Exercise Sheet № 4 Linear systems of equations

Task 4.1: Recall to last Exercise

Given $n \in \mathbb{N}$, we define $A_n \in \mathbb{R}^{n\times n}$ via

$$A_n = (n+1)^2(2I_n + T_n + T_n^T)$$

where

$$T_n = \begin{bmatrix} \mathbf{0}_{n-1} & -I_{n-1} \\ 0 & \mathbf{0}_{n-1}^T \end{bmatrix}$$

i) Show that the eigenvalues of A_n are given by:

$$\lambda_{n,k} = 2(n+1)^2\left(1 - \cos\left(\frac{k\pi}{n+1}\right)\right) \qquad k = 1,\ldots,n$$

with the corresponding eigenvectors

$$\boldsymbol{v}_{n,k} = \sum_{l=1}^{n} \sin\left(\frac{lk\pi}{n+1}\right)\boldsymbol{e}_l$$

ii) Prove $A_n > 0$

iii) Using NumPy's `linalg.eig` function, calculate the maximum and minimum eigenvalues[1] of A_{10}, A_{100} and A_{100}

Subtask i): Given that $A_n = (n+1)^2 D_n$, we know $\operatorname{spec} A_n = (n+1)^2 \operatorname{spec} D_n$. Thus, we want to find $\operatorname{spec} D_n$. Developing the determinant along the first column yields the following recurrence relation, where $\Delta_n = \det(D_n - \lambda I_n)$:

$$\Delta_n = (2-\lambda)\Delta_{n-1} - \Delta_{n-2}$$

where Δ_{n-2} is the second minor of the first column, and $\Delta_0 = 1$ and $\Delta_1 = 2$. Let $a = 2-\lambda$. Solving the recurrence relation yields:

$$\begin{bmatrix} \Delta_n \\ \Delta_{n-1} \end{bmatrix} = \begin{bmatrix} a\Delta_{n-1} - \Delta_{n-2} \\ \Delta_{n-1} \end{bmatrix} = \begin{bmatrix} a & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} \Delta_{n-1} \\ \Delta_{n-2} \end{bmatrix}$$
$$\Rightarrow \begin{bmatrix} \Delta_n \\ \Delta_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} a & -1 \\ 1 & 0 \end{bmatrix}^{n-1}}_{S^{n-1}}\begin{bmatrix} \Delta_1 \\ \Delta_0 \end{bmatrix}$$

Computing the eigenvalues and -vectors of the system matrix S allows us to diagonalize S, given it's eigenvalues all have algebraic multiplicity 1, and compute S^n easily:

$$\chi_S(\sigma) = \sigma(\sigma - a) + 1 = \sigma^2 - a\sigma + 1$$
$$\Rightarrow \sigma_{1,2} = \frac{a}{2} \pm \sqrt{\frac{a^2-4}{4}} = \frac{a \pm \sqrt{a^2-4}}{2}$$

The corresponding eigenvectors are

$$\boldsymbol{v}_1 = \begin{bmatrix} \sigma_1 \\ 1 \end{bmatrix}, \boldsymbol{v}_2 = \begin{bmatrix} \sigma_2 \\ 1 \end{bmatrix} \Rightarrow V = \begin{bmatrix} \sigma_1 & \sigma_2 \\ 1 & 1 \end{bmatrix}$$

Now we can solve the recurrence relation for Δ_n:

$$\begin{bmatrix} \Delta_n \\ \Delta_{n-1} \end{bmatrix} = V\operatorname{diag}(\sigma_1^{n-1}, \sigma_2^{n-1})V^{-1}\begin{bmatrix} \Delta_1 \\ \Delta_0 \end{bmatrix}$$

[1] and hence the condition numbers

Note the following:

$$a - \sigma_2 = \frac{a}{2} + \frac{\sqrt{a^2-4}}{2} = \sigma_1$$
$$a - \sigma_1 = \frac{a}{2} - \frac{\sqrt{a^2-4}}{2} = \sigma_2$$

Thus:

$$\begin{aligned}
\Delta_n &= \frac{a(\sigma_1^n - \sigma_2^n) + \sigma_1\sigma_2^n - \sigma_1^n\sigma_2}{\sigma_1 - \sigma_2} = \frac{\sigma_1^n(a-\sigma_2) - \sigma_2^n(a-\sigma_1)}{\sigma_1-\sigma_2} = \frac{\sigma_1^{n+1} - \sigma_2^{n+1}}{\sigma_1 - \sigma_2} \\
&= \sigma_2^n \left(\frac{\frac{\sigma_1^{n+1}}{\sigma_2^{n+1}} - 1}{\frac{\sigma_1}{\sigma_2} - 1} \right) \\
\Delta_n &\overset{!}{=} 0 \Rightarrow \frac{\sigma_1^{n+1}}{\sigma_2^{n+1}} = 1
\end{aligned}$$

Thereby $\frac{\sigma_1}{\sigma_2}$ must be a $(n+1)$-st root of unity, i.e. $\exists k \in \{1, \ldots, n\}$ such that:

$$\frac{\sigma_1}{\sigma_2} = \mathrm{e}^{\frac{2\pi i k}{n+1}} \Rightarrow \sigma_1 = \sigma_2 \mathrm{e}^{\frac{2\pi i k}{n+1}}$$

Let $\zeta_k = \mathrm{e}^{\frac{2\pi i k}{n+1}}$, then we get:

$$\begin{aligned}
&a + \sqrt{a^2-4} = \zeta_k(a - \sqrt{a^2-4}) \Leftrightarrow a(\zeta_k - 1) = \sqrt{a^2-4}(\zeta_k+1) \\
&\Rightarrow a^2(\zeta_k-1)^2 = (a^2-4)(\zeta_k+1)^2 \Leftrightarrow a^2((\zeta_k+1)^2 - (\zeta_k-1)^2) = 4(\zeta_k+1)^2 \\
&\Leftrightarrow a^2(\zeta_k^2 + 2\zeta_k + 1 - \zeta_k^2 + 2\zeta_k - 1) = 4(\zeta_k+1)^2 \\
&\Leftrightarrow 4\zeta_k a^2 = 4(\zeta_k+1)^2 \\
&\Rightarrow a^2 = \frac{(\zeta_k+1)^2}{\zeta_k} \Rightarrow a = \frac{\zeta_k+1}{\sqrt{\zeta_k}} = \frac{\mathrm{e}^{\frac{2\pi i k}{n+1}} + 1}{\mathrm{e}^{\frac{\pi i k}{n+1}}} \cdot \frac{\mathrm{e}^{-\frac{\pi i k}{n+1}}}{\mathrm{e}^{-\frac{\pi i k}{n+1}}} = \mathrm{e}^{\frac{\pi i k}{n+1}} + \mathrm{e}^{-\frac{\pi i k}{n+1}} = 2\cos\left(\frac{\pi k}{n+1}\right) \\
&a = 2 - \lambda \Rightarrow \lambda = 2\left(1 - \cos\left(\frac{\pi k}{n+1}\right)\right)
\end{aligned}$$

Let :

$$\mathrm{T}_n = \begin{bmatrix} \mathbf{0} & -\mathrm{I}_{n-1} \\ 0 & \mathbf{0}^T \end{bmatrix} \qquad \mathrm{B}_n = \mathrm{T}_n^T$$
$$s_{k,l} = \sin\left(\frac{kl\pi}{n+1}\right) \qquad \theta_k = \frac{k\pi}{n+1}$$

Then $\mathrm{D}_n = 2\mathrm{I}_n + \mathrm{T}_n + \mathrm{B}_n$:

$$\begin{aligned}
\mathrm{D}_n \boldsymbol{v}_k &= 2\sum_{l=1}^{n} s_{k,l}\boldsymbol{e}_l - \sum_{l=2}^{n} s_{k,l}\boldsymbol{e}_{l-1} - \sum_{l=1}^{n-1} s_{k,l}\boldsymbol{e}_{l+1} \\
&= \boldsymbol{e}_1(2s_{k,1} - s_{k,2}) + \boldsymbol{e}_n(2s_{k,n} - s_{k,n-1}) + \sum_{l=2}^{n-1}(2s_{k,l} - s_{k,l-1} - s_{k,l+1})\boldsymbol{e}_l
\end{aligned}$$

We first analyze the summands:

$$\begin{aligned}
&s_{k,l-1} + s_{k,l+1} = \sin((l-1)\theta_k) + \sin((l+1)\theta_k) = \sin(l\theta_k - \theta_k) + \sin(l\theta_k + \theta_k) \\
&= \sin(l\theta_k)\cos(\theta_k) - \cos(l\theta_k)\sin(\theta_k) + \sin(l\theta_k)\cos(\theta_k) + \cos(l\theta_k)\sin(\theta_k) = 2\sin(l\theta_k)\cos(\theta_k)
\end{aligned}$$

Therefore $2s_{k,l} - s_{k,l-1} - s_{k,l+1} = 2\sin(l\theta_k)(1 - \cos(\theta_k)) = \lambda_k s_{k,l}$. Only the first and last entry of $\mathrm{D}_n\boldsymbol{v}_k$ remain:

$$2s_{k,1} - s_{k,2} = s_{k,1}\lambda_k \Leftrightarrow 2s_{k,1} - s_{k,2} = 2s_{k,1} - 2s_{k,1}\cos(\theta_k)$$

$$\begin{aligned}
&\Leftrightarrow s_{k,2} = 2s_{k,1}\cos(\theta_k)\\
&s_{k,2} = \sin(2\theta_k) = 2\sin(\theta_k)\cos(\theta_k) = 2s_{k,1}\cos(\theta_k)\\
\\
&2s_{k,n} - s_{k,n-1} = s_{k,n}\lambda_k = 2s_{k,n} - 2s_{k,n}\cos(\theta_k)\\
&\Leftrightarrow \sin((n-1)\theta_k) = 2\sin(n\theta_k)\cos(\theta_k)\\
&2\sin(n\theta_k)\cos(\theta_k) = \sin(n\theta_k + \theta_k) + \sin(n\theta_k - \theta_k)\\
&\sin(n\theta_k + \theta_k) = \sin\left(\frac{(n+1)k\pi}{n+1}\right) = \sin(k\pi) = 0\\
&\Rightarrow \sin(n\theta_k)\cos(\theta_k) = \sin((n-1)\theta_k)
\end{aligned}$$

It follows:

$$\mathrm{D}_n \boldsymbol{v}_k = \lambda_k \boldsymbol{v}_k$$

Subtask ii): We immediately see, that A_n is symmetric. Therefore, if $\lambda_m = \min \mathrm{spec}\mathrm{A}_n > 0$, we get that $\mathrm{A}_n > 0$. Since $|\cos| \leq 1$, the smallest possible value of λ_m is 0. If we can show that λ_m is never zero, then A_n is positive definite. Note that $\cos(\theta_k) = 1$ for $\theta_k = 2l\pi$ for $l \in \mathbb{N}_0$:

$$\frac{k\pi}{n+1} = 2l\pi \Leftrightarrow k = 2(n+1)l$$

Since $k \in \{1, \dots, n\}$, the value $l = 0$ is invalid, and so is $l = 1$, since this produces $k = 2n + 2 > n$. Therefore $\cos(\theta_k) \neq 1 \forall k = 1, \dots, n$, which in return means, that $\lambda_m > 0$, hence $\mathrm{A}_n > 0$.

Subtask iii): See the submitted Jupyter-Notebook `ex_4_1.ipynb` for the implementation. The results are:

n	$\min \mathrm{spec}\mathrm{A}_n$	$\max \mathrm{spec}\mathrm{A}_n$	$\kappa_2(\mathrm{A}_n)$
10	9.8027	$4.742 \cdot 10^2$	48.36
100	9.8688	$4.078 \cdot 10^4$	4133.63
1000	9.8696	$4.008 \cdot 10^6$	406095.03

Table 1: Minimum and Maximum eigenvalues, as well as the condition numbers of A_n for $n = 10, 100, 1000$

Vandermonde-Matrix. Let $\boldsymbol{v} \in \mathbb{R}^n$. The $n \times n$ Vandermonde matrix V generated by $\boldsymbol{v}$ is defined by:

$$\mathrm{V}(\boldsymbol{v}) = \sum_{k=1}^{n} \langle \boldsymbol{v}, \boldsymbol{e}_k \rangle^{k-1} \mathrm{R}_k \qquad \mathrm{R} = \left[\delta_{ik}\right]_{ij}$$

Task 4.2: Gaussian Elimination

i) Write a python script containing the function `GaussElim(A,b)` that returns the solution vector of the linear equation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ via Gaussian Elimination with pivoting, where $\mathrm{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$. Test your script by computing the solution of $\mathrm{V}(\boldsymbol{v})\boldsymbol{x} = \boldsymbol{b}$, where $\mathrm{V}(\boldsymbol{v})$ is the 6×6 Vandermonde matrix generated by from the vector

$$\boldsymbol{v} = \begin{bmatrix} 1 & 1.2 & 1.4 & 1.6 & 2.8 & 2 \end{bmatrix}^T$$
$$\boldsymbol{b} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

The Vandermonde matrix tends to be ill-conditioned. Discuss the accuracy of the numerical solution $\boldsymbol{x}$ by computing $\mathrm{A}\boldsymbol{x} - \boldsymbol{b}$, which should be equal to $\boldsymbol{0}$.

ii) From Task 4.1: Let $\boldsymbol{b}_n = \pi^2 \boldsymbol{v}_1$. For $n \in \{10, 100, 1000\}$, approximate the solution of the linear system $\mathrm{A}_n \boldsymbol{x}_n = \boldsymbol{b}_n$ by using your function `GaussElim` and provide a visualization of the solution. Calculate the residual norm $\|\boldsymbol{b}_n - \mathrm{A}_n \widetilde{\boldsymbol{x}_n}\|_2$ and the *error* norm $e_n = (n+1)^{-1}\|\widetilde{\boldsymbol{x}}_n - \boldsymbol{v}_1\|_2$, where $\widetilde{\boldsymbol{x}}_n$ is the obtained numerical solution.

Subtask i): The python file `gauss.py` contains the implementation of `GaussElim(A,b)`.

Algorithm 1: Gaussian Elimination

```
name: GaussElim
input: n×n matrix A, n×1 vector b
output: n×1 vector x̃

GaussElim(A, b):
    pivot_indices = [n+1]_{i=1}^n
    Ab = [A  b]

    for i = 1,...,n do
        p = 0
        for k = 1,...,n do
            if k ∈ pivot_indices then
                continue
            else
                if Ab[k,i] ≠ 0 then
                    p = A[k,i]
                    pivot_indices[i] = i
                    if i ≠ k then
                        swap(Ab,k,i)
                    end
                    break
                end
            end
        end
    end

    for j = 1,...,n do
        if j == i then
            continue
        else
            row(Ab,j) = row(Ab,j) − Ab[j,i]·row(Ab,i)
        end
    end
    return col(Ab, n+1)
```

Computing $\widetilde{\boldsymbol{x}}$ and $\boldsymbol{e} = \mathrm{A}\widetilde{\boldsymbol{x}} - \boldsymbol{b}$ numerically, we see that $\|\boldsymbol{e}\| \approx 4 \cdot 10^{-11}$, see the Notebook for „exakt“ numerical data, which is not exactly 0, but very close. This stems from the fact, that Gaussian Elimination norms the pivot-elements, which can lead to multiplication of very small numbers with very big numbers, therefore it's prone to numeric errors. Given the Vandermonde-matrix is ill-conditioned, the not-exact solution is not surprising.

Subtask ii)

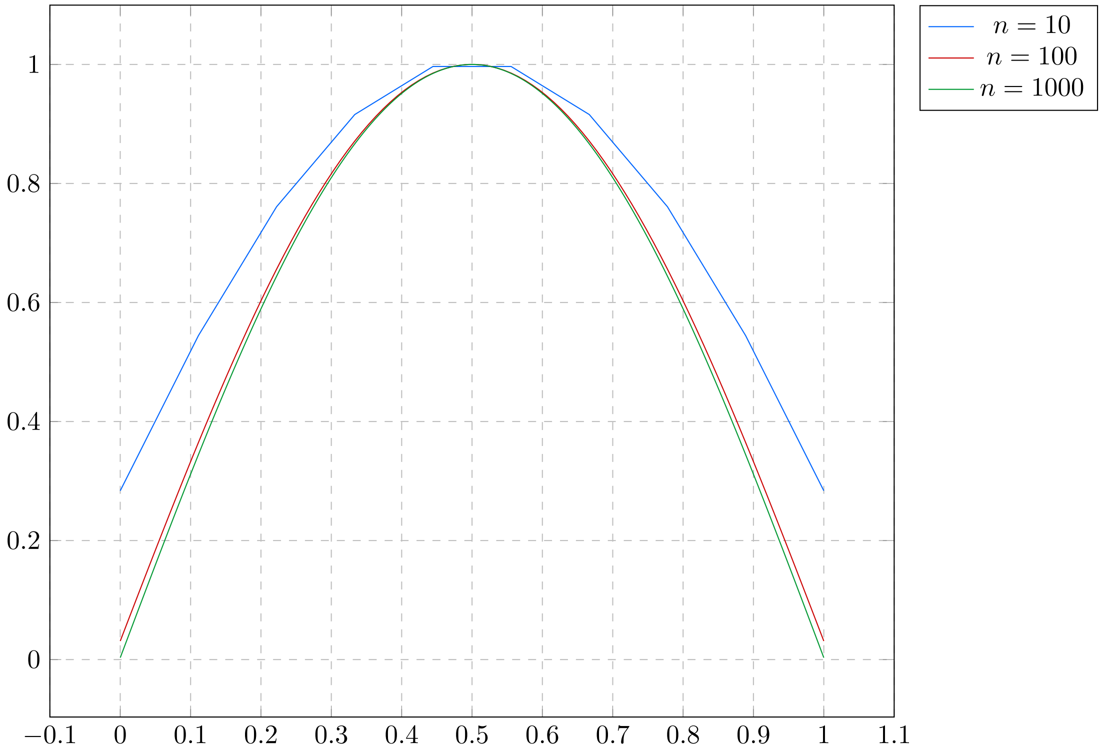

Figure 1: Visualization of the solution vectors for $n = 10, 100, 1000$

Task 4.3: LU-Decomposition

i) Write a python-script containing the following functions, for a given $n \times n$ matrix A and a vector $\boldsymbol{b} \in \mathbb{R}^n$:
   a) `LUP(A)` implements the LU-decomposition with partial pivoting and returns the matrices L, a lower triangular matrix, U, an upper triangular matrix, and P, a permutation matrix, if they exist. Raise a warning otherwise.
   b) `LUPSolver(A,b)` which solves the matrix equation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ via LU-decomposition and returns the solution vector $\boldsymbol{x}$.

ii) From Task 4.1: Let $\boldsymbol{b}_n = \pi^2 \boldsymbol{v}_1$. For each $n \in \{10, 100, 1000\}$, approximate the solution of the linear system $\mathrm{A}_n \boldsymbol{x}_n = \boldsymbol{b}_n$, using your function `LUPSolver` and provide a visualization of the solution. Calculate the residual norm $\|\boldsymbol{b}_n - \mathrm{A}_n \widetilde{\boldsymbol{x}}_n\|_2$ and the *error* norm $e_n = (n+1)^{-1}\|\widetilde{\boldsymbol{x}}_n - \boldsymbol{v}_n\|_2$, where $\widetilde{\boldsymbol{x}}_n$ ist the obtained approximate solution.

iii) Prove or disprove the following:
   a) If all the principal minors of a matrix $\mathrm{A} \in \mathbb{R}^{n\times n}$ are nonzero, then there exists a unique diagonal matrix D, a unique unit lower triangular matrix L and a unique unit upper triangular matrix M, such that $\mathrm{A} = \mathrm{LDM}$.
   b) Let $\mathrm{L}_k = \mathrm{I}_n - \boldsymbol{\ell}^{(k)} \boldsymbol{e}_k^T$ be a Frobenius matrix. Show that
      i) $\mathrm{L}_k^{-1} = \mathrm{I}_n - \boldsymbol{\ell}^{(k)} \boldsymbol{e}_k^T$
      ii) $\mathrm{L} = \prod_{k=1}^{n-1} \mathrm{L}_k = \mathrm{I}_n + \sum_{k=1}^{n-1} \boldsymbol{\ell}^{(k)} \boldsymbol{e}_k^T$

Subtask i): The python file `LUP.py` contains the implementation of `LUP(A)` and `LUPSolver(A,b)`.

Algorithm 2: LU-Decomposition with partial pivoting

```
name: LUP
input: n × n matrix A
output: n × n LT matrix L, n × n UT matrix U, n × n permutation matrix P

LUP(A):
    U = A
    L = I_n
    P = I_n

    for j = 1, ..., n − 1 do
        s = argmax_{k=j,...,n} |U[k, j]|
        if s ≠ j then
            swap(U, s, j)
            swap(P, s, j)
        end

        for i = j + 1, ..., n do
            L[i, j] = U[i,j] / U[j,j]
            for k = j, ..., n do
                U[i, k] = U[i, k] − L[i, j]U[j, k]
            end
        end
    end
    return L, U, P
```

Subtask ii)

The setup for computing the data can be found in the submitted Jupyter-Notebook `ex_4_3.ipynb`. Below is the visualization of the solution vector. The numerical results can be found at the end of the document.

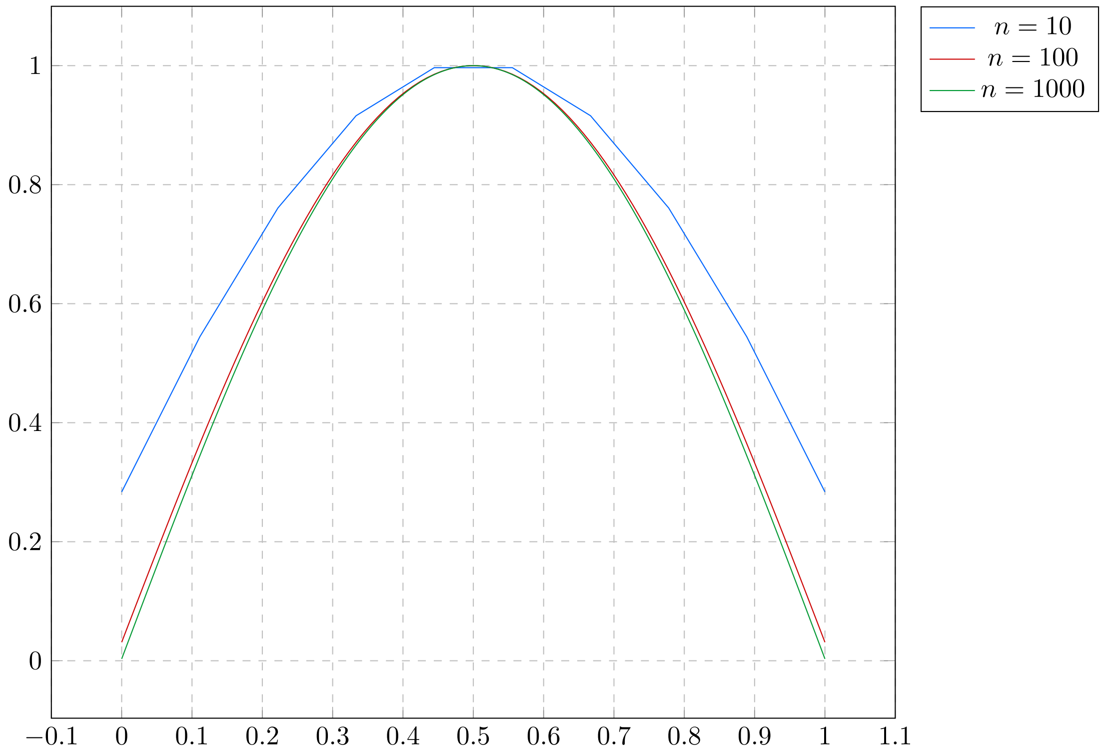

Figure 2: Visualization of the solution vectors for $n = 10, 100, 1000$

Subtask iii):

Sub-Subtask a): If all principal minors of A are non-zero, then $\det \mathrm{A} \neq 0$, since all leading principal minors are non-zero, and the n-th leading principal minor of A is just $\det \mathrm{A}$. Therefore A is regular. Additionally, all principal minors of first order are non-zero, thus all diagonal elements of A are non-zero. Therefore we can apply LU decomposition and, since no diagonal entry is 0, no row-swaps occur, thus $\mathrm{P} = \mathrm{I}_n$, and therefore $\mathrm{PA} = \mathrm{A} = \mathrm{LU}$.

Sub-Subtask b): Sub-Sub-Subtask i): Let $\mathrm{E}_{ij} \in \mathbb{R}^{n\times n}$ be given by:

$$\mathrm{E}_{ij} = \left[\delta_{ik}\delta_{jl}\right]_{k,l=1}^{n}$$

Then $\mathrm{L}_k \in \mathcal{F}_k^{n\times n}$ is given by:

$$\mathrm{L}_k = \mathrm{I} + \sum_{i=k+1}^{n} \lambda_i \mathrm{E}_{ik}$$

Let $\mathrm{L}_k, \mathrm{M}_k \in \mathcal{F}_k^{n\times n}$:

$$\begin{aligned}
\mathrm{L}_k\mathrm{M}_k &= \left(\mathrm{I}_n + \sum_{i=k+1}^{n} \lambda_i \mathrm{E}_{ik}\right)\left(\mathrm{I}_n + \sum_{j=k+1}^{n} \mu_j \mathrm{E}_{jk}\right) \\
&= \mathrm{I}_n + \sum_{i=k+1}^{n} \lambda_i \mathrm{E}_{ik} + \sum_{j=k+1}^{n} \mu_j \mathrm{E}_{jk} + \sum_{i=k+1}^{n}\sum_{j=k+1}^{n} \lambda_i\mu_j \mathrm{E}_{ik}\mathrm{E}_{jk} \\
&= \mathrm{I}_n + \sum_{i=k+1}^{n} (\lambda_i + \mu_i)\mathrm{E}_{ik} \stackrel{!}{=} \mathrm{I}_n \Rightarrow \mu_i = -\lambda_i \\
&\Rightarrow \mathrm{L}_k^{-1} = \mathrm{I}_n - \sum_{i=k+1}^{n} \lambda_i \mathrm{E}_{ik}
\end{aligned}$$

Sub-Sub-Subtask ii): We prove this by induction. Let $L_1, L_2 \in \mathcal{F}_k^{n\times n}$, then we already showed the following:

$$L_1 L_2 = I_n + \sum_{i=k+1}^{n} (\lambda_{1,i} + \lambda_{2,i}) E_{ik}$$

Now let $N = \prod_{k=1}^{n-2} L_k$, $\omega_i = \sum_{j=1}^{n-2} \lambda_{j,i}$ and $L_{n-1} \in \mathcal{F}_k^{n\times n}$, then:

$$\prod_{k=1}^{n-1} L_k = N L_{n-1} = I_n + \sum_{i=k+1}^{n} (\omega_i + \lambda_{n-1,i}) E_{ik}$$

Task 4.4: Cholesky Decomposition

i) Let

$$\mathrm{A} = \begin{bmatrix} 9 & 6 & 3 & 3 \\ 6 & 8 & 6 & -2 \\ 3 & 6 & 6 & -3 \\ 3 & -2 & -3 & 9 \end{bmatrix}$$

Show that $\mathrm{A} = \mathrm{A}^T$, $\mathrm{A} > 0$ and compute L, such that $\mathrm{A} = \mathrm{L}\mathrm{L}^T$.

ii) Write a python script containing the following functions:

a) Given a symmetric positive definite matrix $\mathrm{A} \in \mathbb{R}^{n\times n}$, write a function `CholeskyDecom(A)` which computes the cholesky-decomposition of A. Given a vector $\boldsymbol{b} \in \mathbb{R}^n$, write another function `CholeskySolver(A,b)` which returns the solution $\boldsymbol{x}$ of the matrix equation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$.

b) From Task 4.1: Let $\boldsymbol{b}_n = \pi^2 \boldsymbol{v}_1$. For each $n \in \{10, 100, 1000\}$, approximate the solution of the linear system $\mathrm{A}_n \boldsymbol{x}_n = \boldsymbol{b}_n$, using your function `CholeskySolver` and provide a visualization of the solution. Calculate the residual norm $\|\boldsymbol{b}_n - \mathrm{A}_n \widetilde{\boldsymbol{x}}_n\|_2$ and the *error* norm $e_n = (n+1)^{-1}\|\widetilde{\boldsymbol{x}}_n - \boldsymbol{v}_n\|_2$, where $\widetilde{\boldsymbol{x}}_n$ ist the obtained approximate solution.

Subtask i)

$$\mathrm{A}^T = \begin{bmatrix} 9 & 6 & 3 & 3 \\ 6 & 8 & 6 & -2 \\ 3 & 6 & 6 & -3 \\ 3 & -2 & -3 & 9 \end{bmatrix}$$

We compute the Cholesky-decomposition L:

$$\mathrm{L} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & -2 & 0 & 2 \end{bmatrix} \qquad \mathrm{L}\mathrm{L}^T = \begin{bmatrix} 9 & 6 & 3 & 3 \\ 6 & 8 & 6 & -2 \\ 3 & 6 & 6 & -3 \\ 3 & -2 & -3 & 9 \end{bmatrix}$$

Since there exists $\mathrm{L} \in \mathbb{R}^{4\times 4}$ such that $\mathrm{A} = \mathrm{L}\mathrm{L}^T$, it follows directly, that $\mathrm{A} > 0$.

Subtask ii):

The python-file `Cholesky.py` contains the implementation of `CholeskyDecom(A)` and `CholeskySolver(A,b)`.

Sub-Subtask a):

Algorithm 3: Cholesky Decomposition

```
name: CholeskyDecom
input: symmetric, positive definite n × n matrix A
output: lower triangular n × n matrix L

CholeskyDecom(A):
    L = 0_{n×n}
    for k = 1,...,n do
        L[k,k] = sqrt(A[k,k] − Σ_{i=1}^{k−1} L[i,i]^2)
        for i = k+1,...,n do
            L[i,k] = 1/L[k,k] (A[i,k] − Σ_{j=1}^{k−1} L[i,j]L[k,j])
        end
    end
    return L
```

Sub-Subtask b): The setup for computing the data can be found in the submitted Jupyter-Notebook `ex_4_4.ipynb`. Below is the visualization of the solution vector. The numerical results can be found at the end of the document.

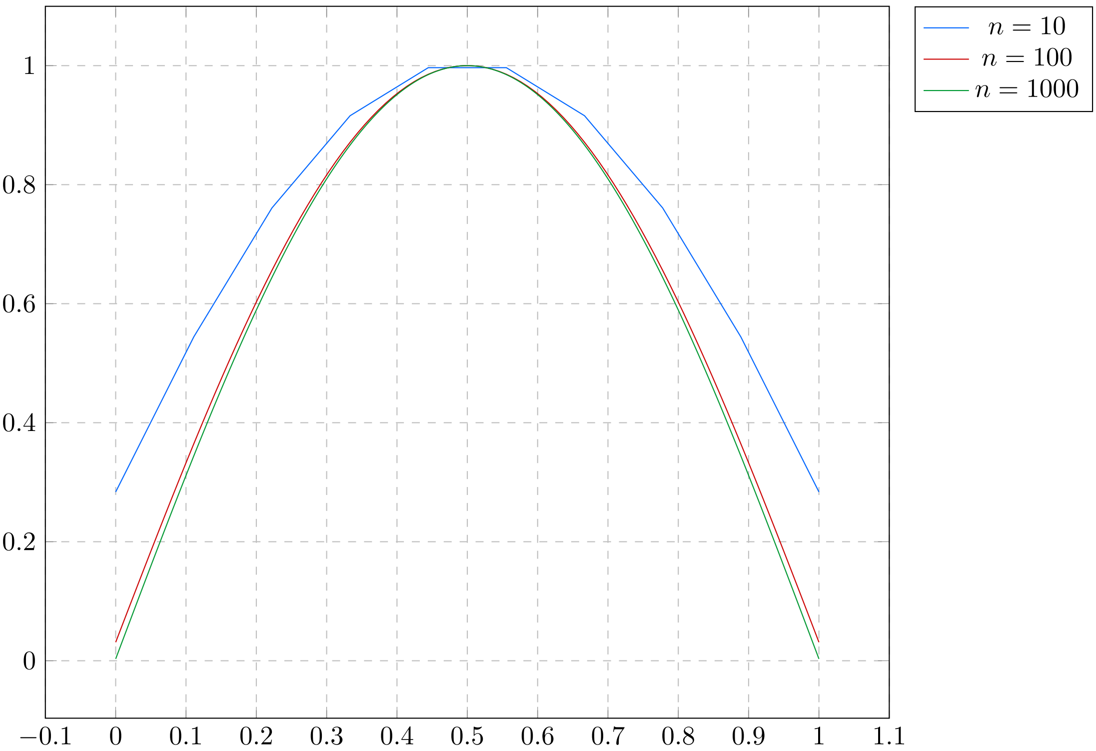

Figure 3: Visualization of the solution vectors for $n = 10, 100, 1000$

Task 4.5: Crout Decomposition

i) An $n \times n$ matrix A is called a ***band-matrix***, if there exist integers p, q with $1 < p$ and $q < n$, such that $a_{ij} = 0$ whenever $p \leq j - i$ or $q \leq i - j$. The ***band width*** of a band matrix is defined as $w = p + q - 1$. Verify that the matrix

$$\mathrm{A} = \begin{bmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & -5 & -6 \end{bmatrix}$$

is a band-matrix and determine it's band-width.

ii) Matrices of band-width 3 are called ***tridiagonal*** because they have the following form:

$$\mathrm{A} = \mathrm{diag}_1^n(a_{ii}) + \underbrace{\begin{bmatrix} \mathbf{0}_{n-1} & \mathrm{diag}_1^{n-1}(a_{i,i+1}) \\ 0 & \mathbf{0}_{n-1}^T \end{bmatrix}}_{=\mathrm{T}_1^{n-1}(a_{i,i+1})} + \underbrace{\begin{bmatrix} \mathbf{0}_{n-1}^T & 0 \\ \mathrm{diag}_1^{n-1}(a_{i+1,i}) & \mathbf{0}_{n-1} \end{bmatrix}}_{\mathrm{B}_1^{n-1}(a_{i+1,i})}$$

Suppose that a tridiagonal matrix can be factored into the triangular matrices L and U, such that $\mathrm{A} = \mathrm{LU}$, where L and U have the following forms:

$$\mathrm{L} = \mathrm{diag}_1^n(l_{ii}) + \mathrm{B}_1^{n-1}(l_{i+1,i}) \qquad \mathrm{U} = \mathrm{I}_n + \mathrm{T}_1^{n-1}(u_{i,i+1})$$

The Crout decomposition is a variation of the LU decomposition, which produces matrices in the from given above. Modify the LU decomposition and write a python-script containing the function `LUPCrout(A)` which decomposes A into L and U using Crout decomposition. Write another function `LUCSolver(A,b)`, which returns the solution of the equation $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$.

iii) From Task 4.1: Let $\boldsymbol{b}_n = \pi^2 \boldsymbol{v}_1$. For each $n \in \{10, 100, 1000\}$, approximate the solution of the linear system $\mathrm{A}_n \boldsymbol{x}_n = \boldsymbol{b}_n$, using your function `LUCSolver` and provide a visualization of the solution. Calculate the residual norm $\|\boldsymbol{b}_n - \mathrm{A}_n \widetilde{\boldsymbol{x}}_n\|_2$ and the *error* norm $e_n = (n+1)^{-1} \|\widetilde{\boldsymbol{x}}_n - \boldsymbol{v}_n\|_2$, where $\widetilde{\boldsymbol{x}}_n$ ist the obtained approximate solution.

Subtask i): Let $p = q = 1$, then for $p \leq j - i$ we see that $a_{ij} = 0$, as well as for $q \leq i - j$. Thus A is a band matrix with band-width $w = p + q - 1 = 1$.

Subtask ii):

We want to exploit the very simple structure of L and U in order to get a linear-time algorithm for the Crout decomposition of tridiagonal matrices. Thus we compute the matrix product LU:

$$\begin{aligned} \mathrm{LU} &= \mathrm{diag}_1^n(l_{ii}) + \mathrm{diag}_1^n(l_{ii})\mathrm{T}_1^{n-1}(u_{i,i+1}) + \mathrm{B}_1^{n-1}(l_{i+1,i}) + \mathrm{B}_1^{n-1}(l_{i+1,i})\mathrm{T}_1^{n-1}(u_{i,i+1}) \\ &= l_{11}\mathrm{E}_{11} = \mathrm{B}_1^{n-1}(l_{i+1,i}) + \mathrm{diag}_2^n(l_{i,i-1}u_{i-1,i} + l_{ii}) + \mathrm{T}_1^{n-1}(l_{ii}u_{i,i+1}) \end{aligned}$$

In matrix form:

$$\mathrm{LU} = \begin{bmatrix} l_{11} & l_{11}u_{12} & 0 & \cdots & 0 \\ l_{21} & l_{21}u_{12} + l_{22} & l_{22}u_{23} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & l_{n-1,n-1}u_{n-1,n} \\ 0 & \cdots & 0 & l_{n,n-1} & l_{n,n-1}, u_{n-1,n} + l_{n,n} \end{bmatrix}$$

Thus we immediately see, that $l_{11} = a_{11}$, and for $i = 2, \ldots, n$ we get $l_{i,i-1} = a_{i,i-1}$. For the remaining entries we get a recurrence relation, for $k = 2, n$:

$$l_{kk} = a_{kk} - \frac{a_{k-1,k}l_{k,k-1}}{l_{k-1,k-1}}$$
$$u_{k-1,k} = \frac{a_{k-1,k}}{l_{k-1,k-1}}$$

Algorithm 4: Crout Decomposition

```
name: LUPCrout
input: n × n tridiagonal matrix A
output: n × n lower triangular matrix L, n × n unit upper triangular matrix U

LUPCrout(A):
    L = 0_{n×n}
    U = I_n
    L = diag(A, −1)
    L[1, 1] = A[1, 1]

    for k = 2, ..., n − 1 do
        if L[k − 1, k − 1] == 0 then
            error decomposition impossible
        end
        L[k, k] = A[k, k] − A[k − 1, k] L[k, k−1] / L[k−1, k−1]
        U[k − 1, k] = A[k−1, k] / L[k−1, k−1]
    end

    return L,U
```

Subtask iii):

The setup for computing the data can be found in the submitted Jupyter-Notebook `ex_4_5.ipynb`. Below is the visualization of the solution vector. The numerical results can be found at the end of the document.

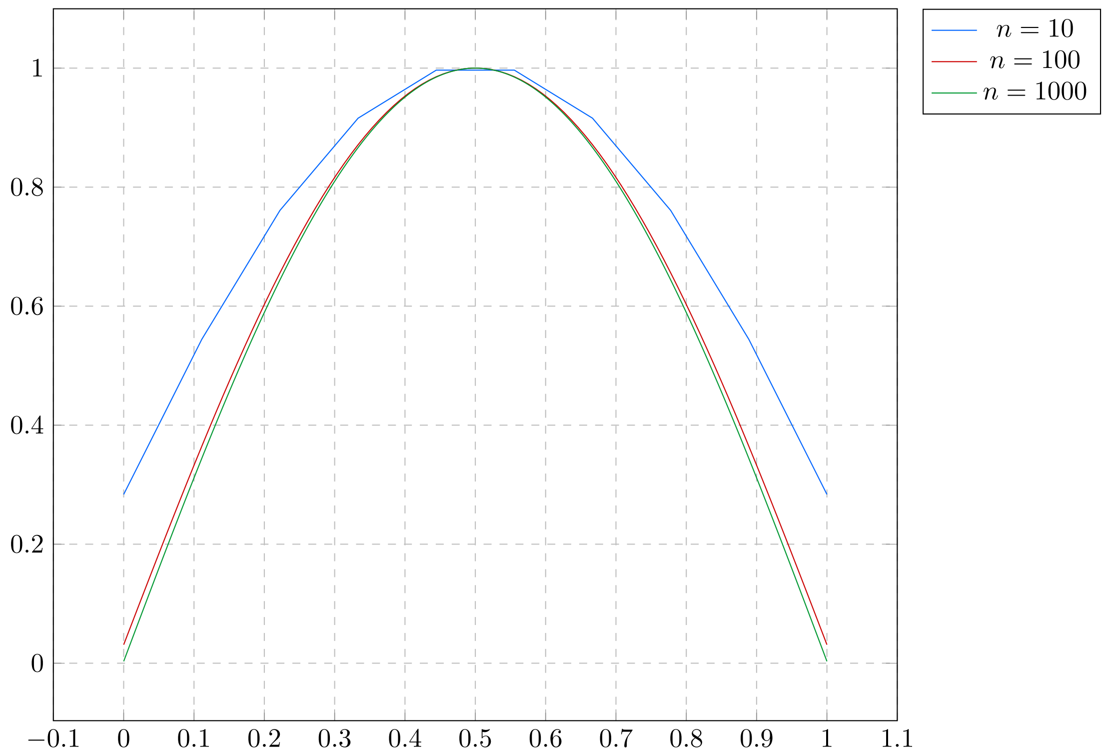

Figure 4: Visualization of the solution vectors for $n = 10, 100, 1000$

Method	n	$\|\mathrm{A}_n\widetilde{\boldsymbol{x}}_n - \boldsymbol{b}_n\|_2$	e_n
Gaussian Elimination	10	$6.5715 \cdot 10^{-14}$	$1.4551 \cdot 10^{-3}$
	100	$4.9939 \cdot 10^{-11}$	$5.6731 \cdot 10^{-6}$
	1000	$4.3038 \cdot 10^{-8}$	$1.8345 \cdot 10^{-8}$
LU-Decomposition	10	$4.4679 \cdot 10^{-14}$	$1.4551 \cdot 10^{-3}$
	100	$1.1328 \cdot 10^{-11}$	$5.6731 \cdot 10^{-6}$
	1000	$2.3963 \cdot 10^{-9}$	$1.8345 \cdot 10^{-8}$
Cholesky-Decomposition	10	$7.8974 \cdot 10^{-14}$	$1.4551 \cdot 10^{-3}$
	100	$1.4089 \cdot 10^{-11}$	$5.6731 \cdot 10^{-6}$
	1000	$3.7026 \cdot 10^{-9}$	$1.8345 \cdot 10^{-8}$
Crout-Decomposition	10	$4.1759 \cdot 10^{-14}$	$1.4551 \cdot 10^{-3}$
	100	$1.0452 \cdot 10^{-11}$	$5.6731 \cdot 10^{-6}$
	1000	$2.9349 \cdot 10^{-9}$	$1.8345 \cdot 10^{-8}$

Table 2: All numerical results from Task 4.2 Subtask ii), Task 4.3 Subtask ii), Task 4.4 Subtask b) and Task 4.5 Subtask iii)

Below are execution times for the various algorithms submitted. Each iteration, the problem $\mathrm{A}_n\boldsymbol{x}_n = \boldsymbol{b}_n$ is solved for $n \in \{10, 100, 1000\}$.

Algorithm	***Iterations***	***Minimum***	***Average***	***Maximum***	***Unit***
Gaussian Elimination	20	7.255	7.425	7.683	s
LU Decomposition	2	151.664	152.492	153.321	s
Cholesky Decomposition	50	2.196	2.219	2.621	s
Crout Decomposition	1000	9.737	10.612	11.906	ms

Table 3: Timings for the submitted algorithms

The program used to measure execution speed is provided in `Timing.py`.

Note

The submission contains a python-file `common.py`, which implements functions that are used throughout the various coding assignments. The supplied functions are:

1. `toeplitz_eigvals(n,a,b,c)` computes all eigenvalues of `toeplitz(n,a,b,c)` for $a \in \mathbb{R}$
2. `toeplitz(n,a,b,c)` constructs a toeplitz tridiagonal matrix $a\mathrm{I}_n + \mathrm{diag}(b, 1) + \mathrm{diag}(c, -1)$
3. `toeplitz_eigvec(n,k,b,c)` computes the k$^{\text{th}}$ eigenvector of `toeplitz(n,a,b,c)` for $a \in \mathbb{R}$
4. `An(n)` constructs A_n from Task 4.1
5. `An_eigvals(n)` computes all eigenvalues of A_n
6. `debug(msg, show)` If `show==True`, then `msg` is printed to stdout
7. `Vandermonde(v)` Construct a Vandermonde-matrix from `v`
8. `backsubs(U,b)` backwards substitution for arbitrary upper triangular matrices
9. `forwsubs(L,b)` forward substitution for arbitrary lower triangular matrices
10. `crout_backsubs(U,b)` backwards substitution for unit upper tridiagonal triangular matrices from Crout decomposition
11. `crout_forwsubs(L,b)` forward substitution for lower tridiagonal triangular matrix from Crout decomposition