homework 20241022

陈皓阳 $23307130004@m.fudan.edu.cn$

第一题

1. $\min\|Ax-b\|_2^2+\lambda\|x\|_2^2=\min\left\|\begin{bmatrix}A\\ \sqrt{\lambda}I\end{bmatrix}x-\begin{bmatrix}b\\ 0\end{bmatrix}\right\|_2^2$

转化为法方程: $\begin{bmatrix}A\\ \sqrt{\lambda}I\end{bmatrix}^T\left(\begin{bmatrix}A\\ \sqrt{\lambda}I\end{bmatrix}x-\begin{bmatrix}b\\ 0\end{bmatrix}\right)=0$, 即 $(A^TA+\lambda I)x=Ab$.

$$A_{m\times n}=\begin{bmatrix}\times & \times & & \\ & \times & \ddots & \\ & & \ddots & \times \\ & & & \times \\ & & & \end{bmatrix}\qquad A^TA=\left[\begin{array}{cccc|c}\times & & & & \\ \times & \times & & & \\ & \ddots & \ddots & & 0\\ & & \times & \times & \end{array}\right]\left[\begin{array}{cccc}\times & \times & & \\ & \times & \ddots & \\ & & \ddots & \times \\ & & & \times \\ \hline & & 0 & \end{array}\right]=\begin{bmatrix}\times & \times & & \\ \times & \times & \ddots & \\ & \ddots & \ddots & \times \\ & & \times & \times\end{bmatrix}$$

是三对角矩阵.

对于增广矩阵 $[A^TA+\lambda I|Ab]=\left[\begin{array}{cccc|c}\times & \times & & & \times\\ \times & \times & \ddots & & \times\\ & \ddots & \ddots & \times & \vdots\\ & & \times & \times & \times\end{array}\right]$

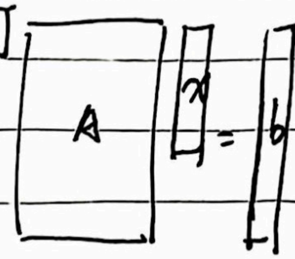
依次对 $r_1,r_2;\ r_2,r_3;\ \cdots;\ r_{n-1},r_n$ 进行Givens变换, 使左边成为上三角阵.

即 $\left[\begin{array}{cccc|c}\times & \times & & & \times\\ \times & \times & \ddots & & \times\\ & \ddots & \ddots & \times & \vdots\\ & & \times & \times & \times\end{array}\right]\xrightarrow[n-1\text{次}]{\text{若干次Givens变换}}\left[\begin{array}{ccccc|c}\times & \times & \times & & & \times\\ & \times & \times & \ddots & & \times\\ & & \ddots & \ddots & \times & \vdots\\ & & & \ddots & \times & \vdots\\ & & & & \times & \times\end{array}\right]=:[R|b']$

问题转化为求解 $Rx=b'$, R是上三角阵.

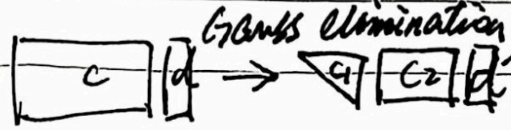
第二题

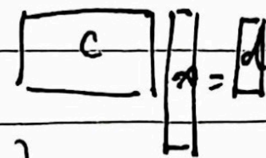
2、对线性约束 $Cx=d$，考虑增广矩阵 $[C|d]$



运用选主元高斯消元，得到：

$[C|d] \longrightarrow [C_1\ C_2 | d']$





其中 C_1 是上三角阵，且可逆的（选主元消除）。

按照 $C=[C_1\ C_2]$，将 x 相应分割为 $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$，则 $[C_1, C_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = d'$。

解得 $x_1 = C_1^{-1}(d' - C_2 x_2)$，$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} C_1^{-1}(d' - C_2 x_2) \\ x_2 \end{bmatrix}$。

按照 $x=\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$，将 A 相应分割为 $[A_1\ A_2]$，则求解 $\min \| [A_1\ A_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - b \|_2$

$\min \| A_1 C_1^{-1}(d' - C_2 x_2) + A_2 x_2 - b \|_2 = \min \| (A_2 - A_1 C_1^{-1} C_2) x_2 + A_1 C_1^{-1} d' - b \|_2$

记 $\tilde{A} = A_2 - A_1 C_1^{-1} C_2$，$\tilde{b} = A_1 C_1^{-1} d' - b$，$\tilde{x} = x_2$

问题转化为无线性约束的最小二乘问题 $\min \| \tilde{A}\tilde{x} - \tilde{b} \|_2$。

第三题

$A\in\mathbb{C}^{m\times n}$
$C\in\mathbb{C}^{r\times n}$

3、考虑Lagrange乘子法：$L(x,l)=\frac{1}{2}\|Ax-b\|_2^2+l^*(Cx-d)$，$l\in\mathbb{C}^p$ $(r<m)$

$L(x,l)=\frac{1}{2}(Ax-b)^*(Ax-b)+l^*(x-l^*d$

$=\frac{1}{2}(x^*A^*Ax-x^*A^*b-b^*Ax+b^*b)+l^*(x-l^*d$. 希望取极值

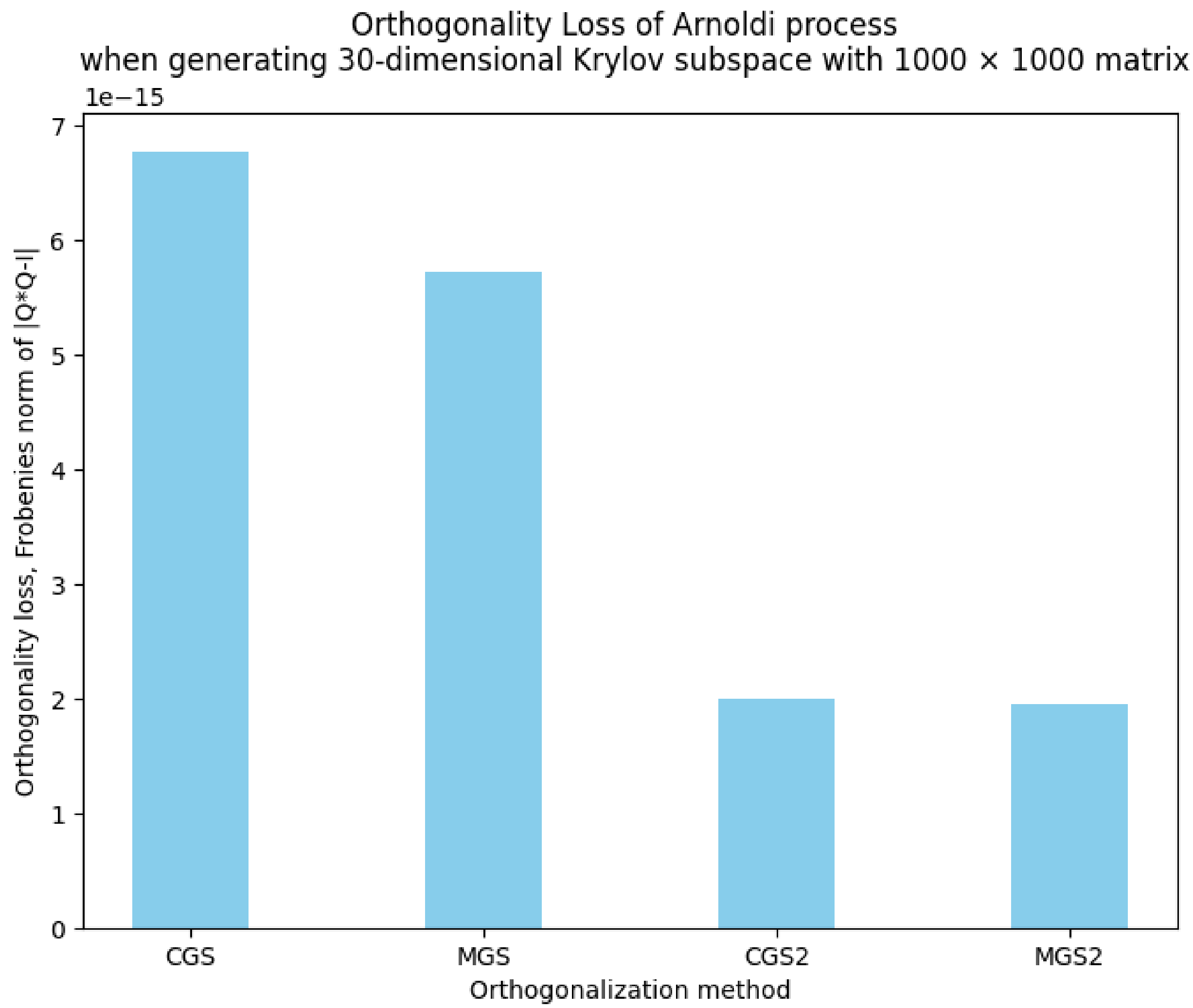
$L_x^*(x,l)=\frac{1}{2}(2A^*Ax-2A^*b)+C^*l=A^*(Ax-b)+C^*l=0$.

记 $Ax-b=-r$. 有 $\begin{cases} b-Ax=r \\ A^*r-C^*l=0 \\ Cx=d \end{cases} \iff \begin{bmatrix} I & A & 0 \\ A^* & 0 & C^* \\ 0 & C & 0 \end{bmatrix}\begin{bmatrix} r \\ x \\ l \end{bmatrix}=\begin{bmatrix} b \\ 0 \\ d \end{bmatrix}$

再令 $\tilde{l}=-l$. 可有 $\begin{bmatrix} I & A & 0 \\ A^* & 0 & C^* \\ 0 & C & 0 \end{bmatrix}\begin{bmatrix} r \\ x \\ \tilde{l} \end{bmatrix}=\begin{bmatrix} b \\ 0 \\ d \end{bmatrix}$ 系数矩阵是Hermitian阵

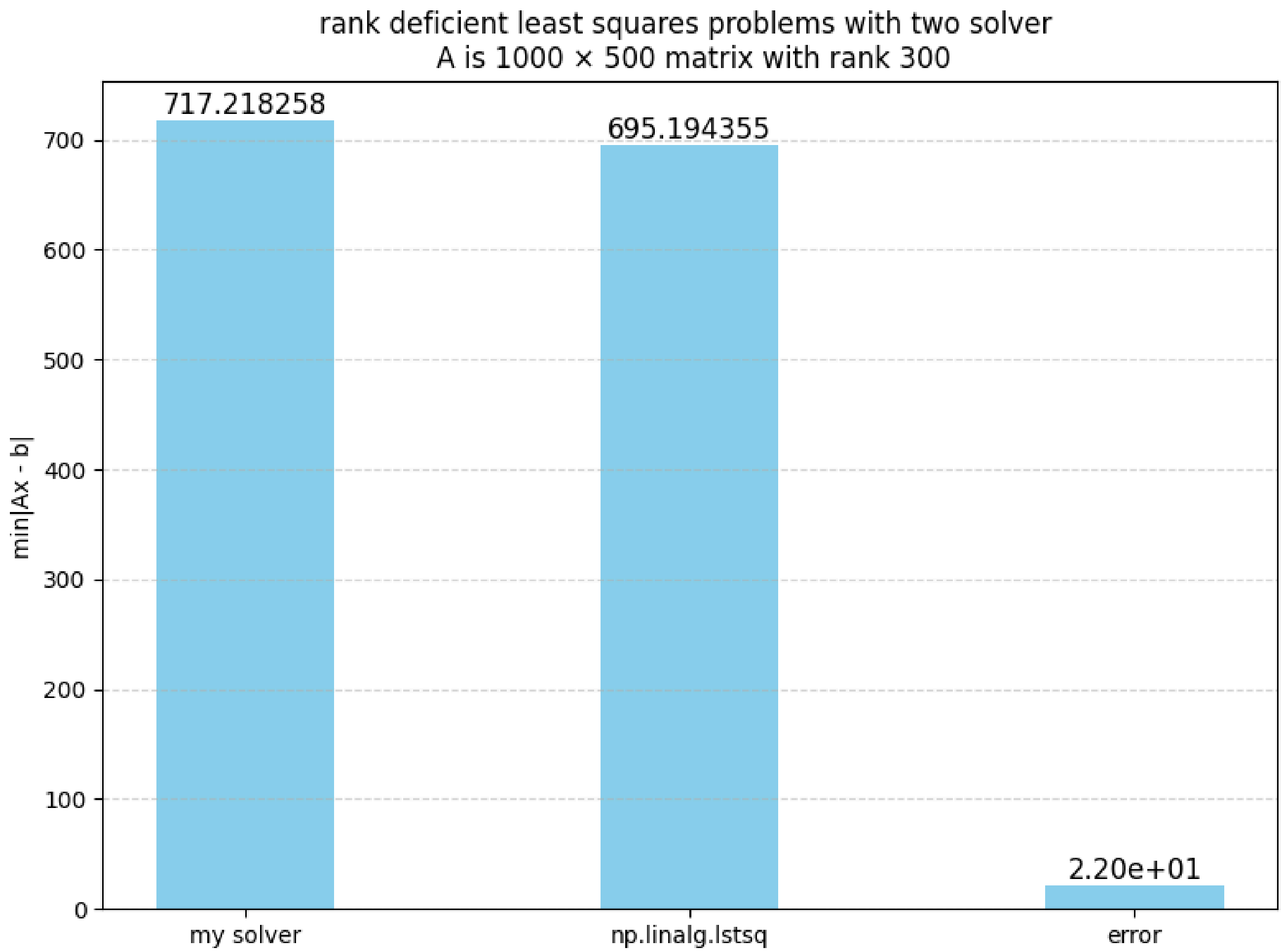
第四题

代码文件 **Arnoldi_precess.py**



第五题

代码文件 **rank_deficient_ls.py**



附录

```python
# part of Arnoldi_precess.py
def Arnoldi_precess(A, b, k, modified, reortho):
    """
    Arnoldi process for matrix A and vector v, using CGS/CGS2/MGS/MGS2 when orthogonaliza
    A[q_1, q_2, ... ,q_k] = [q_1, q_2, ..., q_(k+1)] H
    :param A: matrix A, n x n
    :param b: vector b, iterative initial vector
    :param k: number of iterations
    :param modified: use CGS/CGS2 if modified is not True else use MGS/MGS2
    :param reortho: use CGS/MGS if reortho is not True else use CGS2/MGS2
    :return: Q (n x (k+1) orthogonal matrix), H ((k+1) x k upper hessenberg matrix)
    """
    n = len(b)
    n1, n2 = A.shape
    if n1 != n or n2 != n:
        print(f'input matrix A and vector b have different size, A: {n1} x {n2}, b: {n} x
        return None, None

    Q = np.zeros((n, (k+1)))
    H = np.zeros(((k+1), k))
    Q[:, [0]] = b / np.linalg.norm(b, ord=2)

    if modified is not True:
    # Use BLAS2 may be faster, but here use BLAS1 for simplicity
        for i in range(k):
            cur = A @ Q[:, [i]]
            for j in range(i+1):
                H[i, j] = np.dot(cur.T, Q[:, [j]]).item()
            for j in range(i+1):
                cur = cur - H[i, j] * Q[:, [j]]
            if reortho is True:
                correct = [0] * (i+1)
                for j in range(i+1):
                    correct[j] = np.dot(cur.T, Q[:, [j]]).item()
                    H[i, j] = H[i, j] + correct[j]
                for j in range(i+1):
                    cur = cur - correct[j] * Q[:, [j]]
            H[i+1, i] = np.linalg.norm(cur, ord=2)

            if H[i+1, i] == 0:
                print(f'cannot continue iteration when generating q_{i+1}, H[{i+1}, {i}]
```

```
                return Q, H

            Q[:, [i+1]] = cur / H[i+1, i]

    if modified is True:
        for i in range(k):
            cur = A @ Q[:, [i]]
            for j in range(i+1):
                H[i, j] = np.dot(cur.T, Q[:, [j]]).item()
                cur = cur - H[i, j] * Q[:, [j]]
                if reortho is True:
                    correct = np.dot(cur.T, Q[:, [j]]).item()
                    H[i, j] = H[i, j] + correct
                    cur = cur - correct * Q[:, [j]]
            H[i+1, i] = np.linalg.norm(cur, ord=2)

            if H[i+1, i] == 0:
                print(f'cannot continue iteration when generating q_{i+1}, H[{i+1}, {i}]
                return Q, H

            Q[:, [i+1]] = cur / H[i+1, i]

    return Q, H
```

```python
# part of rank_deficient_ls_py
def qr_decomposition_with_pivoting(origin_A, tol=1e-10):
    """
    matrix A is a rank deficient matrix, return QR decomposition with column pivoting
    :param origin_A: matrix A, m x n, m >= n
    :return: Q (m x m, Q*Q = I), R, P (record column exchange)
    """
    A = np.copy(origin_A).astype(float)
    m, n = A.shape
    if m < n:
        print(f'warning, m={m} < n={n}')
        return None, None, None
    exchange = np.arange(n)
    Q = np.zeros((m, m))
    R = np.zeros((m, n))
    col_norms = np.sum(A**2, axis=0)
    rank = 0

    for i in range(n):
        pivot = np.argmax(col_norms[i:]) + i
        if col_norms[pivot] < tol:
            break
        if pivot != i:
            A[:, [i, pivot]] = A[:, [pivot, i]]
            exchange[i], exchange[pivot] = exchange[pivot], exchange[i]
            col_norms[i], col_norms[pivot] = col_norms[pivot], col_norms[i]

        R[i, i] = np.linalg.norm(A[:, [i]], ord=2)
        Q[:, [i]] = A[:, [i]] / R[i, i]
        R[i, i+1:] = Q[:, [i]].T @ A[:, i+1:]

        A[:, i+1:] = A[:, i+1:] - np.outer(Q[:, [i]], R[i, i+1:])
        col_norms[i+1:] = col_norms[i+1:] - R[i, i+1:]**2
        col_norms[col_norms < tol] = 0

        rank = rank + 1

    # make Q orthogonalized square matrix

    for j in range(m):
        if rank == m:
            break
        e = np.zeros(m, dtype=float)
```

```
        e[j] = 1.0

        for k in range(rank):
            projection = np.dot(e, Q[:, k])
            e = e - projection * Q[:, k]
        norm_e = np.linalg.norm(e, ord=2)
        if norm_e > tol:
            Q[:, rank] = e / norm_e
            rank += 1

    P = np.zeros((n, n), dtype=int)
    for i in range(n):
        P[i, exchange[i]] = 1

    return Q, R, P

# now min|Ax - b| -> min|QRPx - b| -> min|RPx - Q^T b| -> min|[R1 0]^T y - [c1 c2]^T|, in
# so min|Ax - b| = min|R1^T y - c1| + |c2| = |c2|, thanks to R1's rank <= n

def ls_for_rank_deficient_matrix(A, b, tol=1e-10):
    Q, R, P = qr_decomposition_with_pivoting(A, tol)
    c = Q.T @ b
    rank_R = np.linalg.matrix_rank(R)
    c2 = c[rank_R:]
    norm_c2 = np.linalg.norm(c2, ord=2)
    return norm_c2 ** 2
```