Homework set 3

Please **submit this Jupyter notebook through Canvas** no later than **Mon Nov. 21, 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.

Exercise 0

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

Marcel van de Lagemaat - 10886699
Anton Andersen - 14718758

Run the following cell to import NumPy and Pyplot.

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

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.)

(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.

In [ ]:

```
def classical_gram_schmidt(A):
    """ INPUT: Takes an mxn array.
    OUTPUT: Returns the qr factorization with decomposed
orthogonal matrix q
    and triangular matrix r.
    Computes the classical Gram-Schmidt qr factorization of the
array A.
    The function orthonormalizes the vector with respect to
each other. """
    q, r = np.zeros(A.shape), np.zeros(A.shape)
    for k in range(A.shape[1]):
        q_k = A[:,k]

        for i in range(k):
            r[i,k] = q[:,i] @ A[:,k]
            q_k = q_k - q[:,i] * r[i,k]
        r[k,k] = np.linalg.norm(q_k, 2)

        if r[k,k] == 0:
            break

        q_k = q_k / r[k,k]
        q[:,k] = np.transpose(q_k)
    return q, r


def modified_gram_schmidt(A):
    """ INPUT: Takes an mxn array.
    OUTPUT: Returns the qr factorization with decomposed
orthogonal matrix q
    and triangular matrix r.
    Computes the Modified Gram-Schmidt qr factorization of the
array A.
    The function orthonormalizes the vector with respect to
each other. """
    col = A.shape[0]
    r = np.zeros([A.shape[1],A.shape[1]])
    for k in range(A.shape[1]):
        r[k,k] = np.linalg.norm(A[:,k], 2)
        if r[k,k] == 0:
            break
```

```
        A[:,k] = A[:,k] / r[k,k]
        for i in range(k+1, A.shape[1]):
            r[k,i] = A[:,k].T @ A[:,i]
            A[:,i] = A[:,i] - r[k,i] * A[:,k]
    return A, r
```

(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.

In [ ]:

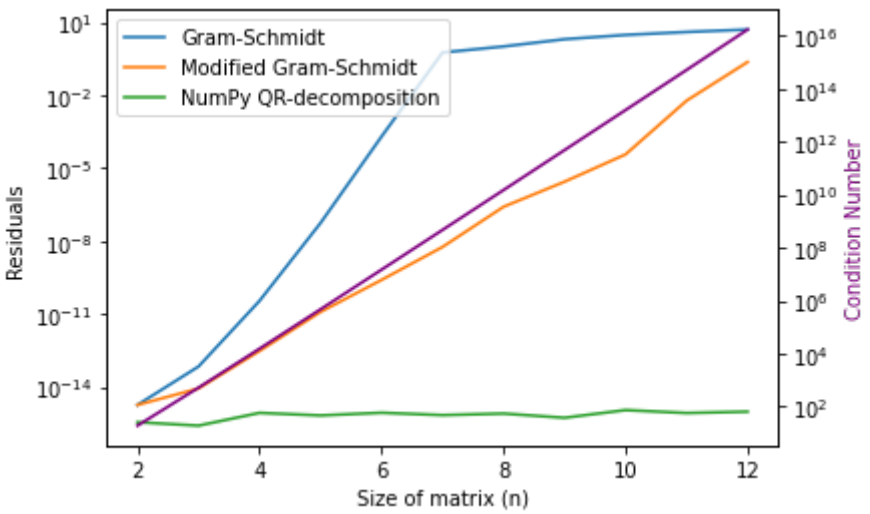
```
from scipy.linalg import hilbert

def residual(func, twice=False):
    q_list = [func(hilbert(n))[0] for n in range(2,13)]
    if twice:
        q_list = [func(q)[0] for q in q_list]
    return [np.linalg.norm(np.eye(len(q))-q.T @ q, 2) for q in
q_list]

hilbert_gs = residual(classical_gram_schmidt)
hilbert_mod_gs = residual(modified_gram_schmidt)
hilbert_np  = residual(np.linalg.qr)

hilbert_cond = [np.linalg.cond(hilbert(n)) for n in
range(2,13)]

fig, ax = plt.subplots()
ax.semilogy(range(2,13), hilbert_gs, label = 'Gram-Schmidt')
ax.semilogy(range(2,13), hilbert_mod_gs, label = 'Modified
Gram-Schmidt')
ax.semilogy(range(2,13), hilbert_np, label = 'NumPy QR-
decomposition')
ax.set_xlabel('Size of matrix (n)')
ax.set_ylabel('Residuals')
ax.legend()
ax2 = ax.twinx()
ax2.semilogy(range(2,13), hilbert_cond, color='purple')
ax2.set_ylabel('Condition Number', color= 'purple')
plt.show()
```

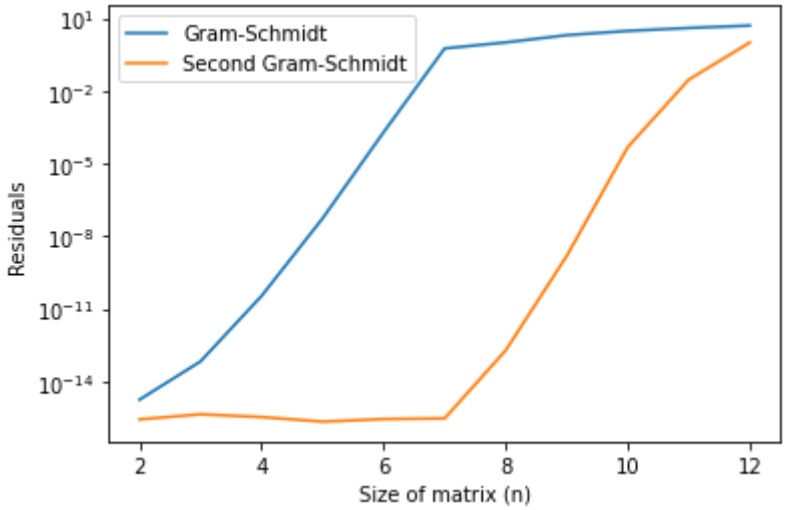


(c) (1.5 pts)

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).

In [ ]:
```
hilbert_second = residual(classical_gram_schmidt, twice=True)

plt.semilogy(range(2,13), hilbert_gs, label = 'Gram-Schmidt')
plt.semilogy(range(2,13), hilbert_second, label = 'Second Gram-Schmidt')
plt.xlabel('Size of matrix (n)')
plt.ylabel('Residuals')
plt.legend()
plt.show()
```



(d) (2 pts)

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

In [ ]:

```
def householder_qr(A):
    ''' Compute QR decomposition of A using Householder
reflections
        Arguments:
        A: mxn array (matrix)

        Returns:
        Q: mxm Orthogonal matrix, product of all Householder
transformations
            (H_1 .. H_n)
        R: nxn Upper triangular matrix
    '''
    rows, cols = A.shape
    Q = np.eye(rows)
    R = A.copy()

    for k in range(min(cols, rows-1)):
        # Compute Householder vector
        alpha_k = -np.sign(R[k,k]) * np.linalg.norm(R[k:,k], 2)
        col_done = range(rows - len(R[k:,k]))
        t = np.array([0 for _ in col_done] + list(R[k:,k]))
        e_k = np.eye(rows)[:,k]
        v_k =  t.T - alpha_k * e_k

        # Skip column if it is already 0
        beta_k = np.dot(v_k.T, v_k)
        if beta_k == 0:
            break

        # Compute Householder matrix H_k
        v = np.array([len(v_k) * [V] for V in v_k])
        vvT =  v @ v.T
        H = np.eye(rows) - 2 * (vvT / np.linalg.norm(vvT,2))
        Q = Q @ H

        # Apply transformation to matrix
        for j in range(k, cols):
            gamma_j = v_k.T @ R[:,j]
            t = (2 * gamma_j / beta_k) * v_k
            R[:,j] = R[:,j] - t
```

```
    return Q, R
```

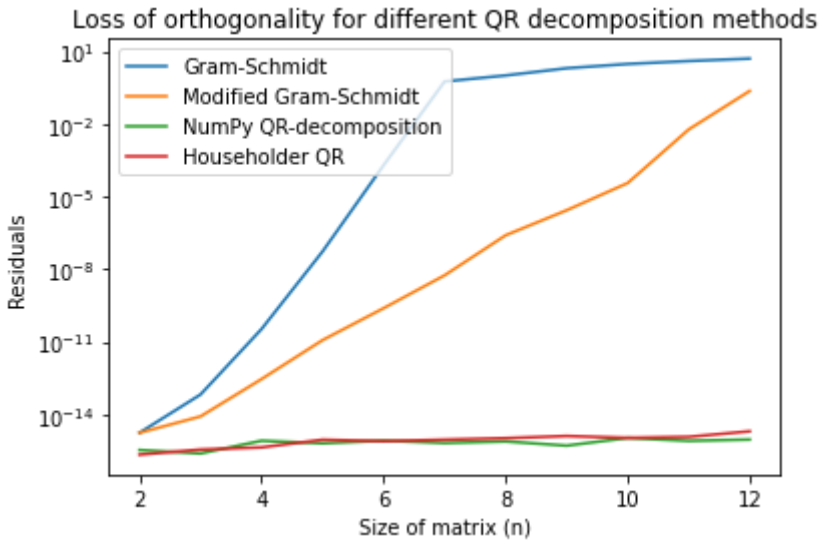
(e) (2 pts)

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).

In [ ]:

```
hilbert_house = residual(householder_qr)

plt.semilogy(range(2,13), hilbert_gs, label = 'Gram-Schmidt')
plt.semilogy(range(2,13), hilbert_mod_gs, label = 'Modified
Gram-Schmidt')
plt.semilogy(range(2,13), hilbert_np, label = 'NumPy QR-
decomposition')
plt.semilogy(range(2,13), hilbert_house, label = 'Householder
QR')
plt.title('Loss of orthogonality for different QR decomposition
methods')
plt.xlabel('Size of matrix (n)')
plt.ylabel('Residuals')
plt.legend()
plt.show()
```



In the graph above we can see the residuals for different QR decomposition methods, showing the loss of orthogonality in Hilbert matrices of size n. We implemented classical and modified Gram-Schmidt (GS/MGS) and Householder transformations. As a baseline we also show the QR decomposition from the numpy package. What algorithm was used for this function is not part of the documentation (however this is most likely a Householder transformation).
The loss of orthogonality is due to the use of finite precision of using machines for calculation. The loss of orthogonality then occurs due to rounding off- and cancellation errors, that may be amplified in later steps, as noted by Giraud et al. (2005). Golub and Van Loan (2013), with reference to Bjorck (1967), show that the loss of orthogonality for housholder transformations is less than for Gram-Schmidt. GS is shown to be proportional to the condition number of A.
From this graph we can conclude that Householder reflections are more numerically stable than Gram-Schmidt variants since the former has smaller residuals for all sizes n of the Hilbert matrix. In the paper, Shao (2011) finds similar results, that GS and MGS lead to more loss of orthogonality than Householder reflectors, indicating that the Householder reflectors are numerically more stable.

1. Shao, M. (2021). Householder orthogonalization with a non-standard inner product. arXiv preprint arXiv:2104.04180.
2. Giraud, L., Langou, J., & Rozloznik, M. (2005). The loss of orthogonality in the Gram-Schmidt orthogonalization process. Computers & Mathematics with Applications, 50(7), 1069-1075.
3. Golub, G. H., & Van Loan, C. F. (2013). Matrix computations. JHU press.
4. Björck, Å. (1967). Solving linear least squares problems by Gram-Schmidt orthogonalization. BIT Numerical Mathematics, 7(1), 1-21.

In [ ]: