CSE 276C Homework 2

Adrian L. Pavlak
U09830551

31. October 2023

1 Problem 1

1.1

Plot the ground truth trajectory.

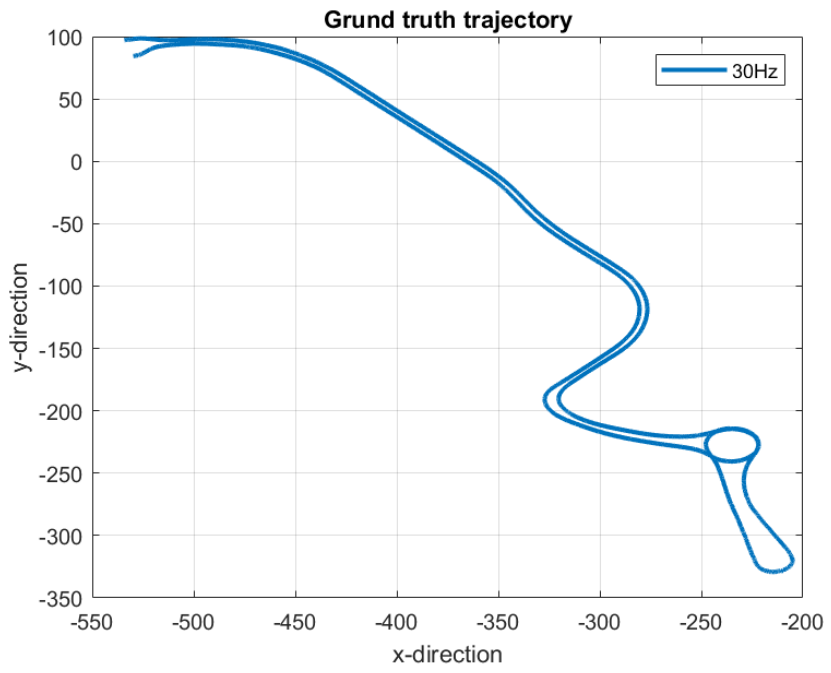

Figure 1: Plot of the ground truth trajectory.

1.2

Consider scenarios in which vehicle's sensors instead recorded at only 10Hz, 1Hz, and 0.2Hz. Downsample waypoints.csv to each sample rate to simulate the observed waypoints in each case. For each test sample rate:

(a) **Linear Interpolation:** Use linear interpolation to approximate the additional waypoints recorded at 30Hz. Plot the interpolated trajectory and report cumulative error.

(b) **Quadratic Polynomial Interpolation:** Use quadratic polynomial interpolation to approximate the additional waypoints recorded at 30Hz. Plot the interpolated trajectory and report cumulative error.

(c) **Cubic Spline Interpolation:** Use cubic spline interpolation to approximate a 30Hz sample rate. Plot the interpolated trajectory and report cumulative error.

I implemented the interpolation in MATLAB and the attached code is here 1.4. The code involved downsampling the data using the MATLAB function A(start: stepsize: end). I also created time arrays and helper variables, such as N10, N1, and N02, to manage the array lengths efficiently.

For linear interpolation, I utilized MATLAB's interp1 function with a linear method. For quadratic interpolation, I generated polynomials for sets of three points from the downsampled (e.g., (1,2,3), (2,3,4), (3,4,5), and so on) and interpolated the values using a 30 Hz time step within each interval.

Similarly, for cubic spline interpolation, I employed MATLAB's interp1 function with the cubic spline method. In all three methods, extrapolation was applied to the final section to match the size of the original 30 Hz data.

To assess the accuracy of each method, I calculated the error by summing the norm differences between the interpolated data and the 30 Hz dataset.

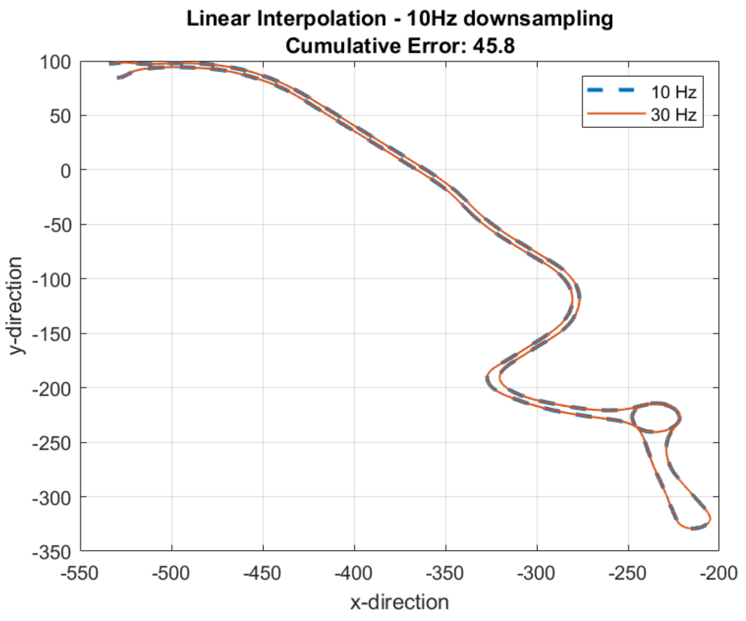

Figure 2: Linear Interpolation 10Hz Downsample.

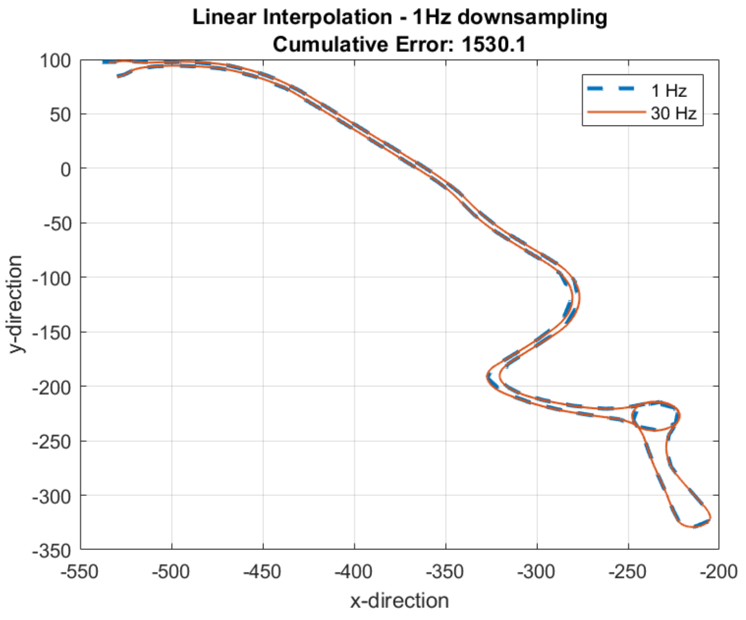

Figure 3: Linear Interpolation 1Hz Downsample.

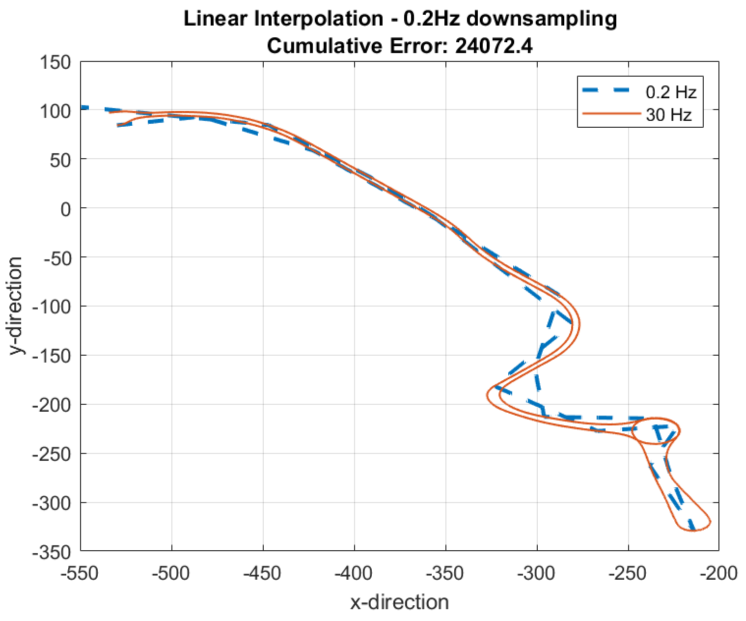

Figure 4: Linear Interpolation 0.2Hz Downsample.

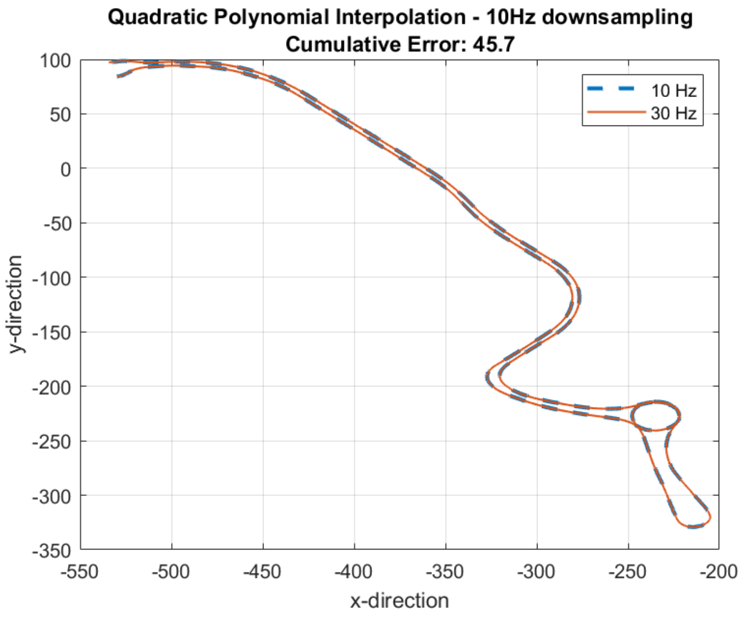

Figure 5: Quadratic Polynomial Interpolation 10Hz Downsample.

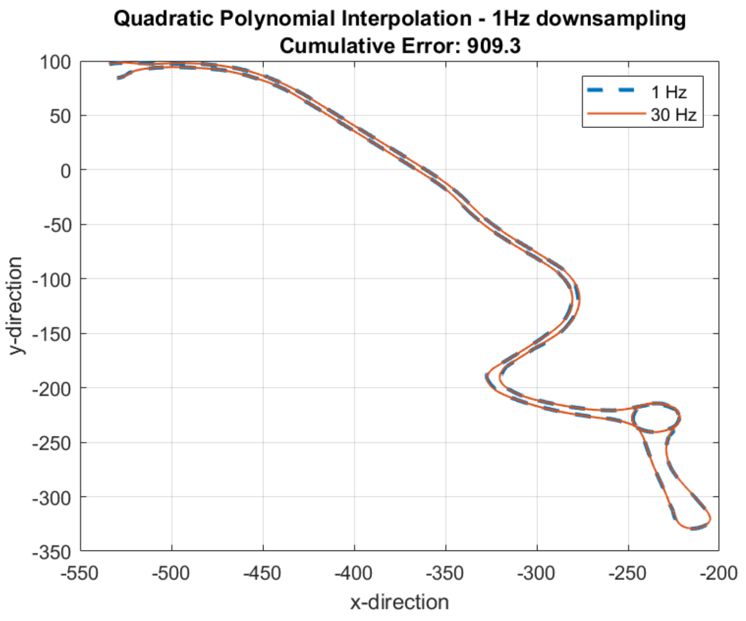

Figure 6: Quadratic Polynomial Interpolation 1Hz Downsample.

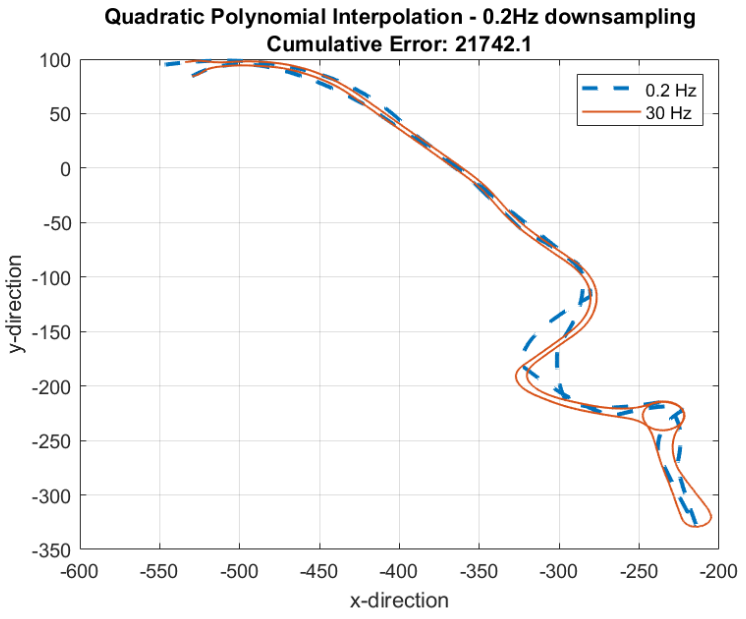

Figure 7: Quadratic Polynomial Interpolation 0.2Hz Downsample.

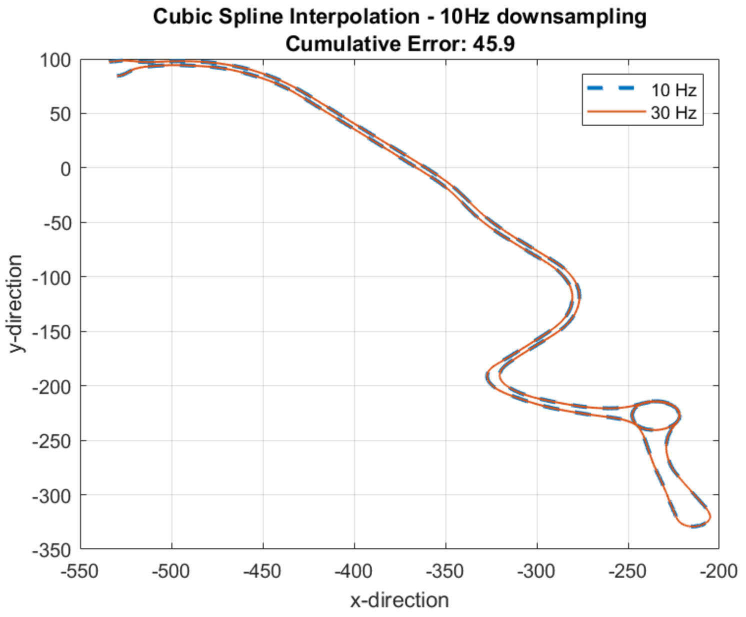

Figure 8: Cubic Spline Interpolation 10Hz Downsample.

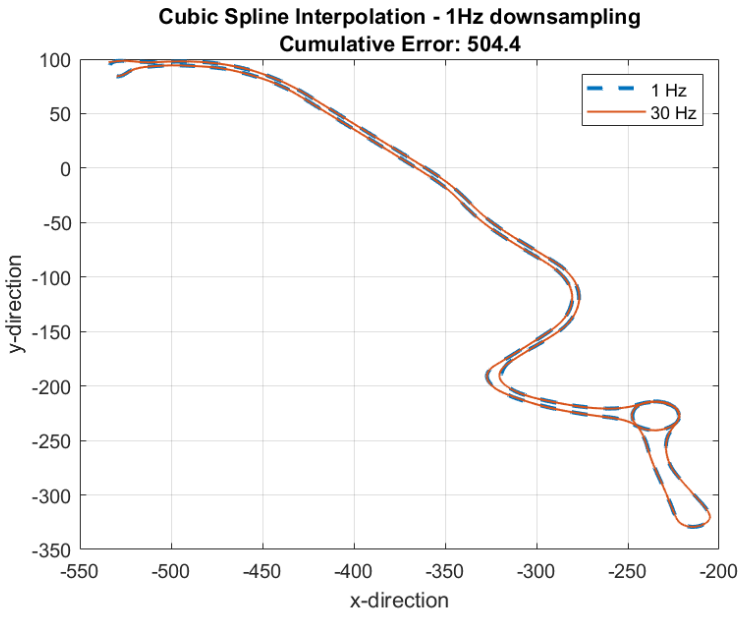

Figure 9: Cubic Spline Interpolation 1Hz Downsample.

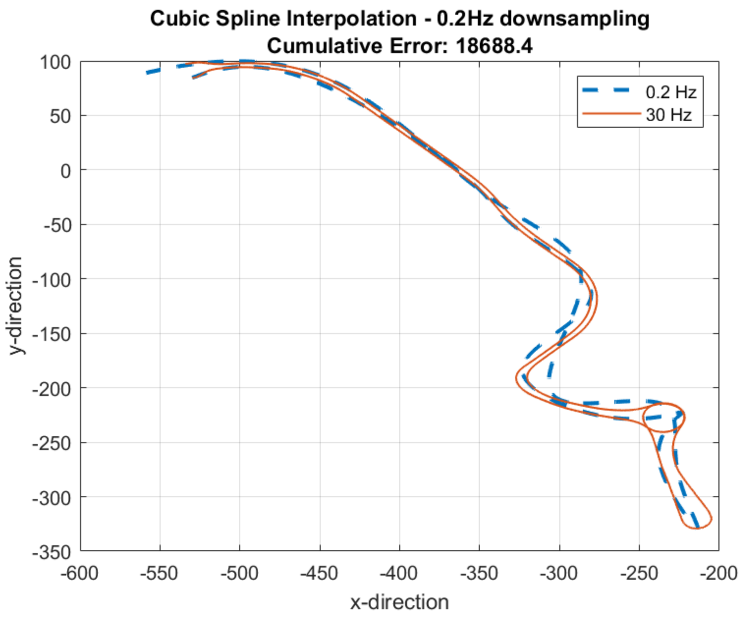

Figure 10: Cubic Spline Interpolation 0.2Hz Downsample.

Sample Rate	Linear	Quadratic	Cubic
10 Hz	45.8	45.7	45.9
1 Hz	1530.1	909.3	504.3
0.2 Hz	24072.4	21742.1	18688.4

Table 1: Cumilative error for different interpolation methods for different downsamples.

1.3

Comment on the quality of your interpolated paths at each test sample rate. Under what conditions is a low sample rate most detrimental?

From both the plots and the cumulative error table, we observe that at a high sampling rate of 10 Hz, the choice of interpolation method has minimal impact; all methods perform well. However, at 1 Hz, the selection of an interpolation method significantly affects the error, ranging from 1530 to 504. In the case of 0.2 Hz sampling, when using linear interpolation for strongly curved segments, we observe the most detrimental deviations from the original trajectory. This result aligns with expectations, as capturing intricate curvature is the most challenging aspect for interpolation methods.

1.4

Comment on the difference between the interpolation methods and the conditions under which each performs best. Under what conditions does their performance differ the most dramatically?

Interpolation methods vary in performance based on the trajectory's characteristics and sample rate:

- Linear: Linear interpolation is efficient and works well with linear segments or high sample rates but struggles with rapid changes and low sample rates.
- Quadratic: Quadratic interpolation excels when there's noticeable but not excessive curvature, offering better accuracy than linear interpolation in such scenarios.
- Cubic: Cubic spline interpolation performs best when handling complex curvature, variable motion, or low sample rates, providing the most accurate results.

The most substantial performance differences emerge in cases of pronounced curvature, abrupt changes, or low sample rates, where cubic spline interpolation significantly outperforms the other methods when you have a low sampling rate.

```
clear all; clc;

% Read the way points
waypoints = csvread('waypoints.csv');

%% 1.1
% Plot the true trajectory
plot_1_1 = 1;
if plot_1_1 == 1
    figure(1);
    plot(waypoints(:,1),waypoints(:,2),'LineWidth', 2);
    title("Grund truth trajectory");
    xlabel("x-direction");
    ylabel("y-direction");
    legend("30Hz");
    grid();
end

%% 1.2
wp_30Hz = waypoints;
t_30Hz = (0:(size(wp_30Hz, 1) - 1)) / 30;
n = size(wp_30Hz, 1);

% Downsample to 10, 1, 0.2 Hz
wp_10Hz = wp_30Hz(1:(30/10):n, :);
wp_1Hz = wp_30Hz(1:(30/1):n, :);
wp_0_2Hz = wp_30Hz(1:(30/0.2):n, :);

% Length of each downsample
n10 = size(wp_10Hz, 1);
n1 = size(wp_1Hz, 1);
n02 = size(wp_0_2Hz, 1);

% Timeseries for each downsample
t_10Hz = (0:(size(wp_10Hz, 1) - 1)) / 10;
t_1Hz = (0:(size(wp_1Hz, 1) - 1)) / 1;
t_0_2Hz = (0:(size(wp_0_2Hz, 1) - 1)) / 0.2;

%% a) Linear interpolation for downsampled waypoints
plot_1a = 1;

wp_10Hz_interp = interp1(t_10Hz, wp_10Hz, (0:(n - 1)) / 30, 'linear', 'extrap');
wp_1Hz_interp = interp1(t_1Hz, wp_1Hz, (0:(n - 1)) / 30, 'linear', 'extrap');
wp_0_2Hz_interp = interp1(t_0_2Hz, wp_0_2Hz, (0:(n - 1)) / 30, 'linear', 'extrap');

% Calculate cumulative error for the downsampling
cumulative_error_10Hz = cumulative_error(wp_10Hz_interp,wp_30Hz,n);
cumulative_error_1Hz = cumulative_error(wp_1Hz_interp,wp_30Hz,n);
cumulative_error_0_2Hz = cumulative_error(wp_0_2Hz_interp,wp_30Hz,n);

% Plot Linear VS Original for 10Hz downsampling
if plot_1a == 1
    figure(2);
    plot(wp_10Hz_interp(:,1), wp_10Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("10 Hz", "30 Hz");
    title(['Linear Interpolation - 10Hz downsampling', newline, 'Cumulative Error:↙
', num2str(cumulative_error_10Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid()

    % Plot Linear VS Original for 1Hz downsampling
    figure(3);
    plot(wp_1Hz_interp(:,1), wp_1Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("1 Hz", "30 Hz");
    title(['Linear Interpolation - 1Hz downsampling', newline, 'Cumulative Error: ',↙
num2str(cumulative_error_1Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid()

    % Plot Linear VS Original for 0.2Hz downsampling
    figure(4);
    plot(wp_0_2Hz_interp(:,1), wp_0_2Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("0.2 Hz", "30 Hz");
    title(['Linear Interpolation - 0.2Hz downsampling', newline, 'Cumulative Error:↙
', num2str(cumulative_error_0_2Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid()
end

%% b) Quadratic polynomial interpolation for downsampled waypoints
plot_1b = 1;
wp_10Hz_interp = quadratic_interpolate(wp_10Hz,n10,t_10Hz);
wp_1Hz_interp = quadratic_interpolate(wp_1Hz,n1,t_1Hz);
wp_0_2Hz_interp = quadratic_interpolate(wp_0_2Hz,n02,t_0_2Hz);

% Calculate cumulative error for the downsampling
cumulative_error_10Hz = cumulative_error(wp_10Hz_interp, wp_30Hz, size↙
(wp_10Hz_interp,1));
cumulative_error_1Hz = cumulative_error(wp_1Hz_interp,wp_30Hz, size(wp_1Hz_interp,↙
1));
cumulative_error_0_2Hz = cumulative_error(wp_0_2Hz_interp,wp_30Hz, size↙
(wp_0_2Hz_interp,1));

if plot_1b == 1
    % Plot Quadratic Polynomial VS Original for 10Hz downsampling
    figure(5);
    plot(wp_10Hz_interp(:,1), wp_10Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("10 Hz", "30 Hz");
    title(['Quadratic Polynomial Interpolation - 10Hz downsampling', newline,↙
'Cumulative Error: ', num2str(cumulative_error_10Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid();

    % Plot Quadratic Polynomial VS Original for 1Hz downsampling
    figure(6);
    plot(wp_1Hz_interp(:,1), wp_1Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("1 Hz", "30 Hz");
    title(['Quadratic Polynomial Interpolation - 1Hz downsampling', newline,↙
'Cumulative Error: ', num2str(cumulative_error_1Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid();

    % Plot Quadratic Polynomial VS Original for 0.2Hz downsampling
    figure(7);
    plot(wp_0_2Hz_interp(:,1), wp_0_2Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("0.2 Hz", "30 Hz");
    title(['Quadratic Polynomial Interpolation - 0.2Hz downsampling', newline,↙
'Cumulative Error: ', num2str(cumulative_error_0_2Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid();
end

% c) Cubic spline interpolation for downsampled waypoints
plot_c = 1;
wp_10Hz_interp = interp1(t_10Hz, wp_10Hz, (0:(n - 1)) / 30, 'spline');
wp_1Hz_interp = interp1(t_1Hz, wp_1Hz, (0:(n - 1)) / 30, 'spline');
wp_0_2Hz_interp = interp1(t_0_2Hz, wp_0_2Hz, (0:(n - 1)) / 30, 'spline');

%% Calculate cumulative error for the downsampling
cumulative_error_10Hz = cumulative_error(wp_10Hz_interp,wp_30Hz,n);
cumulative_error_1Hz = cumulative_error(wp_1Hz_interp,wp_30Hz,n);
cumulative_error_0_2Hz = cumulative_error(wp_0_2Hz_interp,wp_30Hz,n);

% Plot Cubic Spline VS Original for 10Hz downsampling
if plot_c == 1
    figure(8);
    plot(wp_10Hz_interp(:,1), wp_10Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("10 Hz", "30 Hz");
    title(['Cubic Spline Interpolation - 10Hz downsampling', newline, 'Cumulative↙
Error: ', num2str(cumulative_error_10Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid()

    % Plot Cubic Spline VS Original for 1Hz downsampling
    figure(9);
    plot(wp_1Hz_interp(:,1), wp_1Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("1 Hz", "30 Hz");
    title(['Cubic Spline Interpolation - 1Hz downsampling', newline, 'Cumulative↙
Error: ', num2str(cumulative_error_1Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid()

    % Plot Cubic Spline VS Original for 0.2Hz downsampling
    figure(10);
    plot(wp_0_2Hz_interp(:,1), wp_0_2Hz_interp(:,2), '--', 'LineWidth', 2); hold on;
    plot(wp_30Hz(:,1), wp_30Hz(:,2), 'LineWidth', 1);
    legend("0.2 Hz", "30 Hz");
    title(['Cubic Spline Interpolation - 0.2Hz downsampling', newline, 'Cumulative↙
Error: ', num2str(cumulative_error_0_2Hz)]);
    xlabel("x-direction");
    ylabel("y-direction");
    grid()
end

%% Functions
function cum_error = cumulative_error(interpolation, Hz30, n)
    cum_error = 0; % Initialize cumulative error

    for i = 1:n
        cum_error = cum_error + norm(interpolation(i, :) - Hz30(i, :));
    end
    cum_error = round(cum_error,1);
end

function quad_interpolate = quadratic_interpolate(wp,n_x,t_wp)
    quad_interpolate = [];
    for i = 1:n_x-2
        % Make the polynomals
        T = [t_wp(i),t_wp(i+1),t_wp(i+2)];
        x_points = [wp(i,1),wp(i+1,1),wp(i+2,1)];
        y_points = [wp(i,2),wp(i+1,2),wp(i+2,2)];

        coeff_x = polyfit(T,x_points,2);
        coeff_y = polyfit(T,y_points,2);

        % Fit the points from t_30Hz until the i point
        T_fit = T(1):(1/30):T(2) - (1/30);

        x_fit = polyval(coeff_x, T_fit).';
        y_fit = polyval(coeff_y, T_fit).';
        quad_interpolate = [quad_interpolate; x_fit y_fit];
    end

    % We extrapolate the last part
    T_fit = T(2):(1/30):96.8;
    x_fit = polyval(coeff_x, T_fit).';
    y_fit = polyval(coeff_y, T_fit).';
    quad_interpolate = [quad_interpolate; x_fit y_fit];
end
```

2 Problem 2

We decided to build a cheap robot with no floating point unit. Suppose you need to build an interpolation table with entries of the form (x, f(x)) for the function f(x) = cos x over the interval [0, π]. Please use uniform spacing between points.

2.1 a)

What table spacing is required to ensure 6 decimal digit accuracy, assuming that you will use linear interpolation between adjacent points in the table?

To ensure 6 decimal digit accuracy when using linear interpolation we need to calculate the spacing (Δx) between the table points so that the maximum error introduced by linear interpolation is within the desired accuracy range.

The error in linear interpolation can be estimated using the formula given in [1]:

$$\text{Error} = \frac{1}{8} M \Delta x^2 \tag{1}$$

Where:

- **Error:** Maximum error introduced by linear interpolation.
- **M:** Maximum value of the second derivative of the function within the given interval:
$$M = \max_{x \in [0,\pi]} |f''(x)|$$
- Δx Spacing between the table points.

We have the function $f(x) = \cos(x)$ over the interval $[0, \pi]$. The second derivative of $\cos(x)$ is $-\cos(x)$, and it has a maximum value of 1 within this interval: $|cos(0)| = |cos(\pi)| = 1$

To achieve 6 decimal digit accuracy, we need to make sure that the error introduced by linear interpolation is less than 10^{-6}. We need to solve:

$$\frac{1}{8} \cdot 1 \cdot \Delta x^2 \leq 10^{-6}$$

Solving for Δx:

$$\Delta x^2 \leq \frac{10^{-6} \cdot 8}{1} = 0.000008$$

$$\Delta x \leq \sqrt{0.000008} \approx 0.0028284$$

To ensure 6 decimal digit accuracy, we need to choose a spacing (Δx) that is less than or equal to 0.0028284.

2.2 b)

How fine must the spacing be if you use quadratic interpolation?

To ensure 6 decimal digit accuracy when using quadratic interpolation we need to calculate the spacing (Δx) between the table points so that the maximum error introduced by quadratic interpolation is within the desired accuracy range.

The error in quadratic interpolation can be estimated using the formula given in[1]:

$$\text{Error} = \frac{1}{72 * \sqrt{3}} M \Delta x^3 \tag{2}$$

Where:

- **Error:** Maximum error introduced by quadratic interpolation.
- **M:** Maximum value of the third derivative of the function within the given interval:
$$M = \max_{x \in [0,\pi]} |f'''(x)|$$
- Δx**:** Spacing between the table points.

We have the function $f(x) = \cos(x)$ over the interval $[0, \pi]$. The third derivative of $\cos(x)$ is $\sin(x)$, and it has a maximum value of 1 within this interval: $sin(\pi/2)| = 1$

To achieve 6 decimal digit accuracy, we need to make sure that the error introduced by quadratic interpolation is less than 10^{-6}. We need to solve:

$$\frac{1}{72 * \sqrt{3}} \cdot 1 \cdot \Delta x^3 \leq 10^{-6}$$

Solving for Δx:

$$\Delta x^3 \leq \frac{10^{-6} \cdot 72 * \sqrt{3}}{1} = 0.00012470765$$

$$\Delta x \leq \sqrt[3]{0.00012470765} \approx 0.04996099$$

To ensure 6 decimal digit accuracy, we need to choose a spacing (Δx) that is less than or equal to 0.04996099. This means that for quadratic interpolation the stepsize can be 0.04996099/0.0028284 = 17.6 times a larger then for linear.

2.3 c)

In each case, how many entries do you need in the table?

Linear: For linear we can find the number of points by using the equation from [1]:

$$N = \frac{b-a}{\Delta x} + 1 \tag{3}$$

Where N is the number of points, b and a is the start and end of the interval and Δx is the spacing we found. This gives

$$N = \frac{b-a}{\Delta x} + 1 = \frac{\pi - 0}{0.0028284} + 1 = 1111.7313865. \tag{4}$$

This means we need to have 1112 entries to the table to ensure 6 decimal digit accuracy when we use linear interpolation.

Quadratic: It is the same formula for quadratic and we get:

$$N = \frac{b-a}{\Delta x} + 1 = \frac{\pi - 0}{0.04996099} + 1 = 63.880912759 \tag{5}$$

This means we need to have 64 entries to the table to ensure 6 decimal digit accuracy when we use quadratic interpolation. Many fewer points than for the linear interpolation.

References

[1] MIT OpenCourseWare. (n.d.). *Introduction to Numerical Analysis.* Retrieved from `https://ocw.mit.edu/ans7870/2/2.086/S13/MIT2_086S13_Textbook.pdf`