Math340 HW

Programming in Mathematics

Due October 25, 2024

Department of Mathematics and Statistics

San Diego State University

Fall 2024

Your Name: Vince Verdugo

Your ID: 826107601

1. Summary

Question 1: I used the equation for Taylor series approximations to find the first and fourth order Taylor series approximations of the given function, $f(x) = e^x$. I then calculated the relative error of the two approximations against the true function. After I plotted both approximations and both relative errors

Question 2: Using the two equations $y_c = y_0 e^{\sigma t}, y_p = (y_0 + \epsilon)e^{\sigma t}$, with $\sigma = 2$, I determined the growth rate, σ, numerically using the equation $\frac{ln(\frac{y_{n+1}}{y_n})}{dt}$ using y_c. I also calculated σ using the equation $\frac{ln(\frac{y_p - y_c}{\epsilon})}{dt}$, with $dt = 0.001, dt = 1$. I then calculated the average value and standard deviation for each method to determine which was the best at finding the value of σ

Question 3: I used the `random.choice` function to output three odd and three even double digit numbers.

Question 4: I explored the concept of mutable and immutable objects in Python, discussed the differnece between Python's 'int' and NumPy's 'int', and using the given code analyzed the differences between using normal and inplace operation on an integer in Python.

2. Methodology

For question 1, I imported `numpy`, `matplotlib.pyplot`, and `math` packages. I defined 'x' using `np.linspace`, f(x) using the def function and used the `math.factorial` function to define the first and fourth order Taylor Series approximations, `tseries1` and `tseries4`. I used `np.abs` to determine the relative error of the true function and the Taylor Series approximations. I used `matplotlib.pyplot` to plot the four panel plot using `subplot`, `plot`, `title`, `legend`, `tight_layout`, and `show` functions.

For question 2, I used np.arange to define t, with dt=0.001. I then defined `sigma` and `eps` representing σ ϵ in our function. I used the `np.exp` function when defining arrays `yc` and `yp` to the respective functions. I used `y[1:]/y[:-1]` to represent $\frac{y_{n+1}}{y_n}$. I also used `np.mean` and `np.sqrt(np.var())` functions to calculate the average and standard deviation of the computed values for each method.

For question 3, I used `np.arange` to create two arrays of double digit values, one only odds and one only evens. I then used the `list()` function to convert them to lists so that I could use the `remove()` function to ensure uniqueness. I then used a for loop to randomly select three values from each list.

For question 4, I used markdown to discuss and analyze the given code and concepts

3. Code and Results (e.g., Tables, Figures, Outputs)

Results for Question 1

Taylor Series

$$f(x) = f(a) + f'(a)\frac{(x-a)}{1!} + f''(a)\frac{(x-a)^2}{2!} + f'''(a)\frac{(x-a)^3}{3!} + \dots f^n(a)\frac{(x-a)^n}{n!}$$

$$f(x) = \sum f^n(a)\frac{(x-a)^n}{n!}$$

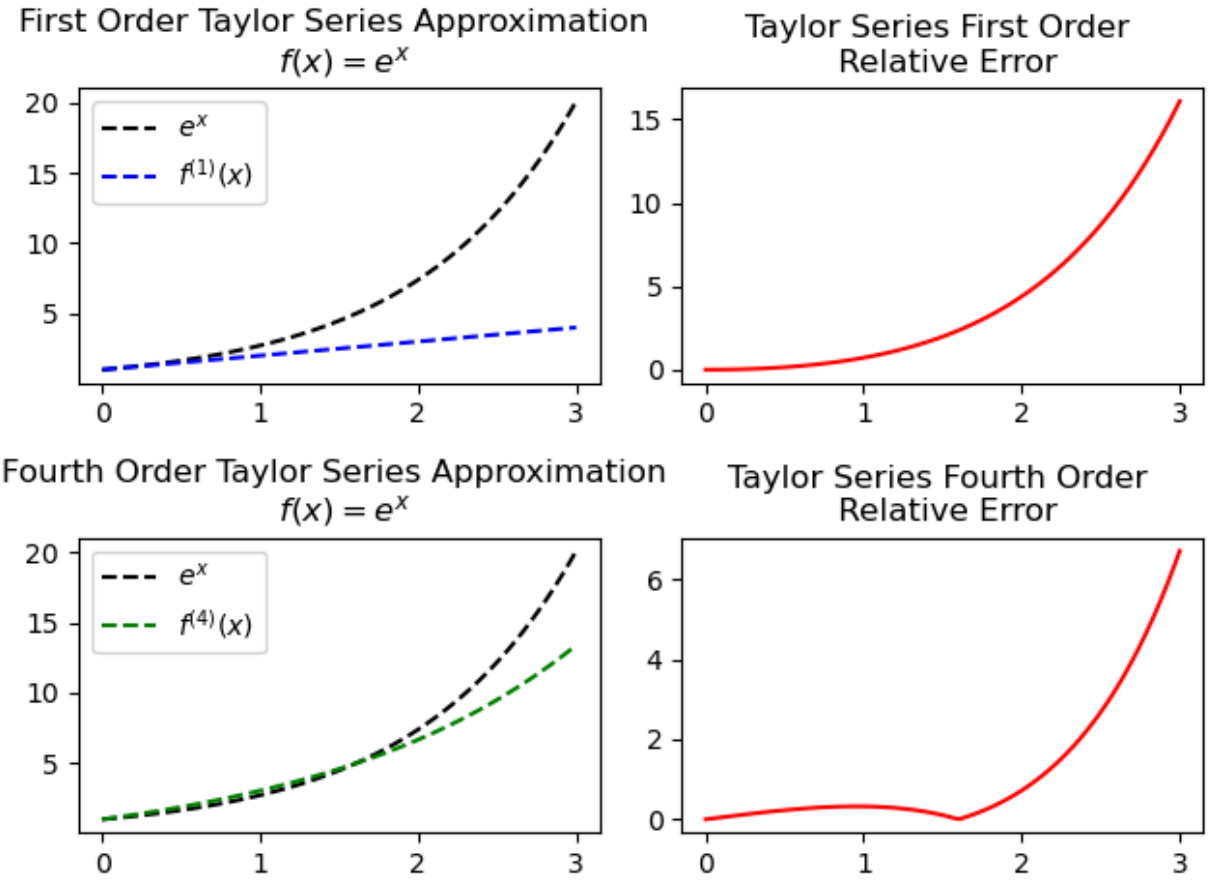
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
#x within [0,3]
x = np.linspace(0,3,100)
```

```
#Define f(x) = e^x
#Not going to define f^n function because f(x) = f'(x) = f''(x) = ... = f^n(x)
def f(x):
    return np.exp(x)

#Assuming we are approximating around a = 0 (?)
#f^n(a) = e^(0) = 1
tseries1 = 1 + (x/math.factorial(1))
tseries4 = tseries1 + (x**2/math.factorial(2)) \
           + (x*3/math.factorial(3)) + (x**4/math.factorial(4))

#Calculate relative error for f^(1) and f^(4)
re1 = np.abs(f(x)-tseries1)
re4 = np.abs(f(x)-tseries4)
```

In [2]:
```
#Four planel plot
#Panel (a)
plt.subplot(2,2,1)
plt.plot(x, f(x), 'k--', label=r'$e^{x}$')
plt.plot(x, tseries1, 'b--', label=r'$f^{(1)}(x)$')
plt.title('First Order Taylor Series Approximation \n $f(x) = e^{x}$')
plt.legend()
#Panel (b)
plt.subplot(2,2,2)
plt.plot(x, re1, 'r')
plt.title('Taylor Series First Order \n Relative Error')
#Panel (c)
plt.subplot(2,2,3)
plt.plot(x, f(x), 'k--', label=r'$e^{x}$')
plt.plot(x, tseries4, 'g--', label=r'$f^{(4)}(x)$')
plt.title('Fourth Order Taylor Series Approximation \n $f(x) = e^{x}$')
plt.legend()
#Panel (d)
plt.subplot(2,2,4)
plt.plot(x, re4, 'r')
plt.title('Taylor Series Fourth Order \n Relative Error')
#Format and show
plt.tight_layout()
plt.show()
```



Results for Question 2

In [3]:
```
#Define t, dt, and sigma
dt = 0.001
t = np.arange(0+dt,10+dt,dt) #For 2b, cannot divide by t when t=0, so start at
sigma = 0.2
#Define initial conditions
y0 = 1
eps = 1e-5
#Define our functions
yc = y0*np.exp(sigma*t)
yp = (y0+eps)*np.exp(sigma*t)
```

2a, 2b

In [4]:
```
#Determine growth rate using y_c
gr1 = np.log(yc[1:]/yc[:-1])/dt
#Compute ln((y_p - y_c)/epsilon)/t
gr2 = np.log((yp-yc)/eps)/t
```

2c

In [5]:
```
#Redefine dt=1
dt2 = 1
#Repeat previous calculations
gr3 = np.log(yc[1:]/yc[:-1])/dt2

#Compare values
```

```
print('Growth rate determined numerically using dt = 0.001:', gr1)
print('Growth rate determined numerically using dt = 1:', gr3)
print('Difference =', np.abs(gr3-gr1))
```

```
Growth rate determined numerically using dt = 0.001: [0.2 0.2 0.2 ... 0.2 0.2
0.2]
Growth rate determined numerically using dt = 1: [0.0002 0.0002 0.0002 ... 0.0
002 0.0002 0.0002]
Difference = [0.1998 0.1998 0.1998 ... 0.1998 0.1998 0.1998]
```

2d

In [6]:
```
#Mean of each growth rate calculation
mu1 = np.mean(gr1)
mu2 = np.mean(gr2)
mu3 = np.mean(gr3)

sd1 = np.sqrt(np.var(gr1))
sd2 = np.sqrt(np.var(gr2))
sd3 = np.sqrt(np.var(gr3))

print(f'Average growth rate for first solution: {mu1}\nAverage growth rate for
print(f'\nStandard Deviation for first solution: {sd1}\nStandard Deviation for
```

```
Average growth rate for first solution: 0.20000000000000712
Average growth rate for second solution: 0.20000000000662713
Average growth rate for third solution: 0.0002000000000000072

Standard Deviation for first solution: 1.5656852789801242e-13
Standard Deviation for second solution: 9.483044133873254e-11
Standard Deviation for third solution: 1.565693786272338e-16
```

As we can see the average of the first solution is more accurate to the actual value than the rest of the values as well as having smaller standard deviation than the other methods of computation.

Results for Question 3

In [7]:
```
import random
#Use np.arange to create arrays of all double digit odds and double digit evens
#Convert arrays to lists to use list.remove function
odds = list(np.arange(11,100,2))
evens = list(np.arange(10,100,2))

for i in range(0,3):
    #Use random.choice to choose three evens and three odds
    #Remove each choice to ensure that each number chosen is unique
    x = random.choice(odds)
    odds.remove(x)
    y = random.choice(evens)
    evens.remove(y)
    print(x,y)
```

```
99 56
17 70
41 72
```

Results for Question 4

4a

In Python, a mutable object is one that can be changed after it is created. An example of this is a list. If I were to declare an empty list, `newlist`, I could easily change the list without making new copies of the list using functions like this:

```
newlist = []
newlist.append(2)
newlist.append(4)
newlist.remove(2)
```

We have added and removed values directly to the list and now `newlist = [4]`

An immutable object in Python is an object who cannot be changed once it has been created. The most common example of these would be integers, float, boolean, and strings. For example, if we had the string `str` if we wanted to change the value of the string like this:

```
str = "Hello!"
str2 = str.replace("Hello", "Bye")
```

We have to create another string value, `str2`, because the string is immutable. So now `str = "Hello!"` and our copy `str2 = "Bye!"`

4b

There are several differences between Python's 'int' and NumPy's 'int'. One difference is Python's built in 'int' is flexible meaning that it can expand to accomodate numbers that may require more bytes, where NumPy's cannot. Another difference is that NumPy 'int's have more attributes and methods thatn Python 'int's

4c

Referencing this code

```
a = 1

a = a + 1
a += 1
```

The difference between the second and third lines are how the data is stored. In the second line, by calling `a = a + 1`, we are creating another object also named `a` that has the value of 2. In the third line, by calling `a += 1`, instead of creating another instance, we are updating the orginal object `a` so that `a=2`