Inverse Interpolation

Leigh J. Little SUNY Brockport December 7, 2022

Contents

1 Introduction

We have examined interpolation. This is the process of using values in a table to predict values that are not in the table. It is probably more appropriate to say that we have been examining forward interpolation. A general statement of forward interpolation is: Given a table and a value x^* not in the table, determine the value of y^* satisfying $y^* = f(x^*)$.

Another problem that frequently arises is *inverse interpolation*. A general statement of inverse interpolation would be: Given a table and a value of y^* not in the table, determine the value of x^* satisfying $y^* = f(x^*)$.

Inverse interpolation is more challenging than forward interpolation. This is because if the table describes a function in the mathematical sense, then the function will pass the vertical line test and there will be a unique y^* for a given x^*.

This is not the case for inverse interpolation. Many functions do not satisfy the horizontal line test (See Figure 1) and as a result there can be many potential values of x^* for a given y^*.

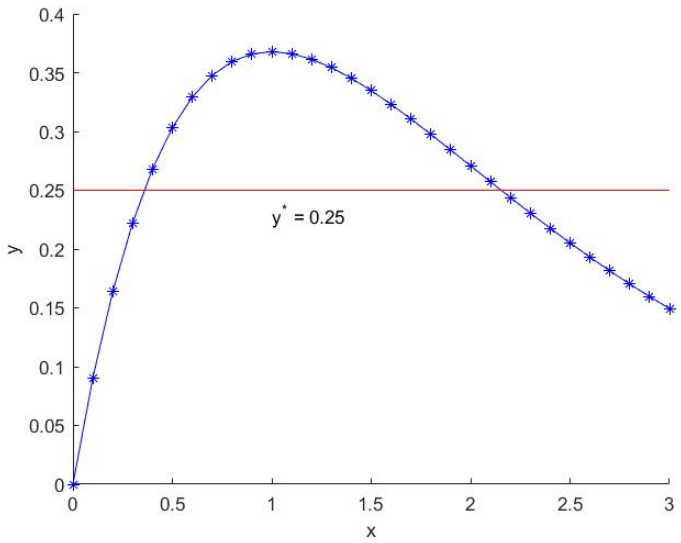

Figure 1: Example of inverse interpolation.

2 Simple Inverse Interpolation

As with forward interpolation, there are many ways to perform inverse interpolation. We will describe one method that is simple and works well provided that the spacing of the x-coordinates is fine enough that there is only one y^* in the neighborhood of the desired x^*.

The inverse interpolation process is straightforward in this case.

1) Locate two y-coordinates in the table that bracket the desired y^* coordinate.

2) Choose four points from the table in the vicinity of these two bracketing points. Generally you would want two points less than y^* and two points greater than y^*, but this needs to be adjusted in the event that the bracketing points are close to the end points of the table.

3) Use the four points to build an inverse spline. This is the same process we have been using, but the order of the inputs is reversed. Instead of doing

```
S = spline(xs,ys)
```

do

```
S = spline(ys,xs)
```

where `xs` and `ys` are arrays containing the four points from Step 2 above.

4) Once the spline is fit, you can obtain the desired x^* value by doing

```
xstar = ppval(S,ystar)
```

3 Example

Suppose we want the two x^* values that satisfy $f(x^*) = 0.25$ as shown in Figure 1. The most difficult part of the process is locating the necessary bracketing points. Because the function can cross the horizontal line in Figure 1 in a positive-to-negative or negative-to-positive sense, it would appear that a complicated `if` test is required:

```
n = length(x);
ystar = 0.25;
for i = 1:n-1
    if( (y(i) > ystar && y(i+1) < ystar) | (y(i) < ystar && y(i+1) > ystar))
       ....  found the index of the first bracket point.
       ....
end
```

A more efficient test can be devised by noting that the value of $y - y^*$ changes sign whenever the function crosses y^*.

```
n = length(x);
ystar = 0.25;
xstar = [];
numxstar = 0;

for i = 1:n-1
    if( (y(i)-ystar)*(y(i+1)-ystar) < 0)
      numxstar = numxstar+1;
      xs = x(i-1:i+2);
      ys = y(i-1:i+2);
      S = spline(ys,xs);
      xstar(numstar) = ppval(S,ystar);
    end
end
```

The table of values from Figure 1 are in the `invdat.dat` file. Using this we see that the two estimates of the x^* values that give y^* are 0.3570 and 2.153.

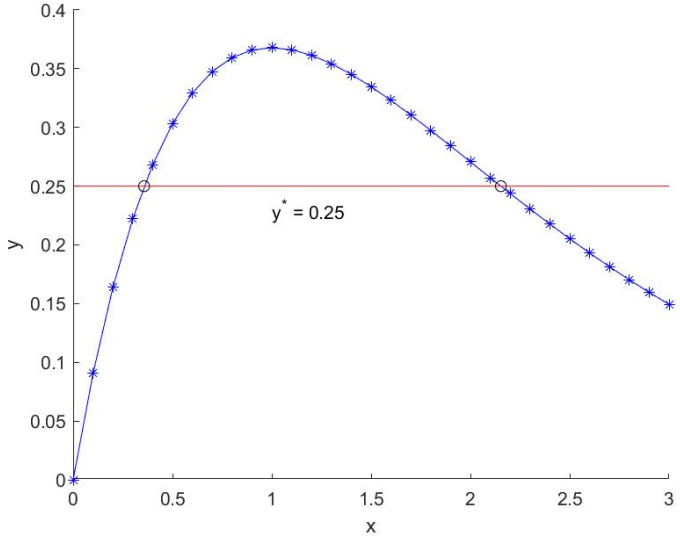

Figure 2: Plotting the x^* estimates.