

## HW Set 01: Interpolation

KAITLYN PARRINELLO<sup>1</sup>

<sup>1</sup>*Homer L. Dodge Department of Physics and Astronomy, University of Oklahoma,  
Norman, OK 73019, USA*

### ABSTRACT

The first real homework: it's interpolating time. All code can be found in the Jupyter Notebook: [https://github.com/kparrine/Computational\\_Astro/blob/main/C1/C1.ipynb](https://github.com/kparrine/Computational_Astro/blob/main/C1/C1.ipynb).

#### 1. QUESTION 1

The derivations are available through these links:

1. [Part a](#)
2. [Part b](#)
3. [Part c](#)

#### 2. QUESTION 2

Below is the code without comments for my linear interpolation function. The full commented function can be found in my Jupyter Notebook (JN) at this link: [JN file](#) which also contains the rest of the code for this assignment.

```
def linear_interp(xi,yi,xi1,yi1,x):
    '''
    Function for linear interpolation.

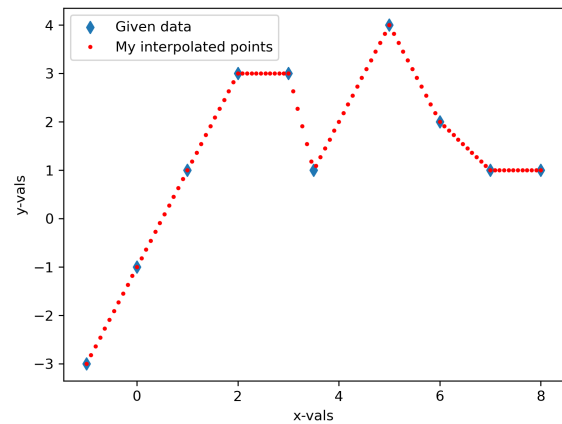
    xi, yi = Initial given points
    xi1, yi1 = Next set of given points
    x = point to perform interp
    '''
    if not (min(xi, xi1) <= x <= max(xi, xi1)):
        raise ValueError("x is outside
            interpolation interval.")

    if xi1 == xi:
        raise ValueError("xi and xi1 cannot
            be equal.")

    ai = (yi1-yi)/(xi1-xi) #Slope
    g = ai*(x-xi)+yi

    return g
```

My function inherently does not allow for extrapolation because I check to ensure that the value to interpolate at is between two specific points. If that point is

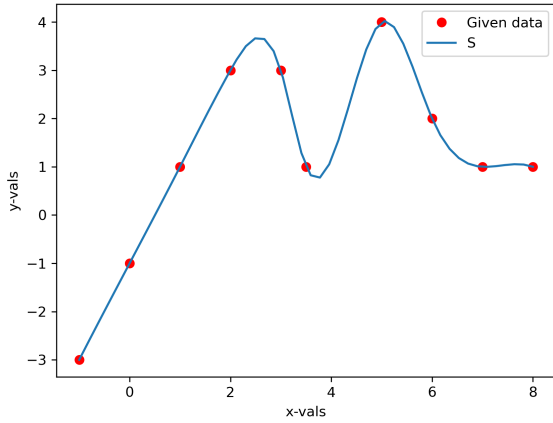


**Figure 1.** Linearly interpolated data are red circles, and the given data are blue diamonds.

outside the domain of the two points, the function stops and raises an error message. One thing to note is that this is a local domain instead of a global domain. When the function is actually put into use there is another check over the global domain of values to interpolate at, again preventing extrapolation. If I wanted to allow for extrapolation, it would be no more than for points 0.5 more or less than the last point. I would assume that the line would continue with a constant slope from the last points.

Figure 1 is the result of interpolating the given HW01.data.txt file at 10 times higher resolution with my function. The 10 times resolution means that there are 10 points between each set of two points from the dataset. As expected, the function draws a line between each set of points.

Figure 2 is a cubic spline interpolation from `Scipy.CubicSpline` plotted over the original dataset. Because the cubic spline is a polynomial, it creates local minima and maxima between given datapoints. Splines are continuous functions with continuous first and sec-



**Figure 2.** Cubic spline interpolation is the blue line and the original dataset are red circles.

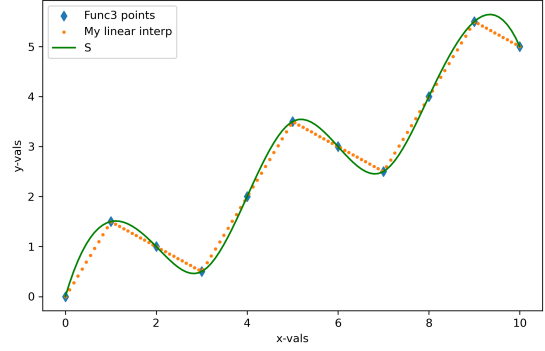
ond derivatives, however, their third derivative is not continuous. A cubic spline would be a good choice for data that reflects a smooth curve like a lightcurve of a variable star because it may predict the maxima and minima of the flux over some time. However, this behavior will not work if the potential underlying function is something more like a piecewise function in which case the linear interpolation would probably do a better job. An example of this case is the position vs time of an object moving at a constant speed. In this case though the data would be fit with a line rather than using interpolation since that is exactly the function beneath the data.

Both types of interpolation have their own advantages and disadvantages. The linear interpolation is simpler, and therefore, easier computationally since it's just a slope and an added offset. However, this type of interpolation is not smooth since only the function itself is continuous and not its derivatives. As I've mentioned before, the cubic spline is a continuous function with continuous first and second derivatives. This type of interpolation is far more complex because a tri-diagonal matrix must be solved which would increase the computation time. Because this function is adding polynomials together, there is a lot of wiggling that creates local minima/maxima in between given datapoints. This behavior can be seen more clearly in the next question.

### 3. QUESTION 3

From the given function

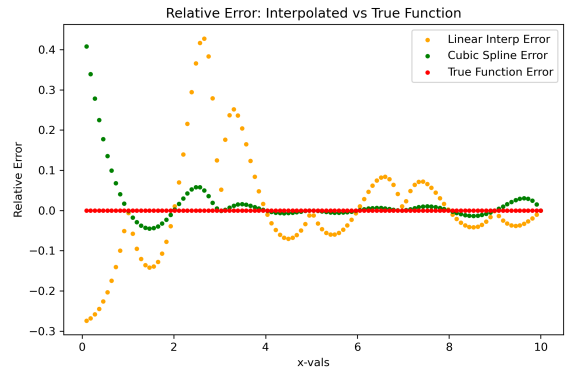
$$y = \sin \frac{\pi}{2}x + \frac{x}{2} \quad (1)$$



**Figure 3.** The original dataset are blue diamonds, the linearly interpolated data are orange points, and the cubically interpolated data are a green line.

I created a dataset to interpolate over. Figure 3 is the resulting dataset with the data interpolated linearly and cubically to 10 times higher resolution.

I also plotted the relative error between my interpolated values for the linear and cubic spline, and for the true function in the domain  $x = [0, 10]$  in Figure 4. I plotted the error for the true function to have a more visual comparison for the interpolations, so naturally, the error for the true function is zero since it is a match. The cubic spline follows the true function more closely because it is polynomial, but struggles at the endpoints. Scipy's function by default chooses the not-a-knot condition for the endpoints, but there are other options such as periodic, clamped, and natural. The linear interpolation shows a lot more errors, both positive and negative showing overestimates and underestimates of the true function. Because it seems the underlying function is curved, the linear interpolation cannot replicate this behavior.



**Figure 4.** The relative error of the true function, the linearly interpolated values, and in the cubically interpolated values in red, orange, and green, respectively.

## 4. QUESTION 4

Probably not since a simple harmonic oscillator (SHO) makes a sine curve, so with linear interpolation one would be underestimating or over estimating the energies at the different points in time. I think the cubic spline would do a better job because a sine curve can be replicated by a polynomial, also cubic spline is a continuous function, and has continuous first and second derivatives. But neither of them will conserve energy: the linear interpolation cannot replicate the sine-like curve of the SHO motion, so energy will be lost. For the cubic, it has been shown that it will struggle at the endpoints, and may introduce extra fluctuations that will result in energy not being conserved.