המכללה האקדמית להנדסה
אורט בראודה
ORT BRAUDE COLLEGE

Stirling Engine

Karmiel 04/07/2019

Yahia Showgan 315777292

Natali Showgan 312341563

Supervisor: Ms. Moran Kupfer

Contents

1. INTRODUCTION

A Stirling engine is an engine that has a fixed amount of gas inside and can use any external heat source to operate. By increasing the temperature of the heat source, we can increase the pressure and therefore increase the amount of power.

The Stirling Engine has three PASCO sensors attached to it. The sensors collect data and use the PASCO software to show them. The PASCO software is very limited and doesn't offer calculations on the data. Our client needed to export the data to excel and do the calculations manually, it took too much time and the results weren't that accurate.

One of the PASCO sensors can sample at 200Hz rate maximum, so the data we get is limited to 200 per second. The calculations on that limited data gave inaccurate results.

In our project we created a user friendly interface. The user only needs to upload the desired file, and then type the engine information and can get the desired graphs with no need of further calculations on his part.

The main goals of the project were: (1) get more accurate results. This goal was achieved by using interpolation methods [2] to calculate the necessary data, to then calculate the Performance graph and Electrical Power by Current graph. (2) Predict unknown RPMs and compensate for the sensors limit of sampling without using better and more expensive sensors. After we calculated the Performance graph, we derive the function and find the optimal point of RPM in which the engine produces the maximum power.

1.1. Client requirements

Pasco's software produces the data that can be exported to an excel file, and then our application loads the raw data and calculates the wanted data, it then produces the wanted graphs:

Make graphs of:

- **Thermal Power and Electrical Power by RPM- $P_E(\omega_N)$** and **$P_T(\omega_N)$**

And **from**:

- I(A)- Current
- P_E(watt)- Wattage

Make a graph of: **$P_E(I)$**

Interpolation between dynamic stages:

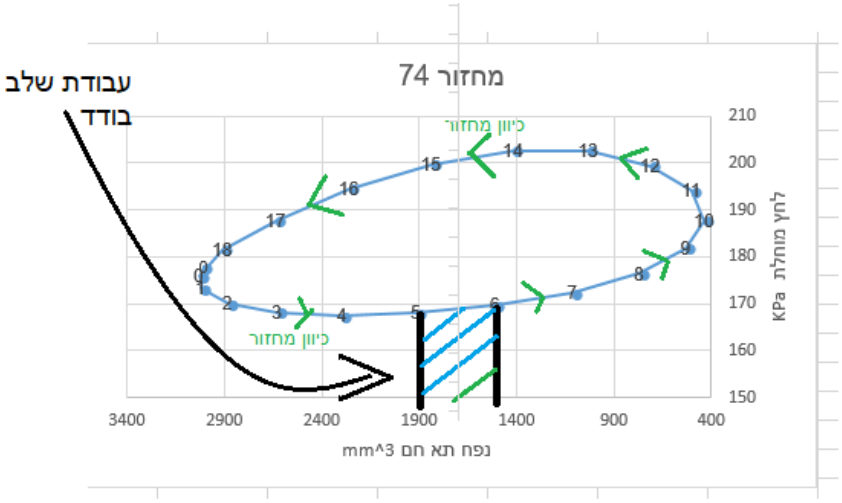
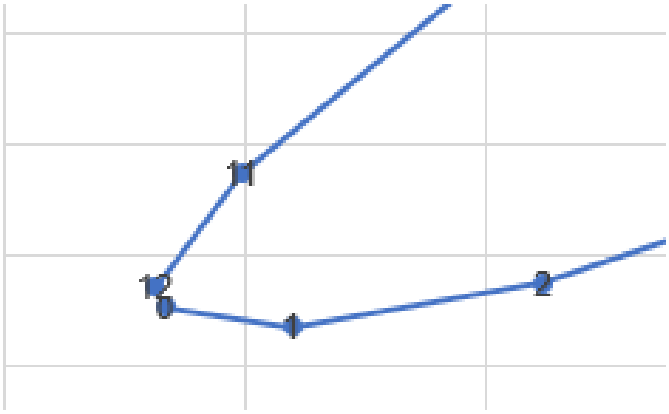


Figure 1

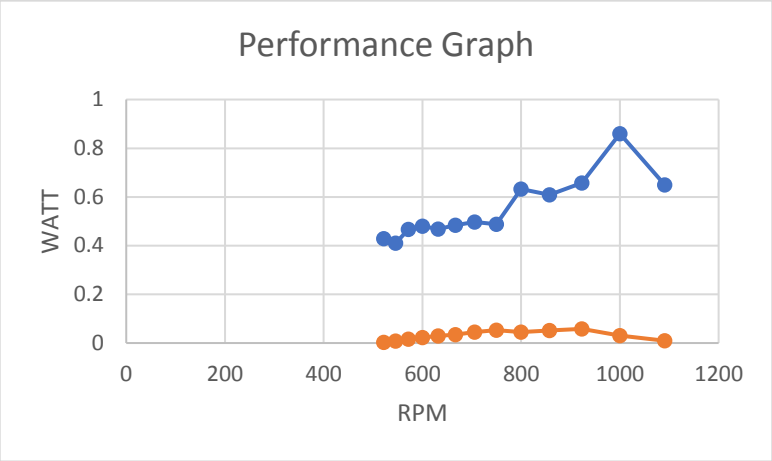
Stages of every cycle are sampled in a rate of 200Hz. In the given situation the data between two samples is assumed to be linear but it's not close. We want to do an interpolation between those sampled stages to estimate the data between them more precisely.

Refining the start and end of the cycle



Cycle doesn't start and end at the same point, that's because of the constant rate of sampling and the engine's rotation rate. The hot cylinder's volume should be cyclic, so we want to add calculated points to make it cyclic. And we can't always get the maximum and minimum volumes of the cylinder, so we want to calculate them as precise as possible.

Predicting a consumer's influence on the engine performance graph



After hooking up a consumer with adjustable load, we see that the results we got aren't like a typical engine performance graph. We can predict possible outcomes by RPMs that aren't possible to get due to the sampling rate of the sensors.

1.2. Results

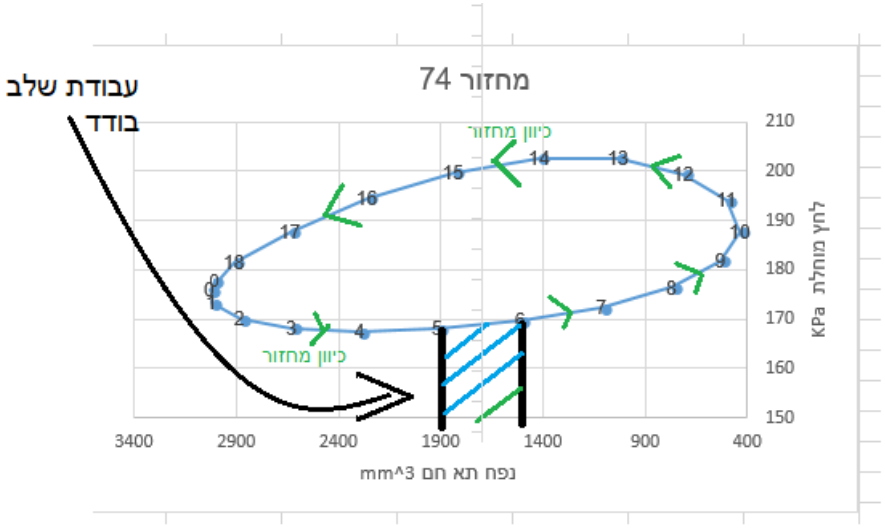
We were able to produce considerably more accurate results than the client could get with his own calculations without interpolation, after trying different 2 different methods of interpolation we found a way to use Natural Cubic Spline interpolation to get the most accurate results. The method was used to interpolate the pressure by piston area, and calculate the area inside the function. Using this method also connected the cycle's end and start points, therefore fulfilling our need of refining.

After getting the results needed for the engine performance graph we can display the graph using interpolation [2.2.1].
Thermal Power by RPM and Electrical Power by Current points are also calculated using physics formulas and then to display the graphs we used Natural Cubic Spline Interpolation again to find a function to display a better graph. We then used OpenGL and GLUT to display the graphs by finding a good amount of points with small delta increases in x and drawing lines between those points.

2. THOERY

Performing analysis for a Stirling Engine performance using data received from sensors requires using physics formulas. Described below the formulas that will be used for this project

- $W = \int_{Vi}^{Vf} P\, dV$

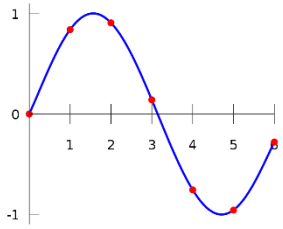


In this graph of pressure by volume the painted area is the work.

2.1. Interpolation Methods

Interpolation is the process of deriving a function from a set of data points so that the function goes through all of them and can be used to estimate other points between the given once.

We're going to define a few methods of interpolation we are using.



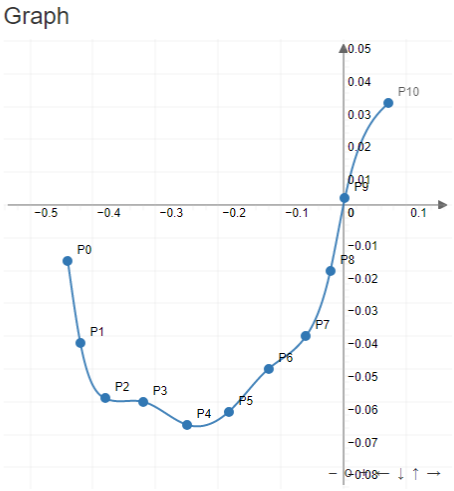
2.1.1. Spline interpolation [3]

Is a form of interpolation where the interplant is a special type of polynomial called a Spline. Spline interpolation is often preferred over polynomial interpolation because the error can be made small even when using low degree polynomials.

Example of Spline Interpolation:

$$f(x) = \begin{cases} 3.2805 \cdot 10^2 \cdot x^3 + 4.3303 \cdot 10^2 \cdot x^2 + 1.8915 \cdot 10^2 \cdot x + 2.7320 \cdot 10^1, & \text{if } x \in [-0.44, -0.42], \\ -1.4048 \cdot 10^2 \cdot x^3 + -1.5733 \cdot 10^2 \cdot x^2 + -5.8797 \cdot 10^1 \cdot x + -7.3928, & \text{if } x \in (-0.42, -0.38], \\ -2.7490 \cdot 10^1 \cdot x^3 + -2.8514 \cdot 10^1 \cdot x^2 + -9.8488 \cdot x + -1.1926, & \text{if } x \in (-0.38, -0.32], \\ 1.9120 \cdot 10^1 \cdot x^3 + 1.6233 \cdot 10^1 \cdot x^2 + 4.4700 \cdot x + 3.3472 \cdot 10^{-1}, & \text{if } x \in (-0.32, -0.25], \\ -1.3039 \cdot x^3 + 9.1435 \cdot 10^{-1} \cdot x^2 + 6.4043 \cdot 10^{-1} \cdot x + 1.5587 \cdot 10^{-2}, & \text{if } x \in (-0.25, -0.183], \\ -1.7922 \cdot 10^1 \cdot x^3 + -8.2090 \cdot x^2 + -1.0291 \cdot x + -8.6256 \cdot 10^{-2}, & \text{if } x \in (-0.183, -0.12], \\ 2.9251 \cdot 10^1 \cdot x^3 + 8.7732 \cdot x^2 + 1.0087 \cdot x + -4.7418 \cdot 10^{-3}, & \text{if } x \in (-0.12, -0.06], \\ 6.8479 \cdot 10^1 \cdot x^3 + 1.5834 \cdot 10^1 \cdot x^2 + 1.4324 \cdot x + 3.7315 \cdot 10^{-3}, & \text{if } x \in (-0.06, -0.021], \\ -3.1454 \cdot 10^2 \cdot x^3 + -8.2957 \cdot x^2 + 9.2566 \cdot 10^{-1} \cdot x + 1.8438 \cdot 10^{-4}, & \text{if } x \in (-0.021, 0.002], \\ 4.9193 \cdot 10^1 \cdot x^3 + -1.0478 \cdot 10^1 \cdot x^2 + 9.3003 \cdot 10^{-1} \cdot x + 1.8147 \cdot 10^{-4}, & \text{if } x \in (0.002, 0.071]. \end{cases}$$

Graph



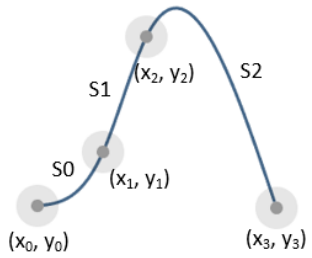
Calculate the Natural Splines

To calculate the Natural Splines, we'll use the following method:

Given a function f defined on [a, b] and a set of nodes a = x0 < x1 <…< xn = b,
A cubic spline interpolate S for f is a function that satisfies the following Conditions:

- S(x) is a cubic polynomial, denoted $S_j(x)$, on the subinterval $[x_j , x_j+1]$ for each j = 0, 1,…, n – 1; (a)
- $S_j(x_j) = f(x_j)$ and $S_j(x_j+1) = f(x_j+1)$ for each j = 0, 1, ... , n − 1; (b)
- $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ for each j = 0, 1, ... , n − 2; (Implied by (b)) (c)
- $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ for each j = 0, 1, … , n − 2; (d)
- $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ for each j = 0, 1, ... , n − 2; (e)
- One of the following sets of boundary conditions is satisfied: (f)
  - i. **$S''(x_0) = S''(x_n) = 0$ (natural (or free) boundary);**
  - ii. $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ (clamped boundary).

Interpolation algorithm and proof of correctness



Construction of a Natural Spline

As the preceding example demonstrates, a spline defined on an interval that is divided into n
subintervals will require determining 4n constants. To construct the cubic spline interpolant
for a given function f , the conditions in the definition are applied to the cubic polynomials

$$S_j(x) = a_j + b_j(x - xj) + c_j(x - xj)^2 + d_j(x - x_j)^3, \text{ for each } j = 0, 1, \ldots, n - 1.$$

Since $S_j(x_j) = a_j = f(x_j)$, condition (c) can be applied to obtain

$$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3,$$

for each $j = 0, 1, \ldots, n - 2$.

The terms $x_{j+1} - x_j$ are used repeatedly in this development, so it is convenient to introduce the simpler notation $h_j = x_{j+1} - x_j$,
for each $j = 0, 1, ..., n - 1$. If we also define $a_n = f(x_n)$, then the equation

$$a_{j+1} = a_j + b_j h_j + c_j h^2_j + d_j h^3_j$$

holds for each $j = 0, 1, \ldots, n - 1$.

In a similar manner, define $b_n = S'(x_n)$ and observe that

$$S'_j(x) = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2$$

implies $S'_j(x_j) = b_j$, for each $j = 0, 1, \ldots, n - 1$. Applying condition (d) gives

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h^2_j ,$$

for each $j = 0, 1, \ldots, n - 1$.

Another relationship between the coefficients of S_j is obtained by defining $c_n = S''(x_n)/2$ and applying condition (e). Then, for each $j = 0, 1, \ldots, n - 1$,

$$c_{j+1} = c_j + 3d_j h_j.$$

for each $j = 0, 1, \ldots, n - 1$, the new equations

$$a_{j+1} = a_j + b_j h_j + h^2_j/3 \cdot (2c_j + c_{j+1})$$

and

$$b_{j+1} = b_j + h_j(c_j + c_{j+1}).$$

The final relationship involving the coefficients is obtained by solving the appropriate equation in the form of equation, first for bj ,

$$b_j = 1/h_j \cdot (a_{j+1} - a_j) - h_j/3 \cdot (2c_j + c_{j+1}),$$

and then, with a reduction of the index, for b_{j-1}. This gives

$$b_{j-1} = 1/(h_j - 1) \cdot (a_j - a_{j-1}) - h_{j-1}/3 \cdot (2c_{j-1} + c_j).$$

Substituting these values into the equation derived, with the index reduced by one, gives the linear system of equations

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = 3/hj \cdot (a_{j+1} - a_j) - 3/h_{j-1} \cdot (a_j - a_{j-1}),$$

for each $j = 1, 2, \ldots, n-1$. This system involves only the $\{c_j\}$ as unknowns. The values of $\{h_j\}$ and {aj}are given, respectively, by the spacing of the nodes $\{x_j\}$ and the values of f at the nodes. So once the values of $\{c_j\}$ are determined, it is a simple matter to find the remainder of the constants $\{b_j\}$and {dj}.

Then we can construct the cubic polynomials $\{S_j(x)\}$.
We get the system:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & & \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \\ & \ddots & \ddots & \ddots & 0 \\ 0 & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2-a_1) - \frac{3}{h_0}(a_1-a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n-a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \\ 0 \end{bmatrix}$$

The matrix A is strictly diagonally dominant, that is, in each row the magnitude of the diagonal entry exceeds the sum of the magnitudes of all the other entries in the row. A system of this kind can be solved by the Thomas algorithm in O(n) instead of $O(n^3)$

2.3.2. Thomas algorithm Pseudo code

Forward elimination phase

loop: k=2 to n

$m = a_k / b_{k-1}$

$b_k = b_k - m \cdot c_{k-1}$

$d_k = d_k - m \cdot d_{k-1}$

end loop (k)

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \cdot & \\ & & \cdot & \cdot & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \cdot \\ \cdot \\ d_n \end{bmatrix}.$$

Backward substitution phase

$x_n = d_n / b_n$

loop: k = n-1 to 1

$x_k = (d_k - c_k x_{k+1}) / b_k$

end loop (k)

2.2. Main stages in the project

2.2.1. Engine performance graph

In order to display the engine performance graph we need to go throw this stages explained below.

1. Import data from Excel - we need to import the file that was received from Pasco's software that was converted to an excel file.

2. Inter engine details - every engine has cold and hot piston. For each piston we need to know: (1) piston area, (2) crank shaft radius, (3) min cylinder volume. Every engine has different details and it affects the results.

3. Calculate data - from the raw data we get from the excel file we need to calculate the following:

- Cyclic degree: degree%2π, to normalize the degrees (0-2π).
- Cycle number: $\frac{\text{degree}-\text{cyclicDegree}}{2\pi}$
- Cycle stage: if(cyclicDegree<previousCyclicDegree) stage= 0,
  else: stage= previousStage+1.
- Work between stages: $\frac{\text{stagePressure}+p\mathit{revStagePressure}}{2}$*(stageVolume- prevStageVolume).
- Total cycle work: ∑workBetweenStages of the same cycle or Reman Sum of the function.[4]
- Cycle power: $\frac{\text{cycleWork}}{\text{Time}}$
- T: 0.005·n (n- number of stages in the cycle).
- RPM: $\frac{1}{\text{T}}$·60
- Angular acceleration: $\frac{\text{RPM}-\text{previousRPM}}{\text{T}}$

4. Calculating work between stages- in order to calculate the work we need to calculate the area of the below graph of Absolute Pressure by volume [Figure 1], by using interpolation method [2.3.2].

Because the Natural Cubic Spline Interpolation only works for x values in increasing order and our function is a closed curve, we decided to divide the functions in to two both starting at the smallest x value and ending at highest x value. We then interpolate both functions,

5. Filtering results and combine same RPMs - after the necessary data was calculated we need to filter them with angular acceleration=0. Then calculate an average power for the same RPMs.

6. Showing the graphs - to show the graphs first we have to use interpolation method [2.3.1] on the points, find the maximum point then display all on the graph using OpenGL and GLUT.

2.2.2. Electrical power by Current graph

In order to display the Electrical Power by Current graph the stages we need to do are:

1. **1. Import data from Excel -** we need to import the file that was received from Pasco's software.

2. **Showing the graphs -** to show the graph first we have to pick the right points we want to display on the graph (Electrical power, Current) then use interpolation method [2.3.1] on the points, find the maximum point then display all on the graph using OpenGL.

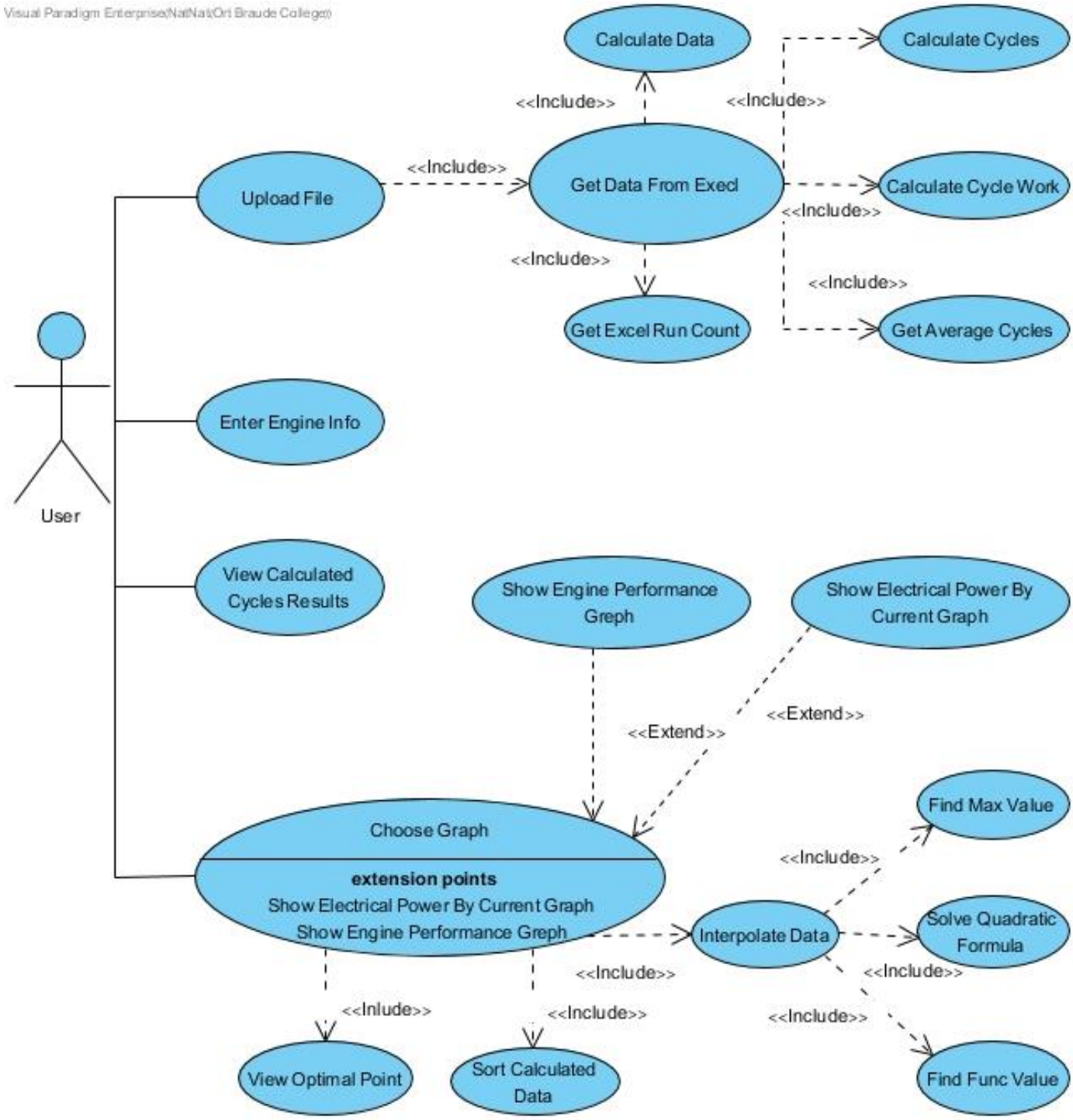
2.4. Programming Languages & Frameworks

In our project we chose to use C++ language. We chose this language for several reasons: (1) our program will need to go over a lot of data and C++ has fast top notch performance. (2) C++ is a language that's derived from C, it has everything that C has and more. (3) C++ would be much easier to write in and would increase usability in the future.

For GUI we are using Windows form application and for displaying our graphs we are using OpenGL it offers reliability, it also supports many platforms and operating systems.

3. SOFTWARE ENGINEERING DOCUMENTS

3.1. Use Case Diagram



UC1: Upload File

- Goal: upload an Excel file.
- Preconditions: file should be of type csv and with format of a Pasco imported file.
- Possible user errors: attempted to upload non csv type file.

Actor	System
1) Click 'Choose file' button.	2) Display ' OpenFileDialog '.
3) Choose desired file and click 'Open' button.	4) Save the path of the request file.
5) Choose run number.	6) Save run number.
7) Click 'Import' button.	8) Open engine info form.

UC2: Get data from Excel (inclusion):

- Get data from an excel file.
- Preconditions: file path and run number should be inserted by user.
- Limitations: number of rows in Excel file must be larger than 100.

Actor	System
1) Click 'input file' button.	2) Find Excel file
	3) Find run number in the file.
	4) Read every row and push it to a vector.

UC3: Enter engine info:

- Enter engine information.
- Preconditions: Excel file uploaded.
- Limitations: user typed words and not numbers.

Actor	System
1) Click 'Import' button.	2) Display 'Engine Info Form'
3) Enter engine details.	4) Save engine details.
5) Choose data units of measurement.	6) Save units of measurement.

UC4: Calculate data (inclusion):

- Calculate data according to given formulas.
- Preconditions: data was read properly from Excel file.

Actor	System
1) Click 'Calculate Data' button.	2) Do necessary calculations on data
	3) Save new data in new vector.

UC5: Calculate cycles (inclusion):

- Calculate data for each cycle.
- Preconditions: data should be after preliminary calculation.

Actor	System
	4) Group data into cycles
	5) Fill a vector of cycles.

UC6: Calculate cycle work (inclusion):

- Calculate work per each cycle.
- Preconditions: data should be grouped into cycles.

Actor	System
	6) Using interpolation find polynomials between every 2 points.
	7) Sum area of all point in cycle by integration.
	8) Save work of every cycle with 0 angular acceleration.

UC7: Get average cycles (inclusion):

- Group cycles with the same rpm.
- Preconditions: cycle work should be calculated for each cycle.

Actor	System
	9) Group cycles by rpm.
	10) Calculate average work for each grouped cycles.
	11) Save work of every cycle.
	12) Save filtered cycles in new vector.

UC8: Sort data (inclusion):

- Sort data by rpm.
- Preconditions: Cycles should be filtered.
- Pseudo code Flow:

Actor	System
	13) Receive filtered cycles.
	14) Sort cycles by rpm.
	15) Return sorted vector.

UC9: Show calculated cycle's results:

- Show calculated data results.
- Preconditions: cycles should be calculated.

Actor	System
	16) Display ' calculated cycles form'.

UC10: Choose Graph:

- Display graphs.
- Preconditions: all data should be calculated.

Actor	System
1) Click 'Choose Graph' button	2) Display 'Choose Graph Form'

UC11: Interpolate data (inclusion):

- Interpolate received points.
- Preconditions: vector x and vector y should be received of wanted graph.
- Pseudo code Flow:

Actor	System
	3) Receive points in form (x,y) to display in graph.
	4) Run interpolation algorithm on points.
	5) Return polynomial equation from interpolation algorithm.

UC12: Find func value (inclusion):

- Find function Y value.
- Preconditions: Polynomial equation should be calculated.
- Pseudo code Flow:

Actor	System
	17) Receive polynomial equation and X value.
	18) Insert X into polynomial equation.
	19) Return Y value.

UC13: Solve quadratic formula (inclusion):

- Solve quadratic formula.
- Preconditions: Polynomial equation should be calculated.
- Pseudo code Flow:

Actor	System
	20) Receives Polynomial equation.
	21) Run quadratic formula algorithm.
	22) Return formula Xs.

UC14: Find max value (inclusion):

- Find maximum value.
- Preconditions: polynomial equation should be calculated in interpolation.
- Pseudo code Flow:

Actor	System
	23) Receive polynomial equation.
	24) Divert polynomial equation and find max value.
	25) Return max value.

UC15: Show Electrical Power by Current graph (extension):

- Show Pe (I) graph.
- Preconditions: all data should be calculated.

Actor	System
1) Click 'Electrical Power By Current Graph ' button	2) Show Electrical Power by Current Graph from polynomial equation.

UC16: Show Engine Performance graph (extension):

- Show Engine Performance graph.
- Preconditions: all data should be calculated.

Actor	System
1) Click 'Engine Performance Graph ' button	2) Show Engine Performance Graph from polynomial equation.

UC17: View optimal point (inclusion):

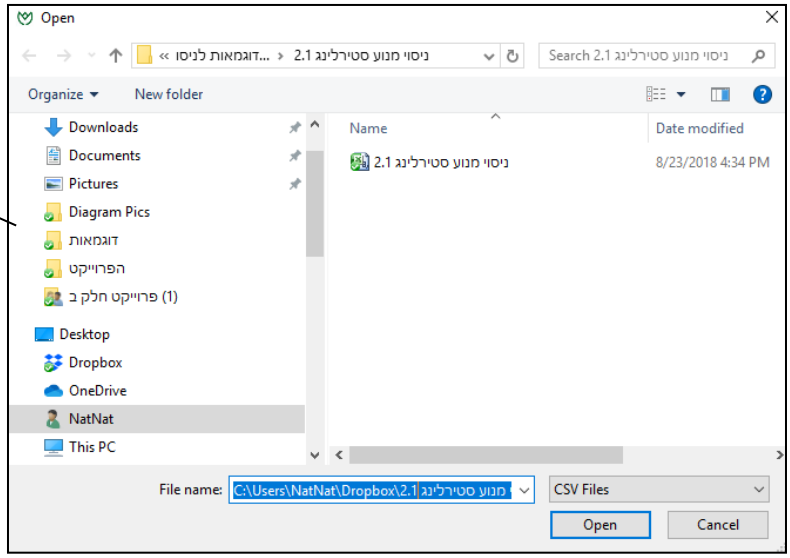
- View optimal point of graph.
- Preconditions: graph should be ready.
- Pseudo code Flow:

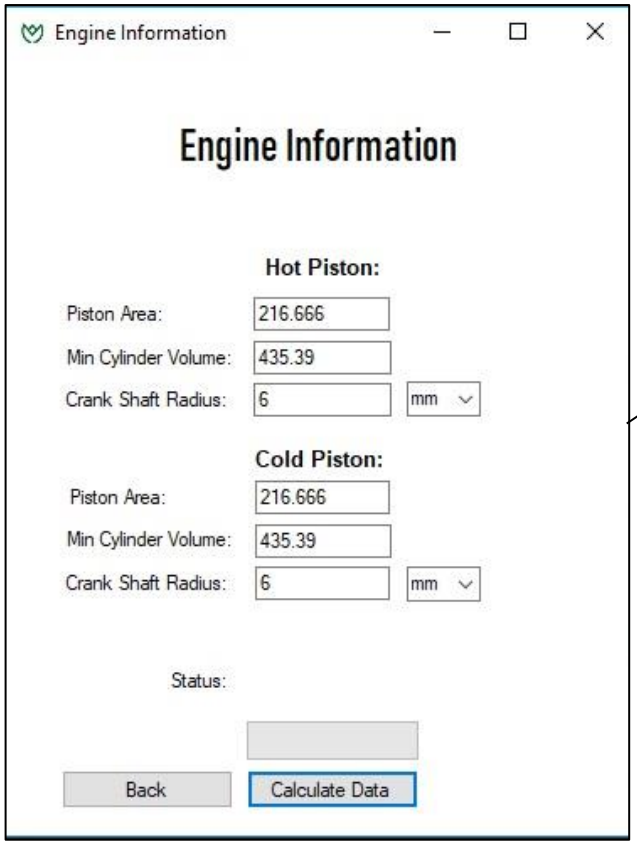
Actor	System
	3) Show optimal point on graphs.

3.2. GUI

Main screen:
The user can press Choose CSV file then press import to upload the file.

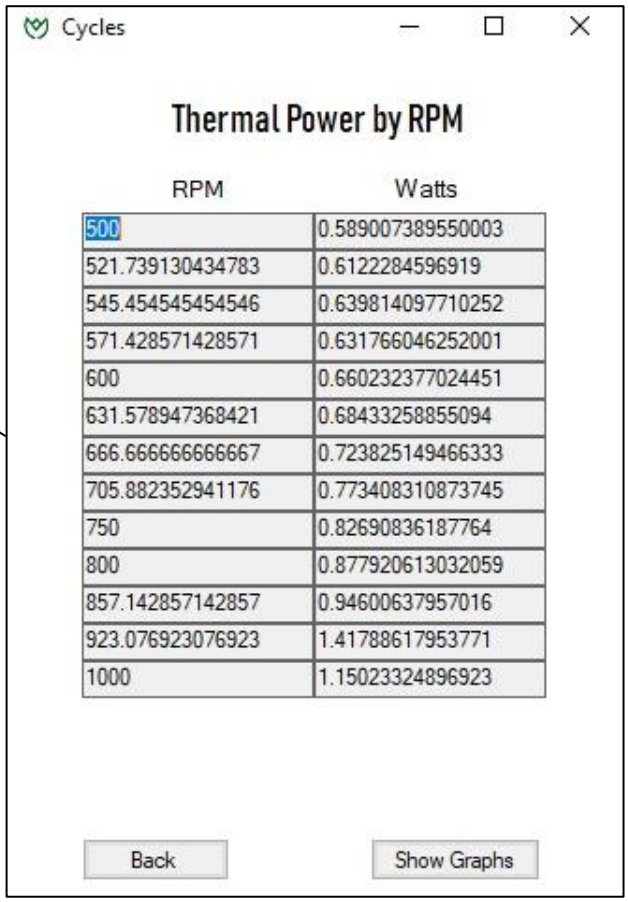
The user must choose a CSV file in order to continue.





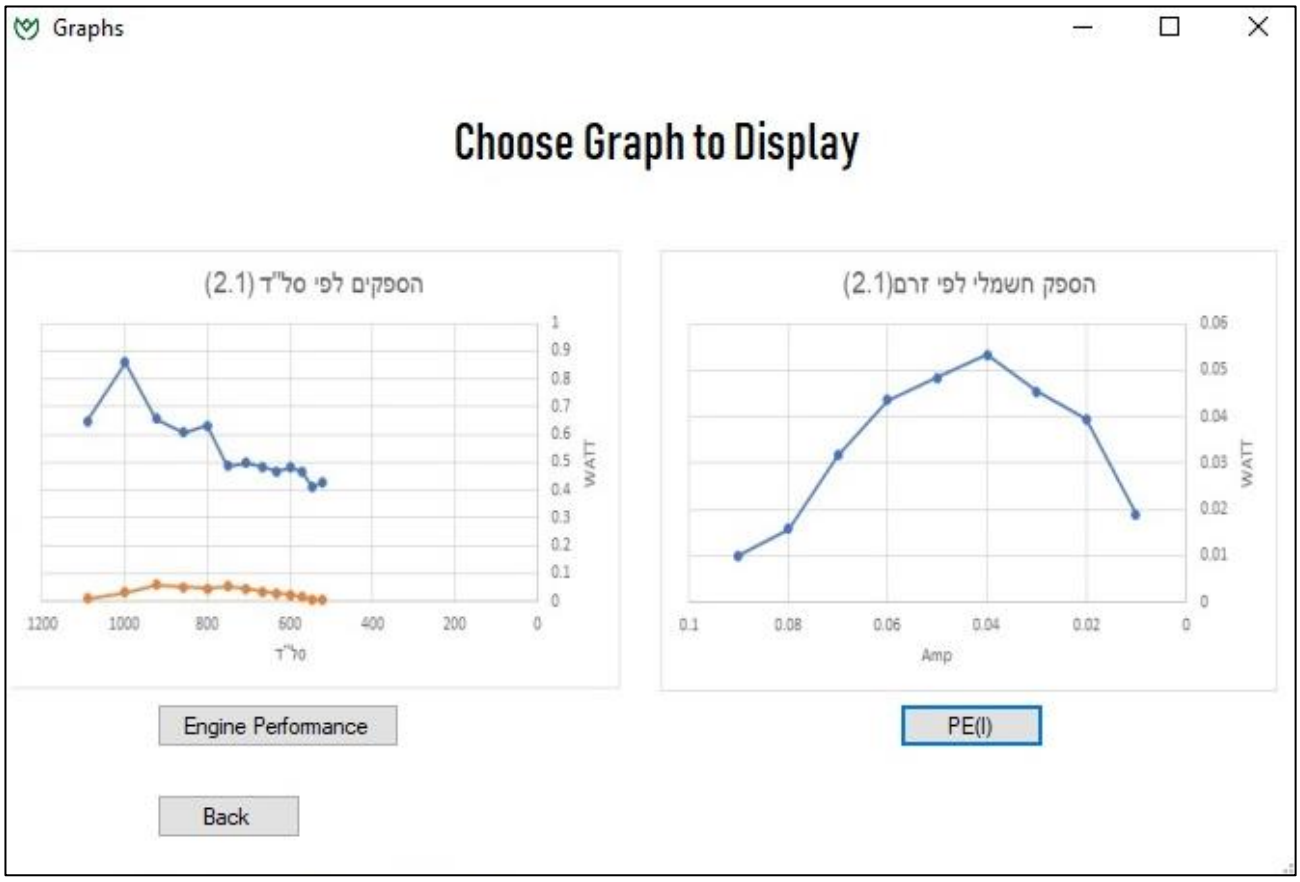
After uploading the file, the user needs to fill the engine information and to press calculate data. The user can also choose if he wants to fill the radius in mm, cm or m.

When everything is uploaded correctly, the software performs all the needed calculation. After the calculations it will display the wanted data for the Engine Performance graph.

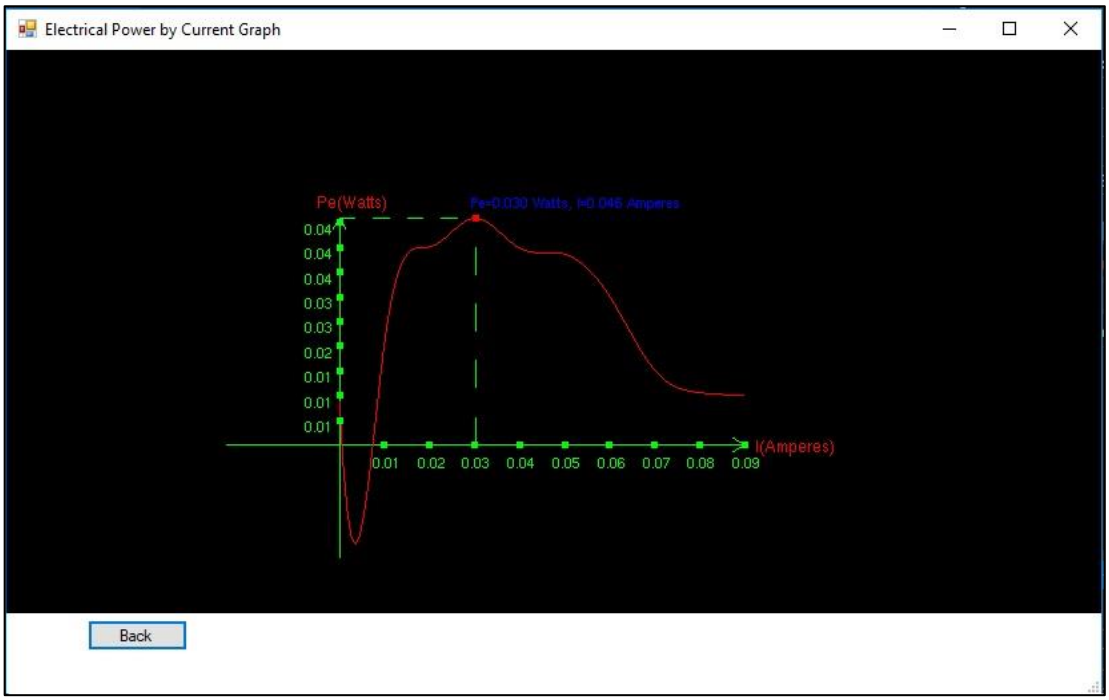


Here the user can choose what graph he wants to display:

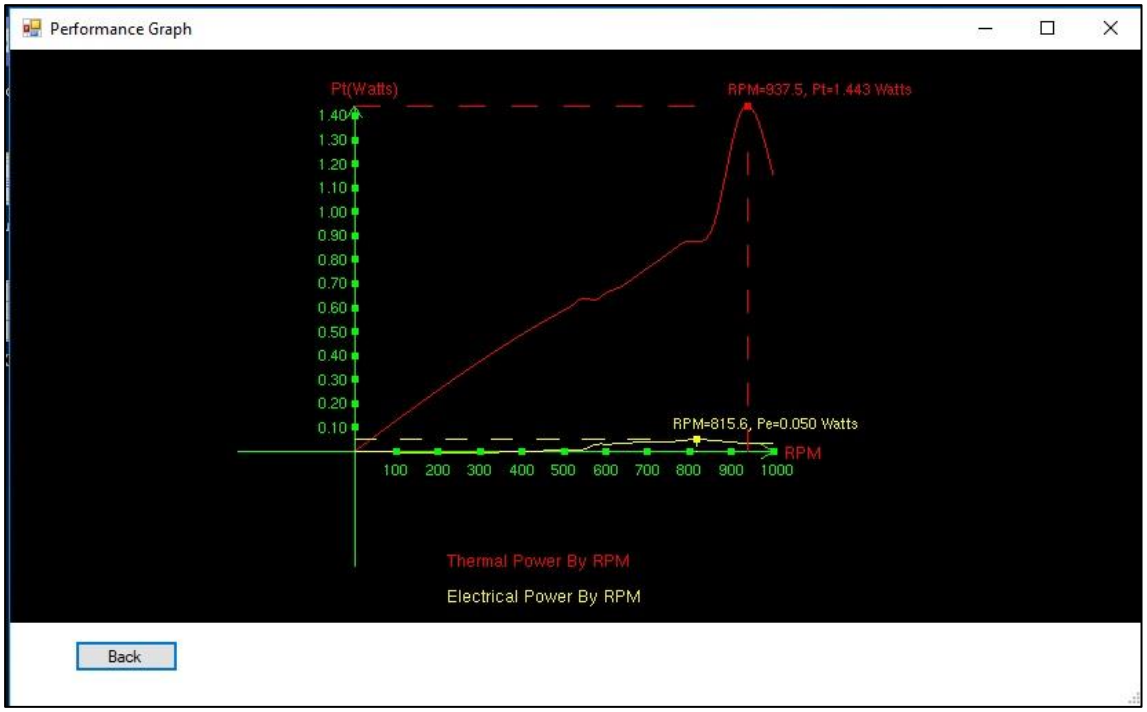
1. Engine Performance graph which contains: Electrical Power by RPM and Thermal Power by RPM.
2. Electrical Power by Current graph.



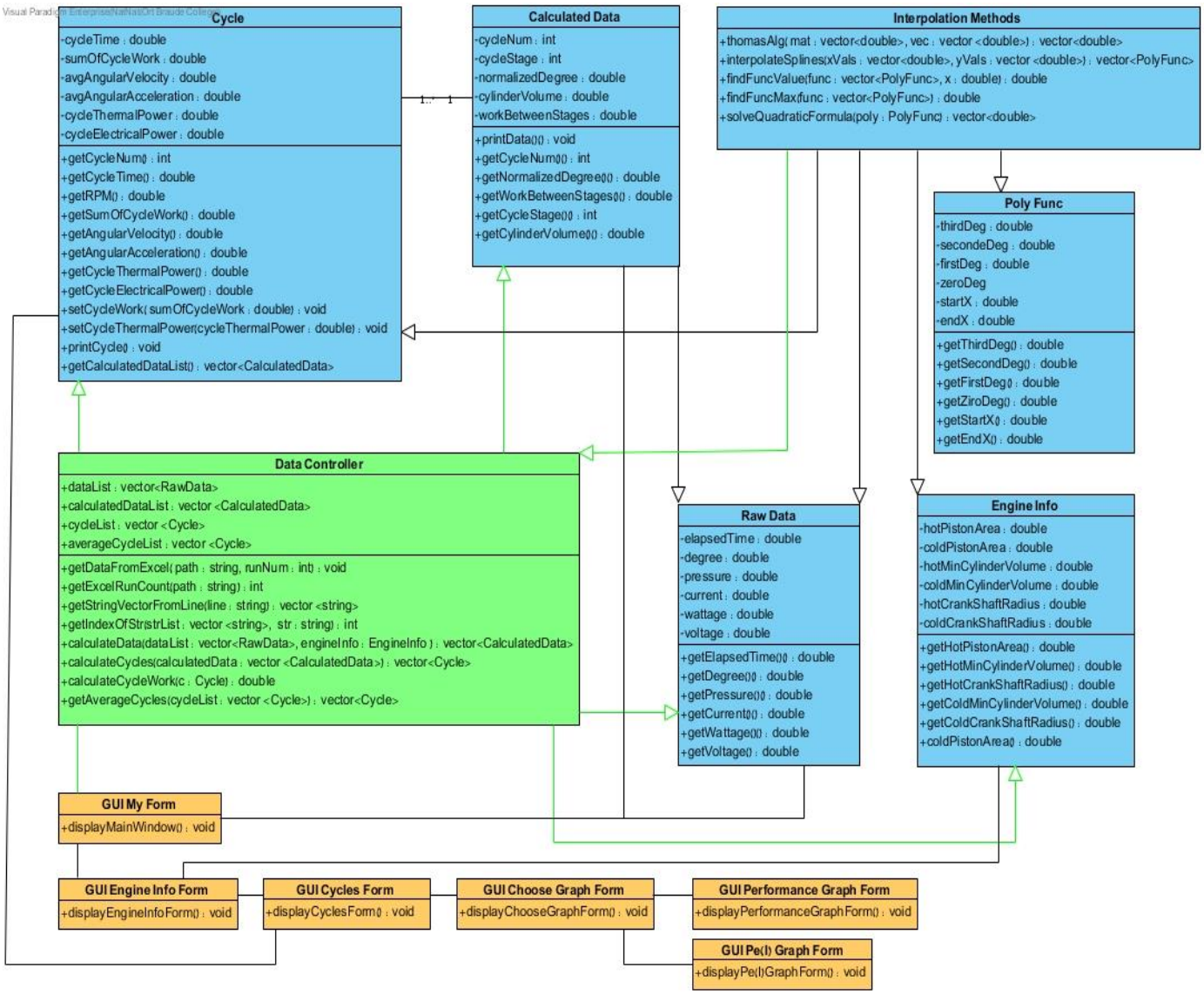
Electrical Power by RPM graph, the user can also see which Current gives us the maximum Electrical Power



Engine Performance graph, here the user can also see in which RPM the engine produces maximum Power



3.3. Design


Cycle
-cycleTime : double
-sumOfCycleWork : double
-avgAngularVelocity : double
-avgAngularAcceleration : double
-cycleThermalPower : double
-cycleElectricalPower : double
+getCycleNum() : int
+getCycleTime() : double
+getRPM() : double
+getSumOfCycleWork() : double
+getAngularVelocity() : double
+getAngularAcceleration() : double
+getCycleThermalPower() : double
+getCycleElectricalPower() : double
+setCycleWork(sumOfCycleWork : double) : void
+setCycleThermalPower(cycleThermalPower : double) : void
+printCycle() : void
+getCalculatedDataList() : vector<CalculatedData>
1..*
1
Calculated Data
-cycleNum : int
-cycleStage : int
-normalizedDegree : double
-cylinderVolume : double
-workBetweenStages : double
+printData()() : void
+getCycleNum()() : int
+getNormalizedDegree()() : double
+getWorkBetweenStages()() : double
+getCycleStage()() : int
+getCylinderVolume()() : double
Interpolation Methods
+thomasAlg(mat : vector<double>, vec : vector <double>) : vector<double>
+interpolateSplines(xVals : vector<double>, yVals : vector <double>) : vector<PolyFunc>
+findFuncValue(func : vector<PolyFunc>, x : double) : double
+findFuncMax(func : vector<PolyFunc>) : double
+solveQuadraticFormula(poly : PolyFunc) : vector<double>
Poly Func
-thirdDeg : double
-secondeDeg : double
-firstDeg : double
-zeroDeg
-startX : double
-endX : double
+getThirdDeg() : double
+getSecondDeg() : double
+getFirstDeg() : double
+getZiroDeg() : double
+getStartX() : double
+getEndX() : double
Data Controller
+dataList : vector<RawData>
+calculatedDataList : vector <CalculatedData>
+cycleList : vector <Cycle>
+averageCycleList : vector <Cycle>
+getDataFromExcel(path : string, runNum : int) : void
+getExcelRunCount(path : string) : int
+getStringVectorFromLine(line : string) : vector <string>
+getIndexOfStr(strList : vector <string>, str : string) : int
+calculateData(dataList : vector<RawData>, engineInfo : EngineInfo) : vector<CalculatedData>
+calculateCycles(calculatedData : vector <CalculatedData>) : vector<Cycle>
+calculateCycleWork(c : Cycle) : double
+getAverageCycles(cycleList : vector <Cycle>) : vector<Cycle>
Raw Data
-elapsedTime : double
-degree : double
-pressure : double
-current : double
-wattage : double
-voltage : double
+getElapsedTime()() : double
+getDegree()() : double
+getPressure()() : double
+getCurrent()() : double
+getWattage()() : double
+getVoltage() : double
Engine Info
-hotPistonArea : double
-coldPistonArea : double
-hotMinCylinderVolume : double
-coldMinCylinderVolume : double
-hotCrankShaftRadius : double
-coldCrankShaftRadius : double
+getHotPistonArea() : double
+getHotMinCylinderVolume() : double
+getHotCrankShaftRadius() : double
+getColdMinCylinderVolume() : double
+getColdCrankShaftRadius() : double
+coldPistonArea() : double
GUI My Form
+displayMainWindow() : void
GUI Engine Info Form
+displayEngineInfoForm() : void
GUI Cycles Form
+displayCyclesForm() : void
GUI Choose Graph Form
+displayChooseGraphForm() : void
GUI Performance Graph Form
+displayPerformanceGraphForm() : void
GUI Pe(I) Graph Form
+displayPe(I)GraphForm() : void

3.4. Testing

Engine Info Class

Test No.	Subject	Expected Results	Actual results
1	Test Engine Info Constructor for valid input.	The constructor will be able to make an Engine Info type variable.	Pass
2	Insert negative numbers for the Engine info.	It will recognize the negative numbers and will ask to load valid ones.	Pass

Raw Data class:

Test No.	Subject	Expected Results	Actual results
1	Test Raw Data Constructor for valid input.	The constructor will be able to make a Raw Data type variable.	Pass
2	Load negative time to Raw Data Constructor.	It will recognize the negative time and will throw an exception.	Pass

Calculated Data Class:

Test No.	Subject	Expected Results	Actual results
1	Test Calculated Data Constructor for valid input.	The constructor will be able to make a Calculated Data type variable.	Pass
2	Load degree larger then 2*PI to Calculated Data constructor.	It will recognize the invalid degree and will throw an exception.	Pass
3	Load negative cycle stage to Calculated Data Constructor.	It will recognize the negative cycle stage and will throw an exception.	Pass

Interpolation Methods Class:

Test No.	Subject	Expected Results	Actual results
1	Send valid data to **FindFuncValue** function.	The function will return the y value of the given x.	Pass
2	Send to **FindFuncValue** function, X value that is not in the range of X values of the given functions.	The function will recognize that X is out of range and will throw an exception.	Pass
3	Call **SolveQuadraticFormula** with 2 different polynomials. One with only one results, other with 2 different results and compare to pre calculated results of x1,x2.	The functions will find the correct x values.	Pass
4	Call **SolveQuadraticFormula** a polynomial that has no real solution.	The function should throw an exception.	Pass
5	Call **FindFuncMax** with 2 polynomials.	Should return the polynomials max value x.	Pass
6	Calculate area of an ellipse using **deCasteljau**'s algorithm to find a function going through 10 points on the ellipse.	After comparing the real area of the ellipse and the area calculated with our interpolation algorithm, the results should be close.	**Fail**
7	Calculate area of an ellipse using **NaturalSplineInterpolation** with 10 points on the ellipse	Comparing to the real area of the ellipse the area we calculated using our interpolation would be close.	Pass(0.17% error)

Data Controller Class:

Test No.	Subject	Expected Results	Actual results
1	Load a valid excel file to Data Controller- import Excel	It will be recognized as a valid file and import it to the Data Controller Class.	Pass
2	Load an excel file with data that doesn't match Pascal exported files.	It will be recognized as an invalid file and will ask to load a valid one.	Pass
3	Load a run number that doesn't exist on the excel file.	It will recognize that the run number doesn't exist, and will ask to load a new run number that exists.	Pass
4	Send valid input to Thomas Algorithm function.	The function will return the answer vector.	Pass
5	Send non triangle matrix to Thomas Algorithm function.	The function will recognize that the matrix is not triangle and will throw an exception.	Pass
6	Sent a vector with size that doesn't match to the matrix size to Tomas Algorithm function.	The function will recognize that the vector size and matrix size doesn't match and will throw an exception.	Pass
7	Send valid input to Interpolation Splines function.	The function will return a list of polynomials-one between every two given points.	Pass
8	Send number of Y values greater than the number of X values to Interpolation Splines function.	The function will recognize that the X values number and the Y values number doesn't match and will throw an exception.	Pass

4. RESULTS AND CONCLUSIONS

4.1. Results

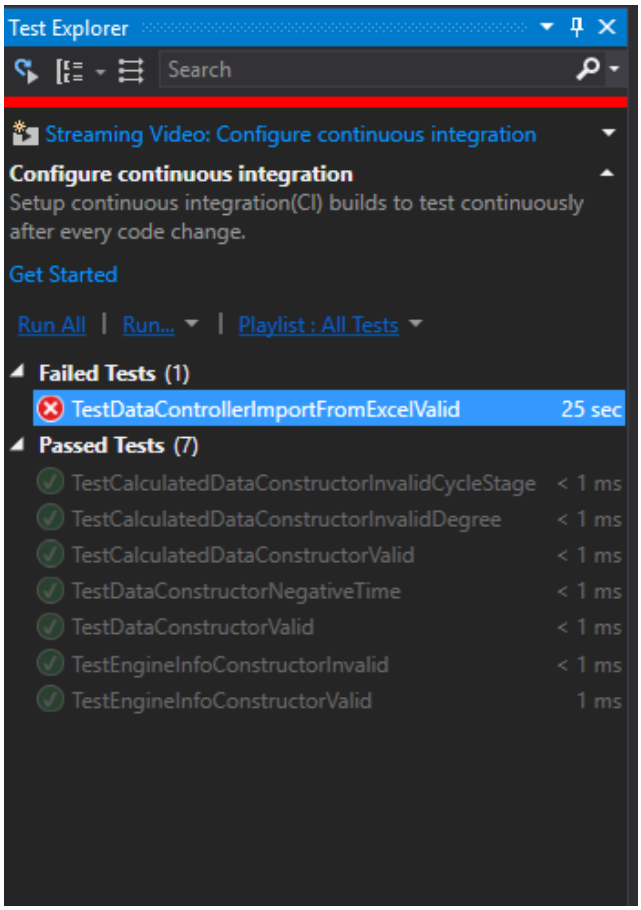
In our project we were able to meet our client's main goals as described in the introduction [1.2]. We created a program that the client can use to analyze the engines results, to then see what he can do in order to improve the engines performance.

In the section below, we will show results of our experiment according to the expected results.

Test 1:

Error: trying to get the pressure from the wrong index in the CSV file.

How to fix: check where the correct index is and change it.

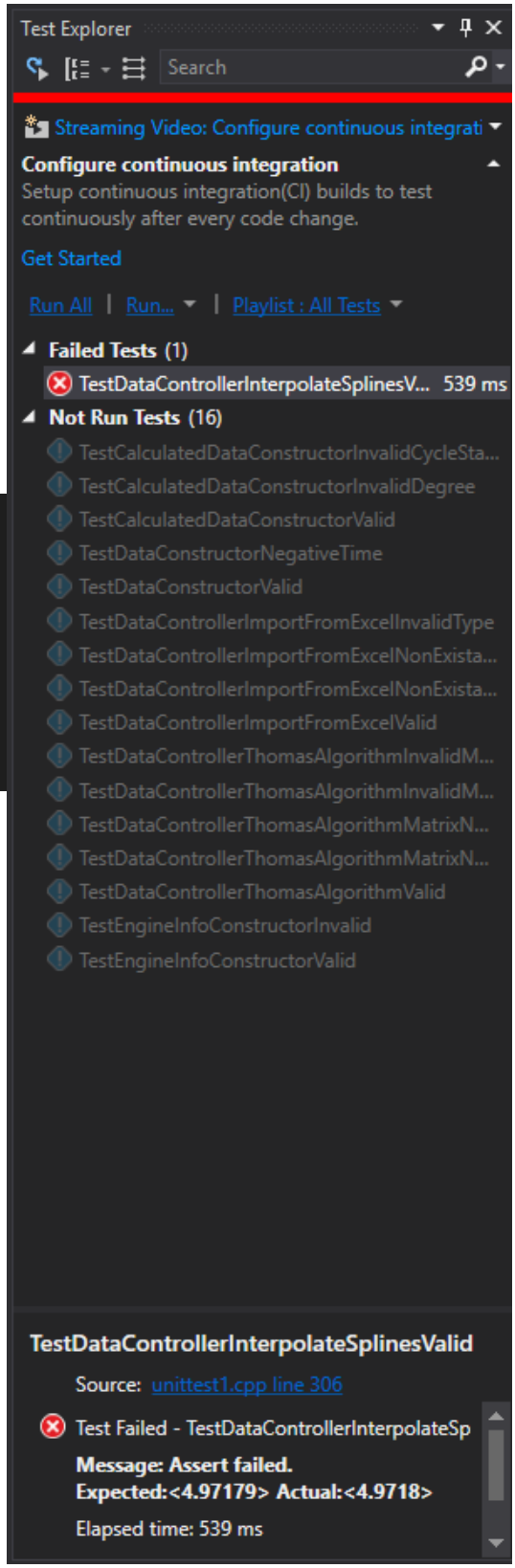


Test 2:

Error: there was a test fail because the interpolation algorithm did not produce exact results.

How to fix: the interpolation algorithm will produce accurate results we allowed a small enough error.
The interpolation finds polynomials with range of error less than 0.01 or 1%.

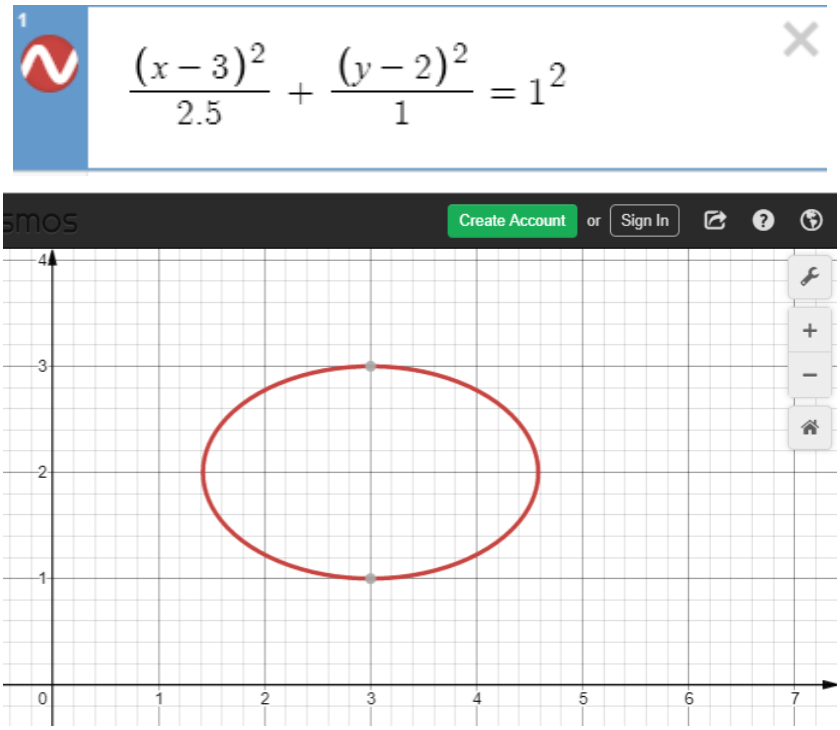
```
double error;
error = abs(actPolyList[i].getThirdDeg() - expPolyList[i].getThirdDeg());
Assert::IsTrue(error < fmax(0.01,0.01*abs(actPolyList[i].getThirdDeg())));

error = abs(actPolyList[i].getSecondDeg() - expPolyList[i].getSecondDeg());
Assert::IsTrue(error<fmax(0.01, 0.01*abs(actPolyList[i].getSecondDeg())));

error = abs(actPolyList[i].getFirstDeg() - expPolyList[i].getFirstDeg());
Assert::IsTrue(error<fmax(0.01, abs(0.01*abs(actPolyList[i].getFirstDeg()))));

error = abs(actPolyList[i].getZeroDeg() - expPolyList[i].getZeroDeg());
Assert::IsTrue(error<fmax(0.01,0.01*abs(actPolyList[i].getZeroDeg())));
```



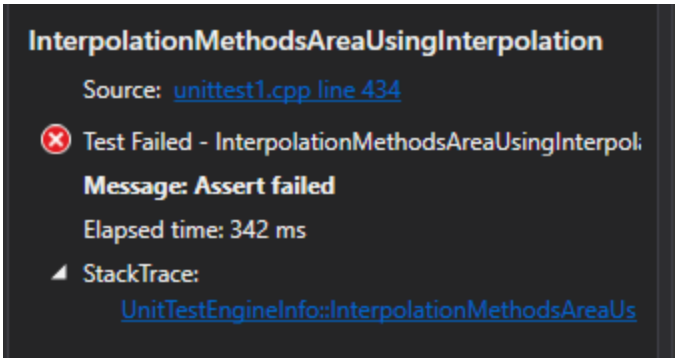
Test 3:

Error**:** in this test we took poins of an ellipse and used our algorithm on them, we then compared the ellipse area to the algorithm's result. There was a test fail because our interpolation algorithm was producing innaccurate results.

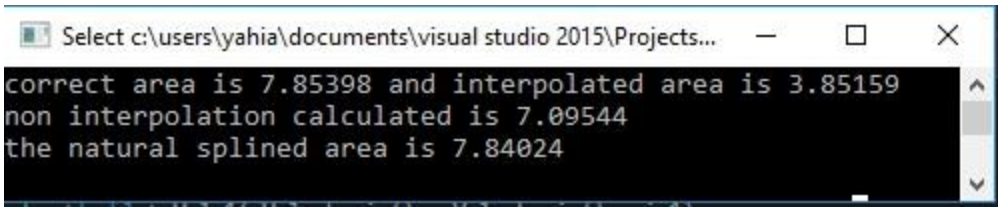
How to fix: we had to change our interpolation method as described in **[**2.2.1. stage]




```
TEST_METHOD(InterpolationMethodsAreaUsingInterpolation)
{
    std::vector<double> xVals{ 0.5,1.27,2.24,3,4.12,
        5.5,5.12,3,1.92,1.06,0.5 };
    std::vector<double> yVals{2,1.278,1.047,1,1.106,
        2,2.53,3,2.902,2.631,2};
    std::vector<double> prevPoint = InterpolationMethods::deCasteljau(xVals, yVals, 0);
    std::vector<double>point;
    double sumCycleWork = 0;
    double step = 0.01;
    double t, workBetweenStages;
    for (t = 0; t <= 1; t += step)
    {
        point = InterpolationMethods::deCasteljau(xVals, yVals, t);
        workBetweenStages = ((point[1] + prevPoint[1]) / 2)*(point[0] - prevPoint[0]);
        sumCycleWork += workBetweenStages;
        prevPoint = point;
    }
    sumCycleWork = abs(sumCycleWork);
    double correctArea = 7.85398163397;
    Assert::IsTrue((sumCycleWork < correctArea+0.1*correctArea) &&
        (sumCycleWork > correctArea -3-0.1*correctArea));
}
```



Here are the results of the correct area, the old interpolation method result and the new interpolation method result:



4.2. Conclusions

In the first phase of the project we didn't consider a few things that later in the second phase caused us problems. We had to waste a lot of time figuring out how to solve these problems. This caused us time loss and we didn't manage to do all the things we wanted to add to the project. Like micro analyses on the results we get from the engine. We wanted to explore what happens when we take angular acceleration that is different from 0 and see how it effects on the final results.

At the experiment stage, we have made executions on different interpolation algorithms, trying to understand how every algorithm works, compare their results and at the end decide which algorithm will provide us the best results, which algorithm will suit our needs.

. After considering different methods of interpolation for both our uses, we first implemented De Casteljau's algorithm for the interpolating a function for an ellipse with 10 points, got us the terrible results with **51%** error.

We concluded that Natural Spline Interpolation works best for both of them. To calculate an area of an ellipse with only 10 points for interpolation, we were able to achieve only **0.17%** error of the correct area compared to 9.6% error with Riemann's sum using only the 10 points.

Using the Natural Cubic Splines method also got us a good function for our graph. The method was tested and would find a function (polynomial list) using few points with less than **1%** error. Given that this interpolation is at the final stage and its main use is to better display the graph, the error is negligible.

REFERENCES

[1] M. Christoph & G. Arnaud *Stirling Engine*, University of Gavle, 2007.

[2] F. T. Krogh *Efficient Algorithms for Polynomial Interpolation and Numerical Differentiation*, California Institute of Technology, 1970.

[3] A. C. Aitken, *On interpolation by iteration of proportional parts, without the use of differences*,Edinburgh Math, 1932.

[4] C. D. Boor, *A Practical Guide to Splines*,Springer New York, 2001.

[5] R. L. Myers, *The Basics of Physics*, Greenwood Publishing Group, 2006.

[6] R. Larson&B. H. Edwards, *Calculus I with Precalculus*, Brooks/Cole, Cengage Learning, 2011.

[7] R. L. Burden, *Numerical Analysis*,Brooks/Cole, Cengage Learning, 2010.