Homework 2 writeup

Name: Zach Gendreau

Section: B

Problem 1

In [ ]:
```
#y1
y_1 = 0
for k in range(100000) :
    y_1 += 0.1

A6 = np.absolute(10000 - y_1)

#y2
y_2 = 0
for k in range(100000000) :
    y_2 += 0.1

A7 = np.absolute(y_2 - 1000000)

#y3
y_3 = 0
for k in range(100000000) :
    y_3 += 0.25

A8 = np.absolute(25000000 - y_3)

#y4
y_4 = 0
for k in range(100000000) :
    y_4 += 0.5

A9 = np.absolute(y_4 - 50000000)
```

Part a

In Problem 2 of the coding portion of the homework, I found the following values for x_1, x_2, x_3, and x_4.

x_1	x_2	x_3	x_4
0.0000000188483	0.0188705	0.0	0.0

Part b

From the answers in part (a), we can see that floating point roundoff error has occurred. Theoretically, all four of these numbers should be 0, however experimentally, they are not. I would guess that x1 and x2 have the error because .1 or 1/10 cannot be written as a base 2 exponent easily. x2 is larger than x1 because it is compounded 1,000,000 times rather than 10,000 times. x3 has no error because we are adding .25, which can be represented as 2^-2 by our machine. Nomatter how many times we add it together, the answer will be exact(unlike x2).

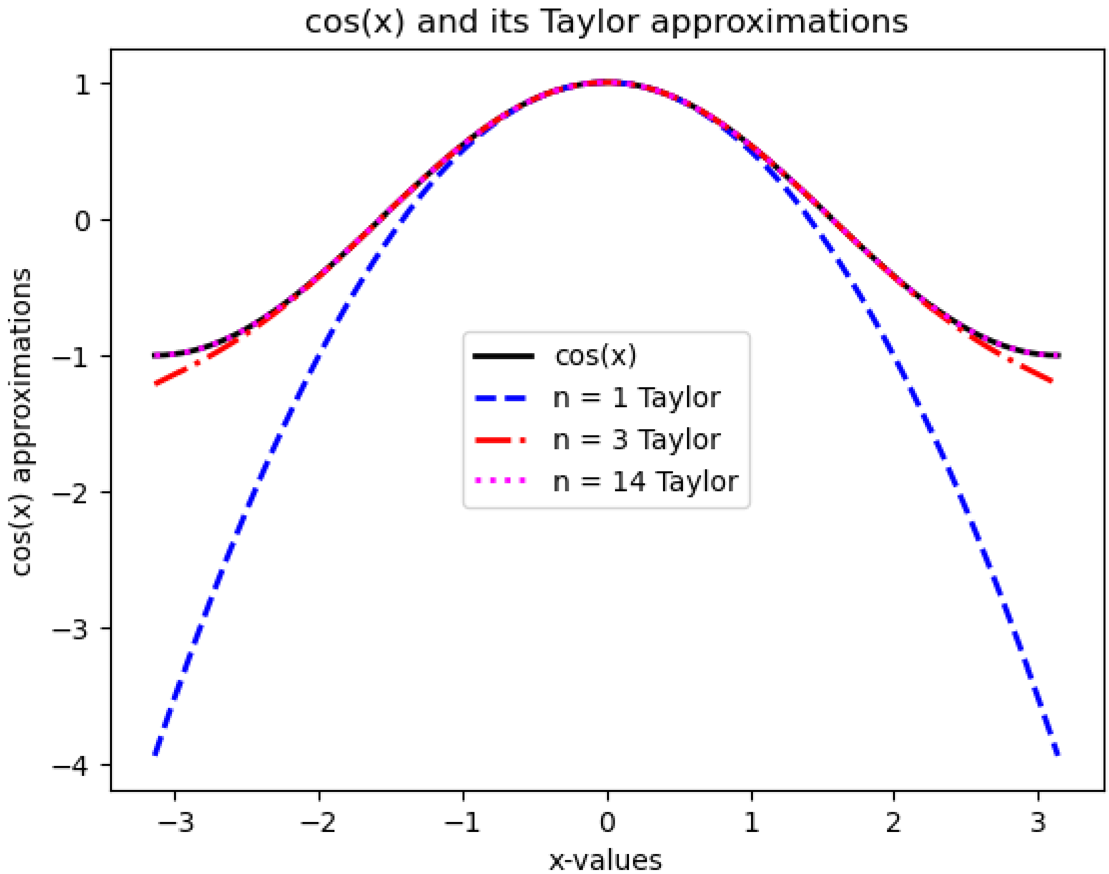
Part c

As explained in part (b), I would guess x3 and x4 show no error because both .25 and .5 can be represented exactly in binary (2^-2 and 2^-1, respectively). This means that nomatter how many times we compute repetetive addition, the computed answer will always align exactly with the theoretical answer.

Problem 2

In [3]:

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

x = np.linspace(-np.pi,np.pi,100)
y = np.cos(x)
plt.plot(x,y,color = "black",linewidth = 2,label = "cos(x)")

#Plot first order Taylor polynomial
def taylor_func(x, dim):
    temp = 0
    for l in range(dim + 1) :
       temp += (((-1)**l)/(math.factorial(2*l))) * (x**(2*l))
    return temp

#Plot first order Taylor polynomial
z = taylor_func(x, 1)
plt.plot(x,z,color = "blue",linestyle = "dashed",linewidth = 2,label = "n = 1 T

#Plot third order Taylor polynomial
p = taylor_func(x, 3)
plt.plot(x,p,color = "red",linestyle = "dashdot",linewidth = 2, label = "n = 3

#Plot fourteenth order Taylor polynomial
q = taylor_func(x, 14)
plt.plot(x,q,color = "magenta",linestyle = "dotted",linewidth = 2,label = "n =

plt.title("cos(x) and its Taylor approximations")
plt.xlabel("x-values")
plt.ylabel("cos(x) approximations")
plt.legend(loc="center")
plt.show()
```


cos(x) and its Taylor approximations
1
0
−1
−2
−3
−4
cos(x) approximations
−3
−2
−1
0
1
2
3
x-values
cos(x)
n = 1 Taylor
n = 3 Taylor
n = 14 Taylor

In [ ]: