AMATH 301 Homework 2 Writeup

Jaiden Atterbury - Section B

01-16-2023

Problem 1:

Part (a):
Q: Record the values of x_1, x_2, x_3, and x_4 you found in Coding problem 2 and rank them in order from smallest to largest.

A: As found in Coding problem 2, $x_1 = 1.8848368 \times 10^{-8}$, $x_2 = 0.0188705$, $x_3 = 0$, and $x_4 = 0$. Thus, in order from smallest to largest we have: x_4, x_3, x_1, and x_2.

Part (b):
Q: Discuss what you found in part (a). Why do you think some are larger than others? Are there any outliers to your conclusion?

A: In part (a), we see that there are two values that are exactly zero, and two values that are greater than zero. In particular, $x_1, x_2 > 0$ and $x_3, x_4 = 0$. Notice that even though x_1 and x_2 were found by using a for loop to sum the same term, we can see that $x_1 < x_2$. This happens because there are more iterations of the for loop in x_2 with the term we are summing staying the same for both x_1 and x_2. In general, the more iterations of the for loop, the larger the total truncation error is going to be. However, x_3 and x_4 are major outliers to this theory because they have the same amount of iterations as x_2, yet they are both zero while x_2 is not. The reason why x_2 is not exactly zero while x_3 and x_4 are, comes down to the fact that computers store information in base 2. The difference between x_2 and x_3, x_4 will be discussed further in part (c) and will shed some light on why the fact that computers store information in base 2 matters when it comes to truncation error.

Part (c):
Q: Which of your answers is exactly zero. Can you guess why that is the case?

A: As shown in part (a), the x_i's that are exactly zero are x_3 and x_4. In order to see why these x_i's are exactly zero and the others aren't, we have to look at the corresponding term values that were added together in the for loop. The term value corresponding to x_3 is $0.25 = \frac{1}{4} = 2^{-2}$ and the term value corresponding to x_4 is $0.5 = \frac{1}{2} = 2^{-1}$. The reason why these term values being of the form 2^{-k} is important is because computers store information in base 2. Thus, computers can store the exact value of any decimal that can be represented as a fraction by $\frac{m}{2^k}$. Where m,k are any integer values.

Code Used:

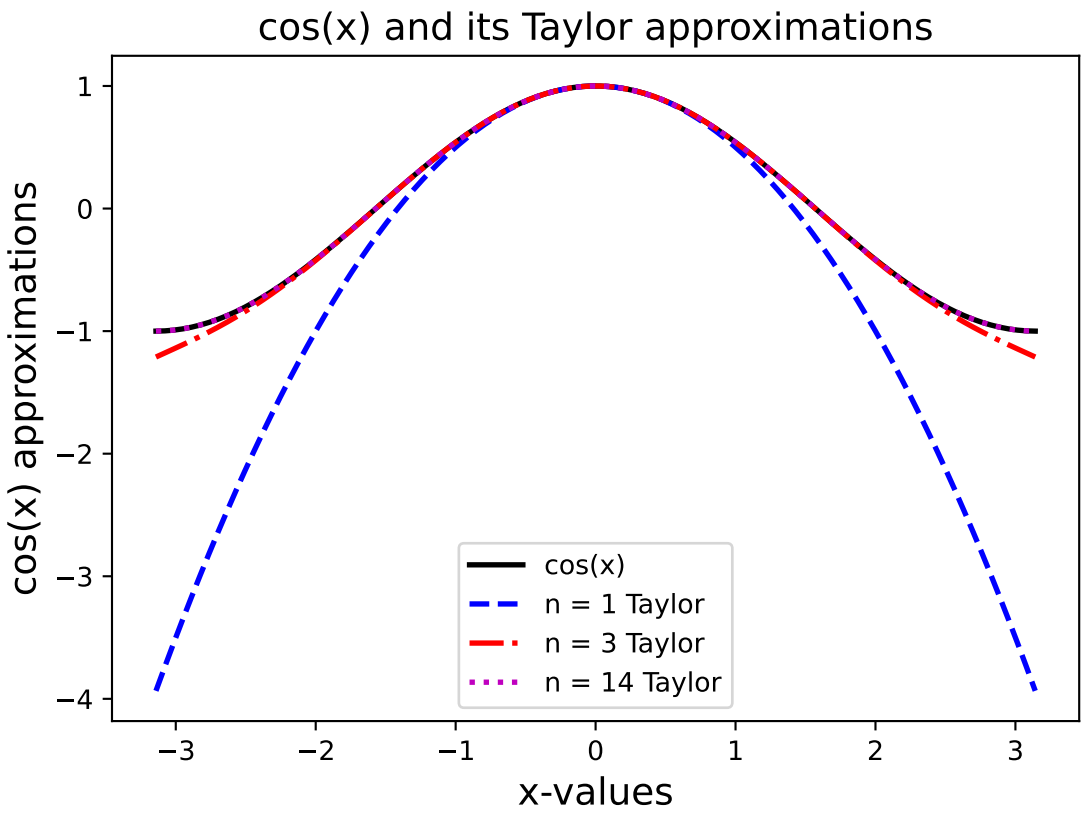
```
# Part (a):
y1 = 0
term = 0.1
for k in range(100000):
    y1 += term

y2 = 0
y3 = 0
y4 = 0
```

```
term1 = 0.1
term2 = 0.25
term3 = 0.5

for k in range(100000000):
    y2 += term1
    y3 += term2
    y4 += term3

# Part (b):
x1 = abs(10000 - y1)
x2 = abs(y2 - 10000000)
x3 = abs(25000000 - y3)
x4 = abs(y4 - 50000000)
```

Problem 2:

Description:
Create a plot of cosine on the interval $[-\pi, \pi]$ and plot the Taylor-series approximations for $n = 1, 3, 14$.

Plot:



Code used:

```
def get_cos_taylor(array, order):
    y_values = np.zeros(100)
    for k in range(order + 1):
```

```
        y_values += ((-1) ** k / np.math.factorial(2 * k)) * (array ** (2 * k))
    return y_values

x = np.linspace(-1 * np.pi, np.pi, 100)

# Part (i):
y = np.cos(x)
plt.plot(x, y, color="k", linewidth=2)

# Part (ii):
y = get_cos_taylor(x, 1)
plt.plot(x, y, color="b", linestyle="--", linewidth=2)

# Part (iii):
y = get_cos_taylor(x, 3)
plt.plot(x, y, color="r", linestyle="-.", linewidth=2)

# Part (iv):
y = get_cos_taylor(x, 14)
plt.plot(x, y, color="m", linestyle=":", linewidth=2)

# Part (v):
plt.xlabel("x-values", fontsize=14)

# Part (vi):
plt.ylabel("cos(x) approximations", fontsize=14)

# Part (vii):
plt.title("cos(x) and its Taylor approximations", fontsize=14)

# Part (viii):
plt.legend(("cos(x)", "n = 1 Taylor", "n = 3 Taylor", "n = 14 Taylor"))

plt.show()
```