HL MATH INTERNAL ASSESSMENT

Method to model the curve of a 1957 Coca-Cola bottle with a single continuous polynomial

1 Introduction

I have always been fascinated by the Coca-Cola bottle. I still remember when I was little, my family used to take me to the countryside every weekend to visit my grandparents, and every time they would treat me with a cold refreshing bottle of Coca-Cola. Therefore, I have always associated Coca-Cola with the fond memories of my grandparents. One time, after finishing the beverage, I remember holding the glass bottle in my hand and looking at it curiously; the design of the bottle shape seemed so familiar yet so different from other beverage bottles; I felt the curvy shape had a special elegance to it. I was fascinated by this beautiful curve, and ever since I developed a curiosity for math, I have always wanted to try and model the curve of the Cola bottle, but never knew how to. But now I have acquired enough knowledge to be confident at trying it, and therefore, in this exploration, I set out to model the Cola bottle curve with a bit of calculus knowledge by developing my own general method, which can then be used to model other smooth curves as well. I chose to model this based on a 1957 glass bottle, hailed as the "special holy grail" of Cola bottles by collectors, because it was the first time the branding name Coca-Cola was no longer embossed but etched directly in white onto the bottle (Glancey). I obtained the image of the bottle online, from this source image:

Figure 1: Image of Coke bottles evolution through the years (Bertie)

Using the image of the 1957 bottle, I could insert into an online graphing calculator, and using

the knowledge of calculus and with a bit of data processing I will be able to find the equation that approximates the curve of the bottle.

2 Investigation

2.1 Obtaining data points

First I crop the 1957 bottle from the image and put it into Desmos. I choose Desmos because I have had much experience with it, but any other advanced graphing software will do. Then I center the image so that the y-axis lies approximately in the middle of the bottle.

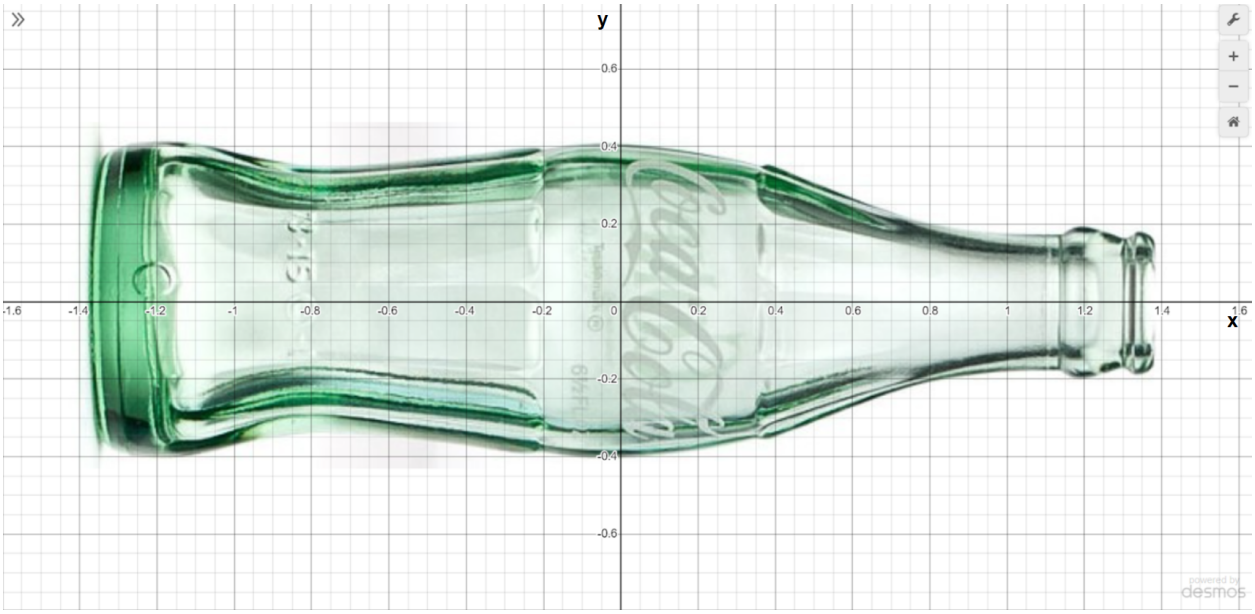

Figure 2: Image of 1957 Coke bottle, cropped and centered in Desmos

Then I zoom in close and put points along the outside curve of the bottle. The points are evenly spaced with a distance of $d = 0.05$ units, and I keep the accuracy of y coordinate of the points up to 2 decimal places, however the distance can be made smaller and the accuracy of y coordinate can be increased for higher accuracy in calculations. Theoretically this would work best if the distance between the points becomes as small as possible, but then it would require a lot more data points and be very computationally intensive. Therefore, for the sake of this exploration, I chose a distance of 0.05 units as I feel it's small enough to reliably approximate the relatively smooth curve of the bottle. These points are put in a table (x_1, y_1) (included in Appendix).

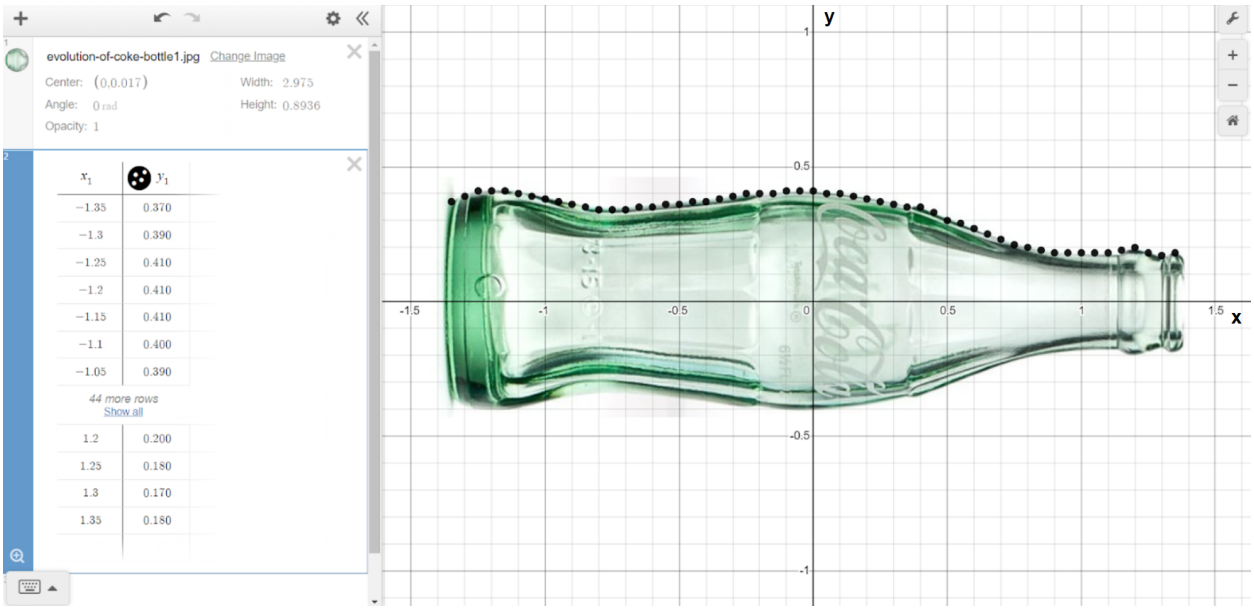

Figure 3: Data points placed along the curve of the bottle.

2.2 Method to approximate the curve

Now, from the points tracing out the curve of the bottle, there are a few approaches I can take to model this curve, such as using piecewise functions, manual curve fittings, etc. However, most of those approaches are pretty laborious, take a lot of steps, involve a lot of "guessing game", and the final function is often in the form of a piecewise function, which further complicates later calculations and manipulations of the function. Therefore, I aimed to approximate this with one single continuous polynomial, more specifically using a **Maclaurin series** to make the calculations less complicated.

Any function $f(x)$ can be expressed as a sum of an infinite series of polynomials (a Maclaurin Series) like so

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 + \ldots \frac{f^{(n)}(0)}{n!}x^n + \ldots$$

Where $f^{(n)}(x)$ is the n^{th} order derivative of the function $f(x)$. (Wolfram Research, Inc.)

In order to find the Maclaurin series, we need to know higher derivatives at point $x_0 = 0$. With discrete points, it is impossible to find the exact derivative at a point; therefore we can only approximate the derivative. There are a few ways to do this, but the approximation I eventually

settled on is using the Central Difference Quotients (Thomas et al.) as illustrated by Figure 4.

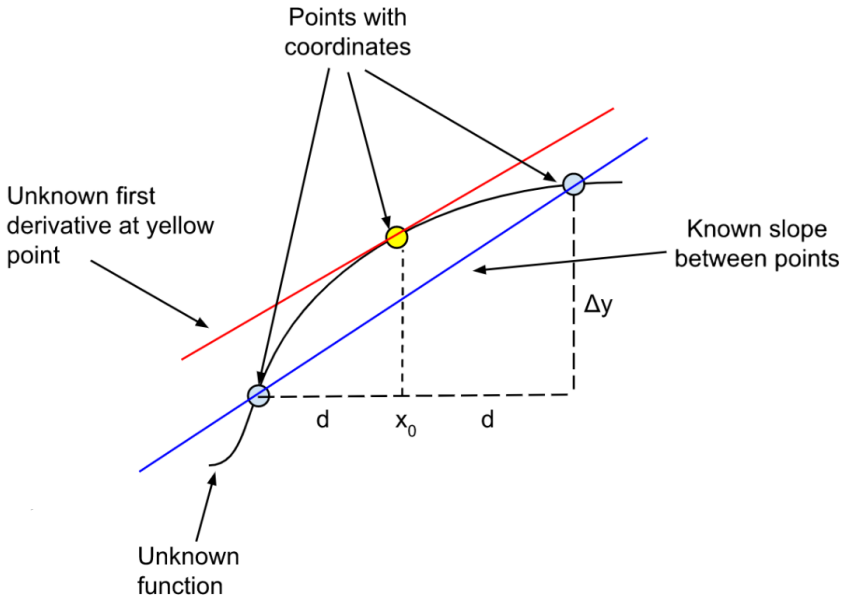

Figure 4: Approximation of derivative with Central Difference Quotient

Explanation: For a relatively smooth small portion of the curve with no sharp turning points, the derivative of the yellow point can be approximated by using the gradient between its neighboring points. This approximation will be more accurate as the distance between the points gets smaller.

$$f'(x_0) = \frac{df(x_0)}{dx} \simeq \frac{\Delta y}{\Delta x}\Big|_{x_0} = \frac{f(x_0+d) - f(x_0-d)}{2d} \qquad (*)$$

From this method, I can now approximate the first derivative at any point. The rate of change of first derivatives can then be used similarly to calculate the second derivatives, third, etc. and so on.

$$f^{n+1}(x) = \frac{df^n(x)}{dx} \simeq \frac{\Delta f^n(x)}{\Delta x} \simeq \frac{f^n(x+d) - f^n(x-d)}{2d}$$

However, there is a significant drawback to this method, which is to calculate the n^{th} derivative at a point, you would need to calculate the derivatives up to order of $(n-1)$ of every other data point. This is indeed very intensive and very slow. Therefore, I set out to find an easier approach. And after a bit of manual manipulations with recursion and some observations, I came up with

this formula:

For a set of data points (x_0, y_0), (x_1, y_1), (x_2, y_2),... the n^{th} derivative of approximating function $f(x)$ at point x is

$$f^n(x) = \left(\frac{1}{2d}\right)^n \cdot \sum_{i=0}^{n} \left((-1)^i \binom{n}{i} f(x + d(n - 2i))\right) \tag{1}$$

Explanation: I first found this formula from pure observation. To demonstrate, in the example below, I have the x and y values of 7 random points, the x coordinates are evenly spaced out with a distance $d = 0.05$. At each point, I calculate the derivative by taking the slope of neighboring bounding points:

C3 | fx | =(B4-B2)/0.1

	A	B	C	D	E	F
1	x	y(x)	1st derivative	2nd derivative	3rd derivative	4th derivative
2	0.4	0.5				
3	0.45	0.51	0.2			
4	0.5	0.52	0.1	-2		
5	0.55	0.52	0	-1	10	
6	0.6	0.52	0	-1	0	0
7	0.65	0.52	-0.1	-1	10	
8	0.7	0.51	-0.1	0		
9	0.75	0.51	-0.1			
10	0.8	0.5				
11						

Figure 5: Spreadsheet data illustrating my formula of approximating higher-order derivatives

For example, if I need to calculate the 2nd derivative at $x = 0.5$ (cell $C3$), I need to first calculate the rate of change of the 1st derivatives at $x = 0.45$ and $x = 0.55$.

To calculate the 1st derivative at $x = 0.45$, I calculated the gradient between neighboring points $(x, y) = (0.4, 0.5)$ and $(x, y) = (0.5, 0.52)$. Using the Central Difference Quotient method, this can be done with the formula shown in Figure 5, $C3 = \frac{B4-B2}{0.1}$. Similarly, 1st derivative at $x = 0.55$ can be calculated with formula $C5 = \frac{B6-B4}{0.1}$.

Therefore, to calculate 2nd derivative at $x = 0.5$ using the formula $D4 = \frac{C5-C3}{0.1}$, I can substitute $C5$ and $C3$, and get $D4 = \frac{B6-2\cdot B4+B2}{0.1^2}$.

Repeating the process, if I need to calculate the 3rd derivative at $x = 0.55$, I get the formula $E5 = \frac{B8-3\cdot B6+3\cdot B4-B2}{0.1^3}$. Similarly, for the 4th derivative at $x = 0.6$, I get $F6 = \frac{B10-4\cdot B8+6\cdot B6-4\cdot B4+B2}{0.1^4}$.

After manually writing out the formula like this for a few higher derivatives, I am able to guess the general form of the formula for the n^{th} derivative as shown in Equation (1):

$$f^n(x) = \left(\frac{1}{2d}\right)^n \cdot \sum_{i=0}^{n} \left((-1)^i \binom{n}{i} f(x + d(n-2i)) \right)$$

Here we can see the common multiplier is simply reciprocal of twice the distance d between the points, raised to the n^{th}: $\left(\frac{1}{2d}\right)^n$. For the terms, it is easy to see the coefficient of each term is simply a binomial expansion with alternating signs. The individual terms are y values of points spread out from the center point of x chosen, and are spaced out by the term $d(n-2i)$

Now that I have made a good guess of the general form of the formula, then I seek out to prove the formula:

Prove:

$$P(n) : f^n(x) = \left(\frac{1}{2d}\right)^n \cdot \sum_{i=0}^{n} \left((-1)^i \binom{n}{i} f(x + d(n-2i)) \right) \forall n \in \mathbb{Z}_+$$

Proof. We will prove this with induction

Base case: $P(n = 1)$

$$\begin{aligned}
f^1(x) &= \left(\frac{1}{2d}\right)^1 \cdot \sum_{i=0}^{1} \left((-1)^i \binom{1}{i} f(x + d(1-2i)) \right) \\
&= \frac{1}{2d} \left((-1)^0 \binom{1}{0} f(x+d) + (-1)^1 \binom{1}{1} f(x + d(-1)) \right) \\
&= \frac{1}{2d} \left(\binom{1}{0} f(x+d) - \binom{1}{1} f(x + d(-1)) \right) \\
&= \frac{1}{2d} \left(f(x+d) - f(x-d) \right) \\
&= \frac{f(x+d) - f(x-d)}{2d}
\end{aligned}$$

Which is the same as equation $(*)$

So $P(n)$ is correct for base case $n = 1$.

Assume correctness for

$$P(n = p) : f^p(x) = \left(\frac{1}{2d}\right)^p \cdot \sum_{i=0}^{p} \left((-1)^i \binom{p}{i} f(x + d(p-2i)) \right)$$

Need to prove correctness for $P(n = p + 1)$

From Central Difference Quotient for higher derivatives, we have:

$$f^{p+1}(x) = \frac{f^p(x+d) - f^p(x-d)}{2d}$$

So we need to prove

$$\frac{f^p(x+d) - f^p(x-d)}{2d} \stackrel{?}{=} \left(\frac{1}{2d}\right)^{p+1} \cdot \sum_{i=0}^{p+1} \left((-1)^i \binom{p+1}{i} f(x + d(p-2i+1))\right) \tag{2}$$

We have

$$f^p(x+d) = \left(\frac{1}{2d}\right)^{p} \cdot \sum_{i=0}^{p} \left((-1)^i \binom{p}{i} f(x + d(p-2i+1))\right)$$

and similarly

$$f^p(x-d) = \left(\frac{1}{2d}\right)^{p} \cdot \sum_{i=0}^{p} \left((-1)^i \binom{p}{i} f(x + d(p-2i-1))\right)$$

Therefore from (2) we need to prove

$$\left(\frac{1}{2d}\right)^{p+1} \left(\sum_{i=0}^{p} \left((-1)^i \binom{p}{i} f(x + d(p-2i+1))\right) - \sum_{i=0}^{p} \left((-1)^i \binom{p}{i} f(x + d(p-2i-1))\right)\right)$$
$$\stackrel{?}{=} \left(\frac{1}{2d}\right)^{p+1} \cdot \sum_{i=0}^{p+1} \left((-1)^i \binom{p+1}{i} f(x + d(p-2i+1))\right)$$

Or simplified as

$$\sum_{i=0}^{p} \left((-1)^i \binom{p}{i} f(x + d(p-2i+1))\right) - \sum_{i=0}^{p} \left((-1)^i \binom{p}{i} f(x + d(p-2i-1))\right)$$
$$\stackrel{?}{=} \sum_{i=0}^{p+1} \left((-1)^i \binom{p+1}{i} f(x + d(p-2i+1))\right) \tag{3}$$

Next we can expand the summations on LHS and combine like terms.

Without loss of generality, assume p odd

$$\begin{aligned}
LHS = &\left((-1)^0\binom{p}{0}f(x+d(p+1)) + (-1)^1\binom{p}{1}f(x+d(p-1)) + (-1)^2\binom{p}{2}f(x+d(p-3)) + \cdots\right.\\
&\left. + (-1)^p\binom{p}{p}f(x+d(1-p))\right) - \left((-1)^0\binom{p}{0}f(x+d(p-1)) + (-1)^1\binom{p}{1}f(x+d(p-3))\right.\\
&\left. + \cdots + (-1)^{p-1}\binom{p}{p-1}f(x+d(1-p)) + (-1)^p\binom{p}{p}f(x+d(-1-p))\right)\\
= &\, f(x+d(p+1)) - \left(\binom{p}{1}+\binom{p}{0}\right)f(x+d(p-1)) + \left(\binom{p}{2}+\binom{p}{1}\right)f(x+d(p-3))\\
&- \cdots - \left(\binom{p}{p}+\binom{p}{p-1}\right)f(x+d(1-p)) + f(x+d(-1-p))
\end{aligned} \tag{4}$$

The coefficients of terms has the general form

$$\binom{a}{b} + \binom{a}{b-1}$$

$$\begin{aligned}
&= \frac{a!}{(a-b)!\,b!} + \frac{a!}{(a-b+1)!\,(b-1)!} = \frac{a!\,(a-b+1)}{(a-b+1)!\,b!} + \frac{a!\,b}{(a-b+1)!\,b!} = \frac{a!}{(a-b+1)!\,b!}((a-b+1)+b)\\
&= \frac{a!}{(a-b+1)!\,b!}(a+1) = \frac{(a+1)!}{(a-b+1)!\,b!} = \binom{a+1}{b}
\end{aligned}$$

Therefore

$$\begin{aligned}
(4) &= \binom{p+1}{0}f(x+d(p+1)) - \binom{p+1}{1}f(x+d(p-1)) + \binom{p+1}{2}f(x+d(p-3))\\
&\quad - \cdots - \binom{p+1}{p}f(x+d(1-p)) + \binom{p+1}{p+1}f(x+d(-1-p))\\
&= \sum_{i=0}^{p+1}\left((-1)^i\binom{p+1}{i}f(x+d(p-2i+1))\right)\\
&= RHS
\end{aligned}$$

So (3) is proven, therefore (2) is correct:

$$f^{p+1}(x) = \frac{f^p(x+d) - f^p(x-d)}{2d} = \left(\frac{1}{2d}\right)^{p+1} \cdot \sum_{i=0}^{p+1}\left((-1)^i\,\binom{p+1}{i}f(x+d(p-2i+1))\right)$$

So we have proven correctness for $P(n=p+1)$

Therefore, by principle of induction, $P(n)$ is true $\forall n \in \mathbb{Z}_+$ $\square$

Once the formula is proven, I can start using it in my Maclaurin Series approximation.

The general form of a Maclaurin Series is

$$P(x) = \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!} x^n$$

(Maclaurin Series — Brilliant Math and Science Wiki)

After substituting in the formula for $f^n(x_0)$ we get

$$P(x) = \sum_{n=0}^{\infty} \left(\frac{1}{n!} \left(\left(\frac{1}{2d} \right)^n \cdot \sum_{i=0}^{n} \left((-1)^i \binom{n}{i} f(x + d(n-2i)) \right) \right) x^n \right)$$

Which can then be rewritten as

$$\begin{aligned} P(x) &= \sum_{n=0}^{\infty} \left(\frac{1}{n!} \left(\left(\frac{1}{2d} \right)^n \cdot \sum_{i=0}^{n} \left((-1)^i \frac{n!}{(n-i)!\, i!} f(x + d(n-2i)) \right) \right) x^n \right) \\ &= \sum_{n=0}^{\infty} \left(\left(\frac{1}{2d} \right)^n \cdot \sum_{i=0}^{n} \left(\frac{1}{n!} (-1)^i \frac{n!}{(n-i)!\, i!} f(x + d(n-2i)) \right) x^n \right) \\ &= \sum_{n=0}^{\infty} \left(\left(\frac{1}{2d} \right)^n \cdot \sum_{i=0}^{n} \left(\frac{(-1)^i}{(n-i)!\, i!} f(x + d(n-2i)) \right) x^n \right) \end{aligned}$$

This is the final polynomial approximation formula that I will use.

2.3 Modelling the curve

Now I've got the general formula for the approximation, I can put this into Desmos and change a few variables as necessary. In this method, to approximate up to order n we need $2n + 1$ points for the calculations, so I have to extend the two endpoints of the curve. However, these points are just there for the function to calculate the gradients of the "useful" points that model our curve, and their y coordinates can be any random values. For simplicity's sake I choose them to be straight out horizontally, but this doesn't affect anything to our method.

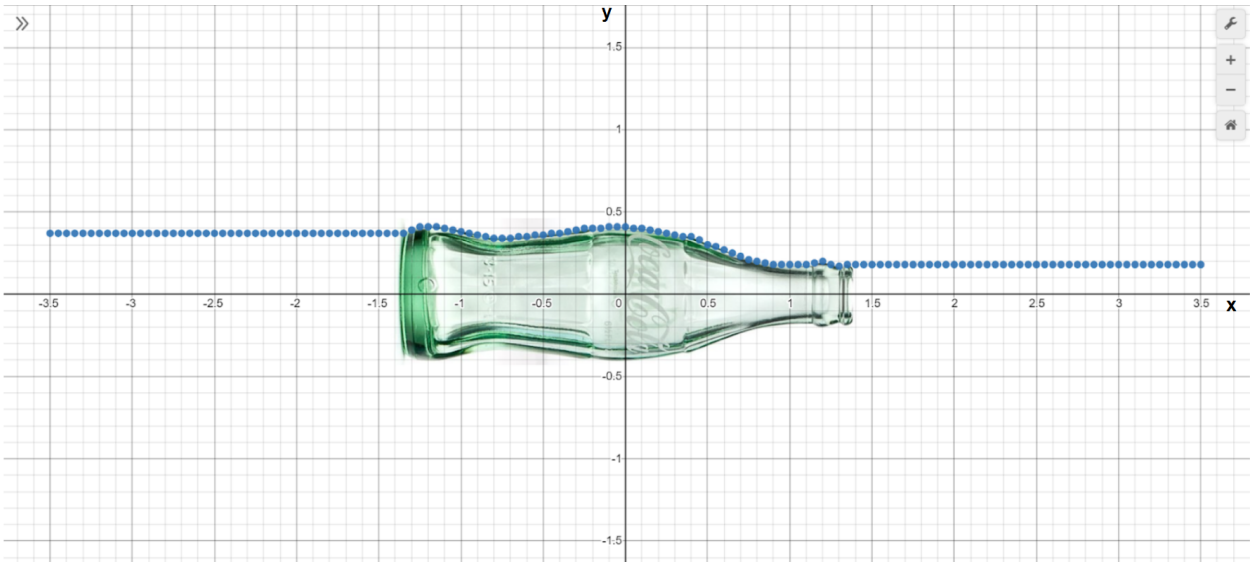

Figure 6: Extended data points from both ends

Once I put the general formula into Desmos, I have to make a few changes. Since Desmos coordinate table uses an indexing system, I have to change the notation for $f(x + d(n - 2i))$ into the corresponding index: $y_1[71 + (n - 2i)]$. Here, 71 is the index of the center x point ($x = 0$). Also, there are 141 points, so this can approximate up to 70^{th} order, so I change the higher bound of the outer summation from infinity to 70. The distance d is 0.05, so I replace $\frac{1}{2d} = 10$ in order to reduce the amount of calculation it has to do. Therefore, finally I get this final formula that I can put into Desmos and sketch it.

$$y = \sum_{n=0}^{70} \left(10^n \sum_{i=0}^{n} \left(\frac{(-1)^i}{i!\,(n-i)!} y_1[71 + (n - 2i)] \right) x^n \right)$$

This is the result after sketching:

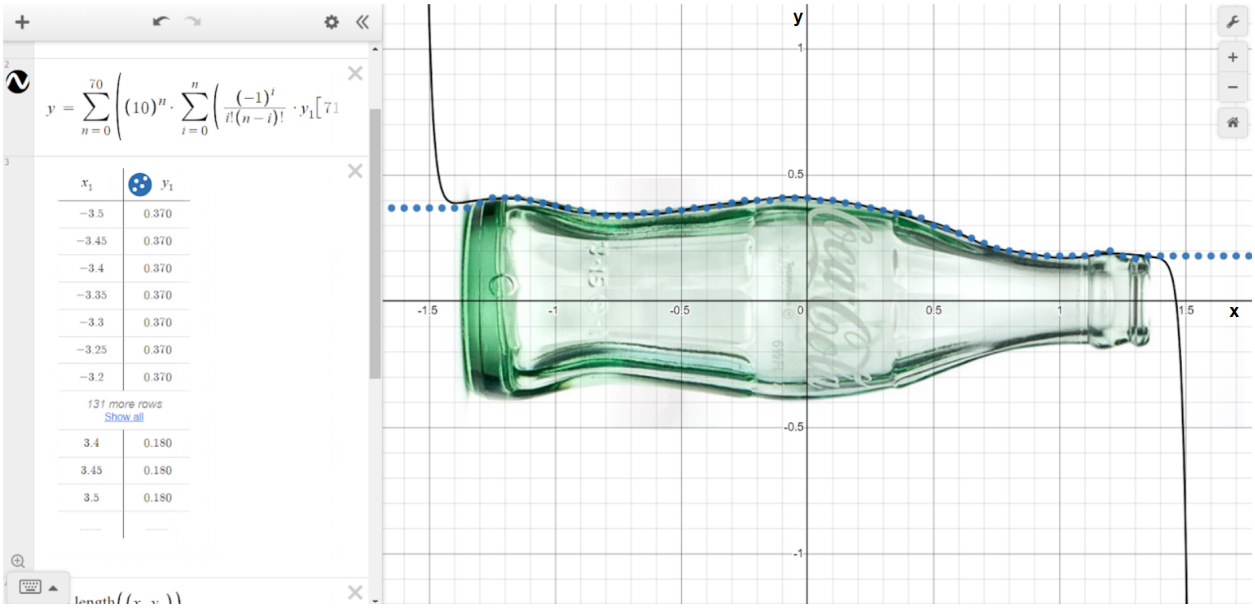

Figure 7: The function approximating the bottle curve

As we can see, the function fits the data points and the curve of the bottle pretty well. The function is able to follow all the main sections of the curve and approximate the shape of the bottle well, and there are no irregular behaviors. However, upon closer inspection, we see that the curve doesn't pass through all the points.

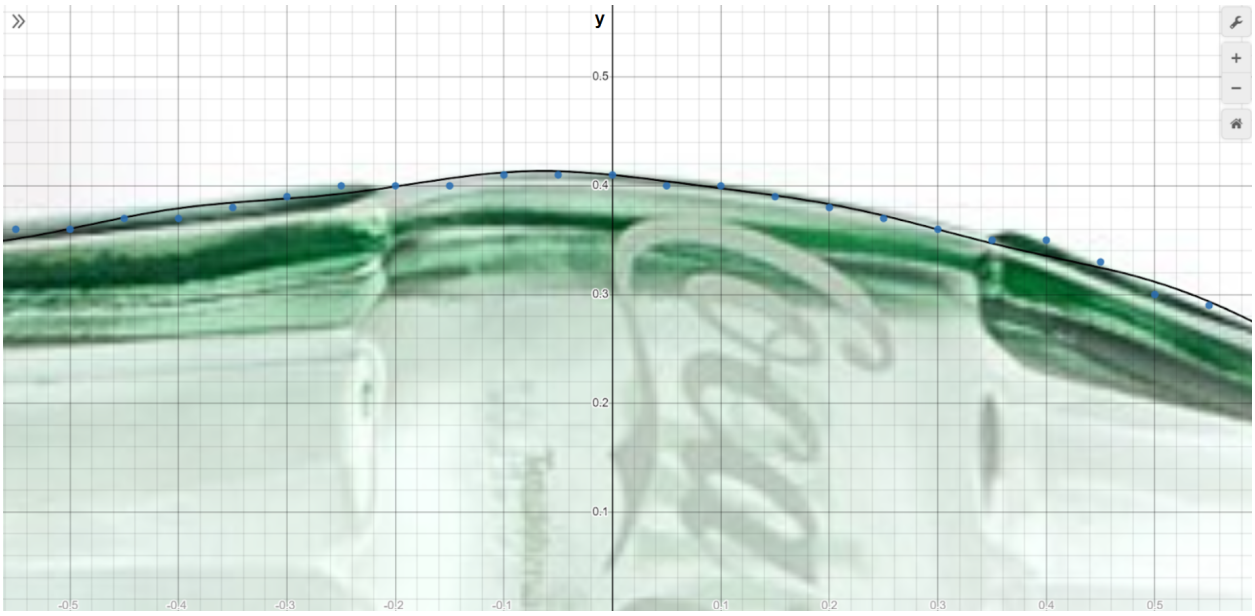

Figure 8: Approximating function not passing through all points

This is because my method isn't a complete polynomial interpolation based directly on the coordi-

nates of the points, but instead uses the points indirectly to approximate higher order derivatives which then is used in the Maclaurin Series. Therefore, if a model needs a high degree of accuracy and the curve has to pass through all points then this method won't be suitable. Also, I observe that the sharper smaller curves in the shape of the bottle (such as near the opening) aren't approximated perfectly.

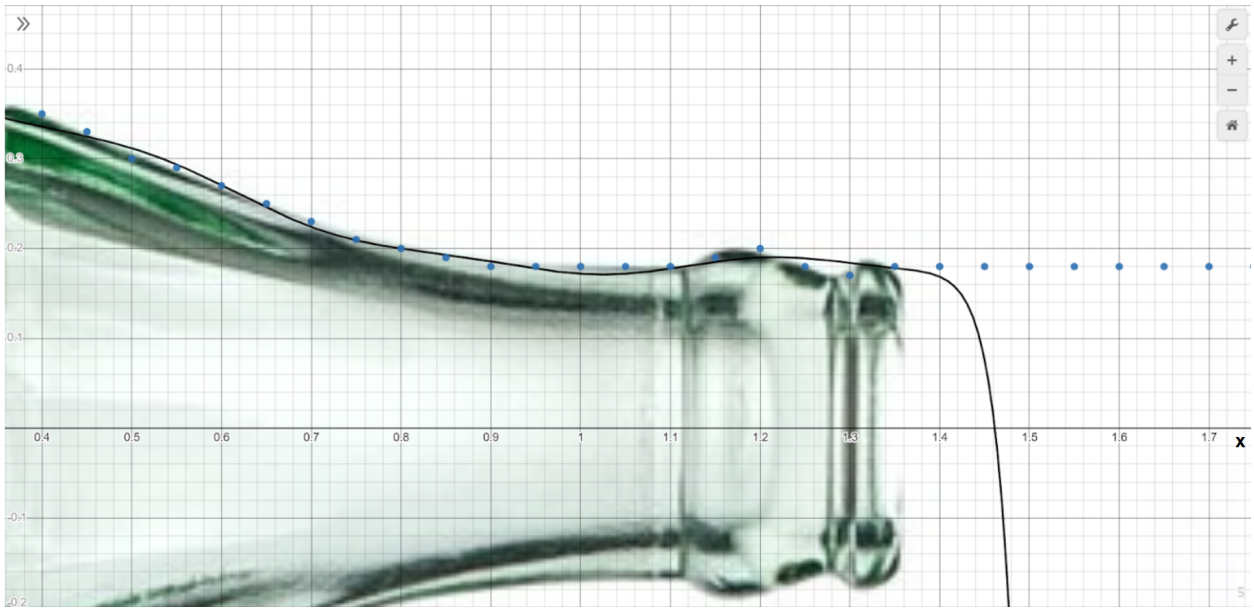

Figure 9: Close up of the bottle opening

I believe this is because there are not enough data points to model the sharp changes in curvature of this section of the curve. I think this can be improved by decreasing the distance between the points, making them closer together, however this will increase the number of points and therefore increase computational time.

There are a few optimizations that can be made to improve the speed of this method. Originally, on my laptop it takes around 30-45 seconds for Desmos to sketch out the full curve. I believe this is relatively slow, mainly because of the nesting summations, bringing this to a time complexity of $O(n^2)$. Another bottleneck in the algorithm could be the calculation of the factorials $i!\,(n-i)!$. For the inner summation alone, for example a value of $b!$ is actually computed twice, once as $i!$ and again as $(n-i)!$, and the speed of this factorial is highly dependent on the algorithm that the graphing program uses. In my case of using Desmos, I couldn't find any document about their factorial algorithm, but this can be up to $O(n)$ speed. However, I believe this bottleneck can be

solved algorithmically by pre-calculating and storing factorials up to about 100! or even higher in a mapping table which can then be queried during the calculation instead of calculating the factorials directly. This will help reduce the factorial speed down to a constant $O(1)$ and improve the overall algorithm speed.

Another drawback of my method that I can notice is the points have to be evenly spaced. Although this simplifies the task of placing the points and makes the calculations more manageable, it requires choosing a suitable point distance beforehand. If the distance is large, it is adequate to approximate smoother curves but will not be able to accurately model sharp changes or smaller sections in the curve. If the distance is small, the number of data points will be very large, and it will slow down the calculation. If I can figure out a way to be able to change the point distance for different segments of the curve, that would solve this problem, because for smoother curves I can place the points further apart and for tighter smaller curves I can place them closer together.

2.4 Comparison with other polynomial interpolation methods

After completing the modelling, I did some additional research and found there are two quite similar methods of polynomial interpolation: Lagrange polynomials and Newton polynomials. These both use a form of difference quotients, but the rest of their methods differ quite a bit from mine. Therefore, I will compare the result of my method with these 2 common interpolations.

Lagrange polynomial approximation formula I use in Desmos:

$$L(x) = \sum_{i=1}^{k} y_1[i] \prod_{j=1}^{k} \left(i = j, \frac{x - x_1[j]}{x_1[i] - x_1[j]} \right)$$

Newton polynomial approximation formula I use in Desmos:

$$N(x) = \sum_{i=1}^{k} \left(\sum_{m=1}^{i} y_1[m] \prod_{n=1}^{i} \left(m = n, x_1[m] - x_1[n] \right)^{-1} \right) \prod_{j=1}^{i-1} (x - x_1[j])$$

Where k is the number of data points.

Both formula are obtained online from an existing Desmos project demonstrating these two methods (Lagrange & Newton Polynomial | Desmos), and are adapted accordingly to my project.

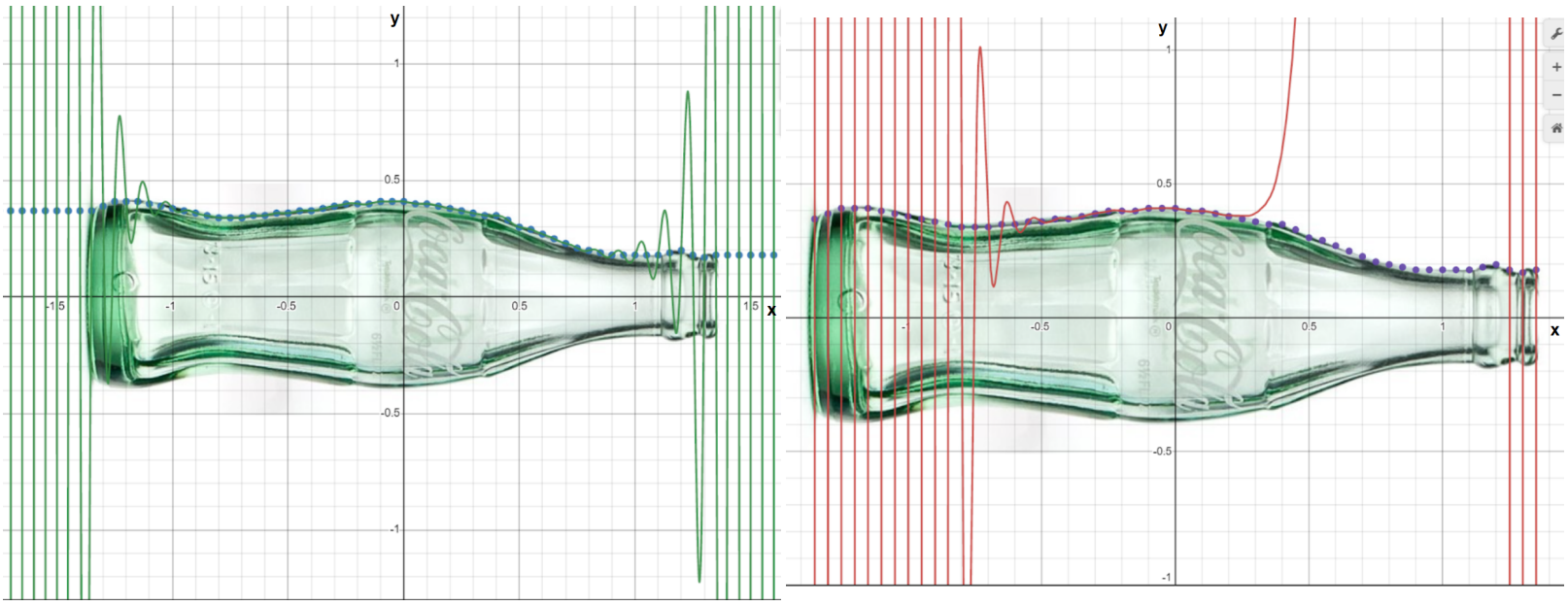

Figure 10: Lagrange and Newton polynomial approximations of the bottle, respectively

The calculation and sketching speed for both the Lagrange and Newton polynomials seem to be around the same as my method, around 30-45 seconds, which is still quite slow. Both are true interpolation methods, so the function can pass through all data points, which my method isn't able to accomplish. This is due to the different natures of my approach versus their approaches. I think the easiest explanation for this is using an analogy of constructing the function as driving a remote controlled car. Both the Lagrange and Newton methods are aiming to hit directly at the points, while my method is like sitting at the center point and steering the car left or right based on the signals (the higher derivatives) from the points. We see that both can approximate the smooth middle part of the bottle pretty well. However, the Lagrange polynomial can fit along more of the curve, while for some reason Newton's method fails to fit a large part of the bottle neck. Also, we see at both ends, both functions exhibit an oscillating behavior and are unable to approximate the curve accurately anymore. This is Runge's phenomenon, and it appears when the number of data points for the interpolation gets too large (Moler). This can be resolved by using spline interpolation with a piecewise function containing smaller segments of Newton or Lagrange polynomials, but this defeats my original purpose of finding a single continuous modelling function. Whereas my method here can approximate the shape quite well, not suffering from this phenomenon, is a continuous function, and doesn't need to use a piecewise function to model the

whole curve. Therefore overall, although there are both strengths and weaknesses to the different methods of approximation, I think my method in this case is better and more suitable than the Lagrange and Newton polynomial interpolations.

3 Conclusion

In conclusion, I think I have achieved my original purpose of successfully modelling the Coke bottle curve. From the equation modelling the curve, it opens up more opportunities to do further exploration with this topic. For example, one can find its surface area, or volume of the bottle, or the amount of material, or optimize the curve shape, etc. In the process, I have developed my own method to approximate a set of data points with a polynomial. I believe this method will prove to be a useful approximation and interpolation method for many other areas, both inside and outside of mathematics. For example, this can be used in art and graphic design software programs to turn raster data points (pixelated data), for example hand-drawn curves and scribbles into vector graphics. Or in signal processing, a signal can be sampled with a fixed sampling frequency and this method can be used to interpolate those data points and reconstruct the original signal. The method can serve as a stepping stone for further developments and improvements as well, and there are quite a lot of areas it can be optimized on. The process of this exploration has honed my skills and helped me learn a lot in different areas of math like data processing and analysis, calculus, polynomial interpolation methods, algorithms and computational process, as well as skills like math typesetting with LaTeX, etc. I am quite happy with the process of coming up with the method, as it demonstrates my mathematical intuition in observing and discovering the formula, as well as my mathematical abilities when writing the proof. The process actually took almost two years on and off, as I discovered the formula first but wasn't able to prove it until two years later only after developing necessary skills in working with sigma notations, binomial coefficients, and inductive reasoning - a key part of the proof. Therefore, overall I think this math exploration has been a really helpful and fruitful learning experience for me, and I am proud of it.

4 Bibliography

1. Glancey, Jonathan. "The Real Thing: The Coke Bottle at 100." BBC Culture, 24 Feb. 2022, www.bbc.com/culture/article/20150514-the-real-thing-the-coke-bottle.

2. Bertie, Alexander. "Message in a Bottle - What the Ever-evolving Design of the Iconic Coca-Cola Bottle Tells Us About the Times in Which We Live." AmEx Essentials, www.amexessentials.com/message-in-a-bottle.

3. Wolfram Research, Inc. Maclaurin Series - From Wolfram MathWorld. www.mathworld.wolfram.com/MaclaurinSeries.html.

4. Thomas, George Brinton, et al. Thomas' Calculus. 2005.

5. Maclaurin Series | Brilliant Math and Science Wiki. www.brilliant.org/wiki/maclaurin-series.

6. Lagrange & Newton Polynomial | Desmos. https://www.desmos.com/calculator/j7r2kec9bu

7. Moler, Cleve. "Explore Runge's Polynomial Interpolation Phenomenon." Cleve's Corner: Cleve Moler on Mathematics and Computing, 10 Dec. 2018, www.blogs.mathworks.com/cleve/2018/12/10/explore-runges-polynomial-interpolation-phenomenon.

All sources are accessed on 4 Mar. 2023 unless stated otherwise.

5 Appendix

All data points used to model the shape of the bottle:

x	y
-1.350	0.370
-1.300	0.390
-1.250	0.410
-1.200	0.410
-1.150	0.410
-1.100	0.400
-1.050	0.390
-1.000	0.380
-0.950	0.370
-0.900	0.360
-0.850	0.350
-0.800	0.340
-0.750	0.340
-0.700	0.340
-0.650	0.350
-0.600	0.350
-0.550	0.360
-0.500	0.360
-0.450	0.370
-0.400	0.370
-0.350	0.380
-0.300	0.390
-0.250	0.400
-0.200	0.400
-0.150	0.400
-0.100	0.410
-0.050	0.410
0.000	0.410
0.050	0.400
0.100	0.400
0.150	0.390
0.200	0.380
0.250	0.370
0.300	0.360
0.350	0.350
0.400	0.350
0.450	0.330
0.500	0.300
0.550	0.290
0.600	0.270
0.650	0.250
0.700	0.230
0.750	0.210
0.800	0.200
0.850	0.190
0.900	0.180
0.950	0.180
1.000	0.180
1.050	0.180
1.100	0.180
1.150	0.190
1.200	0.200
1.250	0.180
1.300	0.170
1.350	0.180