Chapter 19 Interpolation

19.1 One-Dimensional Interpolation

Empirical data obtained experimentally often times conforms to a fixed (deterministic) but unkown functional relationship. When estimates of other points on the function are required, procedures for obtaining those estimates falls under the general category of Interpolation. The simplest case occurs when the data consists of a set of points each made up of an independent variable and a single dependent variable observation.

The MATLAB function '`interp1`' default mode is designed to linearly interpolate, as in a Table Lookup, empirical data to estimate the unknown function corresponding to a set of independent variable values. For example, suppose a consumer testing magazine is interested in the repair costs for a particular luxury passenger vehicle after its been in an accident. The test procedure calls for head-on collisions of the vehicle with a stationary object at various speeds. The cost of restoring the vehicle to its pre-crash condition is determined. Tabulated results after several collisions are given below.

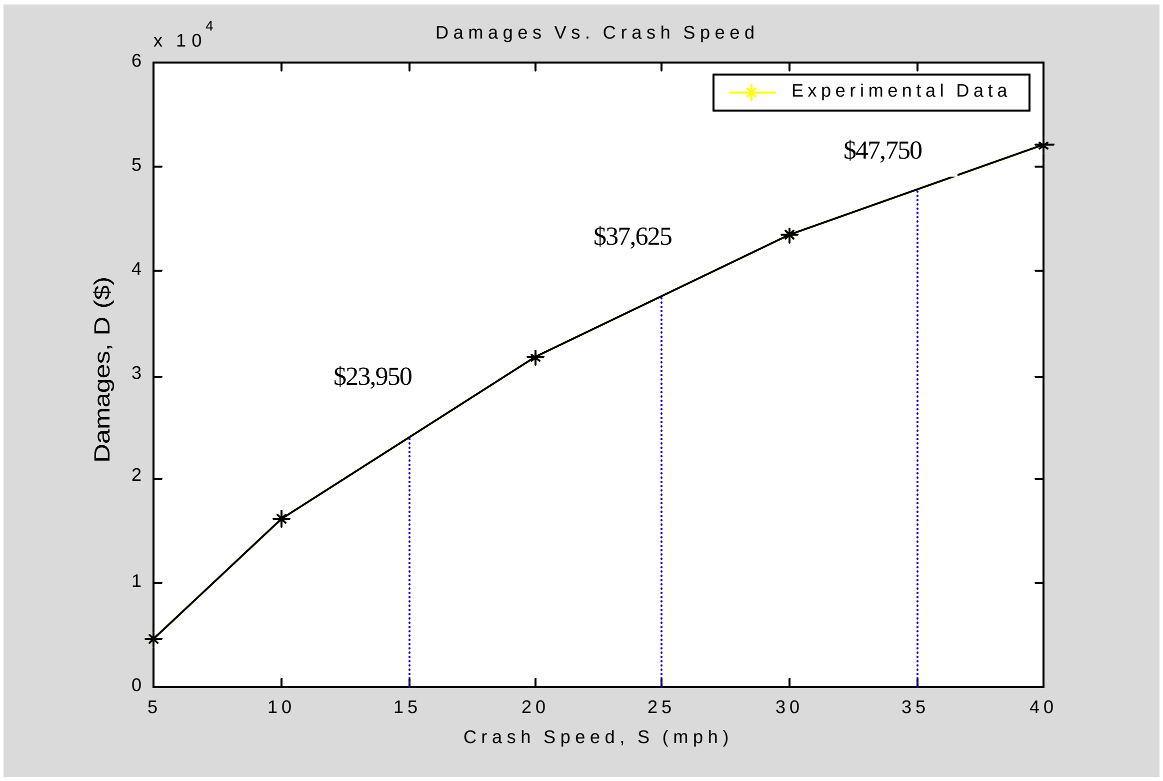
Speed (*S*) mph	Damages (*D*) $
5	4500
10	12150
20	23750
30	43500
40	61000

The first example demonstrates how to estimate the unknown function $D = f(S)$, which generated the tabulated data points, at other speeds between 5 and 40 mph.

Example 19.1.1

```
s=[5 10:10:40] ; % Speed data from table
d= [4500 16150 31750 43500 52000]; % Damages data from table
plot(s,d,'*-k') % Plot raw data
hold on
si=[15 25 35]  % Speeds at which damage estimates are required
d_est=interp1(s,d,si) % Estimate damages at 15,25,35 mph
plot([15 15],[0 d_est(1)],'b:') % Plot vertical line to 1st estimate
plot([25 25],[0 d_est(2)],'b:') % Plot vertical line to 2nd estimate
plot([35 35],[0 d_est(3)],'b:') % Plot vertical line to 3rd estimate
xlabel('Crash Speed, S (mph)')
ylabel('Damages, D ($)')
title('Damages Vs. Crash Speed')
legend('Experimental Data')
```

```
si =     15     25     35
d_est =      23950        37625        47750
```



More accurate results are possible by specifying cubic spline interpolation, a method which connects the given data points by 3rd order polynomials over each interval. The transition between adjacent splines (cubic polynomials) is guaranteed to be smooth because their lower order derivatives are equal on either side of the data points. The following example illustrates the use of cubic splines.

Example 19.1.2

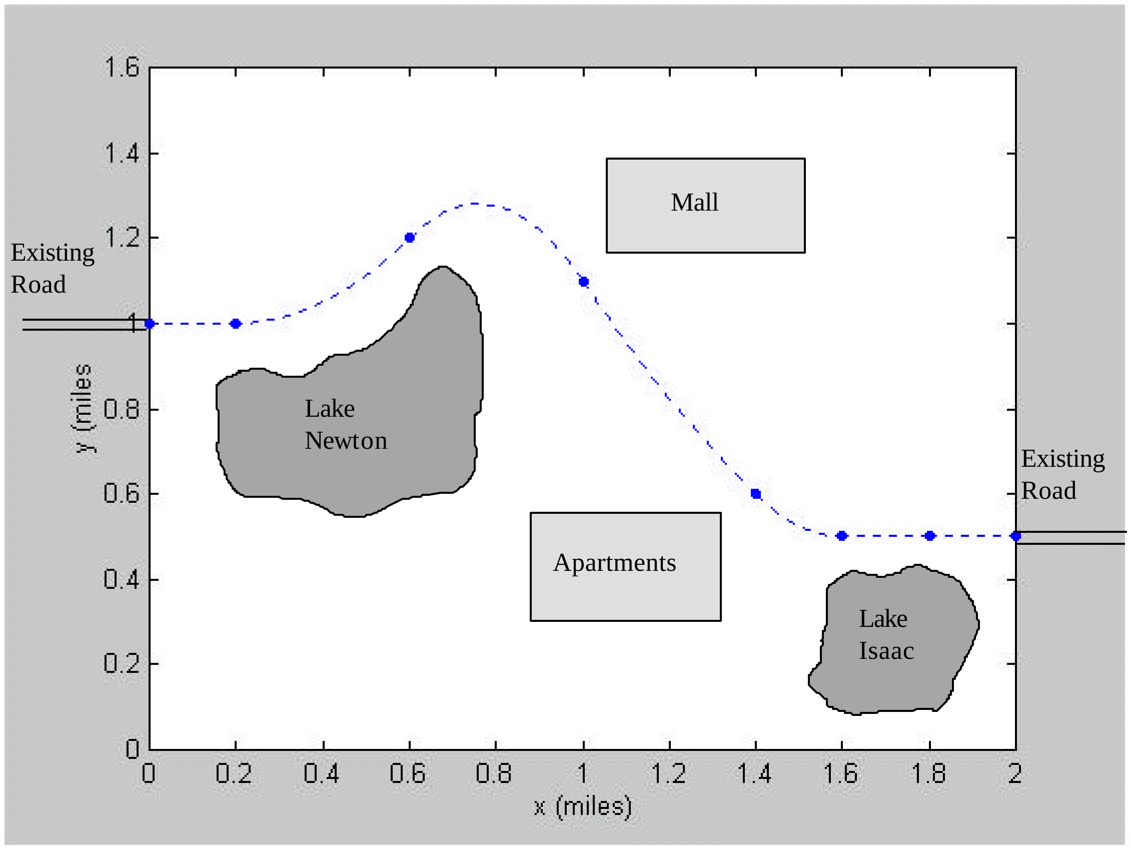
A highway engineer must design a new road which will connect two existing roads. In order to maintain safe clearances from a lake and several structures, measurements were taken to determine several points along the centerline of the proposed road. The geographical locations of these points, relative to some point of reference, are given in the Table below. Cubic splines are to be used to specify the equations of the centerlines for each road segment.

x_i (miles)	y_i (miles)
0.0	1.0
0.2	1.0
0.6	1.2
1.0	1.1
1.4	0.6
1.6	0.5
1.8	0.5
2.0	0.5

The following MATLAB code plots the given centerline points, determines the cubic splines, and plot the continuous centerline comprised of the splines. The results are illustrated in the figure which shows the smooth centerline trajectory of the proposed connector between the existing roads. (The additional graphics are not part of Example 19.1.2 and were added later)

Example 19.1.2

```
x=[0 0.2 0.6 1 1.4 1.6 1.8 2];   % Centerline
y=[1 1 1.2 1.1 0.6 0.5 0.5 0.5]; % Data Points
v=[0 2 0 1.6]; % Set scale for plotting
plot(x,y,'.')     % Plot Centerline Data Points
xlabel('x (miles)')
ylabel('y (miles')
axis(v)
hold on
axis manual
xi=linspace(0,2,250); % Create x vector for plotting
yi=interp1(x,y,xi,'spline'); % Evaluate yi at xi using cubic splines
plot(xi,yi,'b')
```



Suppose the length of the road found in Example 19.1.2 is needed. The length of the road can be approximated by summing incremental steps, i.e

$$. \qquad S \approx \sum_i \Delta s_i = \sum_i \left[(\Delta x)^2 + (\Delta y_i)^2 \right]^{1/2}$$

Example 19.1.3

```
x=[0 0.2 0.6 1 1.4 1.6 1.8 2];
y=[1 1 1.2 1.1 0.6 0.5 0.5 0.5];
n=1000;  % Number of points.  There are n-1 steps.
xi=linspace(0,2,n); % Create vector of x values of step endpoints
yi=interp1(x,y,xi,'spline'); % Calculate y at step endpoints
dx=xi(2)-xi(1); % Calculate the fixed dx for each step
s=0; % Initialize length calculation
for i=1:n-1
   dyi=yi(i+1)-yi(i); % Calculate y increment
   dsi=(dx*dx + dyi.*dyi).^0.5; % Calclulate road length increment
   s=s+dsi;
end
n,s
```

```
n =  100
s =  2.3597
 n = 1000
s =  2.3599
```

19.2 Two-Dimensional Interpolation

Two dimensional interpolation is similar to the one-dimensional case with the obvious difference that there are now two independent variables and a single dependent variable. The MATLAB function `interp2`' in default mode is equivalent to a two way table lookup with linear interpolation in both directions.

The table below represents measurements of a variable *z* at various *x* and *y* levels. If an estimate of *z* when *x*=0.5 and *y*=32 is needed, only four data points are required if the default linear interpolation is satisfactory.

y \ *x*	0.01	0.1	1	10	100
20	523	697	784	855	901
25	684	794	859	897	940
30	778	859	907	936	954
35	830	880	931	957	960
40	864	926	952	983	1002
45	881	932	964	999	1018

$z=f(x,y)$

Example 19.2.1

```
x1=0.1; x2=1.0; y1=30; y2=35;
z11=859; z12=907; z21=880; z22=931;
x=[x1 x2];
y=[y1 y2];
z=[z11 z12; z21 z22];
z_est=interp2(x,y,z,0.5,32)

z_est =  889.2667
```

Using cubic interpolation to estimate the same value requires the entire data set.

Example 19.2.2

```
x=logspace(-2,2,5); % Set x=[0.01 0.1 1 10 100]
y=20:5:45; % Set y=[20 25 30 35 40 45]
z=[523 697 784 855 901; 684 794 859 897 940; 778 859 907 936 954; 830
880 931 957 960; 864 926 952 983 1002; 881 932 964 999 1018];
z_est=interp2(x,y,z,0.5,32,'linear') % Two way linear interpolation
z_est=interp2(x,y,z,0.5,32,'cubic')

z_est =  889.2667
z_est =  990.9967
```