HW2

xh2433

02/20/2024

1 Question 1

1.1 (a)

Suppose $A \in \mathbb{R}^{n\times n}$, $A = LU$ where L is a lower triangular matrix and U is a upper triangular matrix, $L, U \in \mathbb{R}^{n\times n}$. $l_{ij}, 1 \le j \le i \le n$, denotes the i_{th} row and j_{th} column element of L, $u_{ij}, 1 \le i \le j \le n$, denotes the i_{th} row and j_{th} column element of U, and a_{ij} denotes the denotes the i_{th} row and j_{th} column element of A. Since L and U are triangular matrices, elements beyond the ranges of i and j are trivially 0. The function to calculate l_{ij} is $l_{ij} = \frac{1}{u_{jj}}(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj})$ and the function to calculate u_{ij} is $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}$. For l_{ij}, to calculate the summation of $l_{ik}u_{kj}$ in total costs $j-1$ multiplications, $j-2$ additions. a_{ij} minus the summation costs 1 operation, and the difference divided by u_{jj} costs 1 operation. Therefore, to calculate one element in L costs $j-1+j-2+1+1 = 2j-1$ operations. Since the cost only depends on j, the computation cost of elements of L in the same column is the same. Based on this fact, we can have $Cost_L = \sum_{j=1}^{n}\sum_{i=j}^{n-1}(2j-1)$. For u_{ij}, to calculate the summation of $l_{ik}u_{kj}$ in total costs $i-1$ multiplications and $i-2$ additions. a_{ij} minus the summation costs 1 operation. Therefore, to calculate one element in U costs $i-1+i-2+1 = 2(i-1)$ operations. Since the cost only depends on i, the computation cost of elements in U in the same row is the same. Based on this fact, we can have $Cost_U = \sum_{i=1}^{n}\sum_{j=i}^{n} 2(i-1)$. Simplify and we can get the total cost of LU decomposition is

$$
\begin{aligned}
Cost_{LUdecomposition} &= \textstyle\sum_{j=1}^{n}\sum_{i=j}^{n-1}(2j-1) + \sum_{i=1}^{n}\sum_{j=i}^{n} 2(i-1) \\
&= \textstyle\sum_{j=1}^{n}(n-j)(2j-1) + \sum_{i=1}^{n}(n-i+1)(2i-2) \\
&= \textstyle\sum_{j=1}^{n}(2nj - n - 2j^2 + j) + \sum_{i=1}^{n}(2ni - 2n - 2i^2 + 4i - 2) \\
&\text{since both terms sum from 1 to n, change both variables to k} \\
&= \textstyle\sum_{k=1}^{n} 4nk - 3n - 4k^2 + 5k + 2 \\
&= 4n\frac{n(n+1)}{2} - 3n^2 - 4\frac{n(n+1)(2n+1)}{6} + 5\frac{n(n+1)}{2} + 2n \\
&= 2n^3 + 2n^2 - 3n^2 - \frac{4n^3}{3} - 2n^2 - \frac{2n}{3} + \frac{5n^2}{2} + \frac{5n}{2} - 2n \\
&= \frac{2n^3}{3} - \frac{n^2}{2} - \frac{n}{6}
\end{aligned}
$$

1.2 (b)

python code:

```
import numpy as np

def LU_factor(A):
    L, U = np.zeros(A.shape), np.zeros(A.shape)

    for i in range(A.shape[0]):
        for j in range(A.shape[0]):
            if (i <= j):
                U[i, j] = A[i, j] - np.dot(L[i, 0:i], U[0:i, j])

            if (j <= i):
                L[i, j] = (A[i, j] - np.dot(L[i, :j],U[:j, j]))/U[j, j]

    return L, U

A = np.random.uniform(-5, 5, (100, 100))

L, U = LU_factor(A)
print(U[0, :] == A[0, :])
```

2 Question 2

2.1 (a)

python code:

```
import numpy as np

def get_lower_triangular_inverse (L):
    Linverse = np.zeros(L.shape)

    for j in range(L.shape[0]):
        for i in range(L.shape[0]):
            if i == j:
                Linverse[i, j] =
                (1 - np.dot(L[i, j:i], Linverse[j:i, j])) / L[i, i]
            else:
                Linverse[i, j] =
                ( - np.dot(L[i, j:i], Linverse[j:i, j])) / L[i, i]
    return Linverse
```

```
dim = 10
L = np.tril(np.random.randint(-10, 10, size=(dim, dim)), k=0)

Linverse = get_lower_triangular_inverse(L)
```

Suppose $L \in \mathbb{R}^{n\times n}$, L is a full rank lower triangular matrix. We want to find a lower triangular matrix L^{-1} such that $LL^{-1} = I$ where I denotes the identity matrix. For an element in the i_{th} row and j_{th} column of I, it is equal to the inner product of the i_{th} row of L and j_{th} column of L^{-1}. However, for lower triangular matrices L^{-1} and L, when $j > i$, $L^{-1}_{ij} = 0$ and $L_{ij} = 0$. Based on this fact, we can have $L^{-1}_{ij} = \frac{1}{L_{ii}}(I_{ij} - \sum_{k=j}^{i-1} L_{ik}L^{-1}_{kj})$. To calculate one element, L^{-1}_{ij}, in L^{-1}, there are in total of $i-1-j+1 = i-j$ multiplications, $i-1-j+1-1 = i-j-1$ additions, 1 operation for getting difference between the sum and I_{ij}, and 1 operation for dividing the difference by L_{ii}. Therefore, the total result for getting L^{-1}_{ij} is $i-j+i-j-1+2 = 2i-2j+1$. For each row, the computation cost is the same, but the computation cost increases as i increases. In conclusion, the total computation cost of finding the inverse of a lower triangular matrix is

$$\begin{aligned} Cost_{LowerInverse} &= \sum_{i=1}^{n}\sum_{j=i}^{n} 2(j-i)+1 \\ &= \sum_{i=1}^{n} 2\frac{(n+i)(n-i+1)}{2} - 2i(n-i+1) + n - i + 1 \\ &= \sum_{i=1}^{n} n^2 + 2n + i^2 - 3i - 2ni + 1 \\ &= n^3 + 2n^2 + \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - \frac{3n^2}{2} - \frac{3n}{2} - n^3 - n^2 + n \\ &= \frac{n^3}{3} - \frac{n}{3} \end{aligned}$$

2.2 (b)

Suppose A is a full rank matrix, $A \in \mathbb{R}^{n\times n}$. Assume A has than one set of LU-factorization matrices, so $A = L_1U_1$ and $A = L_2U_2$ where $L_1, L_2 \in \mathbb{R}^{n\times n}$ are two unit Lower triangular matrices, $U_1, U_2 \in \mathbb{R}^{n\times n}$ are two upper triangular matrices, and $L_1 \neq L_2, U_1 \neq U_2$.
Since A is full rank, $det(A) \neq 0$, so $det(L_1U_1) = det(L_1)det(U_1) \neq 0$ and $det(L_2U_2) = det(L_2)det(U_2) \neq 0$. Therefore, L_1, L_2, U_1, U_2 are all full rank matrices, and $L_1^{-1}, L_2^{-1}, U_1^{-1}, U_2^{-1}$ exist.
Since $A = L_1U_1$, we have $U_1 = L_1^{-1}A = L_1^{-1}L_2U_2$. Multiply both sides by U_2^{-1} and we can have $U_1U_2^{-1} = L_1^{-1}L_2$. The inverse of unit lower triangular matrix is also a unit lower triangular matrix, and the product of two unit lower triangular matrices is also a unit lower triangular matrix. The product of two upper triangular matrices is a upper triangular matrix as well. Therefore, we have a upper triangular matrix equal to a unit lower triangular matrix. Therefore, we can conclude that $U_1U_2^{-1} = L_1^{-1}L_2 = I$. By definition of inverse, $U_1 = U_2$ and $L_1 = L_2$, which contradicts the initial assumption. Therefore, A can only have one unique set of LU-factorization matrices L, U.

3 Question 3

Statement: Suppose $L \in \mathbb{R}^{n \times n}$ is a invertible lower triangular matrix. The inverse of L is also a lower triangular matrix.
Proof: By definition of matrix inverse, suppose $LY = I$, so Y is the inverse of L. $Y = y_1, y_2, ..., y_n$ where y_i is the ith column of Y. Similarly, define e_i as the ith column of the identity matrix. So we have $Ly_1 = e_1, ..., Ly_n = e_n$. Partition L, y, e such that $L_k = L[1:k, 1:k], y'_k = y_k[1:k], e'_k = e_k[1:k], k = 1, 2, ..., n$, exclusive. We can have $L_1 y'_1 = e'_1, L_2 y'_2 = e'_2, ..., L_n y'_n = e'_n$. It is obvious that for all k, e'_k is zero vector. Since L is full rank, all the elements lie on the diagonal of L must be non-zero. Therefore, for all k, L_k is also full rank. Therefore, for all k, y'_k is the trivial solution for $L_k y'_k = 0$. By definition of lower triangular matrices, we can see that Y is also a lower triangular matrix.

4 Question 4

python code:

```
import numpy as np
import copy

dim = 10
A = np.random.uniform(-10, 10, size=(dim, dim))
b = np.random.uniform(-10, 10, size=dim)

def find_U_b(A, b):
    dim = A.shape[0]
    for i in range(dim - 1):
        maxInd = np.argmax(A[i:, i])
        A[[i, i + maxInd], :] = A[[i + maxInd, i], :]
        b[i], b[i + maxInd] = b[i + maxInd], b[i]
        for j in range(1, dim - i):
            L = A[i + j, i] / A[i, i]
            A[i + j, i:] -= L * A[i, i:]
            b[i + j] -= L * b[i]
    return A, b

A, b = find_U_b(A, b)

def find_x(A, b):
    dim = A.shape[0]
    x = np.zeros(dim)
    for i in range(dim - 1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:]))/ A[i,i]
    return x
```

```
x = find_x(A, b)
```

Suppose $A \in \mathbb{R}^{n\times n}$. We want to find the solution x to $Ax = b$ where $b, x \in \mathbb{R}^n$ through pivoting and LU decomposition. Suppose we already have $A_{i-1} = L_{i-1}P_{i-1}...L_1P_1A, b_{i-1} = L_{i-1}P_{i-1}...L_1P_1b$ and we want to find L_i, P_i. First, we want to find the index of the largest element $A_{i-1}[maxInd, i]$ in the column vector $A_{i-1}[i : n, i]$ and switch the row $A_{i-1}[i]$ with row $A_{i-1}[maxInd]$. Find the largest element costs $n - i + 1$ operations. Switching the ith row with the $maxInd$ row costs $2(n - i + 1)$ operations.
In order to turn $A_{i-1}[i : n, i]$ to 0, we need to go through $n - i$ rows starting at row i. For each row $i + j$ where j range from 1 to $n - i$, we first compute $L = \frac{A_{i-1}[i+j,i]}{A_{i-1}[i,i]}$, which takes 1 operation. Then, every element in the row $i+j$ will subtracted by L multiplied by every element in the row i. The multiplication and subtraction takes $2(n - i + 1)$ operations. Since there are $n - i$ rows, the computation cost for finding A_i is $(n - i)(2(n - i + 1) + 1) + 3(n - i + 1)$.
Similarly for b_i, we first switch the ith element with the $maxInd$th element of b_{i-1}, costing 1 operation. Then, looping through $j = 1, ..., n - i$, $b[i + j] = b[i + j] - L \times b[i]$, costing 1 operation. For in total of $n - i$ elements, to find b_i costs $n - i + 1$ operations.
In total, doing the ith step of LU decomposition costs $(n-i)(2(n-i+1)+1)+3(n - i + 1) + n - i + 1$. There are in total of $n - 1$ steps, so we have

$$\begin{aligned} Cost_{LU} &= \textstyle\sum_{i=1}^{n-1}(n - i)(2(n - i + 1) + 1) + 3(n - i + 1) + n - i + 1 \\ &= \textstyle\sum_{i=1}^{n-1} 2n^2 - 4ni + 7n + 2i^2 - 7i + 4 \\ = 2n^2(n - 1) - 4n\tfrac{n(n-1)}{2} &+ 7n(n - 1) + \tfrac{(n-1)n(2n-1)}{3} - \tfrac{7n(n-1)}{2} + 4(n - 1) \\ &= \tfrac{2n^3}{3} + \tfrac{5n^2}{2} + \tfrac{5n}{6} - 4 \end{aligned}$$

In addition to finding the U with LU decomposition, we need solve $A'x = b'$. A', after LU decomposition, will become an upper triangular matrix. To solve for x_i, it is to solve for $b'_i = A'[i] \cdot x$ where $A'[i]$ denotes the ith row of A. However, since A' is an upper triangular matrix, the first $i - 1$ elements in $A'[i]$ is 0. Therefore, we can have $b'_i = \sum_{k=i}^{n} A'[i, k] \times x_k$. After basic algebra, we can get $x_i = \frac{1}{A'[i,i]}(b'_i - \sum_{k=i+1}^{n} A'[i, k]x_k)$. There are in total $n - i$ multiplications and $n - i - 1$ additions plus two more operations for basic algebra. In total we can find the computation cost for solving $A'x = b$ to be

$$Cost_{linear-system} = \textstyle\sum_{i=1}^{n}(n - i + n - i + 1) = n^2$$

Therefore, the total cost of solving Linear System $Ax = b$ is $\frac{2n^3}{3} + \frac{7n^2}{2} + \frac{5n}{6} - 4$

5 Question 5

Algorithm: Implicit Method, $M(T, x)$
Suppose $S, T, I \in \mathbb{R}^{n\times n}$.
For $n = k+1$, suppose we already know the solution x_k to $(S_kT_k - \lambda I_k)x_k = b_k$.

$$(S_{k+1}T_{k+1}) - \lambda I_{k+1})x_{k+1} = b_{k+1}$$
$$(\begin{bmatrix} \sigma_{k+1} & u_{k+1}^T \\ 0 & S_k \end{bmatrix} \begin{bmatrix} \tau_{k+1} & v_{k+1}^T \\ 0 & T_k \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda I_k \end{bmatrix}) \begin{bmatrix} \gamma_{k+1} \\ x_k \end{bmatrix} = \begin{bmatrix} \beta_{k+1} \\ b_k \end{bmatrix}.$$

From the matrix multiplication, we can solve for $\gamma_{k+1} = \frac{\beta_{k+1} - \sigma_{k+1} v_{k+1}^T x_k - u_{k+1}^T T_k x_k}{\sigma_{k+1}\tau_{k+1} - \lambda}$. Solving for $\sigma_{k+1} v_{k+1}^T x_k$ takes $\mathbb{O}(k)$ operations, and calculating other terms except $T_k x_k$ takes constant time. However, to simply multiply T_k and x_k requires $\mathcal{O}(k^2)$ time. So we let $M(T_k, x_k) = T_k x_k = \begin{bmatrix} \tau_k & v_k^T \\ 0 & T_{k-1} \end{bmatrix} \begin{bmatrix} \gamma_k \\ x_{k-1} \end{bmatrix} = \begin{bmatrix} \tau_k \gamma_k + v_k^T x_{k-1} \\ T_{k-1} x_{k-1} \end{bmatrix}$. If we run a recursive call to calculate $T_{k-1}x_{k-1}$ so that $T_{k-1}x_{k-1} = M(T_{k-1}, x_{k-1})$, we will find the runtime for solving $T_k x_k$ to be the same as solving for $\tau_k \gamma_k + v_k^T x_{k-1}$, which takes $\mathcal{O}(k)$ operations. To sum up, solving for γ_k requires $\mathcal{O}(k)$ operations after running the recursive call. If solving γ_k for $k = 1, 2, ..., n$ is $\mathcal{O}(k)$, solving for x takes $\mathcal{O}(\frac{n(n+1)}{2}) = \mathcal{O}(n^2)$.

6 Question 6

Suppose $A \in \mathbb{R}^{n \times n}$ is a full rank matrix. The goal is to find the inverse of A with LU decomposition. We can first do the LU decomposition on A, so that $A = LU$. As proven in question 1, this step takes approximately $\frac{2n^3}{3}$ operations. $A^{-1} = (LU)^{-1} = U^{-1}L^{-1}$. Next, we want to find the inverse of L and U. As proven in question 2, find the inverse of L takes approximately $\frac{n^3}{3}$ operations. It is easy to prove based on the proof of inverse of L that finding the inverse of U also takes approximately $\frac{n^3}{3}$ operations as L is lower triangular matrix and U is upper triangular matrix. Multiplying U^{-1} and L^{-1} takes around $2\sum_{i=1}^{n}(2(n-i)+1)i \approx \frac{2n^3}{3}$ operations. Therefore, in total, we can solve for A^{-1} with LU decomposition with approximately $2n^3$ operations.