Numerical Linear Algebra: Exam #1

Due on October 1, 2022 at 5:00PM

Instructor: Professor Blake Barker
Section 1

Michael Snyder

Problem 1

Let $A \in \mathbb{R}^{2\times 2}$ be a matrix that satisfies

$$\sup_{\|x\|_2=1} \|Ax\|_2 = 3, \qquad \inf_{\|x\|_2=1} \|Ax\|_2 = 2.$$

What are the singular values of A?

Solution:

Proof. By Theorem 4.1, A has an SVD. Let it be represented in the standard way, $A = U\Sigma V^*$. By Theorem 3.1, for any $A \in \mathbb{C}^{m\times n}$ and unitary $Q \in \mathbb{C}^{m\times m}$, $\|QA\|_2 = \|A\|_2$. Thus, $\|A\|_2 = \|U\Sigma V^*\|_2 = \|\Sigma\|_2$.
Using these relationships we find that

$$\dagger \quad 3 = \sup_{\|x\|_2=1} \|Ax\|_2 = \sup_{\|x\|_2=1} \|\Sigma x\|_2 = \sup_{\|x\|_2=1} \sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2}.$$

By convention, singular values are arranged such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. Hence the quantity $\sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2}$ is maximized for $x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. But this means, completing the equalities in †, we have

$$\sup_{\|x\|_2=1} \sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2} = \sqrt{(\sigma_1 \cdot 1)^2 + (\sigma_2 \cdot 0)^2} = \sigma_1,$$

and thus, $\sigma_1 = 3$.

By similar logic, we have that $\sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2}$ is minimized for $x = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and thus,

$$2 = \inf_{\|x\|_2=1} \|Ax\|_2 = \inf_{\|x\|_2=1} \|\Sigma x\|_2 = \inf_{\|x\|_2=1} \sqrt{(\sigma_1 x_1)^2 + (\sigma_2 x_2)^2} = \sqrt{(\sigma_1 \cdot 0)^2 + (\sigma_2 \cdot 1)^2} = \sigma_2.$$

Thus, the singular values of A are $\sigma_1 = 3$ and $\sigma_2 = 2$. □

Problem 2

Suppose $A \in \mathbb{C}^{m\times m}$ has an SVD $A = U\Sigma V^*$. Find an eigenvalue decomposition (5.1) of the $2m \times 2m$ hermitian matrix

$$\begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix}.$$

Solution:

Proof. I can't seem to figure this one out... I've tried a few things. The thing that seemed the most promising was observing that if $B = \begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix}$, then

$$B^*B = BB^* = BB = \begin{bmatrix} V\Sigma^2V^* & 0 \\ 0 & U\Sigma^2U^* \end{bmatrix} = \begin{bmatrix} V & 0 \\ 0 & U \end{bmatrix} \begin{bmatrix} \Sigma^2 & 0 \\ 0 & \Sigma^2 \end{bmatrix} \begin{bmatrix} V^* & 0 \\ 0 & U^* \end{bmatrix} = Q\Sigma^2Q^*$$

In a typical SVD $Q = \begin{bmatrix} V & 0 \\ 0 & U \end{bmatrix} = U_B$ would form the right singular vectors and $V_B = Q^*$ the left singular vectors. However, I cannot verify this, unless it is the case that $V = U$ in the SVD of A. If $V = U$, then

$$B = U_B\Sigma V_B^* = Q\Sigma Q^*$$

is an SVD of B. This would also be an eigenvalue decomposition of B.

□

Problem 3

We have studied the QR and SVD decompositions. This problem develops a variant of the QR decomposition, the QL decomposition. An $m \times m$ matrix of the form

$$K_m = \begin{bmatrix} & & & & 1 \\ & & & 1 & \\ & & \iddots & & \\ & 1 & & & \\ 1 & & & & \end{bmatrix} = \begin{bmatrix} k_{ij} \end{bmatrix},$$

where $k_{i,m-i+1} = 1$, $1 \leq i \leq m$, and all other entries are zero is termed a *reversal matrix* and sometimes the *reverse identity matrix.*

(a) Show that $K_m^2 = I$ (a very handy feature).

Proof. To avoid confusion in notation, for Part (a) we assume K is $m \times m$ and refer to the matrix K_m as K. Singular subscripts on K will refer to a columns of K, e.g., K_j is the j^{th} column of K. Double subscripts on K will refer to entries in K, i.e., K_{ij} is the ij-entry of K. With that out of the way, let $B = K^2$. Then by (1.6) from the text we have that the j^{th} column of B is

$$\star \quad b_j = \sum_{k=1}^{m} K_{kj} K_k,$$

that is, b_j is a linear combination of the columns of K whose coefficients are the j^{th} columns of K. But $K_{kj} = 0$ unless $k = m - j + 1$, in which case it equals 1. Thus, the formula $\star$ selects the $(m - j + 1)^{st}$ column of K_m for b_j. This selection effectively flips K_m across it's 'y-axis.' That is, $b_1 = K_{m-1+1} = K_m, b_2 = K_{m-2+1} = K_{m-2}, b_3 = K_{m-3+1} = K_{m-2}, \ldots, b_m = K_{m-m+1} = K_1$. The result is that

$$K_m^2 = I.$$

$\square$

(b) If A is $m \times n$ matrix, $m \geq n$, what is the action of $K_m A$? What about AK_n?

Solution: $K_m A$ reflects A across it's 'x-axis.' This can be seen by considering the formula $\star$. The j^{th} column of $K_m A$ is given by $\sum_{k=1}^{m} a_{kj} K_j$, where again, K_j refers to the j^{th} column of K_m. Since the K_j column only has a one in it's $(m - j + 1)^{st}$ entry for $j = 1, \ldots, m$, this selects $a_{(m-k+1)j}$, that is, $b_j = a_{(m-k+1)j}$ for $k = 1, \ldots m$. As already stated, this is a reflection across A's ''x-axis.'

(c) If R is an upper triangular $n \times n$ matrix, what is the form of the product $K_n R K_n$?

Solution: As shown in Part (a), right multiplication by K_n results in a reflection of $K_n R$ across it's 'y-axis.' Thus, in $K_n R K_n$, we have a reflection first across R's x-axis, then that result is refelcted across it's y-axis.

(d) Let $AK_n = \hat{Q}\hat{R}$ be the reduced QR decomposition of AK_n, $m \geq n$. Show that $A = (\hat{Q}K_n)(K_n \hat{R} K_n)$, and from that deduce the decomposition

$$A = QL,$$

where Q is an $m \times n$ matrix with orthogonal columns, and L is an $n \times n$ lower triangular matrix. This is a reduced QL decomposition.

Solution:

Proof. We are given $AK_n = \hat{Q}\hat{R}$. By Part (a), $K_n^2 = I$. Moreover, since $K_n^* = K_n$, K_n is unitary. Using these facts, we have

$$\begin{aligned} AK_n &= \hat{Q}\hat{R} \\ AK_n^2 &= \hat{Q}I\hat{R}K_n \\ A &= \hat{Q}K_nK_n\hat{R}K_n \\ A &= (\hat{Q}K_n)(K_n\hat{R}K_n) \end{aligned}$$

By Part (c), $K_n\hat{R}K_n = L$ and by Part (a), $\hat{Q}K_n = Q$. Therefore,

$$A = (\hat{Q}K_n)(K_n\hat{R}K_n) = QL.$$

□

Problem 4

See attached code.

In [ ]:
```
# Import standard numeric and visualization libs
import numpy as np
import matplotlib.pyplot as plt

# Import my householder QR factorization implementation
# import sys
# sys.path.insert(0, "/home/masvgp/dev/byu_num_lin_alg_2022/src/")
# from factorizations.householder import house, formQ
```

In [ ]:
```
# My implementation of householder factorization and it's associated utility functi
def e(i, length):
    """Generate a unit coordinate vector, i.e., Given an index i and a length, retu

    Args:
        i (int): Integer index in which 1.0 will be assigned
        length (int): Length of the desired output vector

    Returns:
        arr: Unit coordinate vector with 1.0 in the ith position
    """
    e_i = np.zeros(length, dtype=np.float64)
    e_i[i] = 1.0
    return e_i

def reflector(v):
    """Given a unit vector v, return the householder reflector I - 2 * np.outer(v,

    Args:
        v (arr): Vector
    Returns:
        reflector (arr): A len(v) x len(v) array representing a householder reflect
    """
    I = np.eye(len(v))
    outer = np.outer(v, v)


    return (I - 2 * outer)

def house(A, reduced=False):
    """Compute the factor R of a QR factorization of an m x n matrix A with m >= .

    Args:
        A (arr): A numpy array of shape m x n
    Returns:
        R (arr): The upper diagonal matrix R in a QR factorization
    """
    m = A.shape[0] # Get row-dim of A
    n = A.shape[1] # Get col-dim of A
    W = np.zeros((m, n))
    #Q = np.eye(m, m)
    # V = np.zeros((m, n))

    # Cast A as type np.float64
    A = A.astype(np.float64)

    for k in range(n):
        # Get the first column of the (m-k+1, n-k+1) submatrix of A
        x = A[k:m, k]

        # Compute e_1, sign function, and the norm of x
        e_1 = e(0, len(x))
        sign = np.sign(x[0])
        norm_x = np.linalg.norm(x)

        # Compute vk, the vector reflected across the Householder hyperplane
```

```
        vk = (sign * norm_x * e_1) + x
        vk = vk / np.linalg.norm(vk)

        # Store vk in W
        W[k:m, k] = vk

        # Apply the reflector to the k:m, k:n submatrix of A to put
        # zeros below the diagonal of the kth column of A
        A[k:m, k:n] = reflector(vk) @ A[k:m, k:n]

    if reduced == True:
        # return np.around(A[:n, :n], decimals=8), W
        return A[:n, :n], W
    else:
        # Return full R
        return A, W
        # return np.around(A, decimals=8), W

def formQ(W, reduced=False):
    """Form Q from the lower diagonal vk's formed in the householder algorithm.

    Args:
        W (arr): An m x n matrix containing the vk in the kth column

    Returns:
        arr: Returns an m x m orthonormal matrix Q such that QR = A
    """
    m = W.shape[0]
    n = W.shape[1]
    Q = np.eye(m, m)
    for k in range(n):
            Qk = np.eye(m, m)
            Qk[k:m, k:m] = reflector(W[k:m, k])
            Q = Q @ Qk

    if reduced == True:
        # Return reduced Q
        return Q[:m, :n]
        # return np.around(Q[:m, :n], decimals=8)
    else:
        # Return full Q
        return Q
        # return np.around(Q, decimals=6)
```

Problem 4

Use the result of Problem 3 to develop a Python function lq that takes as input a matrix A and returns Q and L where $A = QL$ is a QL-decomposition, and test it with random matrices of sizes 10×7 and 75×50. Organize your code in a single Python notebook and include it in the PDF you turn in. By test your code, I mean that you compute the QL decomposition, that you verify Q has orthonormal columns, you verify that L is lower triangular, and that you compute $\| A - QL \|_2$, where A is a random matrix for wich you compute the QL decomposition. I should be able to run your single file code and observe

In [ ]:
```
def ql(A):
    """Compute the QL factorization of an m x n matrix A.

    Args:
        A (arr): An m x n matrix
    Return:
        Q (arr): a unitary matrix
        L (arr): A lower triangular matrix
    """
    # Get dims of A
    m = A.shape[0]
    n = A.shape[1]

    # Define the reversal matrix K_n
    Kn = np.flip(np.eye(n), axis=0)

    # Compute reduced QR(AK_n) - \hat{Q} is m x n, \hat{R} is n x n.
    Rhat, W = house(A @ Kn, reduced=True)
    Qhat = formQ(W, reduced=True)

    # Compute Q = \hat{Q}K_n
    Q = Qhat @ Kn

    # Compute L = K_n\hat{R}K_n
    L = Kn @ Rhat @ Kn

    # return Q, L
    return Q, L
```

Testing

1. Define matrices A and B of size 10×7 and 75×50, respectively. Then do the following:
2. Compute $QL(A)$ and $QL(B)$
3. Verify that Q has orthonormal columns
4. Verify that L is lower triangular
5. Compute $\| A - QL \|_2$ and $\| B - QL \|_2$, where A and B are the random matrices defined in (1).

Testing for the 10×7 random matrix a

In [ ]:
```
# 1. Define random matrix of size 10 x 7
A = np.random.normal(loc=0.0, scale=1.0, size=(10, 7))
```

In [ ]:
```
# 2. Compute $QL(A)$
QA, LA = ql(A)
```

In [ ]:
```
# 3. Verify that $Q$ has orthonormal columns
# Compute QA.T @ QA
QATQA = QA.T @ QA

# Check the diagonals are 1.0
print(f"The dimensions of $QA.T @ QA are: {QATQA.shape[0]} x {QATQA.shape[1]}")
print(f"Check that the diagonal entries of QA.T @ QA sum to {QATQA.shape[0]} by che

# Set diagonal entries to 0.0
for i in range(QATQA.shape[0]):
    QATQA[i, i] = 0.0

# Compute the max absolute value of the the array QATQA with diagonal set to 0.0.
print("Check the max absolute value of the array after setting the diagonal entries
```

```
The dimensions of $QA.T @ QA are: 7 x 7
Check that the diagonal entries of QA.T @ QA sum to 7 by checking the sum of the di
agonal entries.
The sum is equal to 7: True
Check the max absolute value of the array after setting the diagonal entries to 0.
0.
This effectively checks the max absolute value of the off-diagonal entries.
The max absolute value of the resulting array is: 3.2963142379103426e-16
```

In [ ]:
```
# 4. Verify that $L$ is lower triangular
# Extract upper diagonal part of L
LA_upper_diag = [LA[i, j] for i in range(LA.shape[0]) for j in range(LA.shape[1]) i
# Extract the lower diagonal part of L
LA_lower_diag = [LA[i, j] for i in range(LA.shape[0]) for j in range(LA.shape[1]) i

print("The max absolute value of the upper diagonal entries of LA is: " + str(np.ma
print("The max absolute value of the lower diagonal entries of LA is: \n" + str(np.
```

```
The max absolute value of the upper diagonal entries of LA is: 4.2312988466972233e-
16
The max absolute value of the lower diagonal entries of LA is:
3.0734874283890266
```

In [ ]:
```
# 5. Compute $\begin{Vmatrix} A - QL \end{Vmatrix}_2$, where $A$ is the random matr
print("The L2-norm of A - QL is: " + str(np.linalg.norm(A - QA @ LA)))
```

```
The L2-norm of A - QL is: 3.831465111543962e-15
```

Testing for the 75×50 random matrix B.

In [ ]:
```
# 1. Define random matrix of size 75 x 50
B = np.random.normal(loc=0.0, scale=1.0, size=(75, 50))
```

In [ ]:
```
# Compute QL for the 75 x 50 matrix B
QB, LB = ql(B)
```

In [ ]:
```
# 3. Verify that $Q$ has orthonormal columns
# Compute QA.T @ QA
QBTQB = QB.T @ QB

# Check the diagonals are 1.0
print(f"The dimensions of $QB.T @ QB are: {QBTQB.shape[0]} x {QBTQB.shape[1]}")
print(f"Check that the diagonal entries of QB.T @ QB sum to {QBTQB.shape[0]} by che

# Set diagonal entries to 0.0
for i in range(QBTQB.shape[0]):
    QBTQB[i, i] = 0.0

# Compute the max absolute value of the the array QATQA with diagonal set to 0.0.
print("Check the max absolute value of the array after setting the diagonal entries
```

```
The dimensions of $QB.T @ QB are: 50 x 50
Check that the diagonal entries of QB.T @ QB sum to 50 by checking the sum of the d
iagonal entries.
The sum is equal to 50: True
Check the max absolute value of the array after setting the diagonal entries to 0.
0.
This effectively checks the max absolute value of the off-diagonal entries.
The max absolute value of the resulting array is: 8.721678135614107e-16
```

In [ ]:
```
# 4. Verify that $L$ is lower triangular
# Extract upper diagonal part of L
LB_upper_diag = [LB[i, j] for i in range(LB.shape[0]) for j in range(LB.shape[1]) i
# Extract the lower diagonal part of L
LB_lower_diag = [LB[i, j] for i in range(LB.shape[0]) for j in range(LB.shape[1]) i

print("The max absolute value of the upper diagonal entries of LB is: " + str(np.ma
print("The max absolute value of the lower diagonal entries of LB is: \n" + str(np.
```

```
The max absolute value of the upper diagonal entries of LB is: 2.9251450570491478e-
15
The max absolute value of the lower diagonal entries of LB is:
9.757443792666914
```

In [ ]:
```
# 5. Compute $\begin{Vmatrix} B - QL \end{Vmatrix}_2$ where $B$ is the random matri
print("The L2-norm of B - QL is: " + str(np.linalg.norm(B - QB @ LB)))
```

```
The L2-norm of B - QL is: 1.0843623836693697e-13
```