Graph Plotting and Data Analysis using Mathematica

The purpose of these notes is to show how *Mathematica* can be used to analyze laboratory data. The notes are not complete, since there are many commands that are not discussed here. For further information you should consult the online Help menu or the *Mathematica* Book.

It is good practice to reset everything before you begin a Mathematica session:

```
In[1]:= Clear["Global`*"]
```

Data Lists

Mathematica has some powerful functions for manipulating lists of data. Consider a list of numbers (which we'll call **list1**):

```
In[2]:= list1 = {26, 13, 4, 0.3, 3, -2, 0.08, 19.3}
Out[2]= {26, 13, 4, 0.3, 3, -2, 0.08, 19.3}
```

You can add, subtract, multiply or divide by a constant very easily. For example, we can create a new **list2** by adding 3 to each term:

```
In[3]:= list2 = list1 + 3
Out[3]= {29, 16, 7, 3.3, 6, 1, 3.08, 22.3}
```

If you want to multiply each term in **list1** by $1/4\pi\epsilon_o$, first define ϵ_o and then multiply each element in **list1** by it.

In[4]:= ϵ_o = 8.85 10^-12;

In[5]:= list3 = list1 $\frac{1}{4\pi\,\epsilon_o}$

Out[5]= $\{2.33787\times10^{11}, 1.16893\times10^{11},$
$3.59672\times10^{10}, 2.69754\times10^{9}, 2.69754\times10^{10},$
$-1.79836\times10^{10}, 7.19344\times10^{8}, 1.73542\times10^{11}\}$

Other operations can be applied similarly. For example, you can obtain the natural logarithm of each term with the **Log** function. Note that the name of the function starts with a capital letter and that the argument appears in **[square brackets]**:

```
In[6]:= Log[list2]
Out[6]= {Log[29], Log[16], Log[7], 1.19392, Log[6], 0, 1.12493, 3.10459}
```

Mathematica gives exact values. The exact value of the natural logarithm of 29 is Log[29]. An approximate numerical value is obtained using **N** in one of the following two forms:

```
In[7]:= N[Log[29]]
Out[7]= 3.3673
```

or

```
In[8]:= Log[29]//N
Out[8]= 3.3673
```

You can display a result to any number of decimal places. Here’s π.

```
In[9]:= π (* exact *)
Out[9]= π
```

```
In[10]:= N[π] (* approximate *)
Out[10]= 3.14159
```

```
In[11]:= N[π, 100] (* 100 decimal places *)
Out[11]= 3.14159265358979323846264338327950288419716
          93993751058209749445923078164062862089986
          28034825342117068
```

Other operations on lists

To add together all the elements in **list1**:

```
In[12]:= Plus @@ list1
Out[12]= 63.68
```

To multiply together all the elements in **list1**:

```
In[13]:= Times @@ list1
Out[13]= -3757.48
```

Editing Lists of Data

You can select points from a list of data using the commands **Drop**, **Take** and **Part** .

Drop the first 2 points from **list1**:

```
In[14]:= Drop[list1, 2]
Out[14]= {4, 0.3, 3, -2, 0.08, 19.3}
```

Drop the last 2 points:

```
In[15]:= Drop [list1, -2]
Out[15]= {26, 13, 4, 0.3, 3, -2}
```

Drop the second through fifth points:

```
In[16]:= Drop[list1, {2, 5}]
Out[16]= {26, -2, 0.08, 19.3}
```

Drop alternate points between the first and eighth points:

```
In[17]:= Drop[list1, {1, 8, 2}]
Out[17]= {13, 0.3, -2, 19.3}
```

Take is used similarly. For example, to keep the first two points:

```
In[18]:= Take[list1, 2]
Out[18]= {26, 13}
```

Part also lets you extract one or more data points from the list. It is used in almost the same way as **Take** and **Drop**. These three statements do the same thing:

```
In[19]:= Drop[list1, -3];
         Take[list1, 5];
         Part[list1, {1, 2, 3, 4, 5}];
```

Note also that

```
In[20]:= Part[list1, 3];
```

does the same as

```
In[21]:= list1[[3]];
```

Reading data from a laboratory experiment

Most of the data that you obtain in the laboratory will consists of pairs of (x,y) values, for example:

```
In[22]:= data = {{0, 6.62}, {1, 6.73}, {2, 6.86}, {3, 6.98}, {4, 7.03}};
```

One problem with this method of data entry is that is becomes laborious to type many curly brackets and commas, as well as increasing the possibility of making mistakes. An alternate method is to first create a data file using a text editor. A file which consists of two columns of x and y values might look like this, with a space between each column of numbers:

```
.5      8.1
1       9.2
1.5     10.5
2       13.1
2.5     15.4
3       18
3.5     20.4
4       22.9
4.5     24.5
5       26.3
```

Save the file under a meaningful name, such as “labdata.dat”. The “.dat” file extension tells you that this is a data file, as opposed any other kind of file, like text (.txt), a picture (.jpg, .gif, .bmp), or a program (.exe).

There are several methods for telling *Mathematica* how to read a set of data. The simplest of these is probably the **Import** command to read a data file. If the file is not already in your default working directory, you will need to use **SetDirectory** to make sure that *Mathematica* reads the file from the correct directory. For example (the exact syntax will depend on your operating system - Windows, Macintosh or Linux/Unix):

```
In[23]:= SetDirectory["c : \win98\desktop"];
```

Let's read in the data file "labdata.dat".

```
In[24]:= labdata = Import["labdata.dat"]
Out[24]= {{0.5, 8.1}, {1, 9.2}, {1.5, 10.5}, {2, 13.1},
          {2.5, 15.4}, {3, 18}, {3.5, 20.4},
          {4, 22.9}, {4.5, 24.5}, {5, 26.3}}
```

You can also use **ReadList.** This syntax tells *Mathematica* to read two columns of data. Use whichever form you like.

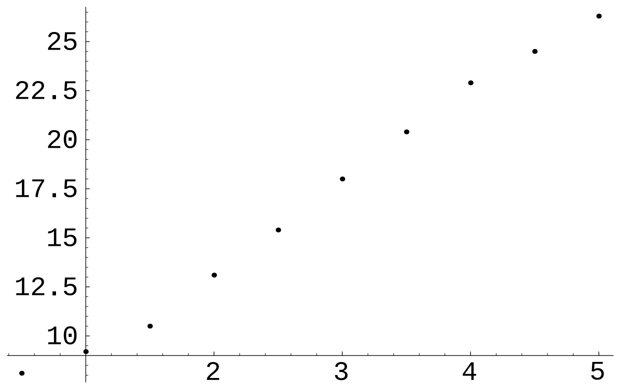
```
In[25]:= data = ReadList["labdata.dat", {Number, Number}]
Out[25]= {{0.5, 8.1}, {1, 9.2}, {1.5, 10.5}, {2, 13.1},
          {2.5, 15.4}, {3, 18}, {3.5, 20.4},
          {4, 22.9}, {4.5, 24.5}, {5, 26.3}}
```

See for yourself what happens when you use only one **Number** or omit the **Number**s completely

Simple Graphs, Fit and Regression

Plot the imported data:

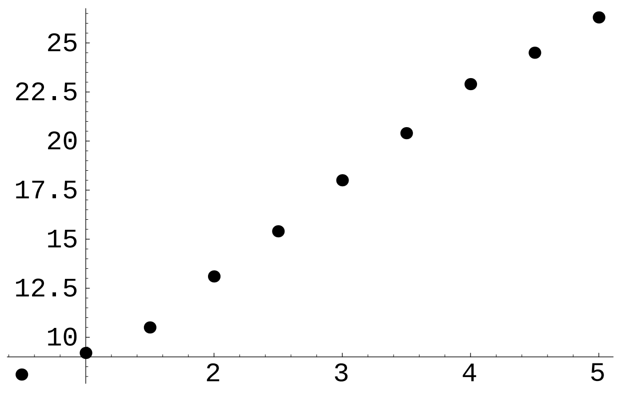
```
In[26]:= rawdata = ListPlot[labdata]
```



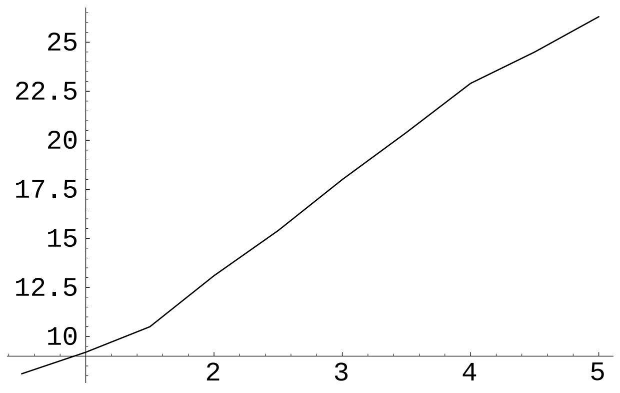
```
Out[26]= -Graphics-
```

ListPlot is probably the most convenient method for displaying raw data. You can add extra parameters if you like. Use whichever is most appropriate for your situation. For example:

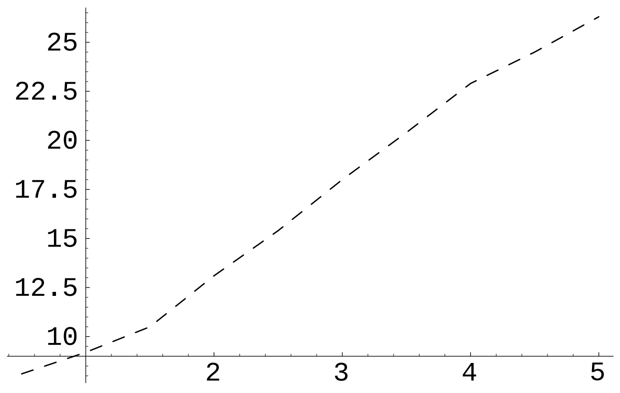
```
In[27]:= ListPlot[labdata, PlotStyle → PointSize[0.02]]
```



```
In[28]:= ListPlot[labdata, PlotJoined → True]
```



```
In[29]:= ListPlot[labdata, PlotJoined → True,  PlotStyle → Dashing[{0.02}]]
```



Fit these points to a straight line:

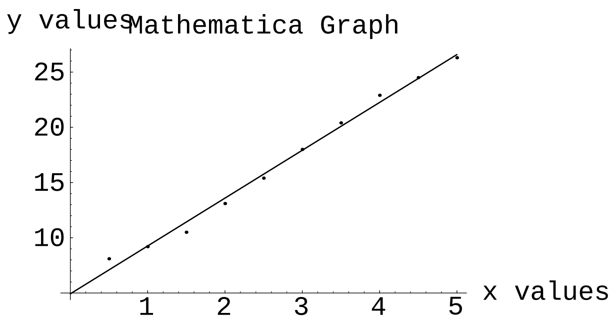
```
In[30]:= result = Fit[labdata, {1, x}, x]
Out[30]= 4.92667 + 4.33212 x
```

Obtain the line of best fit without plotting it (that's what **DisplayFunction** does):

```
In[31]:= bestline = Plot[result, {x, 0, 5}, DisplayFunction → Identity]
```

Plot the data and line of best fit on the same axes. Add a title and axis labels:

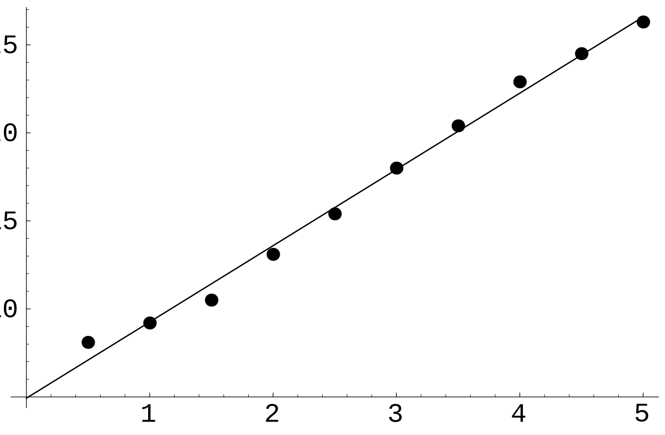
```
In[32]:= Show[rawdata, bestline, AxesLabel → {"x values", "y values"},
           PlotLabel → "Mathematica Graph"]
```



A Shortcut

The above examples have shown how to draw a graph of your laboratory data using the sequence of commands: **ListPlot**, **Fit** and **Show**. You can combine commands to produce a plot from a single line of input:

```
In[33]:= Plot[Fit[labdata, {1, x}, x], {x, 0, 5}, Epilog → {PointSize[0.02],
             Map[Point, labdata]}]
```



For a more detailed statistical analysis, including the errors in the slope and intercept, use the **Regress** command. You will need to load the **LinearRegression** package first.

```
In[34]:= <<Statistics‘LinearRegression‘

In[35]:= Regress[labdata, {1, x}, x]
                              Estimate   SE         TStat     PValue
Out[35]= {ParameterTable → 1  4.92667    0.407664   12.0851   2.03141 × 10^-6   , RSquared → 0.992694,
                           x  4.33212    0.131402   32.9685   7.81561 × 10^-10

          AdjustedRSquared → 0.99178,
          EstimatedVariance → 0.356121,

                              DF   SumOfSq   MeanSq     FRatio    PValue
                        Model 1    387.075   387.075    1086.92   7.81561 × 10^-10
          ANOVATable →  Error 8    2.84897   0.356121
                        Total 9    389.924
```

You don't need to display all these numbers if you don't want to. To display the parameters and their errors only, use

```
In[36]:= Regress[labdata, {1, x}, x, RegressionReport → ParameterCITable]
                                Estimate   SE         CI
Out[36]= {ParameterCITable → 1  4.92667    0.407664   {3.98659, 5.86674}
                             x  4.33212    0.131402   {4.02911, 4.63513}
```

You can fit data to any polynomial by including as many terms as you need inside the curly bracket. Thus to fit data to a quadratic, type

```
In[37]:= Fit[labdata, {1, x, x^2}, x]
Out[37]= 5.45167 + 3.80712 x + 0.0954545 x^2
```

The use of Fit and Regress is not limited to polynomials. You can fit data to any linear combination of the parameters that you specify, such as

```
In[38]:= Fit[labdata, {1, Sin[x], Cos[x]}, x]
Out[38]= 16.9071 - 2.70799 Cos[x] - 7.28491 Sin[x]
```

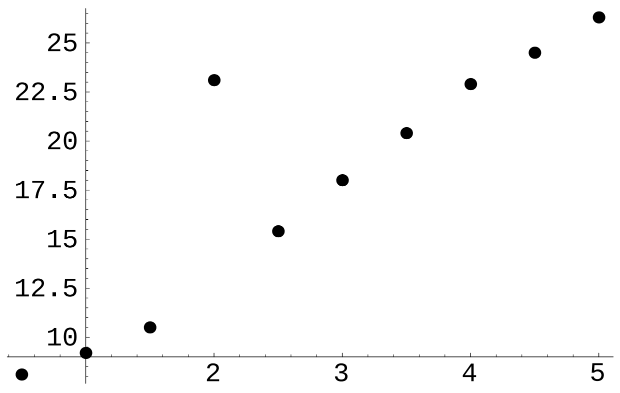
Removing erroneous data points

Before using **Fit** or **Regress,** you need to be sure that the data is correct. Look at the data below:

```
In[39]:= listr = {{0.5, 8.1}, {1, 9.2}, {1.5, 10.5}, {2, 23.1}, {2.5, 15.4}, {3, 18},
         {3.5, 20.4}, {4, 22.9}, {4.5, 24.5}, {5, 26.3}};
```

As usual, you can get a rough idea of what the graph looks like using **ListPlot**.To make it easier to see, we'll increase the size of the points using **PointSize**.

```
In[40]:= firstplot = ListPlot[listr, PlotStyle → PointSize[0.02]]
```



If you attempt to fit this data to a straight line, you will get a meaningless result because the fourth data point lies a long way from the line of best fit. You can remove this point from the fit using **Drop**:

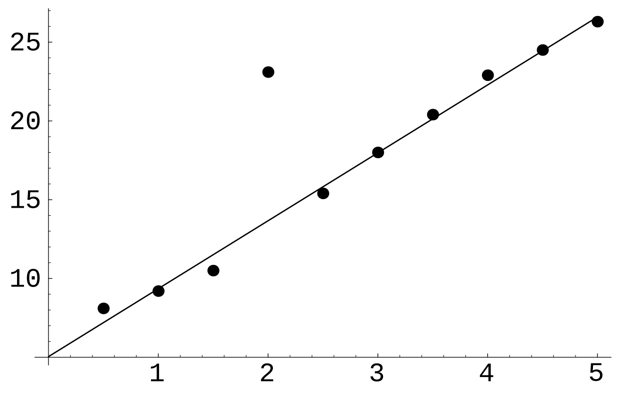
```
In[41]:= dropdata = Drop[listr, {4}]
Out[41]= {{0.5, 8.1}, {1, 9.2}, {1.5, 10.5},
          {2.5, 15.4}, {3, 18}, {3.5, 20.4},
          {4, 22.9}, {4.5, 24.5}, {5, 26.3}}
```

Now you can fit the data to a straight line:

```
In[42]:= corrfit = Fit[dropdata, {1, x}, x]
Out[42]= 5.03917 + 4.31167 x
```

Plot the data points (including the wrong one) and the corrected line of best fit:

```
In[43]:= corrplot = Plot[corrfit, {x, 0, 5}, DisplayFunction → Identity]

In[44]:= g = Show[firstplot, corrplot]
```



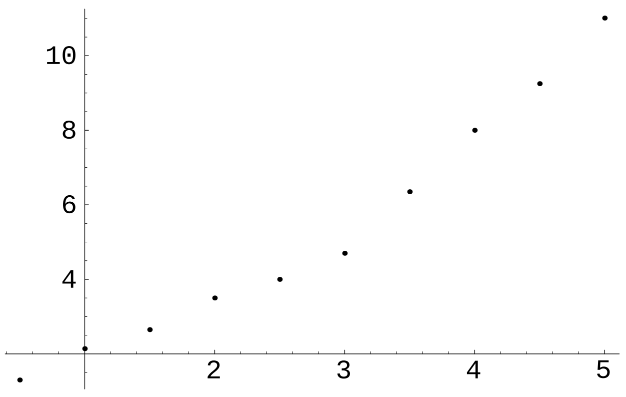
Obviously, if this was experimental data, you should go back and check the suspicious point before continuing.

Consider this data:

```
In[45]:= twoslpes = {{0.5, 1.3}, {1, 2.14}, {1.5, 2.65},
           {2, 3.5}, {2.5, 4}, {3, 4.7}, {3.5, 6.35},
           {4, 8}, {4.5, 9.25}, {5, 11.01}};
```

Now plot it:

```
In[46]:= rawdat = ListPlot[twoslpes]
```



This data does not follow a single straight line because the slope changes at larger values of x. We'll use Take and Drop to draw the lines of best fit for low and high x. First obtain the line of best fit for the low range data:

```
In[47]:= lofit = Fit[Take[twoslpes, 5], {1, x}, x]
Out[47]= 0.69 + 1.352 x
```

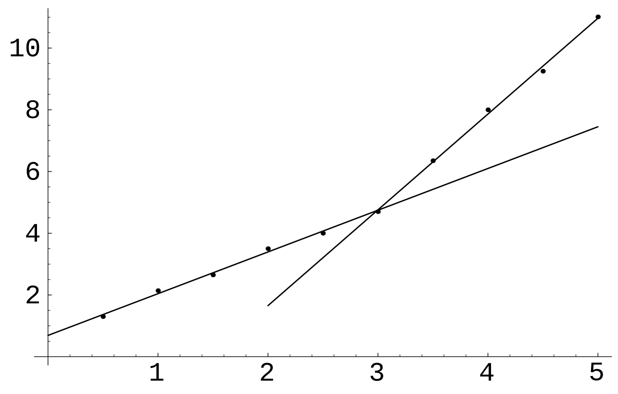
Now generate the plot of the line of best fit (but don't display it yet):

```
In[48]:= loline = Plot[lofit, {x, 0, 5}, DisplayFunction → Identity]
```

Similarly, for the high range data. (You can also combine commands):

```
In[49]:= hiline =  Plot[Fit[Drop[twoslpes, 5], {1, x}, x],
            {x, 2, 5}, DisplayFunction → Identity]
```

```
In[50]:= Show[rawdat, loline, hiline]
```



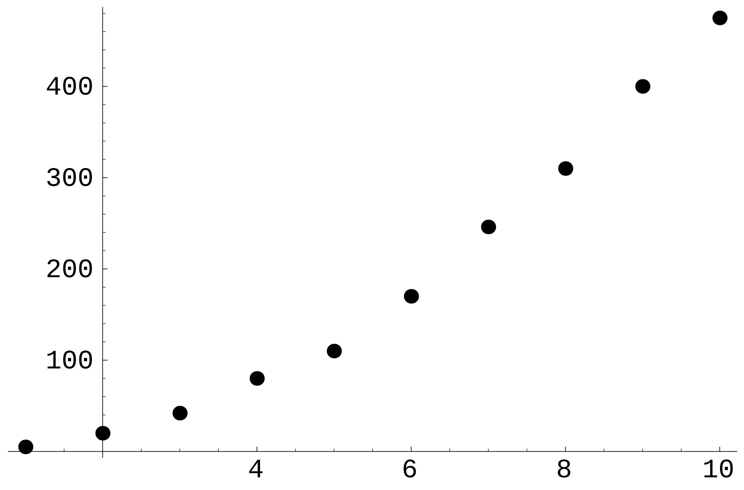
Example: Object in Free-Fall

Consider an object falling freely under gravity, where we measure the distance fallen at one-second intervals:

```
In[51]:= d = {{1, 5.}, {2, 20.}, {3, 42.}, {4, 80.},
            {5, 110.}, {6, 170.}, {7, 246.}, {8, 310.},
            {9, 400.}, {10, 475.}};
```

Plot it:

```
In[52]:= falldata = ListPlot[d, PlotStyle → PointSize[0.02]]
```



Use **Fit** to obtain the coefficient of x^2 only. It should be numerically equal to g/2 (i.e., 4.9 m/sˆ2)

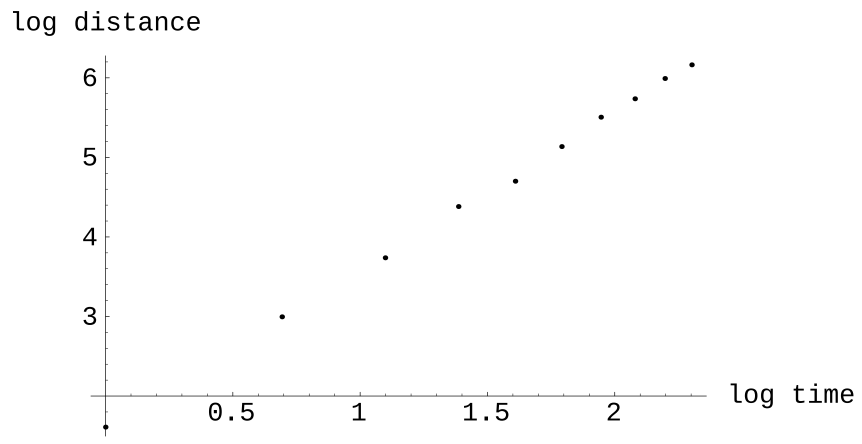
```
In[53]:= Fit[d, {xˆ2}, x]
Out[53]= 4.83192 x^2
```

To obtain the error in the acceleration we use **Regress.** Note how Regress includes the constant term by default.

```
In[54]:= Regress[d, {xˆ2}, x]
                                  Estimate    SE        TStat       PValue
Out[54]= {ParameterTable → 1      -0.551648   3.32996   -0.165662   0.872533            , RSquared → 0.998508,
                           x^2    4.8403      0.06616   73.1606     1.35714 × 10^-12

          AdjustedRSquared → 0.998321,

                                                        DF   SumOfSq    MeanSq     FRatio    PValue
          EstimatedVariance → 46.006, ANOVATable → Model  1    246246.    246246.    5352.47   1.35725 ×
                                                  Error  8    368.048    46.006
                                                  Total  9    246614.
```

We can also use the fact that data which follow a simple power law will appear as a straight line when the logarithm of each term is plotted. The slope of the line gives the expected power law.

```
In[55]:= ListPlot[Log[d],  AxesLabel- > {"log time", "log distance"}]
```



Fit Log[d] to a straight line:

```
In[56]:= logfit = Fit[Log[d], {1, x}, x]
Out[56]= 1.59544 + 1.9864 x
```

or better still,

```
In[57]:= Regress[Log[d], {1, x}, x]
                                    Estimate   SE          TStat     PValue
Out[57]= {ParameterTable → 1        1.59544    0.0331279   48.1602   3.82236 × 10^-11  , RSquared → 0.999196,
                           x        1.9864     0.0199225   99.7062   1.14353 × 10^-13

          AdjustedRSquared → 0.999095,
          EstimatedVariance → 0.00191941,

                             DF   SumOfSq     MeanSq       FRatio    PValue
          ANOVATable → Model  1    19.0815     19.0815      9941.32   1.14353 × 10^-13
                       Error  8    0.0153553   0.00191941
                       Total  9    19.0968
```

Thus the power law is 1.98 +/- 0.02 (as expected) and the intercept gives the natural logarithm of the acceleration. Hence

```
In[58]:= accln = Exp[logfit[[1]]]
Out[58]= 4.93052
```

Nonlinear Curve Fitting

If your data does not follow a straight line or simple polynomial, you will need to use *Mathematica*’s NonlinearFit functions:

```
In[59]:= << Statistics‘NonlinearFit‘
```

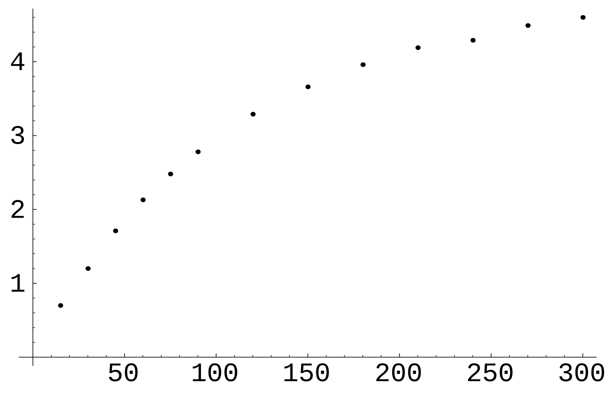
Example: Charging a Capacitor

Here we measure the voltage across the capacitor as a function of time. As usual, we'll read the data from a file:

```
In[60]:= chargedata = Import["capch.dat"]
Out[60]= {{15, 0.7}, {30, 1.2}, {45, 1.71}, {60, 2.13},
          {75, 2.48}, {90, 2.78}, {120, 3.29},
          {150, 3.66}, {180, 3.96}, {210, 4.19},
          {240, 4.29}, {270, 4.49}, {300, 4.6}}
```

Plot the data in the usual way:

```
In[61]:= cdata = ListPlot[chargedata]
```

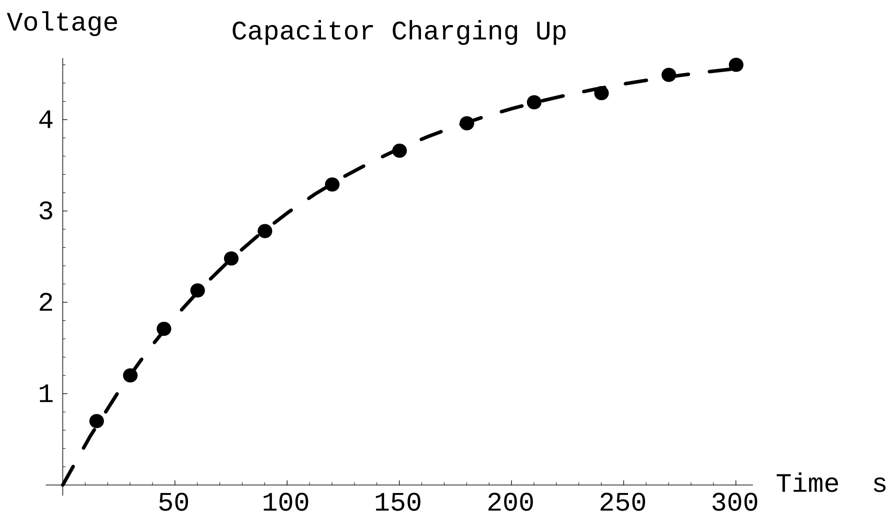


Define the function and ask Mathematica to solve for a and b:

```
In[62]:= chrgft = NonlinearFit[chargedata,
             a (1 - Exp[-x/b]), x, {a, b}]
```
Out[62]= $4.83133\,(1 - e^{-0.00957528\,x})$

Plot it, adding a few extra features:

```
In[63]:= Plot[chrgft, {x, 0, 300},
           AxesLabel- > {"Time (s)", "Voltage"},
           PlotLabel → "Capacitor Charging Up",
           PlotStyle →
             {{Dashing[{0.03}], Thickness[0.005]}},
           Epilog → {PointSize[0.02],
               Map[Point, chargedata]}]
```



The **NonLinearRegress** function gives output similar to **Regress.** You may not want to display all the information.

```
In[64]:= chrgft = NonlinearRegress[chargedata,
           a(1 - Exp[-x/b]), x, {a, b}]
Out[64]= {BestFitParameters →
           {a → 4.83133, b → 104.436},
                                Estimate   Asymptotic SE   CI
          ParameterCITable → a  4.83133    0.0275347       {4.77073, 4.89194} ,
                             b  104.436    1.43567         {101.276, 107.595}
          EstimatedVariance → 0.000897664,
                                             DF   SumOfSq     MeanSq
                      Model                  2    140.442     70.2208
          ANOVATable → Error                 11   0.0098743   0.000897664 ,
                      Uncorrected Total      13   140.451
                      Corrected Total        12   20.5537
          AsymptoticCorrelationMatrix →
            (    1.      0.896223 ), FitCurvatureTable →
            (0.896223       1.    )
                                        Curvature
            Max Intrinsic               0.00800664
            Max Parameter - Effects     0.0288556
            95. % Confidence Region     0.50111
```

Extracting the coefficients for use in future calculations requires the ’/.’ operator

```
In[65]:= values = BestFitParameters/.chrgft
Out[65]= {a → 4.83133, b → 104.436}
```

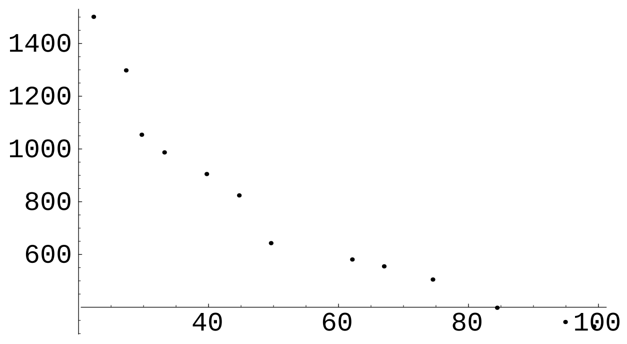
Apply the '/.' operator a second time:

```
In[66]:= a2 = a/.values
Out[66]= 4.83133

In[67]:= b2 = b/.values
Out[67]= 104.436
```

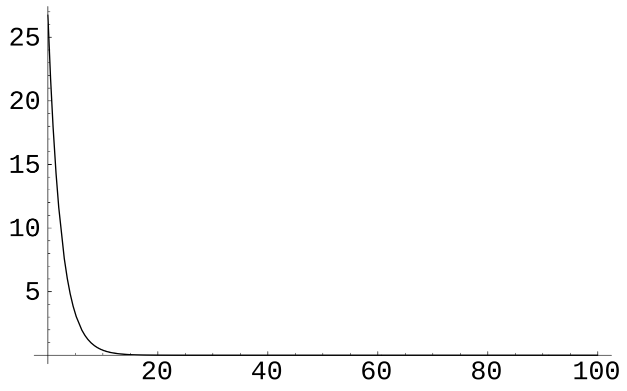
Example: Resistance of a Thermistor

This example shows that you need to be careful when using **NonlinearFit**. The resistance of a thermistor varies with temperature according to R=Aexp(-BT), where A and B are constants. Temperature is in degrees Celsius and resistance is in ohms. In this case we have entered temperature and resistance as two separate lists and used the **Table** command to combine them.

```
In[68]:= temp = {22.3, 27.3, 29.7, 33.2, 39.7, 44.7,
            49.6, 62.1, 67, 74.5, 84.4, 94.9, 99.3};

         ohms = {1501, 1298, 1054, 987, 905, 824,
            643, 581, 555, 505, 398, 344, 327, 257};

         tdata = Table[{temp[[i]], ohms[[i]]}, {i, 1, Length[temp]}];
In[69]:= thermplot = ListPlot[tdata]
```



```
In[70]:= fitonly = NonlinearFit[tdata, a Exp[-b x], x, {a, b}]
Out[70]= 26.7508 e^-0.421807 x

In[71]:= bestline = Plot[fitonly, {x, 0, 100}, PlotRange → All]
```

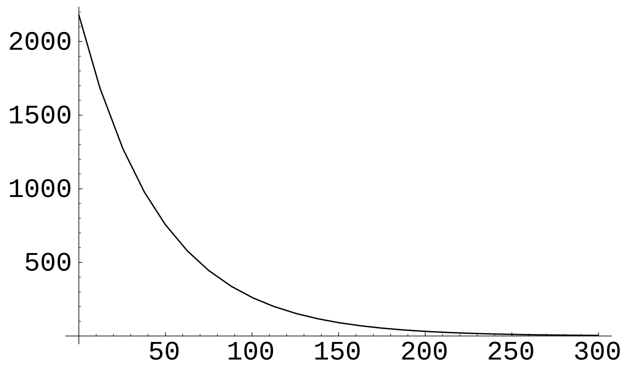


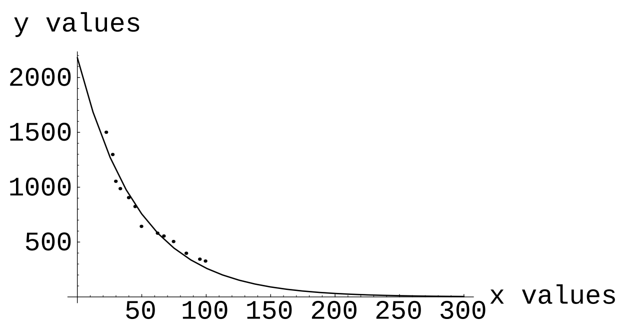
This fit is no good. We need to choose new starting values and increase the number of iterations.

```
In[72]:= result = NonlinearRegress[tdata, a Exp[-b x],
             x, {{a, 100}, {b, -0.05}},
             RegressionReport →
               {StartingParameters, BestFitParameters,
                 BestFit}, MaxIterations → 200]
Out[72]= {StartingParameters → {a → 100, b → -0.05},
           BestFitParameters →
             {a → 2180.84, b → 0.0211993},
           BestFit → 2180.84 e^(-0.0211993 x)}
```

Plot the data and line of best fit. Note how we extract the equation of the line using the '/.' operator.

```
In[73]:= Show[thermplot,
           Plot[BestFit/.result, {x, 0, 300}],
           AxesLabel → {"x values", "y values"}]
```





The values of a and b can be extracted from BestFitParameters for use in further calculations as follows:

```
In[74]:= aandb = BestFitParameters/.result
Out[74]= {a → 2180.84, b → 0.0211993}
```

```
In[75]:= aa = a/.aandb
Out[75]= 2180.84
```

```
In[76]:= bb = b/.aandb
Out[76]= 0.0211993
```

Extracting x and y values from data

You can separate the x-values from the y-values using the **Map** command. Since the x-values are contained in the first column, then for the data set labdata, we write

```
In[77]:= xvals = Map[First, labdata]
Out[77]= {0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5}
```

Similarly for the y-values,

```
In[78]:= yvals = Map[Last, labdata]
Out[78]= {8.1, 9.2, 10.5, 13.1,
           15.4, 18, 20.4, 22.9, 24.5, 26.3}
```

Changing x and y values

If you want to do the same operation on x and y values, the procedure for transforming the data is identical to that for a one-dimensional list. Thus, to take the natural logarithm of all the values, type:

```
In[79]:= logdata = Log[labdata]
```

```
Out[79]= {{-0.693147, 2.09186}, {0, 2.2192},
          {0.405465, 2.35138}, {Log[2], 2.57261},
          {0.916291, 2.73437}, {Log[3], Log[18]},
          {1.25276, 3.01553}, {Log[4], 3.13114},
          {1.50408, 3.19867}, {Log[5], 3.26957}}
```

More often, you will want to transform the x and y data separately. Perhaps one column of data will remain unchanged while the other is multiplied by a constant. Or you might take the reciprocal of one column, or you might want to swap the x and y values because you realised that you have plotted the wrong quantity along each axis of the graph.

```
In[80]:= new = labdata/.{x_, y_} → {x, 1/y}
Out[80]= {{0.5, 0.123457}, {1, 0.108696},
          {1.5, 0.0952381}, {2, 0.0763359},
          {2.5, 0.0649351}, {3, 1/18},
          {3.5, 0.0490196}, {4, 0.0436681},
          {4.5, 0.0408163}, {5, 0.0380228}}
```

Note the use of the replacement operator '/.' To swap the x and y data points, we write:

```
In[81]:= new1 = labdata/.{x_, y_} → {y, x}
Out[81]= {{8.1, 0.5}, {9.2, 1}, {10.5, 1.5}, {13.1, 2},
          {15.4, 2.5}, {18, 3}, {20.4, 3.5},
          {22.9, 4}, {24.5, 4.5}, {26.3, 5}}
```

Another method uses the & /@ operators. The following will take the reciprocal of the y- values while leaving the x-values unchanged:

```
In[82]:= datanew = {#[[1]], 1/#[[2]]}&/@ labdata;
```

Think #[[1]] as meaning "the first column of the data" and #[[2]] as "the second column of the data". Swapping the x and y columns is very easy: you just swap the #[[1]] and #[[2]] columns:

```
In[83]:= dataswap = {#[[2]], #[[1]]}& /@labdata;
```

Do not attempt to transform your data or fit your data to a straight line or curve without first knowing what you are trying to achieve. The *Mathematica* software is very powerful, but it will give meaningless numbers if you cannot assess the physical significance of your results.

Histograms

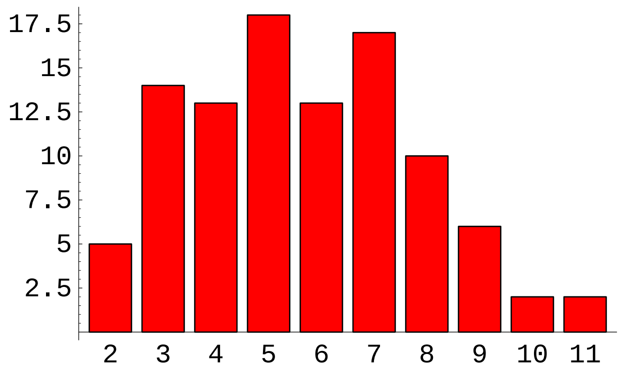
To generate a histogram, you need to load the standard Mathematica package to set up additional graphics functions

```
In[84]:= << Graphics'
```

Now we'll import some raw data from the "Statistics of Nuclear Counting" experiment and use the **Frequencies** command to draw the histogram.

```
In[85]:= bardat = Import["stats.dat"];
```

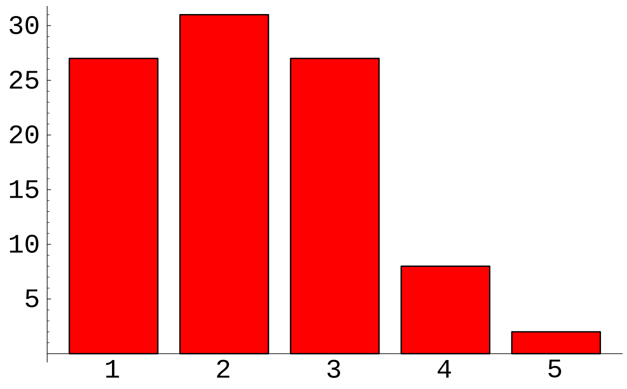
```
In[86]:= BarChart[Frequencies[bardat]]
```



BinCounts[xdat, {xmin, xmax, dx}] lists the number of elements in *xdat* that lie in bins from *xmin* to *xmax* in steps of *dx*. Increasing the bin width to 2 produces the following histogram:

```
In[87]:= abc = BinCounts[bardat, {2, 11, 2}];
```

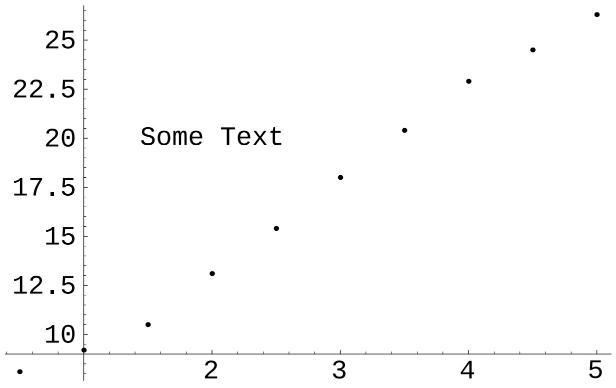
```
In[88]:= BarChart[abc]
```



Text and Legends

You can insert text anywhere on a graph using a combination of **Show**, **Graphics** and **Text**. For example, to put the words "Some Text" (centered at {2, 20}) on the graph which we called *rawdata*, above, type:

```
In[89]:= Show[rawdata, Graphics[Text["Some Text", {2, 20}]]]
```

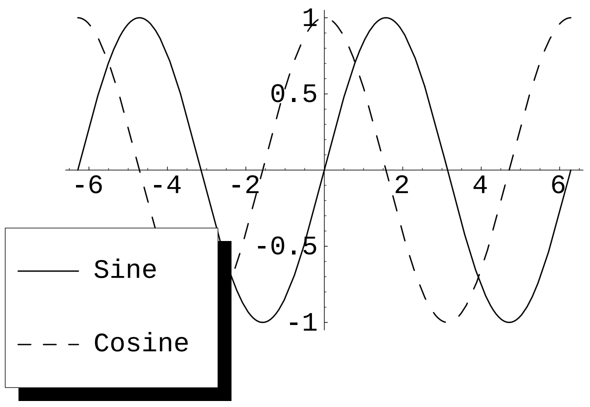


You can also change the orientation and font of the text. See the *Mathematica* Help menu for further details. Legends can be included by loading the appropriate package:

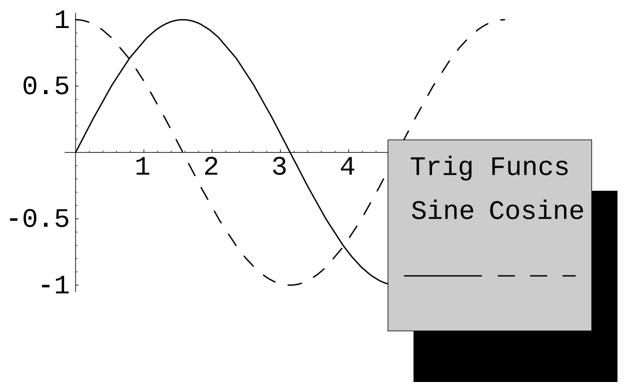
```
In[90]:= << Graphics‘Legend‘
```

You can place a legend in a graph as an option with **Plot.** Simply specify the text for each curve. For example,

```
In[91]:= Plot[Sin[x], Cos[x], x, -2Pi, 2Pi, PlotStyle- > GrayLevel[0], Dashing[.03], PlotLegend- >
"Sine", "Cosine"]
```



You can include more options to change the appearance of the legend, for example,



Saving your Graph

Usually, you will be want to print out the entire *Mathematica* worksheet that you create. This is so that you can show how you performed calculations and obtained the line of best fit to a graph. Sometimes you will want to print out a graph or image separately to in a formal report, for example. *Mathematica* supports many graphics formats, including encapsulated postscript (.eps), Adobe Acrobat portable document formt (.pdf), GIF (.gif) and JPEG (.jpg). Suppose you want to save the graph which we called ”rawdata”. This is achieved using the ”**Display**” command. Decide on a name for the graph file (say, “myfile”) and the format you want to save it in. Thus you would type one of the following commands to save the file in your default directory.

```
In[93]:= Display["myfile.eps", rawdata, "EPS"];
         Display["myfile.pdf", rawdata, "PDF"];
         Display["myfile.gif", rawdata, "GIF"];
         Display["myfile.jpg", rawdata, "JPEG"];
```