12 Curve-Fitting to Relate s-t, v-t, and a-t Graphs

Ref: Hibbeler § 12.3, Bedford & Fowler: Dynamics § 2.2

The relationships between a particle's position, s, velocity, v, and acceleration, a, over time are all related. When the relationship between a particle's velocity (for example) and time is described by a mathematical equation, that equation can be integrated to obtain the particle's position, and differentiated to determine the particle's acceleration. But what if the velocity vs. time relationship is described by a set of data values or a graph, rather than a mathematical equation? You can still obtain position and acceleration information, either directly from the data values (numerical approximations of integration and differentiation), or by fitting an equation to the data, then using calculus on the resulting equation. The curve-fitting approach will be demonstrated here.

Example: Fitting a Polynomial to Velocity Data

A driver on a straight track accelerates his car from 0 to 75 mph in 14 seconds, and then allows the car to begin to slow. A data recorder connected to the speedometer recorded the data shown below. Since the car is moving along a straight path, the car's speed and velocity are equal.

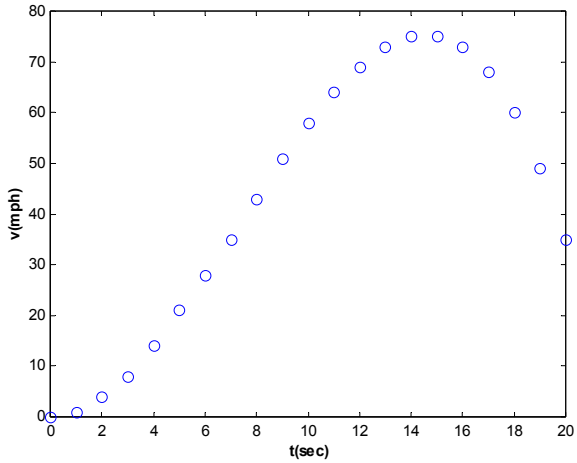
t (sec.)	v (mph)
0	0
1	1
2	4
3	8
4	14
5	21
6	28
7	35
8	43
9	51
10	58
11	64
12	69
13	73
14	75
15	75
16	73
17	68
18	60
19	49
20	35

Fit a curve to the v-t data, and then use the equation of the curve to plot s-t, v-t, and a-t graphs.

Solution – First Attempt

First, let's plot the data to see what we're working with.

```
» t = 0:20;
» v = [0 1 4 8 14 21 28 35 43 51 58 64 69 73 75 75 73 68 60 49 35];
» plot(t,v,'o');
» xlabel('{\bf t(sec)}');
» ylabel('{\bf v(mph)}');
```



The graph shows the increasing velocity for the first 14 seconds then the velocity drops off again. It looks like a smooth curve, and clean (not noisy) data.

A commonly used approach to fitting a curve to a set of data values is polynomial regression. We might try, for example, the following second-order polynomial.[1]

$$v_p = b_1 t^2 + b_2 t + b_3$$

MATLAB's Statistics Toolbox provides a function for fitting linear[2] and nonlinear models such as this polynomial, the nlinfit() function. This function calculates the coefficient values that provide the best fit of the model equation to the data. The regression model is described by either an inline function or an m-file containing the desired function of the independent variable (t, in this case) and coefficients . For our second-order polynomial, the vector could be written as follows:

```
» F = inline('b(1).*t.^2 + b(2).*t  ', 'b', 't')
F =
     Inline function:
     F(b,t) = b(1).*t.^2 + b(2).*t
```

Note: We want the velocity to be zero when t = 0. The way to force this to happen is to leave off the constant term in the F(b,t) function.

Then, nlinfit() uses the t and v values (as vectors), the F(b,t) function shown above, and a vector of initial guesses for each coefficient in F to calculate the b values that produce the "best" fit of the polynomial to the data.

```
» b = nlinfit(t,v,F,[0 0]);
» b
b =
```

[1] A second-order polynomial is not a good choice for this data. A parabola is a second-order equation, and the data shows a lot more curvature than a simple parabola. It will require a higher-order polynomial to adequately fit this data – but the goal of the first attempt is to demonstrate that the "best fit" polynomial may still be a very poor fit to the data.

[2] For curve-fitting, the variables are the model coefficients, the b values. All of the times are known from the data set. This polynomial is linear in the b values, so it is a linear model for curve fitting.

-0.2706

8.2273

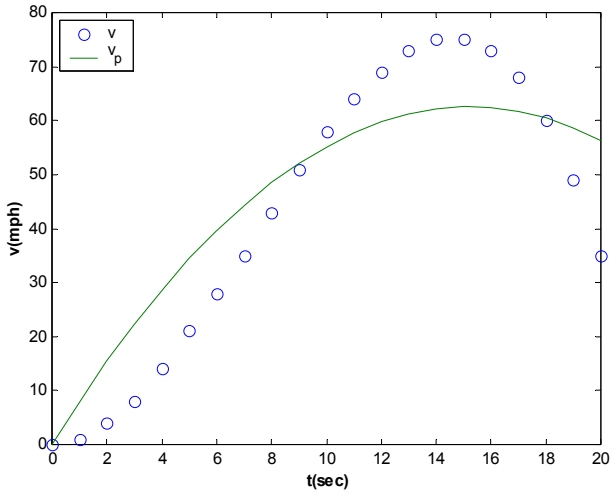
These coefficients tell us that the regression model is

$$v_p = -0.2706\,t^2 + 8.2273\,t + 0$$

The subscript p indicates that the equation is used to calculate predicted values of velocity.

The calculated coefficients provide the *best* possible fit of the model equation to the data, but that does not guarantee that the equation is a *good* fit. You should always calculate predicted values with the regression equation and compare them to the original data to visually determine if the regression equation is actually fitting the data.

```
» v_p = b(1).*t.^2  + b(2).*t;
» plot(t,v,'o',t,v_p);
» xlabel(' t(sec) ');
» ylabel(' v(mph) ');
» legend('v','v_p',2)
```



Clearly, the regression curve (solid line) is not doing a good job of fitting the data.

Solution – Second Attempt

We need a better regression equation. We'll try a third-order polynomial.

$$v_p = b_1 t^3 + b_2 t^2 + b_3 t + b_4$$

The inline function F(b,t) gets another element…

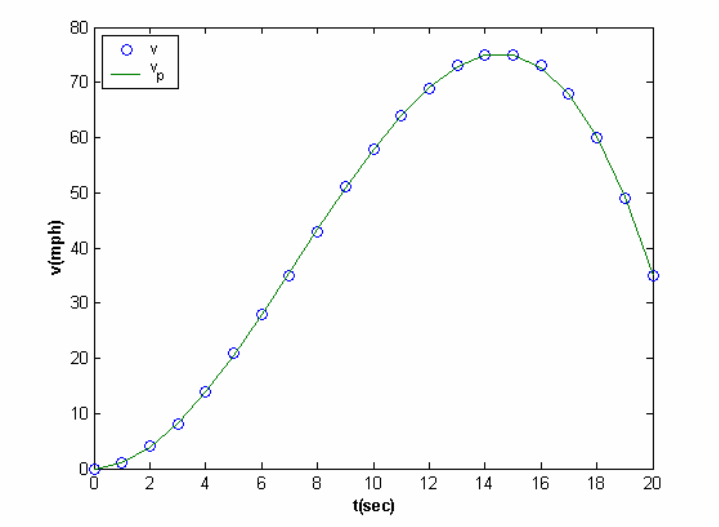
```
» F = inline('b(1).*t.^3 + b(2).*t.^2 + b(3).*t','b','t');
```

…and nlinfit() now finds three b values…

```
» b = nlinfit(t,v,F,[0 0 0]);
» b
b =
  -0.0493
   1.0762
  -0.0403
```

Again, we calculate predicted velocities to check the fit.

```
» v_p = v_p = b(1).*t.^3 + b(2).*t.^2  + b(3).*t;
» plot(t,v,'o',t,v_p);
» xlabel(' t(sec) ');
» ylabel(' v(mph) ');
» legend('v','v_p',2)
```



The new model is doing a good job of fitting the data. The regression equation that fits the data is

$$v_p = 0 - 0.04\,t + 1.076\,t^2 - 0.049\,t^3$$

This equation can be used to generate the s-t and a-t graphs.

MATLAB can integrate the v_p equation to get position, s, and differentiate the equation for acceleration, a, using the polyint() and polydiff() functions respectively. MATLAB represents polynomial's as vectors of coefficients. The first vector element is the coefficient on the highest ordered term and the last vector element is the constant. Therefore, we need to add a zero to the end of our b vector so it is treated as a 3rd order polynomial by MATLAB.

```
» b(4) = 0
b =
  -0.0493
   1.0762
  -0.0403
        0

» b(4) = 0;
» b_a = polyder(b)
b_a =
  -0.1480    2.1524   -0.0403

» b_s = polyint(b)
b_s =
  -0.0123    0.3587   -0.0202         0         0
```

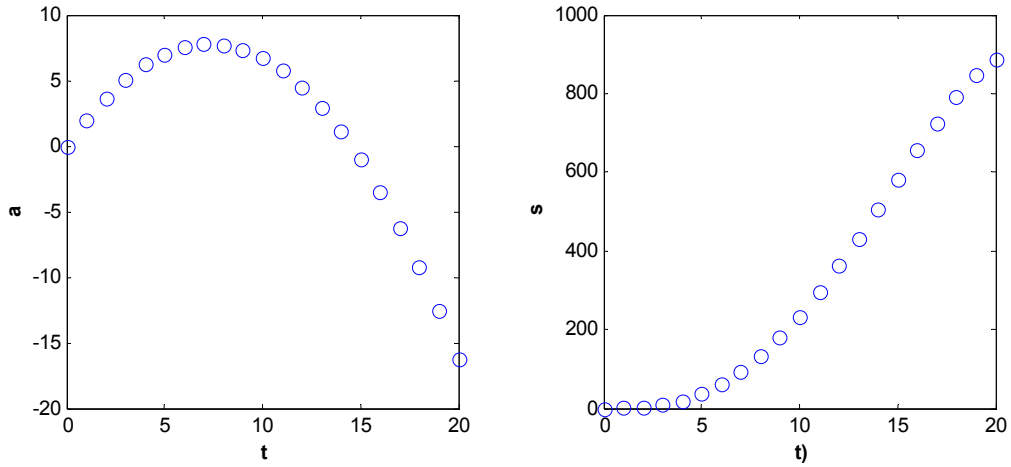
Note: When MATLAB integrates, it does not add the constant of integration. It effectively assumed the car was at s = 0 at t = 0.

```
» a = b_a(1).*t.^2 + b_a(2).*t + b_a(3);
» s = b_s(1).*t.^4 + b_s(2).*t.^3 + b_s(3).*t.^2 + b_s(4)*t;
```

With the equations above, acceleration and position vectors a and s have been calculated by MATLAB. All that remains is to plot the results.

```
» subplot(1,2,1);
» plot(t,a,'o');
» xlabel('t');
» ylabel('a');

» subplot(1,2,2);
» plot(t,s,'o');
» xlabel('t');
» ylabel('s');
```



There are some strange units on the values plotted here. The acceleration has units of mi/sec·hr, and the position, s, has units of mi·sec/hr. We can clean this up a bit by building the unit conversions into the calculations for a and s.

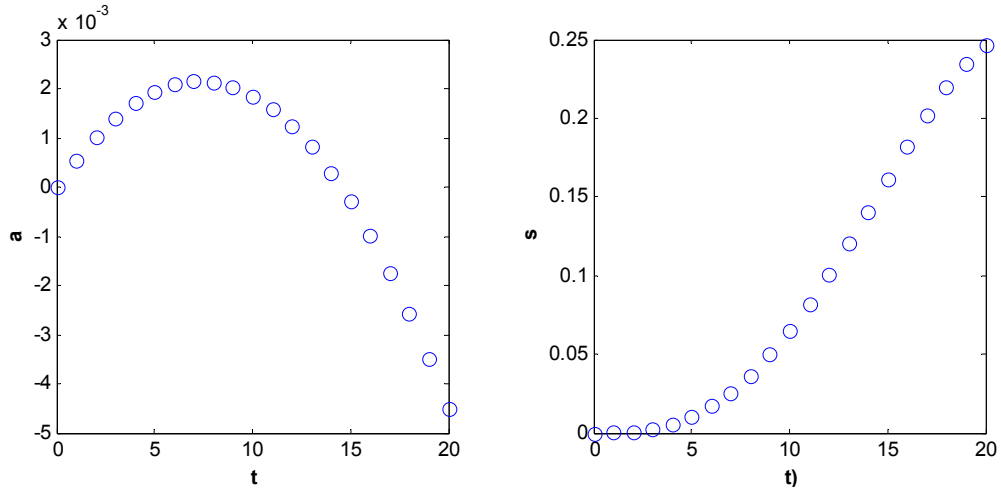
```
» a = a ./ 3600;
» s = s ./ 3600;

» subplot(1,2,1);
» plot(t,a,'o');
» xlabel('t');
» ylabel('a');

» subplot(1,2,2);
» plot(t,s,'o');
```

» xlabel('t');

» ylabel('s');



The units are now a [=] mi/sec^2 and s [=] miles.

Annotated MATLAB Script Solution

```
%Define the time and velocity vectors, and plot the v-t graph
t = 0:20;                                          %t = 0,1,2,...,20
v = [0 1 4 8 14 21 28 35 43 51 58 64 69 73 75 75 73 68 60 49 35];
plot(t,v,'o');
xlabel('{\bf t(sec)}');
ylabel('{\bf v(mph)}');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         First Solution Attempt - Second Order Polynomial                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define the polynomial using the inline function F(b,t).
F = inline('b(1).*t.^2 + b(2).*t','b','t');
%Find the best-fit coefficients using F(b,t).
b = nlinfit(t,v,F,[0 0]);
%Calculate predicted velocity values (at each t value) to see if the
regression equation actually fits the data.
v_p = b(1).*t.^2  + b(2).*t;
subplot(1,2,1);
plot(t,v,'o',t,v_p);
xlabel('{\bf t(sec)}');
ylabel('{\bf v(mph)}');
legend('v','v_p',2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Second Solution Attempt - Third Order Polynomial                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define the polynomial using the inline function F(b,t).
F = inline('b(1).*t.^3 + b(2).*t.^2 + b(3).*t','b','t');
%Find the best-fit coefficients using F(b,t).
b = nlinfit(t,v,F,[0 0 0]);
```

```
%Calculate predicted velocity values (at each t value) to see if the
regression equation actually fits the data.
v_p = b(1).*t.^3 + b(2).*t.^2  + b(3).*t;
subplot(1,2,2);
plot(t,v,'o',t,v_p);
xlabel('{\bf t(sec)}');
ylabel('{\bf v(mph)}');
legend('v','v_p',2);

%Convert b to a MATLAB polynomial vector
b(4) = 0;

%Take the derivative of b to find the equation for a.
b_a = polyder(b);
%Calculate a at each value of t
a = b_a(1).*t.^2  + b_a(2).*t + b_a(3);
%Simplify the units by building in the conversion from hours to seconds.
a = a ./ 3600;

%Integrate b to find the equation for s.
b_s = polyint(b)
%Calculate s at each value of t
s = b_s(1).*t.^4 + b_s(2).*t.^3  + b_s(3).*t.^2 + b_s(4)*t;
%Simplify the units by building in the conversion from hours to seconds.
s = s ./ 3600;

%Plot the results in a new figure.
figure;
subplot(1,2,1);
plot(t,a,'o');
xlabel('{\bf t}');
ylabel('{\bf a}');

subplot(1,2,2);
plot(t,s,'o');
xlabel('{\bf t)}');
ylabel('{\bf s}');
```