Project Name: Canoe Design Program

**Project Description:**

Canoe Design Program (CDP) is design software that aids the design process of the canoe. The CDP will take mathematical inputs such as length, width, depth, the slope of the curve, density, and crew weight to calculate the volume, buoyancy, center of gravity, and waterline, determining the flowability of the canoe. Then, the CDP will generate an STL (Standard Triangle Language) file that is water-tight, 3D printable, and testable in CFD (Computational fluid dynamics) software.

Both calculating and model generating processes can be done manually by using a calculator and CAD software. However, manual methods often confront tremendous efficiency counter-backs considering the voluminous data, a large number of dimensions, and the complexity of functions. For example, manual calculations tend to consume more time when altering parameters and specifications to calculate and construct new models.

CDP automates this process, heavily increasing efficiency, and ensuring accuracy.

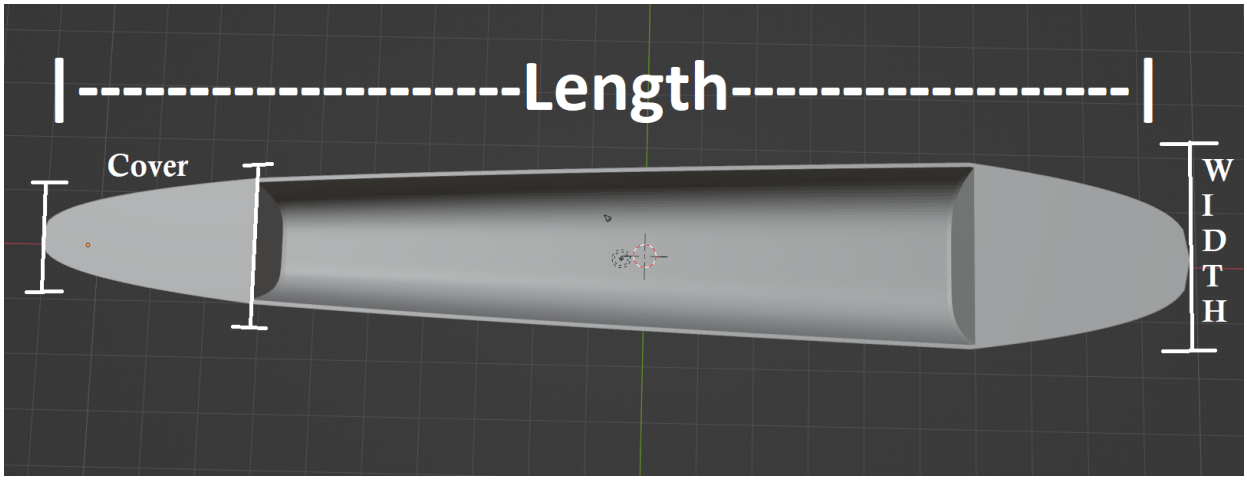
The program includes following functions and characteristics:

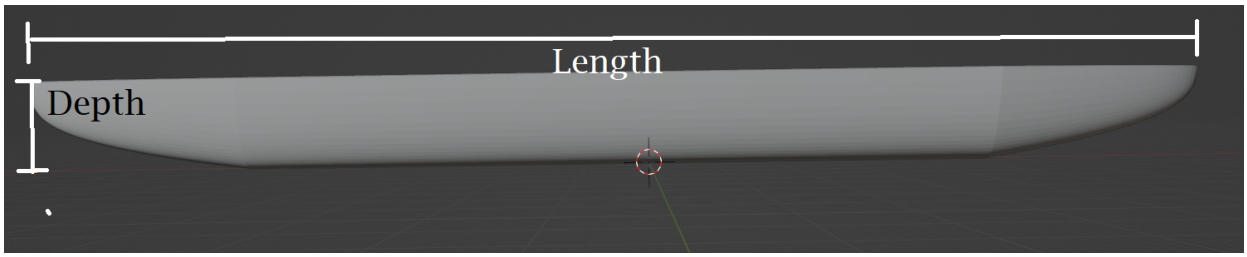
1. User input table for taking data
2. Save and open the file
3. Calculate the spec and generate a model of the canoe
4. Design optimization

CDP can design six types of hulls.

the software here is blender, only used for displaying the model

First, let's get familiar with the terms of the canoe:



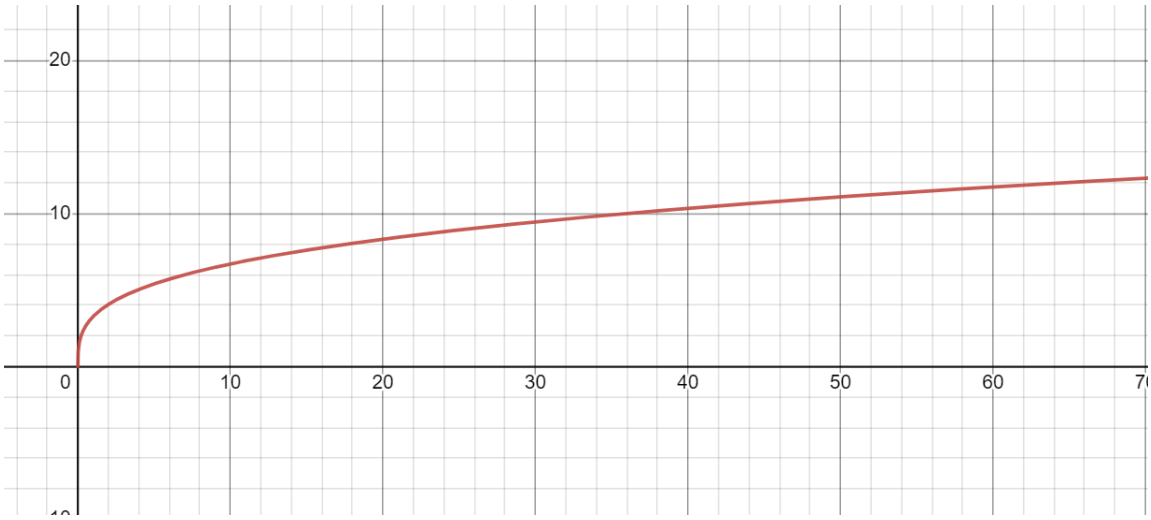




**Important to understand: <u>We must consider the canoe as a 3D object</u>**

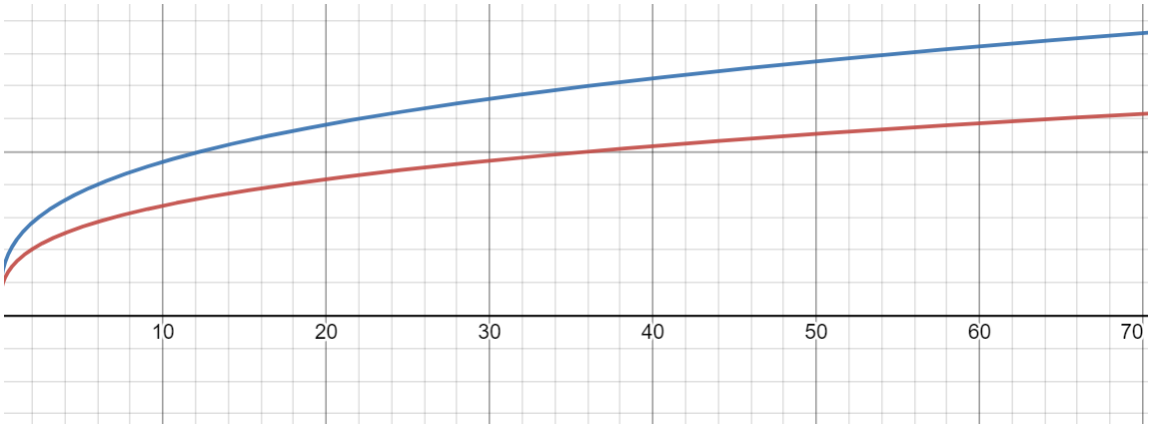
**In the picture above, we can observe that the hull of the canoe doesn't have a fixed geometrical shape, but is composed of two irregular objects. It is because the depth and width is governed by functions: (w = width, sw = semiwidth, l = length, d = depth, ew = exponent of width, ed = exponent of deception, el = exponent of length).**

1. $w(x) = sw \cdot (\frac{x}{l})^{ew}$ . x represents the index on length. By this, we know the sw at specific points in length. As figure displayed: (website is https://www.desmos.com/, only

use for display)

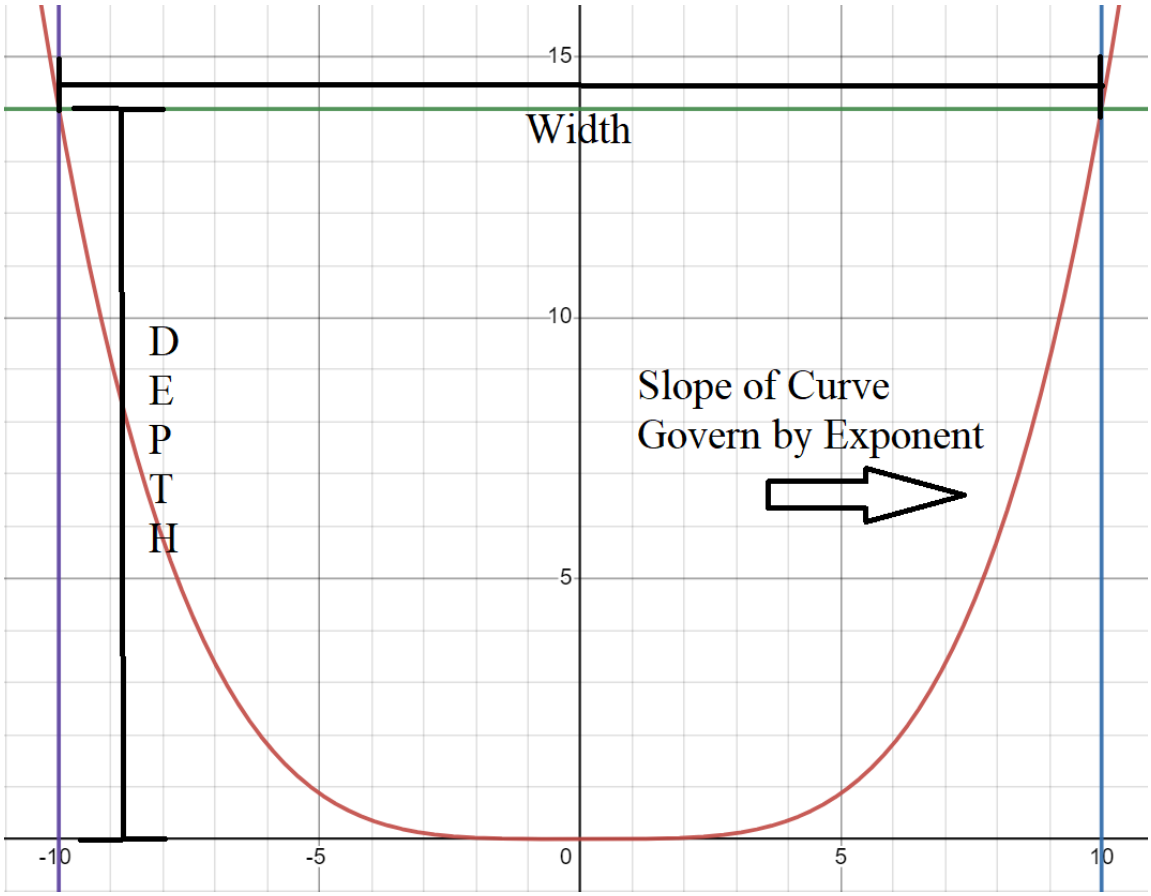



2. d(x) = $d \cdot (\frac{x}{l})^{ed}$ x represents the index on length. By this, we know the d at specific points in length. As figure displayed (blue line):




**3.** Once we know the depth and width, meaning that we know the width and depth at any point of the length axis. Thus, we can construct the "cross-section." Consider the 3D object canoe consists of an infinite amount of cross-section. Thus, any slice of this canoe

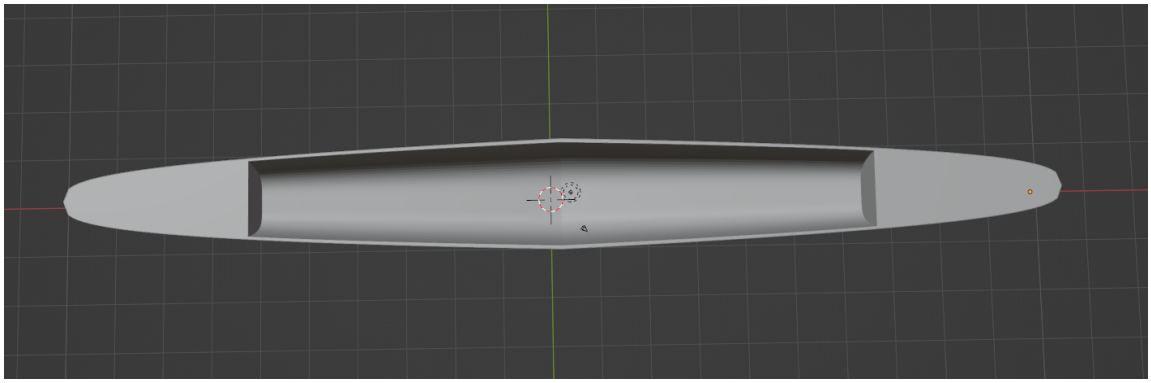
should be a 2D graph like this:



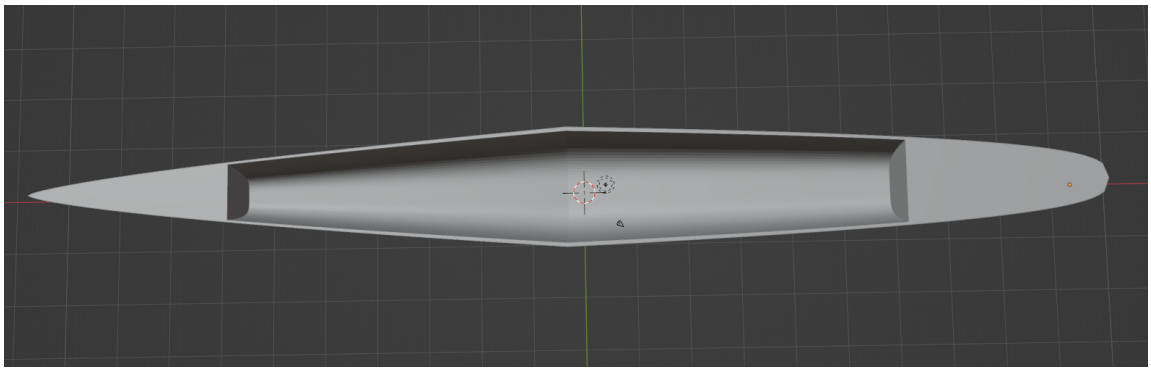

Thus, we know the formula that govern the shape of the correction is:

c(x) = $d \cdot (\frac{x}{sw})^{ex}$ , x represents the indexing on the width axis.

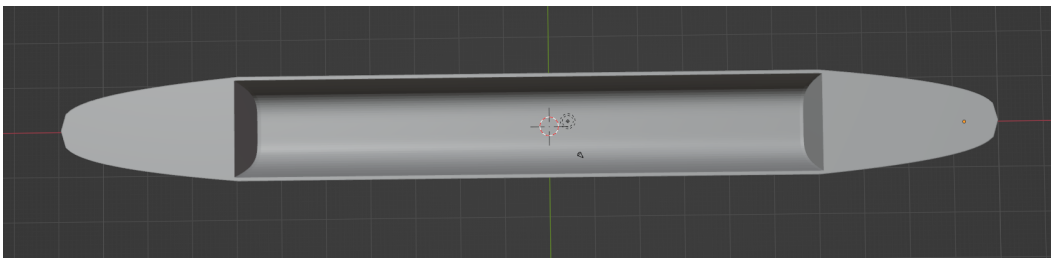
- Symmetric hull (figure 1): also known as one body hull, users only need to input one set of data, the canoe has axis symmetricity if we consider the midpoint of the canoe as its axis.



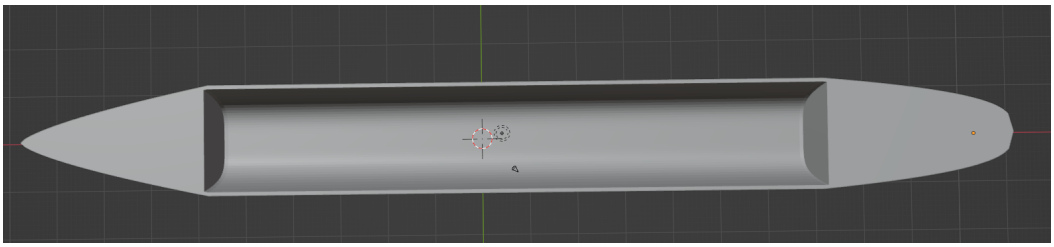
- Two body hulls: users need to input two sets of data, the canoe can be symmetric or not. Consider Long Short Hull as an asymmetric hull of two body hulls. (Figure 2)



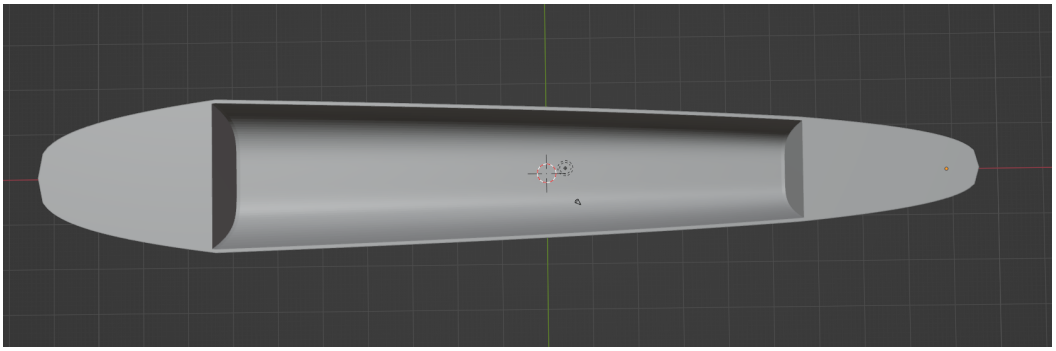
- Three-body hull: users need to input three sets of data, and the canoe can be symmetric. Consider front as section 1, middle as section 2, and end as section 3.
    - Three body hull _symmetric_ constant：section 1,3 has changing width and depth, as one can observe in the above picture, which makes them become the bow (The front of a boat or ship. Compared with sections 1 and 3, section 2 is constant, meaning it has an unchanging width and depth. Consider the figure below:



- Three body hull_ asymmetric_ constant:



- Three body hull_ asymmetric: section 2 has changing width:



**Structural Plan**:

To make the program able to execute the functions mentioned above, the project will be constructed as follows:

1. The folder entitled "code" will contain two folders, "source" and "test_src." The source folder will store the python code that supports the operating of the software, including GUI, calculating algorithm, model generating algorithms, etc. the test_src folder will contain a python code file that is used for testing experimental functions, saving time for

directly testing in the source files, and easy for debugging. The test_src folder will be excluded from the formal edition.

2. The folder entitled "Asset" will contain four source folders. They are ModelFile, Picture, startSetup, and TestProfile.
    - ModelFile folder is used to store the STL model generated by CDP.
    - Picture folder stored pictured that will be invoked by the code files and displayed as GUI in the CDP.
    - startSetup folders contain important files (txt) that store the information that determines how the CDP will be initiated. CDP code file will read these files for each time of its start. The CDP has two modes, normal mode and debug mode. The normal mode will operate as **Project Description** describes, and the debug mode will not open the GUI interface of the CDP but interact with the user through the terminal.

The current startup fill-in startSetup is entitled as setUpinformation.txt. The text stored a string that can be read as a dict.

Four variables that govern the operation of debug mode,

1. "isDebug": (int), 0 means start as normal mode, and one means start as debug mode.
2. "Modell": (int), 0 means debug mode doesn't calculate the Model Of the canoe, one means do.
3. "VolumeCal": (int), 0 means debug mode doesn't calculate the data of the canoe; one means do.

4. "BothMode": (int), 0 means debug mode doesn't calculate the data/model of the canoe; one means do both.

- TestProfile stored six text files that contain a string that can be read as a list, and the list contains a dict that describes the spec of the canoe and a list that describes other information about the canoe. The debug mode will be able to read these files and can then directly calculate and generate the canoe without user input.

1. Code/ source folder: There will be 7 py files, which are: MainGUI.py, Calculation.py, CanoeDataBase.py, DataCalculation.py, HealthCheck.py, ModelCalculation.py, DeignOptimzation.py.
    a. MainGUI.py: The GUI of the CDP is constructed in this file. ***Very important:*** **This file is the major file of CDP; all the processing commands are called from here, so we should start this file first.** The file contains several classes. They are MainGUI_Base(), MainGUI_Init(), MainGUI_CreatNEW, MainGUI_Open, and MainGUI_DOP.
        - MainGUI_Base() is a major class that will call other classes to construct the animation. And the reading of the start file is from here. And it constructs the **Window** of the CDP
        - MainGUI_Init() constructs the base page **Frame object (MainGUI_Init_MainFrame)** of the GUI; the user will choose which function of the CDP to operate. Meaning the base page Frame will be replaced by other frames that contain other animations.
        - MainGUI_CreateNEW replaced the Frame object (**MainGUI_Init_MainFrame**) with its own frame object, allowing the

user to see them in the window. There are mainly three pages (Frame object) of MainGUI_CreateNEW; users can use Button to traverse through these pages, including ahead, backward, and return to the MainGUI_Init.

i. These frame objects are MainGUI_InputTable, MainGUI_InputTable_Two, and MainGUI_DisplayTable_Three. The "inputTable" contains animations that require the user to input data. The display table will display the data to the user.

- MainGUI_Open: MainGUI_Open requires the user to pick a data file of CDP that was previously saved by the user and allow the user to configure the data on it continually.
- MainGUI_DOP: will require the user to pick the data file of CDP that was previously saved by the user and ask the user to determine the range of variables that the user wants to optimize. Then call the optimization algorithms and then display them to the user.

a. CanoeDataBase.py: CanoeDataBase.py contains one class, CanoeDataBase. It will be called MainGUI.py and HealthCheck.py. The CanoeDataBase class contains two important instance attributes, self.SDD = SectionDataDict and self.HDD = HullDataList. The SDD is a dict object that stores the section number as a key, and spec information such as Length, depth, width, and exponents, as value. The HDD is a list object that contains other specs such as Density, coverlength, crew weight, and thickness of canoe.

The CanoeDataBase class is used to construct a CDD (Canoe Data Design) instance, where its attribute can be invoked by other python files such as Calculation.py. The construction of the CDD will be called from MainGUI.py, where it takes the user input. And the HealthCheck.py where directly reads the test profiles and constructs the CDD.

1. Calculation.py: Calculation.py is the major file of the DataCalculation.py, ModelCalculation.py, and DeignOptimzation.py. The Calculation.py contains a class entitled calculation; it takes a CDD instance as an argument. The class Calculation is the parent class of the following classes: DataCalculation class from DataCalculation.py, ModelCalculation class from ModelCalculation.py, and DeignOptimzation class from DesignOptimzation.py.

   Instances attribute:

   - **From HDD**
   - self.Length : List[float]
   - self.Width : List[float]
   - self.SemiWidth : List[float]
   - self.Depth : List[float]
   - self.ECurveF : List[float]
   - self.EWidthF: List[float]
   - self.EDepthF: List[float]
   - **From SDL**
   - self.Density: float
   - self.Thickness: float

- self.CoverLength: float
- self.CrewWeight: float
- **Class Defined**
- self.WidthFList: List[lambda]
- self.WidthFList_Outside: List[lambda]
- self.DepthFList: List[lambda]
- self.DepthFList_Outside: List[lambda]
- self.B2: float
- self.B2_O: float
- self.B2_Diff: float

**Explanations**:

- If you don't understand the meaning of these attributes( Length, Width, Depth, Density, CoverLength, CrewWeight). Please contact me at changbax.andrew.cmu.edu.
- ECurveF is a list object that stores (**float**) the exponent of cross-section, EWidthF stores the exponent of width, and EDepthF stores the exponent of the Depth. The exponents are stored to be used in formula construction, which is the meaning of "F". E represents the "Exponent"

- Thickness is a floating object that describes the thickness of the canoe. Consider the pictures:


- WidthFlist, WidthFList_Outside, DepthFList and DepthFList_Outside are lists that store lambda objects. We consider the Outside as the outer side of the canoe that its width, length, depth is based on the user input. e.g : outside_width = width + Thickness.

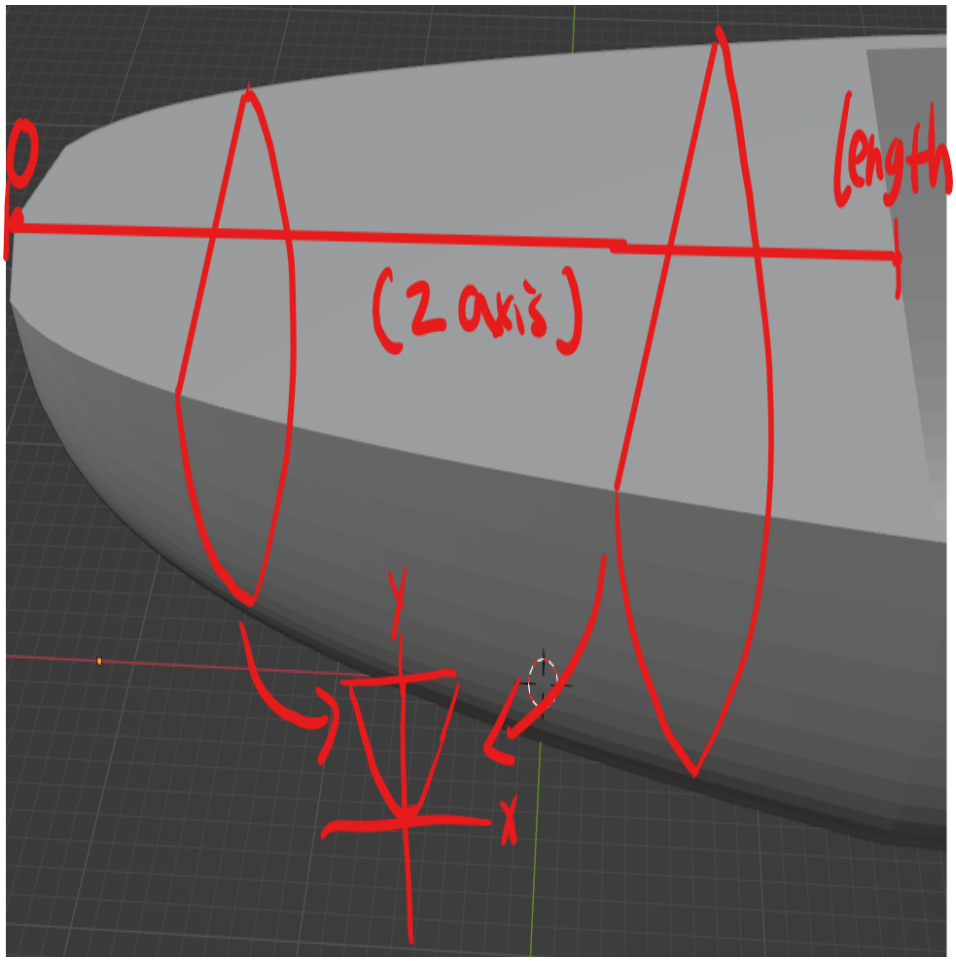
  The lambda objects should be considered as mathematical formulas, e.g: lambda x :self.Depth[index] * (x / self.Length[index]) ** self.EDepthF[index] is **equivalence** to $d \cdot (\frac{x}{l})^{ed}$.

- Self.B2 is specifically used for a special hull type: Three body hulls_ asymmetric. Because in this case, the middle section has a changing width, meaning it is a partition that is specially chosen for pairing the front

section and end section. To understand this, we must understand how to construct the mathematical expression of the canoe's section.

1.

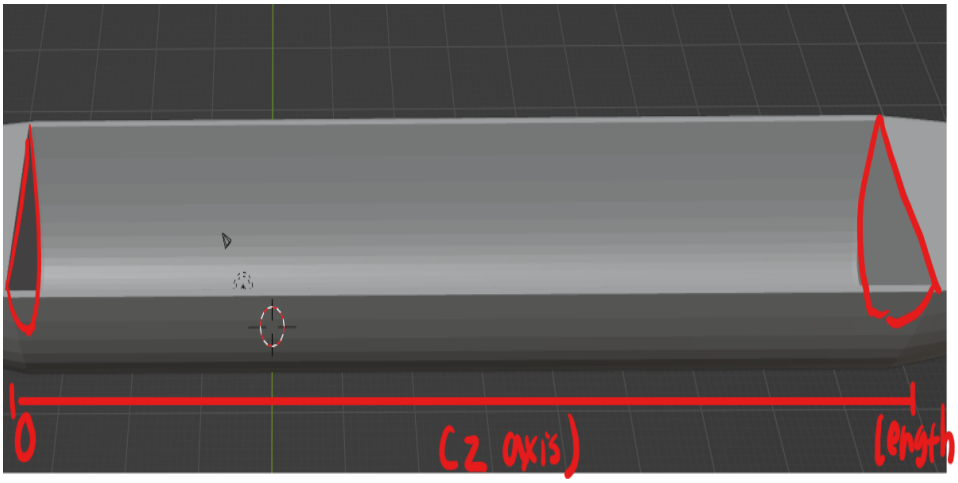
- Each section is a 3D object, and we consider the length as its Z axis, width as its X axis, and depth as its Y axis. We understand how to construct the "bow" in the previous explanation (consisting of an infinite slice of 2D cross sections that continuously change due to the indexing on the Z axis. ). Thus, we know a Bow starts from 0 on the Z axis and ends at length on the Z axis. And all of them are based on outside input. Consider the picture below:




If we sliced such a bow, we should observe a “point” at Z = 0 and a 2D shape (cross section) at Z = length, where this shape should be the largest compared with all other slices.

**2.**

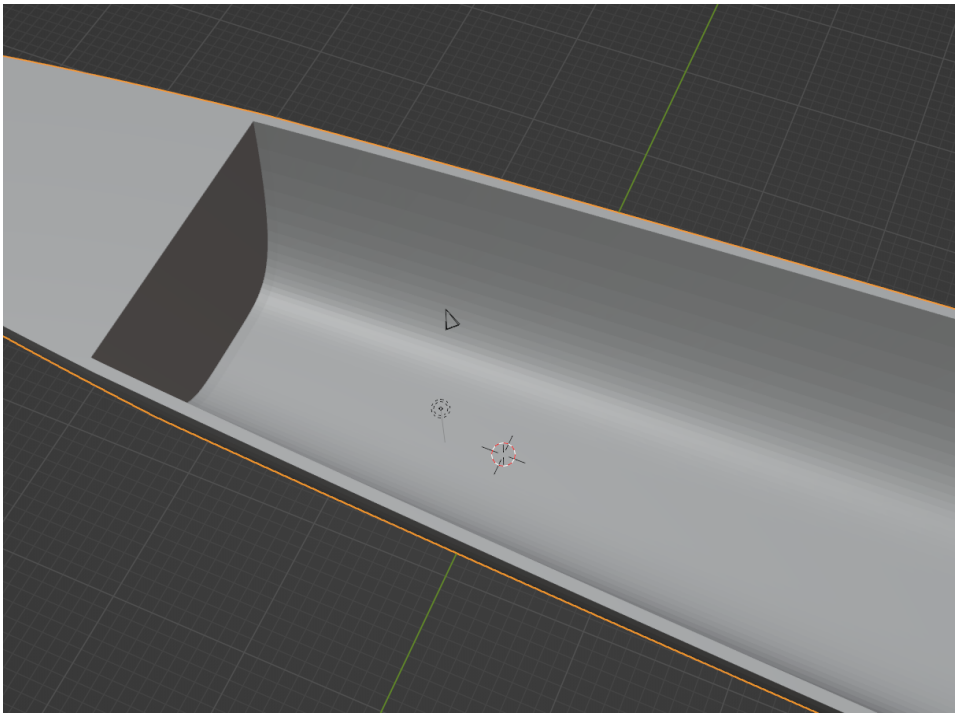
To construct a “constant” section of the middle is easy, we know for a fact the “constant” starts at 0 on the Z axis and ends at length on the Z axis. Consider the picture below:



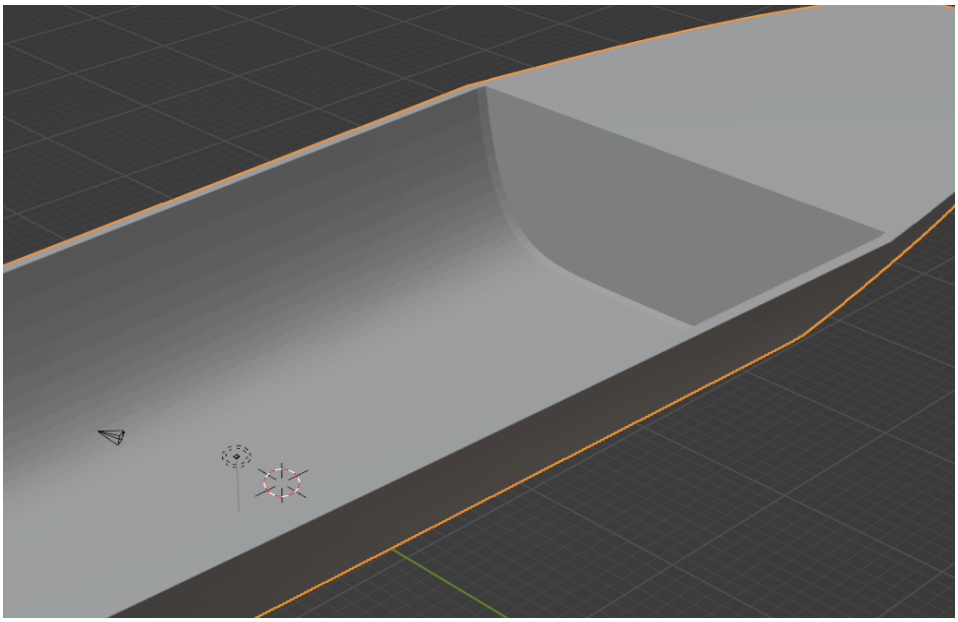

If we sliced such a 3D object, we will observe infinite slices of 2D cross-section shapes that are equal to each other.

**3.**

But for this case, the middle section is not constant anymore, meaning we can't directly consider the section from 0 to its length in the Z axis. And we can't construct such a "Bow" style object based on the current datainput.



Considering the picture above, we can observe a fact: the middle section has the same 2D shape at its “start point” of the Z axis. Meaning the user input of the width of this section is actually defining the width of the shape at the end of the Z axis. Consider the picture below:



**(consider w1(x) as the width function of the middle, w(x) as the width function of the front)**

Thus, we must find a width curve that when (Z = start point (B)), w2(B) = w1(front length). When (Z = middle of length) w2(middle's length + B) = width of middle.

("2" represent the middle section, "1" represent the front section, ex here represents the exponent of width)

Consider the mathematical expression of the width function of the middle:

$$sw_2\left(\frac{x}{b_2+l_2}\right)^{ex_2}$$

.

Thus, based on the above, it is easy to know that: b2 (start point) =

$$b_2=\frac{\left(\left(\frac{sw_1}{sw_2}\right)^{\frac{1}{ew_2}}\cdot l_2\right)}{1-\left(\frac{sw_1}{sw_2}\right)^{\frac{1}{ew_2}}}$$

Because we can see that $sw_1(\frac{l_1}{l_1})^{ew1} = sw_2(\frac{b_2}{b_2+l_2})^{ew_2}$.

B2_O is just the outside version of B2. B2_diff is just the difference between B2_O, and B2.

The Calculation class only defines instance attributes and defines many helper functions that are extremely useful for its child classes, including the SignData() that will be invoked by its child class.

SignData() function will sign the variables from the CDD object directly to the parameters declared in the __init__ function of the class Calculation. Then it will call the SignFunction_Main, where it will identify the type of hull and sign the correct width function (lambda object), depth function (lambda object) to the WidthFlist, WidthFList_Outside, DepthFList, and DepthFList_Outside.

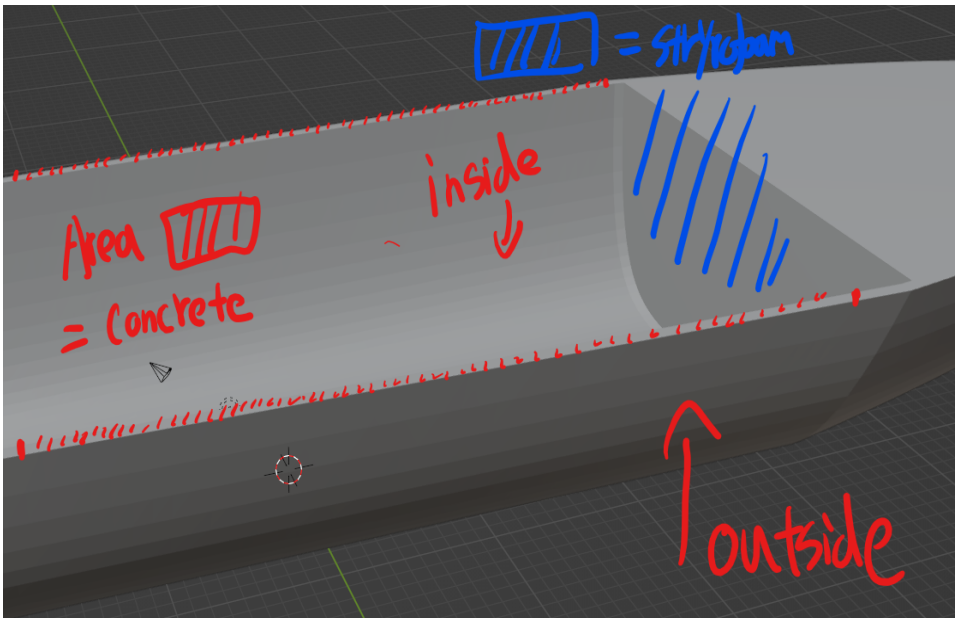
There are a series of BuildLambda functions, which are specifically for constructing different formulas.

a. DataCalculation.py contains a class entitled DataCalculation and an important function specific for the calculation of data of the canoe: CanoeDataCalculation. The class DataCalculation succeeds the attributes of the Calculation class and then assigns the value to them by calling the SignData() in its constructor. And defined several attributes of its class, which are:
    - self.Volume_Inside: float
    - self.Volume_Outside: float
    - self.Volume_Styrofoam: float

- self.Volume_Concrete: float
- 
- self.Buoyancy: float
- self.Buoyancy_Submerge: float
- 
- self.CanoeWeigh: float
- self.TotalWeight: float
- 
- self.SubmergeBoolean: Boolean = False
- self.FlowBoolean: Boolean = False

**Explanation**:

- Volume_Concrete represents the volume of the Solid (concrete) part of the canoe. Its size is defined by the Thickness, and equal to the Volume_Outside - Volume_Inside. Consider the picture below:



- Volume_Styrofoam is the part that is encircled by the cover. Consider the picture above.
- Submerge represents a condition when the canoe is fully submerged underwater. In this case, the submergeBoolean map “canoe come back to surface from the water without outside force” as True, and False as vice versa.
- Total Weight represents the canoe's weight plus the crew weight.

The canoeDataCalculation function is the main base for calculating the data of the canoe, it will call functions that sign those attributes and determine the flowability, and submerge ability of the canoe.

b. ModelCalculation.py contains a class entitled ModelCalculation. It will also succeed with the attributes of the Calculation class and define it by calling the SignData(). It includes serval special attributes for its class:
    - self.Inside_LengthList: List
    - self.Outside_LengthList: List
    - self.CoverLength_end: Float = sum(self.Length) - self.CoverLength
    - self.Inside_Length: List
    - self.Outside_Length: List

    **Explanation**:

- Inside_LengthList or Outside_LengthList is a 2D list, each list of it maps to a finite set that is defined as [{0} ∩ self.Length[i]]]. E.g if self.Length = [3,4,5]. Then Inside_LengthList = [[0,1,2,3],[0.1,2,3,4],[0,1,2,3,4,5]] There two length lists are especially assigned for the model generated.
- Inside_Length or Outside_Length is a list that stores the float values that map to the Z value of the canoe. E.g if self.Length = [3,4,5]. Then Length = [0,1,2,3,4,5,6,7,8,9,10,11,12] that |Length| = $sum(self.Length)$ + 1

It will call the LengthIndexGenerate() that directly signs the value for the LengthList and Length.

The ModelCalculation class defined an important function entitled "Model_Generate" that will call a series of functions that eventually generate enough variables that can be used to construct the STL model. We shull discuss this part in the algorithm plan.

c. HealthCheck.py contains two classes that are entitled as DebugBase and HealthCheckBase. The DebugBase will be called to construct an instance if the "isDebug " of the startUp information file is 1. The DebugBase will interact with the user through the terminal to take the string input of the user and operate as the user state. Users can use it to calculate the spec and model of the canoe that the data is read from the test profiles in the TestProfile folder of the Asset folder.

The HealthCheckBase will be called by the CanoeDataBase class, a calculation

class to check if the user input data is “legal” or not. Avoiding the crush of the program.

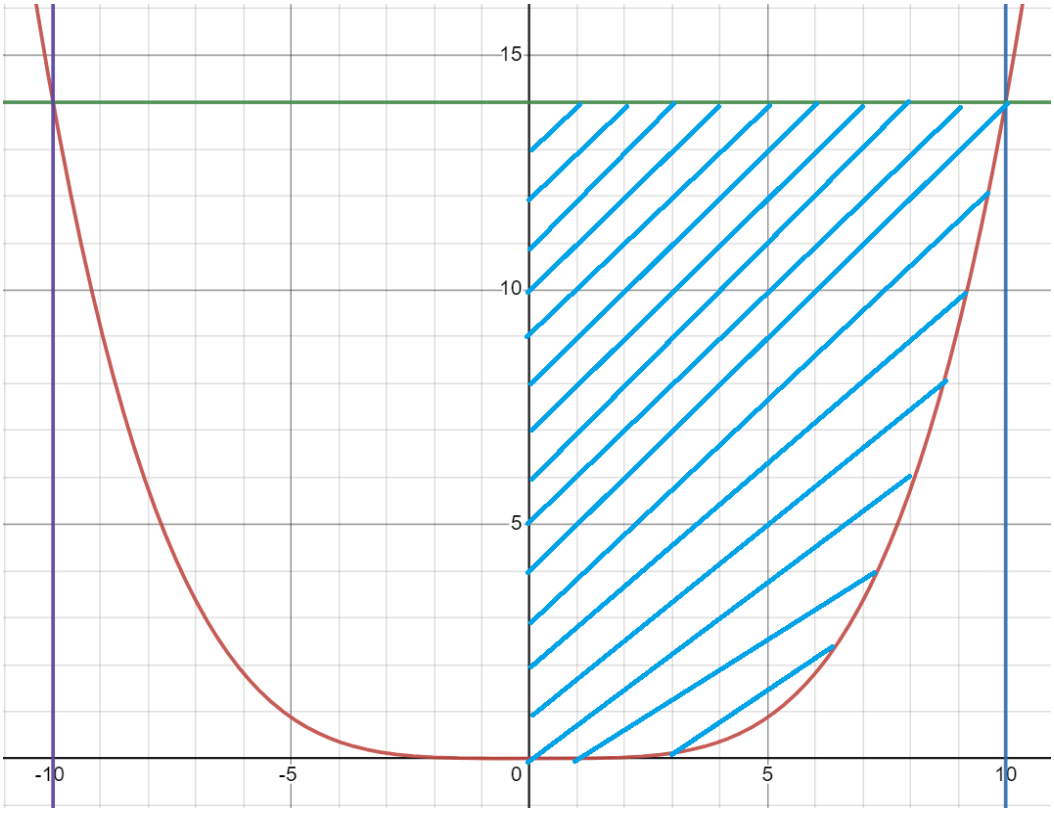
d. DeignOptimzation.py contains a class entitled DeignOptimzation that will basically loop through the possible composition of specs of the canoe under the user-defined range and variable range. E.g: user-defined the a to be 10, b to be 20, c to be DesignOptimization, and c’s range is from 1 to 2. Then the DeignOptimazation class will loop through the possible Combination of (a=10,b=20,c=1) and (a=10,b=20,c=2) to see which is better. And return the better one as the optimized design.

Algorithm plan: Several important algorithms composed the core function of the programs.

1. DataCalculation algorithms:
    a. Hull Volume calculation: By reading the structural plan, we know that a canoe object is a 3D object that has at least two sections of the “Bow” shape section, which have a changing width and depth based on the index on length. Thus we must use calculus to calculate the volume of such an object. If we sliced the canoe, we know that each slice of it is a 2D cross-section-shaped object or the shape of an arc.

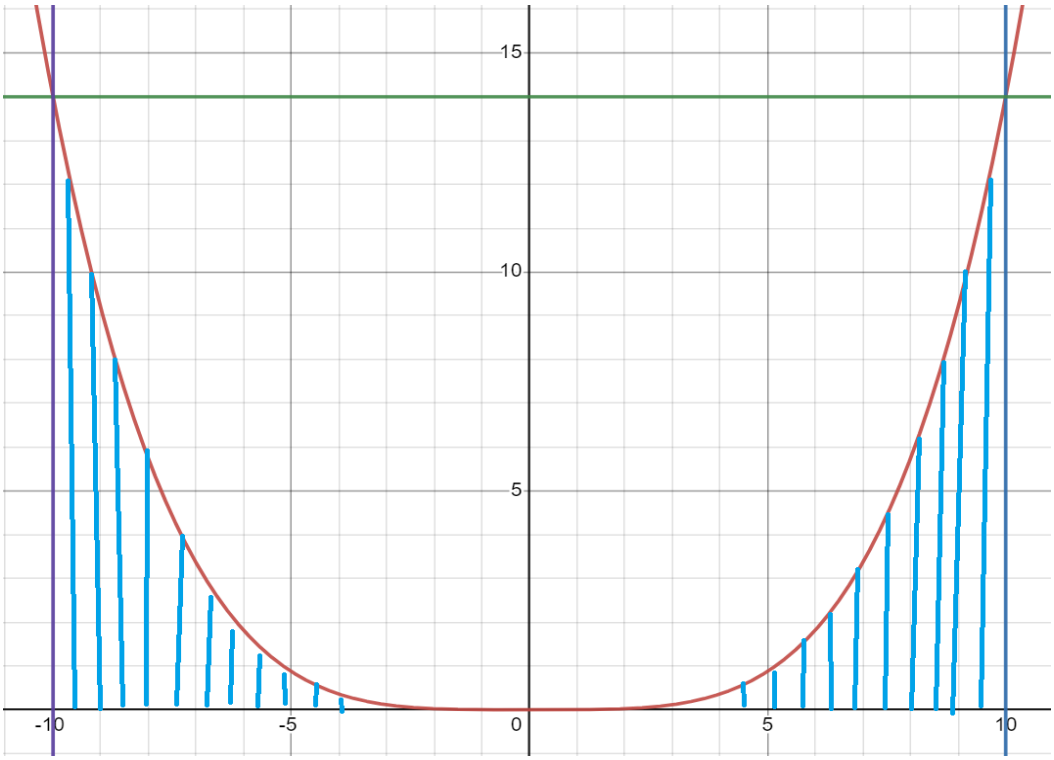
    .($sw = semi - width$, $d = depth$, $ex = exponent\ of\ cross - section$)

    We can use $d \cdot (\frac{x}{sw})^{ex}$ to represent the semi-part of the area. Consider the picture below:

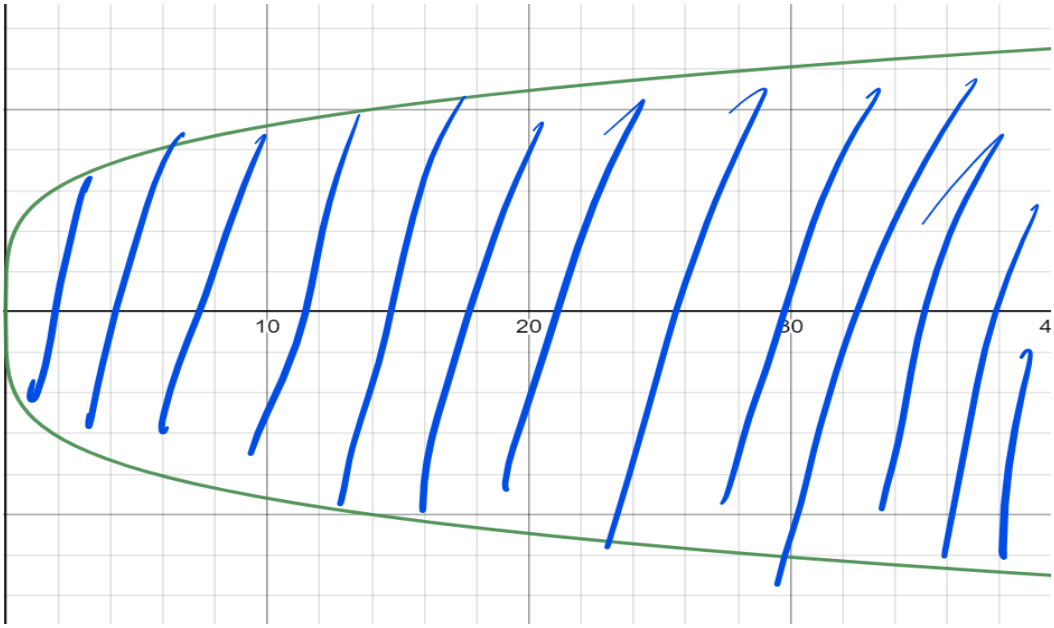



However, when we try to get the area of such a curve and calculate the integral of such a function. $2 \cdot \int_{0}^{sw} d \cdot (\frac{x}{sw})^{ex} \, dx$ , it is actually returning the area under the

area. Consider the picture below:



Thus we want to inverse the function to $sw \cdot (\frac{x}{d})^{\frac{1}{ex}}$ such that the integral of it would be . $2 \cdot \int_0^{depth} sw \cdot (\frac{x}{d})^{\frac{1}{ex}} dx$. Consider the picture below:




Thus we know how to calculate the area of a single cross section of the canoe at any Z-index of the canoe. Because by the width function and depth function:

$w(x) = sw \cdot (\frac{x}{l})^{ew}$ and $d(x) = d \cdot (\frac{x}{l})^{ed}$

We can already calculate the volume of the constant sections where the width and depth is not changed at any z index. So its volume would is simply be

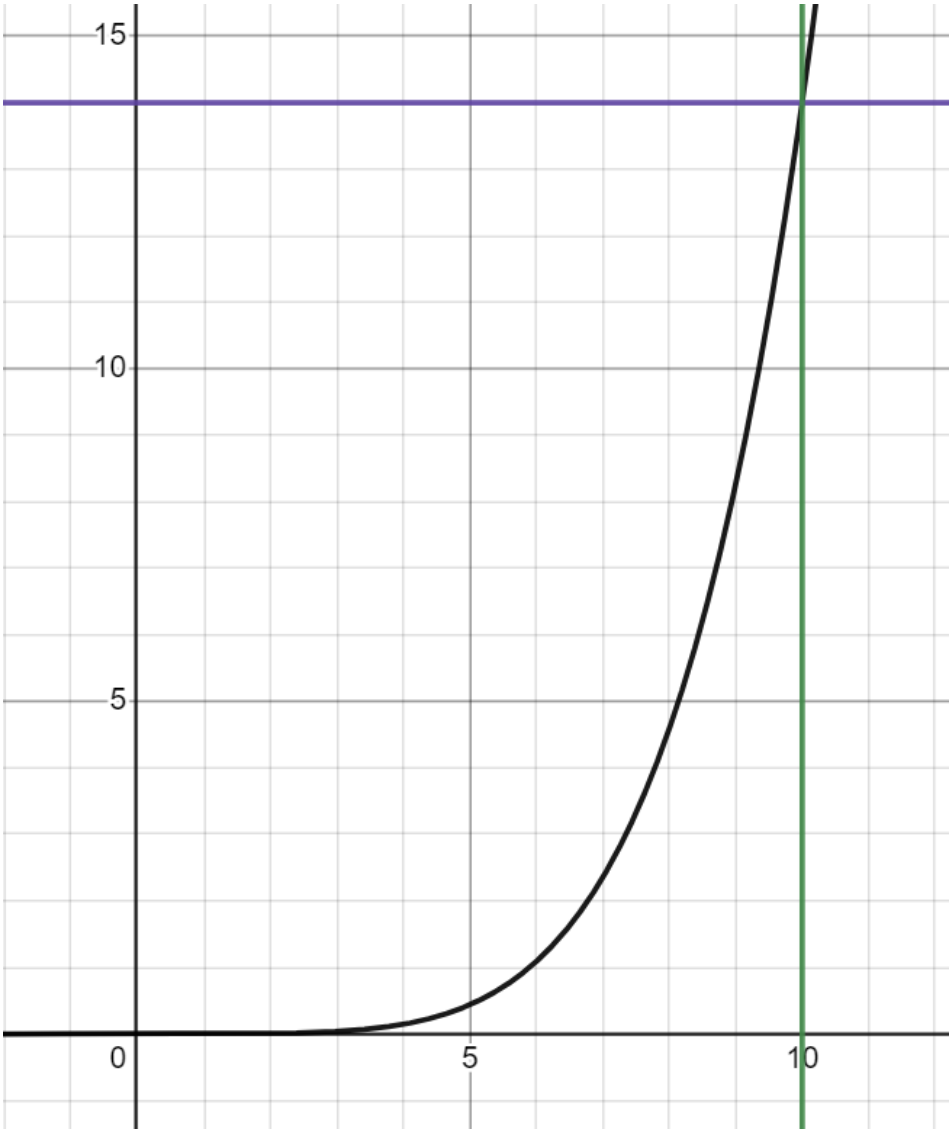
$l \cdot 2 \cdot \int_0^{depth} sw \cdot (\frac{x}{d})^{\frac{1}{ex}} dx.$

However, for the “bow” shape object, we will need to construct a new formula to solve it. Which is

$$2 \cdot \frac{ex}{ex+1} \int_0^l w(x) \cdot d(x)\, dx \;=\; 2 \cdot \frac{ex}{ex+1} \int_0^l sw \cdot \left(\frac{x}{l}\right)^{ew} \cdot d \cdot \left(\frac{x}{l}\right)^{ed} dx$$ (*1.volume formula*)

Let us first consider the $\int_0^l sw \cdot \left(\frac{x}{l}\right)^{ew} \cdot d \cdot \left(\frac{x}{l}\right)^{ed} dx$What does this mean? Let us first consider each slice of it (slicing by Z). We know $sw \cdot \left(\frac{x}{l}\right)^{ew}$ result in a width, and $d \cdot \left(\frac{x}{l}\right)^{ed}$ result in a depth. What does width time depth look like in

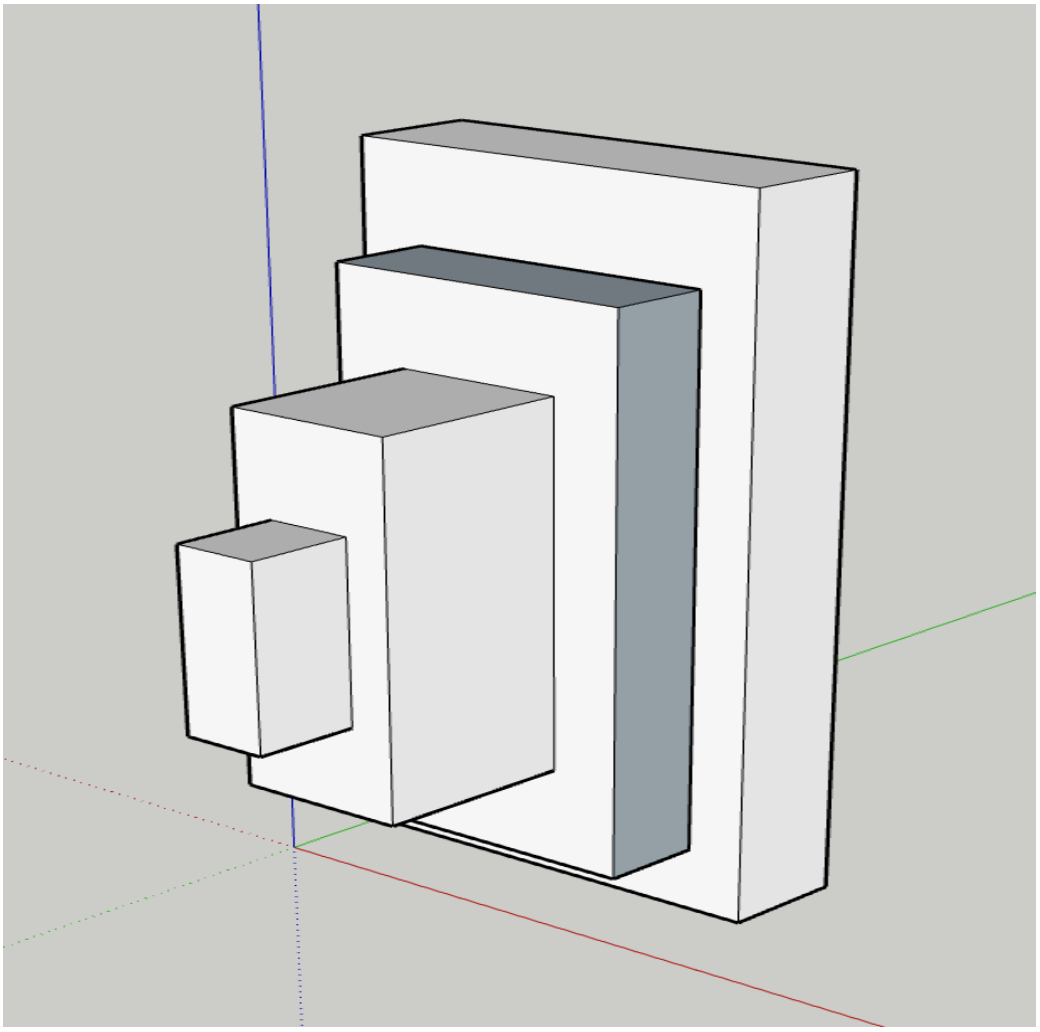
2D shape? It is a rectangle. Consider the picture below:




We can see that our calculus formula for the area of the correct section is just the area of the rectangle minus the area under the curve (result by integral). Then let us consider this in 3D:

Considering a 3D object, when we slice it by the Z axis, each slice will be a rectangle. Thus we know that what object are we describe through the integral

$\int_0^l sw \cdot (\frac{x}{l})^{ew} \cdot d \cdot (\frac{x}{l})^{ed} dx$. Consider the picture below:

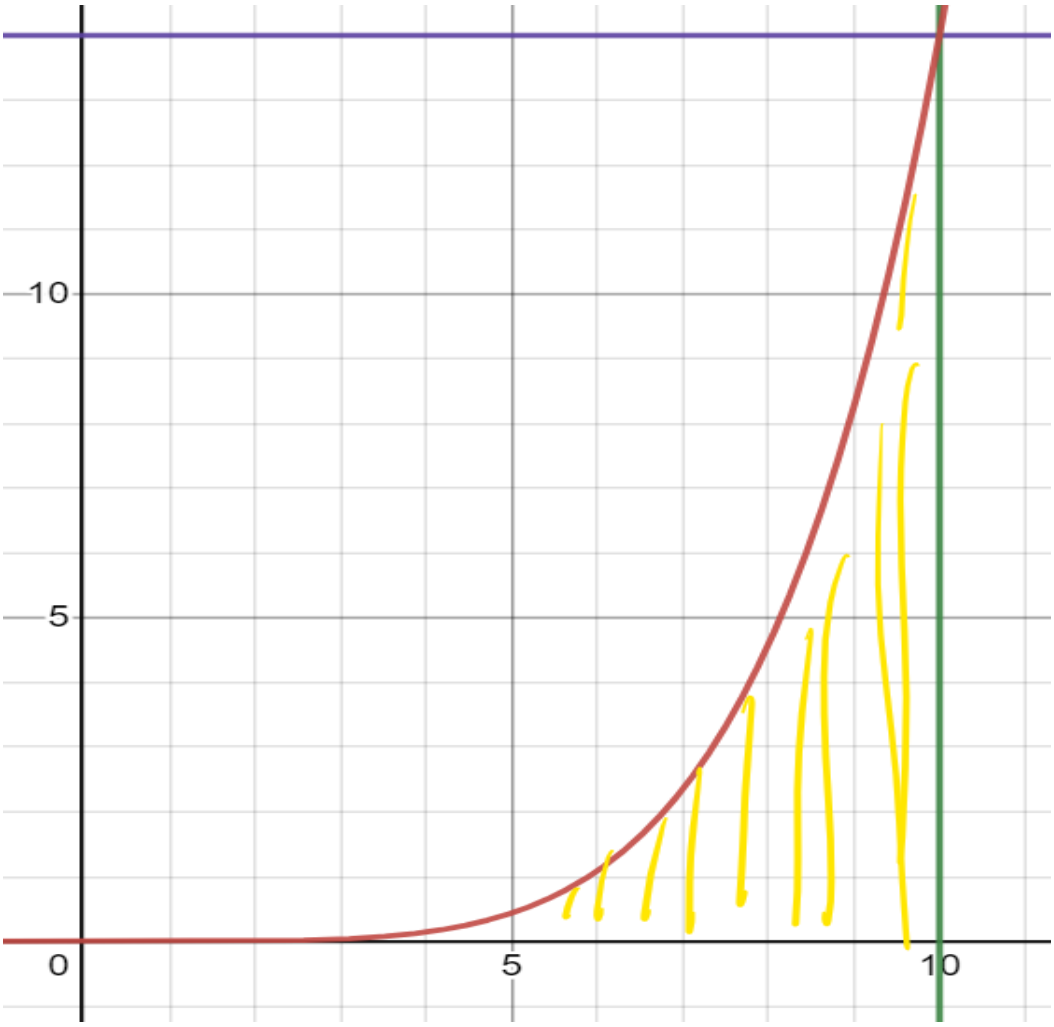


Notice that the object should still remain the property of the curve when we observe it from the side and top.

Considering the previous picture of the rectangle, we can see the arc on it, and we know that the area of such an arc is basically the area of the rectangle minus the area under the curve. Thus, we know that the volume of the canoe is basically the

volume of the “rectangle object” minus the “area under the curve.” The mathematical approach is to calculate the proportion that the canoe takes in the object of the rectangle.

Let's consider a relatively simple example first: Assume there is formula that describe a curve: $14 \cdot (\frac{x}{10})^5$. Its graph would be:



The red line is the curve, the purple and green line simulate the rectangle that contains this curve. Yellow shaded area describe the integral of the formula $14 \cdot (\frac{x}{10})^5$. Which is $\int_0^{10} 14 \cdot (\frac{x}{10})^5 dx$. The value would be 23.3333333333. And the true area that we want to have is the 14*10 - $\int_0^{10} 14 \cdot (\frac{x}{10})^5 dx$, which is 116.666666667.

Now let us solve this integral, which is $\frac{14}{6} \cdot (\frac{x}{10})^6$ when we input the upper limit, we will get 23.3333333333. Thus, we know that $14 \cdot 10 - \frac{14 \cdot 10}{6} = 116.666666667$. Does it look familiar? We now know that the proportion of area under curve to the area of the rectangle is the $\frac{1}{(exponent\ of\ curve + 1)}$. Considering that we want to get the area of the crossetion, which is the remaining area. We can simply do some math that $\frac{exponent\ of\ curve + 1 - 1}{(exponent\ of\ curve + 1)}$ is the proportion of remaining area in the rectangle, which is $\frac{exponent\ of\ curve}{(exponent\ of\ curve + 1)}$

By understanding this 2D example, we should know how to approach the 3D. Which is exactly the formula: $2 \cdot \frac{ex}{ex+1} \int_0^l w(x) \cdot d(x)\, dx$

Thus we can calculate the volume of the canoe.

Once we finish the construction of a mathematical formula, we can implement it in the code. As I previously satiated the usage and the assigning of the WidthFList, and DepthFList, we can call the function of Sign_CurveFormula to construct a new lambda object which return the: Lambda x: WidthFList[index](x) * DepthFList[index](x). Thus we can use the integral function of the Scipy, which is quad(CurveFormulaList[index], 0, self.Length[index]). And it maps the mathematical formula (1. volume formula) we constructed above.

The algorithm of the calculation of the volume of the canoe is basically to loop through the WidthFList and DepthFList, which are previously assigned for each section, and then use it to construct a new lambda object and then calculate its integral to get the volume of each section. Then we add up each one to get the final volume of the canoe.

The difficult part is to let the code know which volume formula should be used based on the user input data, considering the constant section (middle) and inconstant section (middle) have different calculating methods.

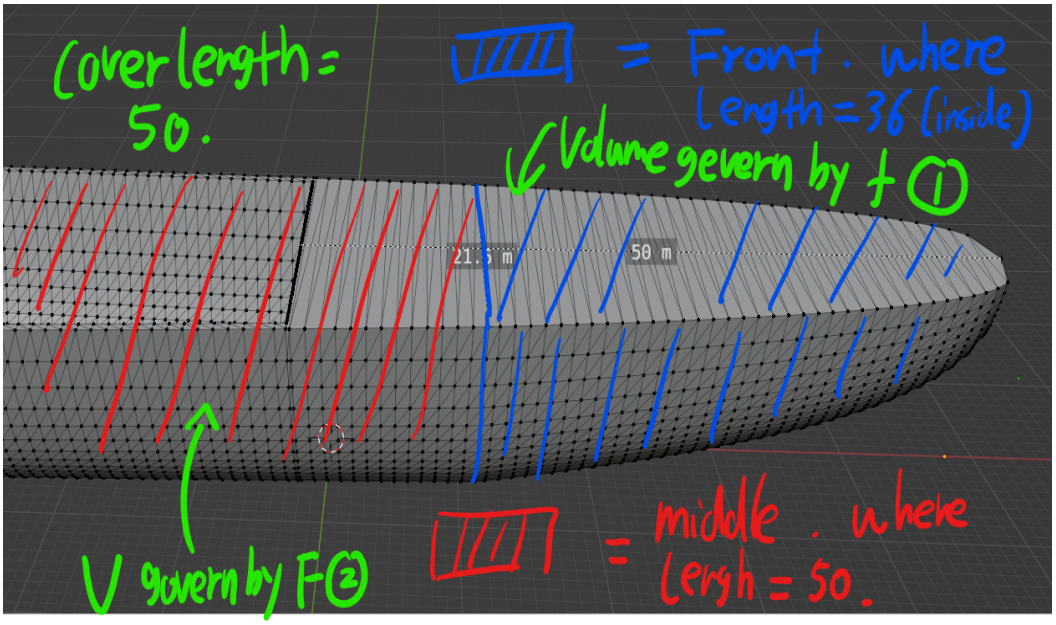
We can accomplish this part simply by applying the fact that the constant section is a constant section if and only if its exponent of width and exponent of depth is equal to 0, meaning that there aren’t any changes in either depth or width as

indexing through the Z axis. Thus, the program shull know which methods to use by simply using the if statement.

b. Concrete volume calculation. As previously stated, concrete represents the space between the outside hull and the inside, which can be easily calculated by getting the difference between the volume of the outside hull and the volume of the inside hull.

c. Styrofoam calculation. The calculation of the styrofoam can not be simply defined as:

$2 \cdot \frac{ex}{ex+1} \int_{0}^{coverlength} w(x) \cdot d(x)$ Because there exists a possibility that the cover length is larger than a single section's length, which required two separate calculations by using two formulas that maped two sections' volume. Consider the picture below:




We can observe that to calculate the volume of the cover , we need to use two volume formulas. Which is :

$$2 \cdot \frac{ex1}{ex1+1} \int_{0}^{l1} w1(x) \cdot d1(x)\, dx \; + \; 2 \cdot \frac{ex2}{ex2+1} \int_{0}^{coverlength - l1} w2(x) \cdot d2(x)\, dx$$

Thus, our priority task to identify that whether or not to use two formula (we don't use three formula because is impossible, because there are two cover, and the largest cover should be from 0 to sum(length)/2)

We can accomplish this by calling the LocateCover(), a static method. LocateCover() take two argument, the value of CoverLength and the length_list, which is defined as $\{0\} \cap \{x_i \in Length \wedge i \in [|Length|]: \sum_{k=0}^{i} x_i\}$. e.g: if length = [3,4,5]. Then the length list is equal to [0,3,7,12].

The LocateCover returns a 2D list that contains two values, an int, and a float. The int value is defined as the index of the list, where the program can use it to know which cover is in which section and invoke the right formula. The float value defined the length of the cover at that specific section, allowing the integral to take the correct upper limit. Consider the example below: if length is [36,120,36]. And the cover length is 50, the LocateCover Function will return a 2D list as : [[0,36.0],[1,14.0]]. If the cover length is 20, the LocateCover will return a 2D list as [[0,20.0]].

Thus, we can calculate the volume of styrofoam in these two conditions.

d. Calculation of other specs (Buoyancy, flowability, sumbergability). We shull understand some simple physics to understand the following.

Formula of calculating buoyancy:

$F_b = \rho \cdot g \cdot v$, $\rho$: $density$, $g$: $gravity$, $v$: $volume$.

Thus, we only need to input the volume (displacement) that is calculated as above, then we shull get the buoyancy of such an object. Considering our canoe will only be used on water, the density is set to 997 kg/m³, and the gravity is considered to be 9.8.

Thus, we can determine the buoyancy force of the object, then we can get its capability (mass) by simply dividing by 9.8, since $F = m \cdot a$ (F: $F_b$ , a :g). By comparing it with the mass (calculated by $volume\ of\ concrete \cdot density$) + crew weight .

When we get to the sumberability, we should only consider the volume of concrete and volume of styrofoam, considering they are the only displacement when underwater. And compare it with the mass of concrete.

e. Waterline Calculation: Waterline is a depth value that when we put a canoe into the water, the distance from the water surface to the height (depth) of the canoe. Thus, the core part is to calculate at what depth, the buoyancy of the canoe is

equal to the weight.

A naive approach would be looping through each depth value to determine at which depth the buoyancy equals weight. This approach is very time-consuming and unnecessary, thus we should use a mathematical approach to calculate the waterline.

We should construct such an integral formula that at the upper limit “a”, the buoyancy is equal to the weight. Thus we need to consider the 3D object on the Y axis instead of the Z axis as before. In which its mathematical expression is that:

$$\frac{1}{\exp\ \text{width}\ +1}\int_0^a (sw+t)\left(\frac{x}{d+t}\right)^{\frac{1}{\exp}}\cdot(l+t)\left(1-\left(\frac{x-d-t}{d+t}\right)^{\frac{1}{\exp\ depth}}\right)dx$$

$$\frac{1}{\exp\ \text{width}\ +1}\int_0^a (sw+t)\left(\frac{x}{d+t}\right)^{\frac{1}{\exp}}\cdot(l+t)\,dx$$

Top for front and rear, low for constant section

Add all sections up, and solve for when the sum equals total weight.

How was the function derived for bow and stern sections?

When sliced vertically, the area of each slice is determined by the width and the depth of the slice, both of which are defined by a function respective to the length from the origin. The function defining depth with respect to width determines how much area the slice takes within the rectangle formed by the width and depth in the plane of the slice. Then, integrate to obtain the volume

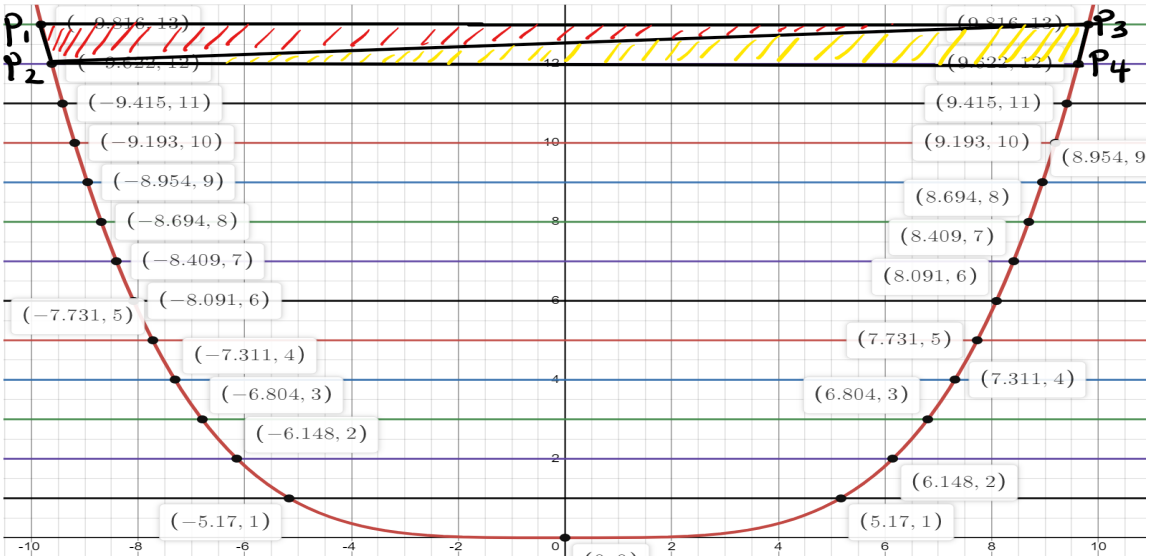
ModelCalculation: Even though we can use integral and calculus to help us to simulate the 3D object easily, we can't use those in the generating of STL. In which each coordinate of the Canoe needs to be manually calculated and assigned as the Vectors of the STL; in other words, we need to know each point's coordinates in 3 axes: (Z: length, Y: depth, X: width). As mentioned above, there are infinitely many slices of the Canoe. However, we can not simulate that due to the finity of our world. But we can approximate it by calculating the finite number of slices that index through the Z-axis from 0 to sum(length). Thus, we only need to calculate a finite number of slice's coordinates.

First, we need to know the curve formula of each slice. We can calculate that by getting the width and depth at a specific Z index that maps the slice. Thus we need to invoke the functions of  WidthFList and DepthFList, which map the correct section of the Canoe, and input the Z index value to get the width and depth of that specific slice. Finally, we can create a curve formula by calling the Build lambda function in the Calculation class from Calculation.py.

The algorithm would loop through each list of the LengthList (A 2D list) and use the index of the list as the index of the section. Then we use each value of the list as a value at Z-axis and input it into the WidthFList[index] and DepthFList[index] to get the width and depth of that specific slice at the Z-axis

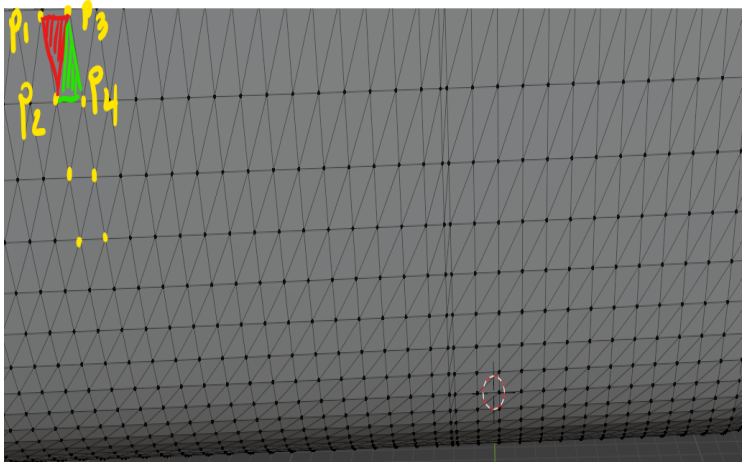
Second, we need to know the value on the X-axis and Y-axis of each point of the slice and assign it a Z value to construct the coordinate of it. By using the curve formula generated from the first part, we can get the X and Y of the point. We can loop through the width to get each point's x value, and then we can input the x value to the curve formula to get the y value. Then we will assign the z value of the point. Thus, we construct the coordinates of each point at each slice of the Canoe.

Third, we need to convert these coordinates into STL form. To understand how we approach that step, we need to understand the STL format first. Compared with other 3D file formats, STL only stores information about the surface geometry of a 3D object, excluding color, texture, or other CAD elements. Moreover, the STL format doesn't store the scale information, meaning the unit of size is undefined. STL defines any 3D object(including torus, Polyhedron, etc.) as a set of triangle surfaces, and the surfaces are composed of vertices that are ordered by the right-hand rule. Therefore, in the STL file of a 3D object, it stores the surfaces set by set, each set contains three vertices, and each vertex contains three critical coordinate information that governs by the Cartesian coordinate system: the value in X, Y, Z dimension, or abscissa, ordinate, and applicate. Thus, our task is to sequentially loop through every three points of each slice of the Canoe and sort each of them into a 3-point triangle set. And each triangle set will connect each other to construct the mesh surface of the Canoe. We know for a fact that for each two adjacent triangle sets, both of them will share at least two points. Consider the picture below:




We can observe that the red triangle and yellow triangle actually share 2 points, the P2, and P3.

However, the picture above is just an example, we are not going to connect the slice's point in 2D or by X axis, but connect the slices by Z axis. This is abstract, but we can consider the picture below:

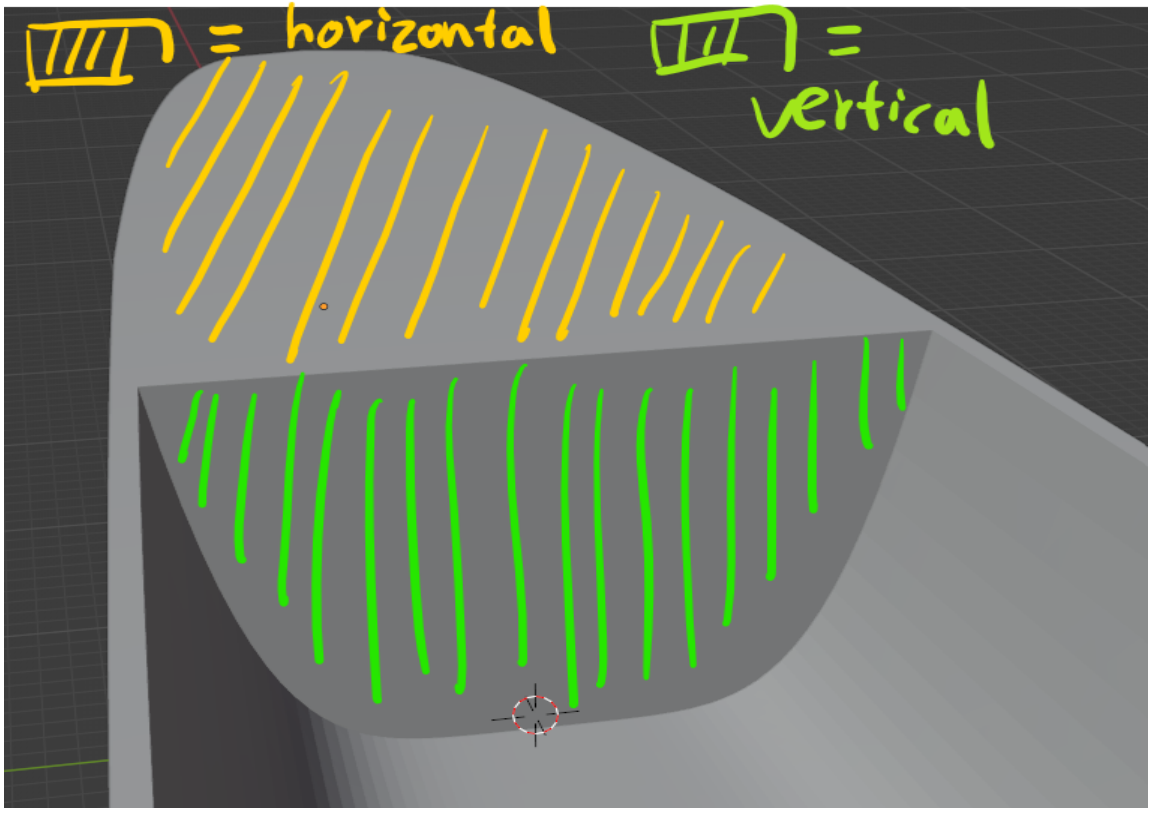



We can observe that the logic of connecting is similar. Thus we can construct the surface mesh of the canoe. The algorithm (4 set construct algorithm) would construct a set of four points where the 0th and last value represent two adjacent triangles, and the 1st and 2nd

are the sharing points of the triangle. Then we will construct two triangle meshes at the same time, in which it is easy that we only need to construct two subset bases on the 4 points where each one has 3 points, and the intersection of two subsets will generate a set that contains two values.

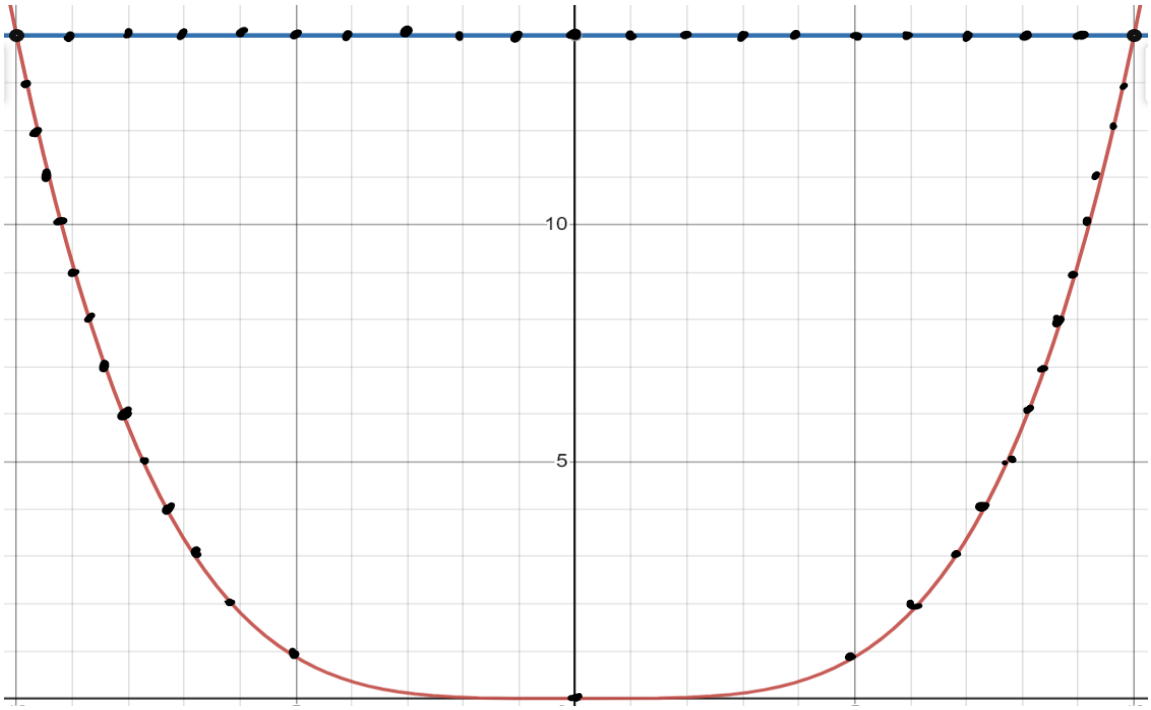
Fourth, after we construct the two mesh of the canoe (outside surface and inside surface). We will need to connect the two surfaces by their vertex (the highest point of each slice). It is simply that we only need to pick the highest two points of each slice (inside) point set and map it to the corresponding outside surface and use the same 4-set construct algorithm.

Fifth, we need to construct the mesh for cover. For each cover, there are two pieces, a vertical cover, and a horizontal cover; and consider the picture below:

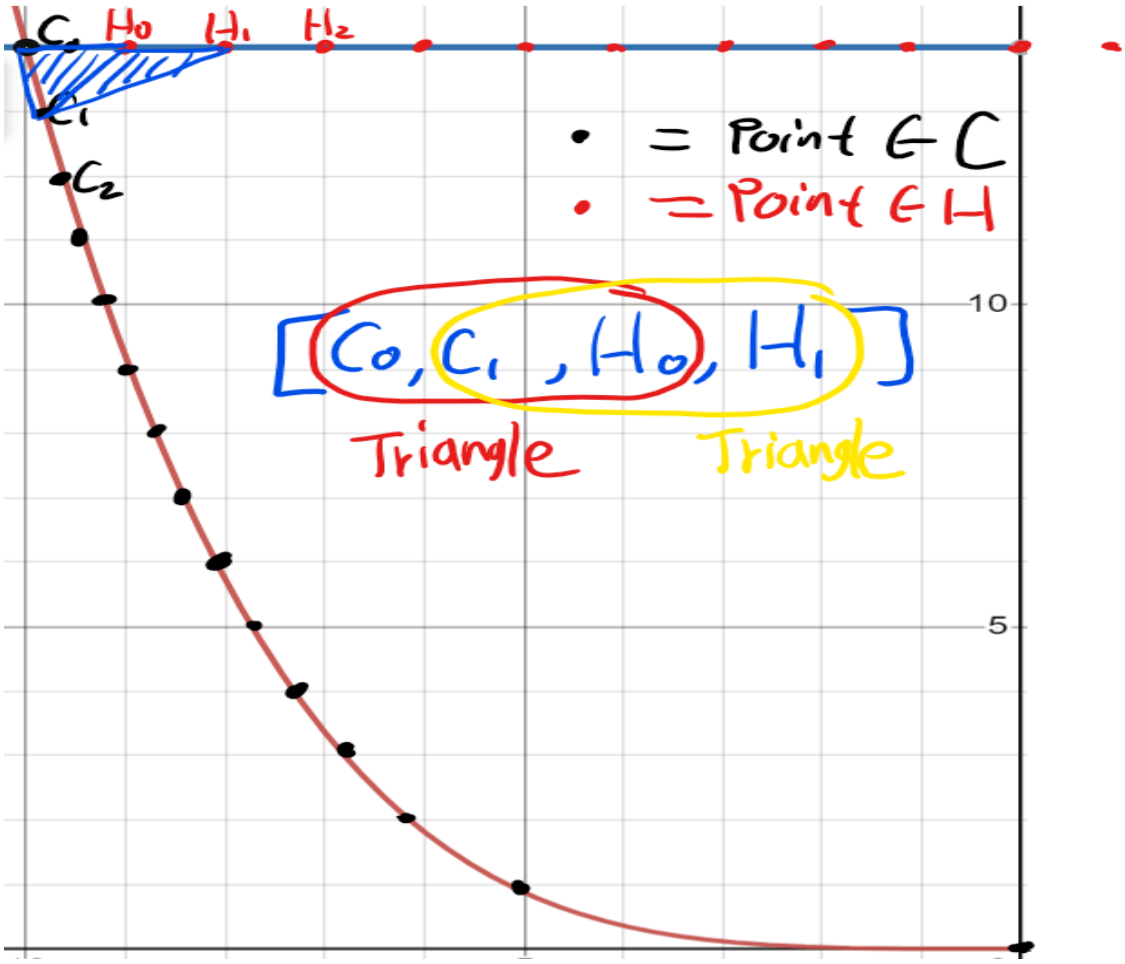



For the vertical cover, it seems that we can use the construct method of 2D shape that is mentioned in the Third section. However, only considering the points that are on the curve is not enough. We will also need to consider the points of the line (horizontal line) that connect the two vertices of the slice. Because we need to ensure the connection between vertical cover and horizontal cover, meaning the points of the horizontal line need to be counted such that there is a connection. Thus, we will need to construct a new

algorithm that results in fours set (two triangle subset)  Consider the picture below:




We can observe that the horizontal line shares two points with the curve, in which they are the vertex points of the curve. Also, the vertex points of the curve are the start point and the end point of the horizontal line. Thus, We can set the number of points of the horizontal line to be equal to the number of  points of the curve - 1 point. The definition is that $\bigcup_{n=1}^{width} P_i = P_1 \cup P_2 \cup \cdots P_{width} = \{x \mid x \in [P_{width}] \; for \; some \; n\}$ . Thus, we know that the size of the point set of the horizontal line is equal to the size of the point set of the curve -1. Considering the point set of the Horizontal line is H, and the point set of the Curve is C, we can construct the 4-point set by indexing through the X-axis. Consider the picture below:




Thus, we can construct the list of four point set as $\bigcup_{n=1}^{width} \{C_{n-1}, C_n, H_{n-1}, H_n\}$

For the **horizontal cover**, we can simply use the 2D shape construct method that is mentioned in the third section.

2. Design optimization algorithms:

   Consider the total number of variables (each section has 6 variables (width, length, depth, Ex, Ed, Ew) and there can be up to 3 sections of the canoe), if the user decided to let the program consider all variables in a range of values, there will be an extremely large

number of combinations. E.g: Users want to consider 3 variables, A, B, and C in a value range. The combination set is defined as:

$A \times B \times C = \{(a,b,c) \mid a \in [range_A] \wedge [range_b] \in B \wedge c \in [range_c]\}$ By the definition that $|X \times Y| = |X| \cdot |Y|$, we know that the number of combinations would be $range_A \cdot range_B \cdot range_C$, to create all combinations would be a O($n^3$) Which is extremely time-consuming and also space-consuming that a normal personal computer would not have the ability to support the massive memory usage of the program. ) algorithms. Consider the code:

```
List = []
For A in range (0, rangeA):
      For B in range(0, rangeB):
            For C in range(0, rangeC):
                  List.append((A,B,C))
```

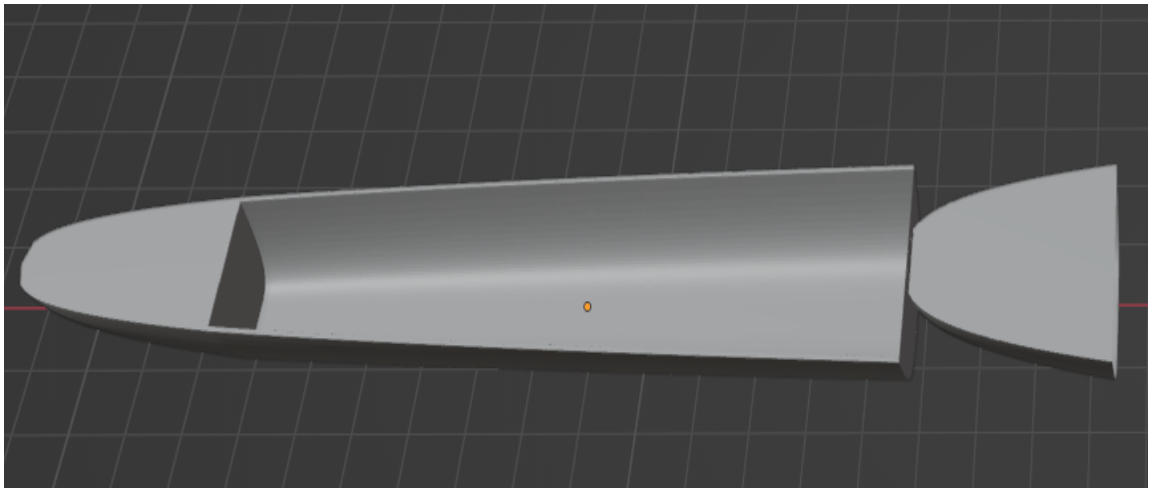
Consider the example above, we know that the time complexity of the algorithm is totally based on the value range of variables, and the highest time complexity of the algorithm would be O($n^{18}$).

A naive approach would be literally looping through all the possible combinations and finding the dataset that maps to the lightest canoe. This naive approach will be extremely time consuming considering the max time complexity is O($n^{18}$), meaning if we multiply range (originally is 1) by 2, the number of combinations will increase from 1 to 262144, if we multiply it by 2 again, it will increase from 262144 to 68719476736.

Thus, we need to apply the mathematical approach, which I still have no idea for now. I believed the right path would be finding the "weight" of each variable that determines which set of variables are most important to the canoe.

3. Generating LengthList and Length of Model_Calculation class. Even though we mathematically defined the LengthList and Length, it is still hard to implement it with code. We have to insert the cover to both LengthList and Length at the correct position, and consider its relative coordinate. By relative coordinate, I mean that cover's position, where different methods are determined for the front one and the end one. Consider the example, (length = [3,4,3], coverlength = 2.75) we know the LengthList would be [[0,1,2,3],[0,1,2,3,4],[0,1,2,3]], and Length to be [0,1,2,3,4,5,6,7,8,9,10], thus we can insert the Coverlength to the first list of the LengthList, and directly insert to the Length, creating [[0,1,2,**2.75**,3],[0, 1,2,3,4],[0,1,2,3]], and [0,1,2,**2.75**,3,4,5,6,7,8,9,10]

   But don't forget! There are two covers, so we can easily insert the front cover, but we need to think properly for the second cover, where its value is sum(length) - cover length instead of cover length. Thus it would be 7.25 in the example, and we might want to have a LengthList and Length like this:
   [[0,1,2,**2.75**,3],[0,1,2,3,4],[0, **0.25** , 1,2,3]], and [0,1,2,**2.75**,3,4,5,6,7,**7.25**,8,9,10].
   However, the result of this is incorrect due to the special shape of the canoe, since the LengthList is actually [[0,1,2,3],[0, 1,2,3,4],[**3,2,1,0**]] since the back section should be inverted to make it like a bow, otherwise, it will look like the picture below:



When we revere the last list of Length_List, we will notice that the coordinate of the cover is incorrect. Previously we considered the cover's coordinate to be the sum(length) - coverlength, which results in a 7.25, and minus the previous section's length to get 0.25. But we should notice that the distance between the cover to the length is now 0.25 instead of its real coverlength 2.75. Thus, we will need to adjust the algorithm to locate the real index of the the end cover, which will result to LengthList that:[[0,1,2,**2.75**,3],[0,1,2,3,4],[3,**2.75**,2,1,0]].