3801ICT Numerical Algorithms
Milestone 1

Lochie Ashcroft - s5080439

Trimester 1 - 2019

Question 1

The task is to find the derivative at $x = 0$ for the function

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.15x + 1.2$$

Analytical solution

The analytical solution can be found by differentiating the function and substituting 0.5 for the value of x

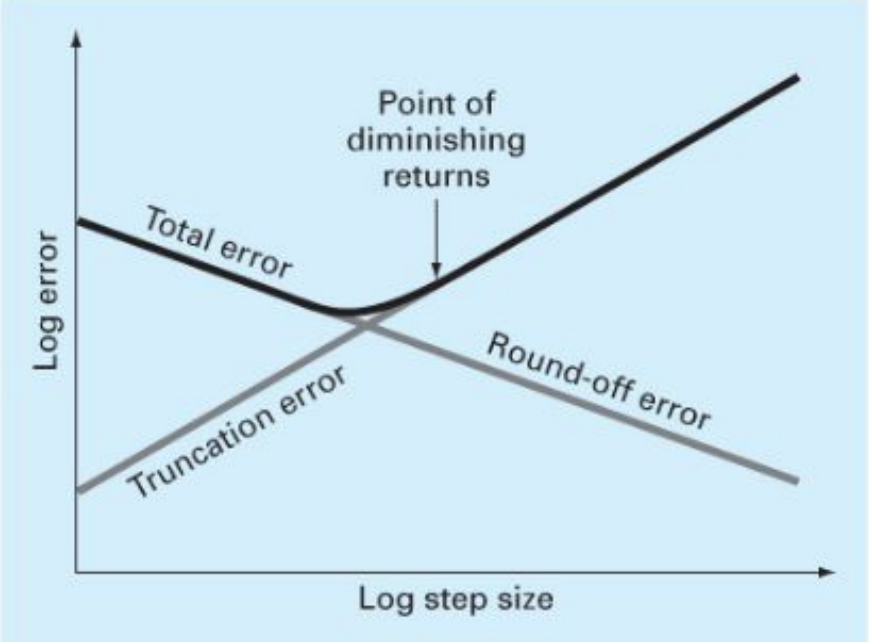
$$f'(x) = -0.4x^3 - 0.45x^2 - x - 0.15$$

$$f'(0.5) = -0.8125$$

Calculating hOpt

The smaller the H value (step size) the more accurate the central finite difference formula will be however when using a computer with floating point numbers truncation and rounding errors are present and there exists a point of diminishing returns in terms of the H value.

This is shown the graph below.



The theoretical optimum H value can be calculated as follows.

$$h_{opt} = \sqrt[3]{\frac{3}{M}}$$

Epsilon is the machine epsilon which an upper bound on the relative error due to rounding in floating point arithmetic. M is the third derivatives maximum **absolute** value

Epsilon can be retrieved by calling the following function (note the float datatype is used for this question)

std::numeric_limits<float>::epsilon()

The third derivative of the function at the start of the question is

$$f'''(x) = -2.4x - 0.9$$

The x value is 0.5

$$f'''(0.5) = -2.1$$

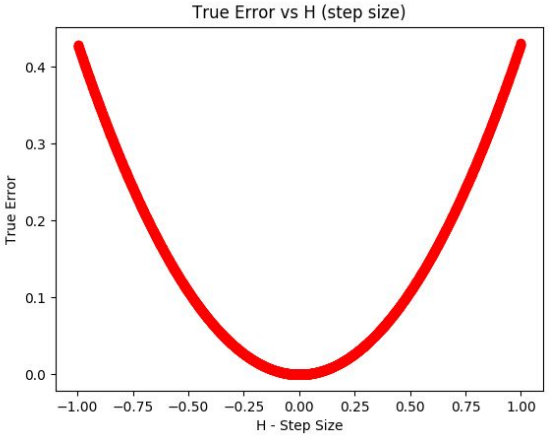
Therefore our value for M is 2.1, by substituting the value for M and machine epsilon the optimum H value is ***0.00554290413856506348***

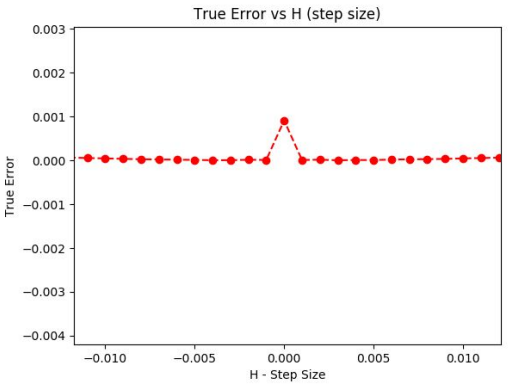
Experiment

The program will use multiple values for H, it is expected that the most accurate result has a similar value to that which was calculated previously.

There will be 2000 data points with a step size of 0.001, the values of H will range from -1 to +1.

Results





The Above charts show that as the step size decreases so does the true error, as expected. An interesting point is when h = 0 there is a spike as seen on the above right. You would expect a divide by 0 error to occur however because h is a floating point number that means that a true 0 value cannot be represented. Therefor when x is supposed to be 0 it is an incredibly small value essentially cancelling out the h values in the central finite difference formula, which is why the increase in error occurs

The experimental h value and the theoretical optimum h are fairly close together, they may not be exact due to the step sizes in the program. I have calculated the derivative with the theoretical optimum H and the results are shown below.

$$value = -0.81251108646392822266$$
$$error = 0.00136446929536759853$$

From the graph and values above it is clear that the theoretical optimum h value was in fact not the exact experimental optimum, however the derivative value is extremely close with a such a low error. Due to the small margin of error I would say that the theoretical optimum value for H is valid.

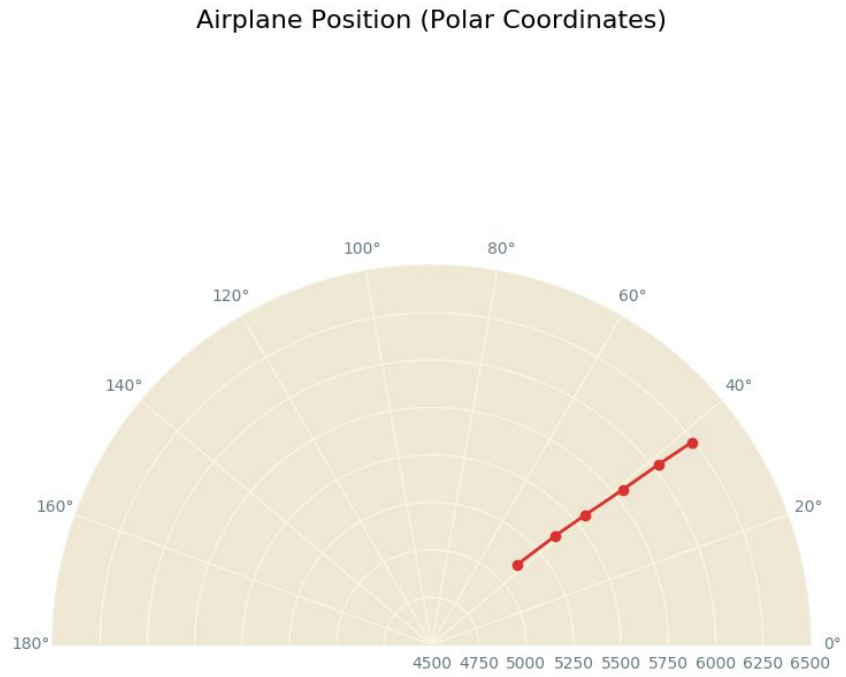
An explanation for the differing values is the actual value for the machine epsilon and M, another is that the hOpt value was calculated on the computer and as a result truncation and rounding errors would have occured.

Question 2

Given the position of a plane over time in polar coordinates calculate the velocity and acceleration.

Plane position (polar)

T, s	200	202	204	206	208	210
Θ, rad	0.75	0.72	0.70	0.68	0.67	0.66
R, m	5120	5370	5560	5800	6030	6240



Converted to cartesian (x, y)

T, s	200	202	204	206	208	210
x	3746.24701	4037.2	4252.52	4509.92	4726.44	4929.55
y	3489.99045	3540.9	3581.85	3647	3744.55	3825.85

$x = r * cos(\Theta)$

$y = r * sin(\Theta)$

The question asks to calculate the velocity and acceleration vectors of the plane based off its position at a given time.

$position\ (x,\ y) = d(t)$

$$velocity\ (x,\ y)\ =\ \frac{dposition}{dt}$$
$$acceleration\ (x,\ y)\ =\ \frac{dvelocity}{dt}$$

The above tables show a function in terms of time which returns the position (x, y).
The first derivative of this function would then be the velocity and second derivative the acceleration.

The function of position in terms of time is not provided so a numerical approach must be used based off the data in the tables.

Centered Finite Differences

The centered finite differences method for approximating the derivatives will be used for this question.

This method is a combination of the forward and backward finite difference formulas, the formula is shown below.

$$f(x)\ =\ \frac{f(x+h)-f(x-h)}{2h}$$

Using the table of data h will equal 1, instead of calculating the values for f(x) a vector lookup will be performed where x is the vector index.

The formula above can be used to calculate the second derivative but there is also another more accurate formula show below.

$$f''(x) \cong \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$$

This formula has an error of

$$E\ =\ -\frac{1}{12}h^2 f''''(\xi)$$

This will also be included in the results.

Pseudocode

```
Function firstDerivative(a, b)
        H =abs(a.getTime() - b.getTime()) / 2

        xPrime = (b.x - a.x) / (2 * h)
        yPrime = (b.y - a.y) / (2 * h)

        Return (xPrime, yPrime)

Function secondDerivative(v1, v2, h)
```

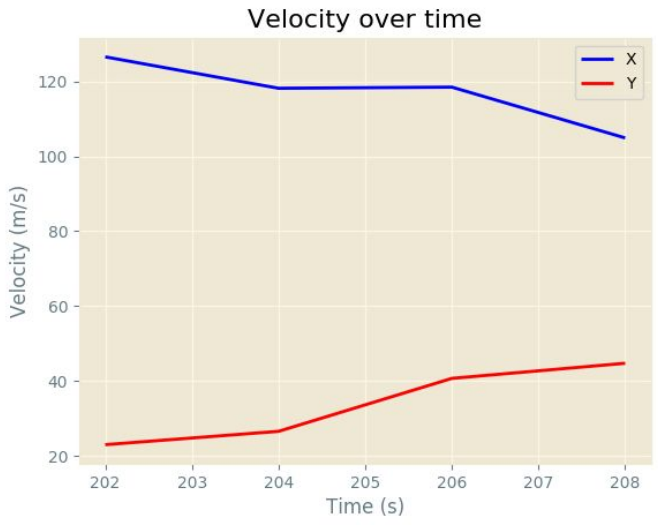
```
        xPrime = (v2.x - v1.x) / (2 * h)
        yPrime = (v2.y - v1.y) / (2 * h)

        Return (xPrime, yPrime)



// more accurate
Function secondDerivativev2(r[records], i)
        H = 2.0
        Left = r[i - 1].toCartesian()
        Middle = r[i].toCartesian()
        Right = r[i + 1].toCartesian()

        xPrime = (right.x - 2 * middle.x + left.x) / (h * h)
        yPrime = (right.y - 2 * middle.y + left.y) / (h * h)

        Return (xPrime, yPrime)

Function calcVelocities(r[records])
        Velocites[(x, y)]

        For i -> r.size() - 1
                velocities.push(firstDerivative(r[i - 1], r[i + 1]))

        Return velocities


Function calcAccelerations(v[(x, y)], h)
        Accelerations[(x, y)]

        For i  = 1 -> v.size() - 1
                accelerations.push(secondDerivative(v[i - 1], v[i + 1], h))

        Return accelerations
```

Results

As shown in the formula above to calculate a derivative at a single point the values to the left and right are required, as a result the first derivative has 2 less data points as the raw data, and the second derivative 2 less than of the first derivative. The alternative formula is also used to calculate the second derivative from the original function and shown below.

Velocity

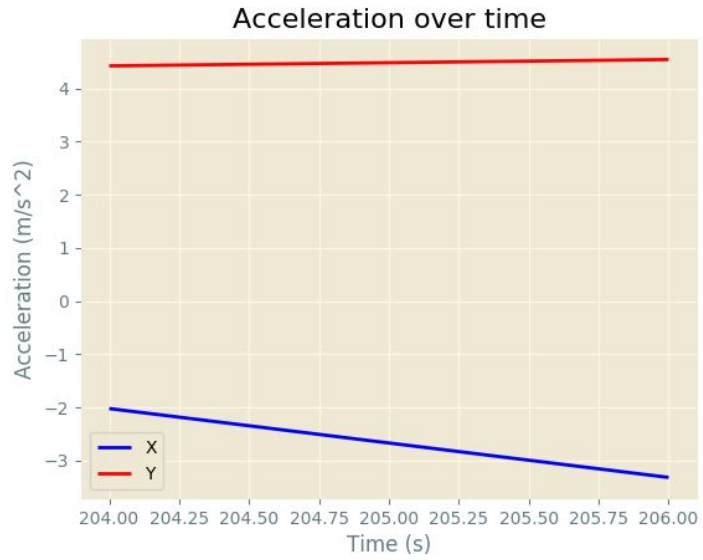
Velocity		
Time	**X**	**Y**
202	126.569	22.965
204	118.181	26.526
206	118.481	40.6738
208	104.907	44.7124



Acceleration

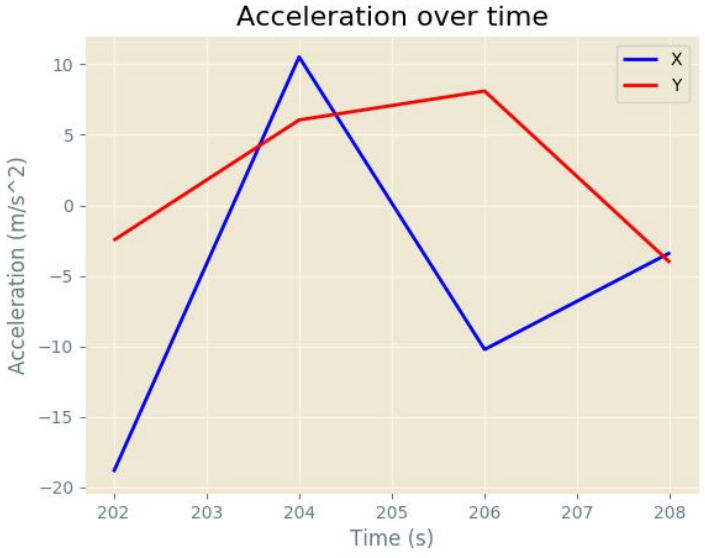
Using the initial formula

Acceleration		
Time	**X**	**Y**
204	-2.02209	4.4272
206	-3.31845	4.54661



Using the more accurate formula

Acceleration		
Time	**X**	**Y**
202	-18.906	-2.48765
204	10.5184	6.04864
206	-10.2191	8.09919
208	-3.354	-4.06058



As you can see there are 4 data points vs 2 data points using this formula, in the previous graphs the values were very linear whereas this graph is not. This graph is the more accurate representation of the planes acceleration over time.

Question 3

Definite integral function

The following function would be used to find the definite integral (area under the curve) of the function H, this would equal the cross sectional area of the river in this questions context. 0 corresponds to the bank of the river and x being the distance from the the river bank.

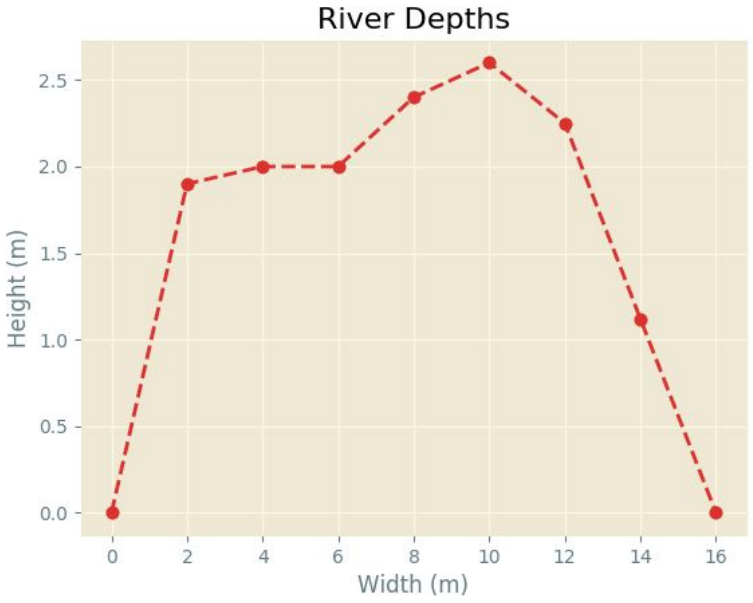
$$a_c = \int_0^x H(x)\,dx$$

Table of values

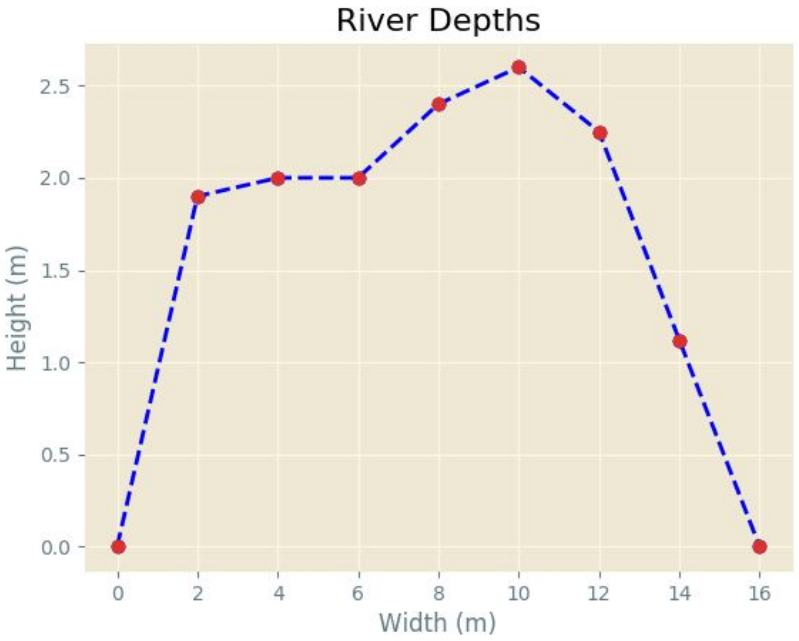
The question does not provide a function of H in terms of X but instead the following table of values, the above function is the analytical solution, a numerical integration method will need to be used with the below data to approximate the analytical solution.

River Depth									
X, m	0	2	4	6	8	10	12	14	16
H, m	0	1.9	2	2	2.4	2.6	2.25	1.12	0

Below is the result of graphing the table



A polynomial was then fit to this curve using numpy the result is below, as you can see this fits the data exactly. The purpose behind this is to have a function to calculate a definite integrate to check for reasonableness.



The coefficients in the equation for the above line are
[5.07657490e-07 -3.26605903e-05 8.58723958e-04 -1.17100694e-02
  8.54414063e-02 -2.95553819e-01 1.96962302e-01 1.21666667e+00
 -5.30538576e-13]

Romberg Integration

Romberg integration is based around the multipart trapezoidal rule and extrapolating the results. With each iteration the step size of the multi part trapezoidal rule is halved - these values are then extrapolated this is shown below

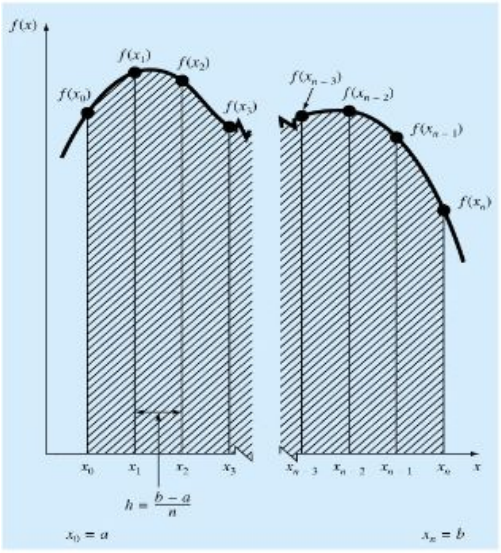
Step size	Multipart trapezoidal	Extrapolated	Extrapolated
H	VALUE		
H/2	VALUE	VALUE	
H/4	VALUE	VALUE	VALUE

As H approaches 0 the values becomes more accurate however it may not be viable for H to be that ‘small’ this could be because the function is too computationally intensive or in this case we do not have a function and instead a table of values.

Multipart trapezoidal rule

The multipart trapezoidal rule estimates the integral (area under the curve) of a function by using specific points from a function and using these points to create a trapezium, these trapeziums areas are then summed. The formula is shown below as well as a graphical representation. The smaller the step size (width) between points the better the approximate of the function is and therefore the integral.

$$I = \frac{b-a}{2n}\left[f(x_0) + f(x_n) + 2\sum_{i=1}^{n-1} f(x_i) \right]$$



Pseudocode

Multipart Trapezoidal

```
Function trapezoidal(data[(x, y)], h)
        Result = data[0].x + data[-1].y

        For i=1; i < data.size / h; i ++
                Result += 2 * data[i * h].y

        // this is normally (h / 2) * result but the way ‘data’ is structured h is already (h / 2)
        Return h * result
```

Romberg Integration

```
// 2d array and dynamic allocation is omitted

Function integrate(data[(x, y)]
        N = 4
        H = 8 // is really 16 but converted to index is 8

        For i = 0 -> n
                Table[i][0] = trapezoidal(data, h)

                For j = 1 -> i
                        // extrapolation
                        Table[i][j] = table[i][j - 1] + (table[i][j - 1] - table[i - 1][j - 1]) / (4^j - 1);
                // halve h
                H /= 2

        Return table[n][n + 1]
```

Romberg Table

Below is the full romberg table with the lowest possible h value of 2 (1 index)

H- step size	**Multipart trapezoidal**	**Extrapolation**		
16	0			
8	19.2	25.6		
4	26.6	29.06666667	29.29777778	
2	28.54	29.18666667	29.19466667	29.19302998

Relative error (stopping criteria)

The relative error is useful when using romberg integration as it can be used as a stopping criteria. It compares the difference between the previous and current approximation, the formula is shown below.

$$\left|\frac{current - previous}{current}\right| * 100\%$$

Generally an exact answer is too computationally expensive and is not needed, instead a value with say a 0.001% relative error is good enough.

This can be done by adding the following code under the extrapolation. Where Es is the relative error

```
if ((fabs(table[i][j] - table[i][j - 1]) / table[i][j]) * 100 < Es)
        return table[i][j];
```

Solution

The question stated however to find the value with a 1% relative error criterion (Es = 1.0) in which case the answer is **29.29777778** this is value is highlighted green in the above table.

Reasonableness of solution

By using the equation of the fitted polynomial and analytically calculating the definite integral

$$a_c = \int_0^{1.6} H(x)\,dx$$

The analytical solution is very close to the numerical solution and therefore is a reasonable method. The solution had a stopping criteria of 1%, more accurate results were calculated and shown in the table. The level of accuracy required is determined by the context of the problem and 1% is what was determined.