Name: Dion Toh Siyong

Matriculation Number: U2021674D

CZ1104 Lab 4

In [1]:
```python
import matplotlib.pyplot as plt
import numpy as np
import string
import pandas as pd
```

Q1a. Write a program to test if a set of vectors {v1,v2,...,vn} is an orthogonal set.

In [2]:
```python
def orthogonal_set(matrix):
    rows, columns = matrix.shape
    for i in range (0,columns):
        col = matrix[:,i]
        for j in range (i, columns):
            if j == i:
                continue
            else:
                col2 = matrix[:,j]
                check = np.dot(col, col2)
                print("Comparing column", i, "and column",  j, "...")
                if check != 0:
                    print("Not an orthogonal set!")
                    return
    print("Orthognal set!")
```

Test 1: Orthogonal Set

In [3]:
```python
matrix = np.array([[3,-1,-0.5],
                   [1,2,-2],
                   [1,1,3.5]])


orthogonal_set(matrix)
```

```
Comparing column 0 and column 1 ...
Comparing column 0 and column 2 ...
Comparing column 1 and column 2 ...
Orthognal set!
```

Test 2: Non Orthogonal Set

In [4]:
```
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7 ,8 ,9],
                   [10, 11, 12]])

orthogonal_set(matrix)
```

```
Comparing column 0 and column 1 ...
Not an orthogonal set!
```

Test 3: Orthogonal Set

In [5]:
```
matrix = np.array([[1/3, -1/2],
                   [1/3, 0],
                   [1/3, 1/2]])


orthogonal_set(matrix)
```

```
Comparing column 0 and column 1 ...
Orthognal set!
```

Test 4: Non Orthogonal Set

In [6]:
```
matrix = np.array([[0, 0],
                   [1, -1],
                   [0, 0]])


orthogonal_set(matrix)
```

```
Comparing column 0 and column 1 ...
Not an orthogonal set!
```

Q1b. Output which pairs are not orthogonal.

In [7]:
```
def orthogonal_set_show(matrix):
    rows, columns = matrix.shape
    for i in range (0,columns):
        col = matrix[:,i]
        for j in range (i, columns):
            if j == i:
                continue
            else:
                col2 = matrix[:,j]
                check = np.dot(col, col2)
                print("Comparing column", i, "and column", j, "...")
                if check != 0:
                    print(i, "and", j, "are NOT orthogonal!")
                else:
                    print("Orthogonal!")
                print()
```

Test with Matrix A

In [8]:
```
matrix = np.array([[-6, -3, 6, 1],
                   [-1, 2, 1, -6],
                   [3, 6, 3, -2],
                   [6, -3, 6, -1],
                   [2, -1, 2, 3],
                   [-3, 6, 3, 2],
                   [-2, -1, 2, -3],
                   [ 1, 2, 1, 6]])


orthogonal_set_show(matrix)
```

```
Comparing column 0 and column 1 ...
Orthogonal!

Comparing column 0 and column 2 ...
Orthogonal!

Comparing column 0 and column 3 ...
Orthogonal!

Comparing column 1 and column 2 ...
Orthogonal!

Comparing column 1 and column 3 ...
Orthogonal!

Comparing column 2 and column 3 ...
Orthogonal!
```

Test with Non-Orthogonal Matrix

In [9]:
```
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7 ,8 ,9],
                   [10, 11, 12]])

orthogonal_set_show(matrix)
```

```
Comparing column 0 and column 1 ...
0 and 1 are NOT orthogonal!

Comparing column 0 and column 2 ...
0 and 2 are NOT orthogonal!

Comparing column 1 and column 2 ...
1 and 2 are NOT orthogonal!
```

Q1c. Is it always true (in executing program on dot product) that 2 vectors v.w ≠ 0 means v is not orthogonal to w? If it is not necessary true, give an example and propose a fix.

It is not always true. Due to the handling of floats. The dot product of 2 vectors might return a value of 2*10^-14, when it should be 0. Therefore when compared to 0, it would return false.

Therefore we should set certain criterias to ensure that the true result is taken.

If check value is less than 10^-13, we would consider it as 0.

In [10]:
```
def orthogonal_set(matrix):
    rows, columns = matrix.shape
    for i in range (0,columns):
        col = matrix[:,i]
        for j in range (i, columns):
            if j == i:
                continue
            else:
                col2 = matrix[:,j]
                check = np.dot(col, col2)
                print("Comparing column", i, "and column",  j, "...")
                if abs(check) > 10**-13: #condition
                    print("Not an orthogonal set!")
                    return
    print("Orthognal set!")
```

In [11]:

```
def orthogonal_set_show(matrix):
    rows, columns = matrix.shape
    for i in range (0,columns):
        col = matrix[:,i]
        for j in range (i, columns):
            if j == i:
                continue
            else:
                col2 = matrix[:,j]
                check = np.dot(col, col2)
                print("Comparing column", i, "and column", j, "...")
                if abs(check) < 10**-13: #condition
                    print("Orthogonal!")
                else:
                    print(i, "and", j, "are NOT orthogonal!")
                print()
```

Q2. Write a module “Orthonormalisation” that defines a a procedure orthonormalise(L) with the following specs:

- Input: a list L of linearly independent Vecs
- Output: a list L* of len(L) orthonormal Vecs such that, for i= 1,2,…, len(L), the first i Vecs of L* and the first i Vecs of L span the same space.

In [12]:
```
def Orthonormalisation(L):
    m, n = np.array(vecs).shape
    L_orthonorm = np.zeros((m,n))
    L_orthonorm[0] = L[0] #v_1 = x_1
    for i in range(1, len(L)):
        v_i = L[i]
        #print("col", i)
        for j in range(0, i):
            #print("X:",i,L[i])
            #print("V:",j,L_orthonorm[j])
            numerator = np.dot(L[i], L_orthonorm[j])
            denominator = np.dot(L_orthonorm[j], L_orthonorm[j])
            subtract = (numerator/denominator) * (L_orthonorm[j])
            v_i -= subtract
        L_orthonorm[i] = v_i
    print("Original Matrix:\n", np.array(L).transpose())
    print()
    print("After Gram-Schmidt Process:\n", L_orthonorm.transpose())
    print()
    print("Check Orthogonality")
    orthogonal_set(L_orthonorm.transpose())
    print()
    print("Normalise")
    for i in range(0, len(L_orthonorm)):
        magnitude = np.linalg.norm(L_orthonorm[i])
        L_orthonorm[i] = L_orthonorm[i]/magnitude
    print()
    print("Check Norm of each column:")
    for i in range(0, len(L_orthonorm)):
        print("Column", i, ":", np.linalg.norm(L_orthonorm[i]))
    print()
    print("Check Orthogonality")
    orthogonal_set(L_orthonorm.transpose())
    print()
    return L_orthonorm.tolist()
```

In [13]:
```python
vecs = [[4,3,2,1], [8,9,-5,-5], [10,1,-1,5]]

vecs_norm = Orthonormalisation(vecs)

print("Type:", type(vecs_norm), "\n") #output is of list type
print("After Orthonormalisation:")
print(np.array(vecs_norm).transpose()) #presenting in array format as it is 
```

```
Original Matrix:
 [[ 4  8 10]
 [ 3  9  1]
 [ 2 -5 -1]
 [ 1 -5  5]]

After Gram-Schmidt Process:
 [[ 4.          2.13333333  3.84159428]
 [ 3.          4.6        -3.65406234]
 [ 2.         -7.93333333 -3.97342872]
 [ 1.         -6.46666667  3.54266735]]

Check Orthogonality
Comparing column 0 and column 1 ...
Comparing column 0 and column 2 ...
Comparing column 1 and column 2 ...
Orthognal set!

Normalise

Check Norm of each column:
Column 0 : 0.9999999999999999
Column 1 : 1.0
Column 2 : 0.9999999999999999

Check Orthogonality
Comparing column 0 and column 1 ...
Comparing column 0 and column 2 ...
Comparing column 1 and column 2 ...
Orthognal set!

Type: <class 'list'>

After Orthonormalisation:
[[ 0.73029674  0.18677078  0.51131052]
 [ 0.54772256  0.4027245  -0.4863503 ]
 [ 0.36514837 -0.69455384 -0.52885749]
 [ 0.18257419 -0.56614893  0.47152379]]
```

Q3a. Write a program to compute the QR factorization of matrix A = QR

Where Q is orthonormal and R is upper triangular. (refer to lecture notes for the theory). You may assume A consists of linearly independent vectors.

In [14]:

```
def QR_factorization(matrix):
    m, n = matrix.shape
    Q = np.zeros((m, n))
    R = np.zeros((n, n))

    x1 = matrix[:,0]
    magnitude = np.linalg.norm(x1)
    Q[:,0] = x1/magnitude
    R[0][0]= magnitude

    for i in range(1, n):
        x_i = matrix[:,i]
        v_i = x_i
        for j in range(0,i):
            v_j = Q[:,j]
            r_val = np.dot(x_i, v_j) / np.dot(v_j, v_j)
            v_i = v_i - (r_val * v_j)
            R[j][i] = r_val
        magnitude = np.linalg.norm(v_i)
        e_i = v_i / magnitude
        Q[:,i] = e_i
        R[i][i] = magnitude
    return Q, R
```

Q3b. Compute the QR factorization of A, formed by the 3 column vectors in Q2.

In [15]:

```
Q, R = QR_factorization(np.array(vecs).transpose())

print("A:\n", np.array(vecs).transpose(), "\n")
print("Q:\n", Q, "\n")
print("R:\n", R, "\n")
```

```
A:
 [[ 4  8 10]
 [ 3  9  1]
 [ 2 -5 -1]
 [ 1 -5  5]]

Q:
 [[ 0.73029674  0.18677078  0.51131052]
 [ 0.54772256  0.4027245  -0.4863503 ]
 [ 0.36514837 -0.69455384 -0.52885749]
 [ 0.18257419 -0.56614893  0.47152379]]

R:
 [[ 5.47722558  8.03326418  8.39841255]
 [ 0.         11.42220061  0.1342415 ]
 [ 0.          0.          7.51323139]]
```

In [ ]:

```

```