2D Interpolation Code

Anthony Tracy

December 2017

1 Abstract

I just wanted to write up a quick overview of the code I wrote up and a few questions I had for polishing up the code and making sure it works well with your application.

2 Method

The objective I had in my mind was that the function would take in two arguments. The first being the voltage being used and the second being the location of the crater impact. It would then return the stretch dependence term used for the spectroscopy.

The issue is that numpy and scipy cannot be used in the application of this function, so I wrote this in base python. Which also required finding a decent method for solving this problem. In order to solve this I made the assumption that I could expand the function into:

$$f(V, X) = a_0 + a_1 v + a_2 x + a_3 v x + ... \tag{1}$$

where I have left off the terms of order 2 or greater, since it just needs to be a linear, or in this case a bi-linear, interpolation. So my code requires four boundary conditions to solve for all unknowns, I used arrows to indicate ares where the data was blank but I used the data from the box in the direction of the arrow to fill that point.

Since I have access to known data, table 1, I just wrote these into a dictionary where the voltage (v), and impact dist (x), are input as keys to look up the related scale value.

Volts (V) → Position (mm) ↓	2500	2600	2685	2700	2800	2900
52.8	950	888	→	837	796	759
75.7	908	856	815	→	769	734
104.2	883	826	788	→	740	707

Table 1: This is just the table I was given of simulated data points.

Using this table of know data values I was able to solve equation 1. Basically if the key exists return the scaling factor without doing interpolation. Otherwise take the two nearest known voltages and impact distances, larger and smaller. Then there are four points with known scaling factors which allows this to become a solvable series of four equations as shown in equation 2.

$$\begin{aligned}
a_0 &= \frac{f(v_1,x_1)v_2x_2}{(v_1-v_2)(x_1-x_2)} + \frac{f(v_1,x_2)v_2x_1}{(v_1-v_2)(x_2-x_1)} + \frac{f(v_2,x_1)v_1x_2}{(v_1-v_2)(x_2-x_1)} + \frac{f(v_2,x_2)v_1x_1}{(v_1-v_2)(x_1-x_2)} \\
a_1 &= \frac{f(v_1,x_1)x_2}{(v_1-v_2)(x_2-x_1)} + \frac{f(v_1,x_2)x_1}{(v_1-v_2)(x_1-x_2)} + \frac{f(v_2,x_1)x_2}{(v_1-v_2)(x_1-x_2)} + \frac{f(v_2,x_2)x_1}{(v_1-v_2)(x_2-x_1)} \\
a_2 &= \frac{f(v_1,x_1)v_2}{(v_1-v_2)(x_2-x_1)} + \frac{f(v_1,x_2)v_2}{(v_1-v_2)(x_1-x_2)} + \frac{f(v_2,x_1)v_1}{(v_1-v_2)(x_1-x_2)} + \frac{f(v_2,x_2)v_1}{(v_1-v_2)(x_2-x_1)} \\
a_3 &= \frac{f(v_1,x_1)}{(v_1-v_2)(x_1-x_2)} + \frac{f(v_1,x_2)}{(v_1-v_2)(x_2-x_1)} + \frac{f(v_2,x_1)}{(v_1-v_2)(x_2-x_1)} + \frac{f(v_2,x_2)}{(v_1-v_2)(x_1-x_2)}
\end{aligned} \tag{2}$$

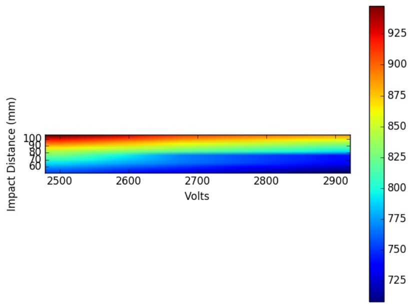

Figure 1: This shows that the scaling term decreases as either the voltage gets larger, or as the distance gets smaller. Which makes sense, given the known data.

So in the end this function is a bi-linear interpolation which uses four known points to approximate an unknown point. Just to test it, made a visual where I ran this for all points within the region of known points creating the images in figure 1.

Looking at figure 1 it seems to match what I would expect, though there are what looks like 3 lines that stay constant at about 2600 Volts, which I think is due to the values that were unknown in the dataset, which are shown as arrows in table 1.

3 Questions I have

1. I am curious how this function will be implemented, by this I mean how will it be called. Should I expect it to be called as a python script with OS inputs, or should I make this into an executable, which would again take in OS arguments. Or will this be used as just importing the functions?

2. As for how the look-up table of known data should be stored. Currently I just have the function creating the dictionary every time this script is called, and it does not create new data every time an interpolation is requested. I can leave it this way, or I could have it write to a local file, csv or hdf5, that would be called every time the function is run, and with that I could store every interpolation that is done so that the data file would get larger every time a new interpolation is one but then there would be no need to interpolate a second time for different voltage/impact inputs.