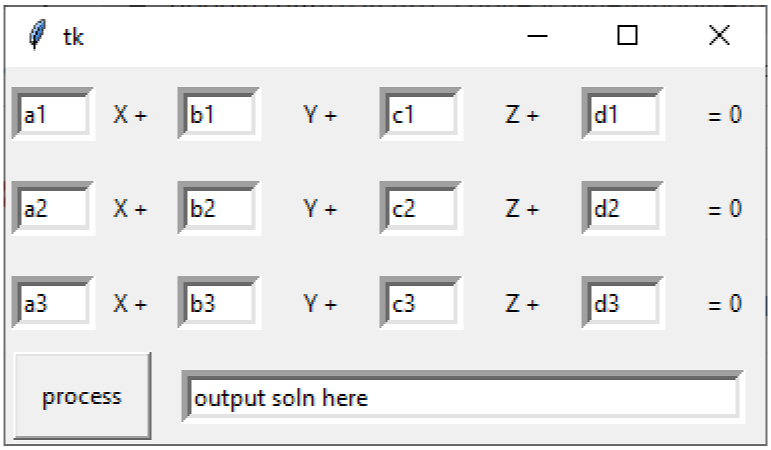
Systems of Linear Equations in Three Dimensions

Your assignment:

You are to put together a windows GUI to take in the equations of three planes



You are then to solve the system using cramer's rule[1] should it have a solution. You will plot the planes in 3-d to support your answer. The code to plot your planes is provided below along with a brief explanation of plane geometry and a few other helpful hints.

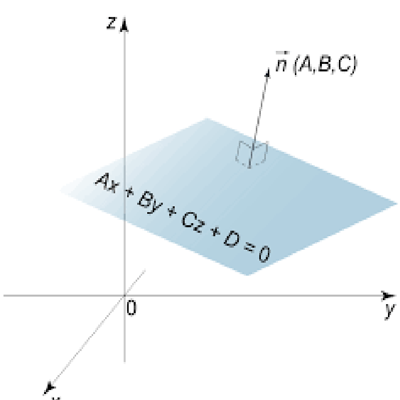
You shall be required to create a separate Plane class that will hold at least the normal.

Due: Friday Tuesday November 1st. NoIfsAndsOrButs

Consider the following linear system:

$$a_1x + b_1y + c_1z + d_1 = 0$$
$$a_2x + b_2y + c_2z + d_2 = 0$$
$$a_3x + b_3y + c_3z + d_3 = 0$$

This could be viewed as three equations with three unknowns *x, y* and *z*



Please note that the equation $a_1x + b_1y + c_1z = d_1$ is the equation of a plane with normal (a_1, b_1, c_1).

A normal is the direction perpendicular to the plane in 3 space. Vector notation is often ascribed to this value as $\vec{n} = (a_1, b_1, c_1)$. This would mean normal vector $\vec{n}$. Vectors have magnitude and direction.

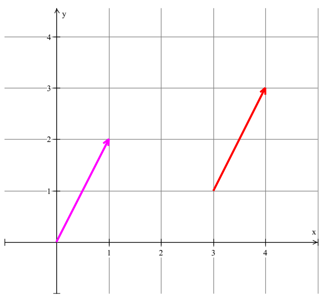
[1] https://www.youtube.com/watch?v=Ot87qLTODdQ

The direction being given by the direction vector is relative to the segment with tail at the origin, $(0,0,0)$ and head at $(a_{1,}b_1, c_1)$. Please note that direction does not change and need not necessarily be literally at or including the origin. Should you add $(a_{1,}b_1, c_1)$ to any pt (x, y, z) you will move in the direction $(a_{1,}b_1, c_1)$ from (x, y, z) to the point $(a_1 + x, b_1 + y, c_1 + z)$ *we add the components

Consider the following example in 2-space.

Say we have direction vector $(1,2)$, where we go right 1 and up 2 from the origin. This may be applied to any point by adding components, i.e., from arbitrary point $(3,1)$, we would still go right 1 and up 2 to the point $(4,3)$.

The direction remains the same.



Please note that it has the same effect in 3-space.
As for magnitude (which we needn't know for this assignment), it is just the length of the line segment, which in this case is pythogoras in three space (which works!)

magnitude = $\sqrt{(a_1)^2 + (b_1)^2 + (c_1)^2}$

There is quite a bit more to the nuances of this topic, however, this is the long and the short of it. To place a plane anywhere in three space, we "dot" the normal with a point on the plane and solve for a unique value of "d" in our equation.

When we say dot, we mean dot product. Say we have point (x, y, z) and normal $(a_{1,}b_1, c_1)$, the dot product of the two would be the sum of the product of its components, or

$(x, y, z) \cdot (a_{1,}b_1, c_1) = a_1x + b_1y + c_1z$

Writing our equation as $a_1x + b_1y + c_1z + d = 0$, then solving for d is easy

Once we have solved for *d*, the equation of the plane is written with defined coefficients representing the normal , yet *x, y and z* remain letters. Note the when we put together the equation of a line we define the slope and y-intercept, here we define our coefficients and d.

Helpful Code:

Definitely review Basic Tkinter lesson

Setting up an entry field:

# creation of entry fields
a1 = Entry(root, width=5, borderwidth=5)

# output placement of entry field by row and column
a1.grid(row=0, column=0, padx=3, pady=10)

# We can have deault text in the input field to help the user enter info
a1.insert(0, "a1")

the " X + " and " = 0 " are labels

****

plotting planes:
#sample program should be copy/paste ready

```
#numpy is number python package
import numpy as np
#matplotlib  is for plotting math stuff
import matplotlib.pyplot as plt

#this is actually new to me
#we no longer use the .gca you will see
#commented out below and this is a new way of
#plotting 3d.  A recent change in replit
from mpl_toolkits.mplot3d import axes3d,
Axes3D
fig = plt.figure()


#a point on the plane
point  = np.array([8, 4, 5])
#the normal to the plane
#normal is a direction perpendicular to the
#plane
normal = np.array([2, 6, 3])

#equation of a plane is Ax + By +Cz + D = 0
# the dot product of two vectors (a,b,c) and
#(d,e,f)
# is denoted (a,b,c)dot(d,e,f) = ad+be+cf  (a
#scalar quatity)
# therefore figure out below

# dash is a negative sign
d = -point.dot(normal)



#give us a grid 0-20,0-20 x y resp
xx, yy = np.meshgrid(range(20), range(20))
```

```
#our normal is being refferenced in an array
#in Python and the first elt starts at zero
#therefore we have just isolated for z and
#assigned all zz coordinates
#in quite a slick way to match all x and y
values #from 0 - 20 should they exist
#isolate z from equation Ax+By+Cz+D=0  for
#below
zz = (-normal[0] * xx - normal[1] * yy - d) *
1. /normal[2]

#syntax to plot our plane with the given
#coordinates
#xx,yy,zz are just ALL the points we are to
plot

#plt3d = plt.figure().gca(projection='3d')
#plt3d.plot_surface(xx, yy, zz)

# Add an axes
ax = fig.add_subplot(111,projection='3d')

# plot the surface
ax.plot_surface(xx, yy, zz, alpha=0.2)

#plt.axes()

#whithout the following line nothing shows
plt.show()
```

https://replit.com/@andrewJJimenez/planes-assignment#main.py

Lastly, here is some code featuring a button w/functionality

```
import matplotlib.pyplot as plt
import numpy as np

from myFirstClass import LineClass

temp = LineClass

temp.isVertical = False

from tkinter import *

root = Tk()
root.title = ("Lines")


# this function would allow you to pull data from first_X field
# currently it is unused, and unneeded for this assignment
def enteredVal():
    return


# creation of entry fields
first_X = Entry(root, width=5, borderwidth=5,
command=enteredVal())
first_Y = Entry(root, width=5, borderwidth=5)
second_X = Entry(root, width=5, borderwidth=5)
second_Y = Entry(root, width=5, borderwidth=5)

# creation of output field
typeOfLine = Entry(root, width=45, borderwidth=5, )

# output placement of entry field by row and column
first_X.grid(row=0, column=0, padx=10, pady=10)
first_Y.grid(row=0, column=1, padx=10, pady=10)
second_X.grid(row=0, column=2, padx=10, pady=10)
second_Y.grid(row=0, column=3, padx=10, pady=10)

# output placement of output field by row and column
typeOfLine.grid(row=2, column=0, columnspan=4, padx=10,
pady=10)

# We can have deault text in the input field to help the user enter
#info
first_X.insert(0, "x1")
first_Y.insert(0, "y1")
second_X.insert(0, "x2")
second_Y.insert(0, "y2")

# this procedure will run if our button is clicked
# note that we must code this ahead of our button as
# out button will be created calling this procedure, so
# if we declare the button first it will not know what
# "button click" is
def button_click():
    temp.x1 = int(first_X.get())
    temp.y1 = int(first_Y.get())
    temp.x2 = int(second_X.get())
    temp.y2 = int(second_Y.get())

    if (temp.x1 - temp.x2 == 0 and temp.y1 - temp.y2 != 0):
        typeOfLine.delete(0, END)
        typeOfLine.insert(0, "this line is vertical")
        # print ("this line is vertical")
    elif (temp.y1 - temp.y2 == 0 and temp.x1 - temp.x2 != 0):
        typeOfLine.delete(0, END)
        typeOfLine.insert(0, "this line is horizontal")
        # print ("this line is horizontal")
    else:
        # compute eqn of the line and plot
        typeOfLine.delete(0, END)
                  eqn = str((temp.y2 - temp.y1) / (temp.x2 -
            temp.x1)) + "x + " +str(temp.y1 - temp.x1 * (temp.y2 -
            temp.y1) / (temp.x2 - temp.x1))
        typeOfLine.insert(0, "the equation of the line is: y = " + eqn)
        x = np.linspace(-5, 5, 100)
                  y = (((temp.y2 - temp.y1) / (temp.x2 - temp.x1))) *
            x + ((temp.y1 - temp.x1 * (temp.y2 - temp.y1) /
            (temp.x2 - temp.x1)))
        plt.plot(x, y, '-r', label="y=" + eqn)
        plt.title("Graph of y=" + eqn)
        plt.xlabel('x', color='#1C2833')
        plt.ylabel('y', color='#1C2833')
        plt.legend(loc='upper left')
        plt.grid()
        plt.show()

    return #not really nec

# define button #s
button_1 = Button(root, text="process", padx="10", pady="10",
command=button_click)  # keep prentheses off
# command button_click
# put buttons on the screen by row and column
button_1.grid(row=1, column=0)

root.mainloop()
```

****

```
class LineClass:  #here is my LineClass in a sep file.  Right click on proj folder "new python file"
    def __init__(self, x1,y1,x2,y2,isVertical,isHorizontal,isRegular):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.isVertical = isVertical
        self.isHorizontal = isHorizontal
        self.isRegular = isRegular
```

https://replit.com/@andrewJJimenez/WigglyGrimyRelationalmodel#main.py ←handy link to the above code