Curve Fitting of UTSA Mascot Rowdy

Final Report

University of Texas at San Antonio

ME4773.001 – Robotics

Fall 2016

Students:

Jose Perez

Ryan Kinsey

Due Date:

December 9, 2016

Table of Contents

Introduction

The concept of finding equations for faces was introduced by Wolfram Alpha in a blog by Michael Trott. In his blog, he describes the steps he took to generate a function for everything; from images of famous people to images of well-known cartoon figures. This report aims to inform of the steps taken to attempt this feat for the University of Texas at San Antonio mascot, Rowdy.

Procedure

First, an image of Rowdy was acquired to turn it into a binary image for edge detection. To insure the best reading in MATLAB, a high-quality image source of the UTSA logo was required. A raster base image from the internet could have worked but the quality of the image would have been subpar. A vector based image is ideal in that the quality is lossless regardless of the size of the image. The UTSA brand identity guide was obtained from the school's local SharePoint. The document is stored as a .pdf file extension and this is ideal because vector based images can be withdrawn from the document without any loss in quality.

Using the open source based software Inkscape, which is a vector based imaging software, the profile of the UTSA mascot was extracted. In order to reduce the complexity of the image, the TM stamp was removed. The color was also removed in order to assist the edge finding in MATLAB. For simplicity, the silhouette was created to reduce the number of edges needed to be detected. The silhouette image was then uploaded into MATLAB and turned into a grey scale image. This step was necessary as the edge detect function can only read images in gray scale. Edge detect turned the image into a binary form identifying the boundary between the black and white within the image, returning an edge image as seen in Figure 2: Binary Image from Gray Scale and a matrix of the image values.

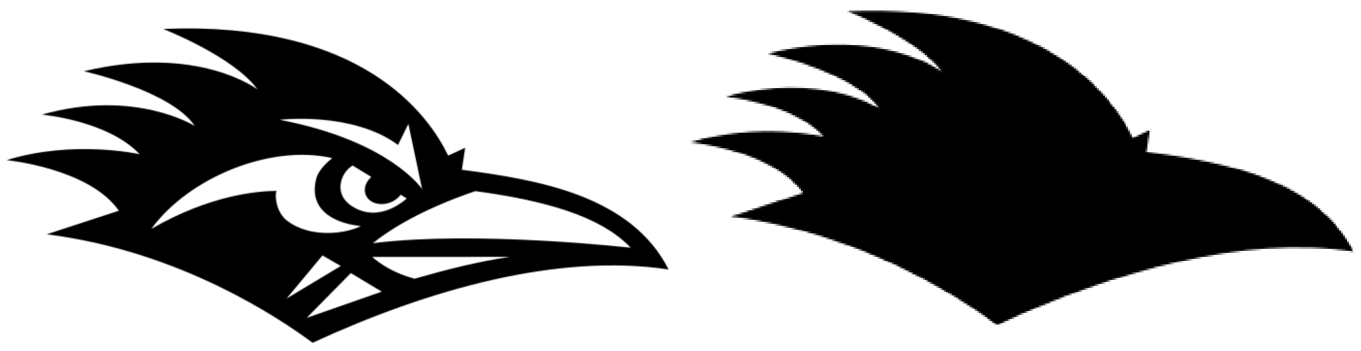


Figure 1: a) Black and White Logo (Left), b) Silhouette (Right)

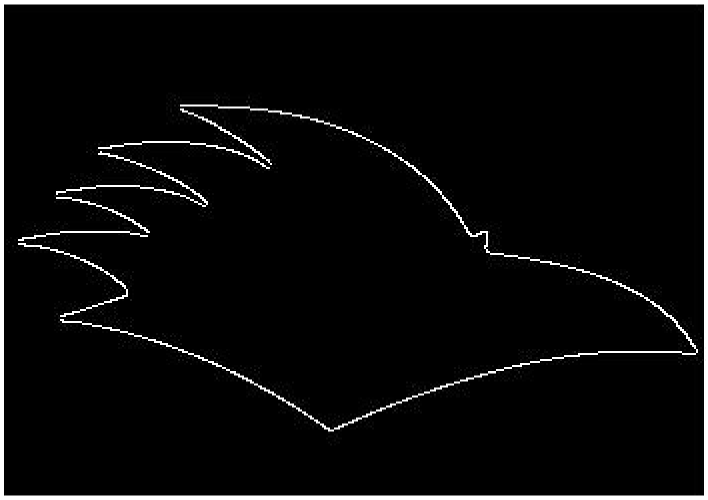


Figure 2: Binary Image from Gray Scale

The information on the binary matrix was then turned into x, y coordinate form. Once the data points were generated in MATLAB, an attempt at plotting the points was made to ensure that the points generated made sense. The plot created conformed to the silhouette of the original image. The points were exported to excel in a 1267x2 matrix. A scatter plot with straight lines was generated using the data and it was quickly observed that the data was arranged in an ascending domain order and was not centered about any axis since the numbers generated were based off the original size of the image. All the data points were translated by finding the average of the difference between the minimum and maximum and subtracting the initial offset from both the domain and range. Shown below in Figure 3 is how the data was initially interpreted.

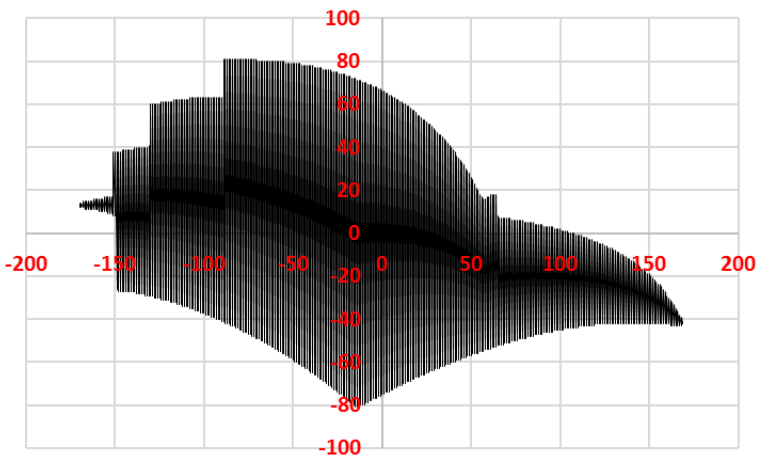


Figure 3- Initial Scatter Straight Line Plot

For the logo to be truly parameterized, a function needed to be generated which would plot the profile as shown in Figure 1b. The feathers towards the rear of Figure 3 are unrecognizable and the plot is being generated like that of a seismic wave. One option was to arrange each set of 1267 points by hand, but this would take a long time. Another option would be to filter the data in such a manner that for every data point on the domain, each range would be sorted. This proved to be difficult in Excel and did not yield the desired results.

The focus was then turned to the original data in MATLAB. In here the order of the points could be rearranged using the array indices. With a simple FOR loop the odd indices were taken and placed in the first half of a 1x1267 array while the even indices were placed on the second half of the array. This was done for both the X array and the Y arrays. The newly rearranged points were then plotted to check if the data was cleaned and resembled the outline of the logo. Once plotted it was discovered that only sections of the data plotted the outline correctly while other sections still oscillated. The feather sections where part of the most troublesome areas.

The points needed to be arranged in such a way that it would connect the data points to trace the outline of the edges. Rearranging the indices to remove the oscillation proved to be ineffective. It was decided that the next best possibility to clean the data into the correct sequence was to rearrange and plot based on the nearest neighbor. The smallest distance between the points needed to be taken and order the data to plot to the next nearest point. This however would have taken too much time to modify. Fortunately, a shared script was found to aid us in this regard. This script, by Tristan Ursell, contained a function that plotted random 2D points into a minimal nearest-neighbor closed contour. Using his function, *points2contour*, the data was finally cleaned up to plot the follow the outline in a clockwise direction.

Even with the clean data, finding the equation of a closed form data was incredibly difficult due to the simple definition of a continuous function; for every x in X there is exactly one element y such that the ordered pair (x, y) is contained in the subset defining the function f. At this point, a reference to the foundations of a popular curve was made to help understand how to arrange the data. It was observed that closed popular curves were generated in a time domain such that for every interval of time, there was a respective x and y value. This information helped in realizing that the data needed a third variable, t. Since t was arbitrary, the range of 0 to pi was defined for the purpose of this project. Defining the t variable did not solve the arrangement of data points directly, but indirectly provided a milestone achievement in the next step.

Since radians populate from quadrant 1-4 in a counter clockwise motion, it seemed best fit that the x and y coordinates followed this path as well. The x and y coordinates with respect to each quadrant were filtered in Excel. Each quadrant was isolated onto another sheet and depending on the quadrant, arranged with respect to counter clockwise motion. For example, in quadrant 1, the data was arranged from highest to lowest with respect to the domain. The 2nd quadrant proved to be the most difficult in filtering in that the feathers on the logo created several range values for a single domain. Also, the sweeping of the features required extra attention in order to ensure the joints of the feathers did not overlap, generated seismic type waves. As shown in Figure 4, the 2nd quadrant is separated into individual data sets for each feather due to the overlapping and of range values and required arrangement with respect to the counter clockwise requirements. Once all four quadrants of coordinates were arranged, a t value was assigned to each set of points from zero to pi in a step size equal to pi divided by the total number of data sets.

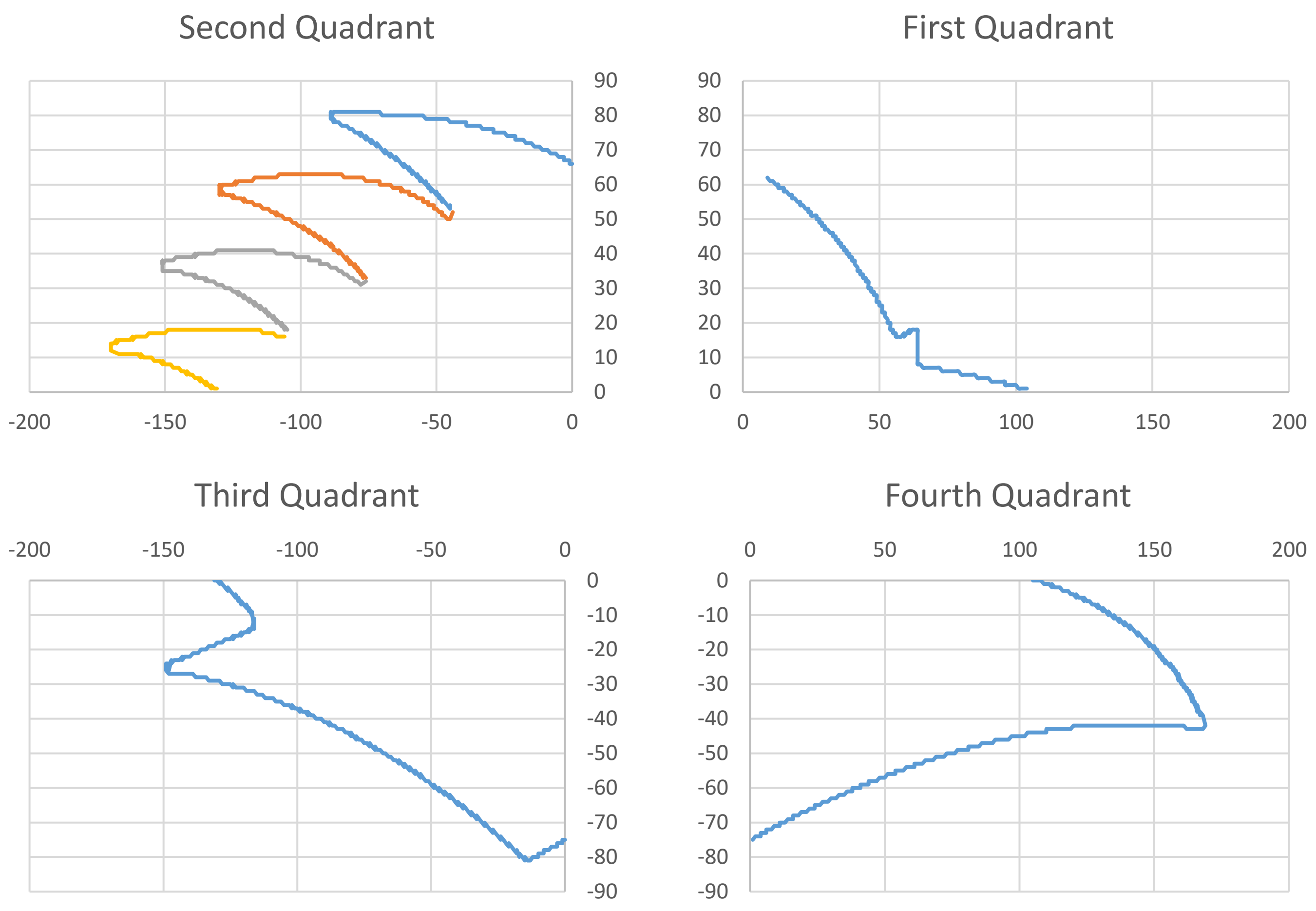
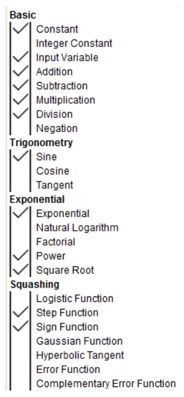


Figure 4- Quadrant Division

The data was now prepared for solving in Nutonian Eureqa. The dataset was imported without any additional smoothing or normalizing. Defining the solution criteria was difficult in that there was a balance between too few and too many parameters. Referencing the foundation of the popular curve, the key parameters of the function were selected as shown in Figure 5.


Basic
- ✓ Constant
- Integer Constant
- ✓ Input Variable
- ✓ Addition
- ✓ Subtraction
- ✓ Multiplication
- ✓ Division
- Negation

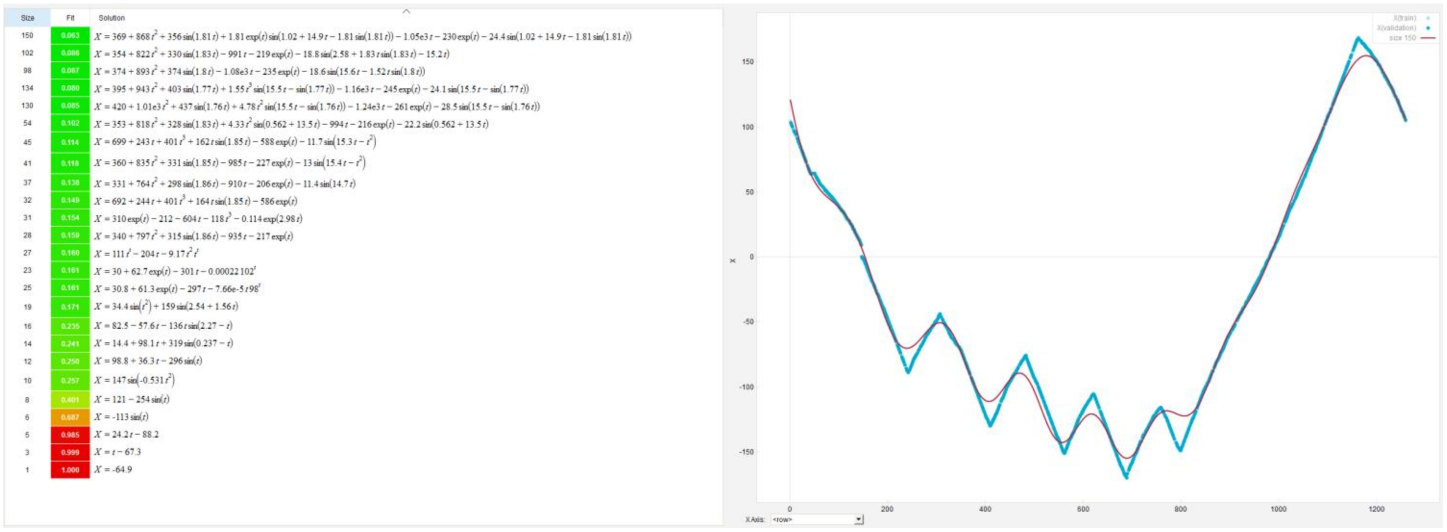
Trigonometry
- ✓ Sine
- Cosine
- Tangent

Exponential
- ✓ Exponential
- Natural Logarithm
- Factorial
- ✓ Power
- ✓ Square Root

Squashing
- Logistic Function
- ✓ Step Function
- ✓ Sign Function
- Gaussian Function
- Hyperbolic Tangent
- Error Function
- Complementary Error Function

Figure 5- Function Parameters

Several iterations were performed in solving equations for x(t) and y(t) with an average solving time of 6 hours per function. It was observed that after running the solver for at least 6 hours and then copying the dataset for another solving iteration, the next run would yield a higher fit resolution. It is unclear as to why the resolution improved when the new iteration did not reference the previous, but this proved to work every time. Figure 6 shows an example of the solver window looking for a function of x(t). The functions are arranged in order of best fit. It was observed that the best fit function did not always provide the most cosmetically pleasing plot, but often reduced the number of noise in the function in the form of corrective oscillations.



Size	Fit	Solution
150	0.063	$X = 369 + 868t^2 + 356\sin(1.81t) + 1.81\exp(t)\sin(1.02 + 14.9t - 1.81\sin(1.81t)) - 1.05e3\,t - 230\exp(t) - 24.4\sin(1.02 + 14.9t - 1.81\sin(1.81t))$
102	0.086	$X = 354 + 822t^2 + 330\sin(1.83t) - 991t - 219\exp(t) - 18.8\sin(2.58 + 1.83t\sin(1.83t) - 15.2t)$
98	0.087	$X = 374 + 893t^2 + 374\sin(1.8t) - 1.08e3\,t - 235\exp(t) - 18.6\sin(15.6t - 1.52t\sin(1.8t))$
134	0.080	$X = 395 + 943t^2 + 403\sin(1.77t) + 1.55t^3\sin(15.5t - \sin(1.77t)) - 1.16e3\,t - 245\exp(t) - 24.1\sin(15.5t - \sin(1.77t))$
130	0.085	$X = 420 + 1.01e3\,t^2 + 437\sin(1.76t) + 4.78t^2\sin(15.5t - \sin(1.76t)) - 1.24e3\,t - 261\exp(t) - 28.5\sin(15.5t - \sin(1.76t))$
54	0.102	$X = 353 + 818t^2 + 328\sin(1.83t) + 4.33t^2\sin(0.562 + 13.5t) - 994t - 216\exp(t) - 22.2\sin(0.562 + 13.5t)$
45	0.114	$X = 699 + 243t + 401t^3 + 162t\sin(1.85t) - 588\exp(t) - 11.7\sin\left(15.3t - t^2\right)$
41	0.118	$X = 360 + 835t^2 + 331\sin(1.85t) - 985t - 227\exp(t) - 13\sin\left(15.4t - t^2\right)$
37	0.138	$X = 331 + 764t^2 + 298\sin(1.86t) - 910t - 206\exp(t) - 11.4\sin(14.7t)$
32	0.149	$X = 692 + 244t + 401t^3 + 164t\sin(1.85t) - 586\exp(t)$
31	0.154	$X = 310\exp(t) - 212 - 604t - 118t^3 - 0.114\exp(2.98t)$
28	0.159	$X = 340 + 797t^2 + 315\sin(1.86t) - 935t - 217\exp(t)$
27	0.160	$X = 111t^t - 204t - 9.17t^2t^t$
23	0.161	$X = 30 + 62.7\exp(t) - 301t - 0.00022\,102^t$
25	0.161	$X = 30.8 + 61.3\exp(t) - 297t - 7.66e\text{-}5\,t\,98^t$
19	0.171	$X = 34.4\sin\left(t^2\right) + 159\sin(2.54 + 1.56t)$
16	0.235	$X = 82.5 - 57.6t - 136t\sin(2.27 - t)$
14	0.241	$X = 14.4 + 98.1t + 319\sin(0.237 - t)$
12	0.250	$X = 98.8 + 36.3t - 296\sin(t)$
10	0.257	$X = 147\sin\left(-0.531t^2\right)$
8	0.401	$X = 121 - 254\sin(t)$
6	0.687	$X = -113\sin(t)$
5	0.985	$X = 24.2t - 88.2$
3	0.999	$X = t - 67.3$
1	1.000	$X = -64.9$

Figure 6 - Nutonian Solver Window

Key deciding factors in determining if the solution provided was ideal included the size, fit, and R^2 or readiness of fit. The size of the solution is also known as the complexity of the equation. A higher size does not mean the solution is the best to choose from, but identifies mathematical complexity in selection criteria. The fit is a measurement of the accuracy to the dataset. As the fit approaches zero, the solution increases in accuracy. Inversely, the further the fit deviates from 0, the less accurate the solution becomes. The R^2 is used as a means of statistically measuring the model from the dataset. Since the dataset evaluated for the logo contained oscillations, often times Eureqa would fine the mean of the peaks and troughs rather than pick up the actual sinusoidal function.

Table 1 shows an example of three sets of data for each function. The data shows that as the fit of the solution approaches zero, the size and R^2 value really do not approach a limit either. As mentioned before, due to some of the oscillations in the datasets, the software was finding the mean of the small amplitudes rather than the actual sinusoidal fluctuation. An example of this can be seen in the second data set for both X(t) and Y(t) as the size reduced, so did the fit.

Function	Iteration	R^2	Size	Fit
X(t)	1	0.993425	130	0.073
	2	0.992013	50	0.081
	3	0.994574	150	0.063
Y(t)	1	0.997973	120	0.055
	2	0.996187	59	0.049
	3	0.996581	130	0.046

Table 1 - Criteria for Solution Selection

In order to improve the solution, more computation time would be required. Eureqa offers cloud based computation for a fee per core-hour. Buying cloud computing time can increase the ability of the solution fit up to 32 times better. For the purpose of this project, a Windows Surface Pro running Windows 10 with an i5 6300U 2.40GHz processor performed all the calculations.

Figure 7 shows an initial plot of an early function set. The figure shows poor definition in the rear feather regions and a lack of general shape. Figure 8 shows the final plot of the function set. The definition in the rear feathers definitely improved and the general shape is better recognizable.

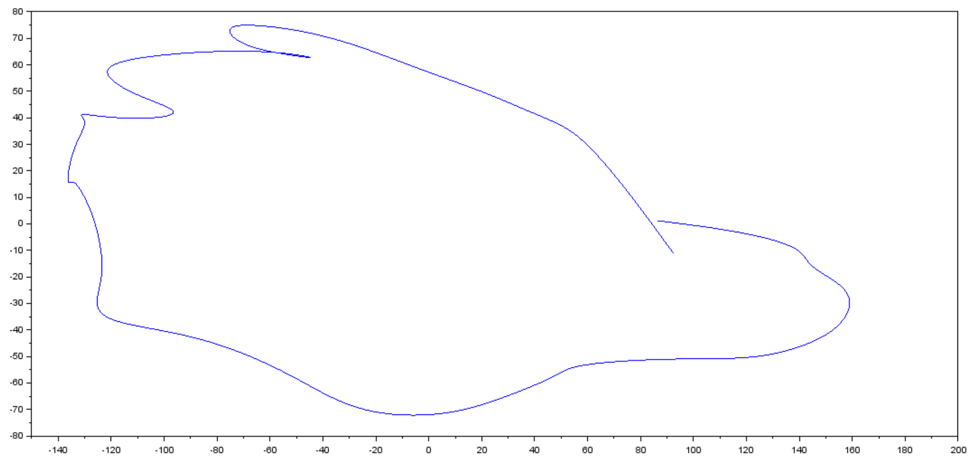


Figure 7- First Plot of Mascot

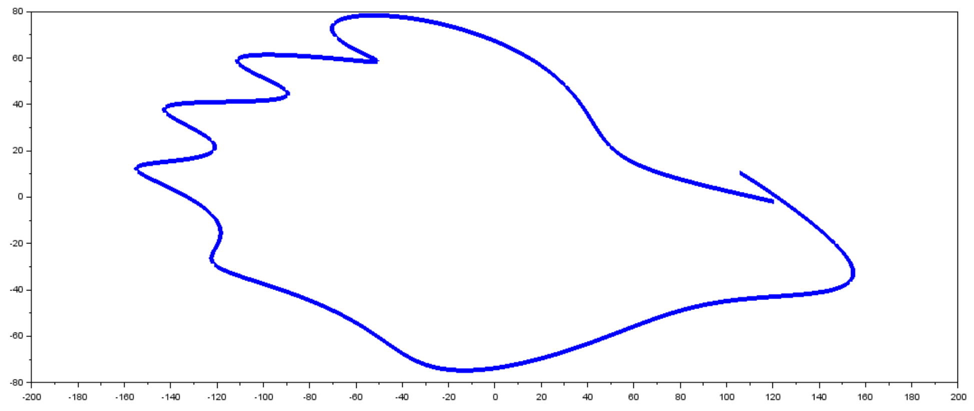


Figure 8- Final Plot of Mascot

Conclusion

To summarize, first a high-quality image of Rowdy was obtained and edited for simplicity in an image based software. This new image was then uploaded into MATLAB where it was turned into a gray scale and then into a binary image rendering a matrix with edge data. This data was then transformed into an X and Y matrix. The new points were cleaned up to plot in a contour of the original image. Form here the cleaned data was transferred to Microsoft Excel to further prepare the data for function finding. The data was re-centered and a time domain was introduced to make curve fitting possible. Once prepared the data was imported into Eureqa for data computation to curve fit. After more than 18 hours of number crunching by Eureqa, a final equation was found that approximated the data with the given time for project completion. Its approximation and equation can be seen below. With further exploring, computing power and time a cleaner form could be found.

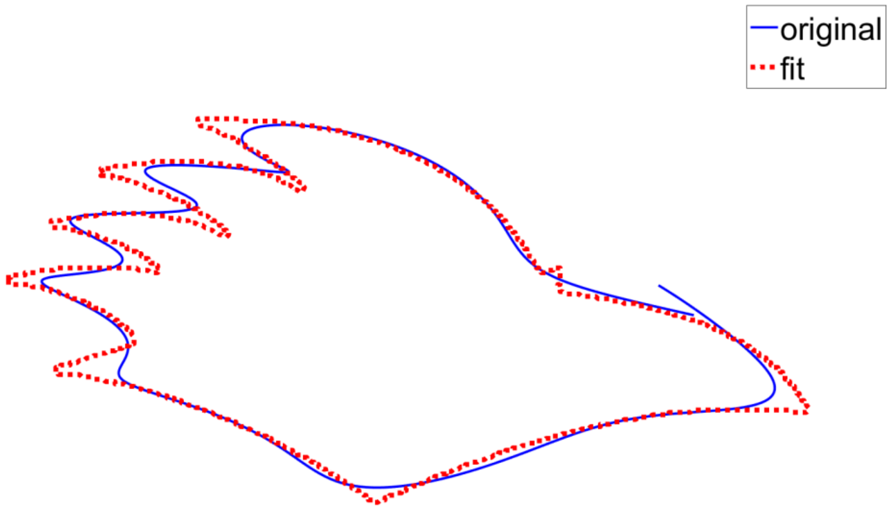


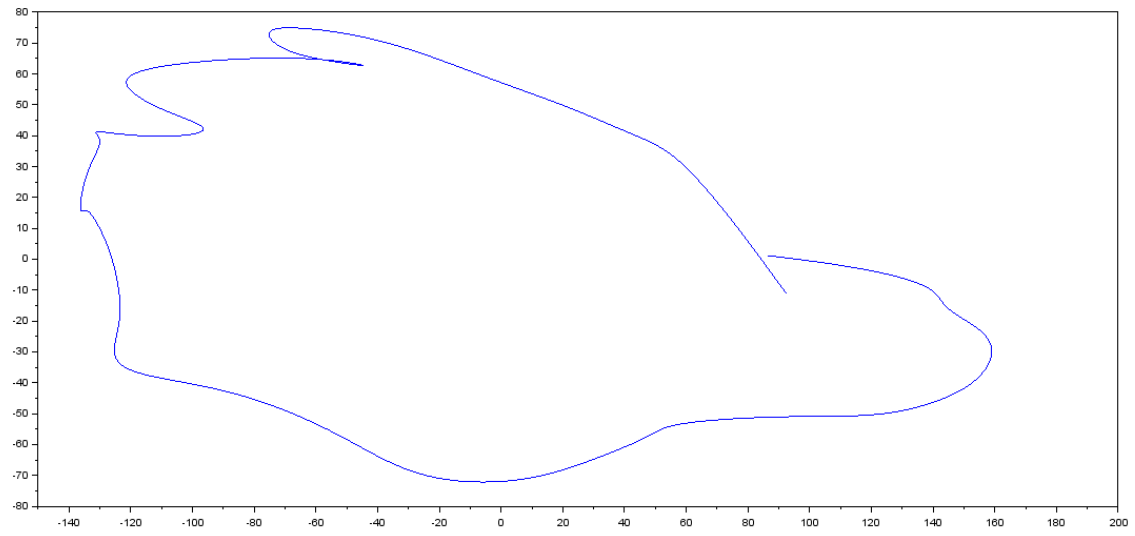
Figure 9: Original Data vs Fitted Curve

Equation 1: Final X(t) Equation

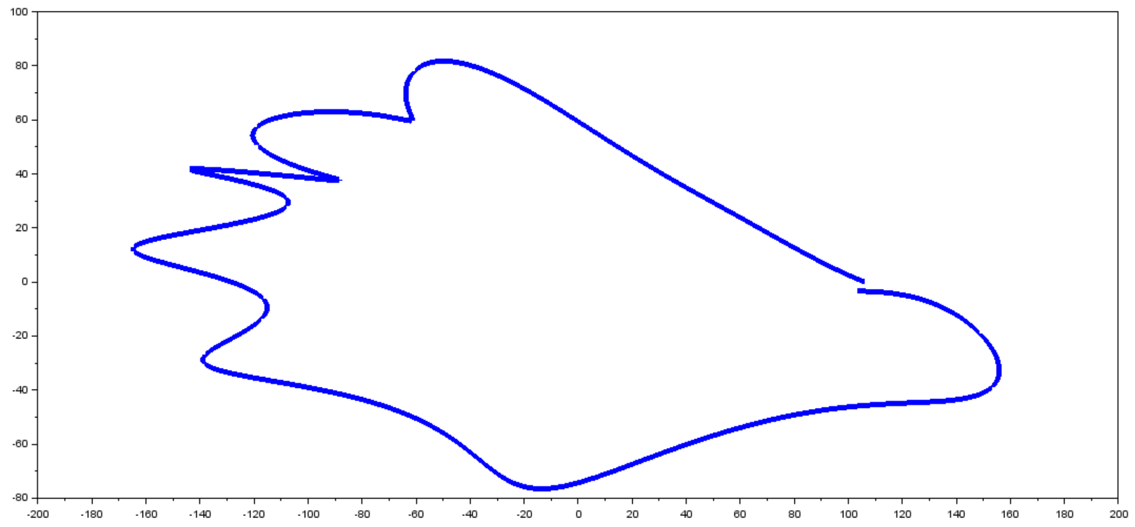
$$X = 369.289952366726 + 868.097877893622 * t^2 + 355.885451199134 * \sin(1.80851770543267 * t) + 1.80851770543267 * \exp(t) * \sin(1.01940232088457 + 14.9409026016358 * t - 1.80851770543267 * \sin(1.80851770543267 * t)) - 1050.6292596086 * t - 229.582690842127 * \exp(t) - 24.3720475907521 * \sin(1.01940232088457 + 14.9409026016358 * t - 1.80851770543267 * \sin(1.80851770543267 * t))$$

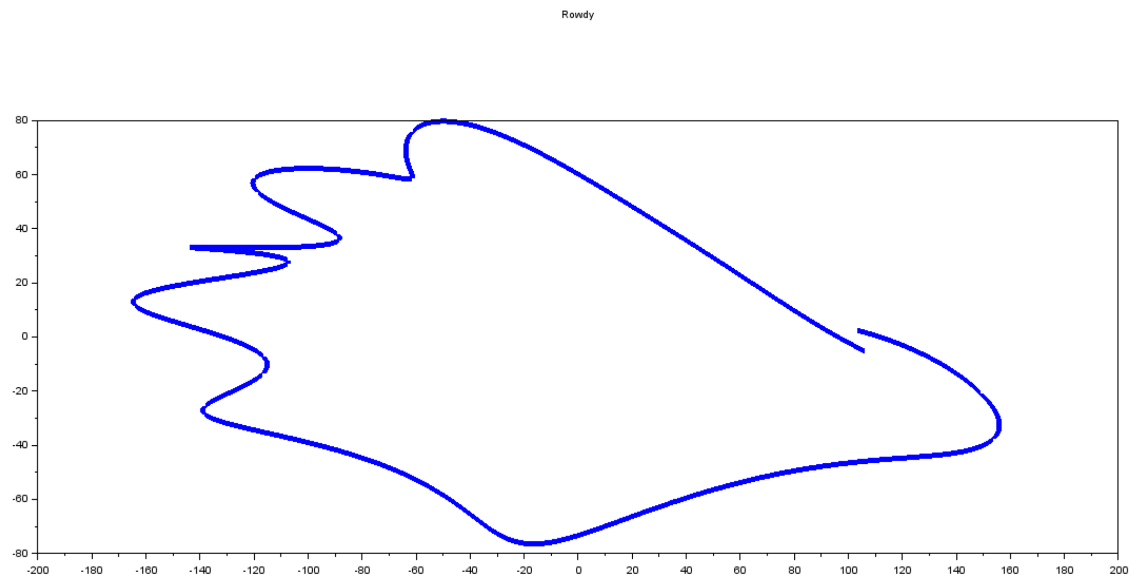
Equation 2: Final Y(t) Equation

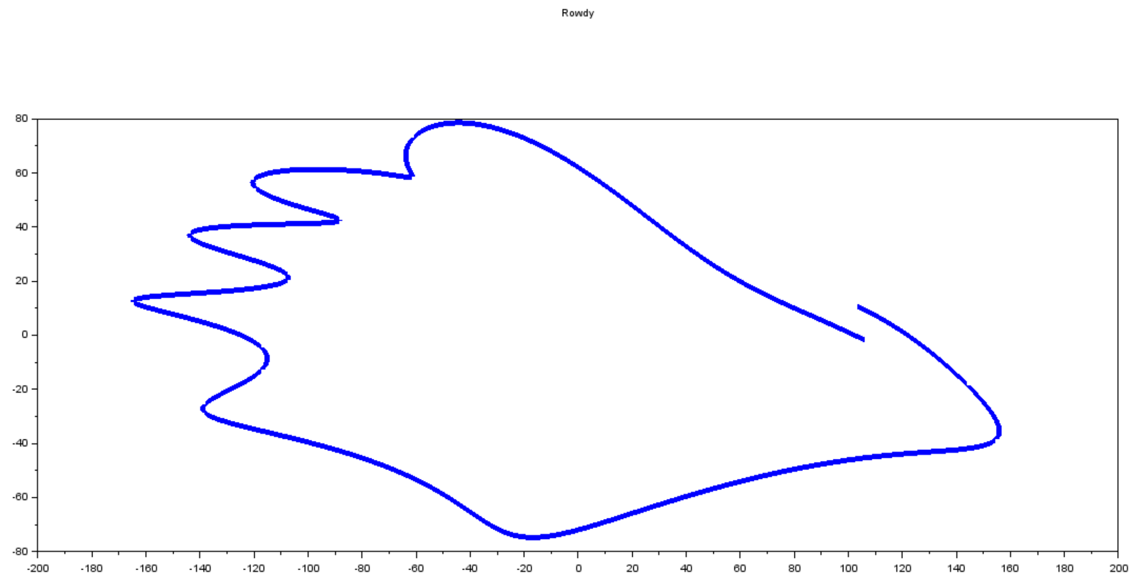
$$\begin{aligned}Y = {} & 197.771135039485 * \sin(t) + 12.2725932807796 * \sin(4.55931255905606 * t) \\ & + 0.000353251212376122 * \sin(t) * \exp(10.6716820645944 * \sin(t)) \\ & * \sin(5.64900764980555 + 11.4302916545718 * t) - 7.12454145678416 \\ & - 111.322673046081 * t * \sin(t) - 8.71761475813689 \\ & * \sin(5.64900764980555 + 11.4302916545718 * t)\end{aligned}$$


80
70
60
50
40
30
20
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-140
-120
-100
-80
-60
-40
-20
0
20
40
60
80
100
120
140
160
180
200

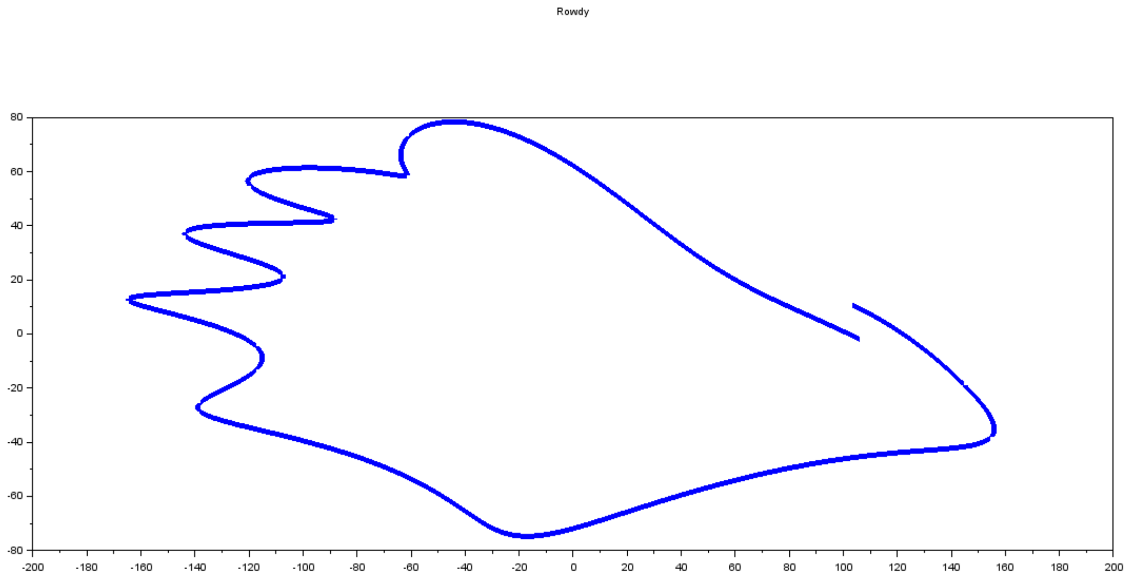
Rowdy

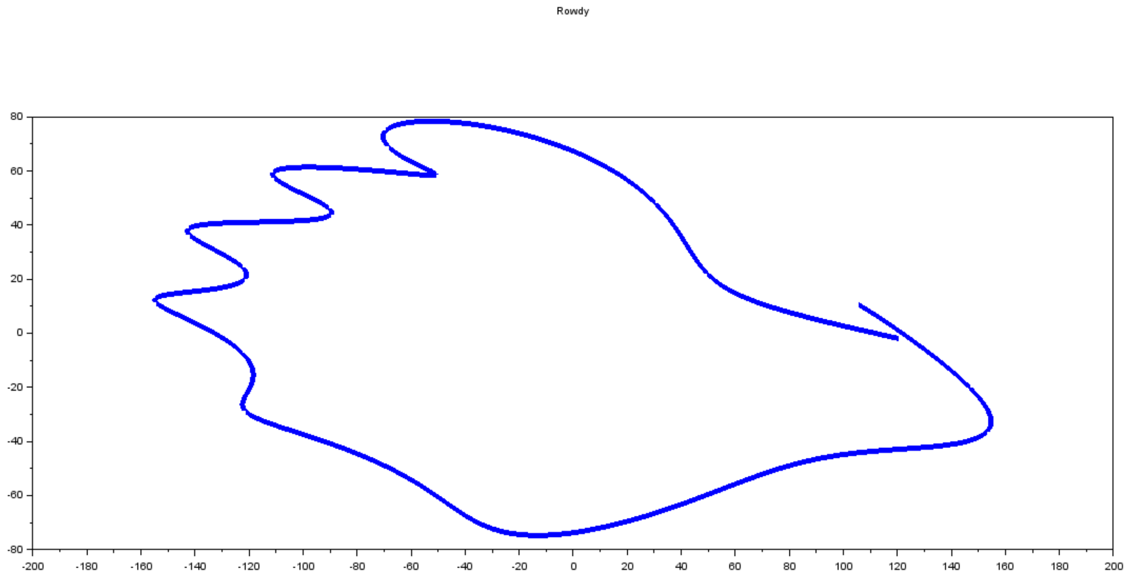

100
80
60
40
20
0
-20
-40
-60
-80
-200
-180
-160
-140
-120
-100
-80
-60
-40
-20
0
20
40
60
80
100
120
140
160
180
200



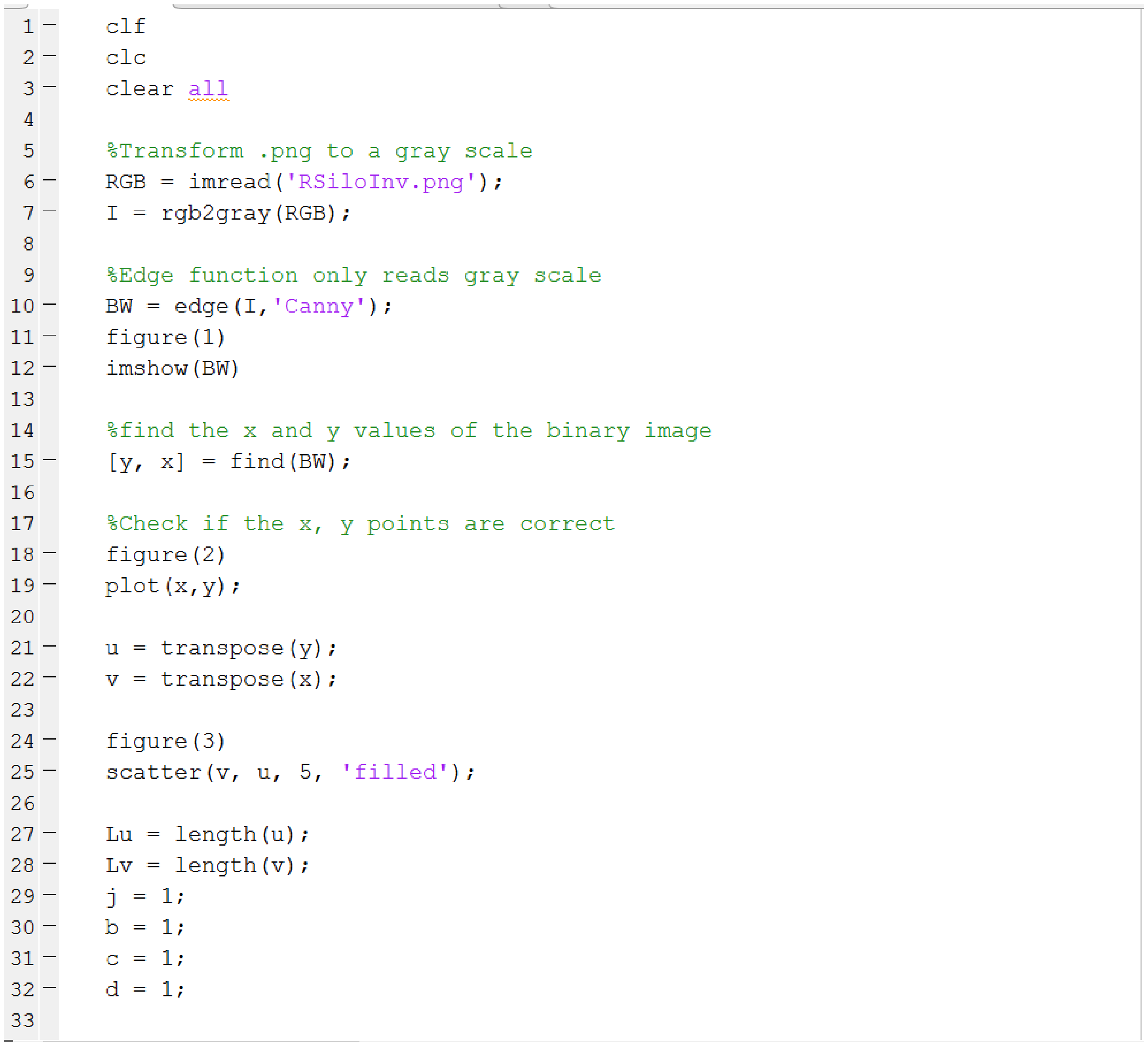


B

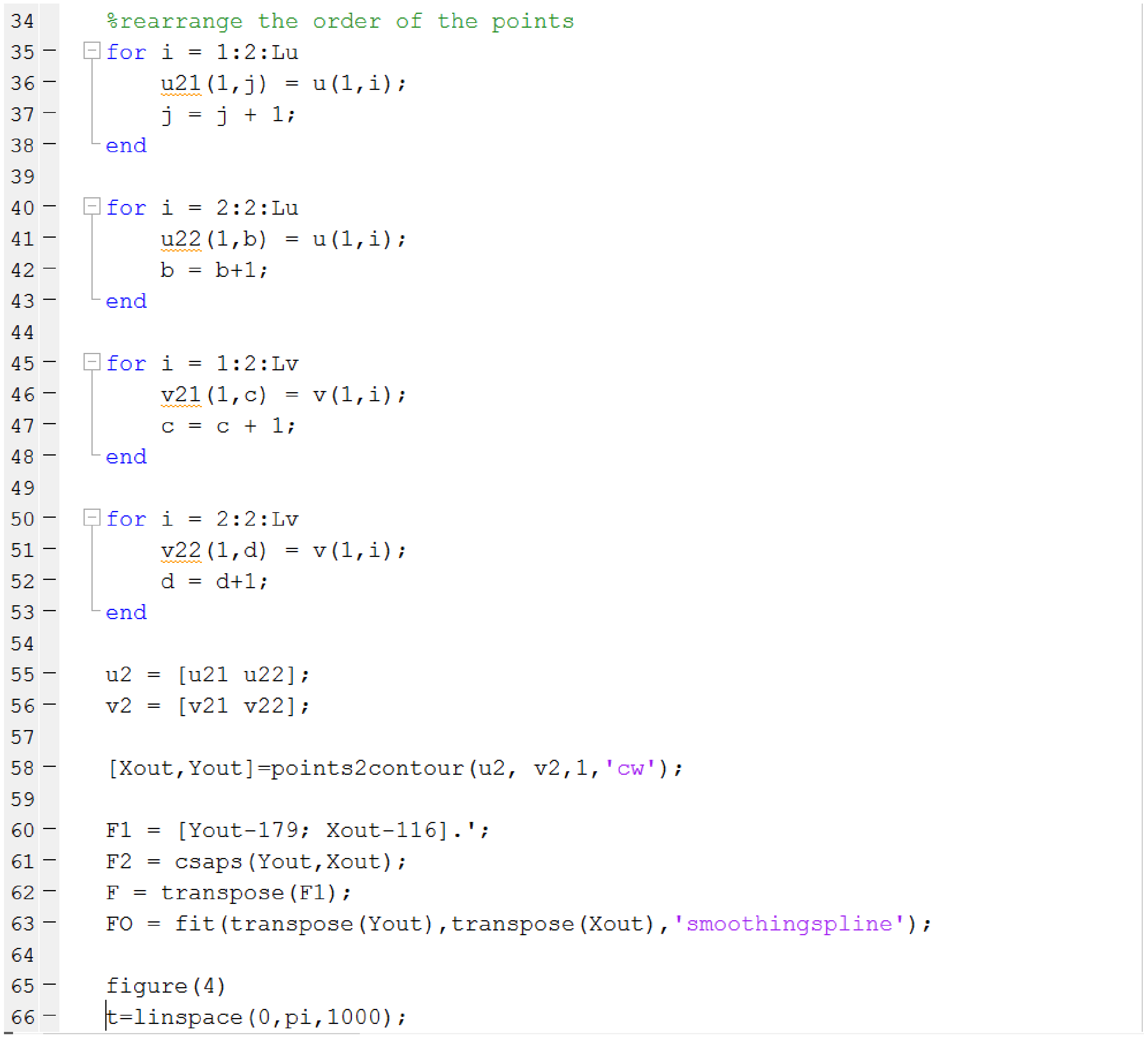

Rowdy
80
60
40
20
0
-20
-40
-60
-80
-200
-180
-160
-140
-120
-100
-80
-60
-40
-20
0
20
40
60
80
100
120
140
160
180
200


Rowdy
80
60
40
20
0
-20
-40
-60
-80
-200
-180
-160
-140
-120
-100
-80
-60
-40
-20
0
20
40
60
80
100
120
140
160
180
200

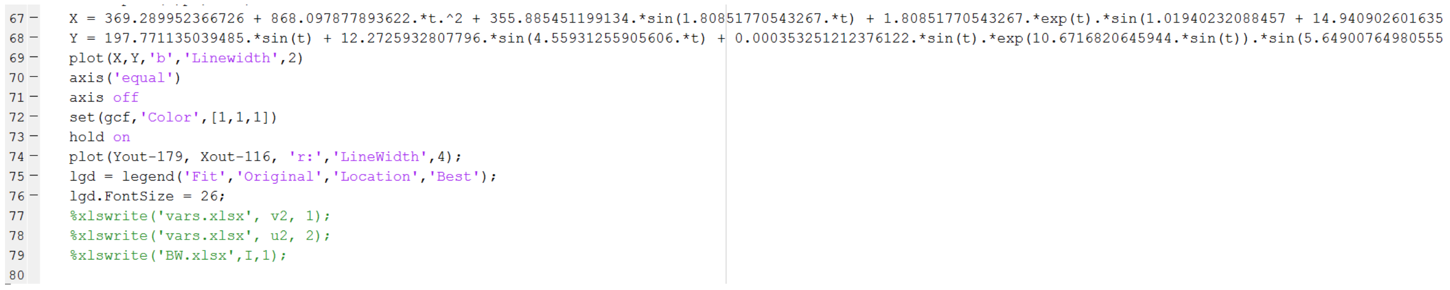
C



```matlab
clf
clc
clear all

%Transform .png to a gray scale
RGB = imread('RSiloInv.png');
I = rgb2gray(RGB);

%Edge function only reads gray scale
BW = edge(I,'Canny');
figure(1)
imshow(BW)

%find the x and y values of the binary image
[y, x] = find(BW);

%Check if the x, y points are correct
figure(2)
plot(x,y);

u = transpose(y);
v = transpose(x);

figure(3)
scatter(v, u, 5, 'filled');

Lu = length(u);
Lv = length(v);
j = 1;
b = 1;
c = 1;
d = 1;

```



```
%rearrange the order of the points
for i = 1:2:Lu
    u21(1,j) = u(1,i);
    j = j + 1;
end

for i = 2:2:Lu
    u22(1,b) = u(1,i);
    b = b+1;
end

for i = 1:2:Lv
    v21(1,c) = v(1,i);
    c = c + 1;
end

for i = 2:2:Lv
    v22(1,d) = v(1,i);
    d = d+1;
end

u2 = [u21 u22];
v2 = [v21 v22];

[Xout,Yout]=points2contour(u2, v2,1,'cw');

F1 = [Yout-179; Xout-116].';
F2 = csaps(Yout,Xout);
F = transpose(F1);
FO = fit(transpose(Yout),transpose(Xout),'smoothingspline');

figure(4)
t=linspace(0,pi,1000);
```



```
X = 369.289952366726 + 868.097877893622.*t.^2 + 355.885451199134.*sin(1.80851770543267.*t) + 1.80851770543267.*exp(t).*sin(1.01940232088457 + 14.940902601635
Y = 197.771135039485.*sin(t) + 12.2725932807796.*sin(4.55931255905606.*t) + 0.000353251212376122.*sin(t).*exp(10.6716820645944.*sin(t)).*sin(5.64900764980555
plot(X,Y,'b','Linewidth',2)
axis('equal')
axis off
set(gcf,'Color',[1,1,1])
hold on
plot(Yout-179, Xout-116, 'r:','LineWidth',4);
lgd = legend('Fit','Original','Location','Best');
lgd.FontSize = 26;
%xlswrite('vars.xlsx', v2, 1);
%xlswrite('vars.xlsx', u2, 2);
%xlswrite('BW.xlsx',I,1);
```

References

Inkscape 0.91

Microsoft Excel 2016

Nutonian Eureqa 1.24.0

Scilab 5-5-2

Trott, Michael. "Making Formulas… for Everything—From Pi to the Pink Panther to Sir Isaac Newton." *WolframAlpha Blog RSS*. WolfRamAlpha, 17 May 2013. Web. 09 Dec. 2016.