Modeling Change: Springs, Driving Safety, Radioactivity, Trees, Fish, and Mammals

Introduction

OBJECTIVE: Practice a modeling process: consider a behavior, observe data, fit a model, analyze the error, improve the model if appropriate, interpret the model, and make predictions.

Mathematical models help us better understand things in the world around us, like the behavior of springs, safe driving practices, radioactivity in medicine, the growth of trees and fish, and the biology of mammals. In this module you will learn how to construct mathematical models, how to analyze and improve them, how to use them to study the behavior you are modeling, and to make predictions. As you will see, a computer algebra system like *Mathematica* is a powerful tool that will aid you in your mathematical modeling.

Technology Guidelines

NOTE: If you have just finished a module, restart *Mathematica* or close the *Kernel* before executing a new module.
TO OPEN CELLS, put your cursor on the right cell bracket and double click.
INITIALIZATION CELLS
When asked if you want to ". . . automatically evaluate all the initialization cells in the notebook . . . ," respond by pressing the "Yes" button.
TO STOP AN EXECUTION
Select the *Kernel* pull-down menu and click on *Abort Evaluation.*
ORDER OF EXECUTION
Execute cells in the order given. Do not skip any Input cells within a given notebook.
SAVING NOTEBOOKS
You can save anytime to any directory you choose, and it is wise to save often.
However, before you do your final save, delete all your output by selecting the
Delete All Output selection under the *Kernel* pull-down menu.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, then shut down *Mathematica* and start it up again.

Part I: Vehicular Stopping Distance

Chapter 1, Section 7 (ET Section 4), Exercise 43

Observing the Data

A rule of thumb that is often given for safe following distance is to allow 2 seconds between your car and the car in front of you. Is this rule reasonable?

The following data set contains ordered pairs of values. The first element of each ordered pair is the traveling speed, and the second element is the total stopping distance, the distance traveled by the car during the driver's reaction time plus the distance required for the vehicle to come to a stop with full braking.

In[3]:=

```
data = {{20, 42}, {25, 56}, {30, 73.5}, {35, 91.
  {45, 142.5}, {50, 173}, {55, 209.5}, {60, 248
  {70, 343}, {75, 401}, {80, 464}}
```

Out[3]=

```
{{20, 42}, {25, 56}, {30, 73.5}, {35, 91.5},
 {40, 116}, {45, 142.5}, {50, 173},
 {55, 209.5}, {60, 248}, {65, 292.5},
 {70, 343}, {75, 401}, {80, 464}}
```

In[4]:=

```
TableForm[data, TableDirections → {Row, Colum
 TableHeadings →
  {None, {"speed(mph)", "stopping distance(f
```

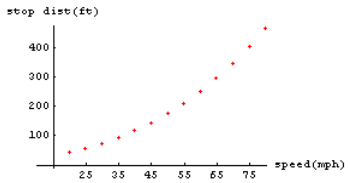
Out[4]//TableForm=

speed(mph)	20	25	30	35	40	45	50	55	60	65	70	75	80
stopping distance(ft)	42	56	73.5`	91.5`	116	142.5`	173	209.5`	248	292.5`	343	401	464

We plot the data to see if there is a recognizable pattern.

In[5]:=

```
xlabel = "speed(mph)";

ylabel = "stop dist(ft)";

p1 = ListPlot[data, PlotRange → {{10, 80}, Aut
  PlotStyle → {PointSize[0.020], RGBColor[1,
  AxesLabel → {xlabel, ylabel},
  Ticks → {Table[15 + i * 10, {i, 0, 27}], Automa
```



▪ Designing a Model

The data suggest a possible quadratic relationship. Now we use the **Fit[ ]** function to find the curve of best-fit or the regression curve.

In[8]:=

```
Clear[y];

y[x_] = Fit[data, {1, x, x^2}, x]
```

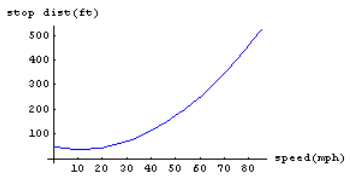
Out[9]=

$$50.0594 - 1.97013\,x + 0.0885914\,x^2$$

Next, we plot the regression curve and then show the plot of the data and the model together on the same graph.

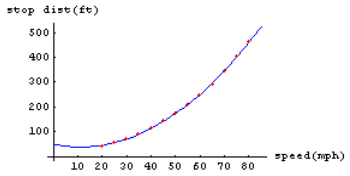
In[10]:=

```
p2 = Plot[y[x], {x, 0, 85}, PlotStyle → {RGBCol
  AxesLabel → {xlabel, ylabel},
  Ticks → {Table[0 + i * 10, {i, 0, 27}], Automat
```



In[11]:=

```
Show[p2, p1];
```



The best-fit quadratic function appears to fit the data very well.

Assessing the Errors

We can provide further verification by calculating the residual errors for each data point and plotting them. First, we calculate the stopping distances predicted by the model.

In[12]:=

```
predictedvalues = Table[{x, y[x]}, {x, 20, 80,
```

Out[12]=

```
{{20, 46.0934}, {25, 56.1758},
 {30, 70.6878}, {35, 89.6294},
 {40, 113.}, {45, 140.801}, {50, 173.031},
 {55, 209.691}, {60, 250.781},
 {65, 296.3}, {70, 346.248},
 {75, 400.626}, {80, 459.434}}
```

In[13]:=

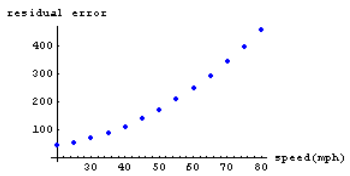
```
residuals = Table[{data[[i, 1]], predictedval
  {i, 1, 13}]
```

Out[13]=

```
{{20, 46.0934}, {25, 56.1758},
 {30, 70.6878}, {35, 89.6294},
 {40, 113.}, {45, 140.801}, {50, 173.031},
 {55, 209.691}, {60, 250.781},
 {65, 296.3}, {70, 346.248},
 {75, 400.626}, {80, 459.434}}
```

In[14]:=

```
ListPlot[residuals,
 PlotStyle → {PointSize[0.023], RGBColor[0, 0
 AxesLabel → {"speed(mph)", "residual error"}
```



Once again we should look at the error in relation to the size of the quantity being estimated. For all but the first data point, we can calculate the relative error as follows.

In[15]:=

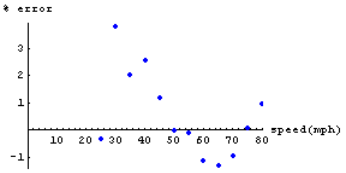
```
percenterrors =
 Table[{data[[i, 1]],
   (data[[i, 2]] - predictedvalues[[i, 2]]) / d
  {i, 2, 13}]
```

Out[15]=

```
{{25, -0.313972}, {30, 3.82611},
 {35, 2.0444}, {40, 2.58578},
 {45, 1.19214}, {50, -0.0181899},
 {55, -0.0913168}, {60, -1.12126},
 {65, -1.29904}, {70, -0.947012},
 {75, 0.0931737}, {80, 0.984038}}
```

In[16]:=

```
ListPlot[percenterrors,
 PlotStyle → {PointSize[0.021], RGBColor[0, 0
 PlotRange → {{0, 80}, All},
 AxesLabel → {"speed(mph)", "% error"}];
```



The maximum relative error of this model is about 0.6%.

Improving the Model

As indicated at the beginning of this module, a rule of thumb that is often given for safe following distance is to allow 2 seconds between your car and the car in front of you. To use this rule, you would note when the car in front of you passes some marker on or near the roadway, and then you count the time it takes you to get to the marker. This time should be at least 2 seconds. The rationale behind this rule is that if the car in front of you were suddenly to come to a complete stop, the 2-second separation distance would give you enough time to stop, to avoid hitting the vehicle in front of you. To test the 2-second rule of thumb, we will calculate how far your car travels in 2 seconds at various speeds and then compare these distances with the corresponding stopping distances in our data set.

The first entry in each element of the following table is the traveling speed in miles per hour, the second entry is the 2-second distance between the two cars in feet, and the third entry is the stopping distance in feet, taken from the test data. (To calculate the distance traveled in 2 seconds, we convert the traveling speeds from mph to ft/sec by using the conversion, 60 mph = 88 ft/sec.)

In[17]:=

```
data2 =
 Table[{data[[i, 1]], (data[[i, 1]] * 88. / 60.)
  {i, 1, 13}]
```

Out[17]=

```
{{20, 58.6667, 42},
 {25, 73.3333, 56}, {30, 88., 73.5},
 {35, 102.667, 91.5}, {40, 117.333, 116},
 {45, 132., 142.5}, {50, 146.667, 173},
 {55, 161.333, 209.5}, {60, 176., 248},
 {65, 190.667, 292.5}, {70, 205.333, 343},
 {75, 220., 401}, {80, 234.667, 464}}
```

In[18]:=

```
TableForm[data2,
 TableHeadings →
  {None, {"v(mph)", "2-sec separation dist(f
    "stopping dist(ft)"}}]
```

Out[18]//TableForm=

v(mph)	2-sec separation dist(ft)	stopping dist(ft)
20	58.666666666666664`	42
25	73.33333333333333`	56
30	88.`	73.5`
35	102.66666666666666`	91.5`
40	117.33333333333333`	116
45	132.`	142.5`
50	146.66666666666666`	173
55	161.33333333333331`	209.5`
60	176.`	248
65	190.66666666666666`	292.5`
70	205.33333333333331`	343
75	219.99999999999997`	401
80	234.66666666666666`	464

The data in the table show that for speeds greater than 20 mph, the stopping distance exceeds the 2-second separation distance.

Let's find an improved model to provide the basis for a better rule of thumb. To do this, we will look at the problem the other way around. We know the stopping distance for various speeds. If we force the stopping distance and the separation distance to be equal, then we can calculate separation time for each speed. We do this by taking the stopping distance (in feet) and dividing it by the travel speed (in ft/sec). In the following table, the first entry in each element is the traveling speed in miles per hour, and the second entry is the separation time in seconds that is required to ensure a separation distance equal to the stopping distance.

In[19]:=

```
data3 =
 Table[{data[[i, 1]], data[[i, 2]] / (data[[i,
   {i, 1, Length[data]}]
```

Out[19]=

```
{{20, 1.43182},
 {25, 1.52727}, {30, 1.67045},
 {35, 1.78247}, {40, 1.97727},
 {45, 2.15909}, {50, 2.35909},
 {55, 2.59711}, {60, 2.81818},
 {65, 3.06818}, {70, 3.34091},
 {75, 3.64545}, {80, 3.95455}}
```

In[20]:=

```
TableForm[data3,
 TableHeadings →
  {None, {"speed (mph)", "separation time (s
```

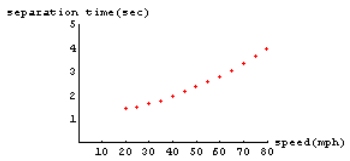
Out[20]//TableForm=

speed (mph)	separation time (sec)
20	1.431818181818182`
25	1.5272727272727273`
30	1.6704545454545454`
35	1.7824675324675328`
40	1.9772727272727275`
45	2.159090909090909`
50	2.3590909090909093`
55	2.5971074380165295`
60	2.8181818181818183`
65	3.0681818181818183`
70	3.3409090909090913`
75	3.645454545454546`
80	3.954545454545455`

Next, we plot the separation time versus speed.

In[21]:=

```
xlabel = "speed(mph)";

ylabel = "separation time(sec)";

p1 = ListPlot[data3, PlotRange → {{0, 80}, {0,
   PlotStyle → {PointSize[0.022], RGBColor[1,
   AxesLabel → {xlabel, ylabel},
   Ticks → {Table[0 + i * 10, {i, 0, 30}], Automat
```



It appears that the separation time is a linear function of the travel speed and not a constant function as the 2-second rule suggests. Let's find the best-fit linear function for the separation-time versus speed data and plot it together with the data points.

In[24]:=

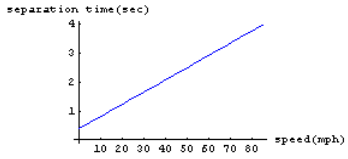
```
Clear[t];

t[v_] = Fit[data3, {1, v}, v]
```

Out[25]=

```
0.377941 + 0.0421825 v
```

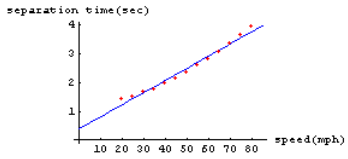
In[26]:=

```
p2 = Plot[t[v], {v, 0, 85}, PlotStyle → {RGBCol
  AxesLabel → {xlabel, ylabel},
  Ticks → {Table[0 + i * 10, {i, 0, 30}], Automat
```



In[27]:=

```
Show[p2, p1];
```



You Try It: Make a Better Rule of Thumb

Using the Model

Based upon the results of the analysis in Part I, formulate a better rule of thumb for separation time versus travel speed. Keep in mind that a rule of thumb should be easy to remember, easy to use, and, most importantly, it should be correct.

Part II: Stretching Springs and Effects of Radioactivity

Spring Elongation

Observing the Data

The response of a spring to various loads can be modeled in order to design a vehicle such as a tank, utility vehicle, or luxury car that responds to road conditions in a desired way. (See also "Bungee Cord Jumping: A Classroom Experiment," another module in this supplement.) We conducted an experiment to measure the stretch of a spring in inches as a function of the number of units of mass placed on the spring. The following list includes these data with the first element in each ordered pair being the mass on the spring and the second element its corresponding stretch.

In[28]:=

```
data = {{0, 0}, {1, 0.875}, {2, 1.721}, {3, 2.6
  {4, 3.531}, {5, 4.391}, {6, 5.241}, {7, 6.12
  {9, 7.869}, {10, 8.741}}
```

Out[28]=

```
{{0, 0}, {1, 0.875},
 {2, 1.721}, {3, 2.641}, {4, 3.531},
 {5, 4.391}, {6, 5.241}, {7, 6.12},
 {8, 6.992}, {9, 7.869}, {10, 8.741}}
```

A set of data values is usually entered in a list. However, to see the data better, we use **TableForm[ ]**, as follows.

In[29]:=

```
TableForm[data, TableDirections → {Row, Colum
 TableHeadings → {None, {"mass", "stretch(in)
```

Out[29]//TableForm=

mass	0	1	2	3	4	5	6	7	8	9	10

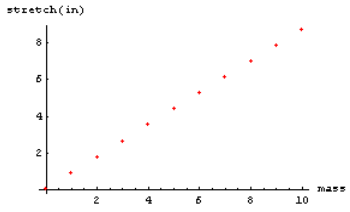
stretch(in) 0 0.875` 1.721` 2.641` 3.531` 4.391` 5.241` 6.12` 6.992` 7.869` 8.741`

"> About Mathematica

Next, we plot the data to see if there is a recognizable pattern and name the plot **p1** for later use.

In[30]:=

```
xlabel = "mass";

ylabel = "stretch(in)";

p1 = ListPlot[data,
  PlotStyle → {PointSize[0.017], RGBColor[1,
  AxesLabel → {xlabel, ylabel}];
```



▪ Designing a Model

The data strongly suggest a linear relationship. Now we use the **Fit[ ]** function to find the line of best-fit or the linear regression line.

"> About Mathematica

In[33]:=

```
Clear[y];

y[x_] = Fit[data, {x}, x]
```

Out[34]=

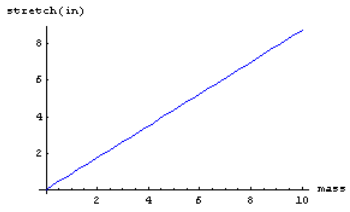
```
0.874732 x
```

"> About Mathematica

The constant of proportionality is 0.874732. Next, we plot the regression line, save it as **p2**, and then show the plots of the data and the regression line together on the same graph.

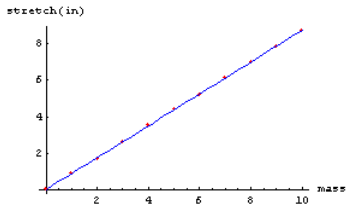
In[35]:=

```
p2 = Plot[y[x], {x, 0, 10}, PlotStyle → {RGBCol
  AxesLabel → {xlabel, ylabel}];
```



In[36]:=

```
Show[p1, p2];
```



- **Assessing the Errors**

As we expected, the line appears to fit the data very well. We can provide further verification by calculating the residual errors for each data point and plotting them. First, we need to use the model to calculate the values of stretch for each mass value in the original data set.

In[37]:=

```
predictedvalues = Table[{x, y[x]}, {x, 0, Leng
```

Out[37]=

```
{{0, 0}, {1, 0.874732}, {2, 1.74946},
 {3, 2.6242}, {4, 3.49893}, {5, 4.37366},
 {6, 5.24839}, {7, 6.12313}, {8, 6.99786},
 {9, 7.87259}, {10, 8.74732}, {11, 9.62206}}
```

```
"> About Mathematica
```

The residual error (or residual) is the difference between the measured value and the value the model predicts for each amount of mass.

In[38]:=

```
residuals = Table[{i - 1, data[[i, 2]] - predict
  {i, 1, Length[data]}]
```

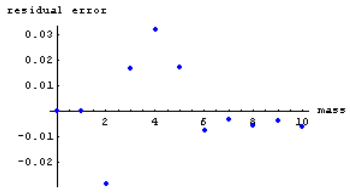
Out[38]=

```
{{0, 0}, {1, 0.000267532}, {2, -0.0284649},
 {3, 0.0168026}, {4, 0.0320701},
 {5, 0.0173377}, {6, -0.00739481},
 {7, -0.00312727}, {8, -0.00585974},
 {9, -0.00359221}, {10, -0.00632468}}
```

```
"> About Mathematica
```

In[39]:=

```
ListPlot[residuals,
 PlotStyle → {PointSize[0.02], RGBColor[0, 0,
 AxesLabel → {"mass", "residual error"}];
```



The residuals show that the differences are indeed small. Small is a relative term, and sometimes it is more meaningful to look at the error in relation to the size of the quantity being estimated. For all but the first data point, we calculate the relative error by dividing the measured value into the residual and multiplying by 100 to express this relative error as a percentage. Then, we plot the percentage error for each data point.

In[40]:=

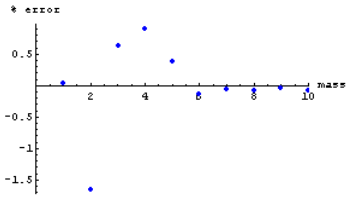
```
percenterrors =
 Table[
  {i - 1, (data[[i, 2]] - predictedvalues[[i, 2
    100}, {i, 2, 11}]
```

Out[40]=

```
{{1, 0.0305751}, {2, -1.65398},
 {3, 0.636221}, {4, 0.908245},
 {5, 0.394845}, {6, -0.141095},
 {7, -0.0510992}, {8, -0.0838064},
 {9, -0.0456501}, {10, -0.0723564}}
```

In[41]:=

```
ListPlot[percenterrors,
 PlotStyle → {PointSize[0.02], RGBColor[0, 0,
 PlotRange → {{0, 10}, All}, AxesLabel → {"mas
```



The maximum relative error for the linear model is -1.7%. Not bad!

Radioactivity

Observing the Data

A radioactive dye is injected into a patient's veins to facilitate an X-ray procedure. Measuring the radioactivity in counts per minute every minute for 10 minutes yielded the table of values shown below.

In[42]:=

```
data = {{0, 10023}, {1, 8174}, {2, 6693}, {3, 5
  {4, 4489}, {5, 3683}, {6, 3061}, {7, 2479}, {
  {9, 1645}, {10, 1326}};

TableForm[data, TableDirections → {Row, Colum
 TableHeadings → {None, {"Time(min)", "Radioa
```

Out[43]//TableForm=

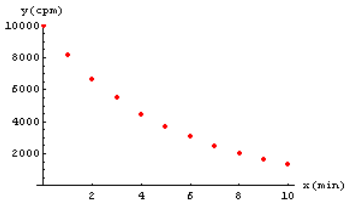
Time(min)	0	1	2	3	4	5	6	7	8	9	10
Radioactivity(cpm)	10023	8174	6693	5500	4489	3683	3061	2479	2045	1645	1326

```
"> About Mathematica
```

We begin by plotting the data points.

In[44]:=

```
xlabel = "x(min)";

ylabel = "y(cpm)";
p1 = ListPlot[data, PlotRange → {0, 10050},
  PlotStyle -> {PointSize[0.02], RGBColor[0.9
  AxesLabel → {xlabel, ylabel}];
```



Designing a Model

There appears to be a trend that we can capture with a mathematical model, and now we try to find a suitable model using the **Fit[ ]** function. What does the

function look like to you? A decaying exponential? We try to find a function of the form $y=ae^{-kx}$, where we vary the value of k and the computer calculates the value of a for the best-fit function of this form.

In[46]:=

```
k = 0.1;

Clear[y];

y[x_] = Fit[data, {Exp[-k * x]}, x]
```

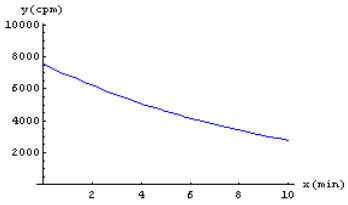
Out[48]=

$$7584.05\, e^{-0.1 x}$$

Next, we plot our model function, save it as **p2**, and then show the plots of the data and the model function together on the same graph.

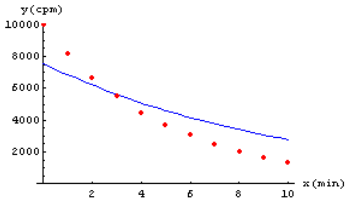
In[49]:=

```
p2 = Plot[y[x], {x, 0, 10}, PlotRange → {0, 100
  PlotStyle -> {RGBColor[0, 0, 1]},
  AxesLabel → {xlabel, ylabel}];
```



In[50]:=

```
Show[p2, p1];
(* Putting p2 first puts the data points on
  the model graph instead of under it.*)
```



"> About *Mathematica*

- **Assessing the Errors**

Our model doesn't appear to be very good! Let's quantify just how good (or bad) it is. You guessed right - we should calculate the residual errors to do this quantification.

In[51]:=

```
residuals = Table[{data[[i, 1]], data[[i, 2]] -
  {i, 1, Length[data]}]
```

Out[51]=

```
{{0, 2438.95}, {1, 1311.66}, {2, 483.701},
 {3, -118.406}, {4, -594.744}, {5, -916.962},
 {6, -1101.22}, {7, -1287.13}, {8, -1362.74},
 {9, -1438.45}, {10, -1464.02}}
```

To get an overall measure of the error for all of the data points in the set, you might be inclined to calculate the average (mean) of the residuals, but this can be very misleading.

In[52]:=

```
residavg = Sum[residuals[[i, 2]], {i, 1, Lengt
  Length[data]
```

Out[52]=

```
-368.122
```

As you can see, the mean residual error is small relative to the average size of the measured values of radioactivity in the data set. On this basis, we might be led to believe that our model isn't so bad after all. Wrong! The problem is that the individual or local errors are actually sizable when compared to the measured radioactivity counts, but because some of them are positive and some are negative, they tend to cancel each other when we add them together to calculate their mean value. Look at the values in the list of residual errors or look at the vertical differences between the data points and the fit function on the graph, and it is easy to see that the average of the residual errors is misleading.

There is an infinite number of ways to address this canceling problem, but one of the most common is to calculate the sum of the squares of the residuals and try to find the smallest or least value of this sum of squared residuals, hence we have the term "least squares." Squaring the residuals removes the canceling effect that occurs when we add them together for a measure of the global error. The *Mathematica* **Fit[ ]** function uses least squares, and some variations of it, to find a best-fit function for a set of data. Finding the minimum value for the sum of squared residuals is a problem that can be solved using calculus, and you will study this problem later on, but for now you can do it by trial and error. Let's get back to our radioactivity problem. What we do now is to calculate the sum of the squares of the residuals for our data set.

In[53]:=

```
residsquaresum = Sum[residuals[[i, 2]]^2, {i,
```

Out[53]=

$$1.80503 \times 10^{7}$$

Next, we calculate the mean of the squared residuals.

In[54]:=

```
msresiduals = residsquaresum / Length[data]
```

Out[54]=

$$1.64094 \times 10^{6}$$

Comparing the average of the squared residuals with the average of the radioactivity counts in the data set would be like comparing apples with oranges because **msresiduals** is an average of squares, whereas the average radioactivity count is not an average of squared values. To make a fair comparison, we take the square root of the mean of the squares.

In[55]:=

```
rmsresiduals = Sqrt[msresiduals]
```

Out[55]=

```
1280.99
```

The value that we calculate in the preceding step is the root of the mean of the squares of the residuals and is oftentimes called the root-mean-square or "rms" value of the residuals.

You Try It: Improving the Model and Making Predictions

▪ Improving the Model

In Part II, we calculated the "rms" value of the residual errors, a number that we can use for a fair comparison with the average of the measured radioactivity values. As we initially expected, this comparison leads us to conclude that our model needs improvement. We will leave that up to you, but, to help out, we group all of the commands that you need for the error analysis into one cell, the one that follows. Use trial and error to find a better model function by changing the value of **k** (in red) to reduce (possibly minimize) the sum of the squared residuals (which will also minimize the "rms" value).

A c*omment about errors in modeling*: Keep in mind that in mathematical modeling you are not able to completely eliminate errors. This is because of random errors that occur in measurements due to the limited precision of all measuring instruments, and because of systematic errors (i.e., errors that follow a pattern) due to shortcomings of the model and/or possible defects in the measuring tools. Systematic errors can often be reduced or eliminated by refining the model and repairing or recalibrating the measuring equipment, but random errors are unavoidable. We can reduce the magnitudes of random errors by using more precise instruments, but we cannot eliminate them. The errors may be smaller, but they are always present.

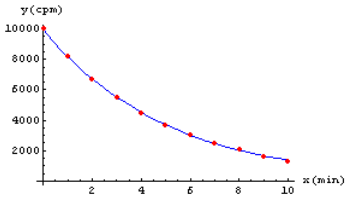
In[56]:=

```
k = 0.2;

Clear[y];
```

```
y[x_] = Fit[data, {Exp[-k * x]}, x]

p2 = Plot[y[x], {x, 0, 10}, PlotStyle -> {RGBCo
  AxesLabel → {xlabel, ylabel}, DisplayFuncti

Show[p2, p1, DisplayFunction → $DisplayFuncti

residuals = Table[{data[[i, 1]], data[[i, 2]] -
  {i, 1, Length[data]}];

residsquaresum = Sum[residuals[[i, 2]]^2,
  {i, 1, Length[data]}];

msresiduals = residsquaresum / Length[data];

rmsresiduals = Sqrt[msresiduals]
```

Out[58]=

$$10009.9\, e^{-0.2\, x}$$



Out[64]=

```
20.8035
```

"> About *Mathematica*

▪ Making Predictions

Now use your improved model to determine when the radioactivity will fall below 500 counts per minute.

In[65]:=

```
Solve[y[x] == 500, x]
```

```
Solve::ifun :
 Inverse functions are being used by Solve, so
   some solutions may not be found; use Reduce
   for complete solution information. More…
```

Out[65]=

```
{{x → 14.9836}}
```

Your improved model should predict that the radioactivity will be below 500 cpm after about 15 minutes. Is that what you get?

"> About *Mathematica*

Part III: Growth of Ponderosa Pines and Black Bass

▪ Ponderosa Pines

▪ Observing the Data

In the table that follows, the girth of a pine tree (the distance around the tree at shoulder height) is measured in inches, and the volume of usable lumber obtained from the tree is measured in board feet (bf). We will formulate and test the following models: that usable board feet is proportional to (a) the square of the girth and (b) the cube of the girth. Which is better? Does one model provide a better explanation than the other?

In[66]:=

```
data = {{17, 19}, {19, 25}, {20, 32}, {23, 57},
  {28, 113}, {32, 123}, {38, 252}, {39, 259}, {

TableForm[data, TableDirections → {Row, Colur
 TableHeadings → {None, {"girth (in)", "lumbe
```

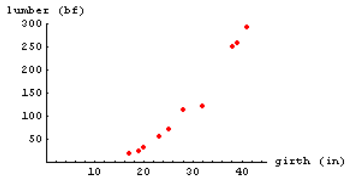
Out[67]//TableForm=

girth (in)	17	19	20	23	25	28	32	38	39	41
lumber (bf)	19	25	32	57	71	113	123	252	259	294

Next, we plot the data to see if there is a recognizable pattern.

In[68]:=

```
xlabel = "girth (in)";

ylabel = "lumber (bf)";

p1 = ListPlot[data, PlotRange → {{0, 45}, {0, 3
  PlotStyle → {PointSize[0.023], RGBColor[1,
  AxesLabel → {xlabel, ylabel}];
```



▪ Designing a Model

We are asked to compare a quadratic and a cubic model, so now we use the **Fit[ ]** command to find the best-fit function for each model.

In[71]:=

```
Clear[y2];

y2[x_] = Fit[data, {x^2}, x]
```

Out[72]=

$$0.157919\,x^2$$

In[73]:=

```
Clear[y3];

y3[x_] = Fit[data, {x^3}, x]
```

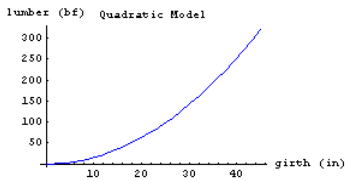
Out[74]=

$$0.00436203\,x^3$$

We plot the two models and superimpose their graphs on the graph of the data.

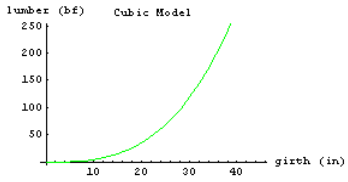
In[75]:=

```
p2 = Plot[y2[x], {x, 0, 45}, PlotStyle → {RGBCo
  AxesLabel → {xlabel, ylabel}, PlotLabel → "Q
```



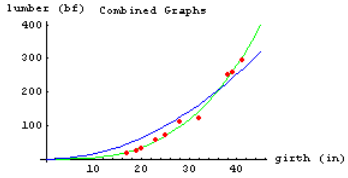
In[76]:=

```
p3 = Plot[y3[x], {x, 0, 45}, PlotStyle → {RGBCo
  AxesLabel → {xlabel, ylabel}, PlotLabel → "C
```



In[77]:=

```
Show[p3, p2, p1, PlotLabel → "Combined Graphs
(*Putting p3 first puts the data points on
  the model graphs instead of under them.*)
```



The graphs seem to show that the cubic model better fits the data.

▪ Assessing the Errors

Now we analyze the fit for each of the models by calculating the percent errors, first for the quadratic function and then for the cubic.

Analysis of the Errors for the Quadratic-Fit Function

Let's use the model to calculate the volume of lumber for each girth measurement in the original data set, and then calculate the residual errors and plot them.

In[78]:=

```
quadpredictedvalues =
 Table[{data[[i, 1]], y2[data[[i, 1]]]}, {i, 1
```

Out[78]=

```
{{17, 45.6385}, {19, 57.0086},
 {20, 63.1675}, {23, 83.539},
 {25, 98.6992}, {28, 123.808},
 {32, 161.709}, {38, 228.035},
 {39, 240.194}, {41, 265.461}}
```

In[79]:=

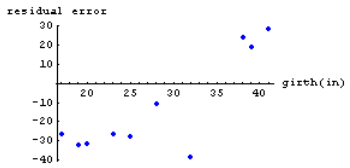
```
quadresiduals =
 Table[{data[[i, 1]],
   data[[i, 2]] - quadpredictedvalues[[i, 2]]}
```

Out[79]=

```
{{17, -26.6385}, {19, -32.0086},
 {20, -31.1675}, {23, -26.539},
 {25, -27.6992}, {28, -10.8082},
 {32, -38.7087}, {38, 23.9654},
 {39, 18.8057}, {41, 28.5387}}
```

In[80]:=

```
ListPlot[quadresiduals,
 PlotStyle → {PointSize[0.023], RGBColor[0, 0
 AxesLabel → {"girth(in)", "residual error"},
 AxesOrigin → {16.5, 0}];
```



Note that the errors aren't completely random. There are more negative errors than there are positive ones, and the error seems to increase as the girth increases. These observations suggest that the quadratic function is not appropriate for modeling the relation between the volume of usable lumber in a tree and its girth.

It is usually more helpful to look at the error in relation to the size of the quantity being estimated. We calculate the relative percent errors as follows.

In[81]:=

```
quadpercenterrors =
 Table[{data[[i, 1]],
   (data[[i, 2]] - quadpredictedvalues[[i, 2]]
    100}, {i, 1, 10}]
```

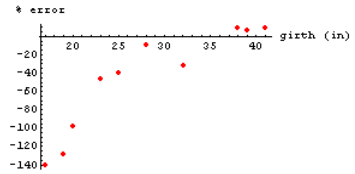
Out[81]=

```
{{17, -140.203}, {19, -128.035},
 {20, -97.3983}, {23, -46.5596},
 {25, -39.0129}, {28, -9.56481},
 {32, -31.4705}, {38, 9.51009},
 {39, 7.26089}, {41, 9.70704}}
```

Now we plot the percent errors and save the graph for a later comparison with the percent errors for the cubic function.

In[82]:=

```
p3 = ListPlot[quadpercenterrors,
  PlotStyle → {PointSize[0.023], RGBColor[1,
  AxesLabel → {"girth (in)", "% error"},
  AxesOrigin → {16.5, 0}];
```



The percent error of largest magnitude is about -140%, and there is a trend in the errors, indicating that they are not random.

Let's look at the errors for the cubic model.

Analysis of the Errors for the Cubic-Fit Function

We do the same as we did for the quadratic function.

In[83]:=

```
cubicpredictedvalues =
 Table[{data[[i, 1]], y3[data[[i, 1]]]}, {i, 1
```

Out[83]=

```
{{17, 21.4307}, {19, 29.9192},
 {20, 34.8963}, {23, 53.0729},
 {25, 68.1568}, {28, 95.7554},
 {32, 142.935}, {38, 239.354},
 {39, 258.752}, {41, 300.636}}
```

In[84]:=

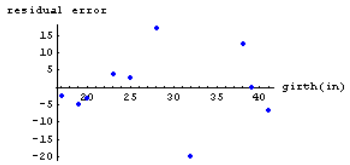
```
cubicresiduals =
 Table[{data[[i, 1]],
   data[[i, 2]] - cubicpredictedvalues[[i, 2]]
```

Out[84]=

```
{{17, -2.43068}, {19, -4.9192},
 {20, -2.89628}, {23, 3.92712},
 {25, 2.84321}, {28, 17.2446},
 {32, -19.9352}, {38, 12.6464},
 {39, 0.248462}, {41, -6.6358}}
```

In[85]:=

```
ListPlot[cubicresiduals,
 PlotStyle → {PointSize[0.023], RGBColor[0, 0
 AxesLabel → {"girth(in)", "residual error"},
 AxesOrigin → {16.5, 0}];
```



Note that the residuals exhibit a more random pattern than they did in the quadratic model. There are as many negative errors as positive ones, and there is no apparent trend in the errors.

In[86]:=

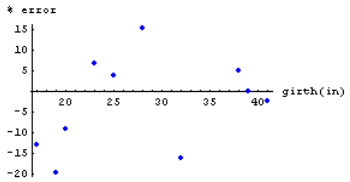
```
cubicpercenterrors =
 Table[{data[[i, 1]],
   (data[[i, 2]] - cubicpredictedvalues[[i, 2]
    100}, {i, 1, 10}]
```

Out[86]=

```
{{17, -12.793}, {19, -19.6768},
 {20, -9.05087}, {23, 6.88969},
 {25, 4.00452}, {28, 15.2607},
 {32, -16.2074}, {38, 5.01842},
 {39, 0.0959314}, {41, -2.25707}}
```

In[87]:=

```
p4 = ListPlot[cubicpercenterrors,
  PlotStyle → {PointSize[0.023], RGBColor[0,
  AxesLabel → {"girth(in)", "% error"}, AxesO
```



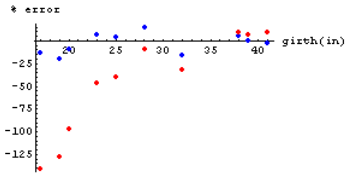
In this case, the percent error of largest magnitude is about -20%, which is much better than for the quadratic regression function.

Compare the Errors

To confirm our earlier assessment that the cubic regression is evidently better, we now plot the percent errors for the two models together.

In[88]:=

```
Show[p4, p3];
```



- **Explaining the Model**

The unit of board feet is a measure of volume. If a tree is modeled as a right circular cone, its volume is approximated by $V = \frac{1}{3}\pi r^2 h$, where V is the volume, h is the height of the tree, and r is its radius. The girth, g, is the circumference of the tree near the base so that $g = 2\pi r$ or $r = \frac{g}{2\pi}$. If we assume that as a tree grows the proportion $\frac{h}{r} = k$ is a constant, then we have that $V = \frac{\pi}{3}\left(\frac{g}{2\pi}\right)^2\left(\frac{kg}{2\pi}\right) = \left(\frac{k}{24\pi^2}\right)g^3$, where the last quantity in parentheses is a constant. This shows that cubic relation between the volume of lumber produced and the girth of the tree near its base is a rational model.

Black Bass

- **Observing the Data**

In the table that follows, L represents the lengths of New York black bass measured in inches, and w represents the weight of the fish. Formulate and test a model that assumes the weight of the fish is proportional to the cube of its length. Can you provide a rational explanation of the model?

In[89]:=

```
data = {{12.5, 17}, {12.63, 16}, {14.13, 17}, {
  {14.5, 26}, {14.5, 27}, {17.25, 41}, {17.75,

TableForm[data, TableDirections → {Row, Colum
 TableHeadings → {{}, {"L(in)", "w(oz)"}}]
```

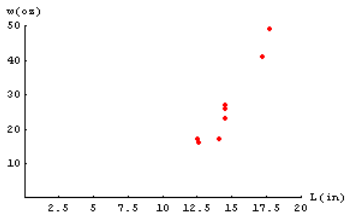
Out[90]//TableForm=

L(in)	12.5`	12.63`	14.13`	14.5`	14.5`	14.5`	17.25`	17.75`
w(oz)	17	16	17	23	26	27	41	49

Next, we plot the data to see if there is a recognizable pattern.

In[91]:=

```
xlabel = "L(in)";

ylabel = "w(oz)";

p1 = ListPlot[data, PlotRange → {{0, 20}, {0, 5
  PlotStyle → {PointSize[0.020], RGBColor[1,
  AxesLabel → {xlabel, ylabel}];
```



- **Designing a Model**

We are asked to construct a cubic model, so now we find the best-fit function using the **Fit[ ]** command.

In[94]:=

```
Clear[y];

y[x_] = Fit[data, {x^3}, x]
```

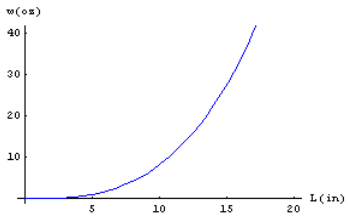
Out[95]=

$$0.00818628\,x^3$$

Next, we plot the model function and superimpose its graph on the graph of the data.

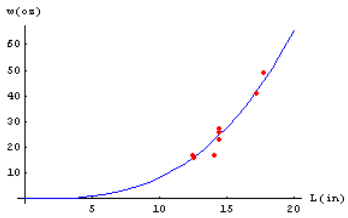
In[96]:=

```
p2 = Plot[y[x], {x, 0, 20}, PlotStyle → {RGBCol
  AxesLabel → {xlabel, ylabel}];
```



In[97]:=

```
Show[p2, p1];
(*Putting p2 first puts the data points on
  the model graphs instead of under them.*)
```



The graphs show that the cubic model fits the data well.

▪ Assessing the Errors

Now we analyze the fit by calculating the percent errors. First, we use the model to calculate the weight of the fish for each length measurement in the original data set, and then we calculate the residuals and plot them.

In[98]:=

```
predictedvalues = Table[{data[[i, 1]], y[data
  {i, 1, Length[data]}]
```

Out[98]=

```
{{12.5, 15.9888}, {12.63, 16.4929},
 {14.13, 23.0947}, {14.5, 24.9569},
 {14.5, 24.9569}, {14.5, 24.9569},
 {17.25, 42.0198}, {17.75, 45.7806}}
```

In[99]:=

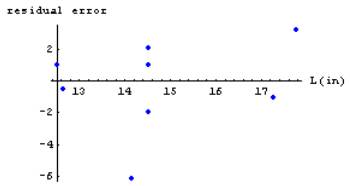
```
residuals =
 Table[{data[[i, 1]], data[[i, 2]] - predicted
  {i, 1, Length[data]}]
```

Out[99]=

```
{{12.5, 1.01117}, {12.63, -0.492885},
 {14.13, -6.09474}, {14.5, -1.9569},
 {14.5, 1.0431}, {14.5, 2.0431},
 {17.25, -1.01979}, {17.75, 3.21938}}
```

In[100]:=

```
ListPlot[residuals,
 PlotStyle → {PointSize[0.020], RGBColor[0, 0
 AxesLabel → {"L(in)", "residual error"},
 AxesOrigin → {12.5, 0}];
```



Note that there are nearly equal numbers of positive and negative errors, and there is no apparent trend in the errors. They appear to be random.

As before, it is more meaningful to look at the error in relation to the size of the quantity being estimated, so we calculate the relative percent errors and plot them.

In[101]:=

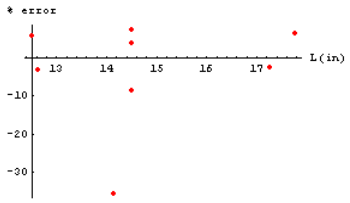
```
percenterrors =
 Table[{data[[i, 1]],
   (data[[i, 2]] - predictedvalues[[i, 2]]) / d
  {i, 1, Length[data]}]
```

Out[101]=

```
{{12.5, 5.94807}, {12.63, -3.08053},
 {14.13, -35.8514}, {14.5, -8.50825},
 {14.5, 4.01194}, {14.5, 7.56705},
 {17.25, -2.48729}, {17.75, 6.57017}}
```

In[102]:=

```
p3 = ListPlot[percenterrors,
  PlotStyle → {PointSize[0.020], RGBColor[1,
  AxesLabel → {"L(in)", "% error"}, AxesOrigi
```



Improving the Model

The largest percent error about -36%, but this data point seems to be an outlier. Let's see what happens if we remove this point from the data set.

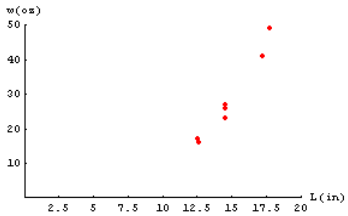
In[103]:=

```
data = {{12.5, 17}, {12.63, 16}, {14.13, 17}, {
  {14.5, 26}, {14.5, 27}, {17.25, 41}, {17.75,

data = Delete[data, 3]

p1 = ListPlot[data, PlotRange → {{0, 20}, {0, 5
  PlotStyle → {PointSize[0.020], RGBColor[1,
  AxesLabel → {xlabel, ylabel}];

Clear[y];

y[x_] = Fit[data, {x^3}, x]

p2 = Plot[y[x], {x, 0, 20}, PlotStyle → {RGBCol
  AxesLabel → {xlabel, ylabel}];

Show[p2, p1];

predictedvalues = Table[{data[[i, 1]], y[data
  {i, 1, Length[data]}];
```

```
residuals =
 Table[{data[[i, 1]], data[[i, 2]] - predicted
  {i, 1, Length[data]}];

percenterrors =
 Table[{data[[i, 1]],
   (data[[i, 2]] - predictedvalues[[i, 2]]) / d
  {i, 1, Length[data]}];

TableForm[percenterrors, TableDirections → {
 TableHeadings → {None, {"L(in)", "% error"}}

p3 = ListPlot[percenterrors,
  PlotStyle → {PointSize[0.02], RGBColor[1, 0
  AxesOrigin → {12.5, 0}, AxesLabel → {"L(in)"
```

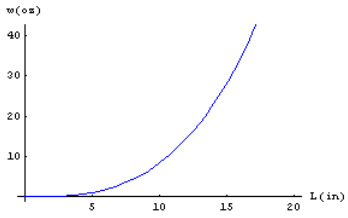
Out[104]=

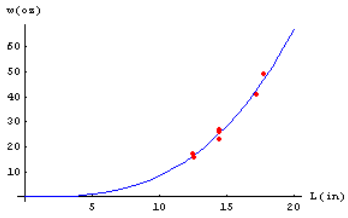
```
{{12.5, 17}, {12.63, 16},
 {14.5, 23}, {14.5, 26}, {14.5, 27},
 {17.25, 41}, {17.75, 49}}
```



Out[107]=

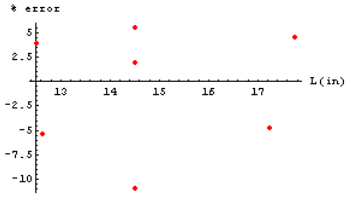
$$0.00837042\,x^3$$





Out[113]//TableForm=

L (in)	12.5`	12.63`	14.5`	14.5`	14.5`	17.25`	17.75`
% error	3.8325433198984835`	-5.399143540185158`	-10.948946689410594`	1.8528548516752446`	5.4879343016131985`	-4.792561063590046`	4.468630205201



After deleting the one data point, we find that the new model gives a largest percent error of about -11%, which is considerably better.

- **Explaining the Model**

The weight of a fish is proportional to its volume. If we think of a fish to be shaped roughly like an ellipsoid, then its volume would be approximated by $V=\frac{4}{3}\pi lwd$ where L, w, and d are its length, width, and depth, respectively. If, as the fish grows, we assume that its proportions remain the same then $w=k_w L$ and $d=k_d L$, where k_w and k_d are constants. Therefore, $V=\frac{4}{3}\pi L(k_w L)(k_d L)=(\frac{4 k_w k_d \pi}{3})L^3$ where the last quantity in parentheses is a constant. Therefore, the weight is proportional to the volume, which is proportional to the length cubed, and our model has a rational basis.

You Try It: Modeling Takes Practice

Data Sets

Drug Levels

In[115]:=

```
drugdata = {{0, 853}, {1, 587}, {2, 390}, {3, 2
  {5, 130}, {6, 97}, {7, 67}, {8, 50}, {9, 40},
```

Out[115]=

```
{{0, 853}, {1, 587}, {2, 390},
 {3, 274}, {4, 189}, {5, 130}, {6, 97},
 {7, 67}, {8, 50}, {9, 40}, {10, 31}}
```

Cell Count

In[116]:=

```
celldata = {{0, 597}, {2, 893}, {4, 1339}, {6,
  {8, 2976}, {10, 4433}, {12, 6612}, {14, 9865
  {18, 21956}, {20, 32763}}
```

Out[116]=

```
{{0, 597}, {2, 893}, {4, 1339},
 {6, 1995}, {8, 2976}, {10, 4433},
 {12, 6612}, {14, 9865}, {16, 14719},
 {18, 21956}, {20, 32763}}
```

Vacuum Pump

In[117]:=

```
vacuumdata = {{0, 100000}, {1, 36788}, {2, 1353
  {4, 1837}, {5, 671}}
```

Out[117]=

```
{{0, 100000}, {1, 36788}, {2, 13537},
 {3, 4986}, {4, 1837}, {5, 671}}
```

Doctoral Degrees

In[118]:=

```
doctoraldata = {{6, 520}, {10, 460}, {14, 680},
  {20, 730}, {21, 810}, {22, 830}}
```

Out[118]=

```
{{6, 520}, {10, 460}, {14, 680}, {18, 630},
 {20, 730}, {21, 810}, {22, 830}}
```

Instructions

Mathematical modeling and a computer algebra system like *Mathematica* help us describe real-world phenomena, understand the mechanisms behind the behavior, and make predictions. The procedure for modeling generally includes the following steps:

1. Plot and observe the data, looking for relationships and patterns.
2. Formulate a function to model the data.

3. Assess your model by analyzing the residual errors and/or relative errors.
4. Improve your model, if possible.
5. Use your model to gain a better understanding of the phenomenon you are modeling.
6. Use your model to make predictions.

With some practice, you too can become an effective mathematical modeler. There are plenty of modeling exercises that you can do yourself. Select a problem and obtain some data (possibly by designing your own experiment) and use the modeling procedures outlined in the preceding Parts of this module as a template for mathematical modeling. To help you get started, we include the data sets for four modeling examples.

Part IV: Heart Rates of Mammals

Observing the Data

The following data relate the weight in grams (g) of some mammals to their heart rate in beats per minute. Plot the data. Is there a trend? If so, find a function that captures the trend of the data. Hint: try the form $y = x^{-1/n}$ for n an integer.

In[119]:=

```
data = {{4, 660}, {25, 670}, {200, 420}, {300, 
  {2000, 205}, {5000, 120}, {30000, 85}, {5000
  {70000, 72}, {450000, 38}, {500000, 40}, {30

TableForm[data, TableDirections → {Row, Colum
 TableHeadings → {None, {"x(g)", "y(bpm)"}}]
```

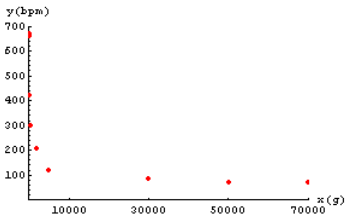
Out[120]//TableForm=

x(g)	4	25	200	300	2000	5000	30000	50000	70000	450000	500000	3000000
y(bpm)	660	670	420	300	205	120	85	70	72	38	40	48

Next, we plot the data to see if there is a recognizable pattern.

In[121]:=

```
xlabel = "x(g)";

ylabel = "y(bpm)";

p1 = ListPlot[data, PlotRange → {{0, 70000}, {
  PlotStyle → {PointSize[0.020], RGBColor[1,
  AxesLabel → {xlabel, ylabel},
  Ticks → {{10000, 30000, 50000, 70000}, Autom
```



Note that we did not plot the data points for the horse, the ox, and the elephant because that makes the scale too large to see if there is a pattern for the smaller mammals; however, these data points are included in the calculations that follow.

Designing a Model

The data suggest that there is a relationship. As suggested in the hint above, we now build a group of functions of the form $y = x^{-1/n}$ where n is an integer. For integer values of n (1, 2, 3, 4, 5), we use the **Fit[ ]** function to find the function of the form $x^{-1/n}$ that best fits the data. (The next command does not display its results. We will look at the fit functions in the cells that follow it.)

In[124]:=

```
Clear[y];

Do[y[x_, n] = Fit[data, {x^(-1/n)}, x], {n, 1

"> About Mathematica
```

Now we look at each function by typing the symbol name representing each one in turn, and then we graph them.

In[126]:=

```
y[x, 1]
```

Out[126]=

$$\frac{3040.89}{x}$$

In[127]:=

```
y[x, 2]
```

Out[127]=

$$\frac{1733.56}{\sqrt{x}}$$

In[128]:=

```
y[x, 3]
```

Out[128]=

$$\frac{1371.61}{x^{1/3}}$$

In[129]:=

```
y[x, 4]
```

Out[129]=

$$\frac{1149.87}{x^{1/4}}$$

In[130]:=

```
y[x, 5]
```

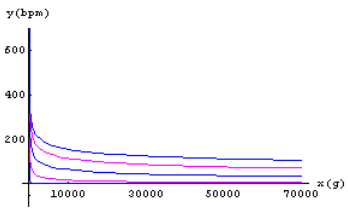
Out[130]=

$$\frac{989.458}{x^{1/5}}$$

```
"> About Mathematica
```

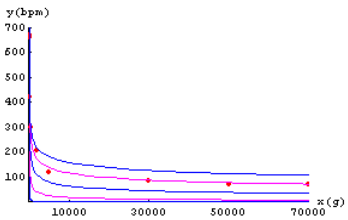
In[131]:=

```
p2 = Plot[{y[x, 1], y[x, 2], y[x, 3], y[x, 4], y
  {x, 0, 70000}, PlotRange → {-100, 700},
  PlotStyle → {{RGBColor[0, 0, 1]}, {RGBColor
    {RGBColor[0, 0, 1]}, {RGBColor[1, 0, 1]},
    {RGBColor[0, 0, 1]}, {RGBColor[1, 0, 1]}},
  AxesLabel → {xlabel, ylabel},
  Ticks → {{10000, 30000, 50000, 70000}, Autom
```



In[132]:=

```
Show[p1, p2];
```



- **Assessing the Errors**

It appears that **y[x_, 4]** provides the best fit. Now we will analyze the errors. First, we calculate the heart rate values that our selected model predicts for each mammal, and then we calculate the residual errors and plot them.

In[133]:=

```
predictedvalues = Table[{data[[i, 1]], y[data
  {i, 1, Length[data]}]
```

Out[133]=

```
{{4, 813.08}, {25, 514.237},
 {200, 305.767}, {300, 276.292},
 {2000, 171.945}, {5000, 136.743},
 {30000, 87.3711}, {50000, 76.8963},
 {70000, 70.6925}, {450000, 44.3961},
 {500000, 43.242}, {3000000, 27.6292}}
```

In[134]:=

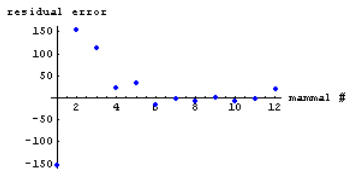
```
residuals = Table[{i, data[[i, 2]] - predicted
  {i, 1, Length[data]}]
```

Out[134]=

```
{{1, -153.08}, {2, 155.763},
 {3, 114.233}, {4, 23.7083}, {5, 33.0546},
 {6, -16.7432}, {7, -2.3711}, {8, -6.89633},
 {9, 1.30746}, {10, -6.39612},
 {11, -3.24198}, {12, 20.3708}}
```

In[135]:=

```
ListPlot[residuals,
 PlotStyle → {PointSize[0.023], RGBColor[0, 0
 PlotRange → All, AxesOrigin → {1, 0},
 AxesLabel → {"mammal #", "residual error"}];
```



As before, it is helpful to look at the error in relation to the size of the quantity being estimated. We calculate the relative percent errors.

In[136]:=

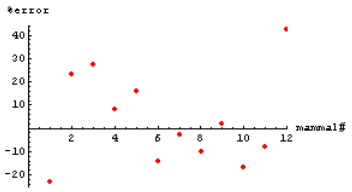
```
percenterrors =
 Table[
  {i, (data[[i, 2]] - predictedvalues[[i, 2]])
    100}, {i, 1, Length[data]}]
```

Out[136]=

```
{{1, -23.1939}, {2, 23.2482},
 {3, 27.1983}, {4, 7.90278}, {5, 16.1242},
 {6, -13.9526}, {7, -2.78953}, {8, -9.8519},
 {9, 1.81591}, {10, -16.8319},
 {11, -8.10496}, {12, 42.4392}}
```

In[137]:=

```
ListPlot[percenterrors,
 PlotStyle → {PointSize[0.023], RGBColor[1, 0
 PlotRange → {{0, 12}, All}, AxesLabel → {"mam
```



The errors appear to be somewhat random, but there is possibly a trend in the errors, with two outliers. Can you see the trend and the outliers? The largest relative percentage errors are for the largest and the smallest animals (i.e., the bat and the elephant) with magnitudes of 23% and 42%, respectively. The model appears to capture a trend in the data, which could be useful in understanding the relationship between mammal size and heart rate; however, it probably would not be useful as a predictive tool since the magnitudes of the individual errors are so large.

You Try It: A Rational Explanation for Heart Rates

Try to provide a rational explanation for the heart rate model that we found in Part IV.

About *Mathematica*

While **TableForm[ ]** presents the data in a more readable form, it is important to know that data sets in this form cannot be used in calculations. That is why we saved the data set as "data" in the first command, and then used **TableForm[ ]** to review my entries for accuracy (and possibly include in a formal report). To learn more about **TableForm[ ]** and lists in *Mathematica*, pull down the Help menu, select the Help Browser, and type **TableForm** or **List**. Go Back.

The **Fit[ ]** command is very useful for mathematical modeling. It allows you to find regression functions of a variety of different forms. Here we use it to find the best-fit function of the form $y=ax$, where a is a constant. You can learn more about the **Fit[ ]** function by going to the Help Browser and typing **Fit**. Go Back.

Whenever we define a new function, like **y[x_]** in the preceding command, we usually clear all previous definitions of **y** with the **Clear[ ]** command. Sometimes you want *Mathematica* to remember previous definitions of a function, but many times you do not. Managing symbol definitions in *Mathematica* is very important, and mismanagement in this regard can lead to some very misleading and confusing results. You can learn more about defining symbols in *Mathematica* by referring to the "Overview of *Mathematica*: Assignment Commands," included in this supplement, and you can go to the Help Browser and type **Clear**. Go Back.

The **Table[ ]** command is used to create a list when a formula can be used to generate each of the elements in the list. In the preceding command, the elements of the list are **{x, y[x]}**. The formula for the first element in each ordered pair is simply **x**, and the formula for each second element is **y[x_]=0.874732x**. The second entry in the **Table[ ]** command (i.e., **{x, 0, Length[data]}**) specifies the set of values for **x**. In the example above, **x** goes from **0** to **Length[data]** in increments of 1. To learn more about the **Table[ ]** command, go to the Help Browser and type **Table**. Go Back.

In the preceding command, we use the double square brackets, e.g., **[[i, 2]]**, to extract elements or parts from the list called data. In this example, the first entry inside the double square brackets is **i**, which specifies which ordered pair we want from the list, and the second entry is **2**, which specifies which of the two elements to extract from the i^{th} ordered pair. The first element of a list is designated by the number 1, not 0, which is why we let the index **i** vary from **1** to **Length[data]** and calculated **x** using the formula **i-1**. To learn more about the use of double square brackets, go to the Help Browser and type **Part**. Go Back.

The semicolon after a *Mathematica* command suppresses the output. In the preceding command cell, we grouped the data list command and the **TableForm[ ]** command together in one cell with a semicolon after the definition of the data list and no semicolon after **TableForm[ ]**. This way the data table is displayed for easy reading and the data list is suppressed. If you remove the semicolon, both forms of the data are displayed. Go Back.

You can include comments in a *Mathematica* command cell by enclosing the comment inside **(* *)** as in the preceding command. This can be useful for documenting your work, which is good programming practice. When *Mathematica* executes a command, it ignores everything enclosed by the **(* *)** markers. Go Back.

Among other things, *Mathematica* is a programming language. When we group a series of commands together in a single cell, like the ones in the preceding cell, we are essentially writing a *Mathematica* program. You can give this group of commands a symbol name and use it like any other *Mathematica* command. You do this with the **Block[ ]** command or the **Module[ ]** command. To learn more about programming in *Mathematica*, see "Overview of *Mathematica*: Making Your Own Commands," included in this supplement. Go Back.

The error message that displays when you execute the **Solve[ ]** command is to warn you that whenever *Mathematica* uses inverse functions to solve an equation there may be some solutions that are missed. A good example of this is the equation sin x=1. *Mathematica* will use the inverse sine function to solve this equation, giving $x = \frac{\pi}{2}$ (with the same warning message). We all know, however, that the equation actually has infinitely many solutions. They are $x = (4n+1)\frac{\pi}{2}$, where n can be any integer. You might try using *Mathematica* to solve sin x=1. Go Back.

The **Do[ ]** command is useful for performing an operation for a list of values that are incremented by a constant amount. In the example above, the operation is to build the functions **y[x_, n]** as **n** varies from 1 to 5 in increments of 1 (the default increment when none is specified). To learn more about the **Do[ ]** command, and the related **For[ ]** and **While[ ]** commands, go to the Help Browser and type **Do** and/or **For** and/or **While**. Go Back.

To define the functions **y[x_, n]** above, we used double indices on the symbol **y**. This way we can use one symbol, i.e., **y**, to represent a group of related functions.
Go Back.