Approximating The Revolved 3D Area of Objects Without the Use of Integration

MATHS APPLICATIONS & INTERPRETATIONS HL INTERNAL ASSESSMENT

Candidate Code: hnh574

Word Count: 2446

Page Count: 16

Table of Contents

Introduction

Figure 1, Winfer

I recently got into to 3D modelling during the Covid-19 lockdown, were I taught myself how to use the different CAD programs and 3D modelling suites. During my time 3D modelling, I always found it interesting how 3D modelling software creates curves. As computers work with binary, they cannot create curves. One way to get around this problem, they use a large number of small straight lines which rotate their angle to match the curve that's trying to be. I believe a similar concept can be represented for integrals, Without using any integration, I will create an algorithm which takes a list of imputed points and finds the revolved 3D area of the shape they form. The set of points can come from any real-world word object.

Data set

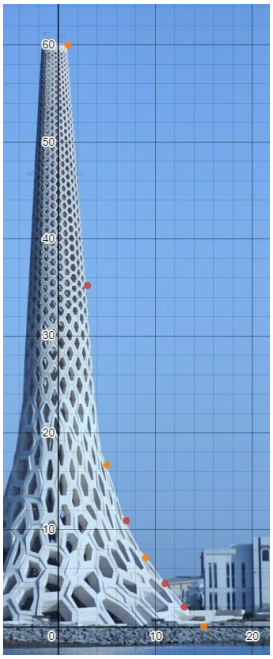


Figure 2, Winfer

The data set can come from any object with a continuous curved edge, which rotates round the object entirely. For this project, the object used will be a structure at my local university, the KAUST Beacon. The Beacon has a continuous curved outer edge. The Beacon is approximately 60 metres high; using a program called Desmos, we can approximately scale the Beacon to the correct size, and placing points along its edge which will later be used in calculations. This process can be repeated for any object or building of the correct nature, for example the Gherkin skyscraper in London or household objects like a wine glass. For the purposes of this investigation, X will be the horizontal axis and Y will be the vertical. For this data set, each of unit represents 1 metre, but this can be adjusted depending on the scale of object that is being used.

Quadratic Curve Approximation

The quadratic curve needs to be found through the points that are placed on the side of the Beacon in Desmos:

n	1	2	3	4	5	6	7	8
x	1	3	5	7	9	11	13	15
y	60	35.20	16.70	10.93	7.12	4.45	2.05	0

As the script is being written without the use of any external software libraries, the least square method is used to approximate the equation. The least-squares method makes a matrix of the sums, which can be used to calculate the quadratic formula through substitution. First, a matrix that follows these equations is created:

$$\begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \begin{Bmatrix} a \\ b \\ c \end{Bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_i^2 y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i \end{bmatrix}$$

Count	x	y	x^2	x^3	x^4	xy	x^2y
1	1	60	1	1	1	60	60
2	3	35.2	9	27	81	105.6	316.8
3	5	16.7	25	125	625	83.5	417.5
4	7	10.93	49	343	2401	76.51	535.57
5	9	7.12	81	729	6561	64.08	576.72
6	11	4.45	121	1331	14641	48.95	538.45
7	13	2.05	169	2197	28561	26.65	346.45
8	15	0	225	3375	50625	0	0
$n = 8$	$\sum x = 64$	$\sum y = 136.45$	$\sum x^2 = 680$	$\sum x^3 = 8128$	$\sum x^4 = 103496$	$\sum xy = 465.29$	$\sum x^2y = 2791.49$

$$\begin{bmatrix} 680 & 8128 & 103496 \\ 64 & 680 & 8128 \\ 8 & 64 & 680 \end{bmatrix} \begin{Bmatrix} a \\ b \\ c \end{Bmatrix} = \begin{bmatrix} 2791.04 \\ 465.29 \\ 136.45 \end{bmatrix}$$

To imitate the matrix, the script cycles through the sum of applying the powers 0 to 4 to the x values. Instead of arranging them into a matrix, the script adds them to a list. The script then does a similar thing to the Y values to make the answer matrix:

```
num = 0

for expon in range(5):
    num = 0
    for i in x:
        num = num + math.pow(i, expon)
    xMatrix.append(round(num, 10))

num = 0
num1 = 0
count = 0

for i in x:
    num = num + math.pow(i, 2) * y[count]
    num1 = num1 + i * y[count]
    count = count + 1

aMatrix.extend((round(num, 10), round(num1, 10), round(sum(y), 10)))
```

Figure 3, Winfer

For each number in Row 2 of the matrix, apply this equation:

Equation Key:

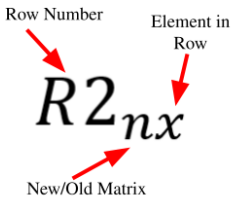


Figure 4

Equation:

$$R2_{nx} = R2_{ox} - \left(\frac{R2_{o1}}{R2_{o2}}\right) R1_{ox}$$

Example:

$$-84.98824 = 680 - \left(\frac{64}{680}\right) \times 8128$$

The equation converts the original values into new values that can be used to find the quadratic equation through substitution. Like done in the manual method, the script applies the second-row equation to all the values that would be in the second row. As one of the values is guaranteed to be 0, the script starts with the second values to save time and avoid unnecessary calculations. These new values are added to the end of the xMatrix list:

```
num = 0
for i in range(2):
    xMatrix.append(round(xMatrix[i + 2] - (xMatrix[1] / xMatrix[2]) * xMatrix[i + 3], 10))

aMatrix[1] = round(aMatrix[1] - (xMatrix[1] / xMatrix[2]) * aMatrix[0], 10)
```

Figure 5, Winfer

For each number in Row 3 we apply the following equation:

$$R3_{nx} = \left(R3_{ox} - \left(\frac{R3_{o1}}{R3_{o3}}\right) R1_{ox}\right) - \frac{R3_{o2} - \left(\frac{R3_{o1}}{R3_{o3}}\right) R1_{o2}}{R2_{n2}} R2_{nx}$$

Example:

$$31.62353 = 8 - \left(\frac{8}{680}\right) \times 680 - \frac{64 - \frac{8}{680} \times 8128}{-84.98824} \times -84.98824$$

Using the previous two equations results in this matrix:

$$\begin{bmatrix} 680 & 8128 & 103496 \\ 0 & -84.98824 & -1612.8 \\ 0 & 0 & 28.2372 \end{bmatrix} \begin{Bmatrix} c \\ b \\ a \end{Bmatrix} = \begin{bmatrix} 2791.04 \\ 202.5615 \\ 62.51163 \end{bmatrix}$$

Finally, the script finds the values of the quadratic formula using substitution. As the values for the calculation are only used once, the third-row equations are implemented into the substitution calculation:

$$R3_{n1}a + R3_{n2}b + R3_{n2}c = R3_a$$

$$R2_{n1}a + R2_{n2}b + R2_{n2}c = R2_a$$

$$R1_{n1}a + R1_{n2}b + R1_{n2}c = R1_a$$

Starting with the Row 3:

$$0a + 0b + 28.2372c = 62.51163$$

$$28.2372c = 62.51163$$

$$a = \frac{28.2372}{62.51163} = 0.45171$$

A similar process is repeated for Row 2 and Row 1:

$$0c + -84.98824b + -1612.8a = 515.78$$

$$-84.98824b + -1612.8 \times 0.45171 = 515.78$$

$$b = -10.95542$$

Row 1:

$$680c + 8128b + -1612.8a = 515.78$$

$$680c + 8128 \times -10.95542 + -1612.8 \times 0.45171 = 515.78$$

$$c = 66.30412$$

Code:

```
a = (round((aMatrix[2] - (xMatrix[0]/xMatrix[2]) * aMatrix[0]) - ((xMatrix[1] - (xMatrix[0] / xMatrix[2]) * xMatrix[3]) / xMatrix[5]) * aMatrix[1], 10))
b = (aMatrix[1] - a * xMatrix[6]) / xMatrix[5]
c = (aMatrix[0] - (xMatrix[4] * a) - (xMatrix[3] * b)) / xMatrix[2]
```

Figure 6, Winfer

The resulting values of a, b and c are the quadratic line of best fit for the inputted points. In this case, the quadratic equation found is:

$$0.452x^2 - 10.96x + 66.30$$

Finding The Area Under the Curve

The area of the curve is found using the trapezoid rule. The trapezoid rule separates the curves into small slices which resemble trapezoids. The area of the total area of the trapezoids can be calculated using the sum of the interior edges multiplied by two, plus the two exterior edges multiplied by the half the width of the edges. The trapezoid rule uses this equation:

$$\frac{1}{2}h(y_0 + y_n + (\sum_{i=1}^{n-1} 2 \times y_i)$$

Unlike finding the integral, the trapezoid rule is only an approximation of the area under the curve. When the slit width is decreased, the trapezoid rule matches more closely with the true value, although it takes more time to calculate as there are more interior edges. There is a limit to how small of a slit width is useful, as beyond a certain point the area under the curve matches closely enough for many practical uses.

In the code for the trapezoid rule, the user has the ability to change the slit distance. First the distance is calculated by subtracting the upper bound from the lower bound, which is divided by the inputted slit width. This value is calculated in the for loop which imitates the sigma notation in the equation. If the number of slits does not perfectly fit between the upper and lower bounds, the last slit is rounded up so it is a integer. Loop increments between all of the slit areas, and adds them to the

total sum. The final area is calculated by completing the rest of the equation that is outside the sigma notation.

```
dist = (uBound - lBound) / sLength

try:
    dist = int(dist)
    extra = 0

except:
    extra = round(dist % 1, 10)
    math.ceil(dist)
    dist = int(dist)


dist1 = dist + sLength
num = 0
xn = lBound + sLength
for i in range(dist - 2):
    num = num + a*math.pow(xn, 2) + b * xn + c
    xn = xn + sLength
highestValue = max(a*math.pow(lBound, 2) + b * lBound + c, a*math.pow(uBound, 2) + b * uBound + c)
area2D = 0.5 * (sLength*((a*math.pow(lBound, 2) + b * lBound + c) + (a*math.pow(uBound, 2) + b * uBound + c) + (2 * num)))
```

Figure 7, Winfer

Finding the 3D Revolved Area

For the 3D area of the shape, I came with up with a method that's reminiscent of the trapezoid rule; they differ in that the trapezoid method creates vertical divisions, this method creates horizontal divides due to the geometry of the solid being measured . This creates a series of horizontal ‘disks’ that match the area under the quadratic curve, and represent the solid being measured

To start, the coordinates for the edges of the disks must be found. As the disks are horizontal the inverse of the quadratic needs to be used.

The equation differs slightly depending on whether the quadratic is concave or convex.
If the quadratic equation is convex this formula is used:

$$x = \pm\sqrt{\frac{y - (c - \frac{|b|^2}{4a})}{a}} + \frac{|b|}{2a}$$

If the quadratic equation is concave :

$$x = \pm\sqrt{\frac{y - (c - \frac{b^2}{4a})}{a}} - \frac{b}{2a}$$

Using the quadratic formula, a table of x and y values can be made, with the y values incrementing by a set amount:

Step Increment: 10

Lower Bound: x = 0

Upper Bound: x = 12

Y Value	0	10	20	30	40	50	60
Negative X Value	11.61	7.39	5.45	3.96	2.70	1.59	0.59
Positive X Value	n/a	n/a	n/a	n/a	n/a	n/a	n/a

In the case of the Beacon, there are no values that fit into the negative X value, as they are excluded by the upper bound.

Using these values, the script goes through the following steps to calculate the revolved area:

Using the values found from the negative inverse equation, the script creates boxes with the height being the increment between the y values. The script goes upwards from the points so the answer will always be larger than the true value found using integrals.

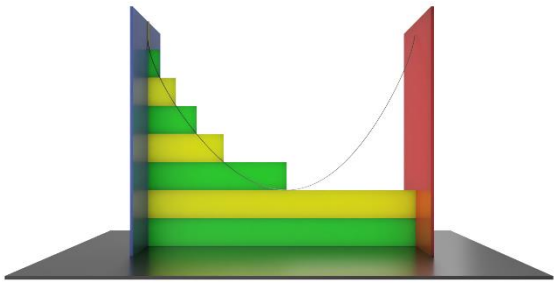


Figure 8, Winfer

If neither of the inverse equations produce a point, it is assumed that the y value is underneath the line. As it is underneath the line, the radius of the disk is assumed to be the upper bound subtracted from the lower bound. Then using the disk equation, the revolved area is found for those values.

Disk Equation:

$$A = \pi r^2 h$$

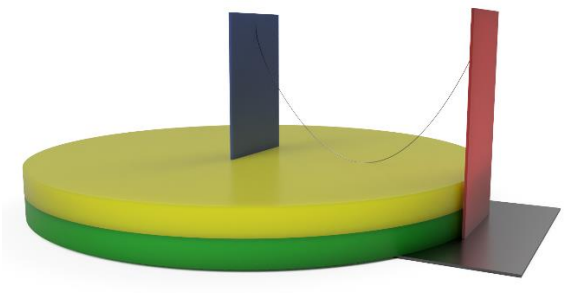


Figure 9, Winfer

This process is repeated for the points on the negative inverse equation. The point found from the inverse equation is subtracted from the lower bound, which is the radius used in the disk calculation. All of the disk calculations are summed to get the approximate area.

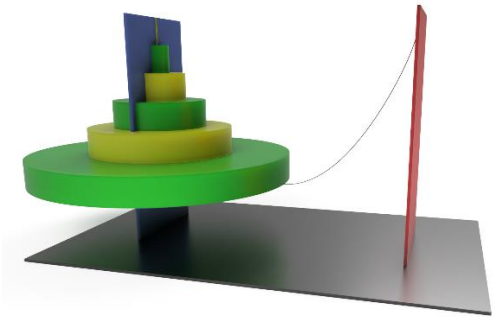


Figure 10, Winfer

If the upper bound allows for the positive inverse equation, a different operation needs to be conducted. Firstly, similarly to the values that came from the positive equation, the values from the negative equation are subtracted from the lower bound to find the radius. This radius is used in the disk equation to get the area.

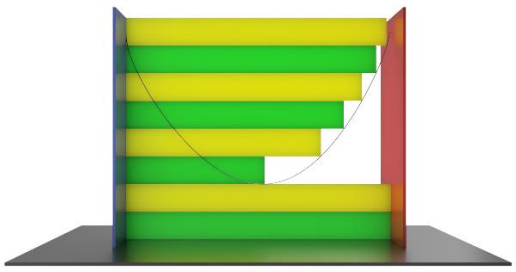
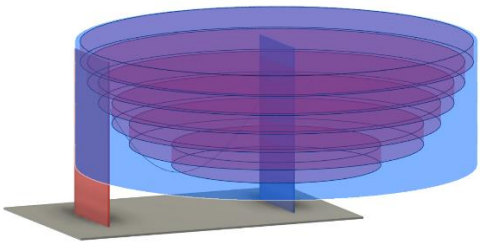
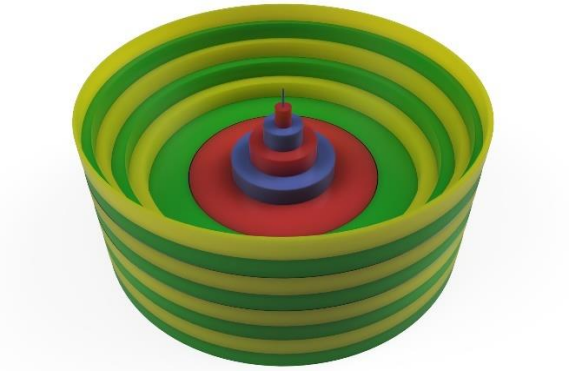


Figure 11, Winfer

Using this area, the value is subtracted from the largest possible disk, which is when the radius is the upper bound subtracted from the lower bound.	 *Figure 12, Winfer*
Finally, the two sides of the quadratic are added together to find the total area of the shape.	 *Figure 13, Winfer*

```
totalDis = math.pi * math.pow(ubound - lbound, 2) * stepHeight
count = math.floor((ubound - lbound) / stepCount)
count1 = 0
yAmount = 0
for i in range(count):
    test = a * math.pow((count1), 2)) + (b * ubound) + c
    yAmount = max(yAmount, test)
    count1 = count1 + stepHeight
yAmount = math.floor(yAmount / stepCount)
```

Figure 14, Winfer

First, the script finds out how many disks will make up the shape. This part of the script finds the highest values present between the upper and lower bound and divides it by the step increment. This is used to give the amount y of values the equation has to cycle through to find the highest value. The program loops through all the possible y values within the limits of the upper and lower bound to find which one has the highest value. The script then divides the highest value by the step increment

to get the total number of y values.

```
if a > 0:
    for i in range (yAmount):
        yValue = stepCount
        xValue = 0
        try:
            xValue = ((-1) * (math.sqrt((yValue - (c - ((math.pow(abs(b), 2)) / (4 * a)))) / a))) + ((abs(b)) / (2 * a))
            if xValue <= ubound and xValue >= lbound:
                negativeCyl.append(math.pi * math.pow(round(xValue, 10) - lbound, 2) * stepHeight)
                xValues3D.append(round(xValue, 10))
                yValues3D.append(yValue)
        except:
            pass

        try:
            xValue = math.sqrt((yValue - (c - ((math.pow(abs(b), 2)) / (4 * a)))) / a) + ((abs(b)) / (2 * a))
            if xValue <= ubound and xValue >= lbound:
                postiveCyl.append(totalDis - (math.pi * math.pow(round(xValue, 10) - lbound, 2) * stepHeight))
                xValues3D.append(round(xValue, 10))
                yValues3D.append(yValue)
        except:
            pass
        stepCount = round(stepCount + stepHeight, 10)
    try:
        postiveCyl.append(((min(yValues3D) / stepHeight)) * totalDis)
    except:
        pass
    total = sum(negativeCyl) + sum(postiveCyl)
```

Figure 15, Winfer

Then, the script identifies whether the quadratic equation is convex or concave by finding if the ‘a’ value is greater or less than 0. If the value is larger than 0, the program starts to find all the x values that correspond to the y values at a set interval. The program first tries to see if the value can fit inside the negative inverse equation. If the equation world the value is checked to see if it is smaller than the upper bound but larger than the lower bound. If the value meets these criteria, the radius gets used in the disk equation. The script then follows the steps mentioned before to find the final answer.

A similar process is repeated for the concave formula, but the distance to the lower bound is found for both x values that share the same y values and are subtracted against each other? to get the disk area.

Evaluation

With the step increment of 1, a lower bound of 0, and an upper bound of 15 the approximate area calculated using the disk method is $850m^2$. Using the integral of:

$$\int_0^{15} \left(\pi(0.452x^2 - 11x + 66.3)\right)dx$$

The exact area of the Beacon is $834.09m^2$. Using the large step increment, the answer only had an error margin of 1.9. Using a step increment of 0.00001, the disk method was exactly the same as the integral result to 6 significant figures, although the code took significantly longer to execute due to the number of calculations.

There are several improvements that could have been made to the code and algorithm. Firstly, the code could have more closely approximated the line by drawing a direct line between the points, instead of a box. This improvement becomes more marginal with smaller step increments, but it allows for a more accurate answer to be found significantly faster than with the current method.

Another potential area for improvement was with code optimisation; One of the most notable points was finding the highest y value between the lower and upper bounds. If the quadratic was convex, the highest y value is guaranteed to be on either the lower bound or upper bound. If the quadratic was concave, then there should have been an exit condition on the loop if the y values started to decrease, as the quadratic curve only has one peak, and all the values after the peak has been reached are guaranteed to be lower. This new method would save a significant amount of processing time, especially with problems that have a small step increment, although it would take up more memory space.

Another optimisation could be with checking if a value has already been calculated. An example of this can be seen with the highest y value script and the trapezoid rule script. Both scripts used the same $\frac{ubound - lbound}{stepCount}$ equation in their calculations. As this value had already been found, the variable could have been reused. The fourth improvement I could have made was by making my code

more readable. At multiple points there where variables where their name does not coincide with its value. To avoid this, more “unnecessary” variables could have been created to improve my codes readability.

Conclusion

Overall, the disk method worked to approximately match the integral. The disk method can be used to approximate the integral to any significant figure with a small enough increment height. Despite this, the disk method was significantly more CPU intensive than the integral. Overall, script was successful in its task, but as mentioned in the evaluation there were various improvements that could have been made.

Works Cited

Estela, Mike. “Inverse of Quadratic Function.” *ChiliMath*, 12 Feb. 2021, https://www.chilimath.com/lessons/advanced-algebra/inverse-of-quadratic-function/.

Jadhav, Keshav. “Problems on Linear and Quadratic Curve Fitting Using Least Square Method.” *Youtube*, 16 Apr. 2019, https://www.youtube.com/watch?v=uEGNR7Zh6Zc&ab_channel=KeshavJadhav.

Winfer, John. “Beacon Example Model”, 1 Mar. 2022

---. “Desmos Pic”, 1 Mar. 2022

---. “Code Pic 1”, 1 Mar. 2022

---. “Equation Key”, 1 Mar. 2022

---. “Code Pic 2”, 1 Mar. 2022

---. “Code Pic 3”, 1 Mar. 2022

---. “Code Pic 4”, 1 Mar. 2022

---. “Fusion 360 Example Visualisation Pic 1”, 21 Mar. 2022

---. “Fusion 360 Example Visualisation Pic 2”, 21 Mar. 2022

---. “Fusion 360 Example Visualisation Pic 3”, 21 Mar. 2022

---. “Fusion 360 Example Visualisation Pic 4”, 21 Mar. 2022

---. “Fusion 360 Example Visualisation Pic 5”, 21 Mar. 2022

---. “Fusion 360 Example Visualisation Pic 6”, 21 Mar. 2022

---. “Code Pic 5”, 1 Mar. 2022

---. “Code Pic 6”, 1 Mar. 2022