hw3_2023

November 20, 2023

1 Homework set 3

Please **submit this Jupyter notebook through Canvas** no later than **Mon Nov. 20, 9:00**. **Submit the notebook file with your answers (as .ipynb file) and a pdf printout. The pdf version can be used by the teachers to provide feedback. A pdf version can be made using the save and export option in the Jupyter Lab file menu.**

Homework is in **groups of two**, and you are expected to hand in original work. Work that is copied from another group will not be accepted.

2 Exercise 0

Write down the names + student ID of the people in your group.

- Amir Sahrani (12661651)
- Jonas Schäfer (14039982)

Run the following cell to import NumPy and Pyplot.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

3 Exercise 1

In this exercise you will study the accuracy of several methods for computing the QR decomposition. You are asked to implement these methods yourself. (However, when testing your implementation you may compare with an external implementation.)

3.1 (a)

Implement the classical and modified Gram-Schmidt procedures for computing the QR decomposition.

Include a short documentation using triple quotes: describe at least the input and the output, and whether the code modifies the input matrix.

[ ]:
```
def classical_gs_qr(A):
    """
    Classical Gram-Schmidt algorithm for QR decomposition, not modifying the␣
 ↪input matrix
```

```
    :param A: an m x n matrix with linearly independent columns
    :return: (Q, R) where Q is an m x n matrix with orthonormal columns and R␣
 ↪is an upper triangular matrix such that Q@R  A
    """
    A_copy = A.copy()
    m, n = A.shape
    Q = np.zeros((m,n))
    R = np.zeros((m,n))

    for k in range(n):
        Q[:,k] = A[:,k]
        for j in range(k):
            R[j,k] = Q[:,j].T @ A[:,k]
            Q[:,k] -= R[j,k] * Q[:,j]
        R[k,k] = np.linalg.norm(Q[:,k])
        if R[k,k] == 0:
            raise ValueError("The columns of A are linearly dependent.")
        Q[:,k] /= R[k,k]

    assert (A==A_copy).all(), "The input matrix was modified."
    return Q,R


def modified_gs_qr(A):
    """
    Modified Gram-Schmidt algorithm, not modifying the input matrix
    :param A: an m x n matrix with linearly independent columns
    :return: (Q, R) where Q is an m x n matrix with orthonormal columns and R␣
 ↪is an m x n upper triangular matrix such that Q@R  A
    """
    A_copy = A.copy()
    m, n = A.shape
    Q = np.zeros((m, n))
    R = np.zeros((m, n))
    V =  A.copy()

    for j in range(n):
        R[j,j] = np.linalg.norm(V[:,j])
        Q[:,j] = V[:,j] / R[j,j]
        if R[j,j] == 0:
            raise ValueError("The columns of A are linearly dependent.")

        for k in range(j+1,n):
            R[j, k] = Q[:, j] @ V[:, k]
            V[:, k] -= R[j, k] * Q[:, j]

    assert (A==A_copy).all(), "The input matrix was modified."
```

```
    return Q,R
```

3.2 (b) (a+b 3.5 pts)

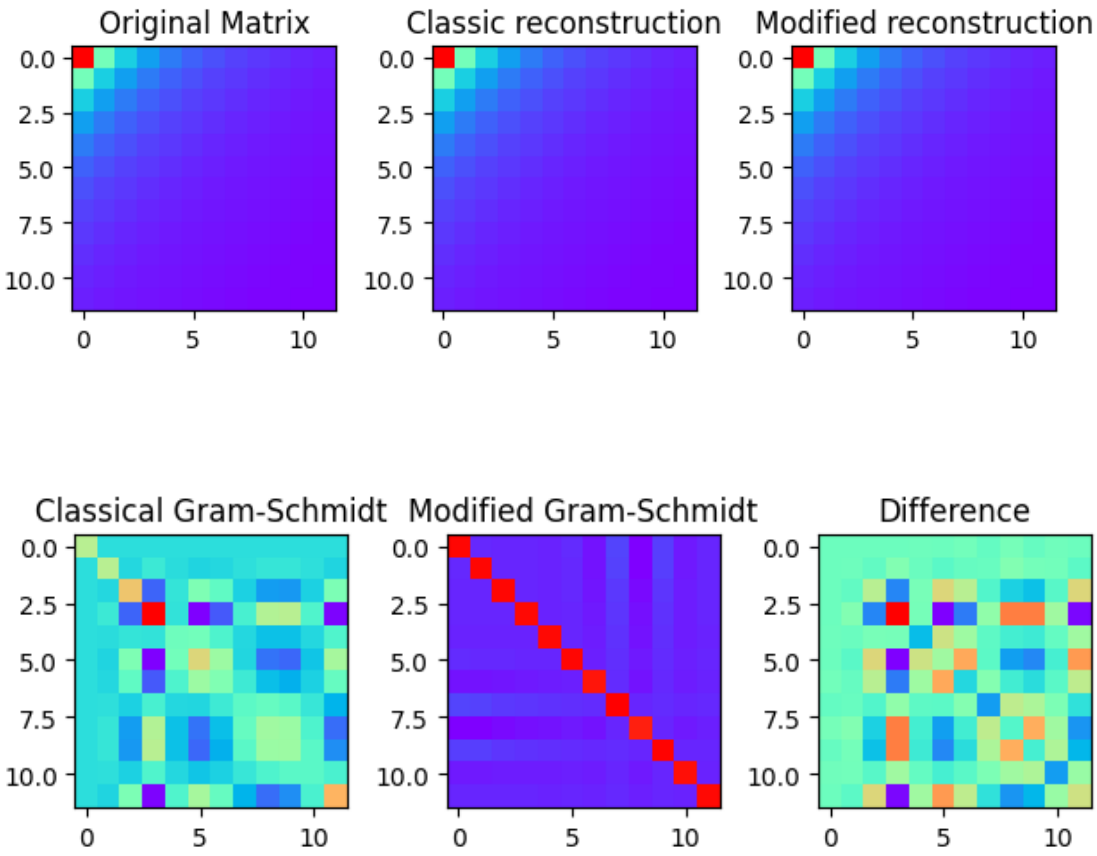
Let H be a Hilbert matrix of size n (see Computer Problem 2.6). Study the quality of the QR decompositions obtained using the two methods of part (a), specifically the loss of orthogonality. In order to do so, plot the quantity $\|I - Q^T Q\|$ as a function of n on a log scale. Vary n from 2 to 12.

[ ]:
```
def HilbertMatrix(n):
    i,j = np.indices((n,n))
    return 1/(i+j+1)

def test_QR(A, Q, R, rtol=1e-4):
    assert np.count_nonzero(np.isclose(A, Q@R, rtol=rtol)) == A.shape[0] * A.
 ↪shape[1], f"QR decomposition is not accurate within error tolerance {rtol}␣
 ↪specified."

def test_orthogonal(Q, rtol=1e-4):
    assert np.count_nonzero(np.isclose(Q.T@ Q, np.eye(*Q.shape), rtol=rtol)) ==␣
 ↪Q.shape[0] * Q.shape[1], f"Q is not orthogonal."

def test_qr_decomp_func(f, n=6, rtol=1e-4):
    hilbert = HilbertMatrix(n)
    Q, R = f(hilbert)
    test_QR(hilbert, Q, R, rtol=rtol)
    test_orthogonal(Q, rtol=rtol)
    print(f'QR decomposition {f.__name__} is accurate to error tolerance of␣
 ↪{rtol}.')

def loss_of_orthogonality(QRs):
    return [np.linalg.norm(np.eye(*Q.shape) - Q.T@Q) for Q, _ in QRs]

def compare_loss_qr_decomp_methods(fs, n_space=(2, 13)):
    QR_pairs = [[] for _ in range(len(fs))]
    n_low, n_high = n_space
    n_space = np.arange(n_low, n_high, 1)
    for n in n_space:
        hilbert = HilbertMatrix(n)
        for i, f in enumerate(fs):
            Q, R = f(hilbert)
            QR_pairs[i].append((Q, R))

    loss_of_orthogonalities = [loss_of_orthogonality(QRs) for QRs in QR_pairs]
    return n_space, loss_of_orthogonalities
```

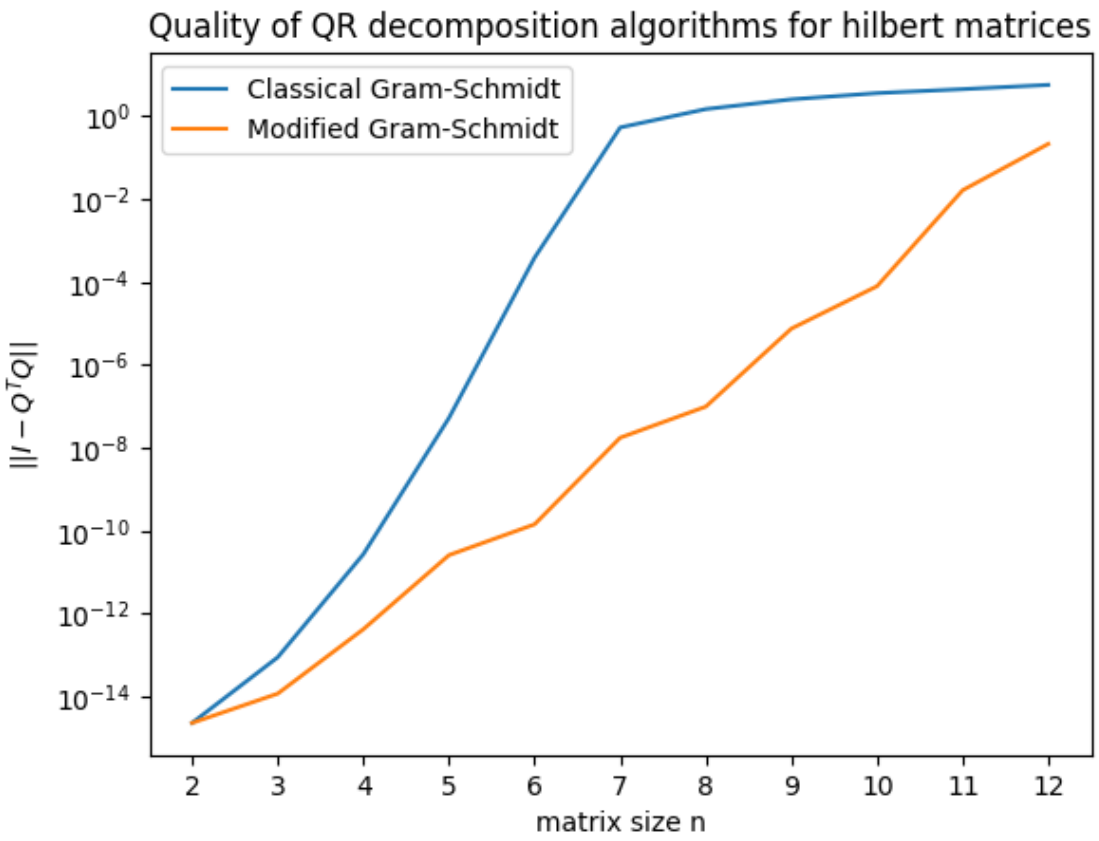
We can see both methods are able to generate a QR decomposition of the matrix that can be used to reconstruct the matrix. Looking at the orthogonal property of the $Q^T Q$ matrices produces

by both however, we see a stark contrast. The numerical error of the classical approach is a lot larger, in the matrix plot we can see the Q matrix produced by the classical approach loses a lot of orthogonality and in the line graph we can clearly see a bigger difference between the Idenity matrix and Q^TQ. The fact that the numerical error gets propagated in the classical approach can be seen in the difference between the modified approach and the classical one, most of the difference can be found after the first two columns and rows.

[ ]:
```
H = HilbertMatrix(12)
Q,R = classical_gs_qr(H)
mQ, mR = modified_gs_qr(H)
recon = mQ @ mR

cmap = 'rainbow'

plt.subplot(1, 3, 1)
plt.imshow(H, cmap=cmap)
plt.title("Original Matrix")

plt.subplot(1, 3, 2)
plt.imshow(Q@R , cmap=cmap)
plt.title("Classic reconstruction")

plt.subplot(1, 3, 3)
plt.imshow(recon, cmap=cmap)
plt.title("Modified reconstruction")
plt.tight_layout()
plt.show()

plt.subplot(1, 3, 1)
plt.imshow(Q @ Q.T, cmap=cmap)
plt.title("Classical Gram-Schmidt")
plt.subplot(1, 3, 2)
plt.imshow(mQ @ mQ.T, cmap=cmap)
plt.title("Modified Gram-Schmidt")
plt.subplot(1, 3, 3)
plt.imshow(Q @ Q.T - mQ @ mQ.T, cmap=cmap)
plt.title("Difference")
plt.tight_layout()
plt.show()
```



[ ]:
```
n_space, losses = compare_loss_qr_decomp_methods(fs=(classical_gs_qr,␣
  ↪modified_gs_qr))
loss_classic, loss_modified = losses[0], losses[1]

plt.plot(n_space, loss_classic, label='Classical Gram-Schmidt')
plt.plot(n_space, loss_modified, label='Modified Gram-Schmidt')
plt.xlabel('matrix size n')
plt.xticks(n_space)
plt.ylabel('$||I - Q^TQ||$')
plt.yscale('log')
plt.title('Quality of QR decomposition algorithms for hilbert matrices')
plt.legend()
plt.show()
```

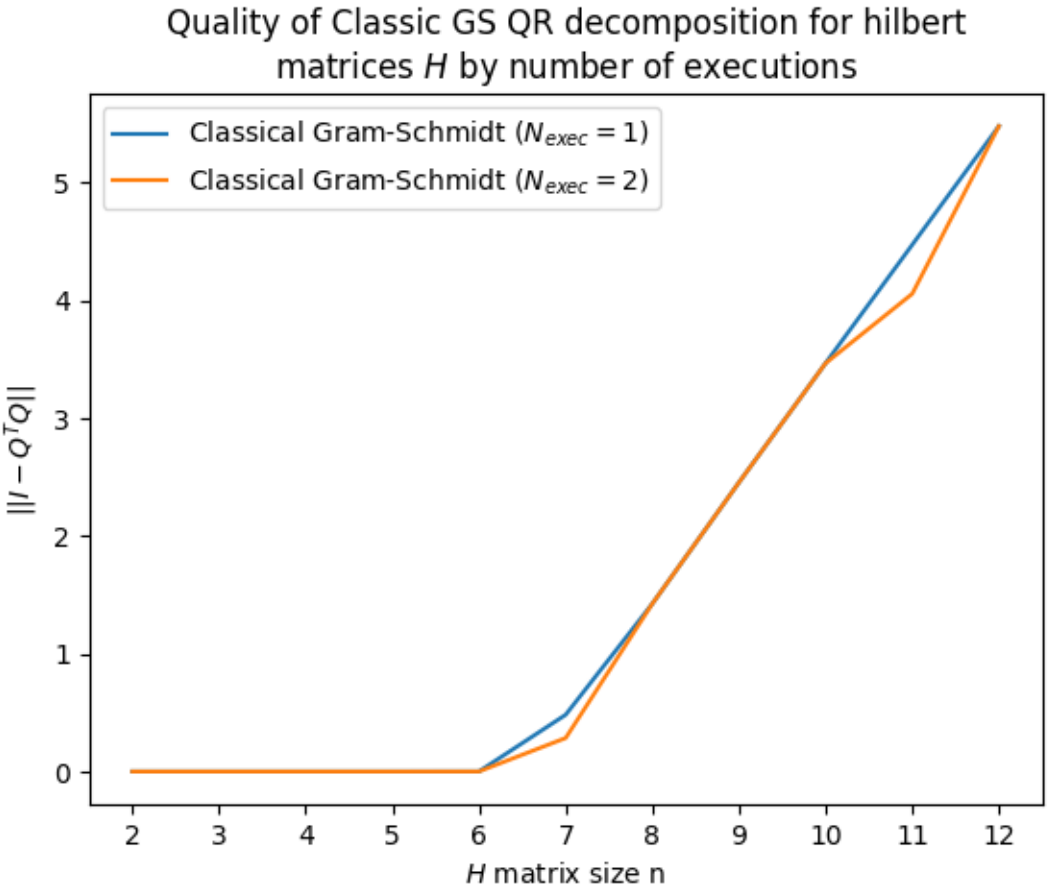


3.3 (c) (1.5 pts)

Try applying the classical procedure twice. Plot again the loss of orthogonality when computing the QR decomposition of the Hilbert matrix of size n as in (b).

[ ]:
```
def classical_gram_schmidt_twice(A):
    Q, R = classical_gs_qr(A)
    Q, R = classical_gs_qr(Q@R)
    return Q, R

n_space, losses = compare_loss_qr_decomp_methods(n_space=(2, 13), fs =␣
 ↪(classical_gs_qr, classical_gram_schmidt_twice))
loss_classic, loss_classic_twice = losses[0], losses[1]

plt.plot(n_space, loss_classic, label='Classical Gram-Schmidt ($N_{{exec}}=1$)')
plt.plot(n_space, loss_classic_twice, label='Classical Gram-Schmidt␣
 ↪($N_{{exec}}=2$)')
plt.xlabel('$H$ matrix size n')
plt.xticks(n_space)
plt.ylabel('$||I - Q^TQ||$')
```

```
# plt.yscale('log')
plt.title('Quality of Classic GS QR decomposition for hilbert\nmatrices $H$ by␣
 ↪number of executions')
plt.legend()
plt.show()
```



We see little difference between the applying the classical procedure twice and applying it once. This might be because the reconstruction is accurate enough that the numerical errors that occur stay of the same order of magnitude. We have not done any experiments to verify this.

3.4 (d) (2 pts)

Implement the Householder method for computing the QR decomposition. Remember to include a short documentation.

[ ]:
```
def householder_qr(A):
    """
    Householder QR decomposition, not modifying the input matrix
    :param A: an m x n matrix with linearly independent columns
```

```
    :return: (Q, R) where Q is an m x n matrix with orthonormal columns and R␣
 ↪is an m x n upper triangular matrix such that Q@R   A
    """
    A_copy = A.copy()
    m, n = A.shape
    Q = np.eye(m)
    R = A.copy()

    for j in range(min(m-1, n)):
        x = R[j:, j]
        v = x.copy()
        v[0] = v[0] + np.sign(x[0]) * np.linalg.norm(x)
        v = v / np.linalg.norm(v)
        total_1 = 2 * np.outer(v, np.dot(v, R[j:, j:]))
        total_2 = 2 * np.outer(Q[:, j:].dot(v), v)
        R[j:, j:] = R[j:, j:] - total_1
        Q[:, j:] = Q[:, j:] - total_2
    assert (A==A_copy).all(), "The input matrix was modified."
    return Q, R

test_qr_decomp_func(householder_qr)
```

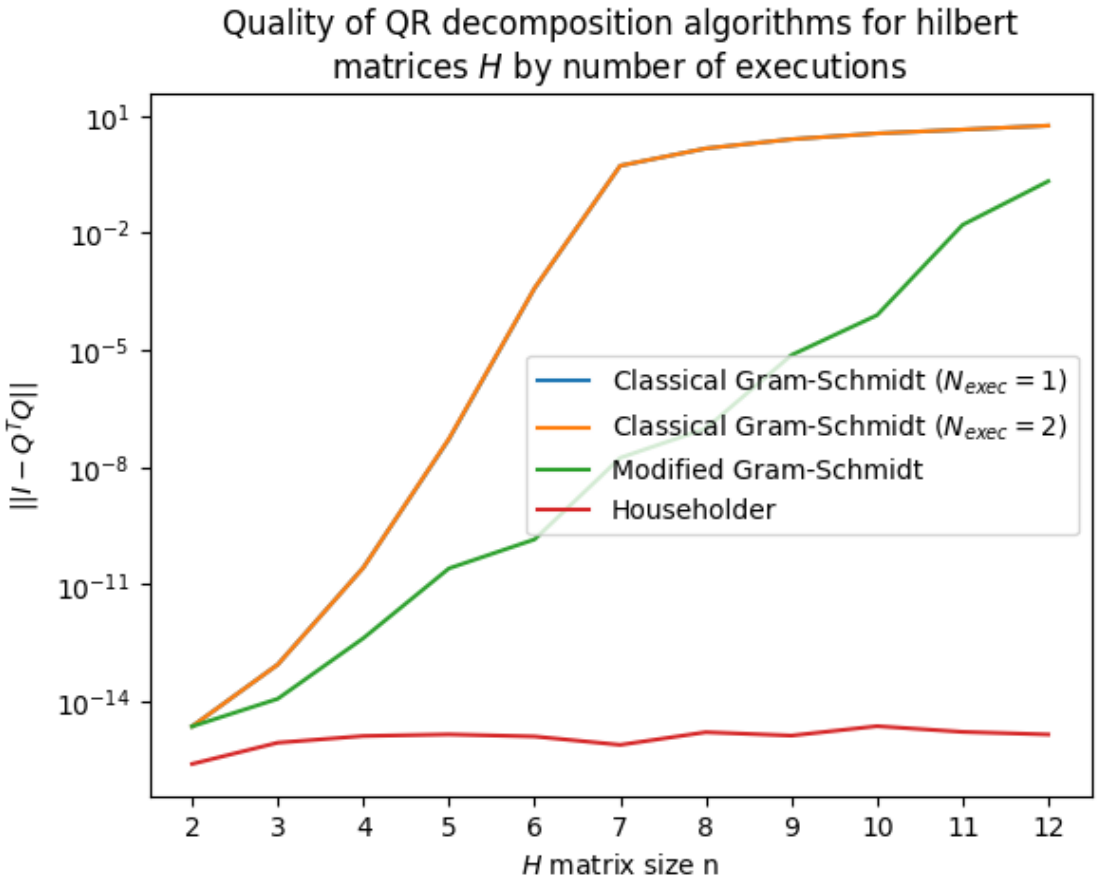
```
QR decomposition householder_qr is accurate to error tolerance of 0.0001.
```

3.5 (e) (2 pts)

Perform the analysis of (b) for the Householder method. Discuss the differences between all the methods you have tested so far. Look online and/or in books for information about the accuracy of the different methods and include this in your explanations (with reference).

[ ]:
```
n_space, losses = compare_loss_qr_decomp_methods(n_space=(2, 13), fs =␣
 ↪(classical_gs_qr, classical_gram_schmidt_twice, modified_gs_qr,␣
 ↪householder_qr))
loss_classic, loss_classic_twice, loss_modified, loss_householder = losses[0],␣
 ↪losses[1], losses[2], losses[3]

plt.plot(n_space, loss_classic, label='Classical Gram-Schmidt ($N_{{exec}}=1$)')
plt.plot(n_space, loss_classic_twice, label='Classical Gram-Schmidt␣
 ↪($N_{{exec}}=2$)')
plt.plot(n_space, loss_modified, label='Modified Gram-Schmidt')
plt.plot(n_space, loss_householder, label='Householder')
plt.xlabel('$H$ matrix size n')
plt.xticks(n_space)
plt.ylabel('$||I - Q^TQ||$')
plt.yscale('log')
plt.title('Quality of QR decomposition algorithms for hilbert\nmatrices $H$ by␣
 ↪number of executions')
```

```
plt.legend()
plt.show()
```



A big reason as to why the regular Gram-Schmidt method for decomposition is unstable is because the calculations from the previous column in Q are used for the computation in the subsequent columns this means any numerical errors and rounding errors get propagated through out the calculation. The modified version of the GS method does not depend on the previous computations and therefore does not propagate these errors.

Although MGS prevents the errors that occurred during the computation of one of the vectors form propagating to the next one. It cannot deal with error that occur within the computation of one vector. Householder reflections can sovle this issue, instead of calculating the vector by subtracting the projections, it calculates the distance between the reflections and subtracts those once, thereby reducing oportunities for numberical error, because we simply have fewer operations and we are using operations that are less prone to error.

Householder reflections has a draw back however, it has higher memory requirements, and is less simple to implement than the classical Gram-Schmidt method.

These conclusions were based on this article by UCI