Group 2

Hervie Emmanuel
Adebayo Ayomikun
Olaleye Ayomide
Ishaq Ramadan
Olanipekun Solace

Computation of Runge-Kutta 4th order Differential Equation and Power Function Fit using Least Square Fit using Python Programming Language

Power Function Fit using Least Square Fit using Python Programming Language.

Code:

Compiled with the python 3.9.1 64-bit program

```
import time
import math
import csv
import pandas as pd
```

```
   ###########################################################
   #                   Using a CSV file                       #
   ###########################################################
   def csv_function():
       filename = input("Enter name of csv file you want to use: ")
       df = pd.read_csv(filename + '.csv')
   #     type(df)
       print("Our data frame can be displayed below as: ")
       print(df)
       print(df.columns)

       x_values = df.Xvalues
       y_values = df.Yvalues

       print(x_values)

       sum_x, sum_y, sum_x2, sum_xy = 0, 0, 0, 0
       n = len(df)

       for i in range(n):
           sum_x += math.log10(x_values[i])
           sum_y += math.log10(y_values[i])
           sum_x2 += math.log10(x_values[i]) * math.log10(x_values[i])
           sum_xy += math.log10(x_values[i]) * math.log10(y_values[i])
```

```
print("======================================================================")
        print("Total sum of x-values(sum_x):" + str(sum_x))
        print("")
        print("Total sum of y-values(sum_y):" + str(sum_y))
        print("")
        print("Total sum of square of x-values(sum_x2):" + str(sum_x2))
        print("")
        print("Total sum of x and y values(sum_xy):" + str(sum_xy))

print("======================================================================")
        print('')



        denominator = ((n * sum_x2) - (sum_x * sum_x))
        p = ((n * sum_xy) - (sum_x * sum_y)) / denominator
        c = (sum_y - p * sum_x) / n
        a = math.pow(10, c)


print("======================================================================")

print("======================================================================")
        print('')
        print("From our algorithm above; ")
        print('a: '+ str(a))
        print('p: '+ str(p))
        print('c: '+ str(c))
        print('')


print("======================================================================")

print("======================================================================")
        print('')
        print("Our output can now be written in the form 'y = ax^p' where ^ =
'raised to a power of' as: ")
        print("y = " + str(a) + "x^" + str(p))
```

```
##########################################################
#             Inputing our values the hard way           #
##########################################################

def no_csv_function():
   n = eval(input("Enter number of data points 'n': "))
   x_values = []
   y_values = []

   for i in range(n):
       x_values.append(eval(input("Enter value x(" + str(i+1) + "): ")))
       y_values.append(eval(input("Enter value y(" + str(i+1) + "): ")))

   sum_x, sum_y, sum_x2, sum_xy = 0, 0, 0, 0

   for i in range(n):
       sum_x += math.log10(x_values[i])
       sum_y += math.log10(y_values[i])
       sum_x2 += math.log10(x_values[i]) * math.log10(x_values[i])
       sum_xy += math.log10(x_values[i]) * math.log10(y_values[i])


print("======================================================================")
   print("Total sum of x-values(sum_x):" + str(sum_x))
   print("")
   print("Total sum of y-values(sum_y):" + str(sum_y))
   print("")
   print("Total sum of square of x-values(sum_x2):" + str(sum_x2))
   print("")
   print("Total sum of x and y values(sum_xy):" + str(sum_xy))

print("======================================================================")
   print('')



   denominator = ((n * sum_x2) - (sum_x * sum_x))
   p = ((n * sum_xy) - (sum_x * sum_y)) / denominator
   c = (sum_y - p * sum_x) / n
```

```
    a = math.pow(10, c)


print("======================================================================")

print("======================================================================")
    print('')
    print("From our algorithm above; ")
    print('a: '+ str(a))
    print('p: '+ str(p))
    print('c: '+ str(c))
    print('')


print("======================================================================")

print("======================================================================")
    print('')
    print("Our output can now be written in the form 'y = ax^p' where ^ =
'raised to a power of' as: ")
    print("y = " + str(a) + "x^" + str(p))
```

```

###########################################################
#                         Header                          #
###########################################################
print("======================================================================")
print("||                                                                  ||")
print("||                         Power function fit                       ||")
print("||                                                                  ||")
print("======================================================================")


print("Welcome, enter '1' to enter your values manually or '2' to read your
values from a csv file")
prompt = eval(input("Enter here: "))

if prompt == 1 or prompt == "1":
    csv_function()
elif prompt == 2 or prompt == "2":
```

```
    no_csv_function()
else:
    print("You entered a wrong value, please try again")
    prompt = eval(input("Enter here: "))
```

Running the code with examples from and using a csv file named “data.csv”, It follows that:

```
======================================================================
||                                                                  ||
||                        Power function fit                        ||
||                                                                  ||
======================================================================
Welcome, enter '1' to enter your values manually or '2' to read your values from a csv file
Our data frame can be displayed below as:
   Xvalues  Yvalues
0        1     1200
1        2      900
2        3      800
3        4      600
4        5      400
5        6      200
6        7      100
7        8       50
8        9       20
Index(['Xvalues', 'Yvalues'], dtype='object')
0    1
1    2
2    3
3    4
4    5
5    6
6    7
7    8
8    9
Name: Xvalues, dtype: int64
```

```
=========================================================================
Total sum of x-values(sum_x):5.559763032876794

Total sum of y-values(sum_y):21.61775497985448

Total sum of square of x-values(sum_x2):4.215159407249778

Total sum of x and y values(sum_xy):12.022383523457846
================================================================================

================================================================================
================================================================================

From our algorithm above;
a: 2858.2055240318678
p: -1.7063831771717648
c: 3.4560934542463233

================================================================================
================================================================================

Our output can now be written in the form 'y = ax^p' where ^ = 'raised to a power
of' as:
y = 2858.2055240318678x^-1.7063831771717648
```

The code can also be run with manually typed examples and we would get the same answer. We import a dataset("csv") for instances where we have to work with very large data.

```
======================================================================
||                                                                  ||
||                          Power function fit                      ||
||                                                                  ||
======================================================================
Welcome, enter '1' to enter your values manually or '2' to read your values from a csv fi
Our data frame can be displayed below as:
   Xvalues  Yvalues
0        1     1200
1        2      900
2        3      800
3        4      600
4        5      400
5        6      200
6        7      100
7        8       50
8        9       20
Index(['Xvalues', 'Yvalues'], dtype='object')
0    1
1    2
2    3
3    4
4    5
5    6
6    7
7    8
8    9
Name: Xvalues, dtype: int64
======================================================================
Total sum of x-values(sum_x):5.559763032876794

Total sum of y-values(sum_y):21.61775497985448

Total sum of square of x-values(sum_x2):4.215159407249778

Total sum of x and y values(sum_xy):12.022383523457846
======================================================================

======================================================================
======================================================================
```

```
3    4
4    5
5    6
6    7
7    8
8    9
Name: Xvalues, dtype: int64
======================================================================
Total sum of x-values(sum_x):5.559763032876794

Total sum of y-values(sum_y):21.61775497985448

Total sum of square of x-values(sum_x2):4.215159407249778

Total sum of x and y values(sum_xy):12.022383523457846
======================================================================

======================================================================
======================================================================

From our algorithm above;
a: 2858.2055240318678
p: -1.7063831771717648
c: 3.4560934542463233

======================================================================
======================================================================

Our output can now be written in the form 'y = ax^p' where ^ = 'raised to a power of' as:
y = 2858.2055240318678x^-1.7063831771717648

```

Runge-kutta 4th order Differential Equation using Python Programming Language.

Code:

Compiled with the python 3.9.1 64-bit program

```
import time
import math
import csv
import pandas as pd
```

```
# Defining our running function f(x) as f(x) = xy
def func(a, b):
   return a * b


x_values = []
y_values = []
y_i = 0
```

```
############################################################
#             Inputing our values the hard way             #
############################################################

def no_csv_function():

print("=======================================================================")
   print("=                      STARTING INTERPOLATION
=")

print("=======================================================================")
   time.sleep(.4)
   n = eval(input("Enter number of data points 'n': "))

   for i in range(n):
```

```
       x_values.append(eval(input("Enter value x(" + str(i+1) + "): ")))

   y_values.append(eval(input("Enter initial value of y 'y1': ")))

   h = eval(input("Enter step-size 'h' of x values: "))

print("======================================================================")

print("======================================================================")
   print('')

   for i in range(n):
       s1 = func(x_values[i], y_values[i])
#         print(s1)
       s2 = func(x_values[i] + h/2, y_values[i] + (h/2) * s1)
#         print(s2)
       s3 = func(x_values[i]  + h/2, y_values[i] + (h/2) * s2)
#         print(s3)
       s4 = func(x_values[i] + h, y_values[i] +h * s3)
#         print(s4)

       s = (s1 + (2 * s2) + (2 * s3) + s4) / 6
       print('s: ' + str(s))
       time.sleep(.5)

       y_i = y_values[i] + (h * s)
       y_values.append(y_i)
       print('y('+str(i+1)+'): ' + str(y_i))

print("======================================================================")

print("======================================================================")
       print('')
       time.sleep(.5)

   print("Hence for range of values xi to xf, our respcective y-values are: ")
   for i in range(n):
       print(y_values[i+1])
```

```
##########################################################
```

```
#                    Using a CSV file                      #
############################################################
def csv_function():

print("======================================================================")
   print("=                       STARTING INTERPOLATION
=")

print("======================================================================")

   time.sleep(.4)
   filename = input("Enter name of csv file you want to use: ")
   df = pd.read_csv(filename + '.csv')
#     type(df)
   print("Our data frame can be displayed below as: ")
   print(df)
   print(df.columns)

   x_values = df.xvalues

   y_i = df.iloc[0, 2]

   h = df.iloc[0,1]

   y_values.append(df.iloc[0, 2])

   print(x_values, h, y_values)
   print(y_values[0])

   n = len(df)

   time.sleep(.5)
   print('')

print("======================================================================")

print("======================================================================")
   print('')

   for i in range(n):
```

```
        s1 = func(x_values[i], y_values[i])
#           print(s1)
        s2 = func(x_values[i] + h/2, y_values[i] + (h/2) * s1)
#           print(s2)
        s3 = func(x_values[i]  + h/2, y_values[i] + (h/2) * s2)
#           print(s3)
        s4 = func(x_values[i] + h, y_values[i] +h * s3)
#           print(s4)

        s = (s1 + (2 * s2) + (2 * s3) + s4) / 6
        print('s: ' + str(s))

        y_i = y_values[i] + (h * s)
        y_values.append(y_i)
        print('y('+str(i+1)+'): ' + str(y_i))

print("========================================================================")

print("========================================================================")
        print('')
        time.sleep(.5)
    print(y_i)

    print("Hence for range of values xi to xf, our respcective y-values are: ")
    for i in range(n):
        print(y_values[i+1])
```

```
############################################################
#                          Header                          #
############################################################
print("========================================================================")
print("||                                                                    ||")
print("||                      Runge-Kutta 4th order DE                      ||")
print("||                                                                    ||")
print("========================================================================")
```

```
print("Welcome, enter '1' to read your values from a csv file or '2' to enter
your values manually or 3 if you have just the initial x-value")
prompt = eval(input("Enter here: "))

if prompt == 1 or prompt == "1":
   csv_function()
elif prompt == 2 or prompt == "2":
   no_csv_function()
else:
   print("You entered a wrong value, please try again")
   prompt = eval(input("Enter here: "))
```

Running the code with examples from and using a csv, It follows that:

```
=================================================================
||                                                             ||
||                    Runge-Kutta 4th order DE                 ||
||                                                             ||
=================================================================
Welcome, enter '1' to read your values from a csv file or '2' to enter your values manually or 3 if you
have just the initial x-value
=================================================================
=           STARTING INTERPOLATION                 =
=================================================================
Our data frame can be displayed below as:
   xvalues  stepsize  initial_yvalue
0     1.0       0.1             5.0
1     1.1       NaN             NaN
2     1.2       NaN             NaN
3     1.3       NaN             NaN
4     1.4       NaN             NaN
5     1.5       NaN             NaN
Index(['xvalues', 'stepsize', 'initial_yvalue'], dtype='object')
0    1.0
1    1.1
2    1.2
3    1.3
4    1.4
5    1.5
Name: xvalues, dtype: float64 0.1 [5.0]
5.0

=================================================================
```

```
================================================================================

s: 5.53552453125
y(1): 5.5535524531250005
================================================================================
================================================================================

s: 6.768295027056779
y(2): 6.230381955830678
================================================================================
================================================================================

s: 8.295639999517478
y(3): 7.0599459557824265
================================================================================
================================================================================

s: 10.204190535067632
y(4): 8.08036500928919
================================================================================
================================================================================

s: 12.608520841015228
y(5): 9.341217093390714
================================================================================
================================================================================

s: 15.661220300178604
y(6): 10.907339123408574
================================================================================
================================================================================

10.907339123408574
Hence for range of values xi to xf, our respcective y-values are:
5.5535524531250005
6.230381955830678
7.0599459557824265
8.08036500928919
9.341217093390714
10.907339123408574
```

```
======================================================================
||                                                                  ||
||                      Runge-Kutta 4th order DE                    ||
||                                                                  ||
======================================================================
Welcome, enter '1' to read your values from a csv file or '2' to enter your values manually or 3 if you have just the initial x-value
======================================================================
=                       STARTING INTERPOLATION                       =
======================================================================
Our data frame can be displayed below as:
   xvalues  stepsize  initial_yvalue
0      1.0       0.1             5.0
1      1.1       NaN             NaN
2      1.2       NaN             NaN
3      1.3       NaN             NaN
4      1.4       NaN             NaN
5      1.5       NaN             NaN
Index(['xvalues', 'stepsize', 'initial_yvalue'], dtype='object')
0    1.0
1    1.1
2    1.2
3    1.3
4    1.4
5    1.5
Name: xvalues, dtype: float64 0.1 [5.0]
5.0

======================================================================
======================================================================

s: 5.53552453125
y(1): 5.5535524531250005
======================================================================
======================================================================

s: 6.768295027056779
y(2): 6.230381955830678
======================================================================
======================================================================

s: 8.295639999517478
y(3): 7.0599459557824265
======================================================================
======================================================================

3
```

```
s: 6.768295027056779
y(2): 6.230381955830678
======================================================================
======================================================================

s: 8.295639999517478
y(3): 7.0599459557824265
======================================================================
======================================================================

s: 10.204190535067632
y(4): 8.08036500928919
======================================================================
======================================================================

s: 12.608520841015228
y(5): 9.341217093390714
======================================================================
======================================================================

s: 15.661220300178604
y(6): 10.907339123408574
======================================================================
======================================================================

10.907339123408574
Hence for range of values xi to xf, our respcective y-values are:
5.5535524531250005
6.230381955830678
7.0599459557824265
8.08036500928919
9.341217093390714
10.907339123408574
```