Numerical Differentiation

submitted by and coded by Group ICHI:

Aquiro, Freddielyn E.

Canoza, Cherrylyn

Joaquin, Christopher Marc

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

x = 0.1
dx = 0.02

f = lambda x : np.cos(2*x)+x**2/20+2.71828**-2*x
f_1 = lambda x: x/10-2*2.71828**-2*x-2*np.sin(2*x)
f_2 = lambda x: 4*2.71828**-2*x+1/10-4*np.cos(2*x)
f_3 = lambda x: -8*2.71828**-2*x+8*np.sin(2*x)


print(f'f(0.1) = {f(x)}')
print(f'f\'(0.1) = {f_1(x)}')
print(f'f\'\'(0.1) = {f_2(x)}')
print(f'f\'\'\'(0.1) = {f_3(x)}')
```

```
f(0.1) = 0.9941001243716443
f'(0.1) = -0.4144057546509279
f''(0.1) = -3.7661321252433555
f'''(0.1) = 1.481086274117268
```

Part 1: Central Finite Function

```
#Central Finite Function
print("Central Finite Differentiation")

def diff_cen(f,x,dx,degree=1):
    cen_eq1 = (f(x + dx) - f(x - dx))/(2*dx)
    cen_eq2 = (f(x+dx)-2*f(x)+f(x-dx))/dx**2
    cen_eq3 = (f(x+2*dx)-2*f(x+dx)+2*f(x-dx)-f(x-2*dx))/(2*dx**3)
    print(f'f(0.1) = {f(x)}')
    print(f'f\'(0.1) = {cen_eq1}, error @ {abs(f_1(x)-cen_eq1)}')
    print(f'f\'\'(0.1) = {cen_eq2}, error @ {abs(f_2(x)-cen_eq2)}')
    print(f'f\'\'\'(0.1) = {cen_eq3}, error @ {abs(24-cen_eq3)}')


diff_cen(f,x,dx)
```

```
Central Finite Differentiation
f(0.1) = 0.9941001243716443
f'(0.1) = -0.2518972477859066, error @ 0.16250850686502127
f''(0.1) = -3.8197436370665527, error @ 0.05361151182319723
f'''(0.1) = 1.588719006212491, error @ 22.41128099378751
```

Part 2: Forward and Backward Finite Function

```
#Forward Finite Function
print("Forward Finite Differentiation")
```

```
print("Forward Finite Differentiation")

def diff_fwd(f,x,dx,degree=1):
    fwd_eq1 = (f(x+dx)-f(x))/dx
    fwd_eq2 = (f(x+2*dx)-2*f(x+dx)+f(x))/dx**2
    fwd_eq3 = (f(x+3*dx)-3*f(x+2*dx)+3*f(x+dx)-f(x))/(dx**3)
    print(f'f(0.1) = {f(x)}')
    print(f'f\'(0.1) = {fwd_eq1}, error @ {abs(f_1(x)-fwd_eq1)}')
    print(f'f\'\'(0.1) = {fwd_eq2}, error @ {abs(f_2(x)-fwd_eq2)}')
    print(f'f\'\'\'(0.1) = {fwd_eq3}, error @ {abs(24-fwd_eq3)}')

diff_fwd(f,x,dx)
```

```
Forward Finite Differentiation
f(0.1) = 0.9941001243716443
f'(0.1) = -0.29009468415657214, error @ 0.12431107049435575
f''(0.1) = -3.7848338801166337, error @ 0.018701754873278187
f'''(0.1) = 2.0562331218554326, error @ 21.94376687814457
```

```
 #Backward Finite Function
print("Backward Finite Differentiation")

 def diff_bwd(f,x,dx,degree=1):
    bwd_eq1 = (f(x)-f(x-dx))/dx
    bwd_eq2 = (f(x)-2*f(x-dx)+f(x-2*dx))/dx**2
    bwd_eq3 = (f(x+3*dx)-3*f(x+2*dx)+3*f(x+dx)-f(x))/(dx**3)
    print(f'f(0.1) = {f(x)}')
    print(f'f\'(0.1) = {bwd_eq1}, error @ {abs(f_1(x)-bwd_eq1)}')
    print(f'f\'\'(0.1) = {bwd_eq2}, error @ {abs(f_2(x)-bwd_eq2)}')
    print(f'f\'\'\'(0.1) = {bwd_eq3}, error @ {abs(24-bwd_eq3)}')

diff_bwd(f,x,dx)
```

```
Backward Finite Differentiation
f(0.1) = 0.9941001243716443
f'(0.1) = -0.21369981141524108, error @ 0.2007059432356868
f''(0.1) = -3.8483826403651333, error @ 0.08225051512177783
f'''(0.1) = 2.0562331218554326, error @ 21.94376687814457
```

Part 3: Taylor Expansion

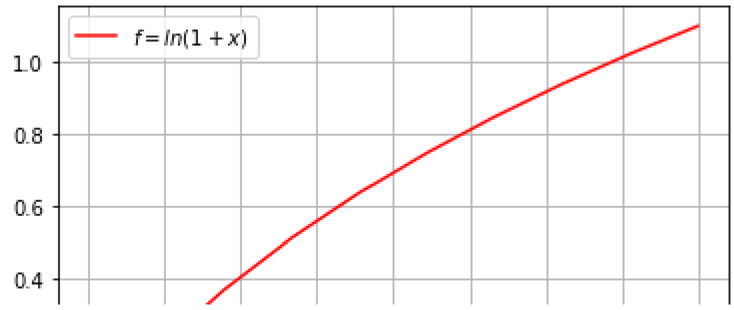
taylor expansion of ln(1+x)

$$ln(1+x) = \left(x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4}.....-+(-1)^n\frac{x^n+1}{n+1}\right)$$

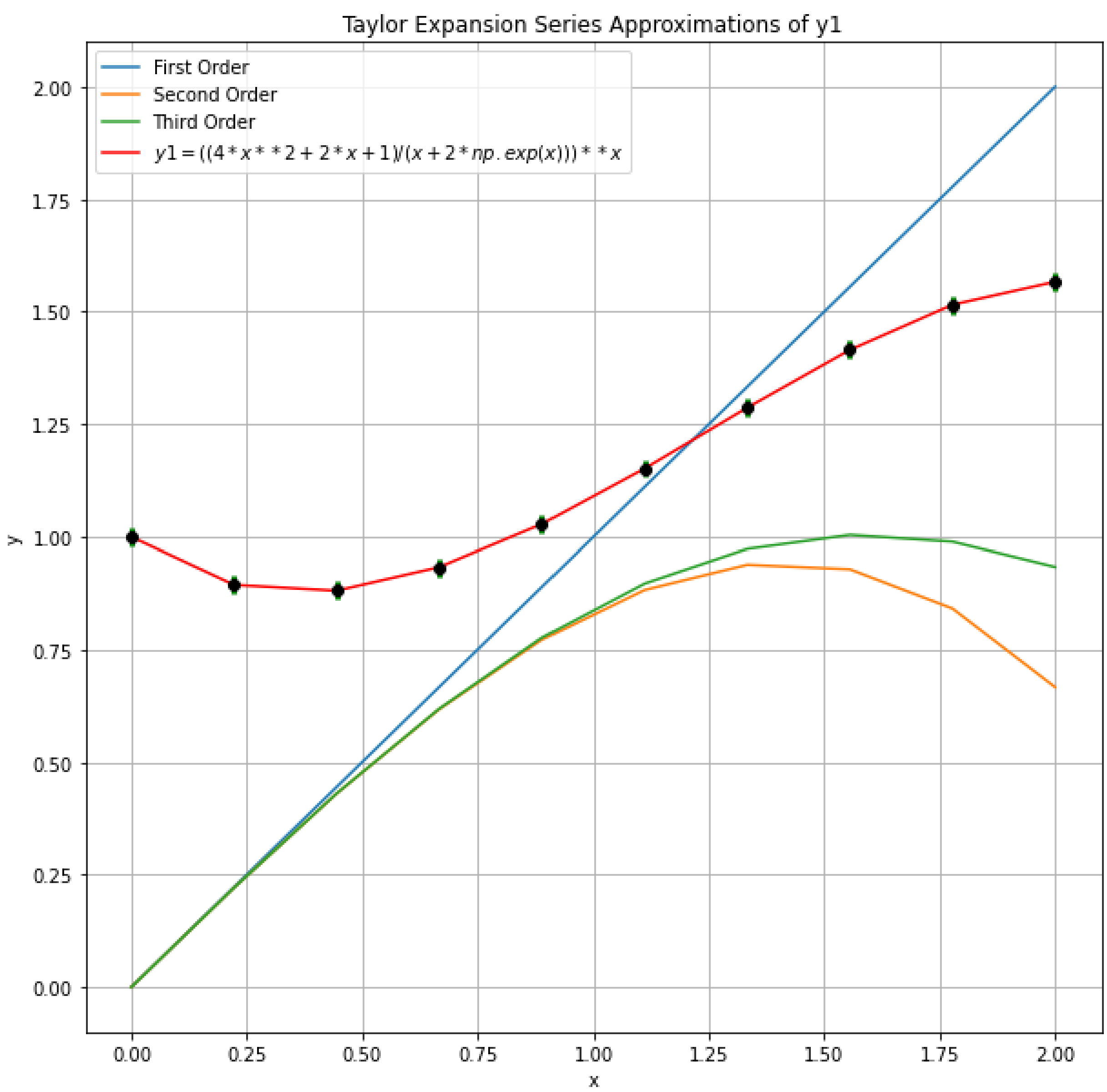
```
x = np.linspace(0,2,10)
y = np.zeros(np.size(x))
y1 = ((4*x**2 + 2*x + 1) / (x + 2 * np.exp(x))) ** x
y2 = np.cos(2*x) + (x**2/20) + np.exp(-2*x)
x0 = np.log(1+x)
```

```
plt.figure()
plt.plot(x,x0,'red',label='$f=ln(1+x)$')
plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f01eb7664d0>
```

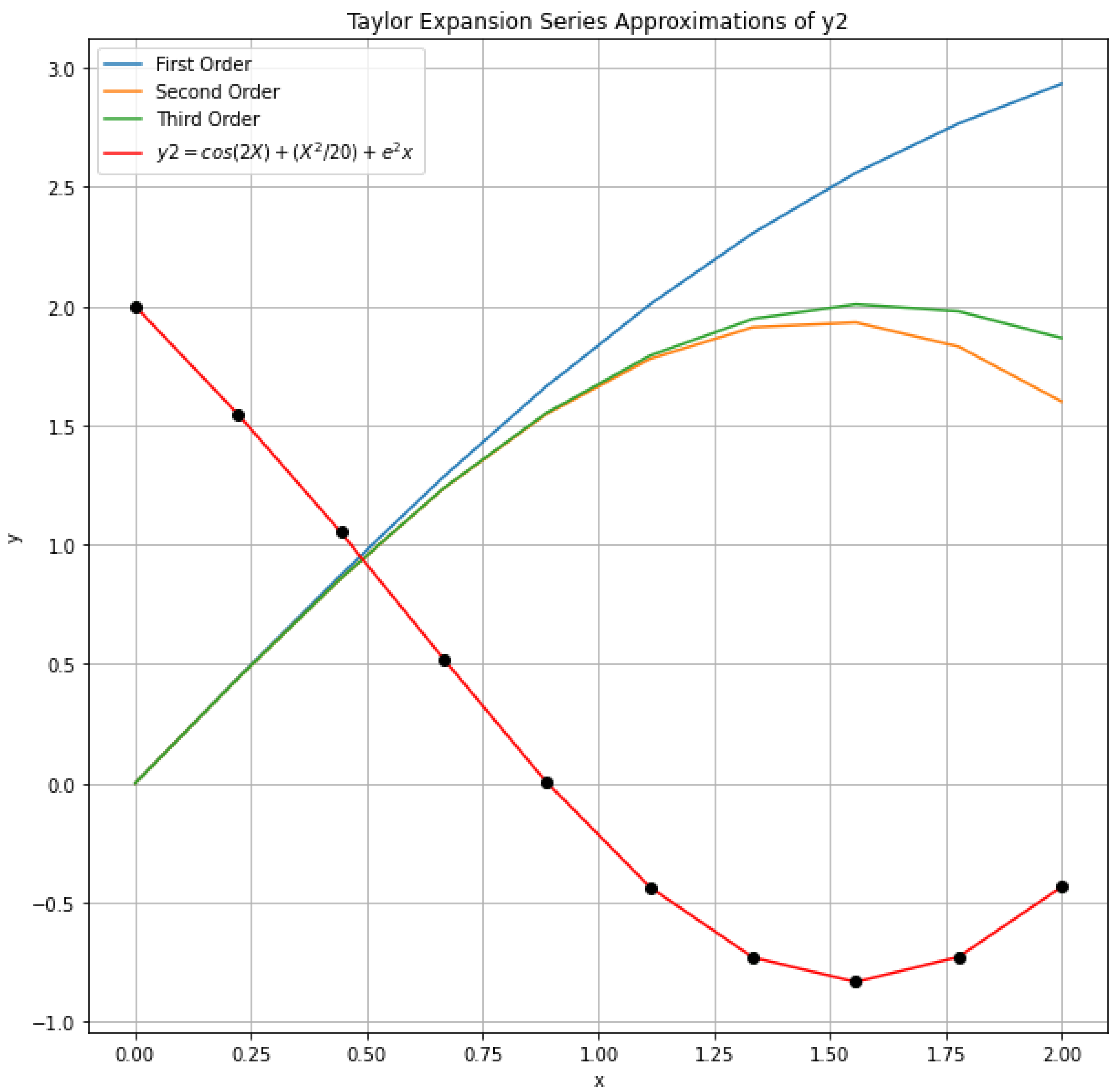



```
# Approximating values of y1
plt.figure(figsize = (10,10))
labels = ['First Order', 'Second Order' , 'Third Order']

for n, label in zip(range(3), labels):
    y = y + ((-1)**n * (x)**(2*n+1)) / np.math.factorial(2*n+1)
    plt.plot(x,y, label = label)
plt.plot(x, y1, 'red', label = '$y1= ((4*x**2 + 2*x + 1) / (x + 2 * np.exp(x))) ** x $')
plt.errorbar(x, y1 ,yerr=0.02, fmt='o', color='black',
             ecolor='green', elinewidth=3, capsize=0)
plt.grid()
plt.title('Taylor Expansion Series Approximations of y1')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```




```
# Approximating values of y2
plt.figure(figsize = (10,10))
labels = ['First Order', 'Second Order' , 'Third Order']

for n, label in zip(range(3), labels):
    y = y + ((-1)**n * (x)**(2*n+1)) / np.math.factorial(2*n+1)
```

```
    plt.plot(x,y, label = label)
plt.plot(x, y2, 'red', label = '$y2=cos(2X)+(X^2/20)+e^2x$')
plt.errorbar(x, y2 ,yerr=0.02, fmt='o', color='black',
             ecolor='green', elinewidth=3, capsize=0)
plt.grid()
plt.title('Taylor Expansion Series Approximations of y2')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



References

[1] Berkeley, 2021. " Approximations with Taylor Series — Python Numerical Methods" Available: [online]

✓ 0s completed at 8:43 PM