```
print('Norm of a2 :', (sum(a2**2))**0.5)
print('Norm of a3 :', (sum(a3**2))**0.5)
```

```
Norm of a1 : 1.0
Norm of a2 : 0.9999999999999999
Norm of a3 : 0.9999999999999999
```

In [ ]:
```
print(a1 @ a2, a1 @ a3, a2 @ a3)
```

```
0.0, 0.0, 0.0
```

In [ ]:
```
x = np.array([1,2,3])
#Get coefficients of x in orthonormal basis
beta1 = a1 @ x
beta2 = a2 @ x
beta3 = a3 @ x
#Expansion of x in basis
x_expansion = beta1*a1 + beta2*a2 + beta3*a3
print(x_expansion)
```

```
[1. 2. 3.]
```

5.4. Gram-Schmidt algorithm

The following is a Python implementation of the Gram-Schmidt algorithm. It takes as input an array `a` which expands into `[a[0], a[1], ..., a[k]]`, containing k vectors $a_1, \dots, a_k$. If the vectors are linearly independent, it returns an array `q` with orthonormal set of vectors computed by the Gram-Schmidt algorithm. If the vectors are linearly dependent and the Gram-Schmidt algorithm terminates early in the $i+1$th iteration, it returns the array `q[0], ..., q[i-1]` of length i.

In [ ]:
```
import numpy as np
def gram_schmidt(a):
    q = []
    for i in range(len(a)):
        #orthogonalization
        q_tilde = a[i]
        for j in range(len(q)):
            q_tilde = q_tilde -  (q[j] @ a[i])*q[j]
        #Test for dependennce
```

```
        if np.sqrt(sum(q_tilde**2)) <= 1e-10:
            print('Vectors are linearly dependent.')
            print('GS algorithm terminates at iteration ', i+1)
            return q
        #Normalization
        else:
            q_tilde = q_tilde / np.sqrt(sum(q_tilde**2))
            q.append(q_tilde)
    print('Vectors are linearly independent.')
    return q
```

Here we initialize the output array as an empty array. In each iteration, we add the next vector to the array using the `append()` method.

Example. We apply the function to the example on page 100 of VMLS.

```
In [ ]: a = np.array([ [-1, 1, -1, 1], [-1, 3, -1, 3], [1, 3, 5, 7] ])
        q = gram_schmidt(a)
        print(q)
        #Test orthonormality
        print('Norm of q[0] :', (sum(q[0]**2))**0.5)
        print('Inner product of q[0] and q[1] :', q[0] @ q[1])
        print('Inner product of q[0] and q[2] :', q[0] @ q[2])
        print('Norm of q[1] :', (sum(q[1]**2))**0.5)
        print('Inner product of q[1] and q[2] :', q[1] @ q[2])
        print('Norm of q[2] :', (sum(q[2]**2))**0.5)
```

```
Vectors are linearly independent.
[array([-0.5,  0.5, -0.5,  0.5]), array([0.5, 0.5, 0.5, 0.5]),
↪  array([-0.5, -0.5,  0.5,  0.5])]
Norm of q[0] : 1.0
Inner product of q[0] and q[1] : 0.0
Inner product of q[0] and q[2] : 0.0
Norm of q[1] : 1.0
Inner product of q[1] and q[2] : 0.0
Norm of q[2] : 1.0
```

Example of early termination. If we replace a_3 with a linear combination of a_1 and a_2, the set becomes linearly dependent.

In [ ]:
```
b = np.array([a[0],a[1], 1.3*a[0] + 0.5*a[1]])
q = gram_schmidt(b)
print(q)
```

```
Vectors are linearly dependent.
GS algorithm terminates at iteration  3
[array([-0.5,  0.5, -0.5,  0.5]), array([0.5, 0.5, 0.5, 0.5])]
```

Here the algorithm terminated at iteration 3. That means, the third iteration is not completed and thus two orthonormal vectors are returned.

Example of independence-dimension inequality. We know that any three 2-vectors must be dependent. Let's use the Gram-Schmidt algorithm to verify this for three specific vectors.

In [ ]:
```
three_two_vectors = np.array([[1,1],[1,2],[-1,1]])
q = gram_schmidt(three_two_vectors)
print(q)
```

```
Vectors are linearly dependent.
GS algorithm terminates at iteration  3
[array([0.70710678, 0.70710678]), array([-0.70710678,
↪  0.70710678])]
```