Math 4610 – HW 4

Jake Daniels A02307117

October 10, 2022

Task 1:

$$f''(x_0) \approx \tfrac{1}{h^2}(f(x_0+h) - 2f(x_0) + f(x_0-h))$$

We want to show...

$$|f''(x_0) - \tfrac{1}{h^2}(f(x_0+h) - 2f(x_0) + f(x_0-h))| \leq Ch^2$$

We will start by writing out the taylor series expansions about x_0 for $f(x_0+h)$ and $f(x_0-h)$ as follows:

$$\begin{aligned} f(x_0+h) &= f(x_0) + hf'(x_0) + \tfrac{h^2}{2}f''(x_0) + \tfrac{h^3}{6}f'''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_1) \\ f(x_0-h) &= f(x_0) - hf'(x_0) + \tfrac{h^2}{2}f''(x_0) - \tfrac{h^3}{6}f'''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_2) \end{aligned}$$

Where ξ_1 and ξ_2 are values between x_0 and h. Plugging these into the equation yields:

$$|f''(x_0) - \tfrac{1}{h^2}(f(x_0) + hf'(x_0) + \tfrac{h^2}{2}f''(x_0) + \tfrac{h^3}{6}f'''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_1) - 2f(x_0) + f(x_0) - hf'(x_0) + \tfrac{h^2}{2}f''(x_0) - \tfrac{h^3}{6}f'''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_2)|$$

$$\begin{aligned} &= |f''(x_0) - \tfrac{1}{h^2}(h^2 f''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_1) + \tfrac{h^4}{24}f^{(4)}(\xi_2)| \\ &= |f''(x_0) - f''(x_0) - \tfrac{h^2}{24}f^{(4)}(\xi_1) - \tfrac{h^2}{24}f^{(4)}(\xi_2)| \\ &= \tfrac{h^2}{24}|f^{(4)}(\xi_1) + f^{(4)}(\xi_2)| = \tfrac{h^2}{12}|f^{(4)}(\xi_3)| \leq Ch^2 \end{aligned}$$

Where $C = \frac{|f^{(4)}(\xi_3)|}{12}$. We get the equality $|f^{(4)}(\xi_1) + f^{(4)}(\xi_2)| = 2|f^{(4)}(\xi_3)|$ by the Mean Value Theorem. I have successfully shown that the given equality above is true so onto Task 2.

Task 2:

The code for this section will be found by following the link below.

https://jake-daniels16.github.io/math4610/

From the directory click on "Derivative Approximation Code" and from there click on the methods folder to get to the code for this task titled 'secondDerivative.py'. At the bottom of this file I added a function test() which outputs the following table for the second derivative of the given function at $x_0 = \frac{\pi}{4}$:

```
h values | approximations | exact | difference
1.0  |  0.08888433309385912  |  -0.03235131011796782  |  0.12123564321182695
0.5  |  -0.10999942340177273  |  -0.03235131011796782  |  -0.07764811328380491
0.25  |  -0.07472439472298283  |  -0.03235131011796782  |  -0.04237308460501501
0.125  |  -0.06930444457864604  |  -0.03235131011796782  |  -0.03695313446067822
0.0625  |  -0.06807675204221697  |  -0.03235131011796782  |  -0.03572544192424915
0.03125  |  -0.0677769923768814  |  -0.03235131011796782  |  -0.03542568225891358
0.015625  |  -0.06770248904469156  |  -0.03235131011796782  |  -0.03535117892672374
0.0078125  |  -0.0676838903286523  |  -0.03235131011796782  |  -0.03533258021068448
0.00390625  |  -0.0676792423422512  |  -0.03235131011796782  |  -0.03532793222428338
0.001953125  |  -0.06767808045242418  |  -0.03235131011796782  |  -0.035326770334456366
0.0009765625  |  -0.06767778999164875  |  -0.03235131011796782  |  -0.035326479873680934
0.00048828125  |  -0.06767771738668671  |  -0.03235131011796782  |  -0.03532640726871889
0.000244140625  |  -0.06767769937869161  |  -0.03235131011796782  |  -0.035326389260723796
0.0001220703125  |  -0.06767769472207874  |  -0.03235131011796782  |  -0.03532638460411092
6.103515625e-05  |  -0.06767769576981664  |  -0.03235131011796782  |  -0.03532638565184882
3.0517578125e-05  |  -0.0676776971668005  |  -0.03235131011796782  |  -0.03532638704883268
1.52587890625e-05  |  -0.06767769902944565  |  -0.03235131011796782  |  -0.03532638891147783
7.62939453125e-06  |  -0.0676778256893158  |  -0.03235131011796782  |  -0.03532651557134798
3.814697265625e-06  |  -0.06767797470092773  |  -0.03235131011796782  |  -0.03532666458295992
```

Task 3:

To find the code for this task click the link provided in task 2 and follow "Derivative Approximation Code" once again. The specific code for this task will be titled 'dataFitting.py' and will also be in the methods folder. I also made a file 'testing.py' which is where I did this task. Using the same function from task 2 but at the point $x_0 = 1$ I created the following code:

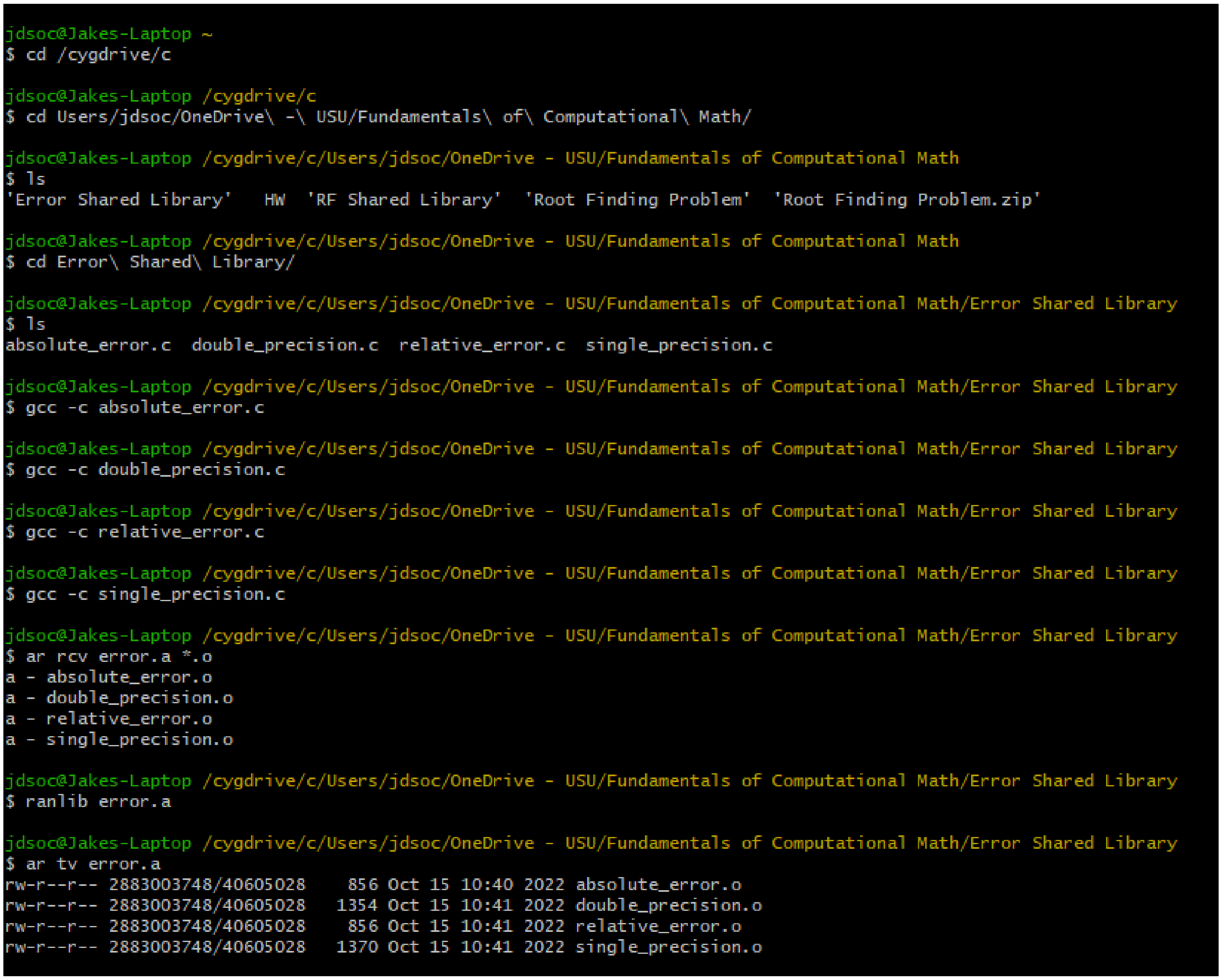
```
import numpy as np
from Methods.secondDerivative import secondDerivative
from Methods.dataFitting import dataFitting

def main():
    exact = -0.0548915699521
    hVals, fVals, diff = secondDerivative(lambda x: (x - np.pi / 2) * np.tan(x) * np.tan(x) / (x * x + 65), 1.0,
                                         1.0, exact)
    a, b = dataFitting(hVals, diff, 19)
    print("y = ", a, "x + ", b)

main()
```

This returns:

```
y =  -0.013871376778266133 x +  -0.11840202632508127
```

Task 4:

The code for all the error routines will again be found at the link above. This time click on the ”Error Routines” and there you will find the C code for absolute and relative error as well as single and double precision. Here are the screenshots of me creating a shared library 'error.a' with the 4 routines.



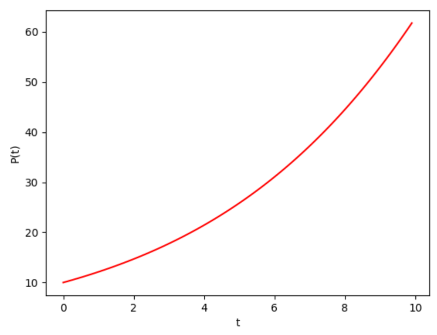
```
jdsoc@Jakes-Laptop ~
$ cd /cygdrive/c

jdsoc@Jakes-Laptop /cygdrive/c
$ cd Users/jdsoc/OneDrive\ -\ USU/Fundamentals\ of\ Computational\ Math/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math
$ ls
'Error Shared Library'   HW  'RF Shared Library'  'Root Finding Problem'  'Root Finding Problem.zip'

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math
$ cd Error\ Shared\ Library/

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ ls
absolute_error.c  double_precision.c  relative_error.c  single_precision.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ gcc -c absolute_error.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ gcc -c double_precision.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ gcc -c relative_error.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ gcc -c single_precision.c

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ ar rcv error.a *.o
a - absolute_error.o
a - double_precision.o
a - relative_error.o
a - single_precision.o

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ ranlib error.a

jdsoc@Jakes-Laptop /cygdrive/c/Users/jdsoc/OneDrive - USU/Fundamentals of Computational Math/Error Shared Library
$ ar tv error.a
rw-r--r-- 2883003748/40605028    856 Oct 15 10:40 2022 absolute_error.o
rw-r--r-- 2883003748/40605028   1354 Oct 15 10:41 2022 double_precision.o
rw-r--r-- 2883003748/40605028    856 Oct 15 10:41 2022 relative_error.o
rw-r--r-- 2883003748/40605028   1370 Oct 15 10:41 2022 single_precision.o
```

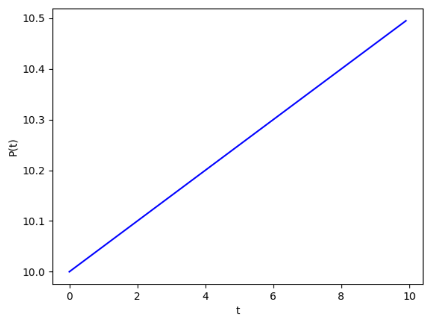
Task 5:

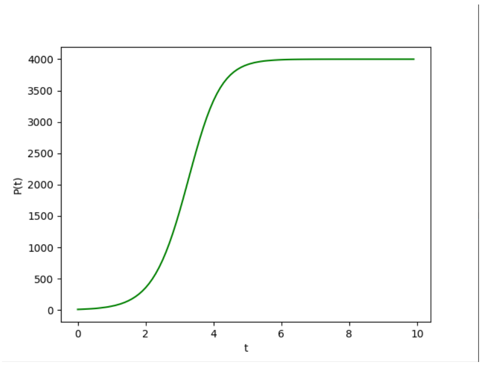
For this task I created two functions, an Explicit Euler method function and a logistic equation function. The Explicit Euler method can be found in the link provided above as well as the Logistic Equation function can be found there as well. Both are in the file 'explicitEuler.py' found again in the methods folder. Here is the input for the three examples wanted:

```
tApprox1, xApprox1 = logisticEquation(0.2, 0.0005, 10.0)
tApprox2, xApprox2 = logisticEquation(0.01, 0.0005, 10.0)
tApprox3, xApprox3 = logisticEquation(2.0, 0.0005, 10.0)
plt.plot(tApprox1, xApprox1, color='red')
plt.xlabel('t')
plt.ylabel('P(t)')
plt.show()
plt.plot(tApprox2, xApprox2, color='blue')
plt.xlabel('t')
plt.ylabel('P(t)')
plt.show()
plt.plot(tApprox3, xApprox3, color='green')
plt.xlabel('t')
plt.ylabel('P(t)')
plt.show()
```

Here are the three graphs returned. The red graph corresponds to the first example, blue to the second, and green to the third.


P(t)
t
10
20
30
40
50
60
0
2
4
6
8
10


P(t)
t
10.0
10.1
10.2
10.3
10.4
10.5
0
2
4
6
8
10


P(t)
t
0
500
1000
1500
2000
2500
3000
3500
4000
0
2
4
6
8
10