In [1]:
```
#1
print('Python Lab 5 - Ryan Stettnisch')
import numpy as np
```

```
Python Lab 5 - Ryan Stettnisch
```

In [2]:
```
#2
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
```

In [3]:
```
#2a
a = np.array([[1,-1.1],[-2.8,-0.3],[-0.4, 1.5]]) #This is linearly dependent be
print(gram_schmidt(a))
```

```
Vectors are linearly dependent.
GS algorithm terminates at iteration  3
[array([ 0.67267279, -0.73994007]), array([-0.73994007, -0.67267279])]
```

In [4]:
```
#2b
b = np.array([[1,0,1,0,1],[0,1,0,0,1],[0,1,0,1,0]])
print(gram_schmidt(b))
```

```
Vectors are linearly independent.
[array([0.57735027, 0.        , 0.57735027, 0.        , 0.57735027]), array([-
0.25819889,  0.77459667, -0.25819889,  0.        ,  0.51639778]), array([ 0.16
903085,  0.3380617 ,  0.16903085,  0.84515425, -0.3380617 ])]
```

In [5]:
```
#2c
c = np.array([[1,2,0],[-2,0,3],[1,0,2]])
print(gram_schmidt(c))
```

```
Vectors are linearly independent.
[array([0.4472136 , 0.89442719, 0.        ]), array([-0.45807867,  0.22903933,
0.8588975 ]), array([ 0.76822128, -0.38411064,  0.51214752])]
```

In [6]:
```
#2d
d = np.array([[1,2,0],[-2,0,3],[-1,2,3]])
print(gram_schmidt(d))
```

```
Vectors are linearly dependent.
GS algorithm terminates at iteration  3
[array([0.4472136 , 0.89442719, 0.        ]), array([-0.45807867,  0.22903933,
0.8588975 ])]
```

In [7]:

```
#3a
q = gram_schmidt(a)
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('Norm of q[1] : ', (sum(q[1]**2))**0.5)
```

```
Vectors are linearly dependent.
GS algorithm terminates at iteration  3
Norm of q[0] :  1.0
Inner product of q[0] and q[1] :  0.0
Norm of q[1] :  1.0
```

In [8]:

```
#3b
q = gram_schmidt(b)
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.
Norm of q[0] : 1.0
Inner product of q[0] and q[1] : -1.1102230246251565e-16
Inner product of q[0] and q[2] : 8.326672684688674e-17
Norm of q[1] : 1.0
Inner product of q[1] and q[2] : -1.1102230246251565e-16
Norm of q[2] : 1.0
```

In [9]:

```
#3c
q = gram_schmidt(c)
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.
Norm of q[0] : 1.0
Inner product of q[0] and q[1] : -2.7755575615628914e-17
Inner product of q[0] and q[2] : 5.551115123125783e-17
Norm of q[1] : 1.0
Inner product of q[1] and q[2] : 0.0
Norm of q[2] : 1.0
```

In [10]:

```
#3d
q = gram_schmidt(d)
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('Norm of q[1] :', (sum(q[1]**2))**0.5)
```

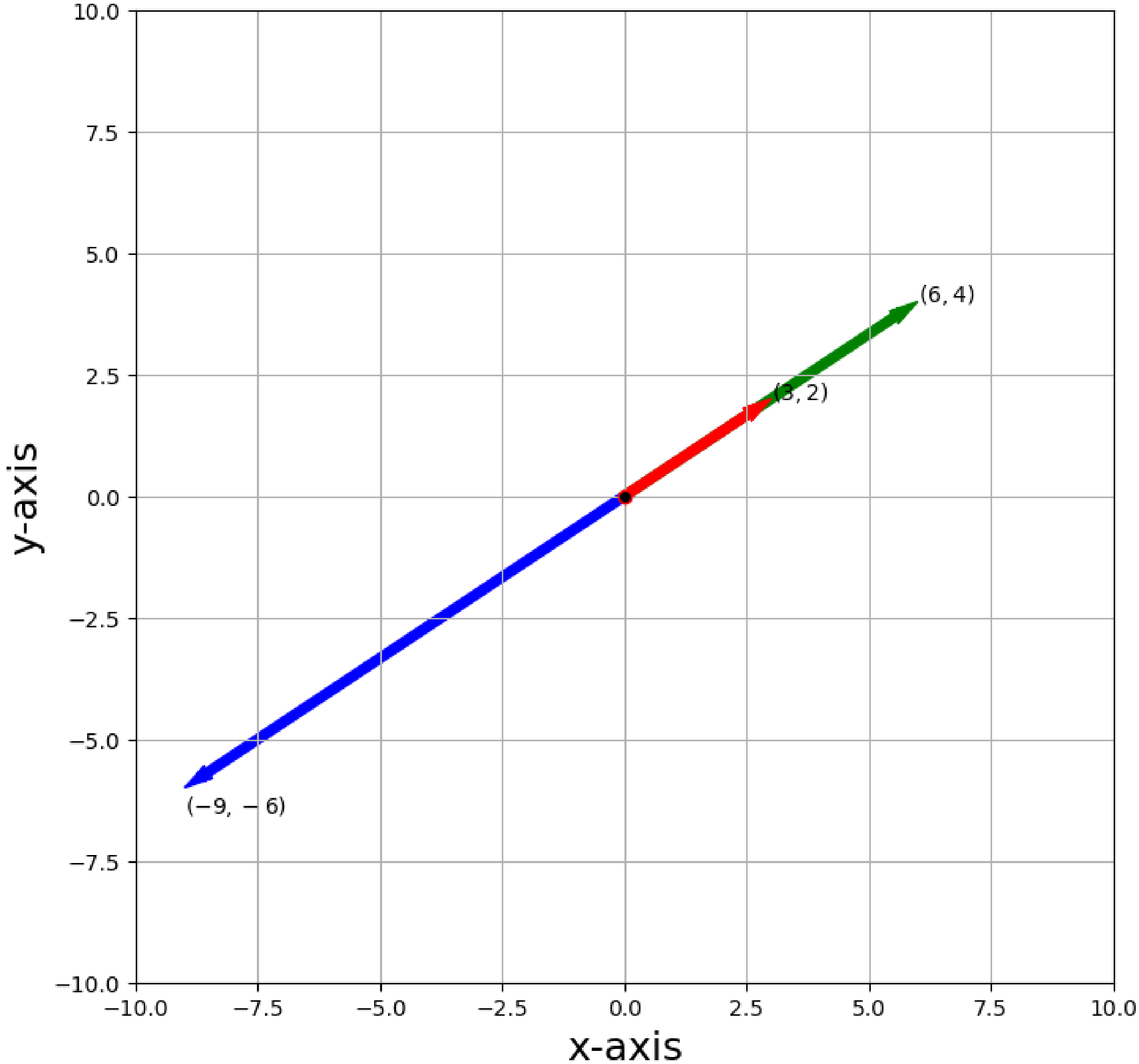
```
Vectors are linearly dependent.
GS algorithm terminates at iteration  3
Norm of q[0] : 1.0
Inner product of q[0] and q[1] : -2.7755575615628914e-17
Norm of q[1] : 1.0
```

In [11]:

```
#4
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import sympy as sy
sy.init_printing()
fig, ax = plt.subplots(figsize = (8, 8))
######################Arrows######################
arrows = np.array([[[0,0,3,2]],
[[0,0,-9,-6]],
[[0,0,6,4]]])
colors = ['r','b','g']

for i in range(arrows.shape[0]):
    X,Y,U,V = zip(*arrows[i,:,:])
    ax.arrow(X[0], Y[0], U[0],V[0], color = colors[i], width = .18,
        length_includes_head = True,
        head_width = .3, # default: 3*width
        head_length = .6,
        overhang = .4, zorder = -i)
ax.scatter(0, 0, ec = "red", fc = "black", zorder = 5)
ax.text(6, 4, '$(6, 4)$')
ax.text(-9, -6.5, '$(-9, -6)$')
ax.text(3, 2, '$(3, 2)$')

ax.grid(True)
ax.axis([-10, 10, -10, 10])
ax.set_xlabel('x-axis', size = 18)
ax.set_ylabel('y-axis', size = 18)
plt.show()
#These vectors are linearly independent
#You can reach any vector that is a multiple of (3,2) like (6,4), (9,6)
```



In [12]:

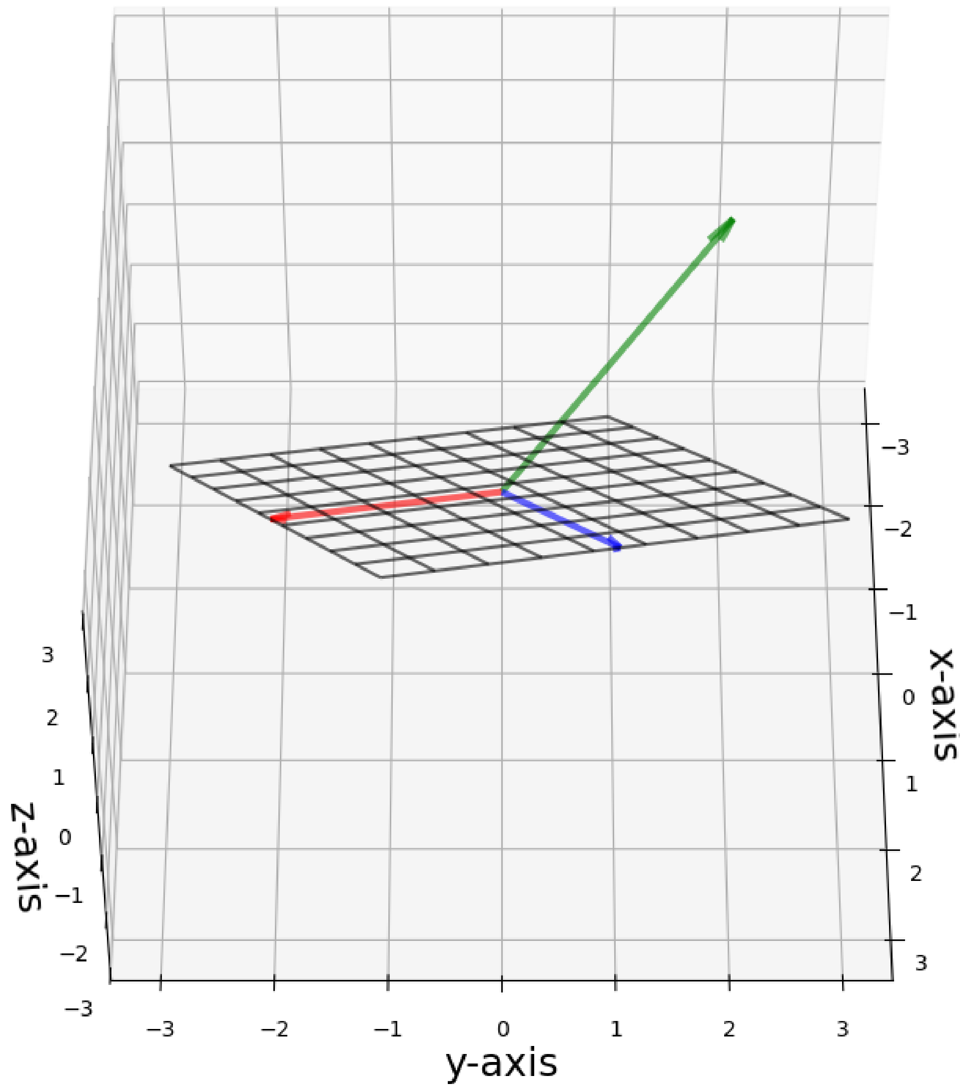
```
#5
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(projection='3d')
s = np.linspace(-1, 1, 10)
t = np.linspace(-1, 1, 10)
S, T = np.meshgrid(s, t)
X = S+2*T
Y = -2*S+T
Z = S+2*T
ax.plot_wireframe(X, Y, Z, linewidth = 1.5, color = 'k', alpha = .6)

vec = np.array([[[0, 0, 0, 1, -2, 1]],
                [[0, 0, 0, 2, 1, 2]],
                [[0, 0, 0, -1, 2, 3]]])

colors = ['r','b','g']
for i in range(vec.shape[0]):
    X, Y, Z, U, V, W = zip(*vec[i,:,:])
    ax.quiver(X, Y, Z, U, V, W, length=1, normalize=False, color = colors[i],
        arrow_length_ratio = .08, pivot = 'tail',
        linestyles = 'solid',linewidths = 3, alpha = .6)

ax.set_xlabel('x-axis', size = 18)
```

```
ax.set_ylabel('y-axis', size = 18)
ax.set_zlabel('z-axis', size = 18)
ax.view_init(elev=50., azim=0)
plt.show()
#Linearly independent since you can reach any vector where
# x = 1a + 2b - 1c
# y = -2a + 1b + 2c
# z = 1a + 2b + 3c
```



In [ ]: