Math 4610 – HW 4

Jake Daniels A02307117

October 10, 2022

Task 1:

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

We want to show...

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

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:

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

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

$$\begin{aligned}
&|f''(x_0) - \tfrac{1}{h^2}(f(x_0) + hf'(x_0) + \tfrac{h^2}{2}f''(x_0) + \tfrac{h^3}{6}f'''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_1) - 2f(x_0) + f(x_0) - hf'(x_0) + \tfrac{h^2}{2}f''(x_0) - \tfrac{h^3}{6}f'''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_2)| \\
&= |f''(x_0) - \tfrac{1}{h^2}(h^2f''(x_0) + \tfrac{h^4}{24}f^{(4)}(\xi_1) + \tfrac{h^4}{24}f^{(4)}(\xi_2)| \\
&= |f''(x_0) - f''(x_0) - \tfrac{h^2}{24}f^{(4)}(\xi_1) - \tfrac{h^2}{24}f^{(4)}(\xi_2)| \\
&= \tfrac{h^2}{24}|f^{(4)}(\xi_1) + f^{(4)}(\xi_2)| = \tfrac{h^2}{12}|f^{(4)}(\xi_3)| \leq Ch^2
\end{aligned}$$

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 copy and pasted below, but if for any reason you need it in another place it can also be found by following the link below.

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

The following is the code I wrote to approximate the second derivative:

```
import numpy as np

def secondDerivative(f, x0, h, exact):
    f0 = f(x0)
    diff = []
    fV = []
    hV = []
    for i in range (1, 20):
        hV.append(h)
        f1 = f(x0 + h)
        f2 = f(x0 - h)
        fval = (f1 - 2 * f0 + f2) / (h * h)
        fV.append(fval)
        diff.append(fval - exact)
        h = h / 2
    return hV, fV, diff
```

To test the given function in this task I wrote the following bit of code:

```
def test():
    exact = 16 * (-np.pi ** 3 + 3 * np.pi ** 2 - 1040 * np.pi + 1040) / (np.pi ** 2 + 1040) ** 2
    hVals, fVals, diff = secondDerivative(lambda x: (x - np.pi / 2) * np.tan(x) * np.tan(x) / (x * x + 65), np.pi / 4, 1.0,
    ↪  exact)
    print("h values | approximations | exact | difference")
    for i in range (0, 19):
        print(hVals[i], " | ", fVals[i], " | ", exact, " | ",  diff[i])

test()
```

Which yielded the following output when ran:

```
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  |  -0.0354256825891358
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:

I wrote the following code for this task which fits data sets to a linear polynomial:

```python
import numpy as np

def dataFitting(x, y, n):
    a11 = n+1
    a12 = x[0]
    for i in range(1, n):
        a12 = a12 + x[i]
    a21 = a12
    a22 = x[0] * x[0]
    for i in range(1, n):
        a22 = a22 + x[i] * x[i]
    b1 = y[0]
    for i in range(1, n):
        b1 = b1 + y[i]
    b2 = x[0] * y[0]
    for i in range(1, n):
        b2 = b2 + x[i] * y[i]
    detA = a11 * a22 - a12 * a21
    b = (b1 * a22 - b2 * a12) / detA
    a = (b2 * a11 - b1 * a21) / detA
    return a, b
```

To test this code on the function from task 2 I wrote the following code:

```python
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:

Here are the four routines I wrote for this task, first I'll start with my code for relative error:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double relativeError();

double relativeError(double approx, double exact)
{
    double relError = fabs(approx - exact) / exact;
    return relError;
}
```

Absolute error:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double absoluteError();

double absoluteError(double approx, double exact)
{
    double absError = fabs(approx - exact);
    return absError;
}
```

Single precision:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float singlePrecision();

float singlePrecision()
{
    float one = 1.0f;
    float seps = 1.0f;
    float appone = 1.0f;
    double count = 0.0;
    for(int i = 1.0; i < 101; i++)
    {
        appone = one + seps;
        if(fabs(appone - one) == 0.0f)
        {
            count = count + 1.0;
            printf("loops: %f\n", count);
            printf("single precision machine epsilon = %f", seps);
            break;
        } else {
            seps = 0.5f * seps;
            count = count + 1.0;
        }
    }
}
```

Lastly, double precision:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double doublePrecision();

double doublePrecision()
{
    double one = 1.0d;
    double deps = 1.0d;
    double appone = 1.0d;
    double count = 0.0;
    for(int i = 1.0; i < 101; i++)
    {
        appone = one + deps;
        if(fabs(appone - one) == 0.0d)
        {
            count = count + 1.0;
            printf("loops: %f\n", count);
            printf("double precision machine epsilon: %f", deps);
            break;
        } else {
            deps = deps * 0.5d;
            count = count + 1.0;
        }
    }
}
```

Proof of me creating a shared library will be on the next page, and all .o files as well as the library can be found in the github link above by following 'Error Routines'.

Here is a screenshot of me creating a shared library:

```
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 which are as follows:

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

def explicitEuler(f, x0, t0, T, n):
    h = (T - t0) / n
    f0 = f(t0, x0)
    tval = []
    xval = []
    tval.append(t0)
    xval.append(x0)
    for i in range(1, n):
        t1 = t0 + h
        x1 = x0 + h * f0
        x0 = x1
```

```
        t0 = t1
        tval.append(t0)
        xval.append(x0)
        f0 = f(t0, x0)
    return tval, xval

def logisticEquation(a, b, p0):
    p = lambda t, x: a * x - b * x * x
    tval, pval = explicitEuler(p, p0, 0, 10, 100)
    return tval, pval
```

I then wrote the following code to run the three examples given in this task:

```
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.