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

Note: You may notice differences between this Maple worksheet and the equivalent Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of major functional differences between Maple and Mathematica.

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 analyze and improve them, and how to use them to study the behavior you are modeling, to make predictions. As you will see, a computer algebra system like *Maple* is a powerful tool that will aid you in your mathematical modeling.

In this worksheet, we use the two *Maple* packages **plots** and **stats**. We initialize them at the beginning of Part I.

Technology Guidelines

NOTE: If you have just finished a worksheet, **restart** *Maple* before executing a new worksheet.
TO OPEN SECTIONS,
Click on the **PLUS** sign at the left hand side of the screen *or* select **Expand All Sections** from the **View** drop down menu.

TO STOP AN EXECUTION
Click on **STOP** button from the toolbar.

ORDER OF EXECUTION
Execute commands in the order given. Do not skip any *Maple* Input lines within a given worksheet

Alternatively, you can execute the entire worksheet by selecting the **Execute Worksheet** command from the **Edit** drop down menu.

SAVING WORKSHEETS.

You can save anytime to any directory you choose, and it is wise to save often.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and then shut down *Maple* and start it up again.

Part I: Spring Elongation

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

▪ 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.

```
> restart;
  with(plots): with(stats):
```

Warning, the name changecoords has been redefined

```
> data:= [[0, 0], [1, .875], [2, 1.721], [3, 2.641], [4, 3.531], [5, 4.391],[6, 5.241], [7, 6.120],
  [9, 7.869], [10, 8.741]];
```

$$data := [[0, 0], [1, 0.875], [2, 1.721], [3, 2.641], [4, 3.531], [5, 4.391], [6, 5.241], [7, 6.1$$
$$[9, 7.869], [10, 8.741]]$$

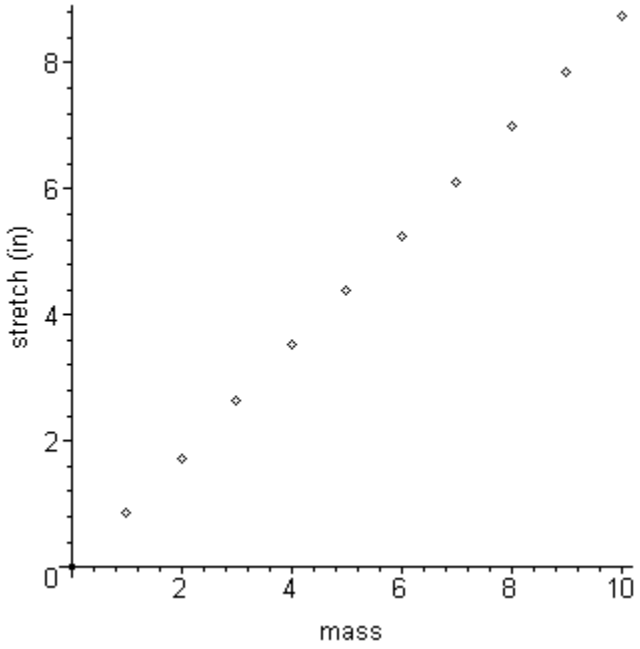
```
> matrix([[`mass `,`stretch (in)`],op(data)]);
```

mass	*stretch (in)*
0	0
1	0.875
2	1.721
3	2.641
4	3.531
5	4.391
6	5.241
7	6.120
8	6.992
9	7.869
10	8.741

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

```
> xlabel:="mass":
  ylabel:="stretch (in)":
```

```
p1:=pointplot(data, labels=[xlabel,ylabel], labeldirections=[HORIZONTAL, VERTI
print(p1);
```



▪ Designing a Model

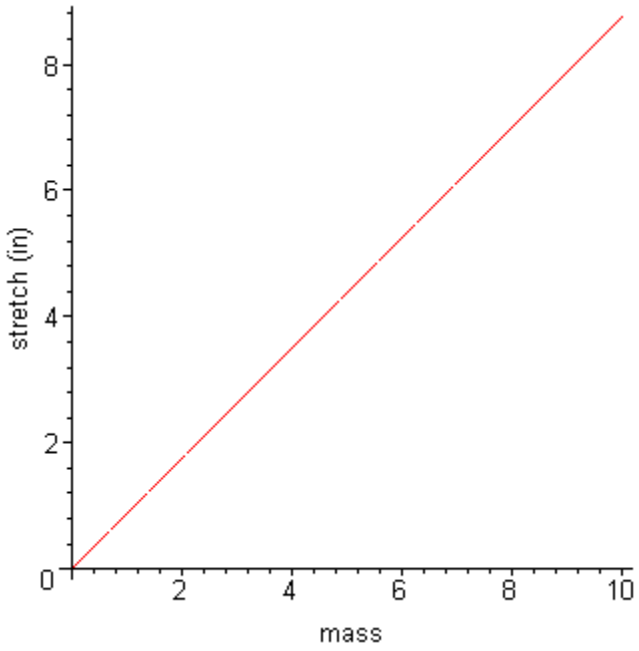
The data strongly suggest a linear relationship. Now we use the **fit[leastsquare]( )** function to find the line of best-fit or the linear regression line.

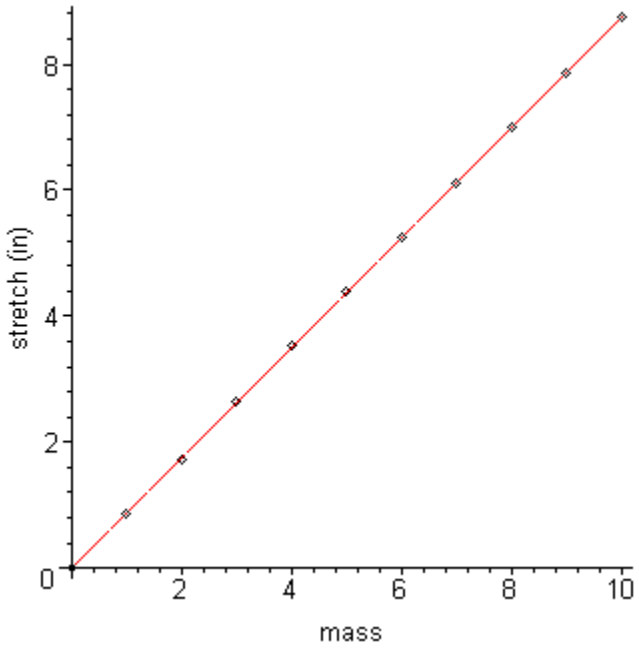
```
> data2 := [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:
  eqn:=fit[leastsquare[[x,y],y=a*x]]([data2[1],data2[2]]);
```

$$eqn := y = 0.8747324675\,x$$

The constant of proportionality is 0.8747. 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.

```
> p2:=implicitplot(eqn , x=0..10, y=-10..10, labels=[`mass`, `stretch (in)`], labeldirectior
  [HORIZONTAL, VERTICAL], numpoints=100): print(p2);
  print(display({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.

```
> f:=unapply(rhs(eqn),x):
  predictvalues:=[seq([i, f(i)], i=0..10)];
```

predictvalues := [[0, 0.], [1, 0.8747324675], [2, 1.749464935], [3, 2.624197402], [4, 3.49

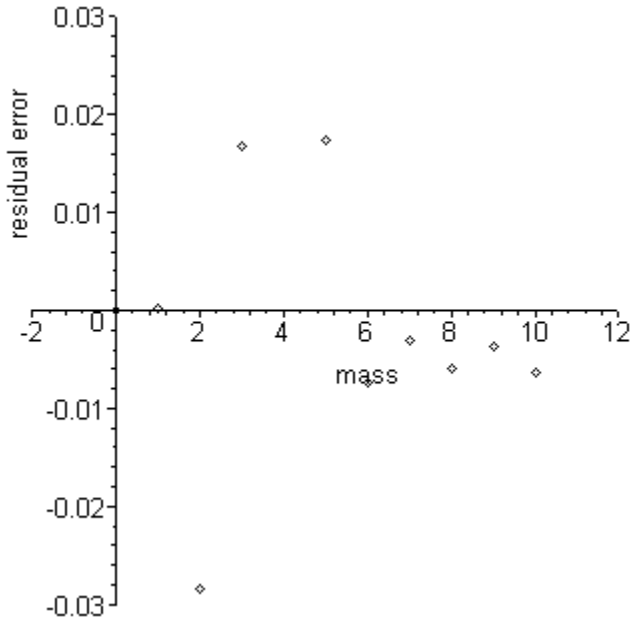
[5, 4.373662338], [6, 5.248394805], [7, 6.123127272], [8, 6.997859740], [9, 7.87259220
[10, 8.747324675]]

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

> **res:=data-predictvalues:**
residuals:=[seq([i-1, res[i,2]],i=1..11)];

residuals := [[0, 0.], [1, 0.0002675325], [2, -0.028464935], [3, 0.016802598], [4, 0.03207
[5, 0.017337662], [6, -0.007394805], [7, -0.003127272], [8, -0.005859740], [9, -0.00359
[10, -0.006324675]]

> **pointplot(residuals, labels=["mass","residual error"], labeldirections=[HORIZONTAL,VERTICAL],view=[-2..12,-0.03..0.03]);**

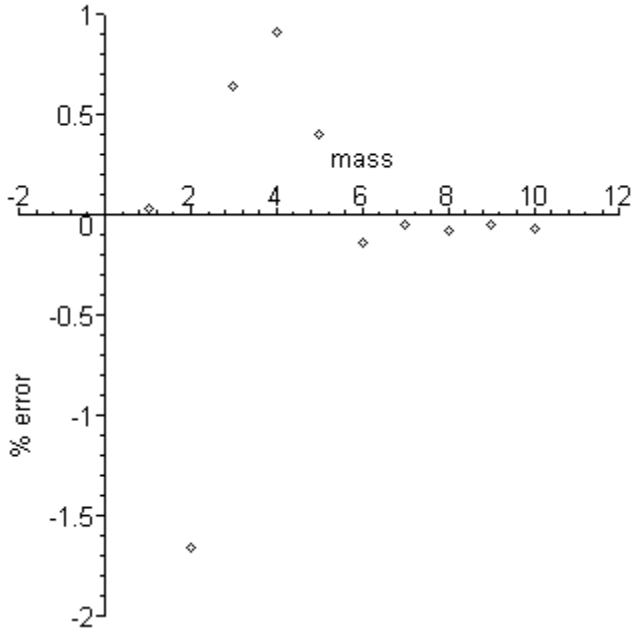


The residuals show that the differences are indeed small. Small is a relative term, and sometimes it is more meaningful to look at the error in relation to the size of the quantity being estimated. For all but the first data point, we calculate the relative error by dividing the measured value into the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

> **percenterrors:=[seq([i-1, 100*residuals[i,2]/data[i,2]],i=2..11)];**

percenterrors := [[1, 0.03057514286], [2, -1.653976467], [3, 0.6362210526], [4, 0.90824
[5, 0.3948454111], [6, -0.1410953062], [7, -0.05109921569], [8, -0.08380635011], [9, -0
[10, -0.07235642375]]

> **pointplot(percenterrors, labels=["mass","% error"], labeldirections=[HORIZONTAL,VERTICAL],view=[-2..12,-2..1]);**



Part II: Safe Following Distance

▪ 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.

```
> data := [[20., 22 + 32], [25, 28 + 47], [30, 33 + 65], [35, 39 + 87], [40,
      44 + 112], [45, 50 + 140], [50, 55 + 171], [55, 61 + 204], [60,
      66 + 241], [65, 72 + 282], [70, 77 + 325], [75, 83 + 376]];
```

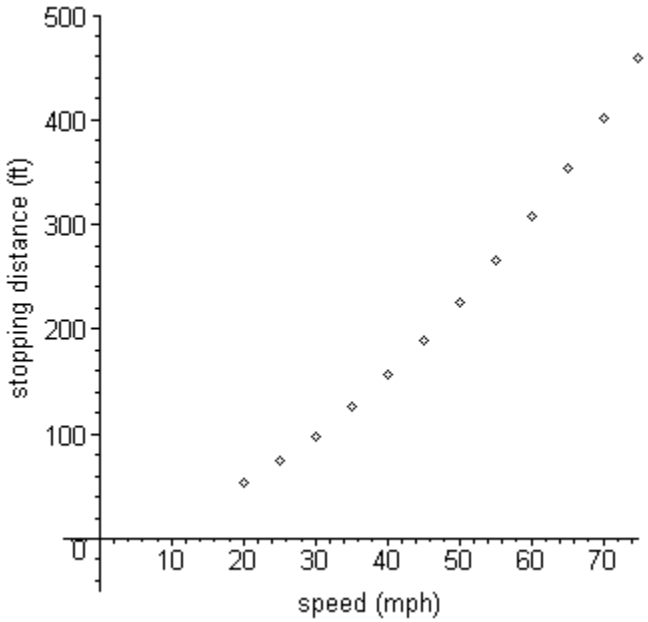
$$data := [[20., 54], [25, 75], [30, 98], [35, 126], [40, 156], [45, 190], [50, 226], [55, 265] [65, 354], [70, 402], [75, 459]]$$

```
> matrix([[`speed`, `stopping dist`],op(data)]);
```

speed	*stopping dist*
20.	54
25	75
30	98
35	126
40	156
45	190
50	226
55	265
60	307
65	354
70	402
75	459

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

```
> xlabel:="speed (mph)":
  ylabel:="stopping distance (ft)":
  p1:=pointplot(data, labels=[xlabel,ylabel], view=[-5..75,-50..500],labeldirections=
  [HORIZONTAL, VERTICAL]):
  print(p1);
```



▪ Designing a Model

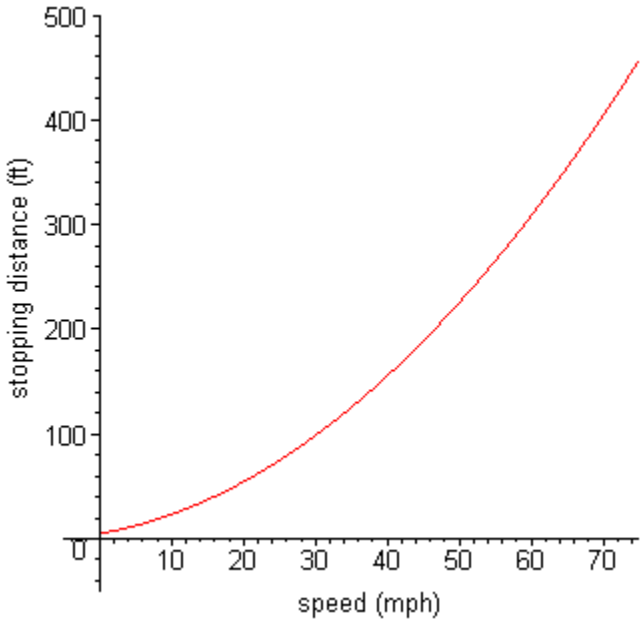
The data suggest a possible quadratic relationship. Now we use the **fit[leastsquare] ( )** function to find the curve of best-fit or the regression curve.

```
> temp := [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:
  eqn:=fit[leastsquare[[x,y], y=a+b*x+c*x^2] ](temp);
  f:=unapply(rhs(eqn),x):
```

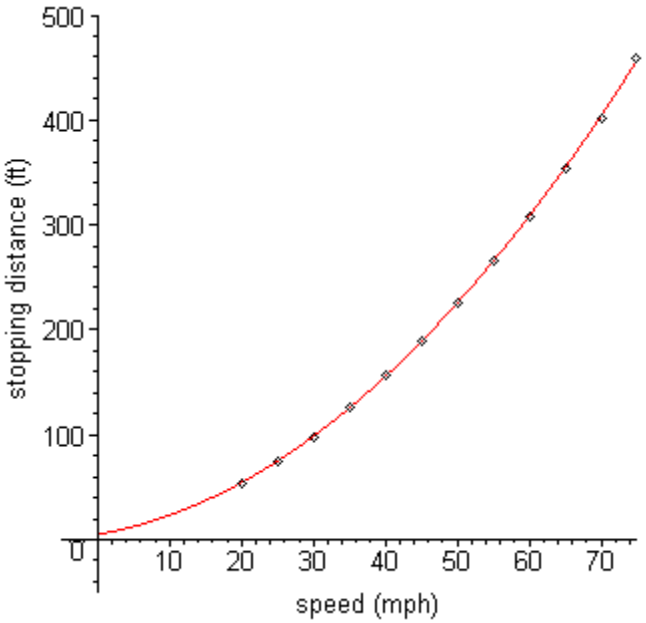
$$eqn := y = 5.039960040 + 1.180519481\,x + 0.06455544456\,x^2$$

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

```
> p2:=plot(f(x), x=0..85, labels=[xlabel,ylabel],view=[-5..75,-50..500], labeldirections=
  [HORIZONTAL, VERTICAL]):
  print(p2);
```




```
> print(display({p1,p2}));
```



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, and then we calculate the residual errors and plot them.

```
> f:=unapply(rhs(eqn),x):
  predictvalues:=[seq([i*5., f(i*5.)], i=4..15)];
```

```
predictvalues := [[20., 54.47252748], [25., 74.90009991], [30., 98.55544457], [35., 125.4
[40., 155.5494506], [45., 188.8881119], [50., 225.4545455], [55., 265.2487513], [60., 30
[65., 354.5204796], [70., 403.9980020], [75., 456.7032967]]
```

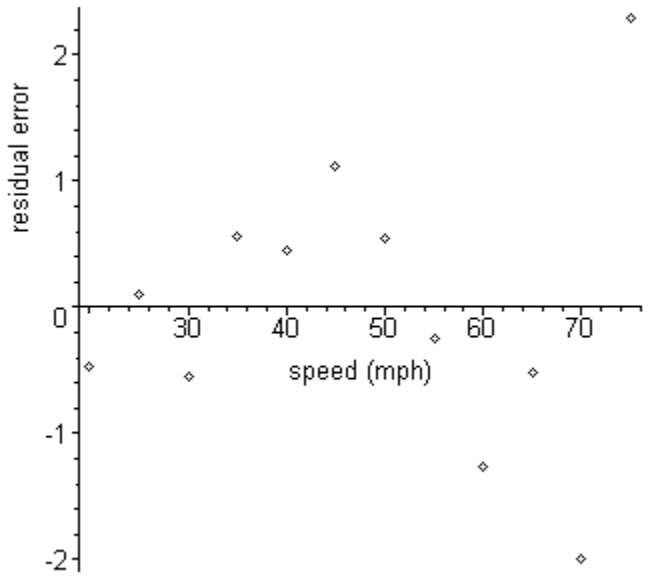
```
> matrix([[`speed `,`predicted values`],op(predictvalues)]);
```

speed	*predicted values*
20.	54.47252748
25.	74.90009991
30.	98.55544457
35.	125.4385615
40.	155.5494506
45.	188.8881119
50.	225.4545455
55.	265.2487513
60.	308.2707293
65.	354.5204796
70.	403.9980020
75.	456.7032967

```
> res:=data-predictvalues:
  residuals:=[seq([20+5*(i-1), res[i,2]],i=1..nops(data))];
```

residuals := [[20, -0.47252748], [25, 0.09990009], [30, -0.55544457], [35, 0.5614385], [4
[45, 1.1118881], [50, 0.5454545], [55, -0.2487513], [60, -1.2707293], [65, -0.5204796],
[70, -1.9980020], [75, 2.2967033]]

```
> pointplot(residuals, labels=[xlabel, "residual error"], labeldirections=[HORIZONTA
  VERTICAL]);
```

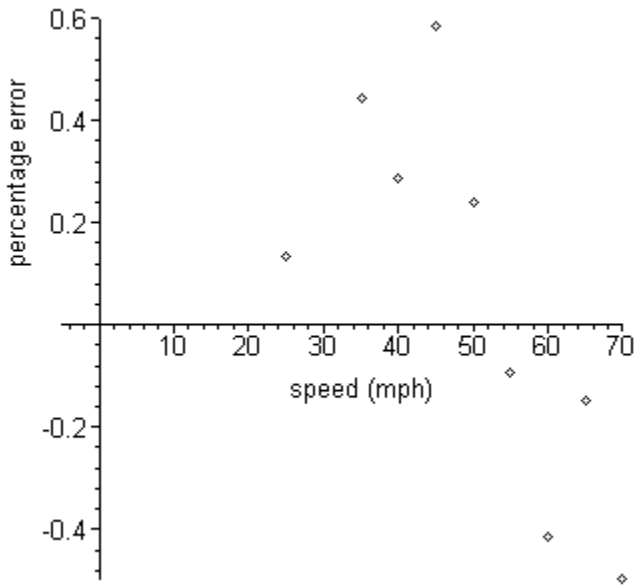


Once again we should look at the error in relation to the size of the quantity being estimated. For all but the first data point, we can calculate the relative error as follows.

```
> percenterrors:=[seq([20+5*(i-1), 100*residuals[i,2]/data[i,2]],i=2..11)];
```

```
percenterrors := [[25, 0.1332001200], [30, -0.5667801735], [35, 0.4455861111], [40, 0.28
[45, 0.5852042632], [50, 0.2413515487], [55, -0.09386841509], [60, -0.4139183388],
[65, -0.1470281356], [70, -0.4970154229]]
```

```
> pointplot(percenterrors, labels=[xlabel, "percentage error"], labeldirections=[HORI
VERTICAL],view=[-5..70,-0.5..0.6]);
```



The maximum relatvie 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.)

> **data2 := [seq([data[i, 1], (data[i, 1]*88./60.0)*2, data[i, 2]], i=1..12)];**

data2 := [[20., 58.66666668, 54], [25, 73.33333335, 75], [30, 88.00000002, 98], [35, 102.
[40, 117.3333334, 156], [45, 132.0000000, 190], [50, 146.6666667, 226], [55, 161.33333
[60, 176.0000000, 307], [65, 190.6666667, 354], [70, 205.3333334, 402], [75, 220.00000(

> **matrix([[`v (mph)`,`2-sec dist. (ft)`, `stopping dist(ft)`],op(data2)]);**

v (mph)	*2-sec dist. (ft)*	*stopping dist(ft)*
20.	58.66666668	54
25	73.33333335	75
30	88.00000002	98
35	102.6666667	126
40	117.3333334	156
45	132.0000000	190
50	146.6666667	226
55	161.3333334	265
60	176.0000000	307
65	190.6666667	354
70	205.3333334	402
75	220.0000000	459

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.

> **data3:=[seq([data[i, 1], data[i, 2]/(data[i, 1]*88./60)], i=1..(nops(data)))];**

data3 := [[20., 1.840909091], [25, 2.045454545], [30, 2.227272727], [35, 2.454545454],

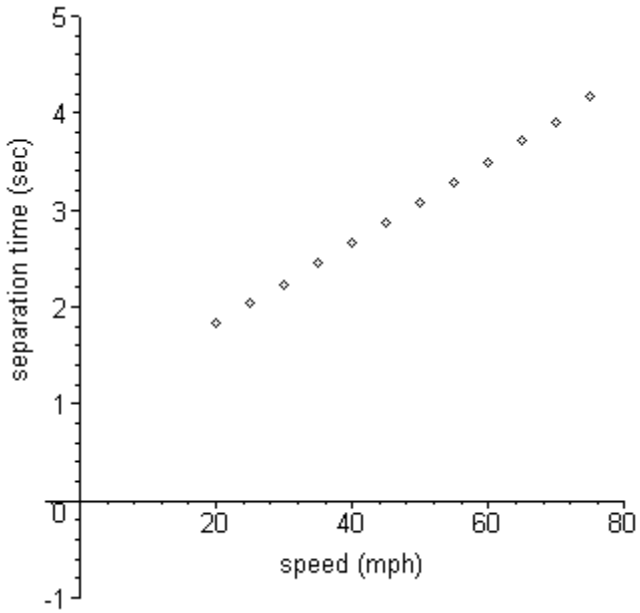
[40, 2.659090909], [45, 2.878787878], [50, 3.081818181], [55, 3.285123966], [60, 3.488
[65, 3.713286713], [70, 3.915584415], [75, 4.172727272]]

> **matrix([[`speed (mph)`,`separation time (sec)`],op(data3)]);**

speed (mph)	*separation time (sec)*
20.	1.840909091
25	2.045454545
30	2.227272727
35	2.454545454
40	2.659090909
45	2.878787878
50	3.081818181
55	3.285123966
60	3.488636363
65	3.713286713
70	3.915584415
75	4.172727272

Next time plot the separation time versus speed.

```
> xlabel:=`speed (mph)`:
  ylabel:=`separation time (sec)`:
  p1:=pointplot(data3, labels=[xlabel,ylabel], labeldirections=[HORIZONTAL, VERT
  view=[-5..80,-1..5]):
  print(p1);
```

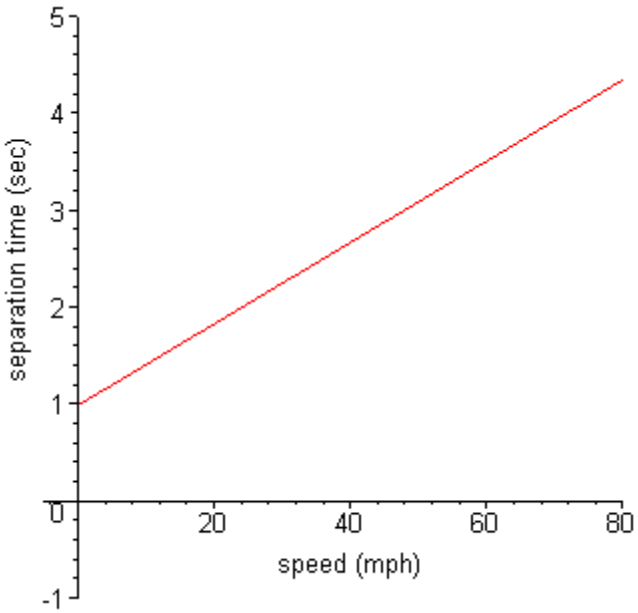


It appears that the separation time is a linear function of the travel speed and not a constant function as the 2-second rule suggests. Let's find the best-fit linear function for the separation-time versus speed data and plot it together with the data points.

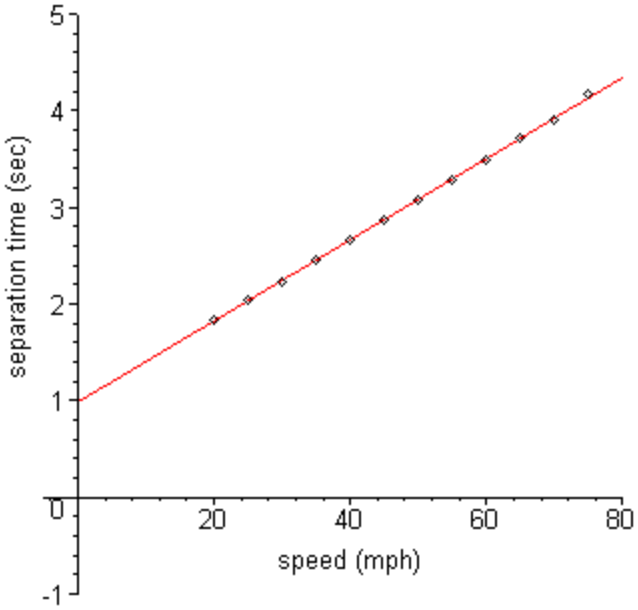
> **temp:= [[seq(data3[i,1],i=1..nops(data3))],[seq(data3[i,2],i=1..nops(data3))]]:**
fit[leastsquare[[x,y], y=a+b*x]](temp):
f:=unapply(rhs(%),x);

$$f := x \rightarrow 0.9827834932 + 0.04205234316\, x$$

> **p2:=plot(f(x), x=0..85, labels=[xlabel, ylabel], labeldirections=[HORIZONTAL, VER**
view=[-5..80,-1..5]):
print(p2);



> **print(display({p1,p2}));**



You Try It: Make a Better Rule of Thumb

Using the Model

Based upon the results of the analysis in Part II, 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 III: 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.

```
> data := [[0, 10023], [1, 8174], [2, 6693], [3, 5500], [4, 4489], [5, 3683], [6,3061], [7, 247
  2045], [9, 1645], [10, 1326]];
```

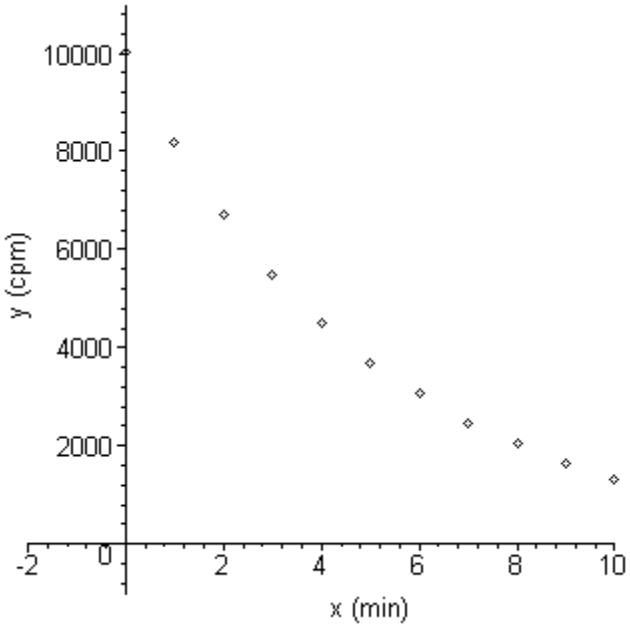
```
data := [[0, 10023], [1, 8174], [2, 6693], [3, 5500], [4, 4489], [5, 3683], [6, 3061], [7, 24
[9, 1645], [10, 1326]]
```

```
> matrix([[`Time (min)`,`Radioactivity (cpm)`],op(data)]);
```

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

We begin by plotting the data points.

```
> xlabel := "x (min)":
  ylabel := "y (cpm)":
  p1:=pointplot(data, labels=[xlabel, ylabel], labeldirections=[HORIZONTAL, VERTI
  view=[-2..10,-1000..11000]):
  print(p1);
```



▪ Designing a Model

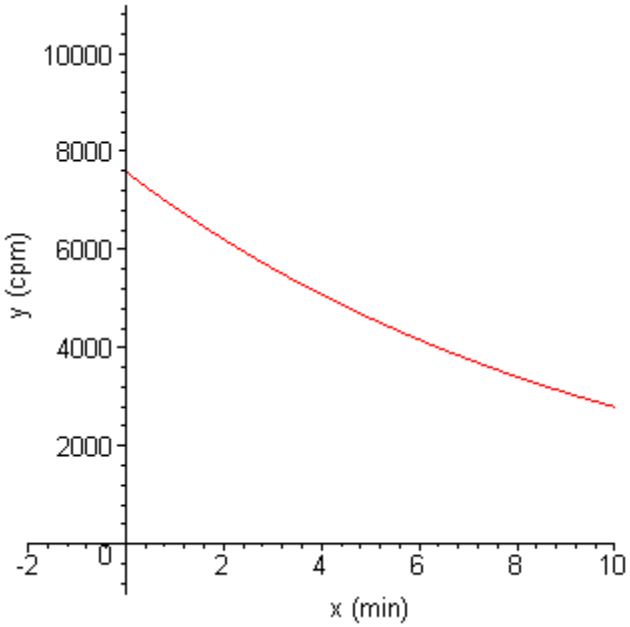
There appears to be a trend that we can capture with a mathematical model, and now we try to find a suitable model using the **fit[leastsquares]( )** function. What does the function look like to you? A decaying exponential? We try to find a function of the form $y = a\,e^{(-k\,x)}$, where we vary the value of **k** and the computer finds the **a** to find the best-fit function of this form.

```
> temp:= [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:
  fit[leastsquare[[x,y], y=a*exp(-.1*x),{a}]](temp):
  f2:=unapply(rhs(%),x);
```

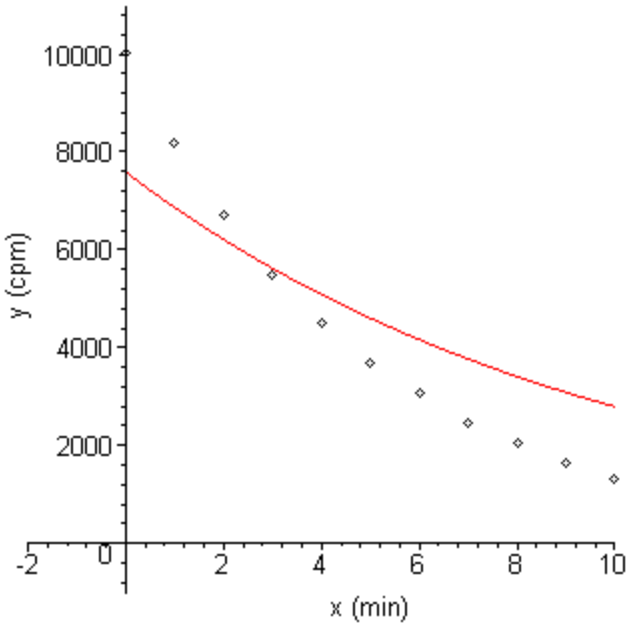
$$f2 := x \rightarrow 7584.054507\ e^{(-0.1\,x)}$$

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.

```
> p2:=plot(f2(x), x=0..12, y=0..10000, labels=["x (min)","y (cpm)"], labeldirections=
  [HORIZONTAL, VERTICAL], view=[-2..10,-1000..11000]):
  print(p2);
```



> **print(display({p1,p2}));**



▪ 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.

> **predictvalues:=[seq([i, f2(i)], i=0..10)]:**
res:=data-predictvalues:
residuals:=[seq([i-1, res[i,2]],i=1..11)];

$$residuals := [[0, 2438.945493], [1, 1311.663702], [2, 483.701342], [3, -118.405766], [4, -$$
$$[5, -916.961583], [6, -1101.217362], [7, -1287.130011], [8, -1362.735355], [9, -1438.44($$
$$[10, -1464.017734]]$$

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) all of the individual or local residuals, but this can be very misleading.

> **residavg:=add(residuals[i,2],i=1..11)/nops(residuals);**

$$residavg := -368.1225000$$

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 *Maple* **fit[leastsquare]( )** 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.

> **residsquaresum:=add((residuals[i,2])^2,i=1..11);**

$$residsquaresum := 0.1805034954\ 10^{8}$$

Before we compare this value, we calculate the mean of the squared residuals.

> **msresiduals:=residsquaresum/nops(residuals);**

$$msresiduals := 0.1640940867\ 10^{7}$$

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.

```
> rmsresiduals:=sqrt(msresiduals);
```

$$rmsresiduals := 1280.992142$$

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

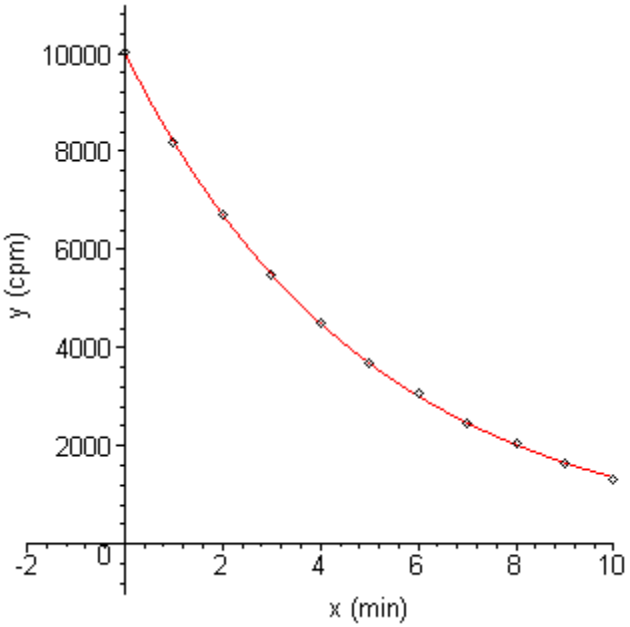
Improving the Model

In Part III, 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 to reduce (possibly minimize) the sum of the squared residuals (which will also minimize the "rms" value).

A *comment about errors in modeling*: Keep in mind that in mathematical modeling you may not be 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 still 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.

```
> k:= 0.2:
  temp:= [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:
  fit[leastsquare[[x,y], y=a*exp(-k*x),{a}]](temp);
  f:=unapply(rhs(%),x):
  p2:=plot(f(x), x=0..10, labels=[xlabel, ylabel]):
  print(display({p2, p1}));
  residuals := [seq([data[i, 1], data[i, 2] - f(data[i, 1])], i=1..nops(data))]:
  residsquaresum:=add((residuals[i,2])^2,i=1..11):
  mresiduals:=residsquaresum/nops(data):
  rmsresiduals:=sqrt(mresiduals);
```

$$y = 10009.89047\, e^{(-0.2\, x)}$$



rmsresiduals := 20.80352440

Making Predictions

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

> **solve(f(x)=500,x);**

14.98360416

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

Part IV: Ponderosa Pines

▪ Observing the Data

In the table that follows, the girth of the pine tree (the distance arond 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?

> **data := [[17, 19], [19, 25], [20, 32], [23, 57], [25, 71], [28, 113], [32,123], [38, 252], [39, 294]];**

data :=

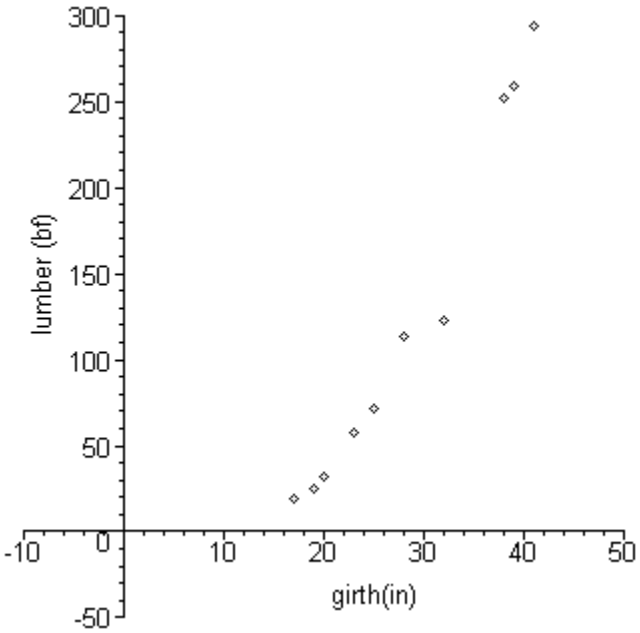
[[17, 19], [19, 25], [20, 32], [23, 57], [25, 71], [28, 113], [32, 123], [38, 252], [39, 259]

> **matrix([[`girth (in)`,`lumber (bf)`],op(data)]);**

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

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

> **xlabel:=`girth(in)`:**
ylabel:=`lumber (bf)`:
p1:=pointplot(data, labels=[xlabel,ylabel], labeldirections=[HORIZONTAL, VERTI
view=[-10..50,-50..300]):
print(p1);



▪ Designing a Model

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

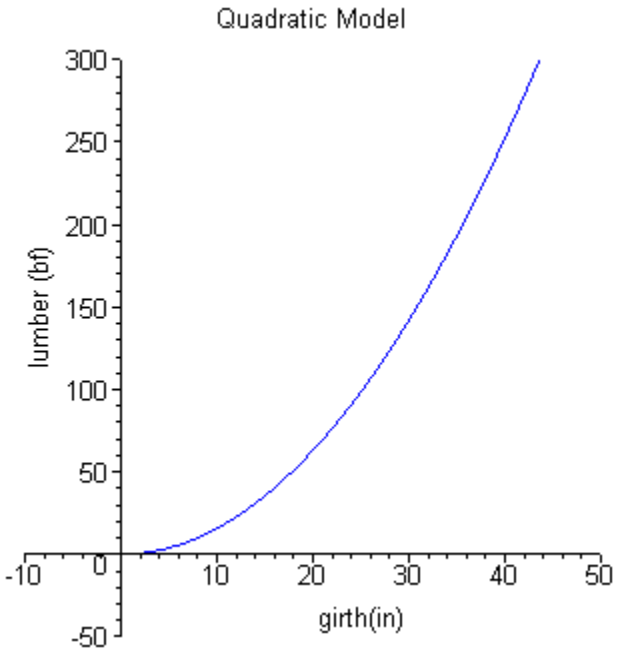
> **temp:= [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:**
fit[leastsquare[[x,y], y=a*x^2.,{a}]](temp):
y2:=unapply(rhs(%),x);

$$y2 := x \to 0.1579186736\, x^{2.}$$

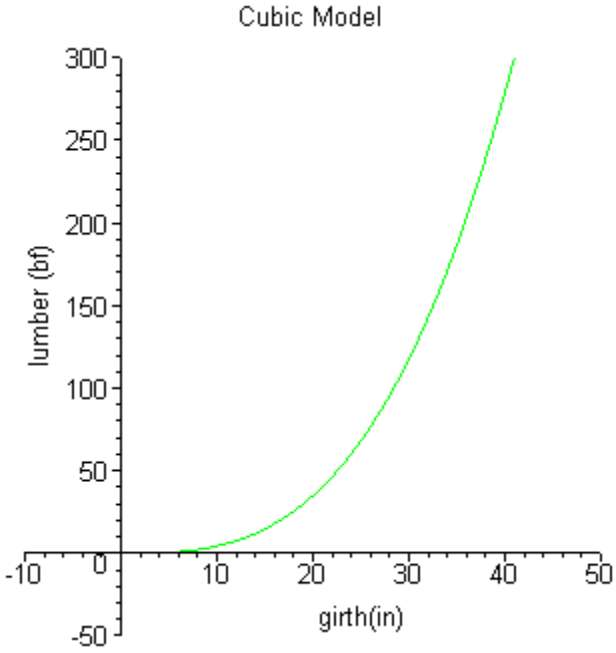
> **fit[leastsquare[[x,y], y=a*x^3.,{a}]](temp):**
y3:=unapply(rhs(%),x);

$$y3 := x \to 0.004362034723\, x^{3.}$$

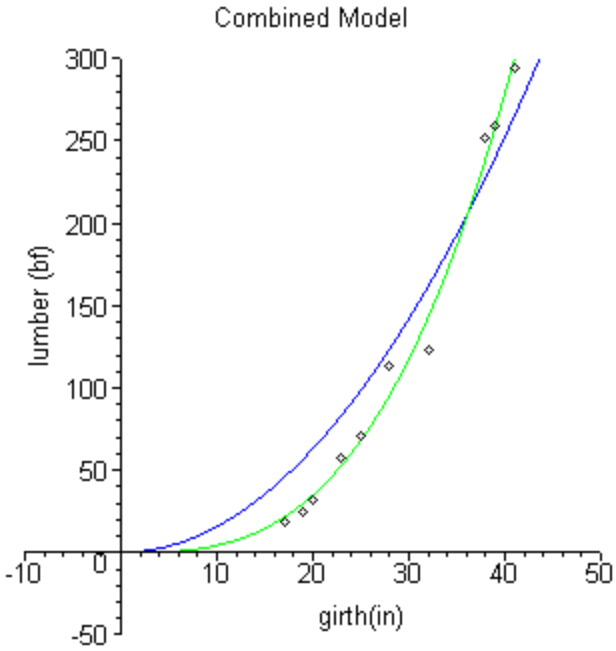
> **p2:=plot(y2(x),x=0..45, labels=[xlabel, ylabel], color=blue, title="Quadratic Model", labeldirections=[HORIZONTAL, VERTICAL], view=[-10..50,-50..300]):**
print(p2);



> **p3:=plot(y3(x),x=0..45, labels=[xlabel, ylabel],title="Cubic Model", color=green, labeldirections=[HORIZONTAL, VERTICAL], view=[-10..50,-50..300]):**
print(p3);



> **print(display({p1,p2,p3}, labels=[xlabel, ylabel], title="Combined Model"));**



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

▪ 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.

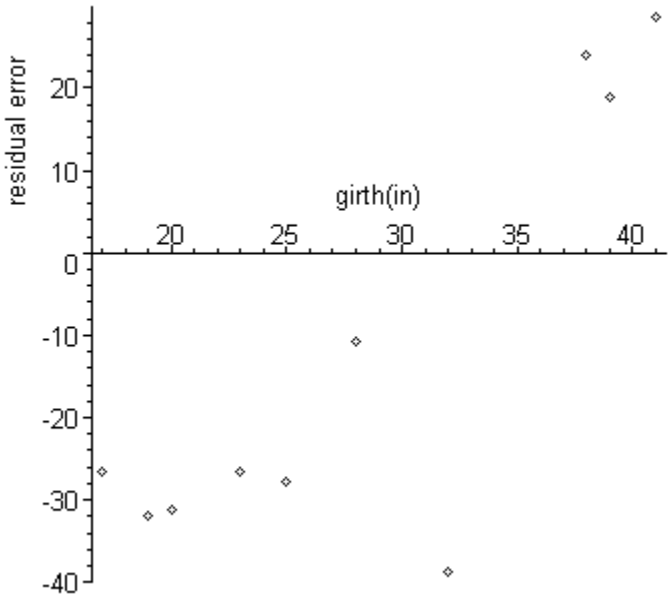
> **quadpredictedvalues:=[seq([data[i,1], y2(data[i,1])], i=1..nops(data))];**

quadpredictedvalues := [[17, 45.63849667], [19, 57.00864117], [20, 63.16746944], [25, 98.69917100], [28, 123.8082401], [32, 161.7087218], [38, 228.0345647], [39, [41, 265.4612903]]

> **quadresiduals:=[seq([data[i,1], data[i,2]-quadpredictedvalues[i,2]],i=1..nops(da**

quadresiduals := [[17, -26.63849667], [19, -32.00864117], [20, -31.16746944], [23, [25, -27.69917100], [28, -10.8082401], [32, -38.7087218], [38, 23.9654353], [39, [41, 28.5387097]]

> **pointplot(quadresiduals, labels=[xlabel,"residual error"], labeldirections=[HO VERTICAL]);**



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.

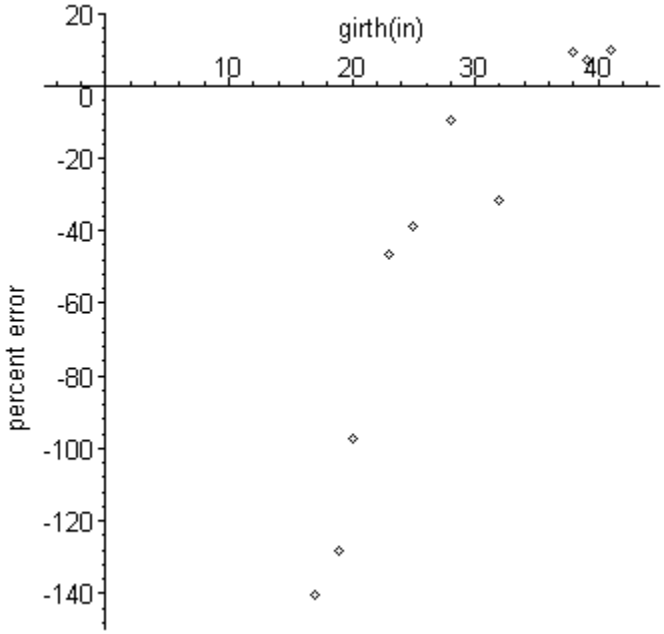
It is usually more helpful to look at the error in relation to the size of the quantity being estimated. We calculate the relative percent errors as follows.

> **quadpercenterrors:=[seq([data[i,1],100*(data[i,2]-quadpredictedvalues[i,2])/da (data))];**

quadpercenterrors := [[17, -140.2026141], [19, -128.0345647], [20, -97.39834200], [23, -46.55961111], [25, -39.01291690], [28, -9.564814248], [32, -31.47050553], [[39, 7.260887066], [41, 9.707044116]]

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

> **p3:=pointplot(quadpercenterrors, labels=[xlabel,"percent error"], labeldirecti [HORIZONTAL, VERTICAL],view=[-5..45,-150..20]):**
print(p3);



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

Analysis of the Errors for the Cubic-Fit Function

Let's look at the errors for the cubic model. We do the same as we did for the quadratic function.

> **cubicpredictedvalues:=[seq([data[i,1], y3(data[i,1])], i=1..nops(data))];**

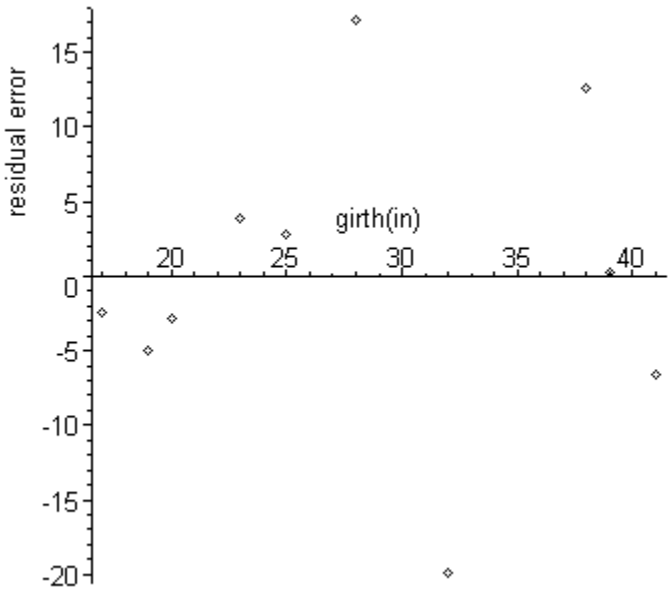
cubicpredictedvalues := [[17, 21.43067659], [19, 29.91919617], [20, 34.89627778], [25, 68.15679255], [28, 95.75538624], [32, 142.9351538], [38, 239.3535693], [39, [41, 300.6357951]]

>

```
> cubicresiduals:=[seq([data[i,1], data[i,2]-cubicpredictedvalues[i,2]],i=1..nops(d
```

cubicresiduals := [[17, -2.43067659], [19, -4.91919617], [20, -2.89627778], [23, 3.
[25, 2.84320745], [28, 17.24461376], [32, -19.9351538], [38, 12.6464307], [39, 0.
[41, -6.6357951]]

```
> pointplot(cubicresiduals, labels=[xlabel,"residual error"], labeldirections=[HO
  VERTICAL]);
```

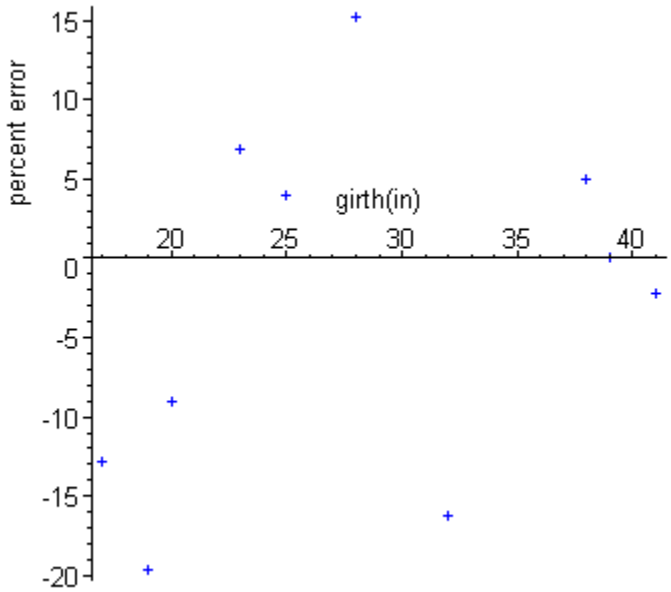


Note that the residuals exhibit a more random pattern than they did in the quadratic model. There are as many negative errors as positive ones, and there is no apparent trend in the errors.

```
> cubicpercenterrors:=[seq([data[i,1],100*(data[i,2]-cubicpredictedvalues[i,2])/d
  (data))];
```

cubicpercenterrors := [[17, -12.79303468], [19, -19.67678468], [20, -9.050868062]
[25, 4.004517535], [28, 15.26072014], [32, -16.20744211], [38, 5.018424881], [39
[41, -2.257073163]]

```
> p4:=pointplot(cubicpercenterrors, labels=[xlabel,"percent error"], color=blue,
  labeldirections=[HORIZONTAL, VERTICAL]):
  print(p4);
```

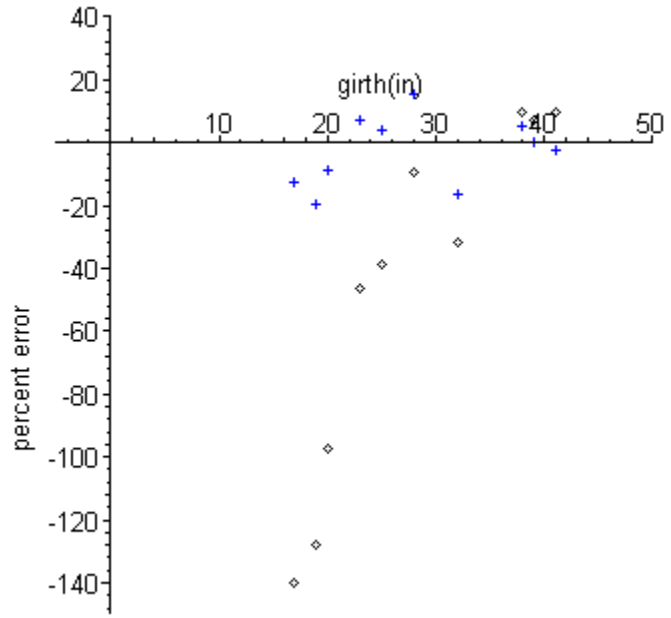


In this case, the percent error of largest magnitude is about -20%, which is much better than for the quadratic regression function.

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.

> **print(display({p3,p4}, view=[-5..50,-150..40]));**



▪ 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 $V = \frac{\pi^2 r h}{3}$ 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 $V = \frac{\pi}{3} \left(\frac{g}{2\pi}\right)^2 \frac{kg}{2\pi} = \frac{k}{24\pi^2} g^3$, where the last quantity $\frac{k}{24\pi^2}$, 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.

Part V: Black Bass

▪ Observing the Data

In the table that follows, L represents the lengths of New York black bass measured in inches (in.), 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?

```
> data := [[12.5, 17], [12.63, 16], [14.13, 17], [14.5, 23], [14.5, 26], [14.5,
      27], [17.25, 41], [17.75, 49]];
```

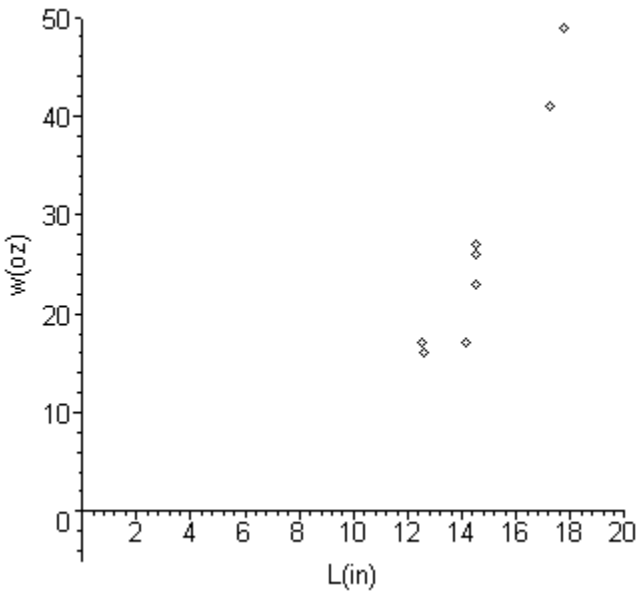
$$data := [[12.5, 17], [12.63, 16], [14.13, 17], [14.5, 23], [14.5, 26], [14.5, 27], [17.25, 41]]$$

```
> matrix([[`L (in)`, `W (oz)`],op(data)]);
```

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

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

```
> xlabel:=`L(in)`:
  ylabel:=`w(oz)`:
  p1:=pointplot(data, labels=[xlabel, ylabel], view=[0..20,-5..50], labeldirections=[HOR
  VERTICAL]):
  print(p1);
```



▪ Designing a Model

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

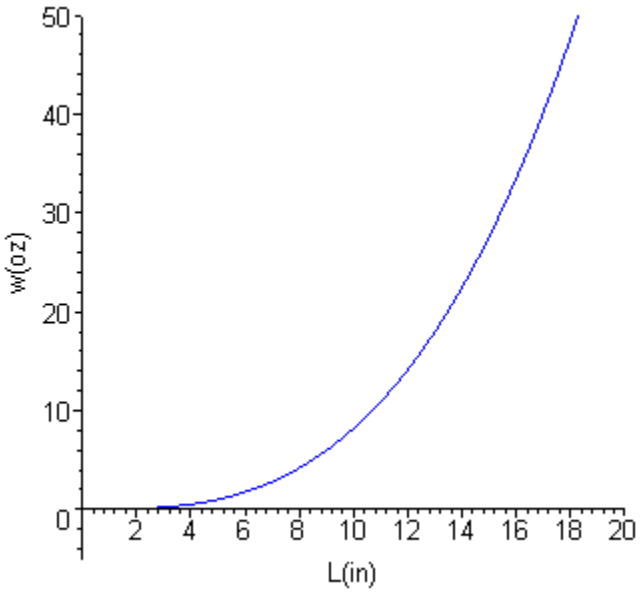
```
> temp := [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:
  eqn:=fit[leastsquare[[x,y],y=a*x^3]]([temp[1],temp[2]]);
```

y:=unapply(rhs(%),x):

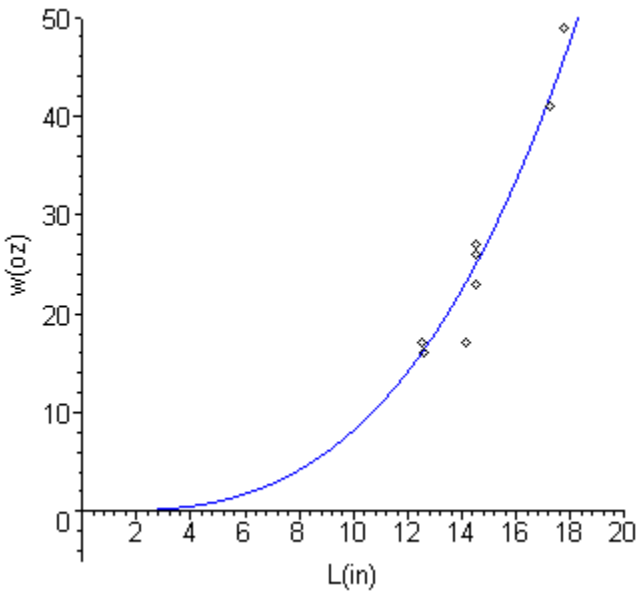
$$eqn := y = 0.008186279627\, x^3$$

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

> **p2:=plot(y(x), x=0..20, labels=[xlabel,ylabel], color=blue, labeldirections=[HORIZON VERTICAL], view=[0..20,-5..50]):**
print(p2);



> **print(display({p1,p2}));**



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.

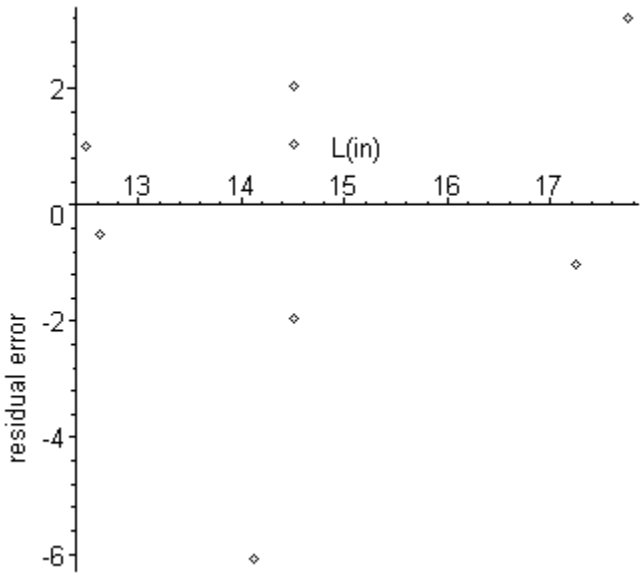
> **predictvalues:=[seq([data[i,1], y(data[i,1])], i=1..nops(data))];**

predictvalues := [[12.5, 15.98882740], [12.63, 16.49288485], [14.13, 23.09473912], [14.5
[14.5, 24.95689673], [14.5, 24.95689673], [17.25, 42.01978959], [17.75, 45.78061762]]

> **res:=data-predictvalues:**
residuals:=[seq([data[i,1], res[i,2]],i=1..nops(data))];

residuals := [[12.5, 1.01117260], [12.63, -0.49288485], [14.13, -6.09473912], [14.5, -1.95
[14.5, 1.04310327], [14.5, 2.04310327], [17.25, -1.01978959], [17.75, 3.21938238]]

> **pointplot(residuals, labels=[xlabel, `residual error`], labeldirections=[HORIZONTAL
VERTICAL]);**



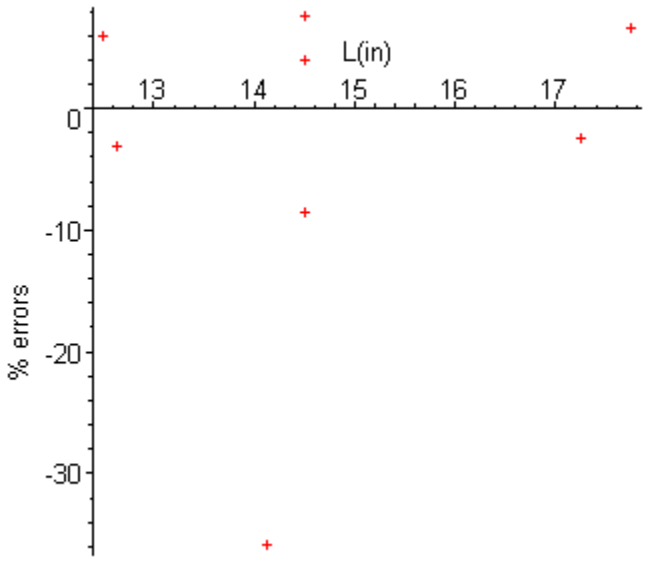
Note that there are nearly equal numbers of positive and negative errors, and there is no apparent trend in the errors. They appear to be random.

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.

> **percenterrors:=[seq([data[i,1],100*(data[i,2]-predictvalues[i,2])/data[i,2]],i=1..nops(d**

percenterrors := [[12.5, 5.948074118], [12.63, -3.080530312], [14.13, -35.85140659], [14.5, -8.508246652], [14.5, 4.011935654], [14.5, 7.567049148], [17.25, -2.487291683], [17.75, 6.570168122]]

> **p3:=pointplot(percenterrors, labels=[xlabel,`% errors`], color=red, symbol=CROSS, labeldirections=[HORIZONTAL, VERTICAL]):**
print(p3);

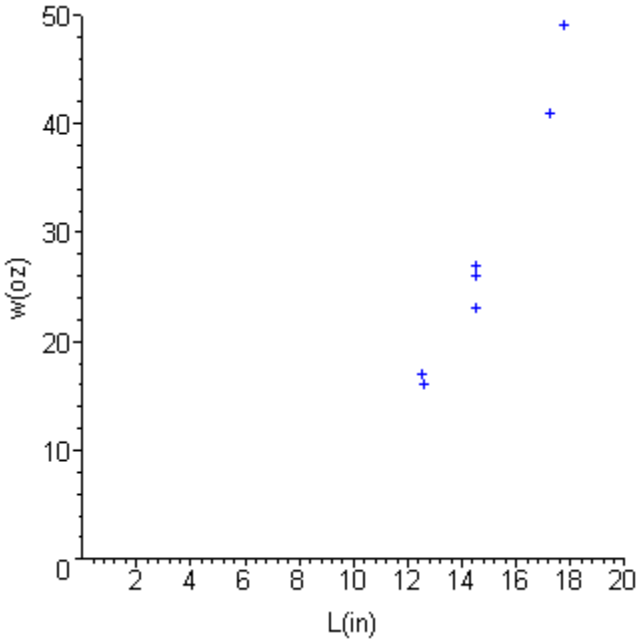


▪ Improving the Model

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.

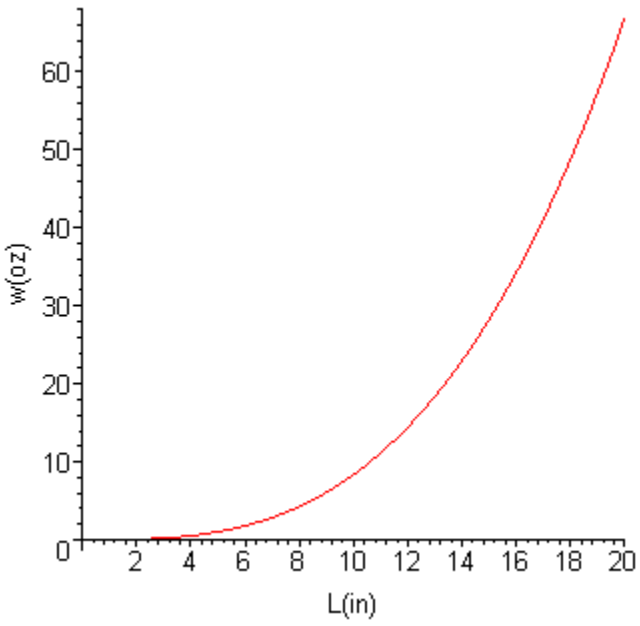
> **data := [[12.5, 17], [12.63, 16], [14.13, 17], [14.5, 23], [14.5, 26], [14.5,27], [17.25, 41], [**
data:=subsop(3=NULL,data);
p1:=pointplot(data, view=[0..20,0..50], symbol=CROSS, color=blue,labels=[xlabel,yla
labeldirections=[HORIZONTAL, VERTICAL]):
print(p1);

data := [[12.5, 17], [12.63, 16], [14.5, 23], [14.5, 26], [14.5, 27], [17.25, 41], [17.75, 49]

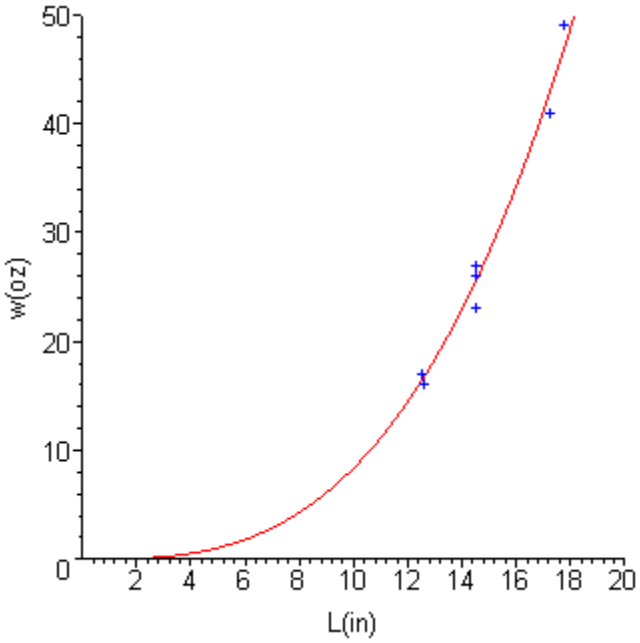



```
> temp := [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:
  eqn:=fit[leastsquare[[x,y],y=a*x^3]]([temp[1],temp[2]]);
  y:=unapply(rhs(%),x):
  p2:=plot(y(x), x=0..20, labels=[xlabel,ylabel], color=red, labeldirections=[HORIZON
  VERTICAL]):
  print(p2);
```

$$eqn := y = 0.008370415426\, x^{3}$$

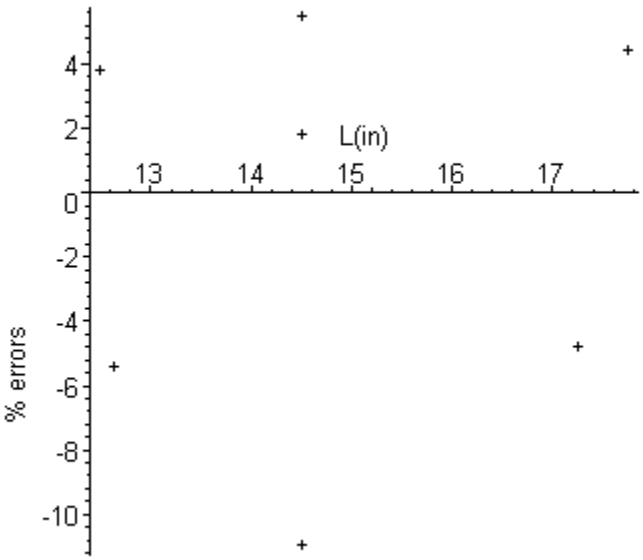



```
> print(display({p1,p2}));
```




```
> predictvalues:=[seq([data[i,1], y(data[i,1])], i=1..nops(data))]:
  res:=data-predictvalues:
  residuals:=[seq([data[i,1], res[i,2]],i=1..nops(data))]:
  percenterrors:=[seq([data[i,1],100*(data[i,2]-predictvalues[i,2])/data[i,2]],i=1..nops(d
  p3:=pointplot(percenterrors, labels=[xlabel,`% errors`], color=black, symbol=CROS
  labeldirections=[HORIZONTAL, VERTICAL]):
  print(p3);
```

percenterrors := [[12.5, 3.832543353], [12.63, -5.399143500], [14.5, -10.94894665], [14.
[14.5, 5.487934333], [17.25, -4.792561024], [17.75, 4.468630245]]



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 $V = \frac{4\pi L w d}{3}$ 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_L and k_d are constants. Therefore, $V = \frac{4\pi L (k_w L)(k_d L)}{3} = \frac{4\pi k_w k_d}{3} L^3$ 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.

Part VI: 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.

```
> data := [[4, 660], [25, 670], [200, 420], [300, 300], [2000, 205], [5000,120], [30000, 85],
  [70000, 72], [450000, 38], [500000,40], [3000000, 48]];
```

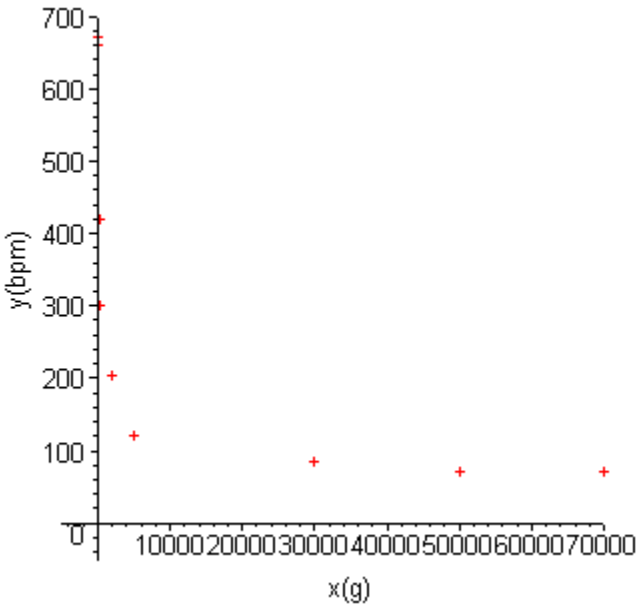
```
data := [[4, 660], [25, 670], [200, 420], [300, 300], [2000, 205], [5000, 120], [30000, 85
[70000, 72], [450000, 38], [500000, 40], [3000000, 48]]
```

```
> matrix([[` x (g) `,`y (bpm) `],op(data)]);
```

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

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

```
> xlabel := "x(g)":
  ylabel := "y(bpm)":
  p1:=pointplot(data, symbol=CROSS, color=red, labels=[xlabel, ylabel], view=[-5000..
  50..700], labeldirections=[HORIZONTAL, VERTICAL]):
  print(p1);
```



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 graph of the data suggests 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[leastsquare]( )** function to find the function of the form $x^{-1/n}$ that best fits the data.

```
> temp := [[seq(data[i,1],i=1..nops(data))],[seq(data[i,2],i=1..nops(data))]]:
  for n from 1 to 5 do
    eqn:=fit[leastsquare[[x,y],y=a*x^(-1.0/n),{a}]]([temp[1],temp[2]]):
    y[n]:=unapply(rhs(%),x);
  od;
```

$$eqn := y = \frac{3040.893063}{x^{1.0}}$$

$$y_1 := x \rightarrow \frac{3040.893063}{x^{1.0}}$$

$$eqn := y = \frac{1733.558345}{x^{0.5000000000}}$$

$$y_2 := x \rightarrow \frac{1733.558345}{x^{0.5000000000}}$$

$$eqn := y = \frac{1371.608344}{x^{0.3333333333}}$$

$$y_3 := x \rightarrow \frac{1371.608344}{x^{0.3333333333}}$$

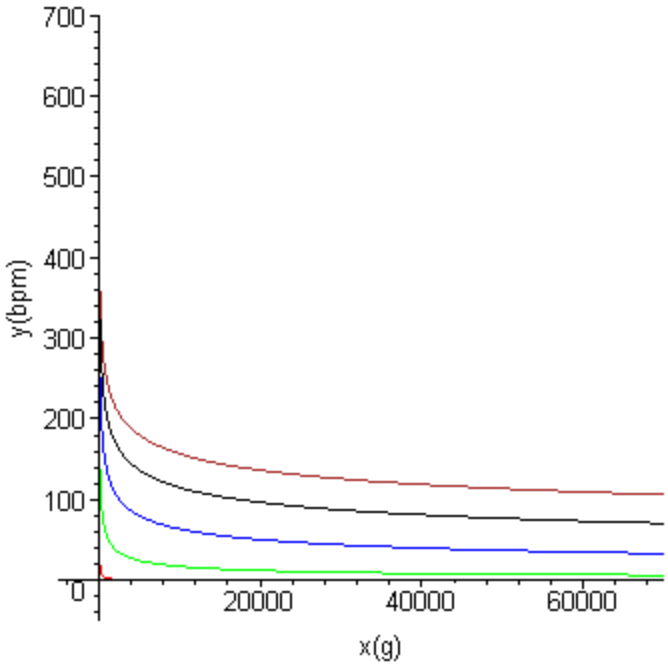
$$eqn := y = \frac{1149.868328}{x^{0.2500000000}}$$

$$y_4 := x \to \frac{1149.868328}{x^{0.2500000000}}$$

$$eqn := y = \frac{989.4577909}{x^{0.2000000000}}$$

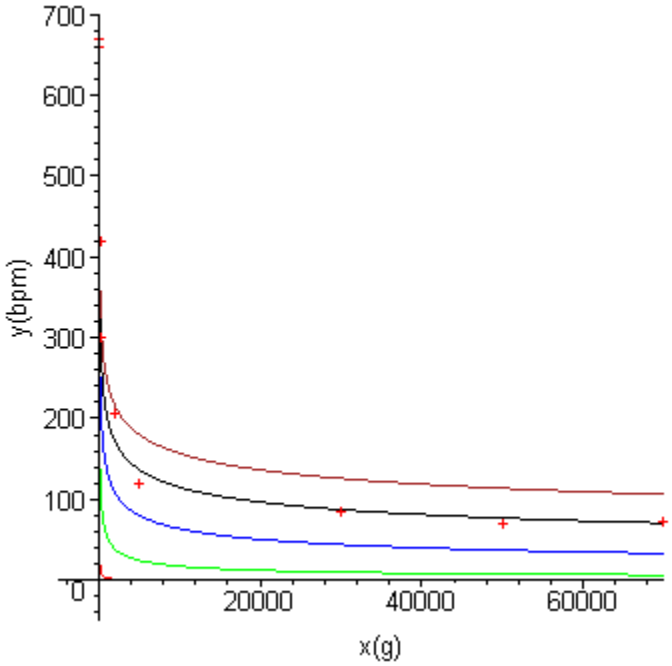
$$y_5 := x \to \frac{989.4577909}{x^{0.2000000000}}$$

> **p2:=plot([y[1](x), y[2](x),y[3](x),y[4](x),y[5](x)], x=0..70000, y=0..700, labels=[x ylabel],tickmarks=[4,7], color=[red,green,blue,black,brown], view=[-5000..700(labeldirections=[HORIZONTAL, VERTICAL]):**
print(p2);



Now we plot the models together with the data points.

> **print(display({p1,p2}));**



▪ Assessing the Errors

It appears that y_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.

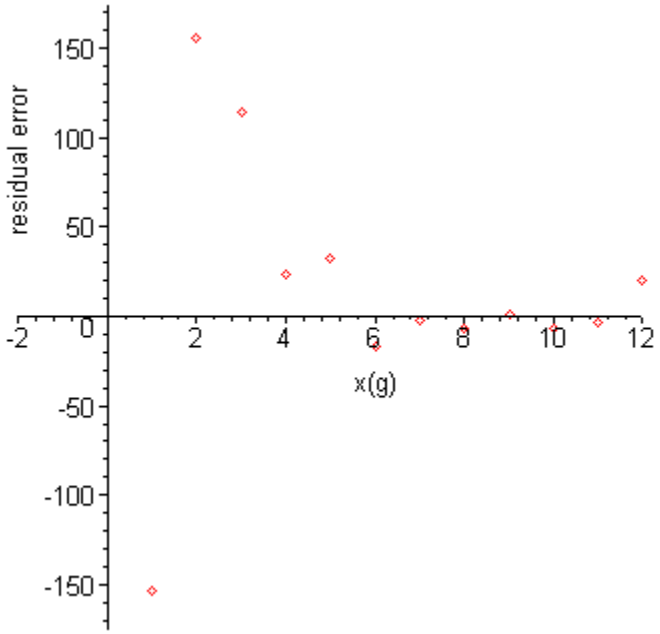
> **predictedvalues:=[seq([data[i,1], y[4](data[i,1])], i=1..nops(data))];**

predictedvalues := [[4, 813.0796922], [25, 514.2367493], [200, 305.7670005], [300
[2000, 171.9454203], [5000, 136.7431597], [30000, 87.37109895], [50000, 76.896
[70000, 70.69254256], [450000, 44.39611661], [500000, 43.24198391], [3000000,

> **residuals:=[seq([i, data[i,2]-predictedvalues[i,2]],i=1..nops(data))];**

residuals := [[1, -153.0796922], [2, 155.7632507], [3, 114.2329995], [4, 23.708325
[6, -16.7431597], [7, -2.37109895], [8, -6.89632964], [9, 1.30745744], [10, -6.396
[11, -3.24198391], [12, 20.37083257]]

> **pointplot(residuals, color=red, labels=[xlabel,`residual error`], view=[-2..12,-17
labeldirections=[HORIZONTAL, VERTICAL]);**

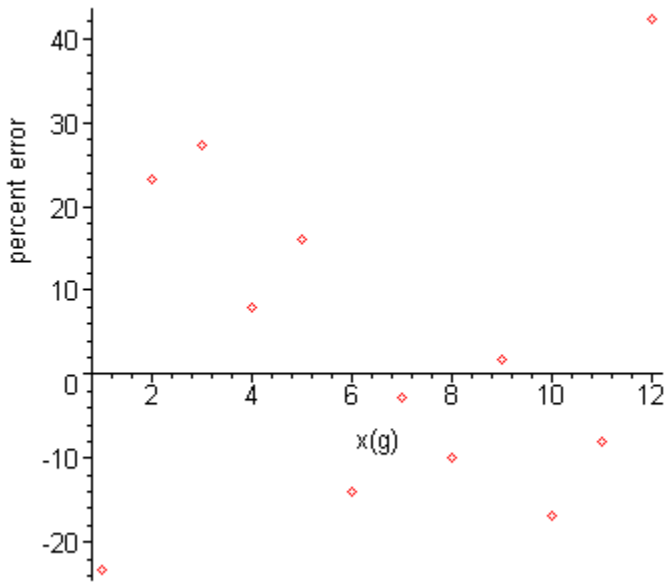


As before, it is helpful to look at the error in relation to the size of the quantity being estimated. We calculate the relative percent errors.

> **percenterrors:=[seq([i,100*(data[i,2]-predictedvalues[i,2])/data[i,2]],i=1..nops(**

percenterrors := [[1, -23.19389276], [2, 23.24824637], [3, 27.19833321], [4, 7.902
[5, 16.12418522], [6, -13.95263308], [7, -2.789528176], [8, -9.851899486], [9, 1.8
[10, -16.83188582], [11, -8.104959775], [12, 42.43923452]]

> **pointplot(percenterrors, color=red, labels=[xlabel,`percent error`], labeldirecti**
[HORIZONTAL, VERTICAL]);



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 VI.

You Try It: Modeling Takes Practice

Mathematical modeling and a computer algebra system like *Maple* 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.

Drug Levels

> **drugdata := [[0, 853], [1, 587], [2, 390], [3, 274], [4, 189], [5, 130], [6,97], [7, 67], [8, 50], [9, 4**
[10, 31]];

drugdata :=
[[0, 853], [1, 587], [2, 390], [3, 274], [4, 189], [5, 130], [6, 97], [7, 67], [8, 50], [9, 40], [10, 1

Cell Count

> **celldata:= [[0, 597], [2, 893], [4, 1339], [6, 1995], [8, 2976], [10,**
4433], [12, 6612], [14, 9865], [16, 14719], [18, 21956], [20, 32763]];

celldata := [[0, 597], [2, 893], [4, 1339], [6, 1995], [8, 2976], [10, 4433], [12, 6612], [14, 9865
[16, 14719], [18, 21956], [20, 32763]]

Vacuum Pump

> **vacuumdata := [[0, 100000], [1, 36788], [2, 13537], [3, 4986], [4, 1837], [5,**
671]];

vacuumdata := [[0, 100000], [1, 36788], [2, 13537], [3, 4986], [4, 1837], [5, 671]]

Doctoral Degrees

> **doctoraldata := [[6, 520], [10, 460], [14, 680], [18, 630], [20, 730], [21,810], [22, 830]];**

doctoraldata := [[6, 520], [10, 460], [14, 680], [18, 630], [20, 730], [21, 810], [22, 830]]

>

>