Elements of Mathematics

Exercise Sheet 8

Submission due date: **21.12.2021, 10:15h**

THEORY

---

1 Solving Linear Systems with Triangular Matrices: Forward/Backward Substitution

Let $U = (u_{ij})_{ij} \in \mathbb{R}^{n\times n}$ be an upper triangular matrix, i.e., $u_{ij} = 0$ for $i > j$, so that U has the form

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & \vdots \\ \vdots & & \ddots & u_{n-1,n} \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}.$$

1. Solve the following linear system by hand:

$$\begin{pmatrix} 1 & -1 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix}.$$

   Do you observe an iterative pattern?
   Hint: Start with the last equation/row.

2. Let $b \in \mathbb{R}^n$ be a given vector. Then derive a general (iterative) formula to obtain $x \in \mathbb{R}^n$, which solves

$$Ux = b.$$

   What requirements have to be put on the diagonal entries u_{ii} of U, so that the system has a unique solution?
   Hint: Write out the system as

$$\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & \vdots \\ \vdots & & \ddots & u_{n-1,n} \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

   and start with the last row to determine x_n. Then use x_n to determine x_{n-1} from the second but last row. Continue this procedure until you reach the first row to determine x_1 with the help of the previously determined values $x_2, \ldots, x_n$.

3. Find a similar formula for lower triangular matrices $L = (\ell_{ij})_{ij} \in \mathbb{R}^{n\times n}$, for which $\ell_{ij} = 0$ for $i < j$.

Remark: A diagonal matrix is a special case of a triangular matrix. *(9 Points)*

Solution:

1.

$$\begin{matrix}(\mathrm{I})\\(\mathrm{II})\\(\mathrm{III})\end{matrix}\begin{pmatrix}1 & -1 & 1\\ 0 & 2 & 1\\ 0 & 0 & \frac{1}{2}\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}3\\0\\1\end{pmatrix}$$

$$\begin{aligned}(\mathrm{III}) \quad &\Rightarrow \quad \frac{1}{2}x_3 = 1 \quad \Rightarrow \quad x_3 = 2\\ (\mathrm{II}) \quad &\Rightarrow \quad 2x_2 + x_3 = 0 \quad \Rightarrow \quad x_2 = -1\\ (\mathrm{I}) \quad &\Rightarrow \quad x_1 - x_2 + x_3 = 3 \quad \Rightarrow \quad x_1 = 0\end{aligned}$$

$$\Rightarrow \quad x = \begin{pmatrix}0\\-1\\2\end{pmatrix}$$

2. First note that a triangular matrix is invertible if and only if the diagonal entries are nonzero (see the formula below or note that $\det(U) = \prod_i u_{ii}$). Now, from considering the i-th row (equation)

$$u_{ii}x_i + u_{i,i+1}x_{i+1} + \cdots + u_{in}x_n = b_i,$$

we obtain a representation for x_i given by

$$x_i = \underbrace{\frac{1}{u_{ii}}}_{[\text{assume } u_{ii}\neq 0]}\left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\right),$$

where all the x_j for $j \in \{i+1,\ldots,n\}$ in the sum can be computed in previous steps.

3. In the same way, by simply changing the indices in the sum, we find a formula for lower triangular matrices given by

$$x_i = \underbrace{\frac{1}{\ell_{ii}}}_{[\text{assume } \ell_{ii}\neq 0]}\left(b_i - \sum_{j=1}^{i-1} \ell_{ij}x_j\right), \quad i \in \{1,\ldots,n\},$$

where the x_j for $j \in \{1,\ldots,i-1\}$ can be determined in previous steps

2 Solving Linear Systems using LU Decomposition

Consider the following linear systems.

1.

$$\begin{aligned}2x_1 + x_2 + 3x_3 &= -3\\ x_1 - x_2 - x_3 &= 4\\ 3x_1 - 2x_2 + 2x_3 &= 5\end{aligned}$$

2.

$$\begin{aligned}x_1 + 2x_2 + 2x_3 &= 1\\ 2x_1 + x_3 &= 3\\ 3x_1 + 2x_2 + 3x_3 &= 4\end{aligned}$$

3.

$$\begin{aligned} x_1 + x_2 + 2x_3 &= 2 \\ 1x_1 - x_2 &= 0 \\ 2x_1 + 2x_3 &= 1 \end{aligned}$$

Please cast them into the form $Ax = b$ and compute the LU-decomposition of A by rigorously applying Algorithm **??** (i.e., use the pivot element determined in Line **??**):

- Find the matrices L, U and P such that $PA = LU$.
- Then determine the solution set $S := \{x \in \mathbb{R}^n : Ax = b\}$.

Hint: System 2. does not have a *unique* solution (but infinitely many). Determine the set of vectors $x \in \mathbb{R}^3$ for which the linear system 2. is valid.

INPUT: $A \in \mathbb{R}^{n \times n}$ (and $b \in \mathbb{R}^n$)
OUTPUT: LU decomposition $PA = LU$ (and if $Ax = b$ is uniquely solvable the solution $x \in \mathbb{R}^n$)

```
# FACTORIZATION
initialize piv = [1,2,...,n]
for j = 1,...,n-1 do
    # Find the j-th pivot:
    k_j := arg max_{k≥j} |a_kj|
    if a_{k_j j} ≠ 0 then
        # Swap rows
        A[k_j,:] ↔ A[j,:]
        (b[k_j] ↔ b[j])
        piv[k_j] ↔ piv[j]
        # Elimination
        for k = j+1,...,n do
            ℓ_kj := a_kj / a_jj
            a_kj = ℓ_kj
            for i = j+1,...,n do
                a_ki = a_ki - ℓ_kj a_ji
            end
            # by hand we also transform b on the fly:
            (b_k = b_k - ℓ_kj b_j)
        end
    end
end

# SOLVE
...
```

Algorithm 1: Gaussian Elimination with Row Pivoting: Factor *and* Solve

(9 Points)

Solution:

You can later use your Python Code to check your LU decomposition $PA = LU$ with all intermediate steps. Therefore we just copy the factors and solution here.

1. Unique solution:

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & -1/7 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 3 & -2 & 2 \\ 0 & 7/3 & 5/3 \\ 0 & 0 & -10/7 \end{pmatrix}$$

Solving steps yields: $x = (1, -2, -1)^T$, thus $S = \{(1, -2, -1)^T\}$

2. Infinitely many solutions: The algorithm outputs the following arrays

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & -1 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 3 & 2 & 3 \\ 0 & 4/3 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

and we obtain

$$z := L^{-1}P^\top b = \begin{pmatrix} 4 \\ -1/3 \\ 0 \end{pmatrix}.$$

Therefore we see that the system $Ux = z$ has infinitely many solutions (last zero row in U and zero in same row in z). By solving $Ux = z$ we find

$$\begin{array}{ll} \text{(II)} & \Rightarrow \quad 4/3x_2 + 1x_3 = -1/3 \quad \Rightarrow \quad x_2 = -\frac{1}{4}(1 + 3x_3) \\ \text{(I)} & \Rightarrow \quad 3x_1 + 2x_2 + 3x_3 = 4 \quad \Rightarrow \quad x_1 = \frac{1}{3}(4 - 2x_2 - 3x_3) = \frac{3}{2} - \frac{1}{2}x_3 \end{array}$$

$$\begin{aligned} \Rightarrow \quad S &:= \{x \in \mathbb{R}^3 : Ax = b\} \\ &= \{x \in \mathbb{R}^3 : x_1 = \tfrac{1}{2}(3 - x_3), \quad x_2 = -\tfrac{1}{4}(1 + 3x_3), \quad x_3 \in \mathbb{R}\} \\ &\overset{s := x_3 \in \mathbb{R}^3}{=} \left\{ \begin{pmatrix} \frac{3}{2} \\ -\frac{1}{4} \\ 0 \end{pmatrix} + s \begin{pmatrix} -\frac{1}{2} \\ -\frac{3}{4} \\ 1 \end{pmatrix} : s \in \mathbb{R} \right\}, \text{(i.e., we have infinitely many solutions!)} \end{aligned}$$

3. No solution: $S = \emptyset$

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/2 & -1 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 2 & 0 & 2 \\ 0 & -1 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

and we obtain

$$z := L^{-1}P^\top b = \begin{pmatrix} 1 \\ -1/2 \\ 1 \end{pmatrix}.$$

Thus, last row in U is a zero row but $1 \neq 0$ in z

PROGRAMMING

---

3 Solving Linear Systems with Triangular Matrices

1. Implement a function `solve_tri(A, b, lower=False)` which takes as input a triangular matrix A, a vector b and an optional boolean parameter `lower`, which is set to `False` by default. This function shall first check if the dimensions of the input parameters fit and if the matrix is invertible and if both is true, compute and output the solution x by applying the above derived formulas. Otherwise return a warning that there is a dimension mismatch or that the matrix is not invertible.
2. Test your routine on some examples with lower and upper triangular matrices. Find the corresponding SciPy Routine and compare.

(8 Points)

Solution:

```
import numpy as np


def solve_tri(A, b, lower=False):
    """
    Solves a system Ax = b where A is triangular.

    Parameters:
    -----------
    A : triangular matrix as numpy.ndarray of shape (n,n)
    b : right-hand side vector as numpy.ndarray of shape (n,)
    lower : boolean determining if lower or upper triangular

    Returns:
    --------
    x : solution of Ax = b as numpy.ndarray of shape (n,)
    """
    m, n = A.shape
    nb = len(b)
    d = A.diagonal()

    # test dimensions of A and b
    if m != nb:
        raise Exception('dimension of A and b must match. The dimension for A\
                        and b are: {} and {}'.format(np.shape(A), len(b)))

    # test if A is invertible: quadratic?
    if n != m:
        raise Exception('A is not invertible, its shape is nonquadratic:\
                        {}'.format(np.shape(A)))

    # test if A is invertible: nonzero diagonals?
```

```
    if np.any(d == 0):
        raise Exception('A is not invertible, it has zero diagonal entries')

    # A is invertible:
    else:
        x = np.zeros(n)
        # solve for lower triangular matrix
        if lower:
            for i in range(n):
                for j in range(i):
                    x[i] += 1. / A[i, i] * (-A[i, j] * x[j])
                x[i] += 1. / A[i, i] * b[i]

        # solve for upper triangular matrix
        else:
            for i in range(n)[::-1]:
                for j in range(i+1, n):
                    x[i] += 1. / A[i, i] * (-A[i, j] * x[j])
                x[i] += 1. / A[i, i] * b[i]
    return x


if __name__ == "__main__":
    ## Test dimension mismatch
#    A = np.array([[2, 0],
#                  [0, 1]])
#    b = np.array([6])
#    x = solve_tri(A, b, lower = True)

    ## Test nonqudratic A
#    A = np.array([[2, 0, 1],
#                  [0, 1, 1],])
#    b = np.array([6, 2])
#    x = solve_tri(A,b, lower = True)

    ## Test noninvertible A
#    A = np.array([[2, 0,],
#                  [0, 0],])
#    b = np.array([6,2])
#    x = solve_tri(A,b, lower = True)

    ## Test: ill-conditioned for small delta and large n
    n = 100
    delta = 1.0
    # draw from uniform distribution, shift with -0.5 and strengthen diagonal
    A = np.tril(np.random.rand(n,n)-0.5) + delta * np.eye(n)
    print("det(A) = {}\ncond(A) = {}\nmin(diag(A)) = {}".format(np.linalg.det(A),
                                                               np.linalg.cond(A),
                                                               np.min(abs(A.diagonal())
                                                               )))
    b = np.random.rand(n)
    x = solve_tri(A,b, lower = True)
    print("our test Ax=b:", np.allclose(A.dot(x),b))
    # in SciPy
    from scipy.linalg import solve_triangular
    x = solve_triangular(A, b, lower = True)
    print("scipy test Ax=b:", np.allclose(A.dot(x),b))
    # imshow on the matrix
#    import matplotlib.pyplot as plt
#    plt.imshow(A)
#    plt.show()
```

```
    ### Test ill-conditioned diagonal matrix ##
#    n = 10
#    D = np.diag(np.arange(1,n+1))
#    b = np.ones(n)
#    x = solve_tri(D, b, lower = True)
#    print("diag test Ax=b:", np.allclose(D.dot(x),b))
#    # imshow on the matrix
#    import matplotlib.pyplot as plt
#    plt.imshow(D)
#    plt.show()
#    # condition and determinant of the matrix
#    print("condition number of = ", np.linalg.cond(D))
#    print("determinant of A = ", np.linalg.det(D))
```

Total Number of Points = 26 (T:18, P:8)