矩阵分解实验报告

实验介绍

- 矩阵分解的 LU、QR（Gram-Schmidt）、Orthogonal Reduction (Householder reduction 和 Givens reduction)和 URV 程序实现

程序说明

LU decomposition

```python
# LU decomposition
def LU_decomposition(A):
    """
    :param A: n*n matrix
    :return: L, U, P
    """
    _A = A.copy()
    if _A.shape[0] != _A.shape[1]:
        print("The matrix is not square! No LU decomposition!")
        return None
    n = _A.shape[0]
    det = np.linalg.det(_A)
    if det == 0:
        print("The matrix is singular! No LU decomposition!")
        return

    P = np.eye(n)

    for i in range(n):
        pivot_idx = np.argmax(np.abs(_A[i:, i])) + i
        _A[i], _A[pivot_idx] = _A[pivot_idx].copy(), _A[i].copy()
        p_i = np.eye(n)
        p_i[i], p_i[pivot_idx] = p_i[pivot_idx].copy(), p_i[i].copy()
        P = p_i @ P
        for j in range(i + 1, n):
            _A[j, i] = _A[j, i] / _A[i, i]
            _A[j, i + 1:n] = _A[j, i + 1:n] - _A[j, i] * _A[i, i + 1:n]

    L = np.tril(_A, -1) + np.eye(n)
    U = np.triu(_A)
    return L, U, P
```

输入待分解矩阵A，输出LU分解结果中的P, L, U矩阵,其中P为旋转矩阵，L为对角线元素为1的下三角矩阵，U为对角线元素不为0的上三角矩阵。具体实现方式为：先判断矩阵是否满足LU分解条件，后续利用部分主元法进行LU分解以避免U矩阵的对角元素出现0。

Gram-Schmidt QR decomposition

```python
# Gram-Schmidt QR decomposition
def Gram_Schmidt_QR_decomposition(A):
    """
    :param A: m*n matrix with independent columns
    :return: Q, R
    """
    if np.linalg.matrix_rank(A) != A.shape[1]:
        print("The matrix's columns are linear dependent! No QR decomposition!")
        return

    m, n = A.shape
    Q = np.zeros((m, n))
    R = np.zeros((n, n))

    for i in range(n):
        for j in range(0, i):
            R[j, i] = np.dot(Q[:, j], A[:, i])
        q_i = A[:, i] - np.dot(Q[:, 0:i], R[0:i, i])
        R[i, i] = np.linalg.norm(q_i)
        Q[:, i] = q_i / R[i, i]

    return Q, R
```

输入待分解矩阵A， 输出QR分解结果中的Q, R矩阵，其中Q的列为R(A)的标准正交基，R为上三角矩阵。具体实现方式为：先判断输入矩阵是否满足QR分解条件，后续对其使用标准的施密特正交化方法进行QR分解。

Householder reduction

```python
def Householder_reduction(A):
    """
    :param A: m*n matrix
    :return: P, T
    """
    m, n = A.shape
    P = np.eye(m)
    T = A.copy()

    for i in range(m - 1):
        x = T[i:, i]
        e = np.zeros_like(x)
        e[0] = np.linalg.norm(x)
        u = x - e
        u = u / np.linalg.norm(u)
        R = np.eye(m)
        R[i:, i:] -= 2 * np.outer(u, u)
        P = R @ P
        T = R @ T

    return P, T
```

输入待分解矩阵A，输出Householder正交分解结果中的P, T矩阵，其中P为正交矩阵，T为上三角矩阵。具体实现方式为：使用反射算子的特殊性质对矩阵每一列进行变化，逐渐将矩阵化简为上三角矩阵。

Givens reduction

```
# Givens reduction
def Givens_reduction(A):
    """
    :param A: m*n matrix
    :return: P, T
    """
    m, n = A.shape
    P = np.eye(m)
    T = A.copy()

    for i in range(n):
        for j in range(i + 1, m):
            G = np.eye(m)
            G[i, i] = T[i, i] / np.sqrt(T[i, i] ** 2 + T[j, i] ** 2)
            G[j, j] = G[i, i]
            G[i, j] = T[j, i] / np.sqrt(T[i, i] ** 2 + T[j, i] ** 2)
            G[j, i] = -G[i, j]
            P = G @ P
            T = G @ T

    return P, T
```

输入待分解矩阵A，输出Givens正交分解结果中的P, T矩阵，其中P为正交矩阵，T为上三角矩阵。具体实现方式为：使用旋转矩阵对矩阵每一列进行变化，逐渐将矩阵化简为上三角矩阵。

URV decomposition

```
# URV decomposition
def URV_decomposition(A):
    """
    :param A: m*n matrix
    :return: U, R, V_T
    """
    P, B = Householder_reduction(A)
    Q, T = Householder_reduction(B.T)
    return P.T, T.T, Q
```

输入待分解矩阵A，输出URV分解结果中的U, R, V矩阵，其中U, V为正交矩阵。 具体实现方式为：使用正交分解（如Householder分解）将矩阵A分解成一个正交矩阵P和上三角矩阵B的乘积，然后对B的转置再次使用正交分解为正交矩阵Q和上三角矩阵T。从而原矩阵A就可以使用P, T, Q表示为URV分解的形式。

求解函数 solve

```python
def solve(A, b, method="LU"):
    """
    :param A: The coefficient matrix
    :param b: The right-hand side vector
    :param method: decomposition method
    :return: The solution vector
    """
    if method == "LU":
        L, U, P = LU_decomposition(A)
        y = np.linalg.solve(L, P @ b)
        x = np.linalg.solve(U, y)
    elif method == "QR":
        Q, R = Gram_Schmidt_QR_decomposition(A)
        y = np.linalg.solve(Q, b)
        x = np.linalg.solve(R, y)
    elif method == "HR":
        P, T = Householder_reduction(A)
        x = np.linalg.solve(T, P @ b)
    elif method == "GR":
        P, T = Givens_reduction(A)
        x = np.linalg.solve(T, P @ b)
    elif method == "URV":
        U, R, V_T = URV_decomposition(A)
        y = np.linalg.solve(R, U.T @ b)
        x = V_T.T @ y
    else:
        raise ValueError("The method is not supported!")
    return x
```

输入待求解方程对应的A，b以及分解方法method，输出为利用相应矩阵分解结果求得的方程的解。

行列式函数 determinant

```python
def determinant(A, method="LU"):
    """
    :param A: n*n matrix
    :param method: decomposition method
    :return: determinant of A
    """
    if method == "LU":
        L, U, P = LU_decomposition(A)
        det = np.linalg.det(P) * np.prod(np.diag(U))
    elif method == "QR":
        Q, R = Gram_Schmidt_QR_decomposition(A)
        det = np.linalg.det(Q) * np.prod(np.diag(R))
    elif method == "HR":
        P, T = Householder_reduction(A)
        det = np.linalg.det(P) * np.prod(np.diag(T))
    elif method == "GR":
        P, T = Givens_reduction(A)
        det = np.linalg.det(P) * np.prod(np.diag(T))
    elif method == "URV":
```

```
        U, R, V_T = URV_decomposition(A)
        det = np.linalg.det(U) * np.linalg.det(R) * np.linalg.det(V_T)
    else:
        raise ValueError("The method is not supported!")
    return det
```

输入为矩阵A，分解方法method，输出为利用相应矩阵分解结果求得的矩阵A的行列式。

实验结果

输入方程组Ax=b中的A，b如下：

```
A:
 [[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
b:
 [ 3. 60.  1.  5.]
```

LU decomposition

```
LU decomposition:
L:
 [[ 1.          0.          0.          0.        ]
 [-0.75        1.          0.          0.        ]
 [ 0.25        0.          1.          0.        ]
 [ 0.5        -0.2         0.33333333  1.        ]]
U:
 [[  4.   8.  12.  -8.]
 [  0.   5.  10. -10.]
 [  0.   0.  -6.   6.]
 [  0.   0.   0.   1.]]
P:
 [[0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]]
A:
 [[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
P * A:
 [[ 4.  8. 12. -8.]
 [-3. -1.  1. -4.]
 [ 1.  2. -3.  4.]
 [ 2.  3.  2.  1.]]
L * U:
 [[ 4.  8. 12. -8.]
 [-3. -1.  1. -4.]
 [ 1.  2. -3.  4.]
 [ 2.  3.  2.  1.]]
PA=LU, LU decomposition is correct!
```

```

Determinant of A:
 120.0
Solution of LU:
 [ 12.   6. -13. -15.]
```

Gram-Schmidt QR decomposition

```
Gram-Schmidt QR decomposition:
Q:
 [[ 0.18257419  0.14007078 -0.92527712 -0.30151134]
 [ 0.73029674  0.56028312  0.30751857 -0.24120908]
 [ 0.36514837  0.03295783 -0.21771226  0.90453403]
 [-0.54772256  0.81570631 -0.04354245  0.18090681]]
R:
 [[ 5.47722558  7.85068999  8.39841255 -2.5560386 ]
 [ 0.          4.04557371  7.18480708 -7.15184925]
 [ 0.          0.          5.98708726 -6.20479952]
 [ 0.          0.          0.          0.90453403]]
A:
 [[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
Q * R:
 [[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
A=QR, QR decomposition is correct!

Determinant of A:
 119.99999999999996
Solution of QR:
 [ 12.   6. -13. -15.]
```

Householder reduction

```
Householder reduction:
P:
 [[ 0.18257419  0.73029674  0.36514837 -0.54772256]
 [ 0.14007078  0.56028312  0.03295783  0.81570631]
 [-0.92527712  0.30751857 -0.21771226 -0.04354245]
 [ 0.30151134  0.24120908 -0.90453403 -0.18090681]]
T:
 [[ 5.47722558  7.85068999  8.39841255 -2.5560386 ]
 [-0.          4.04557371  7.18480708 -7.15184925]
 [ 0.          0.          5.98708726 -6.20479952]
 [-0.         -0.          0.         -0.90453403]]
A:
 [[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
```

```
P.T * T:
 [[ 1.   2. -3.   4.]
 [ 4.   8. 12. -8.]
 [ 2.   3.   2.   1.]
 [-3. -1.   1. -4.]]
PA=T, Householder reduction is correct!

Determinant of A:
 120.00000000000014
Solution of HR:
 [ 12.    6. -13. -15.]
```

Givens reduction

```
Givens reduction:
P:
 [[ 0.18257419  0.73029674  0.36514837 -0.54772256]
 [ 0.14007078  0.56028312  0.03295783  0.81570631]
 [-0.92527712  0.30751857 -0.21771226 -0.04354245]
 [-0.30151134 -0.24120908  0.90453403  0.18090681]]
T:
 [[ 5.47722558  7.85068999  8.39841255 -2.5560386 ]
 [-0.          4.04557371  7.18480708 -7.15184925]
 [ 0.          0.          5.98708726 -6.20479952]
 [-0.         -0.          0.          0.90453403]]
A:
 [[ 1.   2. -3.   4.]
 [ 4.   8. 12. -8.]
 [ 2.   3.   2.   1.]
 [-3. -1.   1. -4.]]
P.T * T:
 [[ 1.   2. -3.   4.]
 [ 4.   8. 12. -8.]
 [ 2.   3.   2.   1.]
 [-3. -1.   1. -4.]]
PA=T, Givens reduction is correct!

Determinant of A:
 119.99999999999991
Solution of GR:
 [ 12.    6. -13. -15.]
```

URV decomposition

```
URV decomposition:
U:
 [[ 0.18257419  0.14007078 -0.92527712  0.30151134]
 [ 0.73029674  0.56028312  0.30751857  0.24120908]
 [ 0.36514837  0.03295783 -0.21771226 -0.90453403]
 [-0.54772256  0.81570631 -0.04354245 -0.18090681]]
R:
 [[12.98845641  0.          -0.          -0.        ]
 [ 8.49846343  6.84932016 -0.          0.        ]
 [ 5.09234768  6.4407454   2.63240286 -0.        ]
```

```
 [ 0.17800606  0.72362117  0.01721768  0.51241742]]
V.T:
 [[ 0.4216995   0.60443595  0.6466059  -0.1967931 ]
 [-0.52323409 -0.15931553  0.24668879 -0.79999331]
 [ 0.46443388 -0.7794731   0.41995302 -0.01903486]
 [ 0.57679875  0.04119991 -0.58709873 -0.56649877]]
A:
 [[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
U * R * V.T:
 [[ 1.  2. -3.  4.]
 [ 4.  8. 12. -8.]
 [ 2.  3.  2.  1.]
 [-3. -1.  1. -4.]]
A=URV.T, URV decomposition is correct!

Determinant of A:
 119.99999999999991
Solution of URV:
 [ 12.   6. -13. -15.]
```