# ENSC 180 – Introduction to Engineering Analysis Tools

# Assignment #4: Interpolation

### Objective

The purpose of this assignment is to get you familiarized with interpolation in MATLAB. *Interpolation* is used frequently in many aspects of engineering, science, and data analysis. It is the process of guessing non-existing values in between the known values of a given sequence.

### Background

To guess the unknown values of a sequence, we often fit a straight line, a polynomial, or some other function to the known values. To get started with the assignment, read the help files of the following MATLAB functions: polyfit and polyval. These two functions, respectively, find the coefficients of a polynomial that fits the data in the least squares sense, and evaluate the polynomial at a given value.

### Assignment details

Table 1 below shows a weather record for Vancouver International Airport (YVR) for November 1, 2014, publicly available at http://climate.weather.gc.ca/. Several parameters such as temperature, dew point, etc. have been measured at 1 hour intervals. Your task is to test several interpolators on this data.

**Table 1:** weather data record for YVR for November 1, 2014 from http://climate.weather.gc.ca/

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Time | Temp (°C) | Dew Point Temp (°C) | Relative Humidity (%) | Wind Dir. (10s deg) | Wind Speed (km/h) | Visibility (km) | Pressure (kPa) |
| 0:00 | 9.2 | 8.1 | 93 | 8 | 9 | 16.1 | 101.15 |
| 1:00 | 9.6 | 7.7 | 88 | 7 | 8 | 24.1 | 101.17 |
| 2:00 | 9.0 | 7.1 | 88 | 9 | 9 | 24.1 | 101.17 |
| 3:00 | 7.7 | 5.7 | 87 | 6 | 8 | 24.1 | 101.13 |
| 4:00 | 6.8 | 5.6 | 92 | 1 | 5 | 24.1 | 101.15 |
| 5:00 | 6.0 | 5.1 | 94 | 10 | 8 | 24.1 | 101.16 |
| 6:00 | 5.9 | 5.0 | 94 | 9 | 3 | 24.1 | 101.16 |
| 7:00 | 4.3 | 3.9 | 97 | 35 | 8 | 19.3 | 101.19 |
| 8:00 | 7.2 | 5.7 | 90 | 36 | 9 | 24.1 | 101.22 |
| 9:00 | 8.9 | 6.4 | 84 | 36 | 3 | 24.1 | 101.23 |
| 10:00 | 9.2 | 6.7 | 84 | 28 | 3 | 24.1 | 101.24 |
| 11:00 | 10.2 | 7.1 | 81 | 30 | 17 | 24.1 | 101.27 |
| 12:00 | 10.9 | 8.5 | 85 | 29 | 16 | 24.1 | 101.25 |
| 13:00 | 10.6 | 8.9 | 89 | 26 | 9 | 24.1 | 101.25 |
| 14:00 | 10.6 | 8.5 | 87 | 26 | 16 | 24.1 | 101.25 |
| 15:00 | 10.7 | 8.8 | 88 | 28 | 12 | 24.1 | 101.31 |
| 16:00 | 10.3 | 8.3 | 87 | 24 | 7 | 24.1 | 101.32 |
| 17:00 | 8.6 | 7.2 | 91 | 18 | 10 | 24.1 | 101.37 |
| 18:00 | 8.2 | 6.7 | 90 | 14 | 11 | 24.1 | 101.43 |
| 19:00 | 8.0 | 6.6 | 91 | 14 | 12 | 24.1 | 101.48 |
| 20:00 | 8.8 | 7.6 | 92 | 15 | 13 | 24.1 | 101.54 |
| 21:00 | 7.1 | 5.9 | 92 | 10 | 8 | 24.1 | 101.58 |
| 22:00 | 7.5 | 6.5 | 93 | 10 | 4 | 24.1 | 101.61 |
| 23:00 | 7.5 | 6.6 | 94 | 8 | 7 | 24.1 | 101.64 |

**Task 1:** From the table, copy the temperature values for even-numbered hours (0:00, 2:00, ..., 22:00) into a vector temp\_even, and the temperature values for odd-numbered hours (1:00, 3:00, ..., 23:00) into a vector temp\_odd. Also, store the even-numbered hours in the vector time\_even and odd-numbered hours in the vector time\_odd. The goal is to interpolate the values at odd-numbered hours and compute the interpolation error using the actual values.

First, implement the following three steps:

Step 1) Fit a straight line (a first-degree polynomial) to the data in temp\_even:

P = polyfit(time\_even,temp\_even,1);

Step 2) Then, using the function polyval, compute the values of this polynomial at time\_odd. Store the resulting values in the vector temp\_int\_odd. This vector now contains the interpolated temperature values at odd-numbered hours.

Step 3) Finally, compute the mean squared error between the true values in temp\_odd and the interpolated values in temp\_int\_odd:

MSE = sum((temp\_int\_odd - temp\_odd).^2)/length(temp\_odd)

Once you have implemented these three steps, make a plot with the following three curves: (1) a blue curve, showing actual temperature values from the table at all hours (0:00, 1:00, ..., 23:00), (2) a red curve showing the actual values at even-numbered hours and interpolated values at odd-numbered hours, and (3) the straight line, shown as green, that represents the fitted polynomial P evaluated at all hours. Remember to label the axes.

What can you conclude from this plot? Are the interpolated values close to the true ones at odd-numbered hours? Is the straight line a good model for this data?

**Task 2:** Make a for loop around the three steps implemented in Task 1. The loop counter should be N (degree of the polynomial), and it should run for N = 1:9. Don't forget to replace 1 in polyfit in step 1 by N. Also, remember to store MSE values of different polynomial degrees in different variables. The best way is to define a vector MSE of length 13 before the loop and then modify step 3) to

MSE(N) = sum((temp\_int\_odd - temp\_odd).^2)/length(temp\_odd);

After you run the for loop and compute all 9 values of MSE, plot them against the polynomial degree. The plot command should look something like

plot(1:9, MSE);

Add the axis labels. Then briefly discuss the shape of the curve and try to explain why it looks like the way it does. Which polynomial degree produces the best interpolation on this data? What is the MSE of the best interpolator?

**Task 3:** By now you may have noticed that there could be better ways to interpolate a sequence than the ones used in Tasks 1 and 2. Intuitively, the temperature value at 7:00 would most likely be similar to the temperature at 6:00 and/or 8:00 (the closest data points), and less related to the temperature at 22:00. Yet, in Tasks 1 and 2, we fitted polynomials to *all* data points, and therefore used *all* data points when interpolating each value. In this task, you will implement another interpolation method called *nearest neighbor interpolation*. It is extremely simple, yet powerful and often used in practice.

Nearest neighbor interpolation uses the nearest known value to interpolate the missing value. For example, to interpolate the temperature value at 7:00, we would simply copy the closest known temperature value which, in our case, is either 6:00 or 8:00. Since there is a choice of two nearest neighbors, let us simply use the previous value as the nearest neighbor, hence the temperature value at 6:00 will be simply copied to 7:00.

Implement the nearest neighbor interpolation for the temperature values at odd-numbered hours. In other words, the temperature value at each odd-numbered hour should be copied from the previous (even-numbered) hour. Compute its MSE. How does MSE of nearest neighbor interpolation compare with MSEs obtained in Tasks 1 and 2?

Make a plot with the following two curves: (1) a blue curve, showing actual temperature values from the table at all hours (0:00, 1:00, ..., 23:00), (2) a red curve showing the actual values at even-numbered hours and interpolated values at odd-numbered hours. Do the curves match up well?

**Task 4:** In Task 3, we arbitrarily picked the previous data point as the nearest neighbor. We could also have used the future data point, since it was equally close. But with either of these approaches, we are only using one of the two nearest neighbors. Perhaps we would get better results if we used both?

In this task you will implement *local linear interpolation*. For each missing value, we find the two nearest available data points, one previous and one future, and fit a straight line (first-degree polynomial) to them. Then we evaluate that line at the location of the missing value. In our case, since the nearest data points are equally distant from the missing data point, the procedure gets really simple - we only need to compute the average of the two nearest data points.

Implement the local linear interpolation for the temperature data at odd-numbered hours. For the temperature value at 7:00, simply average the values 6:00 and 8:00, and so on. Note that the last missing data point is at 23:00, and this one does not have any future data available. Hence, for the temperature at 23:00, we can simply use nearest neighbor interpolation from 22:00. But for all other missing data points, two neighbors are available, so we can use local linear interpolation.

Compute the MSE of local linear interpolation. How does its MSE compare with MSEs obtained in the previous tasks?

Make a plot with the following two curves: (1) a blue curve, showing actual temperature values from the table at all hours (0:00, 1:00, ..., 23:00), (2) a red curve showing the actual values at even-numbered hours and interpolated values at odd-numbered hours. How well do the curves match up?

**Task 5:** What if, instead of using just a straight line between the available data points, we fit a higher degree polynomial? This might potentially improve the interpolation. We would, however, need more neighboring data points for this, not just two - two data points are enough to fit a straight line, but not higher-degree polynomials. This is called *piecewise polynomial interpolation*.

A particular version of this kind of interpolation is called *cubic spline interpolation*. It is implemented in MATLAB as a function spline. Please read its help file.

Implement cubic spline interpolation for the temperature data at odd-numbered hours using the MATLAB function spline. Note that, like local linear interpolation in Task 4, cubic spline interpolation needs data points on both sides of the missing value to work best. Hence, the result for the last missing data point (at 23:00) will not be reliable because this data point does not have any future data available. Hence, as before, for the temperature at 23:00 we can simply use nearest neighbor interpolation from 22:00, while for others we can use cubic interpolation.

Compute the MSE of cubic spline interpolation. How does its MSE compare with MSEs obtained in the previous tasks?

Make a plot with the following two curves: (1) a blue curve, showing actual temperature values from the table at all hours (0:00, 1:00, ..., 23:00), (2) a red curve showing the actual values at even-numbered hours and interpolated values at odd-numbered hours. How well do the curves match up?

Overall, which interpolation method worked the best on this data?