```python
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  1 17:34:07 2019

@author: Alexm
"""

print("Alexandra Mulholland 17336557")
import numpy as np
import matplotlib.pyplot as plt

#Aim: To compute the second derivative of f(x)=xexp(x) using Richardson extrapolation
#Defining the function f(x)=xexp(x) to be differentiated

#PART 1

def function(x):
    f=x*np.exp(x)
    return f
#First derivative, analytic using the product rule
def analytic(x):
    """analytic first derivative"""
    f= (x+1)*np.exp(x)
    return f
#Defining the analytic second derivative, using product rule again
def analytic2(x):
    """analytic second derivative"""
    f= (x+2)*np.exp(x)
    return f

#plot of function
x=np.arange(0.0,2.0,0.001)
plt.figure(1)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Plot of f(x)=xe^x')
plt.plot(x,function(x), color = 'purple')
plt.grid(True)
plt.show()

#Plotting analytical second derivative
x=np.arange(0.0,2.0,0.0001)
plt.figure(3)
plt.xlabel('x')
plt.ylabel('f"(x)')
plt.title('Plot of second derivative f"(x)=(x+2)e^x')
plt.plot(x,analytic2(x), color = 'pink')
plt.grid(True)
plt.show()

#PART 2

h=0.4
#Defining the central difference approximaton for step size h
def D_1_h(x,h):
    f= (function(x+h)-2*function(x)+function(x-h))/h**2
    return f
#Doubling the step size
def D_1_2h(x,h):
    f= (function(x+2*h)-2*function(x)+function(x-2*h))/(4*h**2)
    return f
#Minimising error
def D_2_h(x,h):
    f= (4*D_1_h(x,h)-D_1_2h(x,h))/3
    return f

def D_3_h(x,h):
    f = (16*D_2_h(x,h)-D_2_h(x,2*h))/15
    return f

def D_4_h(x,h):
    f = (64*D_3_h(x,h)-D_3_h(x,2*h))/63
```

```python
    return f
#Above is wrong

#Have to do for decreasing h
#For d1 d2 d3 d4, plotting a matrix M using a loop
x=2
h=0.4
#top row is producing the h values
#analytic value is 29.5562243957226
print (analytic2(x))
print ("")
print(D_1_h(x,h))
print ("")
M = [[h,0,0,0],
     [D_1_h(x,h),0,0,0],
      [0,0,0,0],
      [0,0,0,0],
      [0,0,0,0]]
#j is column no., i is row no.
#With each iteration, halving h
#j ranging from 1 to 4, so that would exclude 1st column (column 0)
#M[row][column]
for j in range(1,4):
    h=h/2
    M[0][j]=h
    M[1][j]=D_1_h(x,h)
#first row- h, h/2,h/4 etc
#second row, CD with decreasing h by half
#for i=1,M[0,1] --- d1(h/2)

#j now ranging from 0 to 3 i.e. inlcuding first column and exclusing first row
#Genral formula applied
#where i takes value of n
for j in range(0,3):
    for i in range(1,4):
        M[i+1][j+1]=((2**(2*(i)))*M[i][j+1]-M[i][j])/(2**(2*(i))-1)
#In order to prevent random values, as code reads the blank spaces
#in the table in the notes as zeros
#so for this range of i and j, if i is greater than j, the value is made to be
#xero in this cell of the matrix
for j in range(1,5):
    for i in range(0,5):
        if i-1>j:
            M[i][j]=0
#printing the matrix
print(M)
print ("")
print ("")

#PART 3

#Second derivative at x=2 to highest accuracy
#now defining a second matrix, which goes to higher values of Dn in order to
#find most accurate value
#now including i and j up to 8 (excluding h row) and renaming them y and z respectively
#Definig a costant x
#new matrix to be called A
x=2
h=0.4
#past D8, rounding errors occur
print ("")
A = [[h,0,0,0,0,0,0,0],
     [D_1_h(x,h),0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0],
      [0,0,0,0,0,0,0,0]]
for z in range(1,8):
    h=h/2
```

```python
    A[0][z]=h
    A[1][z]=D_1_h(x,h)
for z in range(0,7):
    for y in range(1,8):
        A[y+1][z+1]=((2**(2*(y)))*A[y][z+1]-A[y][z])/(2**(2*(y))-1)
#Again, redefining the cells the code will see as having a zero value
for z in range(1,9):
    for y in range(0,9):
        if y-1>z:
            A[y][z]=0
print(A)
print ("")
print ("")
#Creating an empty list
#In range of i values 1 to 8 appending A[i] to [i-1]
#so i is i and j is equal to i-1
#Therefore, goes in a diagonal motion
# i is n in the general formula
Dn=[]
for y in range(1,9):
    Dn.append(A[y][y-1])
errorrel=abs((analytic2(x)-Dn)/analytic2(x))
for y in range(0,7):
    if (errorrel[y])-errorrel[y+1] < 0:
        print("\nMost accurate estimation for initial value of h=0.4 is: D",y+1,"\nGiving a relative error of:",errorrel[y])
        break
#D5 values
print ("")
print (29.556224395715102-analytic2(x))
print (29.55622439574263-analytic2(x))
print (29.5562243957235-analytic2(x))

#D6 Values
print ("")
print (29.556224395742646-analytic2(x))
print (29.556224395715073-analytic2(x))

#D7 value
print ("")
print (29.556224395715066-analytic2(x))


#PART 4 and PART 5

#change in accuracy as h decreases
#plotting D1 and D2 versus h and finding the relationship.
plt.plot(A[0],abs((A[1]-analytic2(x))/analytic2(x)), color = 'green')
plt.ylabel("Relative error")
plt.xlabel("step size, h")
plt.title("Variation of relative error with step size, h")
plt.show()

#Defining the improved estimations
def D_5_h(x,h):
    f = ((256*D_4_h(x,h))-(D_4_h(x,2*h)))/255
    return f

def D_6_h(x,h):
    f = ((1024*D_5_h(x,h))-(D_5_h(x,2*h)))/1023
    return f

def D_7_h(x,h):
    f = ((4096*D_6_h(x,h))-(D_6_h(x,2*h)))/4095
    return f

def D_8_h(x,h):
    f = ((16384*D_7_h(x,h))-(D_7_h(x,2*h)))/16383
    return f

#Defining the relative error for each estimate
x=2
```

```
def error_D1h(x,h):
    f = np.abs(((analytic2(x)-D_1_h(x,h)))/analytic2(x))
    return f

def error_D2h(x,h):
    f = np.abs(((analytic2(x)-D_2_h(x,h)))/analytic2(x))
    return f

def error_D3h(x,h):
    f = np.abs(((analytic2(x)-D_3_h(x,h)))/analytic2(x))
    return f

def error_D4h(x,h):
    f = np.abs(((analytic2(x)-D_4_h(x,h)))/analytic2(x))
    return f

def error_D5h(x,h):
    f = np.abs(((analytic2(x)-D_5_h(x,h)))/analytic2(x))
    return f

def error_D6h(x,h):
    f = np.abs(((analytic2(x)-D_6_h(x,h)))/analytic2(x))
    return f

def error_D7h(x,h):
    f = np.abs(((analytic2(x)-D_7_h(x,h)))/analytic2(x))
    return f

def error_D8h(x,h):
    f = np.abs(((analytic2(x)-D_8_h(x,h)))/analytic2(x))
    return f

#Starting with h=0.4
#plots show the rounding error with decreasing initial h
h = np.arange(0,0.4,0.001)

plt.figure(3)
plt.xlabel('step size, h')
plt.ylabel('f"(x)')
plt.title('Relative error of each estimation versus h, initalised at 0.4')
plt.loglog(h,error_D1h(x,h), label = 'D1(h)',color = 'red')
plt.loglog(h,error_D2h(x,h), label = 'D2(h)', color = 'orange')
plt.loglog(h,error_D3h(x,h), label = 'D3(h)', color = 'yellow')
plt.loglog(h,error_D4h(x,h), label = 'D4(h)', color = 'green')
plt.loglog(h,error_D5h(x,h),label = 'D5(h)', color = 'blue')
plt.loglog(h,error_D6h(x,h), label = 'D6(h)', color = 'purple')
plt.loglog(h,error_D7h(x,h), label = 'D7(h)', color = 'pink')
plt.loglog(h,error_D8h(x,h), label = 'D8(h)', color = 'brown')
plt.legend()
plt.show()

#starting with h=0.2
h = np.arange(0,0.2,0.001)

plt.figure(4)
plt.xlabel('step size, h')
plt.ylabel('f"(x)')
plt.title('Relative error of each estimation versus h, initalised at 0.2')
plt.loglog(h,error_D1h(x,h), label = 'D1(h)', color = 'red')
plt.loglog(h,error_D2h(x,h), label = 'D2(h)', color = 'orange')
plt.loglog(h,error_D3h(x,h), label = 'D3(h)', color = 'yellow')
plt.loglog(h,error_D4h(x,h), label = 'D4(h)', color = 'green')
plt.loglog(h,error_D5h(x,h),label = 'D5(h)', color = 'blue')
plt.loglog(h,error_D6h(x,h), label = 'D6(h)', color = 'purple')
plt.loglog(h,error_D7h(x,h), label = 'D7(h)', color = 'pink')
plt.loglog(h,error_D8h(x,h), label = 'D8(h)', color = 'brown')
plt.legend()
plt.show()


#h=0.1
h = np.arange(0,0.1,0.001)
```

```
plt.figure(5)
plt.xlabel('step size, h')
plt.ylabel('f"(x)')
plt.title('Relative error of each estimation versus h, initalised at 0.1')
plt.loglog(h,error_D1h(x,h), label = 'D1(h)', color = 'red')
plt.loglog(h,error_D2h(x,h), label = 'D2(h)', color = 'orange')
plt.loglog(h,error_D3h(x,h), label = 'D3(h)', color = 'yellow')
plt.loglog(h,error_D4h(x,h), label = 'D4(h)', color = 'green')
plt.loglog(h,error_D5h(x,h),label = 'D5(h)', color = 'blue')
plt.loglog(h,error_D6h(x,h), label = 'D6(h)', color = 'purple')
plt.loglog(h,error_D7h(x,h), label = 'D7(h)', color = 'pink')
plt.loglog(h,error_D8h(x,h), label = 'D8(h)', color = 'brown')
plt.legend()
plt.show()


#h=0.05
h = np.arange(0,0.05,0.001)

plt.figure(6)
plt.xlabel('step size, h')
plt.ylabel('f"(x)')
plt.title('Relative error of each estimation versus h, initalised at 0.05')
plt.loglog(h,error_D1h(x,h), label = 'D1(h)', color = 'red')
plt.loglog(h,error_D2h(x,h), label = 'D2(h)', color = 'orange')
plt.loglog(h,error_D3h(x,h), label = 'D3(h)', color = 'yellow')
plt.loglog(h,error_D4h(x,h), label = 'D4(h)', color = 'green')
plt.loglog(h,error_D5h(x,h),label = 'D5(h)', color = 'blue')
plt.loglog(h,error_D6h(x,h), label = 'D6(h)', color = 'purple')
plt.loglog(h,error_D7h(x,h), label = 'D7(h)', color = 'pink')
plt.loglog(h,error_D8h(x,h), label = 'D8(h)', color = 'brown')
plt.legend()
plt.show()
```