Modelling the volume and surface area of the cabin of an Airbus A350-900

Personal Code: [redacted]

1 Introduction

Aviation has always been my strongest passion since I was young. I would occasionally visit the airport for planespotting, sit at the edge of the runway and admire every aircraft that whizzed past my head. I am always fascinated and impressed by the sheer amount of engineering required to iteratively refine and improve upon aircraft designs, which has evolved planes in the modern era to be significantly safer to fly, much more fuel efficient and rapid. My dream as an aspiring aerospace engineer is to contribute to the aviation community, to make our transport systems more efficient by developing better aircraft designs.

Historically, the fuselage of early aircraft such as the McDonnell Douglas DC-3 (developed in the 1940s), had square-shaped cabins to maximise comfort. However, as the world experienced a massive surge in the need to carry more passengers and to fly faster at an economically viable way, engineers have realised that square cabins were highly inefficient. As a result, modern aircraft designs such as the Airbus A350-900 (developed in the 2010s) have significantly different aircraft characteristics, such as circular cabins, curved wing surfaces and blended winglets. As a keen aviation enthusiast, I keep a collection of different aircraft models:

Figure 1. The newest addition to my collection of aircraft models, the Airbus A350-900.

Since I often find myself glancing at the models and admiring its complex geometry, I wondered: how do aircraft manufacturers trade between the aerodynamic efficiency and the size of the aircraft? For example, if the width of the fuselage is increased, the aircraft will be able to carry more passengers and generate greater profit. But if the width is excessively large, this will cause an increase in aerodynamic drag and cause greater fuel expenditures. Is there a feasible method to compute the equilibrium at which the friction is minimised?

I therefore wanted to know whether I could use integration which I learnt during my lessons, to accurately calculate the volume and surface area of an aircraft, and most importantly optimise the dimensions of the aircraft while minimising the aerodynamic drag of the aircraft. In the following section, I will discuss the various concepts required to perform the investigation.

2 Aim and Approach

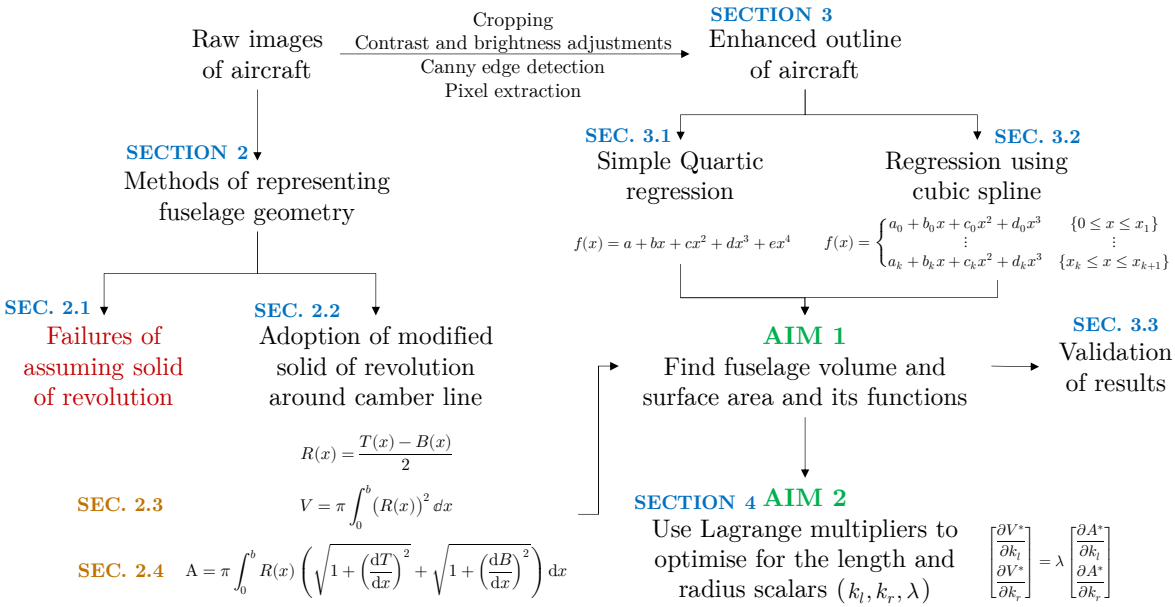

Figure 2. A flowchart of the general process of my investigation and the final aims.

2.1 Solid of revolution

The object I will be investigating is the *fuselage*, which is the main body of the aircraft.

Figure 3. The 2D orthographic projections of the A350-900 fuselage. The x, y and z axes are aligned along the length, width, and height of the fuselage respectively.

Upon a first inspection of the geometry of the fuselage, the top view indicates that there is a symmetry along the x-axis, and that the frontal area forms a circle. In order to represent the solid mathematically, I initially believed that I could represent the fuselage by a solid of revolution, which is an object formed by an infinite number of circular disks of radius $R(x)$, of a small thickness of Δx, as represented by below:

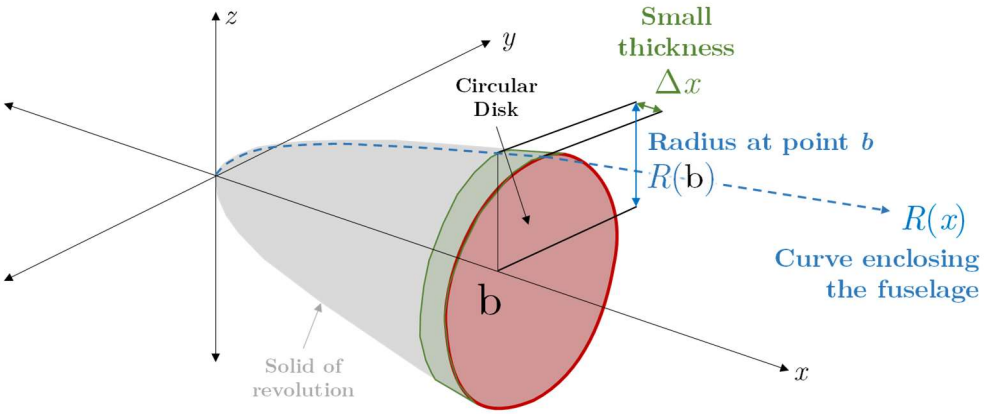

Figure 4. Representation of the fuselage as a solid of revolution, enclosed by the curve $R(x)$ and rotated along the x-axis.[1]

However, I began to realise that it is problematic to assume the fuselage as a solid of revolution, because there is no rotational symmetry along the z-axis, as depicted below:

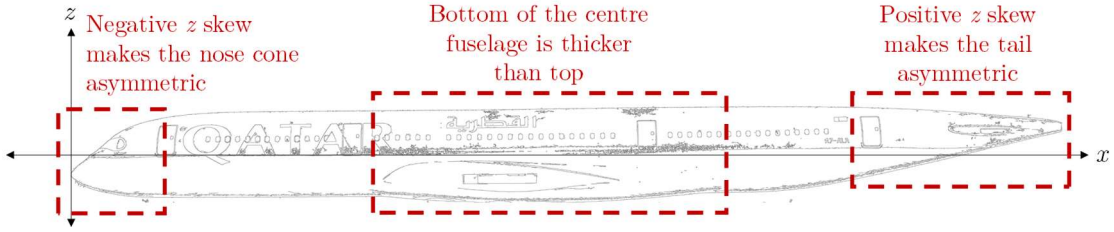

Figure 5. The issues of assuming the fuselage as a solid of revolution along the x-axis.

As shown above, because the fuselage is asymmetric, this means that certain parts of the fuselage will be under- and over-represented, introducing large errors from the true result. To solve this issue, I will illustrate in the following section how I developed my own new method.

2.2 Adoption of new method for representing the fuselage

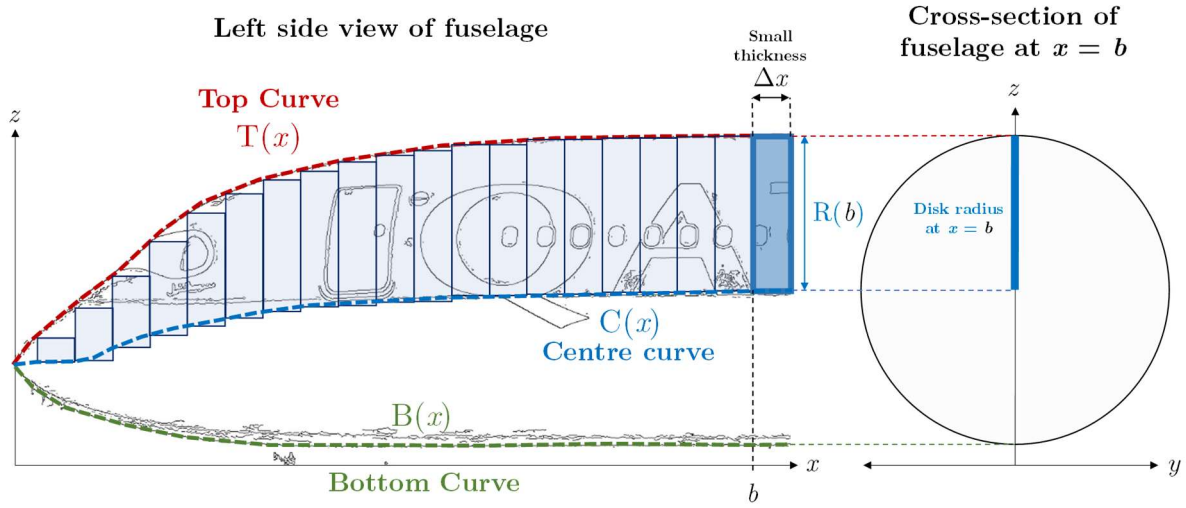

Figure 6. Depiction of the proposed mathematical representation of the fuselage.

[1] Unless specified otherwise, all diagrams are created by the candidate with the use of Python scripts, available in the Appendix.

In this new representation, I will be assuming that the fuselage is composed of an infinite number of circular disks of radius $R(x)$, with small thickness Δx, centred along the curve $C(x)$, where $C(x)$ is the midpoint between the top curve $T(x)$ and bottom curve $B(x)$:

$$C(x) = \frac{T(x) + B(x)}{2} \tag{1}$$

The radius of each disk is expressed by the distance between the centre and the top curve:

$$R(x) = T(x) - C(x) = \frac{T(x) - B(x)}{2} \tag{2}$$

By representing the fuselage as discs centred along the camber line $C(x)$ of the fuselage instead of the x-axis, the issue of asymmetricity as demonstrated in Section 2.1 is eliminated, providing a much more accurate representation than the solid of representation. In the following section, I will be deriving the formulae for the volume and the surface area of the fuselage.

2.3 Volume of the fuselage

The volume of the fuselage can be calculated by the sum of the area of all circular disks (πR^2) multiplied by the width Δx between $x = 0$ and $x = b$:

$$V = \lim_{\Delta x \to 0} \sum_i A(x_i)\Delta x = \lim_{\Delta x \to 0} \sum_i \pi\big(R(x_i)\big)^2 \Delta x = \pi \int_0^b \big(R(x)\big)^2 \, \mathrm{d}x \tag{3}$$

2.4 Surface area of the fuselage

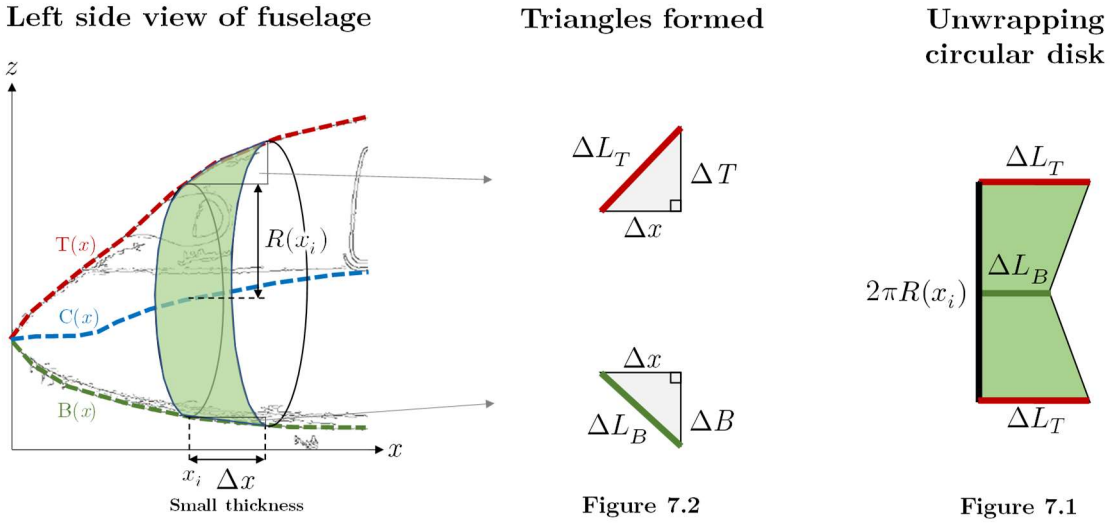

Figure 7. The calculation of the surface area of the fuselage.

With reference to Figure 7.1, the surface area of the entire fuselage can be calculated by the sum of the surface areas of all circular disks:

$$A = \lim_{\Delta x \to 0} \sum_i A(x_i)$$

$$A = \lim_{\Delta x \to 0} \sum_i 2\pi R(x_i)\Delta L_T - \frac{2\pi R(x_i)(\Delta L_T - \Delta L_B)}{2}$$

$$A = \lim_{\Delta x \to 0} \sum_i \pi R(x_i)(\Delta L_T + \Delta L_B) \tag{4.1}$$

By applying the Pythagorean theorem, $(\Delta L_T)^2 = (\Delta x_i)^2 + (\Delta T)^2$ and $(\Delta L_B)^2 = (\Delta x_i)^2 + (\Delta B)^2$ as shown in Figure 7.2, Equation (4.1) can be rewritten to:

$$A = \lim_{\Delta x \to 0} \sum_i \pi R(x)(\sqrt{(\Delta x_i)^2 + (\Delta T)^2} + \sqrt{(\Delta x_i)^2 + (\Delta B)^2})$$

$$A = \lim_{\Delta x \to 0} \sum_i \pi R(x) \left(\Delta x \sqrt{1 + \left(\frac{\Delta T}{\Delta x}\right)^2} + \Delta x \sqrt{1 + \left(\frac{\Delta B}{\Delta x}\right)^2} \right)$$

$$A = \pi \int_0^b R(x) \left(\sqrt{1 + \left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)^2} + \sqrt{1 + \left(\frac{\mathrm{d}B}{\mathrm{d}x}\right)^2} \right) \mathrm{d}x \tag{4}$$

Now that the method of representing the fuselage is established, the following section will aim to find the volume and the surface area of the fuselage by applying Equations (3) and (4).

3 Modelling the aircraft fuselage

To calculate the volume and the surf ace area of the fuselage, the top curve, $T(x)$ and the bottom curve, $B(x)$ must first be found. To increase the correctness of the results, I will be comparing the results obtained from two separate methods by its quality, being:

1. Simple quartic regression,
2. Piecewise regression using cubic splines.

To obtain the data points required for the regression, I first processed the side view of the fuselage through a Canny Edge detector, producing a sharp outline of the image. The coordinates of each pixel are then extracted if its luminance exceeds a certain threshold. After aligning and rescaling the coordinates to match the real-life length of the aircraft of 65.27 m, 13,403 data points representing each boundary are automatically extracted from the image:

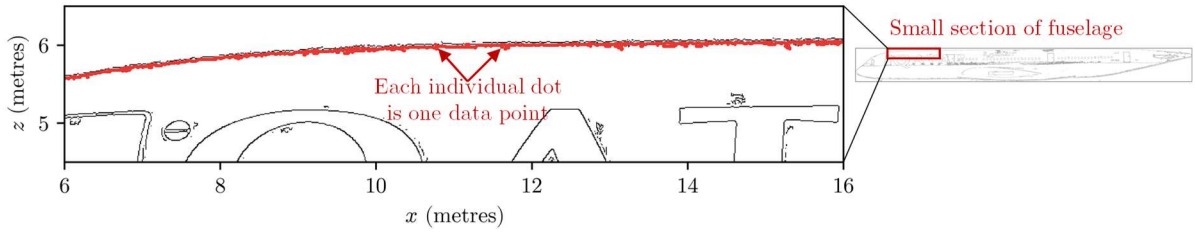

Figure 8. A zoomed-in image of the individual data points after the pixel extraction.

The advantage of automating this process is to reduce the human errors and bias, such as accidentally mismarking the location of the edges. Also, because of the large amount of data points, this allows will allow me to regress the curves with a greater degree of confidence.

3.1 Simple quartic regression

The first method I will be using to obtain the boundary curves is by fitting a quartic polynomial. The reason I chose a polynomial with order 4 is because it is an even-degree polynomial, which as $x \to \pm\infty$, $f(x)$ tends to head off in the same direction. Considering that the top and bottom boundaries of the aircraft also tend to head off in the same direction, this choice is justified. The quartic polynomial is given by:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 \tag{5}$$

, where β_0, β_1, β_2, β_3, β_4 are unknown coefficients to be found and x is the position along the x-axis. To find these coefficients, given a list of coordinates (x_n, y_n), the following matrix equation is to be solved:

$$\begin{bmatrix} x_0^0 & x_0 & x_0^{m-1} \\ x_1^0 & \ddots & x_1^{m-1} \\ x_n^0 & x_n & x_n^{m-1} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} \tag{6}$$

where β are the coefficients to be found, n is the n^{th} data point, m is the degree of the polynomial equation, while minimising the residual sum of squares (RSS):

$$RSS = \sum_{i=0}^{n} (y_i - f(x_i))^2 = \sum_{i=0}^{n} \Big(y_i - (\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4)\Big)^2 \tag{7}$$

, where y_i is the i^{th} predicted value. By finding the global minimum point of Equation (7), the optimum coefficients β can subsequently be found. However, since are 13,403 data points, the matrix in Equation (6) will be too complex to be feasibly solved by hand. Therefore, I decided to use the use the `NumPy` Python library, which automatically computes the equations above. The top and bottom curves of the fuselage is found to be[2]:

$$T(x) = 2.886 + 0.4926x - 0.02484x^2 + 0.0005090x^3 - 3.625\text{E-}6x^4 \tag{8}$$

$$B(x) = 0.9724 - 0.07021x + 0.002115x^2 - 4.355\text{E-}5x^3 + 6.466\text{E-}7x^4 \tag{9}$$

[2] Figures will be rounded to the nearest 4 significant figures to preserve precision, and expressed in the E notation whenever numbers are too small or too large.

Below is an overlay of the top and bottom curves, $T(x)$ and $B(x)$ on the original image:

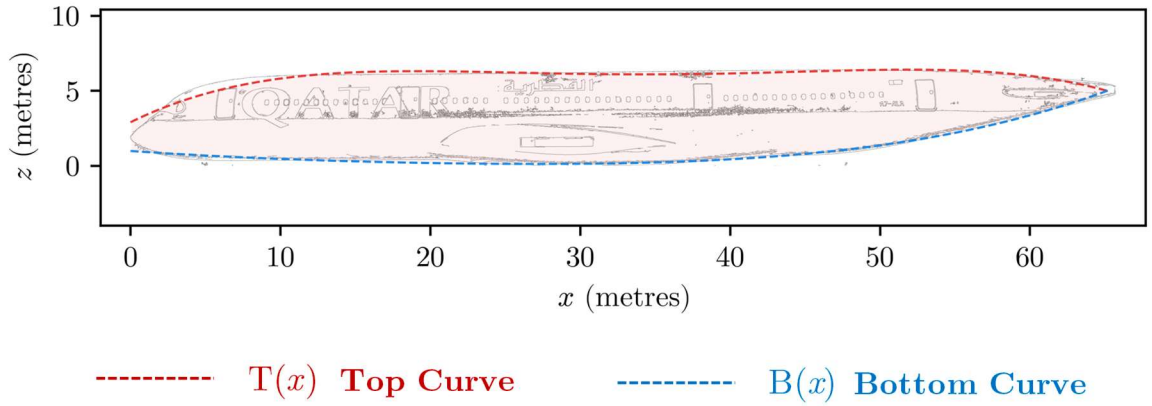

Figure 9. The top and bottom curves obtained using quartic regression.

To visualise the results in 3D, I constructed the following set of parametric equations that describes the surface:

$$\begin{matrix} x(t,\theta) = t \\ y(t,\theta) = R(t)\cos(\theta) \\ z(t,\theta) = R(t)\sin(\theta) + C(t) \end{matrix} \quad \left\{ \begin{matrix} 0 < \theta < 2\pi, \\ 0 < t < 65.27 \end{matrix} \right\} \tag{10}$$

In Equation (10), as θ is being varied from 0 to 2π, this draws a circle with radius $R(x)$ and at $C(x)$ units above the z-axis. Plotting this equation gives the following visualisation:

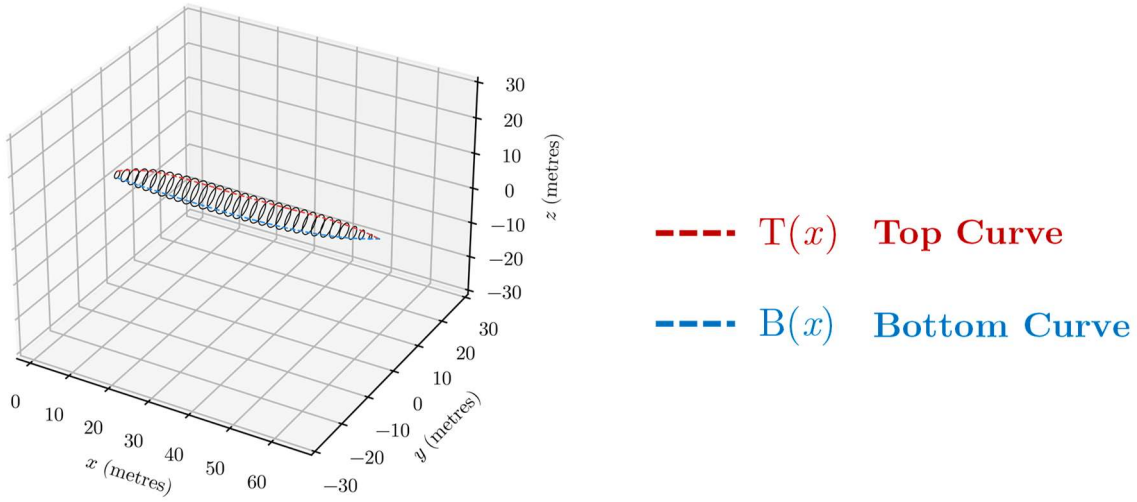

Figure 10. A 3D visualisation of the solid formed using simple quartic regression.

To calculate the volume of the solid, the radius is first calculated using Equation (2):

$$R(x) = T(x) - B(x)$$

$$R(x) = 0.9571 + 0.2814x - 0.01348x^2 + 0.0002763x^3 - 2.136\text{E-}6x^4 \tag{11}$$

The volume can be determined by substituting Equation (11) into Equation (3):

$$V = \pi \int_0^b \big(R(x)\big)^2 \,\mathrm{d}x$$

$$V = \pi \int_0^{65.27} (0.9571 + 0.2814x - 0.01348x^2 + 0.0002763x^3 - 2.136\text{E-}6x^4)^2 \,\mathrm{d}x$$

By expanding $\big(R(x)\big)^2$ and integrating each term and evaluating:

$$V = \pi \int_0^{65.27} \begin{matrix} 0.9161 + 0.5387x + 0.05338x^2 - 0.007055x^3 + 0.00033306x^4 \\ -8.649\text{E-}6x^5 + 1.339\text{E-}7x^6 - 1.180\text{E-}9x^7 + 4.562\text{E-}12x^8 \end{matrix} \,\mathrm{d}x$$

$$V = \pi \left| \begin{matrix} 0.9161x + 0.2693x^2 + 0.01779x^3 - 0.001764x^4 + 6.661\text{E-}5x^5 \\ -1.441\text{E-}6x^6 + 1.913\text{E-}8x^7 - 1.475\text{E-}10x^8 + 5.069\text{E-}13x^9 \end{matrix} \right|_0^{65.27}$$

$$V = 1341.6380637389\ldots$$

$$V = \boxed{1341.64\ \mathrm{m}^3{}_{(2d.p.)}} \tag{12}$$

The surface area can be determined by substituting Equations (8) and (9) into Equation (4):

$$A = \pi \int_0^b R(x) \left(\sqrt{1 + \left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)^2} + \sqrt{1 + \left(\frac{\mathrm{d}B}{\mathrm{d}x}\right)^2} \right) \mathrm{d}x$$

$$A = \pi \int_0^{65.27} \left(\begin{matrix} 0.9571 + 0.2814x - 0.01348x^2 + 0.0002763x^3 - 0.000002136x^4 \\ (\sqrt{1 + \left(\frac{\mathrm{d}(2.886 + 0.4926x - 0.02484x^2 + 0.0005090x^3 - 3.625\text{E-}6x^4)}{\mathrm{d}x}\right)^2} + \\ \sqrt{1 + \left(\frac{\mathrm{d}(0.9724 - 0.07021x + 0.002115x^2 - 4.355\text{E-}5x^3 + 6.466\text{E-}7x^4)}{\mathrm{d}x}\right)^2}) \end{matrix} \right) \mathrm{d}x$$

$$A = \pi \int_0^{65.27} \left(\begin{matrix} 0.9571 + 0.2814x - 0.01348x^2 + 0.0002763x^3 - 0.000002136x^4 \\ (\sqrt{1 + (0.4926 + 0.04968x + 0.001527x^2 - 1.450\text{E-}5x^3)^2} + \\ \sqrt{1 + (0.07021 + 0.004231x - 0.0001307x^2 + 2.586x^3)^2}) \end{matrix} \right) \mathrm{d}x$$

Since the integral above could not be computed analytically, A is evaluated using a graphical calculator:

$$A = 1011.6973136310\ldots$$

$$A = \boxed{1011.69\ \mathrm{m}^2{}_{(2d.p.)}} \tag{13}$$

Therefore, the volume of the fuselage is 1341.64 m^3 and the surface area is 1011.69 m^2.

While the volume and the surface area of fuselage has been successfully found above, I realised that an apparent weakness for the simple quartic regression is that it is highly susceptible to issues such as underfitting and overfitting, as demonstrated below:

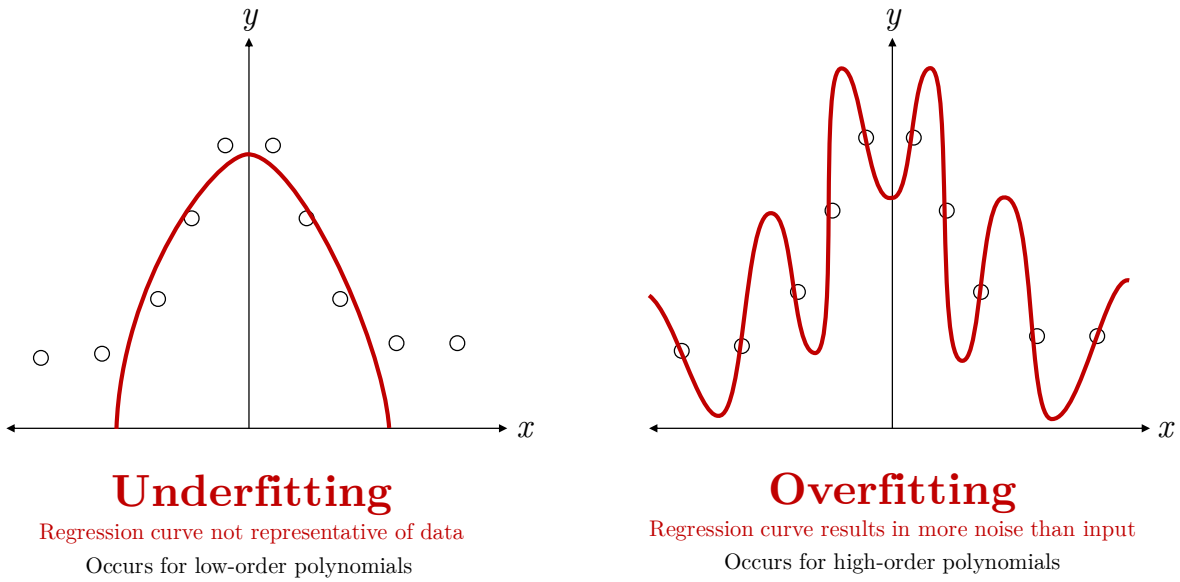

Figure 12. Issues of underfitting and overfitting occurs when an incorrect order of polynomial is used.

In this specific case, where a quartic polynomial is used for regression:

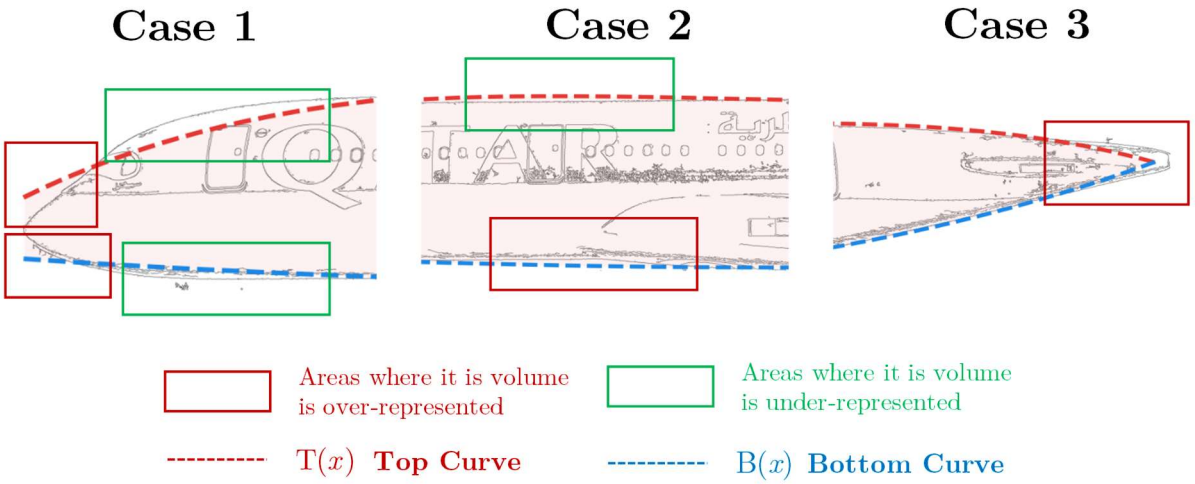

Figure 11. Demonstration the misrepresentation of the fuselage geometry at several locations.

There are multiple instances when the quartic polynomial is unable to resolve the minute details of the sharp edges, indicating that it is suffering with the issue of underfitting and overgeneralisation. While I could easily mitigate this by increasing the number of degrees, such that more detail can be captured within the polynomial, I ran into the risk of experiencing overfitting. Therefore, I wondered: is there an optimum number of degrees such that the maximum detail is captured, while not being overly noisy and thus impacting the accuracy?

Therefore, in the following section, I will attempt to explore a different method of calculating the volume and surface area of the fuselage, using the technique of piecewise regression with cubic splines, which is expected to have less issues of underfitting and overfitting.

3.2 Piecewise regression using cubic splines

Noticing that the fuselage contains an appreciable number of sharp edges, particularly the nose cone, I will be using cubic splines to perform a piecewise regression on the data. In this this new method, instead of representing the entire length of the surface with a single curve, the surface is first subdivided into equal intervals of length of L, then a simple cubic regression is performed for each interval individually and independently, forming the following function:

$$f(x) = \begin{cases} \beta_{0_0} + \beta_{1_0}x + \beta_{2_0}x^2 + \beta_{3_0}x^3 & \{0 < x < L\} \\ \vdots & \vdots \\ \beta_{0_k} + \beta_{1_k}x + \beta_{2_k}x^2 + \beta_{3_k}x^3 & \{Lk < x < L(k+1)\} \end{cases} \tag{14}$$

, where β_0, β_1, β_2, β_3 are unknown coefficients to be found, x is the position along the x-axis , k is the k^{th} interval and L is the length of each interval. According to Silverman (1985) and Wand (2000), because each regression is treated independently from each interval, there is no guarantee that the function is continuous and differentiable, and hence, it is necessary to impose several constraints on $f(x)$:

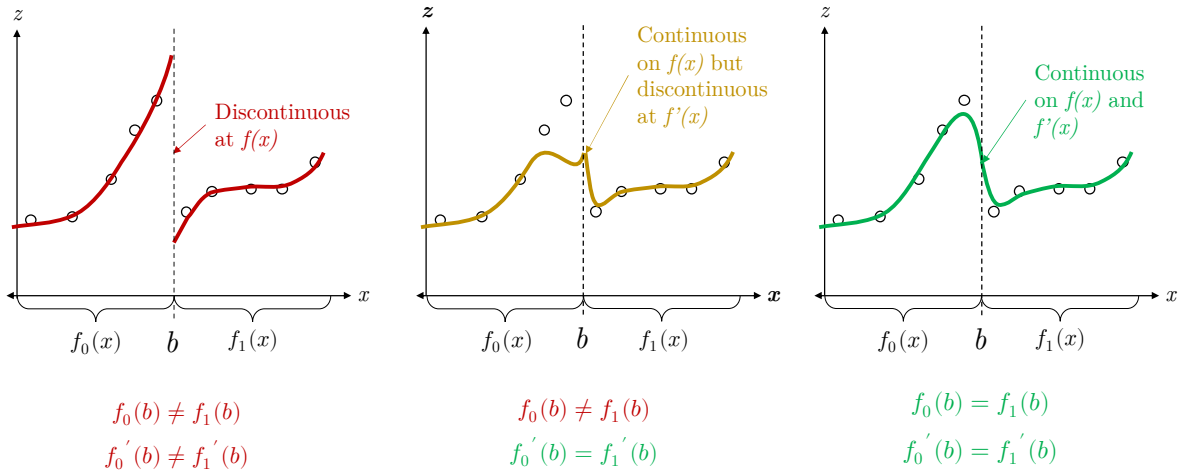

Figure 12. Demonstrations of discontinuities in $f(x)$ when $f_k(b) \neq f_{k+1}(b)$ and $f_k{}'(b) \neq f_{k+1}{}'(b)$.

Here, by constraining $f_k(b) = f_{k+1}(b)$, this eliminates the issue of $f(x)$ being discontinuous at the edge of each interval k, and by also constraining $f_k{}'(b) \neq f_{k+1}{}'(b)$, this ensures that the overall curve is smooth (Hall and Opsomer, 2005).

In combination with the constraints above, while minimising the residual sum of squares:

$$RSS = \sum_{i=0}^{n} \left(y_i - f(x_i)\right)^2 \tag{15}$$

, where y_i is the i^{th} predicted value and $f(x_i)$ is Equation (14) evaluated at the i^{th} x-value. By finding the global minimum of Equation (15), all sets of the optimum coefficients β_k for all k intervals can be found. However, as Perperoglou et al. (2019) suggests, because this approach is an iterative process that uses many complex computational algorithms to minimise Equation (15), there is currently no feasible method to compute β by hand. Therefore, I decided to use the `statsmodels` and `patsy` Python libraries, which automatically computes the coefficients:

$$T(x) = \begin{cases} 1.911 + 1.342x - 0.5741x^2 + 0.1933x^3 & \{0 < x < 1.004\} \\ \vdots & \vdots \\ 2.292\text{E+4} - 1070x + 16.66x^2 - 0.08646x^3 & \{64.25 < x < 65.26\} \end{cases} \tag{16}$$

$$B(x) = \begin{cases} 1.841 - 1.482x + 0.9862x^2 - 0.2932x^3 & \{0 < x < 1.004\} \\ \vdots & \vdots \\ -2.503\text{E+4} + 1167x - 18.12x^2 + 0.09382x^3 & \{64.25 < x < 65.26\} \end{cases} \tag{17}$$

Below is an overlay of the top and bottom curves, $T(x)$ and $B(x)$ on the original image:

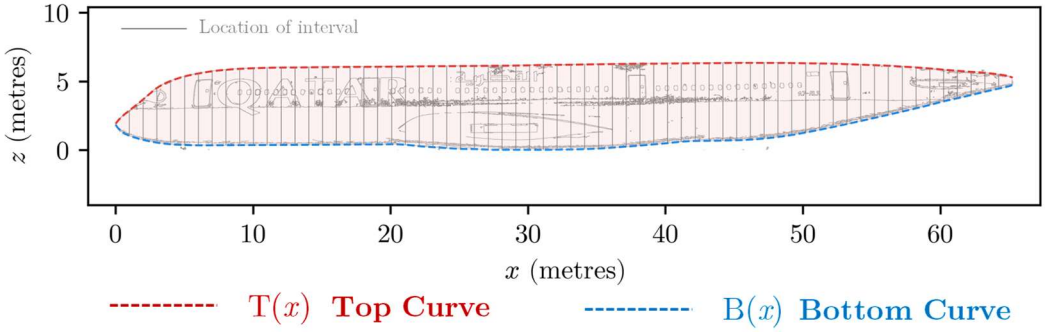

Figure 13. The top and bottom curves obtained using piecewise cubic regression and its intervals.

Applying the set of parametric equations in Equation (10), a 3D visualisation is generated:

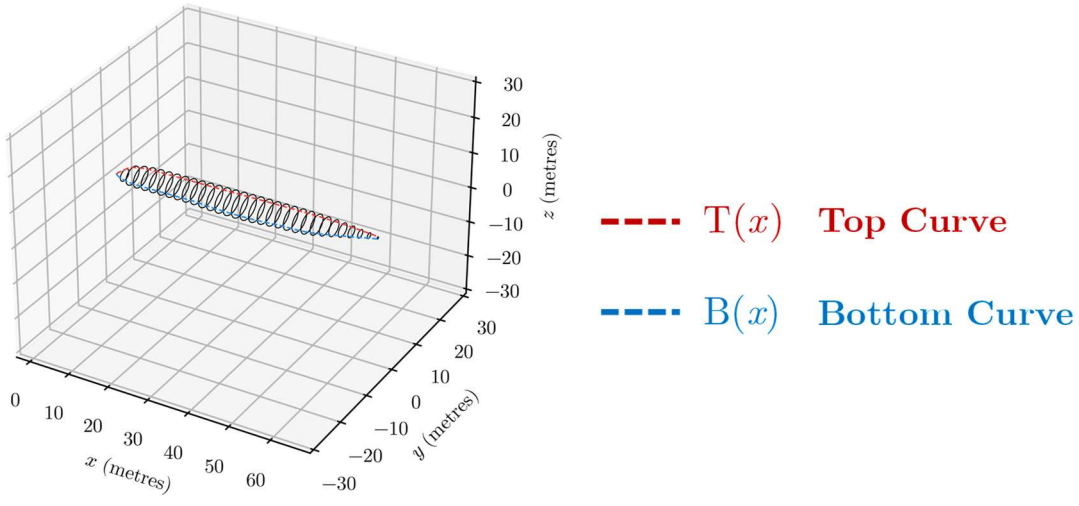

Figure 14. A 3D visualisation of the solid formed using piecewise cubic regression.

To calculate the volume and surface area of the solid, I will begin by first obtaining the radius curve, which is calculated by substituting Equations (16) and (17) into Equation (2):

$$R(x) = \frac{T(x) - B(x)}{2}$$

$$R(x) = \begin{cases} 0.03532 + 1.412x - 0.7802x^2 + 0.2432x^3 & \{0 < x < 1.004\} \\ \vdots & \vdots \\ -2.398\text{E}{+}4 + 1118x - 17.39x^2 + 0.09014x^3 & \{64.25 < x < 65.26\} \end{cases} \tag{18}$$

The volume of the solid can be then determined by from Equation (3):

$$V = \pi \int_0^b \big(R(x)\big)^2 \, \mathrm{d}x$$

Since $R(x)$ is a piecewise function, the total volume is calculated by summing up each individual volumes for each interval i:

$$V = \pi \sum_{i=0}^{k} \int_{iL}^{(i+1)L} \big(R_i(x)\big)^2 \, \mathrm{d}x \tag{19}$$

, where L is the interval length and $R_i(x)$ is the i^{th} function of $R(x)$, which lies between $iL < x < (i+1)L$. While the above can certainly be computed individually, this process is very time-consuming. I have therefore created a method which *collapses* the set of piecewise functions, $R(x)$ into a single function, $R(l)$. Since all intervals are of the same length L, each radius function $R_i(x)$ can be translated towards the left, and aggregated vertically:

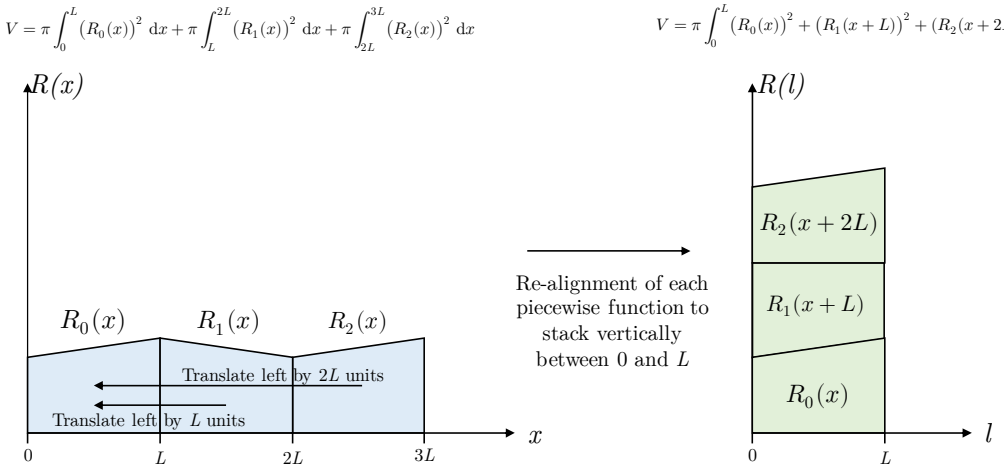

Figure 15. The additive property of the integrals of piecewise functions leading to a simplified method of computing volume.

By performing the operations as depicted in Figure 15, after *collapsing* the piecewise functions, Equation (19) can be simplified to:

$$V = \pi \int_0^L \sum_{i=0}^{k} \big(R_i(x + iL)\big)^2 \mathrm{d}x \tag{20}$$

The summation symbol can be evaluated by substituting (18) into (20):

$$V = \pi \int_0^L \begin{pmatrix} (0.03532 + 1.412x - 0.7802x^2 + 0.2432x^3)^2 + \\ (0.2605 + 0.7391(x+L) - 0.1101(x+L)^2 + 0.02076(x+L)^3)^2 + \\ \vdots \\ (-2.398\text{E}{+}4 + 1118(x+64L) - 17.39(x+64L)^2 + 0.09014(x+64L)^3)^2 \end{pmatrix} \mathrm{d}x$$

$$V = \pi \int_0^{1.004} \begin{pmatrix} (0.03532 + 1.412x - 0.7802x^2 + 0.2432x^3)^2 + \\ (0.9126 + 0.5809x - 0.04753x^2 + 0.02076\mathrm{x}^3)^2 + \\ \vdots \\ (0.5117 - 0.1688x - 0.01317x^2 - 0.09014x^3)^2 \end{pmatrix} \mathrm{d}x$$

$$V = \pi \int_0^{1.004} \begin{pmatrix} 0.001247 + 0.09973x + 1.9383x^2 \\ -2.1858x^3 + 1.2955x^4 - 0.3796x^5 + 0.05917x^6 \end{pmatrix} + \begin{pmatrix} 0.8329 + 1.060x + 0.2507x^2 \\ -0.01732x^3 + 0.02638x^4 - 0.001974x^5 + 0.0004311x^6 \end{pmatrix} + \begin{pmatrix} \vdots \\ 0.2619 - 0.1728x + 0.04197x^2 \\ -0.09670x^3 + 0.03061x^4 - 0.002374x^5 0.008126x^6 \end{pmatrix} \mathrm{d}x$$

$$V = \pi \int_0^{1.004} 428.5 + 0.06012x + 1.4770x^2 + 2.3717x^3 + 1.353x^4 - 0.4013x^5 + 0.07348x^6 \mathrm{d}x$$

$$V = |428.5x - 0.03006x^2 + 0.4923x^3 + 0.5929x^4 + 0.2706x^5 - 0.06688x^6 + 0.01050x^7|_0^{1.004}$$

$$V = 1351.830627659625\ldots$$

$$V = \boxed{1351.83\ \mathrm{m}^3{}_{(2d.p.)}} \tag{21}$$

The surface area can be calculated by substituting Equations (16-18) into Equation (4):

$$A = \pi \int_0^b R(x) \left(\sqrt{1 + \left(\frac{\mathrm{d}T}{\mathrm{d}x}\right)^2} + \sqrt{1 + \left(\frac{\mathrm{d}B}{\mathrm{d}x}\right)^2} \right) \mathrm{d}x$$

Since $T(x), B(x)$ and $R(x)$ are piecewise functions, the total surface area can be calculated by summing up the surface area for each individual i^{th} interval:

$$A = \pi \sum_{i=0}^{k} \int_{iL}^{(i+1)L} R_i(x) \left(\sqrt{1 + \left(\frac{\mathrm{d}T_i}{\mathrm{d}x}\right)^2} + \sqrt{1 + \left(\frac{\mathrm{d}B_i}{\mathrm{d}x}\right)^2} \right) \mathrm{d}x \tag{22}$$

, where L is the interval length and $R_i(x)$ is the i^{th} function of $R(x)$, T_i is the i^{th} function of $T(x)$, B_i is the i^{th} function of $B(x)$, which all lies between $iL < x < (i+1)L$. Applying the function *collapsing* as depicted in Figure 15, Equation (22) can be simplified to:

$$A = \pi \int_0^L \sum_{i=0}^{k} R_i(x + iL) \left(\sqrt{1 + \left(\frac{\mathrm{d}(T_i(x+iL))}{\mathrm{d}x}\right)^2} + \sqrt{1 + \left(\frac{\mathrm{d}(B_i(x+iL))}{\mathrm{d}x}\right)^2} \right) \mathrm{d}x$$

$$A = \pi \int_0^L \left(\begin{array}{c} (0.03532 + 1.412x - 0.7802x^2 + 0.2432x^3)(\\ \sqrt{1 + \left(\frac{\mathrm{d}}{\mathrm{d}x}(1.911 + 1.342x - 0.5741x^2 + 0.1933x^3)\right)^2} + \\ \sqrt{1 + \left(\frac{\mathrm{d}}{\mathrm{d}x}(1.841 - 1.482x + 0.9862x^2 - 0.2932x^3)\right)^2}) \\ \vdots \\ ((-2.398\text{E+}4 + 1118(x + 64L) - 17.39(x + 64L)^2 + 0.09014(x + 64L)^3)(\\ \sqrt{1 + \left(\frac{\mathrm{d}}{\mathrm{d}x}\begin{pmatrix} 2.292\text{E+}04 - 1070(x + 64L) + \\ 16.66(x + 64L)^2 + -0.08646(x + 64L)^3 \end{pmatrix}\right)^2} + \\ \sqrt{1 + \left(\frac{\mathrm{d}}{\mathrm{d}x}\begin{pmatrix} -2.503\text{E+}04 + 1167(x + 64L) - \\ 18.12(x + 64L)^2 + 0.09382(x + 64L)^3 \end{pmatrix}\right)^2}) \end{array} \right) \mathrm{d}x$$

Since the above cannot be solved analytically, A is calculated using a graphical calculator:

$$A = 1018.059027142908\ldots$$

$$A = \boxed{1018.06\ \mathrm{m}^2{}_{(2d.p.)}} \tag{23}$$

Therefore, the volume of the fuselage is 1351.83 m^3 and the surface area is 1018.06 m^2.

With regard to the reliability of the results, as Figure 13 indicates, the top and bottom curves seem match the true boundaries exactly, without experiencing major issues with underfitting or overfitting, suggesting that the results are highly reliable. However, I wondered, did this method fail or succeed, and does the volume and surface area in fact match the real-life?

3.3 Validation of results with real-life model

To validate my results, I downloaded the official CAD drawings for the A350-900 aircraft (Airbus, 2014). After isolating the fuselage of the aircraft, I was able to load the 3D model:

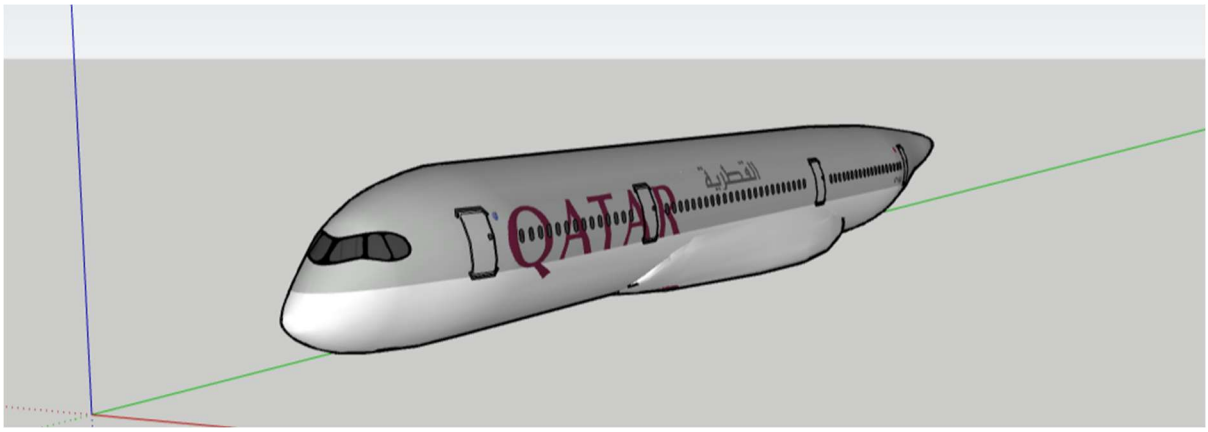

Figure 16. A 3D model of the A350-900 loaded using SketchUp.

By inspecting the entity information of the aircraft in the side panel of SketchUp, I was able to compare the volume and the surface area of the fuselage to my results:

Method	**Volume (m^3)**	**Surface Area (m^2)**	**Percentage Error in Volume**	**Percentage Error in Surface Area**
Exact (Official)	1516.92	1097.09	--	--
Simple Quartic Regression	1341.64	1011.69	11.55%	7.784%
Piecewise Regression with Cubic Splines	**1351.83**	**1018.06**	**10.88%**	**7.204%**

Table 1. A comparison of the results.

As shown in the table above, the method of piecewise regression with cubic splines is superior in terms of accuracy in both the volume and surface area of the fuselage. Since the percentage error is also reasonably low, it can be said that my model is highly accurate and reliable. While I was able to successfully obtain the volume and surface area of the fuselage, this brings me back to the second main aim of this investigation – how do I optimise the dimensions of the fuselage, such that there is a minimal aerodynamic drag incurred?

4 Optimisation of length and radius of fuselage

In the airline industry, one of the major aspects for determining the profitability of an airline is the fuel efficiency of the aircraft. If the shape of the aircraft is suboptimal, this will cause large amounts of aerodynamic drag, reducing the fuel efficiency of the flight. I am therefore intrigued to find out, given two scalars:

1. The length multiplier (k_l), controls the scale factor of the length of the fuselage
2. The radius multiplier (k_r), controls the scale factor of the radius of the fuselage

What are the optimum values for k_l and k_r to minimise aerodynamic drag D, given that the volume of the aircraft V must stay constant at 1351.83 m^3?

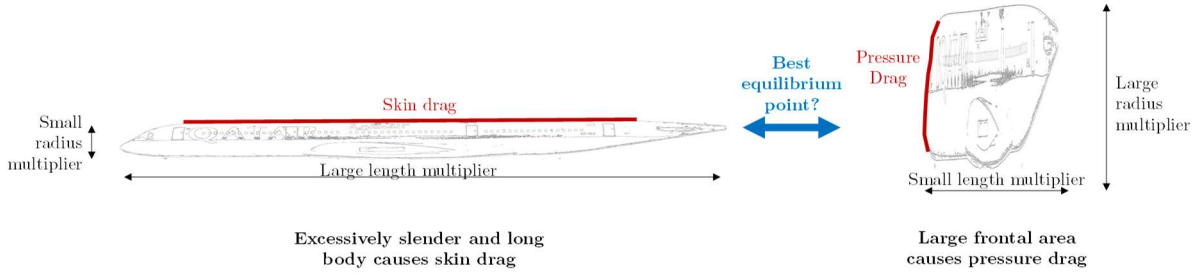

Figure 17. Issues with disproportionate multipliers resulting in large amounts of aerodynamic drag.

4.1 Formulation for the Drag Equation (D)

According to NASA (2021), the aerodynamic drag of an aircraft varies largely between different aircrafts and conditions, such as speed, altitude, temperature. However, the aerodynamic drag of an aircraft is generally agreed to follow the following relation:

$$D \propto 0.07447 A_{wetted} + 0.9255 A_{frontal} \tag{24}$$

where A_{wetted} is the wetted area of the fuselage, which corresponds to the surface area of the aircraft, $A_{frontal}$ is the area of the fuselage projected onto the front view. By substituting in the area function obtained previously in Equation (22) into (24), this gives:

$$D = 0.07447 \int_0^L \sum_{i=0}^{k} R_i(x+iL) \left(\sqrt{1 + \left(\frac{\mathrm{d}(T_i(x+iL))}{\mathrm{d}x} \right)^2} + \sqrt{1 + \left(\frac{\mathrm{d}(B_i(x+iL))}{\mathrm{d}x} \right)^2} \right) \mathrm{d}x + 0.9255(\pi R_{ma}\ )^2$$

To represent the drag equation as a multivariable function, according to the geometry of the aircraft, when k_r is changed, this vertically scales the position of the curves along the z-axis with a scale factor of k_r. When k_l is changed, this horizontally scales the position of the curves with a scale factor of $\frac{1}{k_l}$. Therefore, the equation for aerodynamic drag is:

$$D(k_l, k_r) = 0.07447 \int_0^{k_l L} \sum_{i=0}^{k} k_r R_i \left(\frac{x}{k_l} + iL \right) \left(\sqrt{1 + \left(k_r \frac{\mathrm{d}(T_i(\frac{x}{k_l} + iL))}{\mathrm{d}x} \right)^2} + \sqrt{1 + \left(k_r \frac{\mathrm{d}(B_i(\frac{x}{k_l} + iL))}{\mathrm{d}x} \right)^2} \right) \mathrm{d}x + 0.9255\pi(k_r R_{ma}\ )^2 \tag{26}$$

4.2 Formulation of the Volume Equation (V)

Similar to the approach in Equation (26), by applying the relevant scale factors to Equation (20), a multivariable equation with inputs k_l and k_r is calculated to be:

$$V(k_l, k_r) = \pi \int_0^{k_l L} \sum_{i=0}^{k} \left(k_r R_i \left(\frac{x}{k_l} + iL \right) \right)^2 \mathrm{d}x \tag{27}$$

4.3 Lagrange Multipliers

In order to visualise how the aerodynamic drag function $D(k_l, k_r)$ is minimised given a constrained volume, a graph of the drag function with k_l on the x-axis and k_r on the y-axis is created using the `matplotlib` and `sympy` Python packages, as shown below:

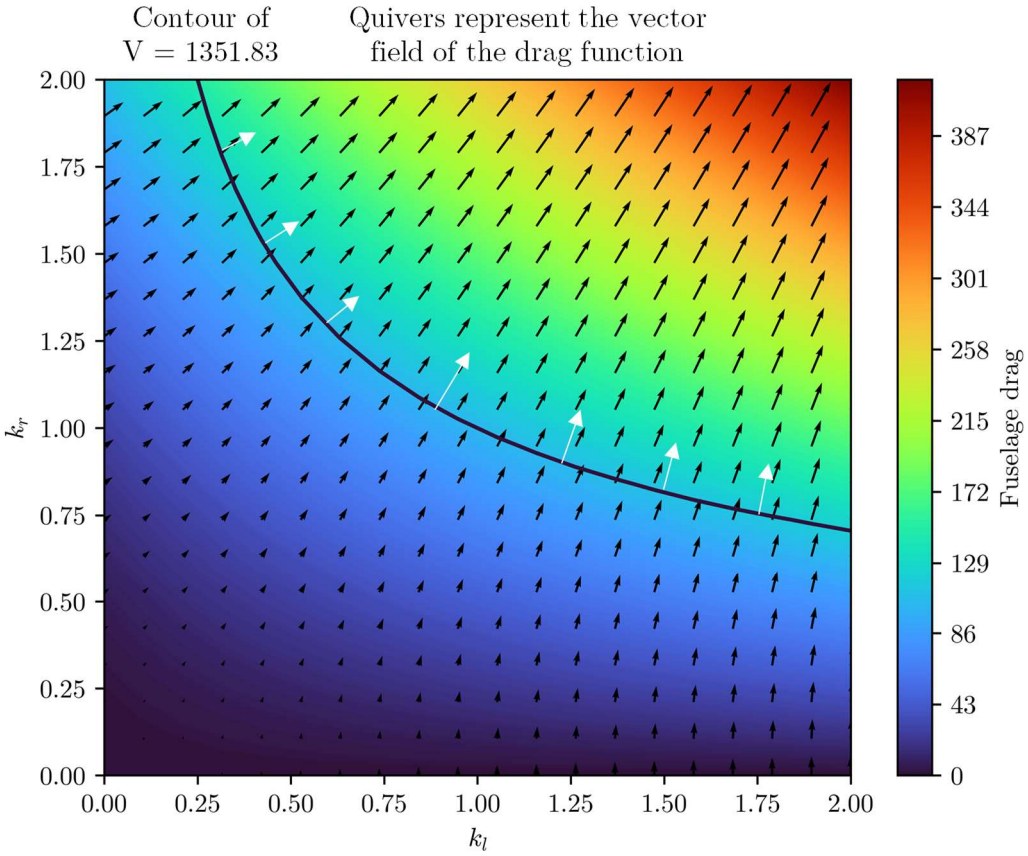

Figure 18. A colourmap and the vector field of the drag function with the inputs k_l and k_r and a contour line for which $V = 1351.83$ m^3

In the graph above, the colour of a specific position denotes the strength of the aerodynamic drag D. As expected, the aerodynamic drag appears to increase as k_l and k_r increases. The black arrow on the graph denotes the vector field of the aerodynamic drag function, which points to the direction of its gradient of steepest ascent, ∇D, whereas the white arrows on the graph denote the vector field of the volume function, which has the direction ∇V.

Given that the aircraft manufacturer would like to minimise the aerodynamic drag D for the A350-900, then the optimum dimensions would be the coordinates somewhere along the contour line for which the D is at its minimum. To find this location, ∇D (black arrows) must point in the same direction as the ∇V (white arrows), forming the following equality:

$$\nabla D = \lambda \nabla V \tag{28}$$

, where ∇D is the gradient of the drag function, ∇V is the gradient of the volume function, and λ is the Lagrange multiplier. By rewriting the gradients in its vector form:

$$\begin{bmatrix} \frac{\partial D}{\partial k_l} \\ \frac{\partial D}{\partial k_r} \end{bmatrix} = \lambda \begin{bmatrix} \frac{\partial V}{\partial k_l} \\ \frac{\partial V}{\partial k_r} \end{bmatrix} \tag{29}$$

And combining it with the constraint function of $V(k_l, k_r) = 1351.83\ m^3$, this yields a system of equations with three unknowns, k_l, k_r and λ:

$$\begin{cases} \frac{\partial D}{\partial k_l} - \lambda \frac{\partial V}{\partial k_l} = 0 \\ \frac{\partial D}{\partial k_r} - \lambda \frac{\partial V}{\partial k_r} = 0 \\ V(k_l, k_r) = 1351.83 \end{cases} \tag{30}$$

However, because the partial derivatives of the drag equation and the volume equation are too complex to be solved by hand, I decided to pass Equation (30) through a solver in the `scipy` Python package. Solving Equation (30) yielded one solution:

k_l	k_r	λ
0.8195290969499447	1.1046324860579495	0.05084322098344

Table 2. The results after the optimisation of the fuselage dimensions using Lagrange multipliers.

As the results indicate, to increase the aerodynamical efficiency of the aircraft, the length of A350-900 cabin should be reduced by 18.04% and the radius should be increased by 10.46%. However, I wondered how much aerodynamic drag is reduced by this change. By substituting in k_l and k_r into the drag equation, I am surprised to find that the aerodynamic drag has only been reduced by 0.94%, which leads me to believe that the engineers at Airbus has already optimised the fuselage to its greatest extent.

5 Conclusion

In conclusion, this investigation was able to accurately determine the volume and the surface area of the fuselage of an A350-900, through the use of simple quartic regression and piecewise regression using cubic splines. This investigation was also able to optimise the dimensions of

the aircraft to minimise the aerodynamic drag, through the use of Lagrange multipliers. This investigation has also demonstrated that similar concepts can easily applied to other types of aircraft.

6 Evaluation

Throughout this investigation, the maximisation of accuracy and the minimisation of human bias has been placed at a high priority. By processing the images through a sophisticated algorithm that automatically extracts the location of the pixels, this ensures that there are plenty of data for the regression, increasing the representativeness of the curves. By conducting the measurement of the volume and surface area using different techniques, I was able to demonstrate a high level of precision throughout the process. Most of the tedious calculation works, such as summation and integration has also been performed using Python scripts, which increases the reproducibility of the data and also minimises human input errors.

Despite efforts to maximise the precision of the measurements there are several issues with the depiction of the aircraft as being formed entirely with circular disks. The generalisation that the cross-section of the aircraft is perfectly circular is also untrue, specifically near the fuel tanks, where there is a bulge that is obviously not circular. This is potentially one of the main reasons why the volume of the aircraft is underestimated by 10%, in which major volumes of the aircraft are not included in the calculations. The assumption that the fuselage is composed of circular disks are also not true to some aircraft families, such as the Antonov-225, Boeing B747 or the DC-1.

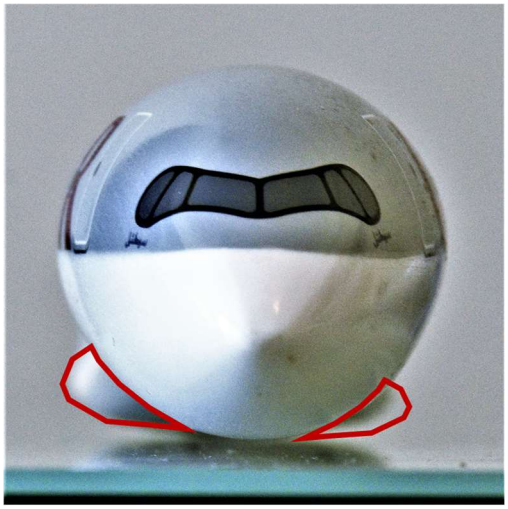

Figure 19. Demonstration that the fuel tank of the fuselage is often not composed of circular disks, which causes an underestimation of the volume and surface area.

In order to solve this issue, the accuracy can be further improved by obtaining regular cross-sections of the aircraft, and to model its shape using parametric equations. With the use of triple integrals, the issue with the underrepresentation of the volume and surface area of the fuselage can be mitigated. Furthermore, there may also be a possibility that the aircraft model I own do not actually match the real aircraft. To solve this, I could have used a 3D laser scanner or use photogrammetry but considering that that the volume and surface area has already been calculated to a high degree of accuracy, this was unneeded.

One of the major limitations during the piecewise regression using cubic splines was that I was uncertain of the number of intervals to use. Despite the abundance of data, I only chose 65 intervals for the regression due to limitations in the hardware performance. This can be solved by using professional statistical programming languages such as R, which would also have allowed me to have access to a greater number of tools and data visualisations.

Regardless, while this IA has exhibited some limitations when it comes to the assumptions of the aircraft geometry, it has demonstrated that it is possible to accurately model the volume and surface area of an aircraft, as well as perform the optimisation of aircraft dimensions solely from a single photo, this process can easily be replicated by other researchers and applied to other aircraft models.

7 References

"Autocad 3D View Aircraft Drawings." Airbus, https://www.airbus.com/aircraft/support-services/airport-operations-and-technical-data/autocad-3-view-aircraft-drawings.html.

Bertsekas, Dimitri P. Constrained Optimization and Lagrange Multiplier Methods. Academic Press, 1982.

Cleveland, William S. "Robust Locally Weighted Regression and Smoothing Scatterplots." Journal of the American Statistical Association, vol. 74, no. 368, Dec. 1979, pp. 829–36. DOI.org (Crossref), https://doi.org/10.1080/01621459.1979.10481038.

Hall, Peter, and J. D. Opsomer. "Theory for Penalised Spline Regression." Biometrika, vol. 92, no. 1, Mar. 2005, pp. 105–18. DOI.org (Crossref), https://doi.org/10.1093/biomet/92.1.105.

Perperoglou, Aris, et al. "A Review of Spline Function Procedures in R." BMC Medical Research Methodology, vol. 19, no. 1, Dec. 2019, p. 46. DOI.org (Crossref), https://doi.org/10.1186/s12874-019-0666-3.

Silverman, B. W. "Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting." Journal of the Royal Statistical Society: Series B (Methodological), vol. 47, no. 1, Sept. 1985, pp. 1–21. DOI.org (Crossref), https://doi.org/10.1111/j.2517-6161.1985.tb01327.x.

The Drag Equation. https://www.grc.nasa.gov/www/k-12/airplane/drageq.html. Accessed 16 Feb. 2022.

Wand, M. P. "A Comparison of Regression Spline Smoothing Procedures." Computational Statistics, vol. 15, no. 4, Dec. 2000, pp. 443–62. DOI.org (Crossref), https://doi.org/10.1007/s001800000047.

8 Appendix

Filename: 1_sobel.py

```
import cv2

raw = cv2.imread('data/0_side.jpg')
cv2.imwrite('data/0_side_cropped.jpg', raw[2673:3237, 102:5871])

raw_grey = cv2.cvtColor(raw, cv2.COLOR_BGR2GRAY)
raw_grey_canny = cv2.Canny(raw_grey, threshold1=10, threshold2=50, L2gradient=True)
cv2.imwrite('data/1_side_edges.jpg', raw_grey_canny)
cv2.imwrite('data/0_side_edges_cropped.jpg', cv2.bitwise_not(raw_grey_canny)[2673:3237,
102:5871])
```

Filename: 3_coords.py

```
import cv2
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

ASPECT_RATIO = 564/5769
fm.fontManager.addfont('fonts/latinmodern-math.otf')
plt.rcParams["font.family"] = "Latin Modern Math"
plt.rcParams["mathtext.fontset"] = "cm"
plt.imshow(plt.imread('data/0_side_edges_cropped.jpg'), extent=[0, 65.27, 0,
65.27*ASPECT_RATIO], cmap='gray')

for part in ['top', 'bottom']:
    image = cv2.cvtColor(cv2.imread(f'data/3_side_edges_clean_{part}.png')[2673:3237,
102:5871], cv2.COLOR_BGR2GRAY)
    h, w = image.shape
    pxToM = 65.27 / (w-1) # multiplier

    coords = [(x*pxToM, (h-y)*pxToM) for x in range(w) for y in range(h) if image[y, x] >
128]
    df = pd.DataFrame(coords, columns=('x', 'y'))
    df.to_csv(f'data/3_coords_{part}.csv', index=False)

    plt.plot(df.x, df.y, 'o', color='#E53935' if part == 'top' else '#1E88E5', markersize=.5)

x0, x1, y0, y1 = plt.axis()
plt.axis((6, 16, 4.5, 6.5))
plt.xlabel('$x$ (metres)')
plt.ylabel('$z$ (metres)')
plt.savefig('data/3_coords.png', dpi=600, transparent=True)
```

Filename: 4_polynomial.py

```
import numpy as np
import pandas as pd
from rich import inspect, print
import sympy as sp
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

NUM_INTERVALS = 65
SAMPLES_PER_INTERVAL = 1000
ASPECT_RATIO = 564/5769

df_top = pd.read_csv('data/3_coords_top.csv')
```

```
df_bottom = pd.read_csv('data/3_coords_bottom.csv')

MIN = max(df_top.x.min(), df_bottom.x.min())
MAX = min(df_top.x.max(), df_bottom.x.max())
BREAKPOINTS = np.linspace(MIN, MAX, num=NUM_INTERVALS+1)
INTERVAL_WIDTH = (MAX - MIN) / NUM_INTERVALS
x = sp.Symbol('x', positive=True, real=True)

def fit(deg):
    poly_top, poly_bottom = None, None

    df_top_filtered = df_top[(df_top.x >= MIN) & (df_top.x <= MAX)].dropna()
    df_bottom_filtered = df_bottom[(df_bottom.x >= MIN) & (df_bottom.x <= MAX)].dropna()

    poly_top = np.polynomial.polynomial.polyfit(df_top_filtered.x, df_top_filtered.y,
deg=deg, full=False)
    poly_bottom = np.polynomial.polynomial.polyfit(df_bottom_filtered.x,
df_bottom_filtered.y, deg=deg, full=False)
    poly_radius = (poly_top - poly_bottom) / 2
    poly_camber = (poly_top + poly_bottom) / 2
    R = sum(coeff*x**order for order, coeff in enumerate(poly_radius))
    T = sum(coeff*x**order for order, coeff in enumerate(poly_top))
    B = sum(coeff*x**order for order, coeff in enumerate(poly_bottom))
    V = np.pi*sp.Integral(R**2, (x, MIN, MAX)).evalf()
    S = np.pi*sp.Integral(R*(sp.sqrt(1 + sp.diff(T, x)**2) + sp.sqrt(1 + sp.diff(B, x)**2)),
(x, MIN, MAX)).evalf()
    return [deg, V, S, poly_top, poly_bottom, poly_radius, poly_camber]

def printeq(poly):
    print(' + '.join(f'{coeff:.4g}x^{order}' for order, coeff in enumerate(poly)))

if __name__ == '__main__':
    _deg, V, S, poly_top, poly_bottom, poly_radius, poly_camber = fit(4)

    printeq(poly_top)
    printeq(poly_bottom)
    printeq(poly_radius)
    print(V)
    print(S)

    xs = np.linspace(MIN, MAX, 10000)
    yts = sum(coeff*xs**order for order, coeff in enumerate(poly_top))
    ybs = sum(coeff*xs**order for order, coeff in enumerate(poly_bottom))

    fm.fontManager.addfont('fonts/latinmodern-math.otf')
    plt.rcParams["font.family"] = "Latin Modern Math"
    plt.rcParams["mathtext.fontset"] = "cm"
    fig, ax = plt.subplots()
    ax.imshow(plt.imread('data/0_side_edges_cropped.jpg'), extent=[0, 65.27, 0,
65.27*ASPECT_RATIO], cmap='gray')
    ax.plot(xs, yts, linestyle='dashed', color='#E53935', linewidth=.75)
    ax.plot(xs, ybs, linestyle='dashed', color='#1E88E5', linewidth=.75)
    ax.fill_between(xs, yts, ybs, color='#E5393511', linewidth=0)
    x0, x1, y0, y1 = plt.axis()
    plt.axis((x0-2, x1+2, y0-4, y1+4))
    ax.set_xlabel('$x$ (metres)')
    ax.set_ylabel('$z$ (metres)')
    plt.savefig('data/4_polynomial.png', dpi=600, transparent=True)
```

```
    plt.clf()

    ax = plt.axes(projection='3d')
    for x in np.linspace(MIN, MAX, 32):
        theta = np.linspace(0, 2*np.pi, 360)
        radius = sum(coeff*x**order for order, coeff in enumerate(poly_radius))
        camber = sum(coeff*x**order for order, coeff in enumerate(poly_camber))
        ax.plot3D(np.full(360, x), radius*np.sin(theta), radius*np.cos(theta)+camber,
color='black', linewidth=.5)

    ax.plot3D(xs, np.zeros(10000), yts, color='#E53935', linestyle='dashed', linewidth=.75)
    ax.plot3D(xs, np.zeros(10000), ybs, color='#1E88E5', linestyle='dashed', linewidth=.75)

    ax.set_xlabel('$x$ (metres)')
    ax.set_ylabel('$y$ (metres)')
    ax.set_zlabel('$z$ (metres)')

    ax.set_xlim((-2, 68))
    ax.set_ylim((-30, 30))
    ax.set_zlim((-30, 30))

    plt.savefig('data/4_polynomial_3d.png', dpi=600, transparent=True)

    # vs = [(deg, v, s) for deg, v, s, *_ in [fit(deg) for deg in range(2, 30)]]
    # df = pd.DataFrame(vs)
    # df.to_csv('data/4_vs_polynomial.csv', index=False)
```

Filename: 4_spline.py

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from patsy import dmatrix
import scipy.integrate as integrate
from rich import inspect, print
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
import math
import sympy as sp

ASPECT_RATIO = 564/5769
NUM_INTERVALS = 65
SAMPLES_PER_INTERVAL = 1000

df_top = pd.read_csv('data/3_coords_top.csv')
df_bottom = pd.read_csv('data/3_coords_bottom.csv')

MIN = max(df_top.x.min(), df_bottom.x.min())
MAX = min(df_top.x.max(), df_bottom.x.max())
BREAKPOINTS = np.linspace(MIN, MAX, num=NUM_INTERVALS+1)
INTERVAL_WIDTH = (MAX - MIN) / NUM_INTERVALS

def printeq(poly):
    print('+'.join(f'{coeff:.4g}x^{order}' for order, coeff in enumerate(poly)))

if __name__ == '__main__':
    poly_top, poly_bottom = [], []
    for df, df_type in zip((df_top, df_bottom), ('top', 'bottom')):
```

```
        # constuct a generalised linear model from B-spline basis matrix, of degree 3.
        model = sm.GLM(df.y, dmatrix(
            "bs(x, knots=k, include_intercept=False, degree=3)", {
                'x': df.x,
                'k': BREAKPOINTS
            }, return_type='dataframe'
        )).fit()

        # since we cannot extract coefficients of the polynomial,
        # we must resample it based on the model above.
        sample = np.linspace(df.x.min(), df.x.max(), SAMPLES_PER_INTERVAL*NUM_INTERVALS)
        sample_df = pd.DataFrame({
            "x": sample,
            "y": model.predict(dmatrix(
                "bs(x, knots=k, include_intercept=False, degree=3)", {
                    'x': sample,
                    'k': BREAKPOINTS
                }, return_type='dataframe'
            ))
        })

        for k in range(1, len(BREAKPOINTS)):
            lower_bound, upper_bound = BREAKPOINTS[k-1], BREAKPOINTS[k]
            sample_bounded = sample_df[(sample_df.x >= lower_bound) & (sample_df.x <=
upper_bound)].dropna()
            # plt.plot(sample_bounded.x, sample_bounded.y)

            poly = np.polynomial.polynomial.polyfit(sample_bounded.x, sample_bounded.y,
deg=3, full=False)
            poly_translated = np.polynomial.polynomial.polyfit(sample_bounded.x -
INTERVAL_WIDTH*(k-1), sample_bounded.y, deg=3, full=False)

            (poly_top if df_type == 'top' else poly_bottom).append({
                'i': k,
                'poly': poly,
                'poly_translated': poly_translated,
            })

    t_coeffs_ut = [poly_top[k]['poly'] for k in range(NUM_INTERVALS)]
    t_coeffs = [poly_top[k]['poly_translated'] for k in range(NUM_INTERVALS)]
    t_coeffs_df = pd.DataFrame(t_coeffs, columns=('d', 'c', 'b', 'a'))
    t_coeffs_df.to_csv('data/4_t_coeffs.csv', index=False)

    b_coeffs_ut = [poly_bottom[k]['poly'] for k in range(NUM_INTERVALS)]
    b_coeffs = [poly_bottom[k]['poly_translated'] for k in range(NUM_INTERVALS)]
    b_coeffs_df = pd.DataFrame(b_coeffs, columns=('d', 'c', 'b', 'a'))
    b_coeffs_df.to_csv('data/4_b_coeffs.csv', index=False)

    r_coeffs_ut = [(poly_top[k]['poly'] - poly_bottom[k]['poly']) / 2 for k in
range(NUM_INTERVALS)]
    r_coeffs = [(poly_top[k]['poly_translated'] - poly_bottom[k]['poly_translated']) / 2 for
k in range(NUM_INTERVALS)]
    r_coeffs_df = pd.DataFrame(r_coeffs, columns=('d', 'c', 'b', 'a'))
    r_coeffs_df.to_csv('data/4_r_coeffs.csv', index=False)

    c_coeffs_ut = [(poly_top[k]['poly'] + poly_bottom[k]['poly']) / 2 for k in
range(NUM_INTERVALS)]
```

```
    printeq(t_coeffs_ut[0])
    printeq(t_coeffs_ut[-1])
    printeq(b_coeffs_ut[0])
    printeq(b_coeffs_ut[-1])

    xs = np.linspace(MIN, MAX, 10000)
    yts, ybs = [], []
    for x in xs:
        ii = min(math.floor(x/INTERVAL_WIDTH), NUM_INTERVALS-1)
        yts.append(sum(coeff*x**order for order, coeff in enumerate(t_coeffs_ut[ii])))
        ybs.append(sum(coeff*x**order for order, coeff in enumerate(b_coeffs_ut[ii])))

    fm.fontManager.addfont('fonts/latinmodern-math.otf')
    plt.rcParams["font.family"] = "Latin Modern Math"
    plt.rcParams["mathtext.fontset"] = "cm"
    fig, ax = plt.subplots()
    ax.imshow(plt.imread('data/0_side_edges_cropped.jpg'), extent=[0, 65.27, 0,
65.27*ASPECT_RATIO], cmap='gray')
    ax.plot(xs, yts, linestyle='dashed', color='#E53935', linewidth=.75)
    ax.plot(xs, ybs, linestyle='dashed', color='#1E88E5', linewidth=.75)
    ax.fill_between(xs, yts, ybs, color='#E5393511', linewidth=0)
    for ii in range(1, NUM_INTERVALS):
        x = ii*INTERVAL_WIDTH
        ax.plot([x, x], [
            sum(coeff*x**order for order, coeff in enumerate(t_coeffs_ut[ii])),
            sum(coeff*x**order for order, coeff in enumerate(b_coeffs_ut[ii]))
        ], linewidth=.25, color='gray')
    x0, x1, y0, y1 = plt.axis()
    plt.axis((x0-2, x1+2, y0-4, y1+4))
    ax.set_xlabel('$x$ (metres)')
    ax.set_ylabel('$z$ (metres)')
    plt.savefig('data/4_spline.png', dpi=600, transparent=True)

    plt.clf()

    ax = plt.axes(projection='3d')
    for x in np.linspace(MIN, MAX, 32):
        ii = min(math.floor(x/INTERVAL_WIDTH), NUM_INTERVALS-1)
        theta = np.linspace(0, 2*np.pi, 360)

        radius = sum(coeff*x**order for order, coeff in enumerate(r_coeffs_ut[ii]))
        camber = sum(coeff*x**order for order, coeff in enumerate(c_coeffs_ut[ii]))
        ax.plot3D(np.full(360, x), radius*np.sin(theta), radius*np.cos(theta)+camber,
color='black', linewidth=.5)

    ax.plot3D(xs, np.zeros(10000), yts, color='#E53935', linestyle='dashed', linewidth=.75)
    ax.plot3D(xs, np.zeros(10000), ybs, color='#1E88E5', linestyle='dashed', linewidth=.75)

    ax.set_xlabel('$x$ (metres)')
    ax.set_ylabel('$y$ (metres)')
    ax.set_zlabel('$z$ (metres)')

    ax.set_xlim((-2, 68))
    ax.set_ylim((-30, 30))
    ax.set_zlim((-30, 30))

    plt.savefig('data/4_spline_3d.png', dpi=600, transparent=True)
```

```
    # encapsulating the sum within the integrand
    vol, vol_unc = integrate.quad(
        lambda w: sum(np.pi*np.power(a*np.power(w,3) + b*np.power(w,2) + c*w + d, 2) for d,
c, b, a in r_coeffs),
        0, INTERVAL_WIDTH
    )
    sa, sa_unc = integrate.quad(
        lambda w: sum(np.pi*(
            a*np.power(w,3) + b*np.power(w,2) + c*w + d) * (
                np.sqrt(1 + np.power(3*ta*np.power(w, 2) + 2*tb*w + tc, 2)) +
                np.sqrt(1 + np.power(3*ba*np.power(w, 2) + 2*bb*w + bc, 2))
            ) for (d, c, b, a), (td, tc, tb, ta), (bd, bc, bb, ba) in zip(r_coeffs, t_coeffs,
b_coeffs)),
        0, INTERVAL_WIDTH
    )

    print(f'Volume: {vol} ± {vol_unc} m3')
    print(f'Surface Area: {sa} ± {sa_unc} m2')
```

Filename: 6_lagrange.py

```
import pandas as pd
import sympy as sp
from scipy.optimize import fsolve
from rich import print, inspect
from timeit import default_timer as timer
from matplotlib import pyplot as plt
import matplotlib.font_manager
import numpy as np
spline = __import__('4_spline')

# setups
matplotlib.font_manager.fontManager.addfont('fonts/latinmodern-math.otf')
plt.rcParams["font.family"] = "Latin Modern Math"
plt.rcParams["mathtext.fontset"] = "cm"
printl = lambda f: print(sp.latex(f, long_frac_ratio=3, order='lex'), '\n____')

# variables
t_coeffs = tuple(pd.read_csv('data/4_t_coeffs.csv').itertuples(index=False, name=None))
b_coeffs = tuple(pd.read_csv('data/4_b_coeffs.csv').itertuples(index=False, name=None))
r_coeffs = tuple(pd.read_csv('data/4_r_coeffs.csv').itertuples(index=False, name=None))
l = sp.Symbol('l', positive=True, real=True) # length
kr = sp.Symbol('k_r', real=True) # radius multiplier
kl = sp.Symbol('k_l', real=True) # length multiplier
INTERVAL_WIDTH = spline.INTERVAL_WIDTH # 1.0039797556812118
V_TARGET = 1351.8306276596234 # m3
# MAX_R = max(sp.maximum(a*l**3 + b*l**2 + c*l + d, l, sp.sets.Interval(0, INTERVAL_WIDTH))
for d, c, b, a in r_coeffs) # 3.08446945187677
MAX_R = 3.08446945187677
C_P_REL = 0.9255319158724 # relative because of floating point precision issues

V = sp.pi*kr**2*kl*sp.Integral(sum((a*l**3 + b*l**2 + c*l + d)**2 for d, c, b, a in
r_coeffs), (l, 0, INTERVAL_WIDTH))
V_func = sp.lambdify([kl, kr], V)
V_grad_func = sp.lambdify([kl, kr], [sp.diff(V, kl), sp.diff(V, kr)])

A = sp.pi*kl*kr*sp.Integral(sum((a*l**3 + b*l**2 + c*l + d) * (
    sp.sqrt(1 + kr**2*(3*ta*l**2 + 2*tb*l + tc)**2) +
    sp.sqrt(1 + kr**2*(3*ba*l**2 + 2*bb*l + bc)**2)
```

```
) for (d, c, b, a), (td, tc, tb, ta), (bd, bc, bb, ba) in zip(r_coeffs, t_coeffs, b_coeffs)),
(l, 0, INTERVAL_WIDTH))

# A = 2*sp.pi*kl*kr*sp.Integral(sum((a*l**3 + b*l**2 + c*l + d)*sp.sqrt(1 + kr**2*(3*a*l**2 +
2*b*l + c)**2) for d, c, b, a in r_coeffs), (l, 0, INTERVAL_WIDTH))
A_func = sp.lambdify([kl, kr], A)
A_grad_func = sp.lambdify([kl, kr], [sp.diff(A, kl), sp.diff(A, kr)])

D = A * (1-C_P_REL) + sp.pi*(MAX_R*kr)**2 * C_P_REL
D_func = sp.lambdify([kl, kr], D)
D_grad_func = sp.lambdify([kl, kr], [sp.diff(D, kl), sp.diff(D, kr)])

## CALCULATIONS FOR CONSTRAINED OPTIMISATION
if True:
    def eqs(X):
        klv, krv, λv = X
        δVδkl, δVδkr = V_grad_func(klv, krv)
        δDδkl, δDδkr = D_grad_func(klv, krv)
        # print(f'@ {klv}, {krv}: ∇V=[{δVδkl}, {δVδkr}], ∇D=[{δDδkl}, {δDδkr}],
V={V_func(klv, krv)}')
        return [
            δDδkl - (λv * δVδkl),
            δDδkr - (λv * δVδkr),
            V_func(klv, krv) - V_TARGET,
        ]

    duo = D_func(1, 1)
    klo, kro, λo = fsolve(eqs, [1., 1., 1.])
    do = D_func(klo, kro)

    print(f'Solution: kl={klo}, kr={kro}, λ={λo}, d={do/duo:.2%}')

## VISUALISATIONS
if True:
    resolution = 20
    kl_range, kr_range = np.linspace(0, 2, resolution), np.linspace(0, 2, resolution)
    kl_mesh, kr_mesh = np.meshgrid(kl_range, kr_range)

    # automatic vectorisation
    V_grad = V_grad_func(kl_mesh, kr_mesh)
    V_val = V_func(kl_mesh, kr_mesh)

    # manual vectorisation because sympy fails to handle it
    D_val = []
    D_grad_l, D_grad_r = [], []
    for kri in kr_range:
        D_val.append([D_func(kli, kri) for kli in kl_range])
        D_grad_l0, D_grad_r0 = [], []
        for kli in kl_range:
            klg, krg = D_grad_func(kli, kri)
            D_grad_l0.append(klg); D_grad_r0.append(krg)
        D_grad_l.append(D_grad_l0); D_grad_r.append(D_grad_r0)

    # contour = plt.contourf(kl_range, kr_range, V_val, 500, cmap='turbo')
    # plt.quiver(kl_mesh, kr_mesh, V_grad[0], V_grad[1])
    # cbar = plt.colorbar(contour)
    # cbar.set_label('Volume of fuselage')
    # plt.savefig('data/6_V_contour.png', dpi=600, transparent=True)
```

```
# plt.clf()

contour = plt.contourf(kl_range, kr_range, D_val, 500, cmap='turbo')
plt.contour(kl_range, kr_range, V_val, [V_TARGET], cmap='turbo')
plt.quiver(kl_mesh, kr_mesh, D_grad_l, D_grad_r)
cbar = plt.colorbar(contour)
cbar.set_label('Fuselage aerodynamic drag')
plt.xlabel('$k_l$')
plt.ylabel('$k_r$')
plt.savefig('data/6_D_countour.png', dpi=600, transparent=True)
```