**Abstract**

This paper evaluates the accuracy of Euler's method and Taylor approximations of various orders. We found that high order Taylor approximations yield better results than Euler's method at the cost of determining derivatives of the differential equation. Finally we calculated a particular solution to the Gompertz equation for when a certain cancer achieved a lethal amount of cells. Which we calculated to be 55.4 months.

## 1 Introduction

In this assignment we seek to find the solutions of two different differential equations using Euler's method and Taylor approximations of various orders. We hope to validate the solutions of the numerical methods using a test function and then apply Euler's method to the Gompertz function. A real world application of differential equations, modeling the growth of cancer. Finally, we conduct an error analysis to determine the most accurate numerical method.

## 2 Methods

Our analysis begins with a test function.

$$y' = y + cos(t) - t^2, \quad 0 \leq t \leq 4, \quad y(0) = 1 \tag{1}$$

And it's analytic solution.

$$y(t) = \frac{1}{2}(4 - e^t + 4t + 2t^2 - cos(t) + sin(t)) \tag{2}$$

In order to analyze this function we used Euler's method, and second, fourth, and sixth order Taylor approximations. Euler's method is defined by equation 3.

$$y(t_{i+1}) = y(t_i) + h * f(t_i, y(t_i)) + \frac{h^2}{2} y''(\xi_i) \tag{3}$$

Euler's method works by beginning at an initial condition $y(t_i)$. We then solve the differential equation at the time $t_i$ and multiply it by a step size h. We then add this value to the initial condition. This yields a point $y(t_{i+1})$ which then becomes our "initial condition" for the next iteration. This is repeated for the desired number of steps on a given interval. The local error of Euler's method is $h^2$ while the global error is $h$. This makes the potential error quite high which is why in practice this method is rarely used.

We also analyzed our test function using Taylor approximations given by equation 4.

$$T^{(n)}(t_i, w_i) = f(t_i, w_i) + \frac{h^{n-1}}{n!} f^{(n-1)}(t_i, w_i) \tag{4}$$

Taylor approximations work very similarly to Euler's method in that they start at an initial condition and the particular solution to the differential equation is calculated at a time $t_i$, multiplied by a step size h and added to that initial condition. However, Taylor approximations use Taylor polynomials to reduce the truncation error. The order of the error of the Taylor approximation is determined by the number of terms added to the base differential equation $f(t_i, w_i)$. For example a 4th order Taylor approximation has an n=5 and a global truncation error of order $h^4$. The drawback is that we need to compute derivatives to construct the Taylor polynomial. This may not always be able to to be done analytically and using a numerical method to do so introduces more error.

In order to determine the relative error for our functions we used a common relative error scheme given by equation 5. Where x is the value calculated by the numerical method and y is the value of the analytic value of the solution to the differential equation given by equation 2.

$$error = \frac{|x-y|}{y} \tag{5}$$

Next we use Euler's method to calculate a particular solution to the Gompertz differential equation shown in equation 6.

$$N'(t) = \alpha \ln\left(\frac{K}{N(t)}\right) N(t) \tag{6}$$

In order to solve this equation we used Euler's method, an $\alpha = 0.0439$, $K = 12,000$, and $N(0) = 4,000$. We sought to find where $N(t) = 11,000$ cells. In order to do this I created a tuple and filled it with values of the differential equation evaluated at 1000 point. I did this by creating a vector and using it as one of the bounds for our Euler function and then iterating through the tuple until I found a value equal to 11,000 with in a tolerance of 10 cells.

## 3 Results and Analysis

We began our analysis by conducting a visual comparison of the different methods. As we can see in figure 1 the 6th order Taylor method matches most closely to the analytic solution of the differential equation. It is worth noting that the methods tended to perform worse as the rate of change of the function increased.

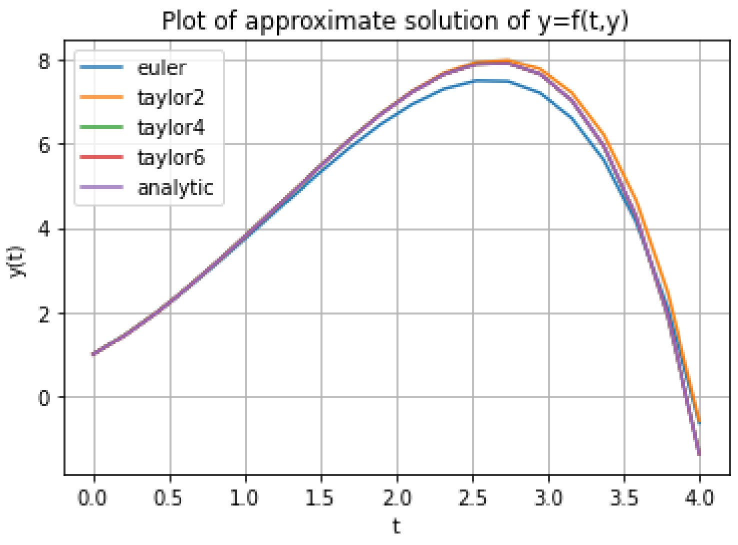

Figure 1: Solutions to trial function

Next we considered the error of the various functions based on step size. As we can see, generally the high order methods performed better than the lower order methods and as the step size increased the relative error decreased. A full list of the error values can be found in Appendix A.

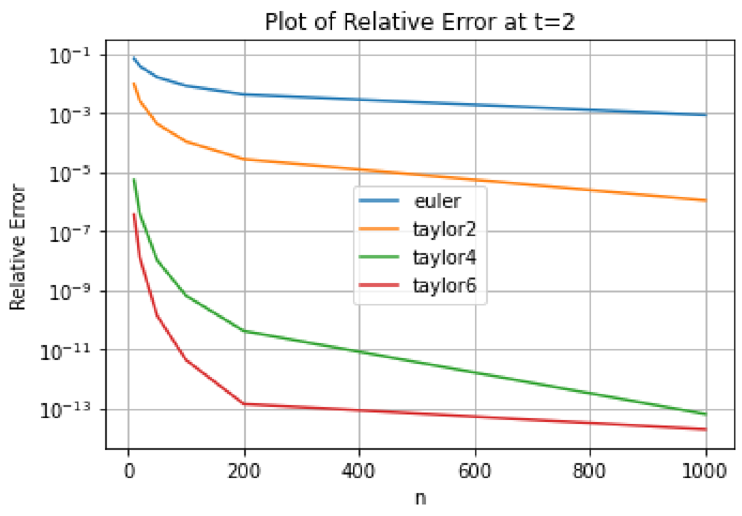

Figure 2: Error of the various methods

Finally, we analyze the Gompertz equation. I solved the Gompertz equation using Euler's method which revealed that it behaved as expected, in a logarithmic fashion. This makes sense because of the nature of what the Gompertz equation is modeling. In populations, including cancer cells, we expect to see a logarithmic growth, until the population can no longer sustain itself.

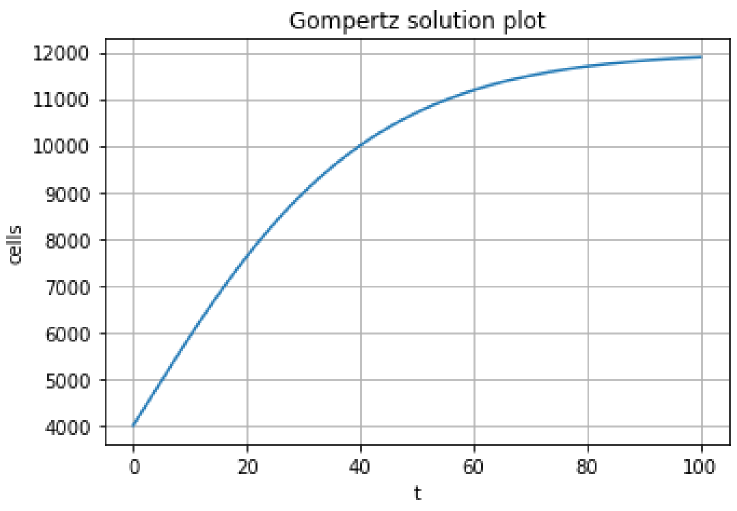

Figure 3: Solution to the Gompertz equation

Using the iteration method outlined in the methods section I found the lethal amount of cells to be present at:

$$t = 55.4 \ \ months \tag{7}$$

## 4 Conclusion

Through our analysis we were able to determine that higher order methods tended to more accurate than lower order methods. Additionally we explored several ways to solve differential equations and some of the pros and cons for each method. Finally we solved a real world equation and found the growth curve of a cancer model and that given our initial conditions the lethal number of cells occurred at 55.4 months.

## 5 Appendix A

This appendix contains the error of the test fuction for the various orders

| n | eulerErr | t2Err | t4Err | t6Err |
|---|---|---|---|---|
| **10** | 0.068599 | 0.009388 | 5.28E-06 | 3.56E-07 |
| **20** | 0.037705 | 0.002521 | 3.71E-07 | 1.24E-08 |
| **50** | 0.01605 | 0.000421 | 1.02E-08 | 1.36E-10 |
| **100** | 0.008201 | 0.000107 | 6.52E-10 | 4.33E-12 |
| **200** | 0.004146 | 2.68E-05 | 4.12E-11 | 1.39E-13 |
| **1000** | 0.000837 | 1.08E-06 | 6.33E-14 | 1.94E-14 |

Table 1:

## 6 Appendix B

```
    """
Created on Tue Nov  9 07:52:36 2021

@author: Ezra
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


def f(t,y):
    return y+np.cos(t)-t*t

def fa(t,y):
    return (1/2)*(4-np.exp(t)+4*t+2*t*t-np.cos(t)+np.sin(t))

def fp(t,y):
    return f(t,y)-np.sin(t)-2*t

def fdp(t,y):
    return fp(t,y)-np.cos(t)-2

def ftp(t,y):
    return fdp(t,y)+np.sin(t)

def fqp(t,y):
    return ftp(t,y)+np.cos(t)

def euler(n,a,b,alpha):
    '''
    Parameters
    ----------
    n : integer
        number of steps.
    a : float
        starting value for iVar.
    b : tuple
        ending value for iVar.
    alpha : float
        intial condition.

    Returns
    -------
    t : float
        time.
    w : float
        dVar.
    '''
    h=(b-a)/n
    t=a
    w=alpha
    for i in range(n):
        w=w+h*f(t,w)
        t=t+h
    return t,w

def taylor2(n,a,b,alpha):
    '''
    Parameters
    ----------
    n : integer
        number of steps.
    a : float
        starting value for iVar.
    b : tuple
        ending value for iVar.
    alpha : float
        intial condition.

    Returns
    -------
    t : float
        time.
    w : float
        dVar.
    '''
    h=(b-a)/n
    t=a
    w=alpha
    for i in range(n):
        w=w+h*f(t,w) + (h*h/2) * fp(t,w)
        t = t+h
    return t,w

def taylor4(n,a,b,alpha):
    '''
     See taylor 2
    '''
    h=(b-a)/n
    t=a
    w=alpha
    for i in range(n):
        w=w+h*f(t,w) + (h*h/2) * fp(t,w)+(h**3/6)*fdp(t,w)+(h**4/24)*ftp(t,w)
        t = t+h
    return t,w

def taylor6(n,a,b,alpha):
     '''
      See taylor 2
     '''
     h=(b-a)/n
     t=a
     w=alpha
     for i in range(n):
        w=w+h*f(t,w) + (h*h/2) * fp(t,w)+(h**3/6)*fdp(t,w)+(h**4/24)*ftp(t,w)+(h**5/120)*fqp(t,w)
        t = t+h
     return t,w

def err(x,y):
    '''
    Parameters
    ----------
    x : approximate solution
    y : exact solution

    Returns
    -------
    relative error between x and y
    '''
    return abs((y-x)/y)

yEval= 2
anaSoln = fa(yEval, 34.2343)
t,eulerSoln = euler(20,0,yEval,1)

n = np.array([10,20,50,100,200,1000])
eulerErr = []
t2Err = []
t4Err=[]
t6Err=[]

y1=np.linspace(0,4,20)
t,esol=euler(20,0,y1,1)
t,t2=taylor2(20,0,y1,1)
t,t4=taylor4(20,0,y1,1)
t,t6=taylor6(20,0,y1,1)
ana=fa(y1,34.2343)

plt.figure(0)
plt.grid(True)
plt.title("Plot of approximate solution of y=f(t,y)")
plt.xlabel("t")
plt.ylabel("y(t)")
plt.plot(y1,esol,label="euler")
plt.plot(y1,t2,label="taylor2")
plt.plot(y1,t4,label="taylor4")
plt.plot(y1,t6,label="taylor6")
plt.plot(y1,ana,label="analytic")
plt.legend()
plt.show()


for i in range(len(n)):
    t,eulerSoln = euler(n[i],0,yEval,1)
    eulerErr.append(err(eulerSoln,anaSoln))
    t,t2Soln = taylor2(n[i],0,yEval,1)
    t2Err.append(err(t2Soln,anaSoln))
    t,t4Soln = taylor4(n[i],0,yEval,1)
    t4Err.append(err(t4Soln,anaSoln))
    t,t6Soln = taylor6(n[i],0,yEval,1)
    t6Err.append(err(t6Soln,anaSoln))


error=pd.DataFrame({'eulerErr': eulerErr,'t2Err':t2Err,'t4Err':t4Err,
                        't6Err':t6Err})
file_name='techlab4error.xlsx'
error.to_excel(file_name)

print("Analytic Soln:", anaSoln, "Euler Soln:", eulerSoln, "Difference",
      anaSoln-eulerSoln, "taylor2:", t2Soln, "taylor4:", t4Soln)

plt.figure(1)
plt.grid(True)
plt.title("Plot of Relative Error at t=2")
plt.xlabel("n")
plt.ylabel("Relative Error")
plt.semilogy(n,eulerErr,label="euler")
plt.semilogy(n,t2Err,label="taylor2")
plt.semilogy(n,t4Err,label="taylor4")
plt.semilogy(n,t6Err,label="taylor6")
plt.legend()
plt.show()


print(eulerErr, "test")


'''
Solution to the Gompertz differential equation using eulers method
'''

z=.0439
K=12000.0
def G(t,N):
    return z*np.log(K/N)*N

# def Gp(t,N):
#     return z*K+z*np.log(K/N)*G(t,N)


def euler2(n,a,b,alpha):
    h=(b-a)/n
    t=a
    w=alpha
    for i in range(n):
        w=w+h*G(t,w)
        t=t+h
    return t, w

# def taylor22(n,a,b,alpha):
#     h=(b-a)/n
#     t=a
#     w=alpha
#     for i in range(n):
#         w=w+h*G(t,w) + (h*h/2) * Gp(t,w)
#         t = t+h
#     return t,w


x1=np.linspace(0, 100, 1000, endpoint=True)
t, eulerSoln = euler2(20,0,x1,4000)
# t,t23=taylor22(20,0,x1,4000)

#print(t, eulerSoln, "test")

plt.figure(2)
plt.grid(True)
plt.title("Gompertz solution plot")
plt.xlabel("t")
plt.ylabel("cells")
plt.plot(t,eulerSoln,label="euler")
# plt.plot(t,t23,label="taylor2")
plt.legend
plt.show()


def Iter(numIter,p0,tol):
    '''


    Parameters
    ----------
    numIter : integer
        how many cells in the tuple you want it to check.
    p0 : float
        value you are looking for.
    tol : float
        how close you need the value to be for it to return the cell.

    Returns
    -------
    p : float
        the value that it found within the tol.
    i : float
        the value of the cell.
    '''
    i=1
    while i <= numIter:
        p=eulerSoln[i]
        if abs(p-p0) < tol:
            return p,i
        i = i+1

p,m=Iter(1000,11000,10)

print(p,m)
```