Linear Interpolation and Cubic Splines

Leigh J. Little SUNY Brockport November 30, 2022

Contents

1 Introduction

We have examined several techniques for estimating the value of a definite integral and have seen that these methods have good accuracy. Simpson's rule will be more accurate than the trapezoidal method, but both of these methods have the drawback that you don't necessarily know if you have chosen enough points in your table in order to obtain a desired level of accuracy. The benefit of MATLAB's `integral` function is that you don't need to specify the value of n. It will be able to achive the requested accuracy automatically. In addition, it seems that the `integral` function can also compute improper integrals.

All of the methods we have examined so far have one aspect in common; the integrand is a known function and can be evaluated at whatever x-coordinate is needed. Suppose we had a slightly different situation. What if all we know is a table of values? These values might have been very expensive to obtain (maybe several days worth of computing time) and they are all we have to work with. What options for computing the definite integral of the values in the table do we have and is there any way these can be improved?

2 A Practical Example

Suppose we have the table of values given in the file `intdata.dat` and we want to compute the definite integral of the data in the table. Note that the values are only available to 4 digits in both x and y. This is a common occurrence when data is obtained using instrumentation. This also means that the accuracy of the integral approximation is going to be 4 digits in the most optimistic of scenarios.

We can use MATLAB's `trapz` function to give a trapezoidal approximation and because the data is equally spaced and has an odd number of points, we can also use the Simpson's rule function we wrote. The exact value of the integral (to 8 digits) we are looking for is 1.1064840. How accurate are these methods? The script `int_ex1` will test this.

```
% Test of integration using table data
     clear
     load intdata.dat
     x = intdata(:,1);
     y = intdata(:,2);
     h = x(2) - x(1);
```

```
exact = 1.1064840;

% Trap rule
t = trap_fun(h,y);
rt = (t-exact)/exact

% Simpsons rule
s = simp_fun(h,y);
rs = (s-exact)/exact

% rt =
%    -1.024634789115796e-01
% rs =
%     1.530614089313202e-03
```

As can be seen, the trapezoidal rule is not very accurate, but Simpson's rule is very accurate. In fact, given the number of data points and accuracy of the data, it would be hard to do much better than this.

Look at the data file `intdata2.dat`. This data is shown in Figure 1. This table has an even number of

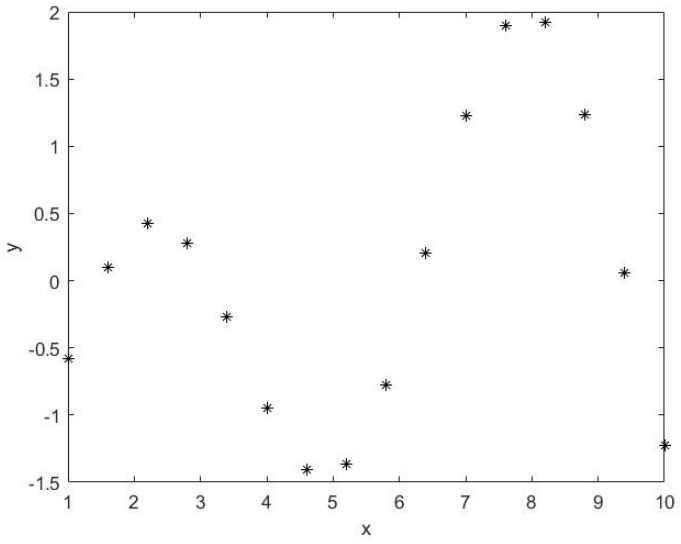

Figure 1: Raw data for integration example.

points, so Simpson's rule can't be used. Running the previous script gives a trapezoidal method error of

```
rt =
    -8.910386413178983e-02
```

Adding a single point improved the integral estimate. These two examples indicate that more data points would be useful. This leads to the notion of `interpolation`.

3 Interpolation

One of the reasons that we have been working with tables of values is that when equations are solved computationally, the solution is frequently expressed as a table of values. For example, it is easy (though somewhat tedious) to show that the solution to the second-order, constant coefficient ordinary differential equation

$$y''(t) - y'(t) + 6y(t) = e^{-t},\ y(0) = 1, y'(0) = 0$$

is

$$y(t) = \frac{1}{8}e^{-t} - e^{-\frac{t}{2}}\left(\frac{5}{8\sqrt{23}}\sin\left(\frac{\sqrt{23}}{2}t\right) - \frac{7}{8}\cos\left(\frac{\sqrt{23}}{2}t\right)\right).$$

However, the nonlinear, third-order differential equation

$$f'''(t) + \frac{1}{2}f(t)f''(t) = 0,\ \ f(0) = 0, f'(0) = 0, f'(8) = 1$$

can't be solved using pencil and paper. This equation (and others like it) is critical in the analysis of airfoils to determine lift and drag characteristics. While we can't produce a pencil and paper solution, we can obtain an approximate solution that is represented as a table of values.

Prior to the widespread use of computers in the sciences, tables of various quantities (the trigonometric, exponential, natural log functions and properties of chemical substances) were used extensively. In some cases they still are.

Suppose you have a table of values for some function $y = f(x)$. A section of this table might look as shown in Table 1. What if you needed to know the value of y when $x = 1.63$? This value of x not in the table. Is

x	$y = f(x)$
$\vdots$	$\vdots$
1.60	2.58
1.70	2.82
1.80	3.06
$\vdots$	$\vdots$

Table 1: Table of values for some $y = f(x)$.

there a way to estimate the value of y in this situation? Fortunately, the answer is yes. The process of using values in a table to estimate values that are not in the table is called *interpolation*. There are many ways to do this, but we will focus only on two of these; linear interpolation and cubic splines.

3.1 Linear Interpolation

The easiest way to obtain an estimate for y when the known x value is not in the table is linear interpolation. Notice that the desired value $x = 1.63$ lies between the table values for $x = 1.6$ and $x = 1.7$.

x	$y = f(x)$
$\vdots$	$\vdots$
1.60	2.58
1.63	???
1.70	2.82
1.80	3.06
$\vdots$	$\vdots$

Table 2: Location of $x = 1.63$ in the table.

To determine an estimate for the corresponding value of y, use the table values on either side of $x = 1.63$. These two values can be used to find an equation for the line between them. This line is called the *interpolant*. The slope of this line is

$$m = \frac{\Delta y}{\Delta x} = \frac{2.82 - 2.58}{1.7 - 1.6} = \frac{0.24}{0.1} = 2.4.$$

To determine the y-intercept, use the formula

$$y = mx + b$$

x	$y = f(x)$
1.60	2.58
1.70	2.82

Table 3: Points used for computing the linear interpolant.

and substitute in the value of m and either pair of x, y values from the table.

$$2.58 = 2.4 \cdot 1.6 + b.$$

Then

$$b = 2.58 - 2.4 \cdot 1.6 = 2.58 - 3.84 = -1.26.$$

The equation of the interpolant is then

$$y = 2.4x - 1.26.$$

This can be used to estimate y when $x = 1.63$.

$$y = 2.4(1.63) - 1.26 = 3.912 - 1.26 = 2.652.$$

This should be rounded back to the precision of the other y values, so we obtain

$$y \approx 2.65 \quad \text{for} \quad x = 1.63.$$

3.1.1 An Algorithm for Linear Interpolation

The process above can be translated into an algorithm. Given a table of values $(x, f(x))$ and a value x^* that lies within the range of the table, estimate the value of $y^* = f(x^*)$.

1) Determine which two x values in the table bracket the value x^*. Call these values x_1 and x_2.

2) Look up the corresponding values y_1 and y_2.

3) Use these to compute the slope

$$m = \frac{y_2 - y_1}{x_2 - x_1}.$$

4) Compute the y-intercept from

$$b = y_1 - mx_1.$$

5) Estimate y^* from

$$y^* = mx^* + b.$$

3.1.2 Accuracy

A valid question that can be asked is how accurate is the linear interpolation process? It turns out that this depends on several factors.

The first of these is the function itself. In order for linear interpolation to work, the function must be a smooth function (*i.e.*, one that has derivatives).

The second is the spacing of the x values in the table. An important assumption is that the spacing of the x values is small enough that there are no relative maxima or minima between any two successive points in the table. This is easy to guarantee in situations where the function is strictly increasing or decreasing (as is the case in many tables of thermodynamic properties), but becomes more difficult to guarantee for other functions.

Another consideration is the precision of the table values. In the example above, the y values are known only to 3 digits. This means that the interpolation process is going to be accurate to (at most) 3 digits.

A formal mathematical analysis of the problem (which is based on Taylor series) reveals that the absolute error in y^* satisfies

$$|y^* - y_{exact}| \leq \frac{(x_2 - x_1)^2}{2} \max_{x_1 \leq x \leq x_2} |f''(x)|.$$

This formula indicates that the error depends on the distance between the two x values used to compute the interpolant, but also on the magnitude of $f''(x)$ between these x values. In many situations, $f(x)$ itself is unknown, which means that $f''(x)$ is also unknown. However, if the table spacing is fine enough, then $f(x)$ is nearly linear between any two successive x values, which means that $f''(x)$ will be very small. In fact, many of the printed tables that are still used are generated in a manner such that linear interpolation will be accurate to within 1 digit in the last place of the interpolated value.

3.2 Getting More Points Using Linear Interpolation

Suppose we had the idea that we could use linear interpolation to generate a table that had 101 points in it. The code to do this is in the `growlin` function. Using the `int_ex2` driver script to test this gives

```
>> int_ex2
rt =
    -8.633655796197683e-01
rs =
    -8.329909876690488e-01
```

As can be seen, this did not improve the accuracy. In fact, it made it worse. If you change the value of 100 in the `int_ex2` script, you will notice similar behavior.

The reason for this can be seen by plotting the interpolated points against the raw data. As can be seen in Figure 2, this process is not good. The local minimum and maximum points are being cut off rather than being filled in with a smooth curve.

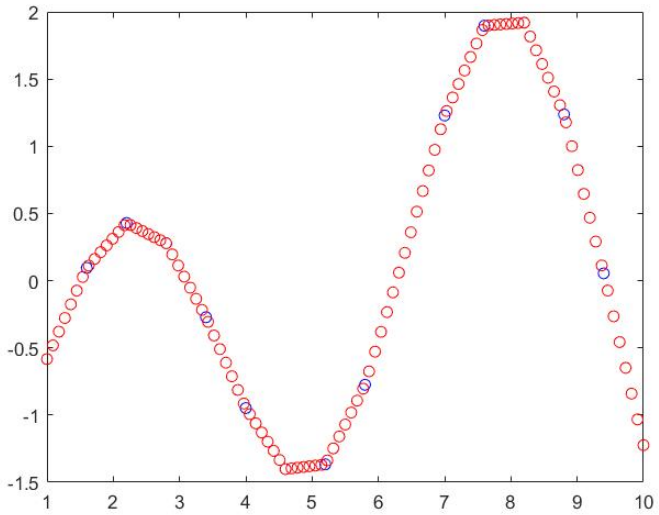

Figure 2: Using linear interpolation to get more data points. Notice that this doesn't seem to work very well.

The reason for this is that the data in the table is too coarse for linear interpolation to be effective. How can this be improved? When transitioning from the trapezoidal method to Simpson's rule, the points were taken in groups of three and the quadratic that goes through them was integrated. We could adopt a similar train of thought here. Instead of linear interpolation, we could use *quadratic* interpolation. In this case, we could take 3 points from the table that bracket x^*, compute the quadratic function that goes through them, then compute this quadratic function at x^*. This can be done, but we will skip to the punchline and state that this will not work much better, again due to the coarse nature of the raw data.

We need something better, but more importantly we would like to get back to the idea of using more advanced computing methods that are already programmed. This is more efficient because it will take less time and hopefully incorporate a much wider variety of special or pathological cases than we could account for ourselves.

3.3 Cubic Splines

What we would like for an interpolation process is something that is easy to use, reasonably accurate and only requires the data values in the table. The closest computational object that fits these criteria is called a *cubic spline*.

An example of a cubic spline is shown in Figure 3. There are $n+1$ points in the table and these are used

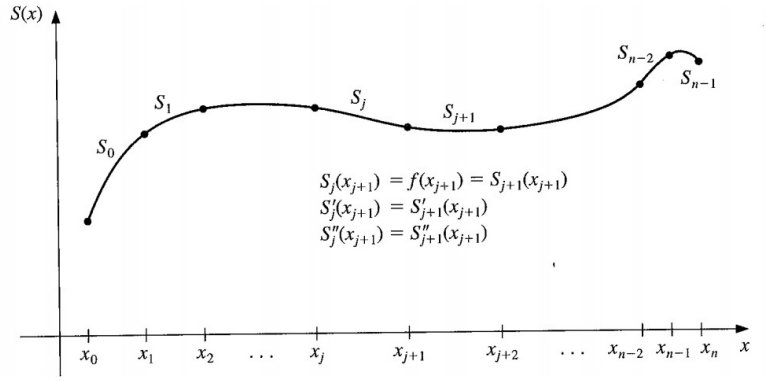

Figure 3: An example of a cubic spline. A table of $n+1$ points is used to construct a sequence of n cubic polynomials, S_j. Note that the points start numbering from 0. Taken from *Numerical Analysis*, Burden, Faires, 9th Edition.

to create n cubic polynomials. In Figure 3, these cubics are indicated by $S_0(x), S_1(x), \ldots, S_n(x)$. Note how these cubics go through all the data points in the table and that the value of $S_0(x_1) = S_1(x_1)$, the value of $S_1(x_2) = S_2(x_2)$., an so on. This is called the continuity condition.

What makes a cubic spline so effective as an interpolation process is that rather than simply fit the points in the table, it attempts to fit the behavior of both the slope and curvature of the data in the table. The *derivative condition* states that the slopes of the cubics should match where one cubic ends and the next begins. Mathematically, this means that $S_0'(x_1) = S_1'(x_1)$; $S_1'(x_2) = S_2'(x_2), \ldots$. Similarly, the *curvature condition* states that $S_0''(x_1) = S_1''(x_1)$; $S_1''(x_2) = S_2''(x_2), \ldots$.

There is one drawback to the cubic spline and that is that the points in the table are not quite sufficient to define it. Two additional conditions beyond the raw data are needed. Because of this, there are multiple versions of cubic splines. The three most common are:

- **Natural Spline** - This is the most common spline. For this spline, the second derivative $f''(x)$ of $f(x)$ is assumed to be zero at the endpoints. With these conditions no additional information beyond what is in the table of values is required.
- **Not-a-Knot** - For this spline, the condition that the spline have a continuous third derivative at points x_1 and x_{n-1} in Figure 3 is imposed. Again, with this assumption no additional information beyond what is in the table of values is required.
- **Clamped Spline** - For this spline, known values of $f'(x)$ at the endpoints of the table are required. If these values are not known exactly, estimates can be obtained from the values in the table.

3.3.1 Building the Spline

Fortunately MATLAB has a built-in function that will build either a not-a-knot or clamped spline. For now, assume we are building the not-a-knot spline. To build the spline, do

```
load intdata2.dat;
x = intdata2(:,1);
y = intdata2(:,2);
S = spline(x,y);
```

If you look at `S` you can see that it is an unusual variable. S is called a structure. As with a vector, a structure is a way to store a large amount of data in a single variable. However, unlike a vector, a structure can store many different types of data at once (similar to a list in Python). In particular, if you look at `S.coefs`, you can see this is a matrix with 15 rows and 4 columns. It has 15 rows because the table has 16 raw data points and hence the spline is composed of 15 cubics. The 4 columns are the coefficients of the cubic on each interval. The first few of these cubics are shown below.

```
      a         b         c         d
-------------------------------------
-0.1443   -0.2191    1.3142   -0.5832  S1 = ax^3 + bx^2 + cx + d
-0.1443   -0.4788    0.8954    0.0953  S2 = ax^3 + bx^2 + cx + d
 0.0748   -0.7385    0.1651    0.4290  S3 = ax^3 + bx^2 + cx + d
 0.2313   -0.6041   -0.6391    0.2790  S4 = ax^3 + bx^2 + cx + d
```

3.3.2 Using the Spline

Once we have created the spline, we can use it like any other function. For example, suppose we want to interpolate the value of the table at $x = 2.56$. This can be done using the `ppval` function.

```
>> x = 2.56;
>> y = ppval(x,S)
y  = 0.3962
```

The exact value of the raw data (to 4 digits) for $x = 2.56$ is 0.3970, so this seems to have good accuracy.

3.3.3 Accuracy of the Cubic Spline

A formal proof of the accuracy of the interpolated values obtained from the spline reveals that the absolute error in the values on any section of the spline satisfy

$$|y^* - y_{exact}| \leq \frac{(x_2 - x_1)^4}{24} \max_{x_1 \leq x \leq x_2} |f''''(x)|$$

where x_1 and x_2 are the endpoints of the spline. This is generally going to be a small number unless the magnitude of $f''''(x)$ is extremely large.

3.3.4 Growing the Table with the Spline

We previously saw that using linear interpolation to increase the values in the table is not going to be very accurate. Does this improve when we try to use the spline to do this? The `spline_ex` script computes the spline at 101 points and plots these. As can be seen in Figure 4, the spline does a much better job of filling in the gaps in the data.

3.3.5 Does This Improve the Integral Approximation?

Now that we seem to have a method for accurately increasing the values in the table, return to the original problem. We want to integrate the function in the table, but the trapezoidal method is not good, we can't use Simpson's rule because there are an even number of data points and using linear interpolation to get more points didn't work. Does using the spline to get more points improve the accuracy of the integral approximation? The `int_ex3` script will test this.

```
>> int_ex3
rt =
    -8.639096605889516e-03
rs =
    -6.229433511548368e-04
```

As can be seen, this is a large improvement compared to linear interpolation. Given that the raw data is only accurate to 4 digits, the Simpson's rule result is very good.

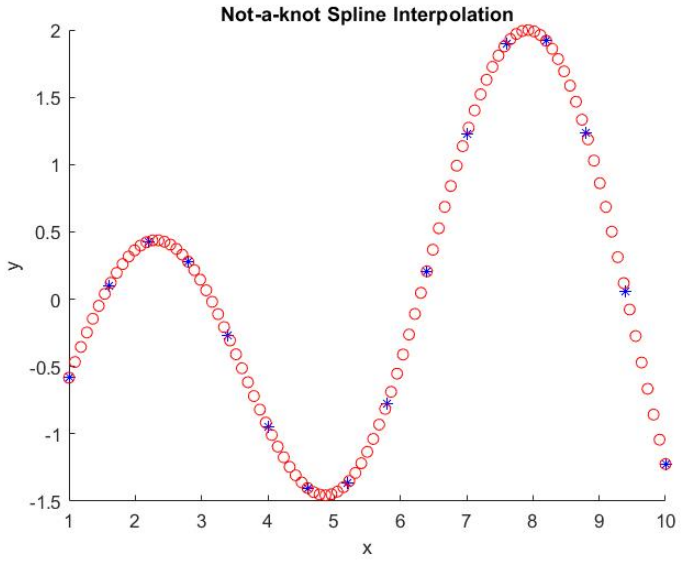

Figure 4: Not-a-knot spline built from the raw data.

3.3.6 Using the `integral` Function with a Spline

We have seen that one drawback to the traditional trapezoidal and Simpson's rule is that we don't know how many points to use in order to obtain a desired level of accuracy. We have also seen that using the `integral` function gives the desired accuracy automatically (provided the integral exists). This seems to imply that sending the spline as the integrand to the `integral` might be the best way to go. Can this be done? Fortunately the answer is yes. We can do this using

```
>> sp = integral(@(x)ppval(x,S),1,10)
```

Note that we are sending in the spline structure `S` as a parameter to the `integral` function. The resulting relative error in the spline integration is

```
rsp =
-6.385960751810642e-04
```

which is about the same error as Simpson's rule.

Finally, while the x-coordinates in the example table we have been using are equally spaced, this is not a requirement for the spline function.

4 Summary

In this section, we have leaped over some significant intermediate steps and theory but ultimately arrived at what we wanted to be able to do. We have only a table of values of raw data points and we need to obtain accurate approximations to the definite integral. We used a cubic spline to obtain an accurate way to interpolate values in the table and then sent this spline as input to the `integral` function. If you look at the final version of the script (`int_ex4`), the computational portion consists of only two lines. Moreover, this process is providing an integral that is accurate to the same precision as the raw data.

The quality of the cubic spline leads to the question: Would a quartic (4th degree polynomial) or quintic (5th degree) spline work even better? The answer is yes in theory, but no in practice. The reason for this is that the raw data does not provide enough information to build these splines. Recall that even for the cubic spline, two additional conditions need to be specified. Quartic and quintic splines would require many more additional conditions.