# Tech Lab 3

Ezra Harris

October 22, 2021

**Abstract**

This paper evaluates five numerical differentiation schemes: forward, backward, center, center with a fourth order error factor, and center with a sixth order error factor. These methods were analyzed against the analytic solution and the determination was made that the sixth order differentiation scheme was most accurate. Finally the functions were timed to determine speed of computation. This revealed that center difference was the fastest method which is a surprising result given that the number of floating point operations.

## 1 Introduction

In this assignment we seek to improve numerical integration beyond that of a base coding of the definition of the derivative. We begin our analysis using the forward difference, which is ultimately the definition of the derivative, and proceed up to a sixth order method. We examined the time, and accuracy of using these models to calculate the derivative of a test function. This analysis is important because the accuracy of calculating numerical derivatives is critical when conducting math modeling using differential equations which is typical of most physical systems.

## 2 Methods

First I will begin by describing the numerical methods evaluated in this report. Then I will describe how these methods were analyzed for accuracy and computational speed.

The forward difference and definition of the derivative are the same. The definition of the derivative works by calculating the secant line of two points separated by a very small step size h such that the slope as h approaches 0 is the tangent line to the function at the point of interest. This point of interest is typically denoted as $x0$ with the point very close to that as $x0 + h$.

$$\lim_{h \to 0} \frac{f(x_0 + h) - f(x0)}{h} \tag{1}$$

This is know as the forward difference method and the definition of the derivative. The backward difference method is the same process except the value at the point very close to the point of interest is subtracted from the value of the function.

$$\lim_{h \to 0} \frac{f(x_0) - f(x0 - h)}{h} \tag{2}$$

These methods are useful tools. However, they are limited by the step size, the order of the error associated with calculating the value of the functions, and the concavity of the function at the point of interest. In order to limit the error caused by the concavity of the function we used the center difference method:

$$\lim_{h\to 0} \frac{f(x_0+h)-f(x0-h)}{2h} \tag{3}$$

This creates a second order method that also reduces the error associated with the concavity of the function. In the results section there is a table of the errors of each method based on step size for comparison. The bottom line it is much more accurate. Using this method we were able to construct fourth and sixth order methods that were even more accurate. This was done by first setting the center difference equal to a function S which is a function of the step size h.

$$S(h) = \frac{f(x_0+h)-f(x0-h)}{2h} \tag{4}$$

With errors of

$$M = S(h) + e_2h^2 + O(h^4) \tag{5}$$

We then passed S(h) the value of $\frac{h}{2}$ and did some algebraic manipulation to reduce the second order term of the function

$$M = S*(\frac{h}{2}) + e_2\frac{h^2}{2} + O(h^4) \tag{6}$$

$$\frac{(4*M-M)}{3} = \frac{4S(\frac{h}{2})-S(h)}{3} + O(h^4) \tag{7}$$

Which is our fourth order method. A similar process was used to construct our sixth order method. Where we set T(H) equal to the fourth order method

$$M = \frac{16T(\frac{h}{2})-T(h)}{15} + O(h^6) \tag{8}$$

In order to evaluate the accuracy of these methods we used a test function and its analytic derivative to compare our numerical method. The test function and its derivative are:

$$f(x) = -xsin(x) - \frac{x^2}{4}cos(x) \tag{9}$$

$$f'(x) = \frac{x^2}{4}sin(x) - \frac{3x}{2}cos(x) - sin(x) \tag{10}$$

By calculating the analytic value of the derivative at the point of interest we were able to compare our methods with a relative error scheme where x is the value calculated by the numerical method and y is the value of the derivative of the function calculated at the point of interest.

$$error = \frac{|x-y|}{y} \tag{11}$$

The error scheme allowed us to compare the accuracy of our results for various methods. In order to analyze the computation time of the various methods we used the time module and measured the elapsed time to run the function. Finally we graphed the values of the error and the value of derivative of the function as calculated by the various methods.

# 3 Analysis and Results

We began our analysis by calculating the values of the derivatives of our test function using the methods outlined in the methods section. We varied the step size by a value of $h^n$ iterating n from 1 to 5. A table of these values can be found in Appendix A. We can immediately see the discrepancies associated with each set of calculations with an smaller step size generally leading to better results. Also listed in Appendix A are the error calculations of the various methods. With the sixth order differentiation scheme being the best.

Next we looked at the graph of the values of the methods as a function of their step size. We can see that the error associated with the values of each of these measurements increases as h increases.

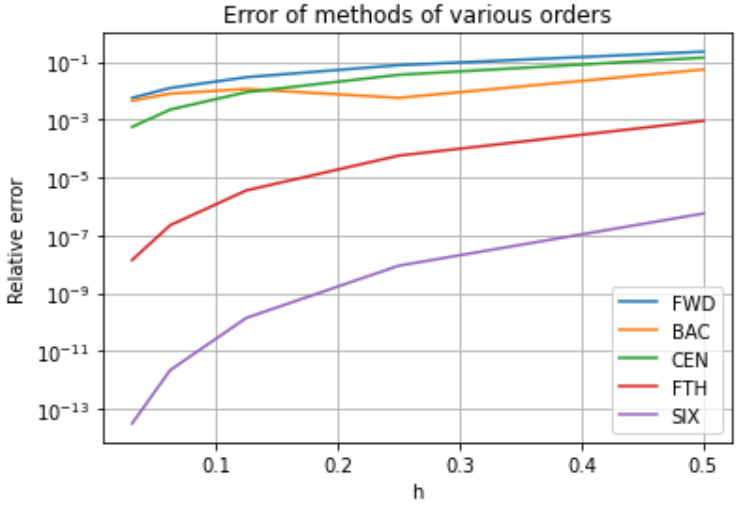


Figure 1: Error as a function of h

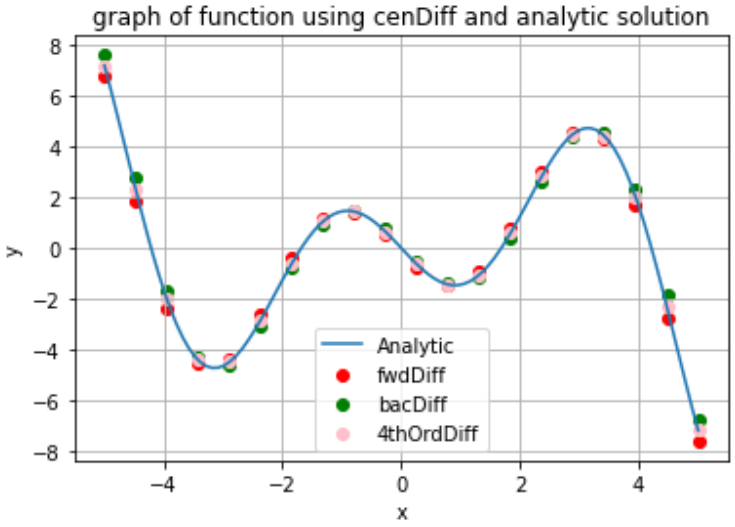


Figure 2: Derivative as calculate by the various methods

We then calculated the graph of the analytic solution of the derivative and the various methods used to calculate a numerical derivative. By visual inspection we can see that the fourth order method was the most accurate. One interesting thing to note is that the position of the forward difference and backwards difference above or below the function changed as the concavity of the derivative changed.

Finally, we timed the functions. Interestingly the most time consuming method was the forward difference. Followed by the sixth order which is to be expected. I believed that factors governing speed of computation would be number of floating point operations however this was not the case. This may be more of a problem with my methods than with the theory itself.

| **Method** | **Time (s)** |
|---|---|
| **FWD** | 0.0001689000055193901 |
| **BAC** | 7.71000050008297e-05 |
| **CEN** | 4.640000406652689e-05 |
| **4TH** | 8.430000161752105e-05 |
| **6TH** | 0.00015359999088104814 |

# 4 Conclusion

Based on our analysis we can see that higher order methods produce more accurate results. One drawback is computation time however this is not always the case with the 1st order forward difference method taking the most time to calculate its results. Finally, we can see that decreasing the step size yields better results.

# 5 Appendix A

| **h** | **FwdDiff** | **BacDiff** | **CenDiff** | **FourthDiff** | **SixthDiff** |
|---|---|---|---|---|---|
| **0.5** | -1.11897 | -1.36397 | -1.24147 | -1.44026 | -1.44156 |
| **.25** | -1.33143 | -1.44969 | -1.39056 | -1.44147 | -1.44156 |
| **0.125** | -1.39945 | -1.45804 | -1.42875 | -1.44155 | -1.44156 |
| **0.0625** | -1.42374 | -1.45296 | -1.43835 | -1.44156 | -1.44156 |
| **0.03125** | -1.43345 | -1.44806 | -1.44075 | -1.44156 | -1.44156 |

| Method | **errFwd** | **errBac** | **errCen** | **err4thOrder** | **err6thOrder** |
|---|---|---|---|---|---|
| .5 | 0.223776 | 0.053822 | 0.138799 | 0.000901 | 5.66E-07 |
| .25 | 0.076396 | 0.005645 | 0.035375 | 5.68E-05 | 8.88E-09 |
| .125 | 0.029208 | 0.011435 | 0.008886 | 3.56E-06 | 1.39E-10 |
| .0625 | 0.012361 | 0.007913 | 0.002224 | 2.23E-07 | 2.17E-12 |
| .03125 | 0.005622 | 0.004509 | 0.000556 | 1.39E-08 | 3.1E-14 |

# 6 Appendix B

```
    """
Created on Thu Sep 30 08:41:19 2021

@author: Ezra
"""

import numpy as np
import time
import pandas as pd

def f(x):
    return -x*np.sin(x)-x*x/4*np.cos(x)

def fPrime(x):
    return x**2/4*np.sin(x)-3*x/2*np.cos(x)-np.sin(x)


def fwdDiff(x0,h):
    dfdx=(f(x0+h)-f(x0))/h
    return dfdx


def bacDiff(x0,h):
    dfdx=(f(x0)-f(x0-h))/h
    return dfdx

def cenDiff(x0,h):
    dfdx=(f(x0+h)-f(x0-h))/(2*h)
    return dfdx

def fourthOrdDiff(x0,h):
    dfdx = (4*cenDiff(x0, h/2)-cenDiff(x0,h))/3
    return dfdx


def sixthOrdDiff(x0,h):
    dfdx=(16*(fourthOrdDiff(x0, h/2))-fourthOrdDiff(x0, h))/15
    return dfdx


x0=np.array([-3,-2.5,-2,-1.5,-1,-.5,0,.5,1,1.5,2,2.5,3])

t1 = time.perf_counter()
fde=fwdDiff(x0,.1)
tfwd  = time.perf_counter()-t1
print("time to run fwdDiff", tfwd)

t2 = time.perf_counter()
bde=bacDiff(x0,.1)
tbc  = time.perf_counter()-t2
```

```
print("time to run bacDiff", tbc)

t3 = time.perf_counter()
cde=cenDiff(x0,.1)
tcen  = time.perf_counter()-t3
print("time to run cendiff", tcen)

t4 = time.perf_counter()
foe=fourthOrdDiff(x0, 0.1)
tfourth  = time.perf_counter()-t4
print("time to run fourthOrdDiff", tfourth)

t5 = time.perf_counter()
soe=sixthOrdDiff(x0,.1)
tsix  = time.perf_counter()-t5
print("time to run sixthOrdDiff", tsix)

act=fPrime(x0)

print("Forward Difference Estimate=", fde)
print("Backwards Difference Estimate=", bde)
print("Center Difference Estimate=", cde)
print("Fourth Order Difference Estimate=", foe)
print("Sixth Order Difference Estimate=", soe)
print("Analytic Value of Derivative=", act)

def err(x,y):

    return abs((y-x)/y)

h=np.array([.5,.5**2,.5**3,.5**4,.5**5])

x0=1
analytical= fPrime(x0)
errFwd=[]
errCen=[]
errBac=[]
errFourthOrder=[]
errSixthOrder=[]
dfdxFwd=[]
dfdxBac=[]
dfdxCen=[]
dfdxFourth=[]
dfdxSixth=[]

for i in range(len(h)):
    dfdxFwd.append(fwdDiff(x0,h[i]))
    dfdxBac.append(bacDiff(x0,h[i]))
    dfdxCen.append(cenDiff(x0,h[i]))
    dfdxFourth.append(fourthOrdDiff(x0,h[i]))
    dfdxSixth.append(sixthOrdDiff(x0, h[i]))
    errFwd.append(err(dfdxFwd[i],analytical))
```

```
    errBac.append(err(dfdxBac[i],analytical))
    errCen.append(err(dfdxCen[i],analytical))
    errFourthOrder.append(err(dfdxFourth[i],analytical))
    errSixthOrder.append(err(dfdxSixth[i],analytical))


import matplotlib.pyplot as plt

plt. figure(0)
plt.semilogy(h,errFwd,label="FWD")
plt.semilogy(h,errBac,label="BAC")
plt.semilogy(h,errCen,label="CEN")
plt.semilogy(h,errFourthOrder,label="FTH")
plt.semilogy(h,errSixthOrder,label="SIX")
plt.title("Error of methods of various orders")
plt.xlabel("h")
plt.ylabel("Relative error")
plt.grid(True)
plt.legend()
plt.show()


x=np.linspace(-5,5,100)
z=np.linspace(-5,5,20)

plt.figure(1)
plt.plot(x,fPrime(x),label="Analytic")
plt.scatter(z,fwdDiff(z, .1),color='red',label= 'fwdDiff')
plt.scatter(z,bacDiff(z, .1),color='green',label='bacDiff')
#plt.scatter(z,cenDiff(z, .1),color='orange',label="cenDiff")
plt.scatter(z,fourthOrdDiff(z, .1),color='pink',label="4thOrdDiff")
plt.title("graph of function using cenDiff and analytic solution")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()



dfdxFwd=tuple(dfdxFwd)
dfdxBac=tuple(dfdxBac)
dfdxCen=tuple(dfdxCen)
dfdxFourth=tuple(dfdxFourth)
dfdxSixth=tuple(dfdxSixth)
errFwd=tuple(errFwd)
errBac=tuple(errBac)
errCen=tuple(errCen)
errFourthOrder=tuple(errFourthOrder)
errSixthOrder=tuple(errSixthOrder)

Estimates=pd.DataFrame({'FwdDiff': dfdxFwd,'BacDiff':dfdxBac,'CenDiff':dfdxCen,
                        'FourthOrdDiff':dfdxFourth,'SixthOrdDiff':dfdxSixth,
```

```
                        'errFwd':errFwd,'errBac':errBac,'errCen':errCen,
                        'errFourthOrder':errFourthOrder,'errSixthOrder':errSixthOrder})
file_name='dfdx.xlsx'
Estimates.to_excel(file_name)
```

# 7 References

CDT Christian, Reiger. Company E2 Assistance given through visual instruction. Sent me a picture of his code for the graphs. I had forgotten to include h[i] and my function was creating several lines rather than one continuous line. West Point NY. 21 Oct 21.

Table Generator. https://www.tablesgenerator.com/ Used to create tables for project. Accessed 21 Oct 21.