- (1 points): The time complexity is:

$$f(n) = O(\frac{1}{3}n^3).$$

- (0.5 points): Cholesky, if admissible, is more time efficient than LU, since the complexity is half of LU.

(b)
- (0.5 points): Number of operations $f(n) = 4n + 1$
- (0.5 points): Time complexity: $f(n) = O(4n)$