Cubic Spline Interpolation

Leigh J. Little SUNY Brockport October 13, 2020

Contents

1 Course Summary so Far

At this point in the semester, we have reviewed most of the basic functionality of MATLAB, however we have also covered what would be the essential elements of any programming language. It turns out, all programming languages do more or less the same things. What differs between them is the programming environment, the built-in command set and the command syntax. For example, if you wanted to learn another language, you would need to learn the following:

- The programming environment (how you write programs, *etc.*).
- The types of variables and how to use them.
- How to define and work with scalars, vectors and matrices.
- The syntax for `if-then` structures.
- The syntax for looping structures.
- The syntax for creating your own functions.

Once these are known, you can write programs in many different languages.

There is one important topic we have not discussed and that is character variables. Character variables are used in situations where your data is not numerical but rather consists of values such as names, addresses, phone numbers, *etc.*.

2 Interpolation

Interpolation is an important area in scientific computing. The essence of interpolation is this: Given a table of values for some data, determine values that are not in the table. Consider the data shown in Table 1. This data represents some function, but the formula for the function is unknown. All we know is the data in the table. How could we determine what the value of $T(x)$ is when $x = 0.33$? How could we determine what value(s) of x correspond to $T(x) = 1.00$?

This a common situation. Often times, a formula for the function is not known. Instead, all that is known is a table of values. This table can come from either experimental measurements or the table might be the result of some other calculation (for example, the solution to a differential equation).

x	$T(x)$
-1.00	3.08
-0.70	1.48
-0.40	0.90
-0.10	0.91
0.20	1.25
0.50	1.80
0.80	2.51
1.10	3.40
1.40	4.53
1.70	6.00
2.00	7.93

Table 1: Table of values for some function $T(x)$.

3 What We are Looking For

Interpolation is a broad field and the specific techniques used depend on the quality of the data and what information needs to be obtained from the data. So far, we have used linear interpolation. This computes the line that goes through successive points in the table. This is the most common form of interpolation but it suffers from lack of accuracy and flexibility.

What we would like to have is a method of accurately predicting values in the table to the point that we can use this method as easily as we use any other function. In other words we would like to be able to compute

$$T = \text{Value of } T \text{ for some input value of } x.$$

as easily as we compute a known function like

$$y(x) = \cos(x) - x.$$

4 Cubic Spline Interpolation

The most common technique for general purpose interpolation is known as cubic spline interpolation. A cubic spline is a set of cubic polynomials that go through the data points in the table, but its construction also permits accurate values for the first and second derivatives of the data. There are many forms of cubic splines, but the one we are going to use here is the most common.

The standard cubic spline is appropriate in the following circumstances:

- The raw data points form a smooth curve when plotted. Spline interpolation is not appropriate for many best-fit line data sets due to the lack of smoothness in the data.
- The data do not suggest the presence of step-functions.
- The range of x values we intend to input lie with the x range for the table. Attempting to predict function values that lie outside this range is called *extrapolation* and is a separate area of study.

Fortunately, MATLAB has built-in functions to construct the spline and to use it to evaluate the T value for any set of input values x.

There are two types of the standard cubic spline. The two types arise because the data in the table is not quite sufficient enough to determine the spline. We need to supply two additional pieces of information.

4.1 The Non-a-Knot Spline

This is probably the most common version of the standard spline. This version assumes that the second derivative of the function in the table is 0 at the endpoints. With this assumption, no additional information is required to determine the spline.

Type in the following commands

```
>> load testdata.dat      % Table 1 data to 5 digits instead  of 3
>> xdata = testdata(:,1);
>> Tdata = testdata(:,2);
>> T = spline(xdata,Tdata);   % Create the spline
```

This is the process for creating the spline `T`. If you look at the spline, it is not a number. It a more complex way of storing data called a *structure*. For now, all we need to know is that `T` is what is needed to compute $T(x)$ values for any x value. To do this, use the `ppval` function. In order to compute the T value for $x = 1.33$, we would do

```
>> x = 1.33;
>> Tval = ppval(T,x)
Tval =
    4.2489
```

One property of the spline is that it should reproduce the data in the table. We can test this by sending `xdata` to the `ppval` function. We should recover the original `Tdata` (with some small rounding errors).

```
>> Ttest = ppval(T,xdata);
>> [Tdata Ttest]
ans =
    3.0862    3.0862
    1.4833    1.4833
    0.9090    0.9090
    0.9159    0.9159
    1.2542    1.2542
    1.8004    1.8004
    2.5131    2.5131
    3.4069    3.4069
    4.5385    4.5385
    6.0019    6.0019
    7.9304    7.9304
```

We can also plot the spline by generating a new set of equally spaced x-coordinates and evaluating the spline at these points.

```
>> xval = linspace(-1,2,51)';
>> tval = ppval(T,x);
>> hold on
>> plot(xval,tval,'b-')
>> plot(xdata,Tdata,'k*')
>> hold off
>> xlabel('x')
>> ylabel('T(x)')
```

The values for $T(x)$ come from the function

$$T(x) = x^2 e^{-x} + e^x.$$

This allows to gain some insight into the accuracy of the spline. To do this, evaluate the exact expression for $T(x)$ and compare these to the values obtained from the spline.

```
>> Texact = xval.^2.*exp(-xval) + exp(xval);
>> error = (tval-Texact)./Texact;
>> [xdata tval Texact error]
```

Note that at the left end of the domain, the error is somewhat larger than at other points in the domain. This is because of the non-a-knot assumption that the function has a second derivative of 0 (*i.e.*, zero curvature) at the endpoints. The graph shows that this is not a bad assumption at the right end of the domain, but not so good an assumption at the left end.

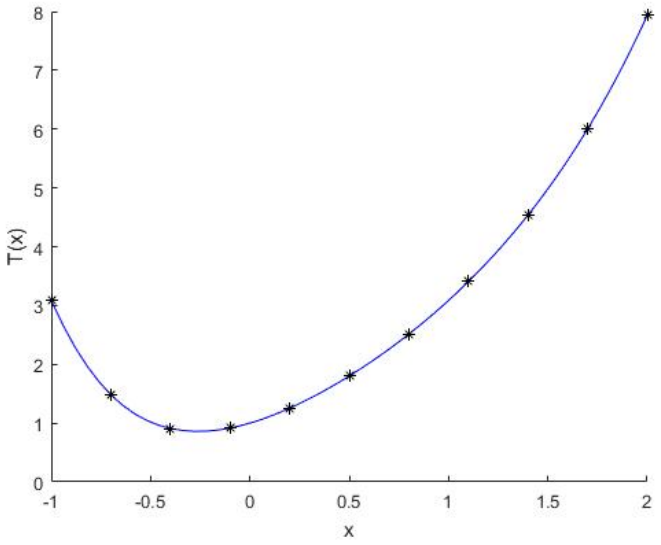

Figure 1: Plot of the cubic spline approximation for $T(x)$. The stars indicate the points used to construct the spline.

4.2 The Clamped Spline

The second version of the standard cubic spline is the clamped spline. This spline attempts to address potential accuracy errors in the non-a-knot spline by providing values of the derivative of the function $T'(x)$ at the endpoints. These can either be exact derivative values or approximations, though in most instances, the exact values are not known.

For demonstration purposes, we will use the fact that we can get the exact derivative because we know the formula for $T(x)$. We obtain

$$T'(x) = xe^{-x}(2-x) + e^x.$$

The process for computing the clamped spline is slightly different. The variables `m1` and `m2` are the exact derivatives of $T(x)$ at the endpoints.

```
>> load testdata.dat      % Table 1 data to 5 digits instead  of 3
>> xdata = testdata(:,1);
>> Tdata = testdata(:,2);
>> xtemp = x(1);
>> m1 = xtemp*exp(-xtemp)*(2-xtemp) + exp(xtemp);
>> p = length(xdata);
>> xtemp = xdata(p);
>> m2 = xtemp*exp(-xtemp)*(2-xtemp) + exp(xtemp);
>> Ttemp = [m1;Tdata;m2]
>> TC = spline(xdata,Ttemp);   % Create the clamped spline
```

Note that the two slopes need to be prepended and appended to the existing `Tdata` vector. This new vector `Ttemp` is sent into the `spline` function.

We can now test the accuracy of the clamped spline

```
>> xval = linspace(-1,2,51)';
>> tcval = ppval(TC,x);
>> Texact = xval.^2.*exp(-xval) + exp(xval);
>> error = (tcval-Texact)./Texact;
>> [xdata tcval Texact error]
```

From the new table, we can see that the error at the left end is much better than it was before.

The power of the spline is that we can now work the data in the table like any other function. We will see how we can use this over the next few lectures.

5 Accuracy

The cubic spline is an approximate technique and as such it will have limitations to its accuracy. Can this error be quantified? The answer is yes. We will just give the main results, but the details can be found in any book on numerical analysis.

a) The accuracy is limited by the raw data. If the raw data is only accurate to n digits then any calculation that uses the spline is going to have at most n digits of accuracy.

b) There is also an error in the spline itself. This error is going to be bounded by

$$|\text{Absolute Spline Error in T(x)}| \leq Ch^4$$

where C is a constant of moderate size (usually less than 10 or so) and

$$h = \max_{1 \leq i \leq n-1} |x_{i+1} - x_i| .$$

is the maximum difference between any 2 successive x values.

c) As the number of data points used to construct the spline increases, the error associated with b) decreases.