AM 205 - HW 2 - Exercise 1

October 12, 2021

[ ]:
```
#Problem 1:

import numpy as np
import scipy.linalg as linalg
from numpy.linalg import norm as norm
import matplotlib.pyplot as plt
```

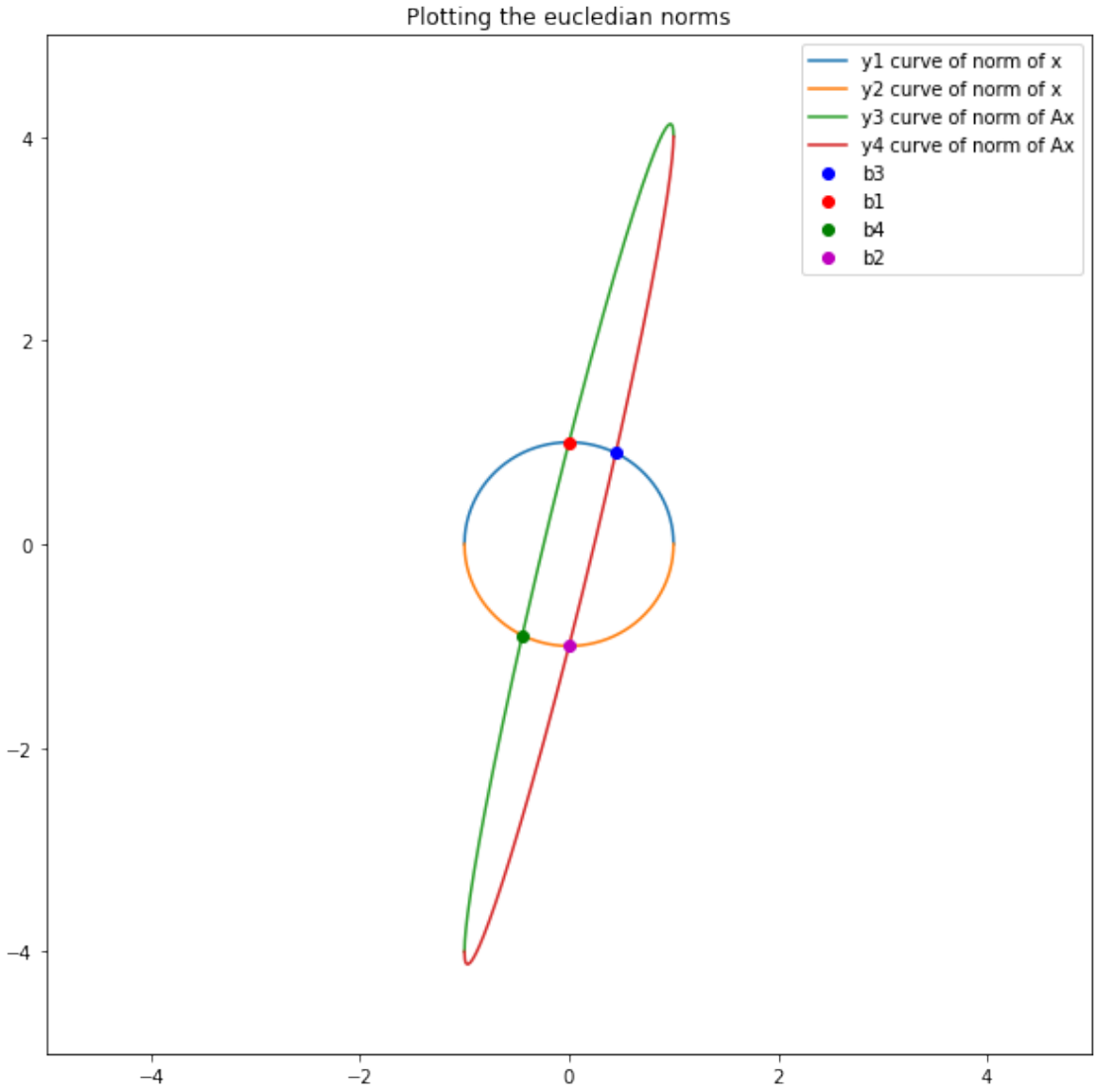
0.1 Exercise 1

Explanation is provided in the Latex writeup.

[ ]:
```
########## 1A

A = np.matrix([[4,-1], [1,0]])
x = np.linspace(-1,1,1000)

# plot norm(x) = 1
y1 = np.sqrt(1-x**2)
y2 = - np.sqrt(1-x**2)

# plot norm(Ax) = 1
y3 = 4*x + np.sqrt(1-x**2)
y4 = 4*x - np.sqrt(1-x**2)

b3 = (0.447,0.8944)
b1 = (0,1)
b4 = (-0.447,-0.8944)
b2 = (0,-1)

xmin = -5
xmax = 5
ymax = 5
ymin = -5

plt.figure(figsize=(10,10))
plt.axis([xmin,xmax,ymin,ymax])
plt.plot(x,y1, label = 'y1 curve of norm of x')
plt.plot(x,y2, label = 'y2 curve of norm of x')
```

```
plt.plot(x,y3, label = 'y3 curve of norm of Ax')
plt.plot(x,y4, label = 'y4 curve of norm of Ax')
plt.plot(0.447,0.8944, "bo", label = "b3")
plt.plot(0,1, "ro", label = "b1")
plt.plot(-0.447,-0.8944, "go", label = "b4")
plt.plot(0,-1, "mo", label = "b2")
plt.plot
plt.title("Plotting the eucledian norms")
plt.legend()
plt.draw()
plt.savefig("graph_1a.jpeg", dpi=300, bbox_inches='tight')

b1_n = norm(b1)
b2_n = norm(b2)
b3_n = norm(b3)
b4_n = norm(b4)

b_points = [b1, b2, b3, b4]



ab1 = A @ b1
ab1_n = norm(ab1)
print(f"The norm of Ab for b1: {ab1_n}")

ab2 = A @ b2
ab2_n = norm(ab2)
print(f"The norm of Ab for b2: {ab2_n}")

ab3 = A @ b3
ab3_n = norm(ab3)
print(f"The norm of Ab for b3: {ab3_n}")

ab4 = A @ b4
ab4_n = norm(ab4)
print(f"The norm of Ab for b4: {ab4_n}")


print(f"The norms for b1 - b4 in order {b1_n,b2_n, b3_n, b4_n}")
```

```
The norm of Ab for b1: 1.0
The norm of Ab for b2: 1.0
The norm of Ab for b3: 0.9991646310793832
The norm of Ab for b4: 0.9991646310793832
The norms for b1 - b4 in order (1.0, 1.0, 0.9998801728207236,
0.9998801728207236)
```



0.2 1B

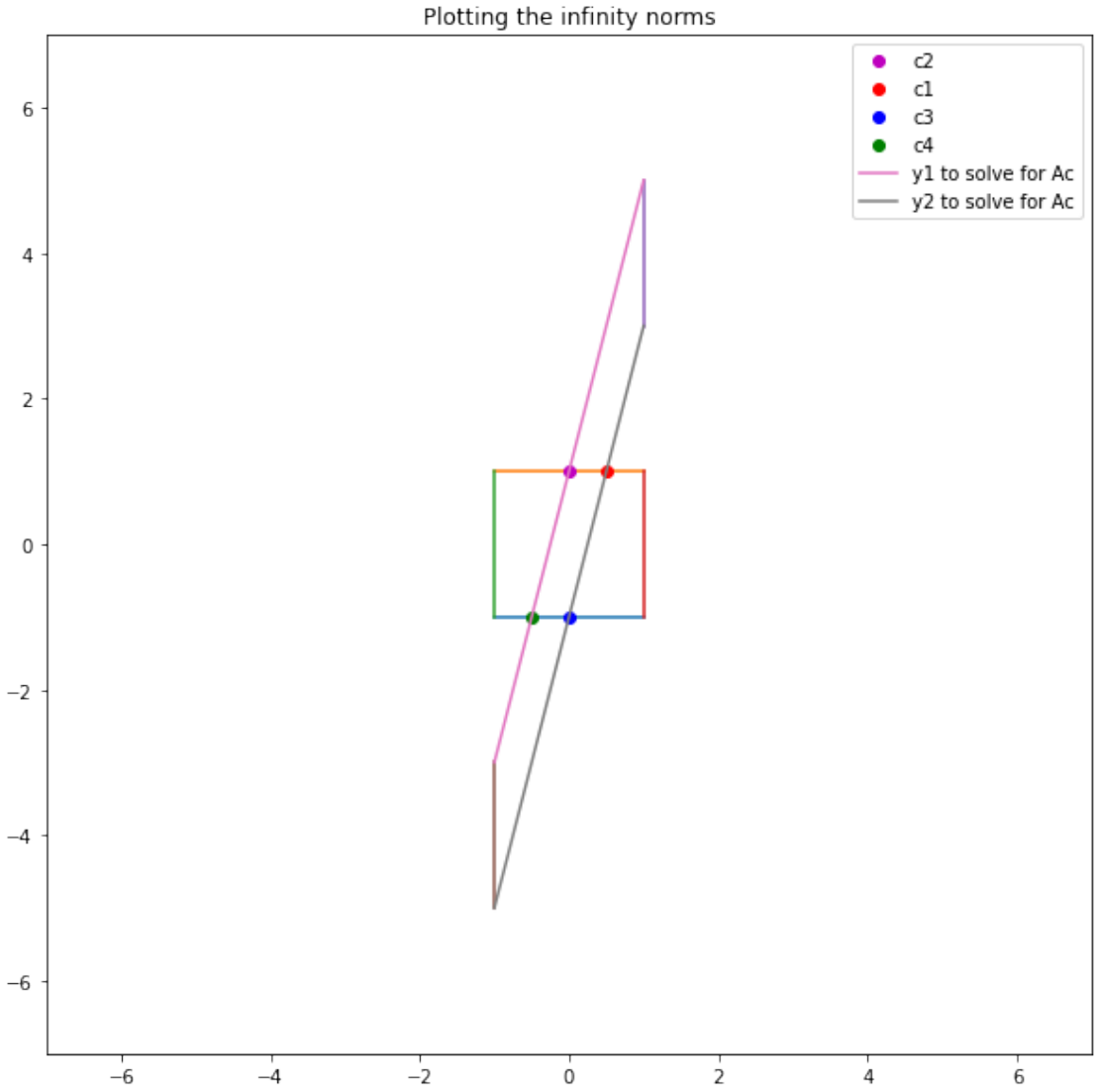
Explanation is provided in the Latex Writeup.

[ ]:
```
# Problem 1b


A = np.matrix([[4,-1], [1,0]])
x = np.linspace(-1,1,1000)


# From the analytical solution in the Latex writeup, my points are:
```

```
c3 = [0,-1]
c1 = [0.5,1]
c2 = [0,+1]
c4 = [-0.5,-1]

c_points = [c1, c2, c3, c4]



y1 = 4*x + 1
y2 = 4*x - 1

c1_n = np.max(np.abs(c1))
c2_n = np.max(np.abs(c2))
c3_n = np.max(np.abs(c3))
c4_n = np.max(np.abs(c4))


ac1 = A @ c1
ac1_n = np.max(np.abs(ac1))
print(f"The norm of Ac for c1: {ac1_n}")

ac2 = A @ c2
ac2_n = np.max(np.abs(ac2))
print(f"The norm of Ac for c2: {ac2_n}")

ac3 = A @ c3
ac3_n = np.max(np.abs(ac3))
print(f"The norm of Ac for c3: {ac3_n}")

ac4 = A @ c4
ac4_n = np.max(np.abs(ac4))
print(f"The norm of Ac for c4: {ac4_n}")

print(f"The norms for c1 - c4 in order {c1_n,c2_n, c3_n, c4_n}")
xmin = -7
xmax = 7
ymax = 7
ymin = -7

plt.figure(figsize=(10,10))
plt.axis([xmin,xmax,ymin,ymax])
plt.plot([-1,1], [-1,-1]) #graph for infinity norm of x
plt.plot([1,-1],[1,1])   #graph for infinity norm of x
plt.plot([-1,-1],[1,-1]) #graph for infinity norm of x
plt.plot([1,1],[-1,1])   #graph for infinity norm of x
```

```
plt.plot([1,1], [3,5])
plt.plot([-1,-1],[-3,-5])

plt.plot(0,1,'mo',label = 'c2')
plt.plot(0.5,1,'ro',label = 'c1')
plt.plot(0,-1,'bo' ,label = 'c3')
plt.plot(-0.5, -1,'go',label = 'c4')

plt.plot(x, y1, label = 'y1 to solve for Ac')
plt.plot(x, y2, label = 'y2 to solve for Ac')


plt.legend()
plt.title("Plotting the infinity norms")
plt.draw()
plt.savefig("graph_1b.jpeg", dpi=300, bbox_inches='tight')
```

```
The norm of Ac for c1: 1.0
The norm of Ac for c2: 1
The norm of Ac for c3: 1
The norm of Ac for c4: 1.0
The norms for c1 - c4 in order (1.0, 1, 1, 1.0)
```



[ ]:

0.3 1 c

In problem 1c I worked with Adriana, Leon, Emma, from Yue Sun OH

Here the newton method is used to find the roots, i.e. the x values. However, here we want to find a x and a y value, here we want to find a point d. The point d is a vector with two values, x and y. As a consequence, here we are using Newton in a 2-dimensional case and not in a 1-dimensional case. To solve for 2-dimensional cases of rootfinding, we have to take the derivative of the function we are considering not for one value, but for each.

Additionally, we are dealing with two functions here, given the fact that we have a system of e quations of 2 functions. These equations are:

```
$d = (x^4  + y^4)^{1/4} = 1$
$d = (x^4  + y^4)^{1/4} - 1$
```

And:

```
$Ad = (4*x + y^4)^{1/4} + x^4 =  1 $
$Ad = (4*x + y^4)^{1/4} + x^4 -1 $
```

The elements of the Jacobian are the derivative of each of these functions in respect to x and in respect to y. Hence our Jacobian will be a 2x2 Matrix with 4 elements

[ ]:
```
#Problem 1c
#Worked with Adriana, Leon, Emma, from Yue Sun OH

# d = (x^4 + y^4)^0.25 - 1
# Ad = (4*x + y^4)^0.25 + x^4 - 1
#can place bounds for the iterations or error 10e-12
#use a while loop
# can use numpy . solve x

def Newton(x,y): # Change n to threshold of ||f(x) - 0|| > 10 e ^-12


    n = 0
    while n < 10000:
        #find a det of Jacobian
        # a,b,c,d = J(x,y)
        # x = x-(d*f1 - b*f2)*det
        # y = y-(-c*f1 + a*f2)*det

        xx, xy, yx, yy = J(x,y)
        fx, fy = f(x,y)
        det = 1 / ((xx * yy) - (xy * yx))
        x = x -(yy * fx - xy * fy)*det
        y = y - (-yx* fx + xx * fy)*det

        # Solve using Linalg
        #fxy = f(x,y)
        #Jxy = J(x,y)
        #delta = np.linalg.solve(Jxy, -fxy)
        #x = x + delta[0]
        #y = y + delta[1]

        n += 1
    return x,y

def f(x,y):
    fx = (x**4 + y**4)**(0.25) - 1  #Need to change
    fy = ((4*x - y)**4 + x**4)**(0.25) - 1
```

```
    #return np.array([[fx],[fy]])
    return [fx, fy]

def J(x,y):
    xx = x**3/(x**4 + y**4)**0.75
    xy = y**3/(x**4 + y**4)**0.75
    yx = (x**3 + 4*(4*x - y)**3)/(x**4 + (4*x-y)**4)**(0.75)
    yy = (-(4*x-y)**3)/(x**4 + (4*x-y)**4)**0.75
    #return np.array([[xx, xy], [yx, yy]])
    return [xx, xy, yx, yy]
```

[ ]:
```
# Helps find our Jacobian terms
from sympy import *
x, y, z = symbols('x y z')
init_printing(use_unicode=True)

fx = (x**4 + y**4)**(1/4) - 1
fy = ((4*x - y)**4 + x**4)**(1/4) - 1
diff(fy,y)
```

```
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/IPython/lib/latextools.py:126: MatplotlibDeprecationWarning:
The to_png function was deprecated in Matplotlib 3.4 and will be removed two
minor releases later. Use mathtext.math_to_image instead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/IPython/lib/latextools.py:126: MatplotlibDeprecationWarning:
The to_rgba function was deprecated in Matplotlib 3.4 and will be removed two
minor releases later. Use mathtext.math_to_image instead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/IPython/lib/latextools.py:126: MatplotlibDeprecationWarning:
The to_mask function was deprecated in Matplotlib 3.4 and will be removed two
minor releases later. Use mathtext.math_to_image instead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/IPython/lib/latextools.py:126: MatplotlibDeprecationWarning:
The MathtextBackendBitmap class was deprecated in Matplotlib 3.4 and will be
removed two minor releases later. Use mathtext.math_to_image instead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
```

[ ]:
$$-\frac{1.0\,(4x-y)^3}{\left(x^4+(4x-y)^4\right)^{0.75}}$$

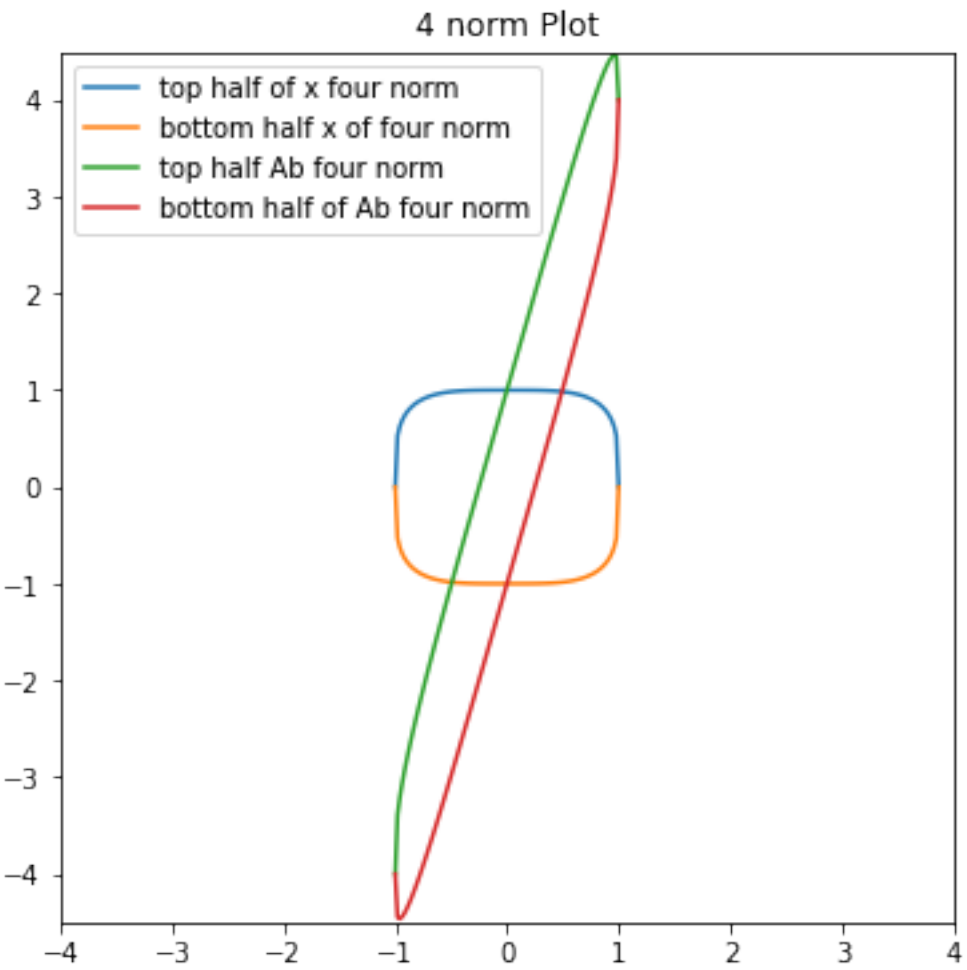
[ ]:
```
d1 = Newton(0,1)
d2 = Newton(0, -1)
d3 = Newton(0.447, 0.8944)
```

```
d4 = Newton(-0.447, -0.8944)
d_points = (d1,d2,d3,d4)
d_points
```

[ ]: $((0.0,\ 1.0),\ (0.0,\ -1.0),\ (0.492479060505452,\ 0.984958121010905),\ (-0.492479060505452,\ -0.984958121010905$

[ ]:
```
x = np.linspace(-1,1,100)

def f(x):
    return (1-x**4)**(0.25)

def f2(x):
    return -(1-x**4)**(0.25)

def ab(x):
    return (1-x**4)**(0.25) + 4*x

def ab2(x):
    return -(1-x**4)**(0.25) + 4*x


fx = ([f(xx) for xx in x])
f2x = ([f2(xx) for xx in x])
abx = ([ab(xx) for xx in x])
ab2x = ([ab2(xx) for xx in x])



plt.figure(figsize = (6,6))
plt.plot(x,fx, label = "top half of x four norm")
plt.plot(x,f2x, label = "bottom half x of four norm")
plt.plot(x, abx, label = "top half Ab four norm")
plt.plot(x, ab2x, label = "bottom half of Ab four norm")
plt.axis([-4,4,-4.5,4.5])
plt.legend()
plt.title("4 norm Plot")
```

[ ]:
```
Text(0.5, 1.0, '4 norm Plot')
```

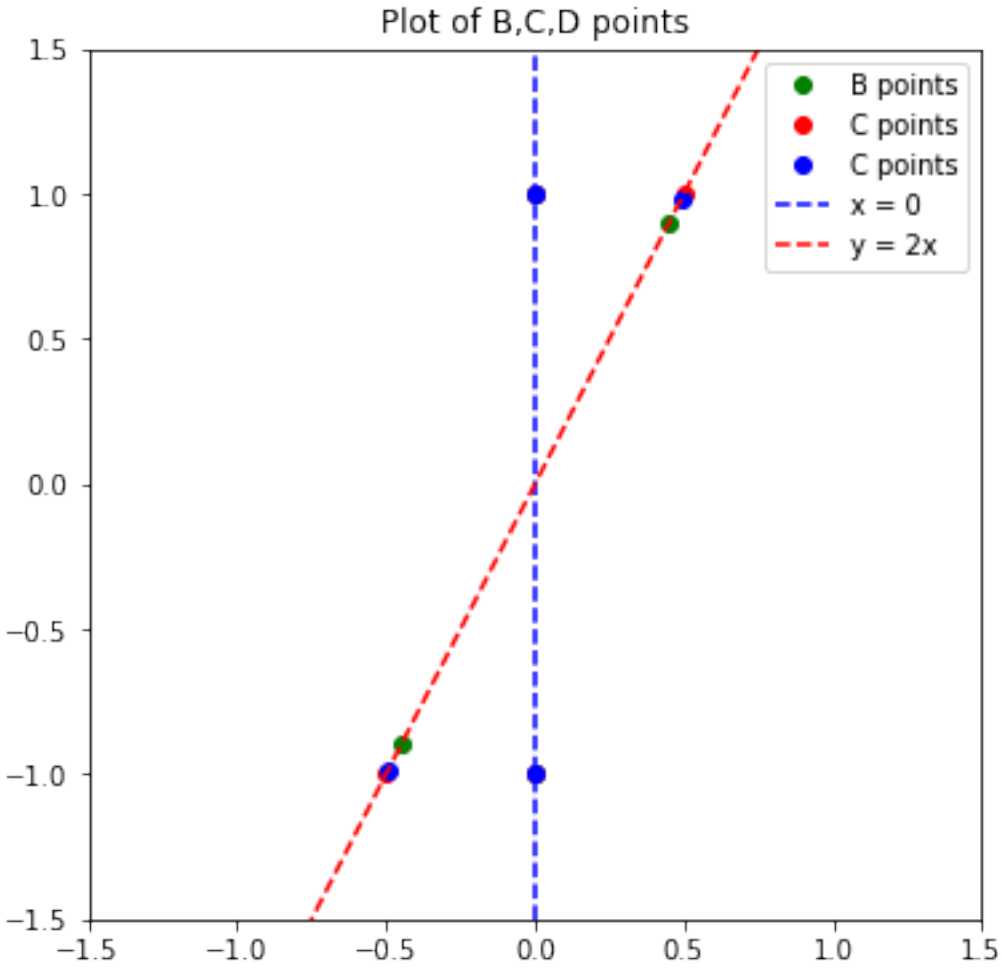


[ ]:
```
#Problem 1d

print(f"The B points: {b_points}")
print(f"The C points: {c_points}")
print(f"The D points: {d_points}")

bx = [x[0] for x in b_points]
by = [x[1] for x in b_points]
cx = [x[0] for x in c_points]
cy = [x[1] for x in c_points]
dx = [x[0] for x in d_points]
dy = [x[1] for x in d_points]

x = np.linspace(-1.5,1.5,100)
y = np.zeros(x.shape)
```

```
plt.figure(figsize = (6,6))
plt.plot(bx,by,'go' ,label = "B points")
plt.plot(cx, cy,'ro' ,label = "C points")
plt.plot(dx, dy,'bo' ,label = "C points")
plt.plot(y, x,'b--' ,label = "x = 0")
plt.plot(x,2*x, 'r--', label = "y = 2x")
plt.title("Plot of B,C,D points")
plt.axis([-1.5,1.5,-1.5,1.5])
plt.legend()
```

```
The B points: [(0, 1), (0, -1), (0.447, 0.8944), (-0.447, -0.8944)]
The C points: [[0.5, 1], [0, 1], [0, -1], [-0.5, -1]]
The D points: ((0.0, 1.0), (0.0, -1.0), (0.4924790605054523,
0.9849581210109046), (-0.4924790605054523, -0.9849581210109046))
```

[ ]: `<matplotlib.legend.Legend at 0x7fd48ee25e10>`



Created in Deepnote