Miniproject 1

Linus Falk

October 6, 2022

1 introduction and method

The aim of this miniproject is to implement pivoting in the QR factorization algorithm that was constructed in a previous assignment. This will make it possible to solve the least square problem. $\min\limits_{x\in\mathbb{R}^r} \|Ax - b\|_2$.

1.1 QR factorization with Givens rotation

QR factorization algorithm can be constructed with use of Givens rotations.By applying the rotation matrix G constructed with help of equation(1) multiplied from the left with the original matrix we can construct an orthogonal matrix Q from the product of the rotation matrices and upper triangular matrix R. The matrix G(i,j,θ) only affect row i and j of matrix A and matrix G is constructed by multiple Givens rotation matrices $Q = G_1^T G_2^T \dots G_n$. An example of the first step formation of R i given below: [1]

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \begin{bmatrix} 6 & 5 & 0 \\ 5 & 1 & 4 \\ 0 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 7.8102 & 4.4814 & 2.5607 \\ 0 & -2.4327 & 3.0729 \\ 0 & 4 & 3 \end{bmatrix} \tag{2}$$

The implementation in a Python script is presented here:

```python
def grot(a,b):
    if abs(a) > abs(b):
        t = b/a; c = 1/np.sqrt(1+t**2); s = c*t
    else:
        t = a/b; s = 1/np.sqrt(1+t**2); c = s*t
    return np.array([c, s])


def G(R, c, s, i):
    R[i][i] = s
    R[i][i - 1] = c
    R[i - 1][i] = -c
    R[i - 1][i - 1] = s
    return R
```

By constructing two nested for loops with this function we can construct the algorithm for QR factorization of a matrix A:

```
def qrfac(A, mode='Q&R'):
Qsave = np.eye(m)
R = A
    for j in range(1, n + 1, 1):
        for i in range(m, j, -1):
            Q = np.eye(m)
            [a, b] = grot(R[i - 1][j - 1],
                          R[i - 2][j - 1])
            R = np.transpose(G(Q, a, b, i - 1)) @ R
            Qsave = Qsave @ Q
            print(Qsave@R)
    return np.triu(R), Qsave[0:m][0:m]
```

1.2 QR factorization with pivoting

The next step before using QR factorization for least square method is to add pivoting to the algorithm. This in each iteration: n of QR steps shifts the column n-1 with the column of number $> n-1$ with the highest L_2 norm. This method moves the zero pivots to lower right corner of the R matrix, making it useful for solving rank-deficient least square problems. The swapping of the column can be done by adding function for moving the columns to the QR algorithm. This functions creates permutations matrices for each step such that the move of the previous column is not affected [2].

```
def permutationMatrix(A, k):
    m, n = np.shape(A)
    P = np.eye(n)
    buff = A[k:, k:]
    m, n = np.shape(buff)
    lst = []
    for x in range(0, n, 1):
        lst.append(npl.norm(buff[:, x]))
    index = np.argmax(np.array(lst))+k
    Swap(P, k, index)
    return P
```

permutation matrices are multiplied from the right and together with each new Q_n matrix we get:

$$Q_nQ_{n-1}\dots Q_1AP_1P_2\dots P_n = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \tag{3}$$

The resulting R matrix consists of R_{11} that is a upper triangular $r \times r$ matrix. We can from this derive the rank of the matrix. An implementation of this is presented in Python code on the next page:

```
def QRcPivot(A):
    rank = 0
    m, n = np.shape(A)
    Qsave = np.eye(m)
    Psave = np.eye(n)
    R = A
    for j in range(1, n + 1, 1):
        P = permutationMatrix(R, j-1)
        R = R@P
        Psave = Psave@P
        for i in range(m, j, -1):
            Q = np.eye(m)
            a, b = grot(R[i - 1][j - 1],
                        R[i - 2][j - 1])
            R = np.transpose(G(Q, a, b, i - 1)) @ R
            Qsave = Qsave @ Q
    for i in range(n):
        rank = i
        if R[i-1,i-1] == 0:
            break
    return R, Qsave[0:m][0:m], Psave, rank
```

1.3 Least square method

The least square problem: $\min_{x\in\mathbb{R}^r} \|Ax - b\|_2$ can now be solved by the use of the obtained QR decomposition and rank after QR factorization with pivoting. We can obtain the solution vector x by slicing Q and R matrices with the rank. We obtain the solution, more details and derivation of the expression can be found in [2]. The solution of the least square problem can be implemented in Python described below:

```
def LeastSquaresQRcPivot(A, b):
    r, q, p, rank = QRcPivot(A)
    Q_1 = q[:, 0:rank]
    R_11 = r[0:rank, 0:rank]
    result = np.zeros(np.shape(A)[1])
    result[:rank] = inv(R_11) @ Q_1.T @ b
    result = p @ result
    return result, norm(A @ result - b)
```

2 Result

In this section are the results from testing the constructed algorithms presented

2.1 QRcPivot

The QR factorization with pivoting algorithm was tested with randomly generated matrices and the norm of $\|AP - QR\|_2$ was calculated to check the result:

```
A=np.random.random([50,30])
r, q, p, rank = QRcPivot(A)
print(round(np.linalg.norm((A @ p - q @ r),2), 6))

OUTPUT: 0.0

A=np.random.random([50,50])
r, q, p, rank = QRcPivot(A)
print(round(np.linalg.norm((A @ p - q @ r),2), 6))

OUTPUT: 0.0
```

2.2 LeastSquaresQRcPivot

The least square problems was given in the assignment. Here is the results presented

```
A = np.array([[1, -1, 2, 0],
              [1, 2, -1, 3],
              [1, 1, 0, 2],
              [1, -1, 2, 0],
              [1, 3, -1, 4]])

b = np.array([1, -1, 0, 1, 0])
result, norm = LeastSquaresQRcPivot(A, b)

OUTPUT:
[ 0.          1.92592593  1.48148148 -1.07407407]
0.2721655269759087

#checking result with numpy linalg library
print(linalg.lstsq(A,b))

OUTPUT:
[-1.        ,  0.92592593,  1.48148148, -0.07407407]

n = 100
B = rogues.neumann(n)
A = B[0]
b = np.zeros(n)
LeastSquaresQRcPivot(A, b)

OUTPUT:
ValueError: The truth value of an array with more than
one element is ambiguous. Use a.any() or a.all()
```

3 Discussion and conclusion

When calculating the norm of random matrices the QR factorization pivot works as expected. For solving the least square problem there seems to be missing one last step. The results are very similar with just -1 difference between some of the results. when unpacking the matrix in the Givens rotation part. Testing the QRcPivot on matrices of the same size don't give the same problem and seems therefore to be syntax related.

References

[1] Wikipedia. *Givens rotation*, 2022. https://en.wikipedia.org/wiki/Givens$_{r}otation$.