**Title:** **Exploring Interpolating Functions**

**Authors:** **William Franzen, Noah Harbor, Brandon Mitchell, Logan Reed**

## Abstract

This report summarizes the findings and results of various methods of deriving interpolating polynomials. In particular, Newton's interpolation method and the cubic spline interpolation method are explored. Several interpolating polynomials were generated based on points from a known function. In addition, several interpolating functions were generated using points from an image in an attempt to match the function to an edge of an object in the image. This showed the situations where these two interpolation methods may fail or produce extreme errors.

## Diff Function

```
# Netwon's Divided Difference
# Params:
#   x: list, x part of coordinates
#   y: list, y part of coordinates
# Return:
#   list, coefficients in the form [a0, a1, ... an]
# Description:
#   Uses Netwon's interpolation method to find the divided differences
def diff(x, y):

    if len(x) != len(y):
        raise ValueError("Inputs x and y must have the same length")

    n = len(y)
    coef = np.zeros([n, n])

    # The first column is y
    coef[:, 0] = y

    for j in range(1, n):
        for i in range(n - j):
            coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])

    # First row contains all coefficients we need
    return coef[0]
```

*Figure 1: Diff Function*

The diff function takes a list of x and y values as parameters. It throws an error if the length of the lists don't match. Afterwards, it iterates through the points, and it calculates the divided difference for each point. It then returns the array of the coefficients.

## Newton Function

```
# Newton's Polynomial Interpolation Function
# Params:
#   x: list, x part of coordinates
#   y: list, y part of coordinates
# Return:
#   function pointer, the interpolating function
# Description:
#   Uses Newton's interpolation method to create an interpolating function
def newtonPolynomialFunc(xVals, yVals):

    coef = diff(xVals, yVals)

    func = f"lambda x: {coef[0]}"
    part = ""

    for i in range(1, len(coef)):
        func += f" + {coef[i]}"
        part += f" * (x - {xVals[i - 1]})"
        func += part

    return eval(func)

# Newton's Interpolation
# Params:
#   x: list, x part of coordinates
#   y: list, y part of coordinates
# Return:
#   float, the value estimated at pointToEval
# Description:
#   Uses Newton's interpolation method to estimate the value at pointToEval
def newton(x, y, pointToEval):

    return newtonPolynomialFunc(x, y)(pointToEval)
```

*Figure 2 : Newton's Polynomial Function*

The newtonPolynomialFunc function takes a list of x and y values as parameters. It then finds the divided differences by calling the diff function on the x and y lists. Afterwards, it iterates through the points in the list, and appends them to an interpolating function. After this loop, the interpolating function is then returned.

The newton function uses the function returned from newtonPolynomialFunc, and passes a parameter to it (pointToEval). The point is estimated by checking the value at a point in the interpolating function, and then the value is returned.

## Cubic Spline Evaluation Function

```
# Evaluate a point using the cubic spline method
# Params:
#   pointToEval: float, the point to evaluate
#   x:, float list, the coordinate of the x points
#   coef: 2D float list, the return value of CubicNatural function
# Return:
#   float, the interpolated value of pointToEval
# Description:
#   Locates the correct range of pointToEval and then evaluates it using the
#   proper x value and coef terms
def evalCubicSpline(pointToEval, x, coef):
    if pointToEval < x[0] or pointToEval > x[-1]:
        raise ValueError("Point to evaluate out of range")

    # As long as points are in order, it is easy to find which range they are in
    for i in range(len(x) - 1):
        if pointToEval < x[i + 1]:
            break

    return coef[0][i] + coef[1][i] * (pointToEval - x[i]) + \
        coef[2][i] * (pointToEval - x[i]) ** 2 + coef[3][i] * (pointToEval - x[i]) ** 3
```

*Figure 3 : Cubic Spline Evaluation Method*

The evalCubicSpline function takes three parameters: a point to evaluate, a list of x points, and a 2D list of coefficients (which is the return value of the provided CubicNatural function). It throws an error if any of the x values are out of bounds.

It then iterates through the list of x points to find the interval that contains pointToEval. With this interval located, it estimates the value at pointToEval using the correct coefficient values and x values.

## Newton's Interpolation Results

To test the divided differences function and Newton's evaluation method above, the function $f(x) = 1/(1 + 25x^2)$ was approximate over the interval $[-1, 1]$. This function and the interval remained constant, but different points along the interval were chosen each time, which led to different results.

Part A requested that the data used be generated by $(xi, f(xi))$ where $i = 0, \ldots, 4$ and $xi = -1 + 0.5i$. Below is Figure 4, which shows these points, the original function for reference, the interpolating function generated from these points, and the error. As expected, the function has zero error at the points but the error varies elsewhere. As is typical with Newton's interpolation, the error is quite extreme near the endpoints. The error is less extreme, but still sizable near the center of the graph. Since the overall number of points is low, it makes sense that the error would be relatively high for most parts.

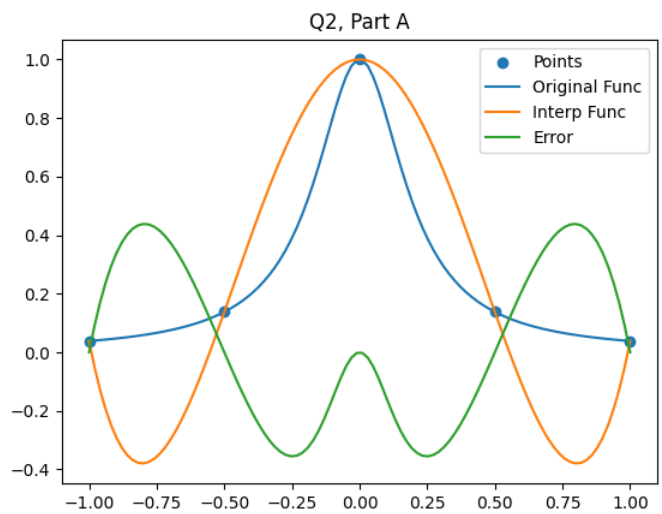


*Figure 4: Question 2, Part A Interpolating Function*

Part B requested that the data used be generated by $(xi, f(xi))$ where $i = 0, \ldots, 10$ and $xi = -1 + 0.2i$. Below is Figure 5, which shows these points, the original function for reference, the interpolating function generated from these points, and the error. This interpolating performs better interpolating values near the center. This likely has to do with the increase in points. However, the error is far more extreme near the edges than with the previous part. This makes sense as certain interpolation errors may increase as the number of nodes increase.

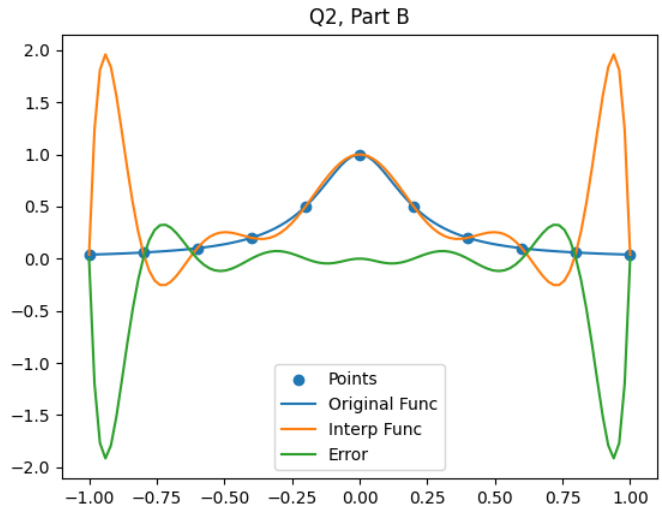


*Figure 5: Question 2, Part B Interpolating Function*

Part C requested that the data used be generated by $(xi,\ f(xi))$ where $i = 0, \ldots, 10$ and $xi = \ -cos((2i + 1)/22 * pi)$. Below is Figure 6, which shows these points, the original function for reference, the interpolating function generated from these points, and the error. Note that the points used are not evenly spaced. This interpolating function performs the best of three. The error near the center is more than the second interpolating function but less than the first. Of the three, this interpolating function has the least amount of error near the edges. It seems that interpolation errors are bad when dealing with evenly spaced nodes. Since these points are not evenly spaced, the interpolation errors in the previous two functions are less severe.

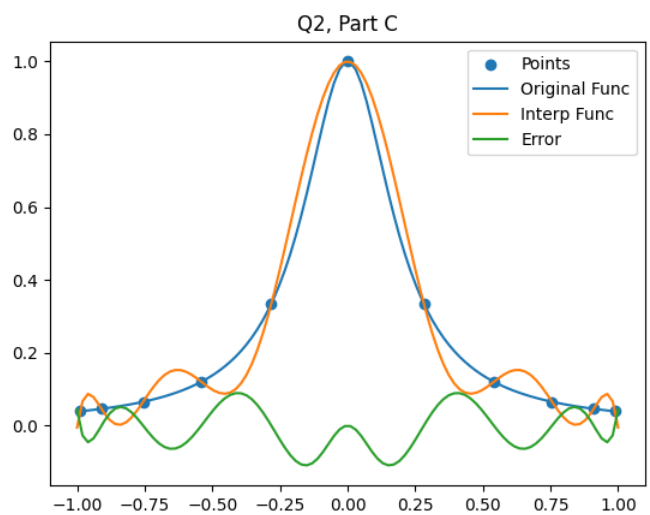


*Figure 6: Question 2, Part C Interpolating Function*

## Cubic Spline Interpolation Results (Camel)

Below are several figures showcasing the camel image (Figure 7), the spline generated from the camel's back (Figure 8), and a version with the two images overlaid (Figure 9).

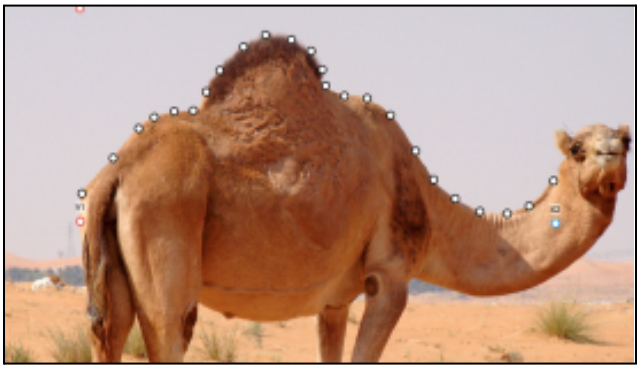

*Figure 7: Camel With Data Points*

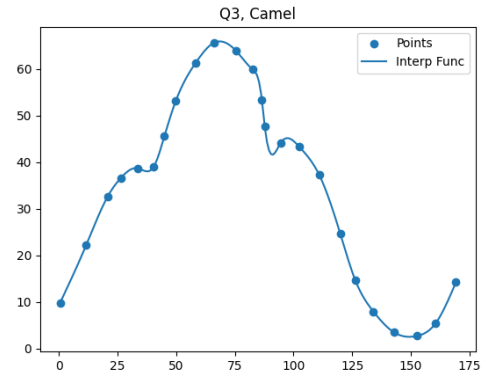


*Figure 8: Graphical Representation of Camel*

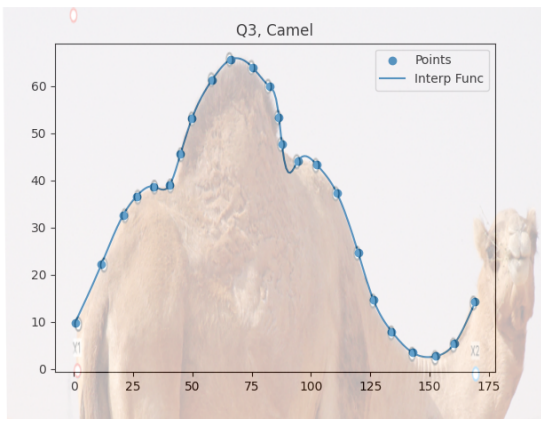


*Figure 9: Camel Graphical Overlay*

The camel spline is a relatively smooth function as the camel's back doesn't have any significant vertical changes. However, there is a small spike near the right side of the camel's hump. This is likely due to the fact that the hump is nearing vertical in that area, so a small spike is needed so the interpolating function can correct itself. An extra point in that area may be able to correct this error.

Below in Table 1 is part of the cubic spline that makes up the camel's back. Only a few parts of the spline are shown for brevity, but the entire spline is in the attached "camel spline.csv" file.

| Si | a | b | c | d |
|---|---|---|---|---|
| S1 | 9.806774 | 1.070384 | 0 | 0.000345 |
| S2 | 22.2837 | 1.200197 | 0.011587 | -0.00202 |
| S3 | 32.68829 | 0.904203 | -0.04393 | 0.001535 |
| S4 | 36.61425 | 0.558493 | -0.01839 | -0.00256 |
| S5 | 38.73146 | -0.09913 | -0.07334 | 0.013709 |
| | | … | | |
| S19 | 14.68772 | -1.27165 | 0.073822 | -0.00284 |
| S20 | 7.971287 | -0.63952 | 0.008177 | 0.000776 |
| S21 | 3.493347 | -0.31296 | 0.028765 | -0.00048 |
| S22 | 2.730311 | 0.110594 | 0.014598 | 0.001814 |
| S23 | 5.393773 | 0.678186 | 0.057457 | -0.00219 |

*Table 1: Camel Spline Values*

## Cubic Spline Interpolation Results (Our Images)

Using the same cubic spline interpolation as with the camel, four different images were chosen and had points selected along their edge. Each data point used needed to have a distinct x-coordinate. Additionally, each successive point had to have a larger x-value than the previous.

The first one chosen was a cat stretching on a tree.

*Figure 10: Stretching Cat With Data Points*

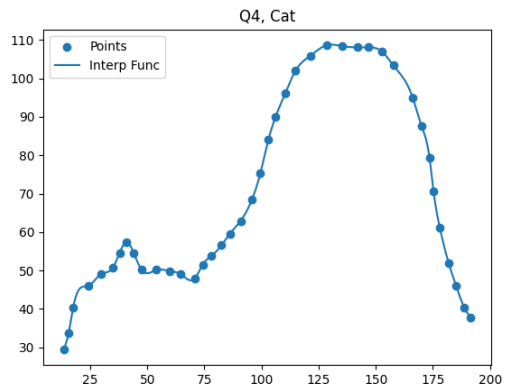


*Figure 11:Graphical Interpretation of Stretching Cat*

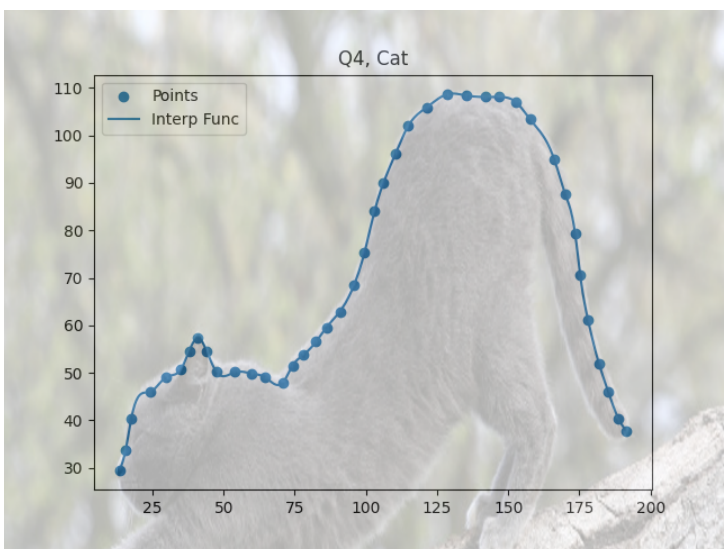


*Figure 12: Stretching Cat Graphical Overlay*

Although the y-scale is off, the function that results is quite smooth. This is likely due to the cat's back not having any large jumps that might cause the cubic spline to misinterpret what the original function was.

Below in Table 2 is part of the cubic spline that makes up the cat's back. Only a few parts of the spline are shown for brevity, but the entire spline is in the attached "cat spline.csv" file.

| Si | a | b | c | d |
|---|---|---|---|---|
| S1 | 29.37644 | 1.529042 | 0 | 0.092975 |
| S2 | 33.81062 | 2.913279 | 0.621369 | -0.20015 |
| S3 | 40.46189 | 3.019683 | -0.56763 | 0.035896 |
| S4 | 46.00462 | 0.242299 | 0.152056 | -0.01722 |
| S5 | 49.05312 | 0.366821 | -0.12919 | 0.023117 |
| | | … | | |
| S36 | 70.66975 | -4.72002 | 0.654097 | -0.07036 |
| S37 | 61.24711 | -2.72294 | 0.079376 | 0.001872 |
| S38 | 51.82448 | -2.00612 | 0.101622 | -0.01251 |
| S39 | 46.00462 | -1.74066 | -0.01913 | 0.017277 |
| S40 | 40.46189 | -1.25081 | 0.160483 | -0.01801 |

*Table 2: Cat Spline Values*

The second image that was chosen was a tree.

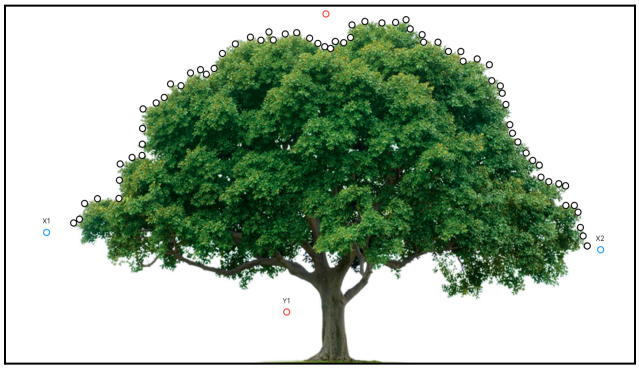

*Figure 13: Tree With Data Points*

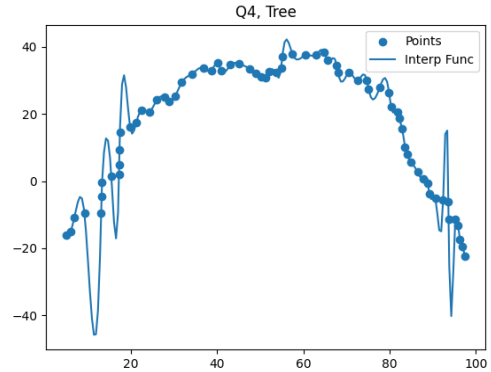


*Figure 14: Graphical Representation of Tree*

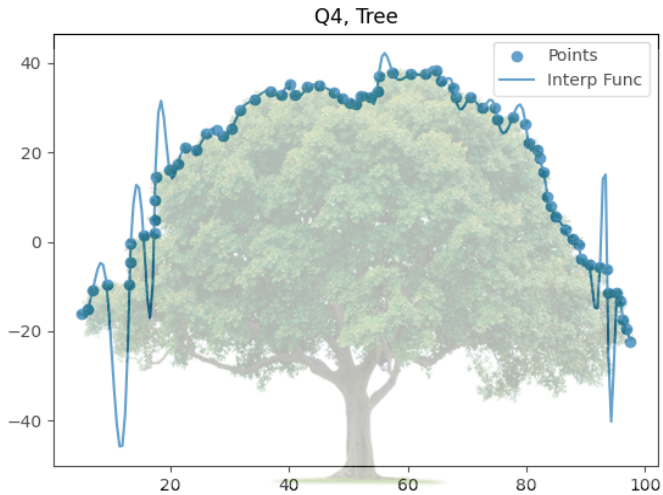


*Figure 15: Tree Graphical Overlay*

The resulting graph of cubic spline functions is not nearly as smooth as that of the previous image. There are a couple large oscillations near both edges of the image. One possible explanation for these oscillations is that in each instance there is a relatively large vertical change between points with only a small horizontal change.

Below in Table 3 is part of the cubic spline that makes up the top of the tree. Only a few parts of the spline are shown for brevity, but the entire spline is in the attached "tree spline.csv" file. One thing of note is that the c and d values of the spline are often very extreme in the areas where large spikes are seen. For example, S10 has a d value of -8220.13 and it lies on a more vertical part of the interpolating function. Similar behavior can be seen elsewhere.

| Si | a | b | c | d |
|---|---|---|---|---|
| S1 | -16.0873 | 0.058741 | 0 | 0.804319 |
| S2 | -15.0798 | 2.746514 | 2.54666 | -0.60497 |
| S3 | -10.848 | 5.902313 | 0.870627 | -1.3593 |
| S4 | -9.57179 | -12.1911 | -8.6337 | 3.046246 |
| S5 | -9.50462 | 57.84517 | 26.7317 | -673.229 |
| S6 | -4.66835 | 46.92446 | -150.901 | 349.2527 |
| S7 | -0.30227 | 25.34475 | -12.676 | 0.598737 |
| S8 | 1.444163 | -20.9438 | -8.80558 | 11.41828 |
| S9 | 1.981524 | 58.65799 | 52.95548 | 4005.59 |
| S10 | 5.004194 | 86.55546 | 581.4136 | -8220.13 |
| S11 | 9.235932 | 90.00076 | -503.07 | 1786.037 |
| S12 | 14.47523 | 42.95682 | -31.8218 | 5.903165 |
| | | … | | |
| S60 | -3.8623 | -8.68457 | 14.93136 | -6.65889 |
| S61 | -5.07137 | -6.21996 | -13.1799 | 9.13177 |
| S62 | -5.60873 | 35.30902 | 36.2134 | -52.9872 |
| S63 | -6.1461 | -116.517 | -159.518 | 2272.278 |
| S64 | -11.3854 | -117.364 | 140.2641 | -41.5888 |
| S65 | -11.3854 | 18.84243 | -51.7687 | 28.85811 |
| S66 | -13.199 | -11.7844 | 5.339056 | 0.615515 |
| S67 | -17.2964 | -6.73153 | 6.151084 | -4.125 |
| S68 | -19.5802 | -3.67628 | 0.165059 | -0.0736 |

*Table 3: Tree Spline Values*

The third image chosen was two cats mimicking an iconic panel from *JoJo's Bizarre Adventure*.

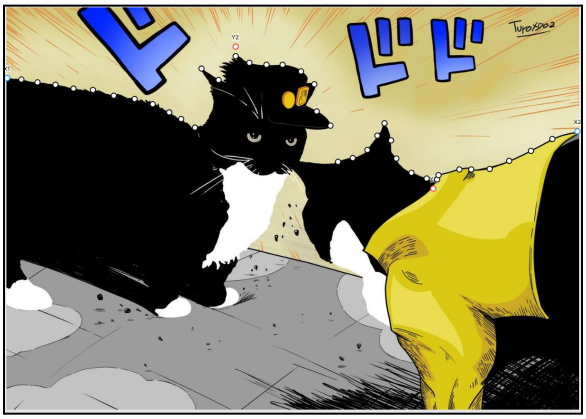

*Figure 16: JoJo Cats With Data Points*

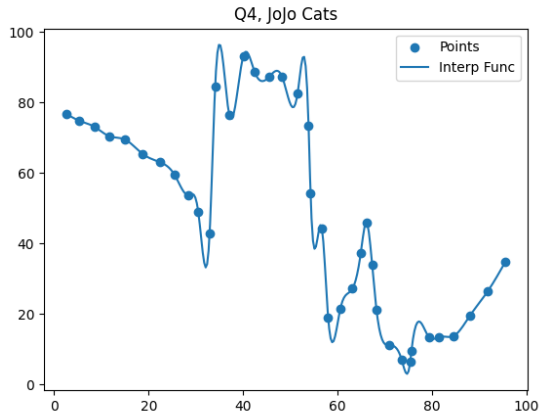


*Figure 17: Graphical Representation of JoJo Cats*

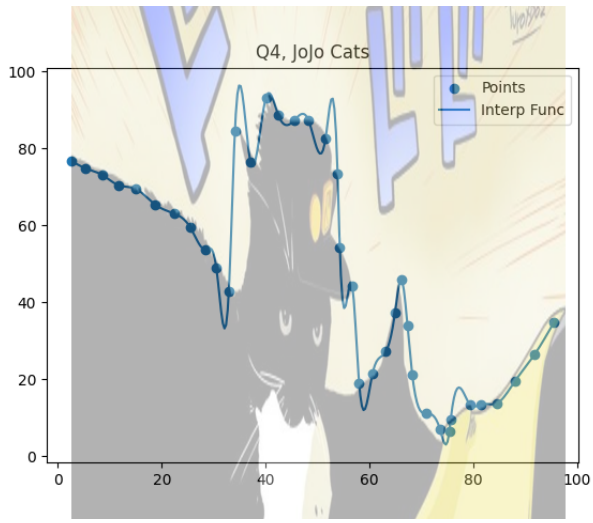


*Figure 18: JoJo Cats Graphical Overlay*

While the right cat is relatively smooth with its left ear and back being easily identifiable, there is an unexpected jump though where the graph hits his jacket collar causing it to overshoot on the next hump.

The left cat is a little less recognizable, and the graph covering its back portion is jagged thanks to too many points along its fur. This could be smoothened by reducing the amount of points in that area. When the graph reaches its head it becomes a bit of a mess, but its ear and hat are somewhat identifiable.

Below in Table 4 is part of the cubic spline that makes up the JoJo cats. Only a few parts of the spline are shown for brevity, but the entire spline is in the attached "jojo cats spline.csv" file. Similar to the tree, the c and d values of the spline increase in magnitude dramatically around the spikes. S11, S19, and S31 all have extreme values and are near vertical jumps.

| Si | a | b | c | d |
|---|---|---|---|---|
| | | … | | |
| S8 | 59.39597 | -2.27655 | -0.50323 | 0.210271 |
| S9 | 53.69128 | 0.134837 | 1.332058 | -1.19567 |
| S10 | 48.99329 | -9.81989 | -6.12227 | 3.798332 |
| S11 | 42.95302 | 26.88136 | 21.34696 | -11.7907 |
| S12 | 84.56376 | 28.66114 | -19.8176 | 3.121347 |
| S13 | 76.51007 | -6.0915 | 8.204449 | -1.43533 |
| S14 | 93.28859 | 4.451499 | -4.68133 | 0.84056 |
| S15 | 88.5906 | -3.46456 | 1.397533 | -0.13022 |
| S16 | 87.24832 | 1.401333 | 0.228485 | -0.25628 |
| S17 | 87.24832 | -3.44843 | -1.94444 | 0.790014 |
| S18 | 82.55034 | 8.85309 | 5.738988 | -4.81851 |
| S19 | 73.48993 | -42.742 | -27.9064 | 48.16225 |
| S20 | 54.36242 | -40.9798 | 32.14625 | -7.0019 |
| S21 | 44.2953 | -8.0618 | -18.491 | 7.620233 |
| S22 | 18.79195 | -16.8093 | 11.91396 | -1.93512 |
| | | … | | |
| S29 | 11.07383 | -0.0013 | -0.02471 | -0.20447 |
| S30 | 7.04698 | -4.47318 | -1.65643 | 2.133558 |
| S31 | 6.375839 | 10.8746 | 10.04888 | -20.4188 |
| S32 | 9.395973 | 12.07706 | -5.22701 | 0.608623 |
| | | … | | |

*Table 4: JoJo Cat Spline Values*

The fourth image is a render of Shadow the Hedgehog from the 2011 game, *Sonic & Sega All-Stars Racing*. Shadow is riding his motorcycle, the Dark Rider.

*Figure 19: Shadow the Hedgehog With Data Points*

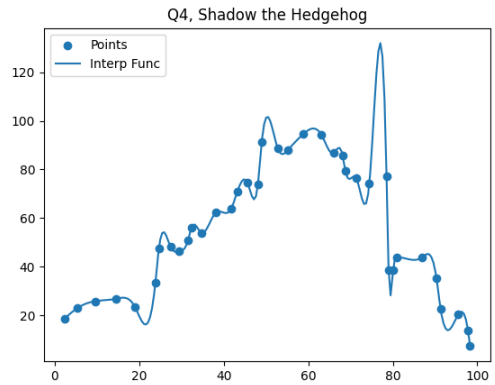


*Figure 20: Graphical Representation of Shadow*

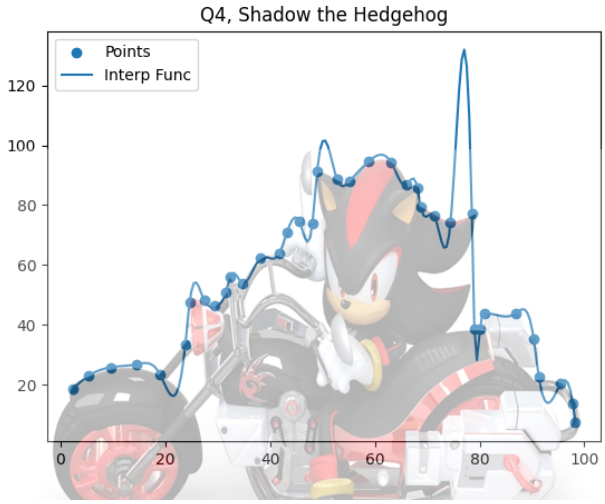


*Figure 21: Shadow the Hedgehog Graphical Overlay*

The graph has some areas that are identifiable from the original image, but there are some areas where the graph has spikes.

The left half of the points clearly show the outline of the Dark Rider, with a small spike near the point with the headlight. When the points get to Shadow's ear, there are a few more spikes in the graph. The next few points show the outline of Shadow's head.

There is then a large spike after Shadow's quill, which is caused by a small x difference and large y difference between the quill and the next part of the Dark Rider. The final points after this spike are more accurate to the original image.

Below in Table 5 is part of the cubic spline that makes up Shadow. Only a few parts of the spline are shown for brevity, but the entire spline is in the attached “shadow spline.csv” file. Like with the tree and the JoJo cats, the c and d values increase dramatically in the areas surrounding a large vertical jump or a spike in the graphed interpolating function.

| Si | a | b | c | d |
|---|---|---|---|---|
| S1 | 18.51107 | 1.57196 | 0 | -0.01468 |
| S2 | 22.93763 | 1.151007 | -0.13614 | 0.005388 |
| S3 | 25.75453 | 0.299402 | -0.06908 | 0.009764 |
| S4 | 26.76056 | 0.347929 | 0.078694 | -0.07388 |
| S5 | 23.54125 | -3.2397 | -0.89521 | 0.404028 |
| S6 | 33.4004 | 16.90831 | 5.022198 | -5.56576 |
| S7 | 47.48491 | 14.02655 | -8.56387 | 1.291374 |
| S8 | 48.08853 | -3.70677 | 2.153798 | -0.37813 |
| S9 | 46.27767 | 0.361681 | -0.15375 | 0.480634 |
| S10 | 50.90543 | 6.164423 | 2.896657 | -3.15979 |
| S11 | 56.13682 | 4.602362 | -4.81643 | 1.040224 |
| S12 | 53.92354 | -1.14599 | 2.293316 | -0.36624 |
| | | … | | |
| S22 | 94.16499 | -3.03987 | -0.98419 | 0.398085 |
| S23 | 86.72032 | 1.040146 | 2.416862 | -1.45984 |
| S24 | 85.91549 | -8.33457 | -6.84822 | 6.84648 |
| S25 | 79.47686 | -8.54715 | 6.521638 | -1.41496 |
| S26 | 76.45875 | -3.36379 | -4.5309 | 1.74237 |
| S27 | 74.24547 | 18.58941 | 11.63103 | -3.93721 |
| S28 | 77.26358 | -82.2723 | -36.4228 | 86.05787 |
| S29 | 38.43058 | -56.3024 | 89.61797 | -34.4031 |
| S30 | 38.43058 | 17.80952 | -19.5537 | 7.192017 |
| S31 | 43.86318 | 0.091315 | -0.24236 | 0.037732 |
| | | … | | |

*Table 4: JoJo Cat Spline Values*