Least Squares Fitting:
Physics 338

In Chapter 8 of *An Introduction to Error Analysis*, by John R. Taylor, the author discusses the general theme of least-squares fitting. This is based on the normal distribution discussion in Chapter 5. Here, we will take the theoretical background as given, and show how the minimization of the least square difference leads to the standard results for some simple cases.

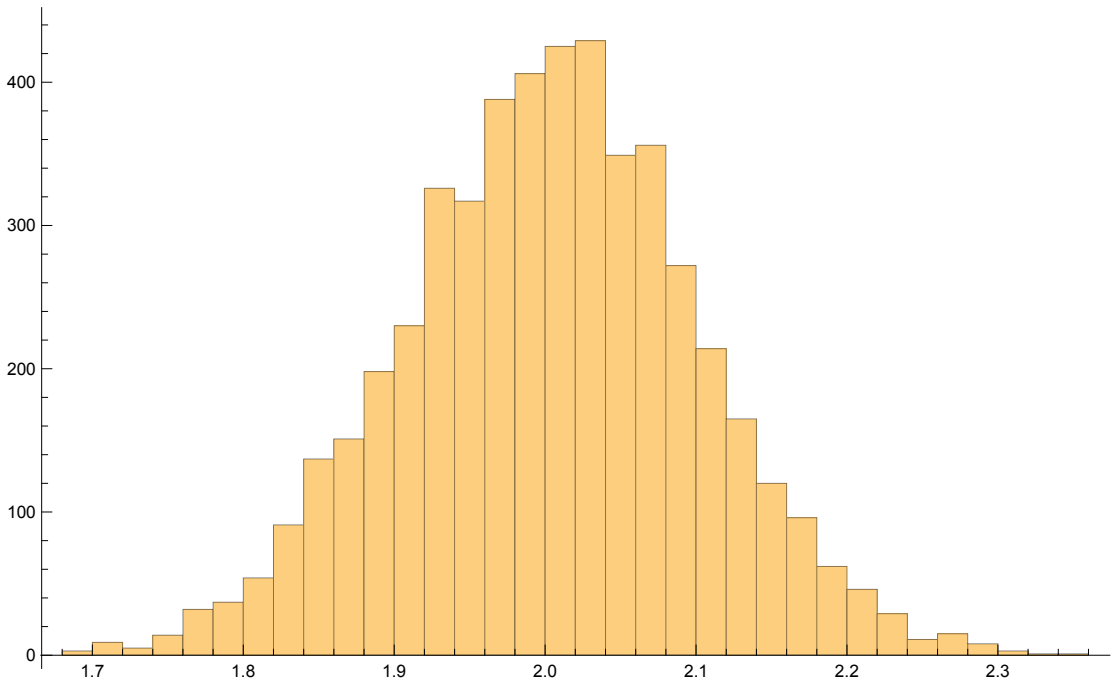
Computing the Mean

As a simple application, we will use the principle of least squares to calculate the mean of a set of numbers.

First, we will make up a data set of random numbers. The following command creates a list of 5000 numbers selected from a normal distribution with a mean of 2.0 and a standard deviation of 0.1

In[58]:=
```
rdata = RandomReal[NormalDistribution[2, 0.1], 5000];
npts = Length[rdata];
Histogram[rdata, ImageSize → Large]
```

Out[60]=



Computing χ^2

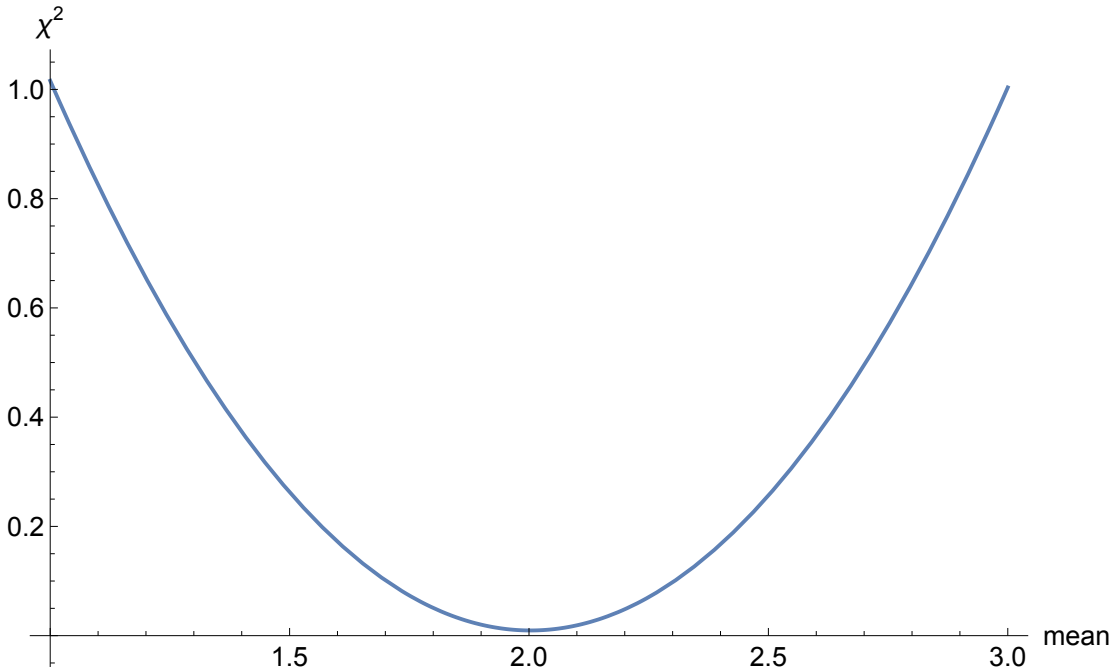
Next, we calculate χ^2. For N data points with a mean value of X, (and assuming all have the same uncertainty) the formula is

In[61]:=
```
chisq1[data_, X_] := 1/(npts - 1) Sum[(data[[i]] - X)^2, {i, 1, npts}]
```

Of course, we don't know X yet. The purpose of this exercise is to find the value of X which minimizes χ^2. We claim that value of X is our best estimate of the mean. Here is a plot showing χ^2 for various guesses of X.

In[62]:=
```
Plot[chisq1[rdata, X], {X, 1, 3},
 AxesLabel → {"mean", "χ²"}, LabelStyle → Larger, ImageSize → Large]
```

Out[62]=



In[63]:=
```
FindMinimum[chisq1[rdata, X], X]
```

Out[63]=
```
{0.00952305, {X → 2.00308}}
```

This tells us the value of X that minimizes χ^2, and also what that minimum value is.

In[64]:=
```
Sqrt[%[[1]]] (* This will be the standard deviation *)
```

Out[64]=
```
0.0975861
```

Mean and Standard Deviation

The usual definition for the mean results from minimizing χ^2, and that minimum value is the same as the usual definition for the standard deviation.

In[65]:=
```
Mean[rdata]
```
Out[65]=
```
2.00308
```

In[66]:=
```
StandardDeviation[rdata]
```
Out[66]=
```
0.0975861
```

Fitting a Straight Line

First, here is some sample data of {current, mass} pairs from the magnetic susceptibility experiment.

Data

Typing the data directly:

The data is a list of data points. All lists are made of elements in curly braces { } separated by commas. Each data point is itself a list, consisting of two numbers (in curly braces) separated by a comma. To see what it looks like, display it with TableForm (or MatrixForm).

In[67]:=
```
rawdata = {
    {0.000, 181.184},
    {0.497, 180.964},
    {1.005, 180.736},
    {1.507, 180.514},
    {1.930, 180.326},
    {0.000, 181.185},
    {-0.504, 181.409},
    {-1.008, 181.633},
    {-1.502, 181.853},
    {-1.930, 182.045},
    {0.000, 181.184}
};
```

In[68]:=
```
npts = Length[rawdata]
```
Out[68]=
```
11
```

Typing the data using the Classroom Assistant:

From the Palettes -> Classroom Assistant palette, look for the matrix item: $\begin{pmatrix} \square & \square \\ \square & \square \end{pmatrix}$. (Depending on the version of *Mathematica*, it might be in the “Typesetting” menu, or in the “Advanced” menu. (Use Ctrl-Enter to add a new row.)

In[69]:=
```
rawdata = {{0, 181.184}, {0.497, 180.964}, {1.005, 180.736}, {1.507, 180.514}, {1.930, 180.326}, {0, 181.185}, {-0.504, 181.409}, {-1.008, 181.633}, {-1.502, 181.853}, {-1.930, 182.045}, {0, 181.184}};
```

Reading from a file:

To read the data from a text file in the same directory as your notebook, first tell *Mathematica* what directory to use, and then tell it This particular file consists of two columns (current, period). Read it in as a "Table". If you store your data in the same directory as the Mathematica notebook, then it is probably simplest to just set the notebook directory as the default.

In[70]:=
```
SetDirectory[NotebookDirectory[]]
```

Out[70]=
```
/Users/doughera/notes/www/public_html/courses/phys338-2024f/notes/curve-fit
```

Typically, files might have header information (names of columns, dates, etc.) Often they are delimited by some sort of comment symbol. I often use '#' for that purpose when I type the files in by hand. Here is what my raw data file looks like:

In[71]:=
```
FilePrint["mass_vs_current.txt"]
```

```
# Mass reading (in g) vs Current (in Amps) for Magnetic Susceptibility
# Experiment.  January 26, 2018.   -- A. Dougherty
# Zero current obtained by pulling plug.  Leads were reversed
# for negative current.  Zero was checked at the beginning, middle, and end.
# Current (A)  Mass (g)
 0.000 181.184
 0.497 180.964
 1.005 180.736
 1.507 180.514
 1.930 180.326
 0.000 181.185
-0.504 181.409
-1.008 181.633
-1.502 181.853
-1.930 182.045
 0.000 181.184
```

In[72]:=
```
rawdata = Import["mass_vs_current.txt", "Table"]
```

Out[72]=
```
{{#, Mass, reading, (in, g), vs, Current, (in, Amps), for, Magnetic, Susceptibility},
 {#, Experiment., January, 26,, 2018., --, A., Dougherty},
 {#, Zero, current, obtained, by, pulling, plug., Leads, were, reversed}, {#, for,
  negative, current., Zero, was, checked, at, the, beginning,, middle,, and, end.},
 {#, Current, (A), Mass, (g)}, {0., 181.184}, {0.497, 180.964}, {1.005, 180.736},
 {1.507, 180.514}, {1.93, 180.326}, {0., 181.185}, {-0.504, 181.409},
 {-1.008, 181.633}, {-1.502, 181.853}, {-1.93, 182.045}, {0., 181.184}}
```

The default import includes all those comment lines. We want to select only those lines that contain the actual data. The Selectp[comnmand is handy for this. In this specific case, we want only lines where there are exactly two elements, and both of them are numeric. The following does that. The first test, Length[#] == 2 checks that there are 2 elements. The second test checks that the element is a vector (i.e. a list) that consists of only numeric items. You use '&&' to mean logical "AND". You would use '||' to mean logical OR. The '&' at the end is needed to make Mathematica set the '#' to each individual item in the list.

In[73]:=
```
data = Select[rawdata, Length[#] == 2 && VectorQ[#, NumericQ] &]
```

Out[73]=
```
{{0., 181.184}, {0.497, 180.964}, {1.005, 180.736},
 {1.507, 180.514}, {1.93, 180.326}, {0., 181.185}, {-0.504, 181.409},
 {-1.008, 181.633}, {-1.502, 181.853}, {-1.93, 182.045}, {0., 181.184}}
```

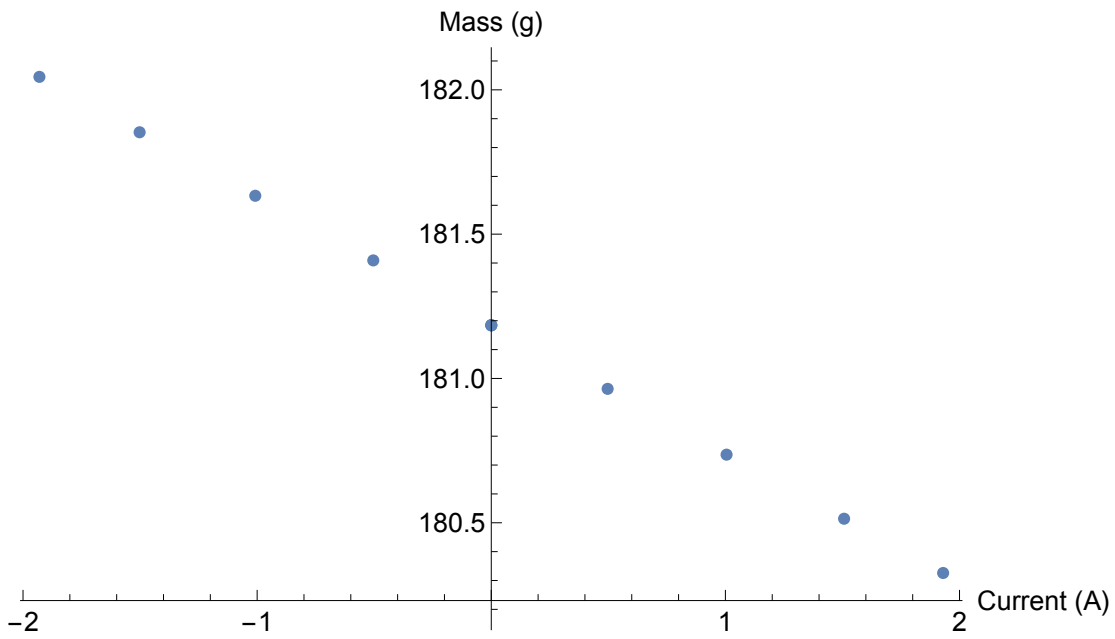
Mathematica also can import CSV (comma-separated-values) files. More information is in the Import[ ] function.

Plot the data.

Be sure to give your axes meaningful labels and make them large enough to be read. I have assigned it to a variable for later reuse.

In[74]:=
```
dataplot = ListPlot[data, AxesLabel → {"Current (A)", "Mass (g)"},
  LabelStyle → Larger, ImageSize → Large]
```

Out[74]=



Theoretical Curve: A straight-line fit.

Here we explore how to find the best-fit straight line by the method of least squares. First, define the target function. Use the unknown items (intercept, a0, and slope, a1) as parameters to the function.

In[75]:=
```
yfit[x_, a0_, a1_] := a0 + a1 x
```

Calculating χ^2

Define a χ^2 function for the linear fit. For given a0 and a1 values, compute the average of the differences squared between the data and the fit value. Within the function, the data is in {i, m} pairs, so we pull out the current of the i^{th} value with data[[i, 1]], and the mass of the i^{th} value with data[[i, 2]]. You can use the Classroom Assistant Palette to format the sum, or you can use *Mathematica*'s Sum[ ] function directly. Use whichever is easier for you to read. The denominator has the '-2' because with N data points and 2 free parameters, there are only N-2 degrees of freedom.

In[76]:=
```
calculateChisq[data_, a0_, a1_] :=
  1/(Length[data] - 2) Sum_{i=1}^{Length[data]} (yfit[data[[i, 1]], a0, a1] - data[[i, 2]])^2
```

In[77]:=
```
calculateChisq[data_, a0_, a1_] :=
 Sum[(yfit[data[[i, 1]], a0, a1] - data[[i, 2]])^2, {i, 1, Length[data]}] /
  (Length[data] - 2)
```

Initial Explorations

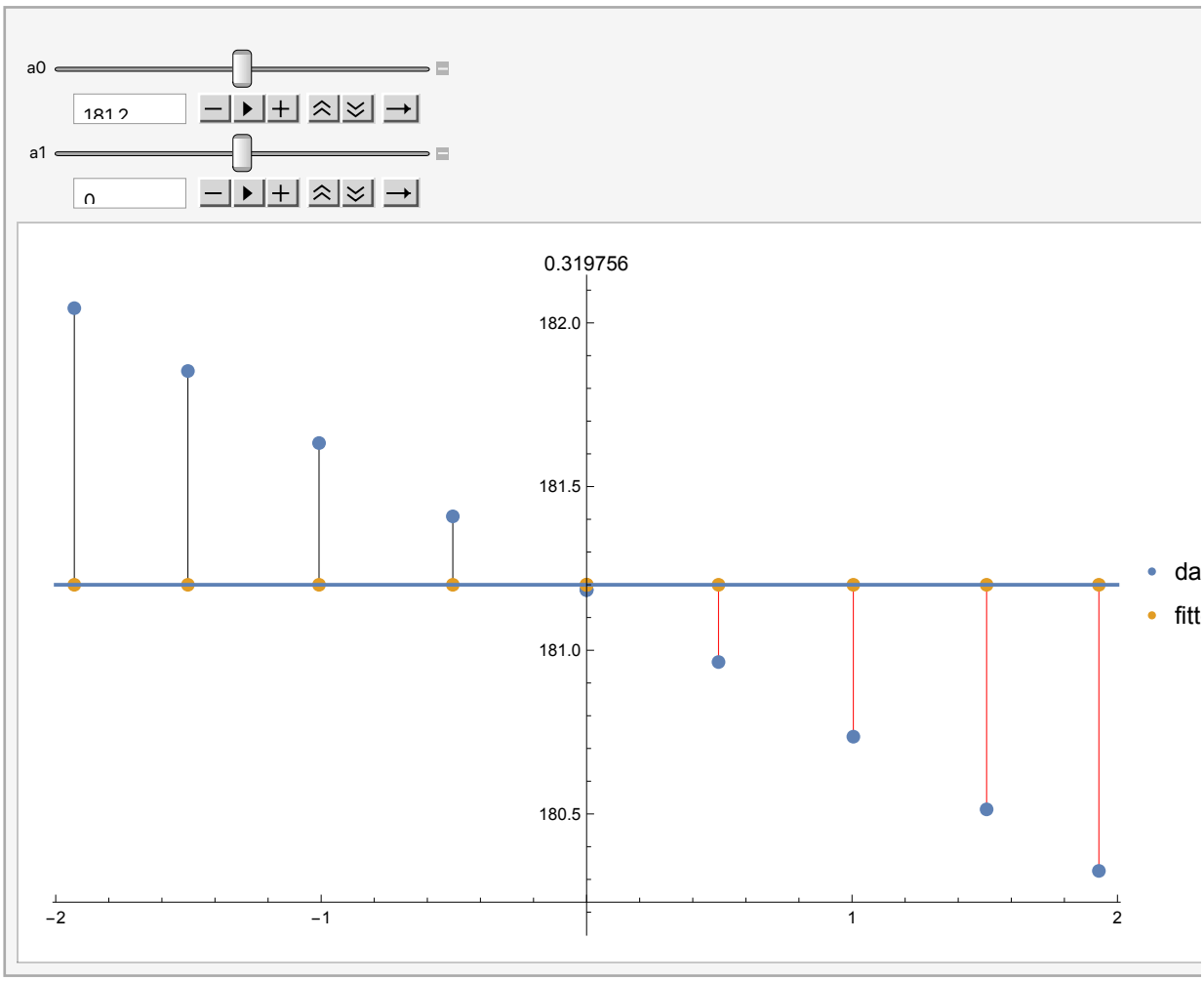
This command builds an interactive window showing the data, the current fit, and the chi squared value (as the plot title). It draws vertical lines from each data point to the fit line. Move the a0 and a1 sliders to minimize χ^2. Mathematica's "Filling" option gets confused if the data isn't sorted, so let's go ahead and sort the data by the first entry in each line.

In[78]:=
```
data = SortBy[data, First]
```

Out[78]=
```
{{-1.93, 182.045}, {-1.502, 181.853}, {-1.008, 181.633},
 {-0.504, 181.409}, {0., 181.184}, {0., 181.184}, {0., 181.185},
 {0.497, 180.964}, {1.005, 180.736}, {1.507, 180.514}, {1.93, 180.326}}
```

In[79]:=
```
Manipulate[
 fitted = Table[{data[[i, 1]], yfit[data[[i, 1]], a0, a1]}, {i, 1, Length[data]}];
 Show[{ListPlot[{data, fitted },
    Filling → {1 → {{2}, {Red, Black}}}, PlotLegends → {"data", "fitted"}],
   Plot[yfit[x, a0, a1], {x, -2, 2}]},
  PlotLabel → calculateChisq[data, a0, a1], ImageSize → Large] ,
 {{a0, 181.2}, 180.7, 181.7, 0.001,  Appearance → "Open"},
 {{a1, 0}, -1.00, 1.00, 0.02, Appearance → "Open"}
]
```

Out[79]=

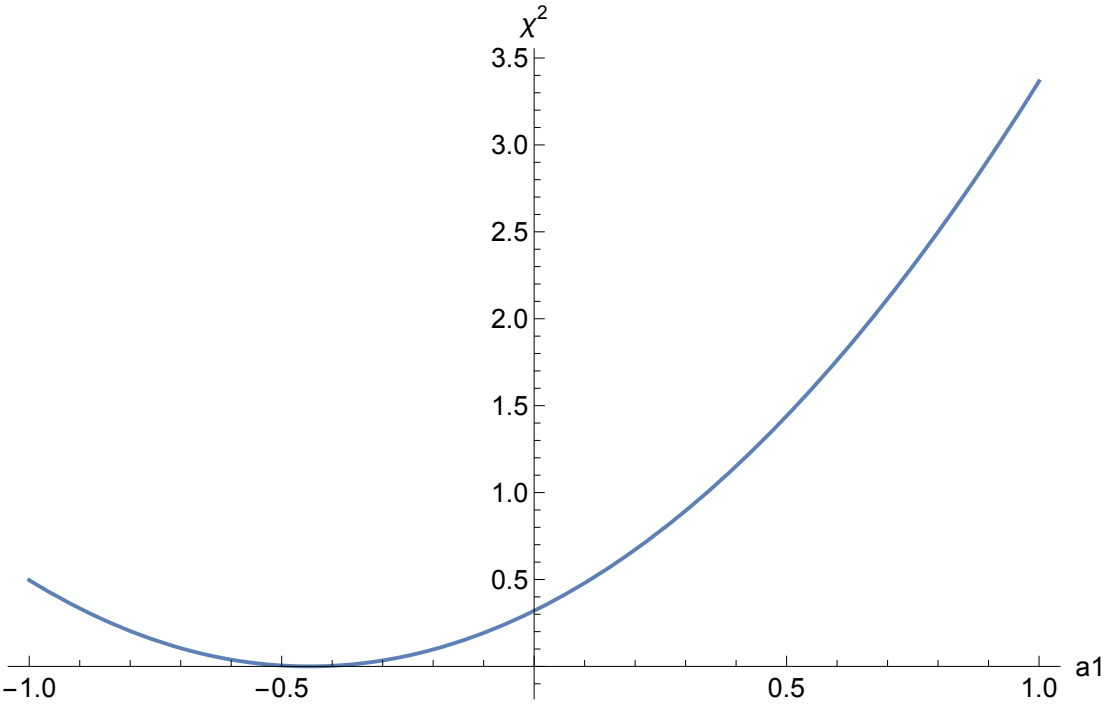


Minimizing χ^2

Picking the intercept a0 = 181.2 (close to the expected value), look at how χ^2 varies with a1.

In[80]:=
```
Plot[calculateChisq[data, 181.2, a1],
 {a1, -1, 1}, AxesLabel → {"a1", "χ²"}, LabelStyle → Larger]
```

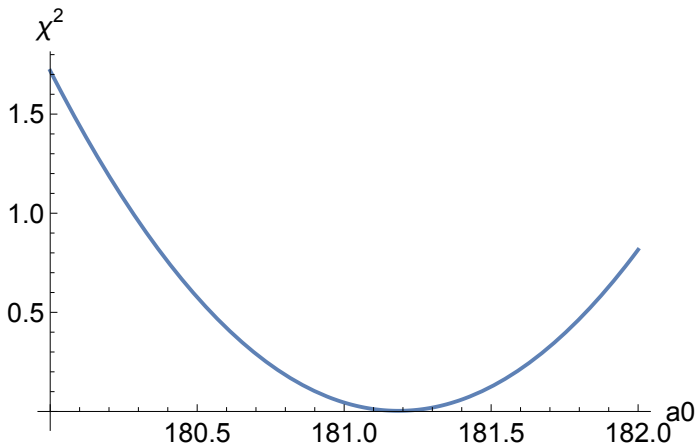
Out[80]=



Picking a1 = -0.4 (again, close to the expected value), look at how χ^2 varies with a0.

In[81]:=
```
Plot[calculateChisq[data, a0, -0.4], {a0, 180, 182},
 AxesLabel → {"a0", "χ²"}, LabelStyle → Larger]
```

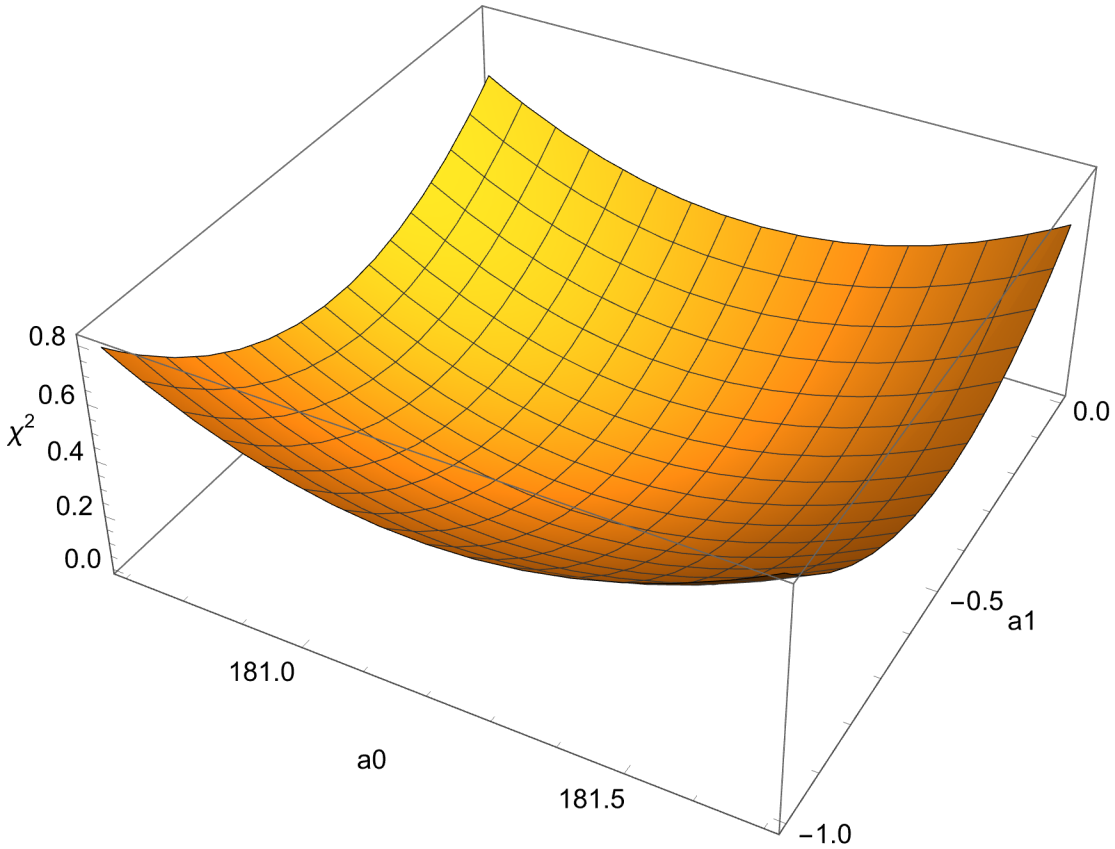
Out[81]=



Ultimately, the “best” fit involves a two-dimensional minimization of χ^2. Sometimes it helps to visualize this sort of thing as a 3D plot or a density plot.

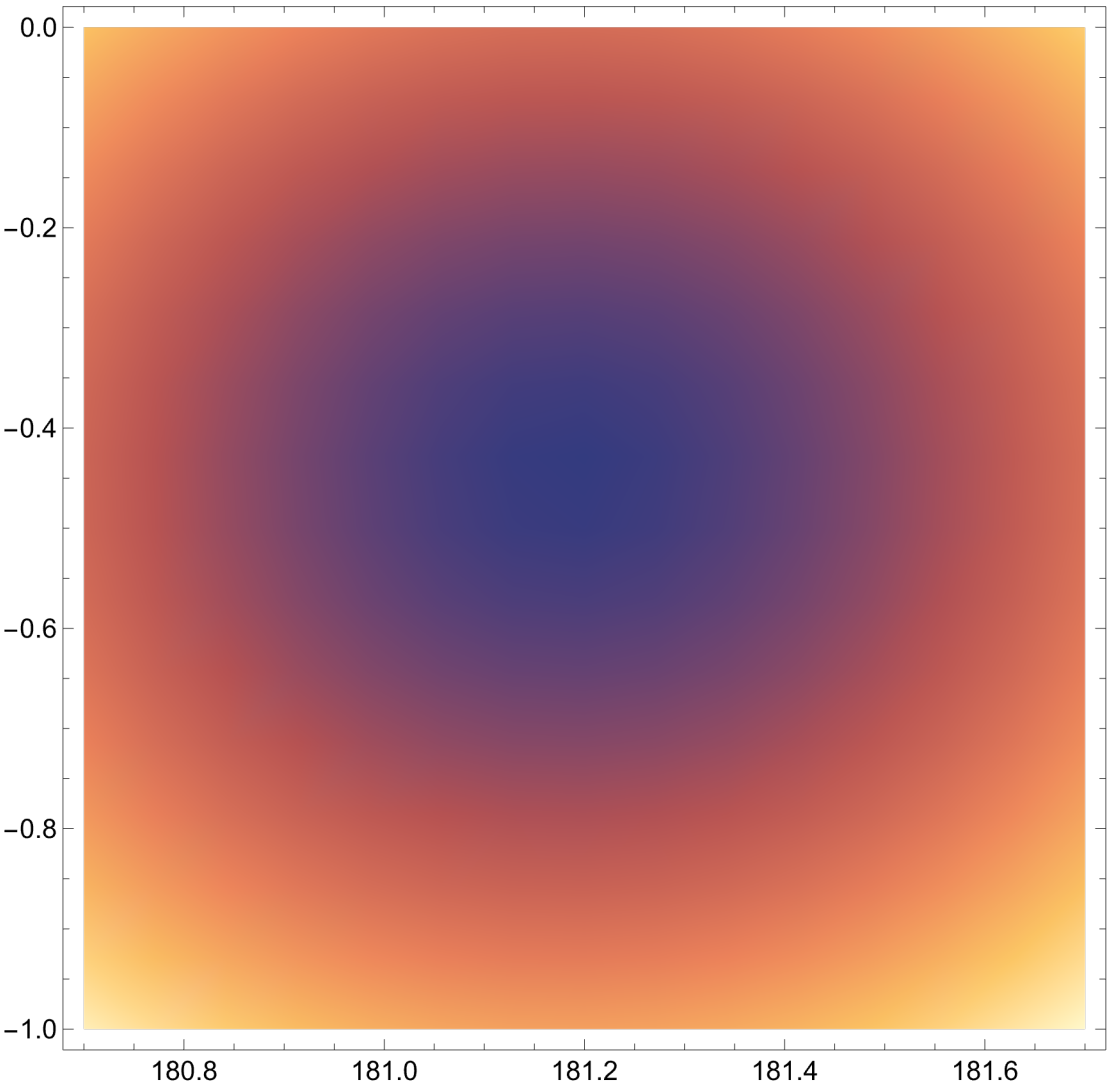
In[82]:=
```
Plot3D[calculateChisq[data, a0, a1], {a0, 180.7, 181.7}, {a1, -1, 0},
 AxesLabel → {"a0", "a1", "χ²"}, LabelStyle → Larger, ImageSize → Large]
```

Out[82]=



In[83]:=
```
DensityPlot[calculateChisq[data, a0, a1], {a0, 180.7, 181.7},
 {a1, -1, 0}, LabelStyle → Larger, ImageSize → Large]
```

Out[83]=



Mathematica can find the minimum here as well.

In[84]:=
```
result = FindMinimum[calculateChisq[data, a0, a1 ], {a0, a1}]
```

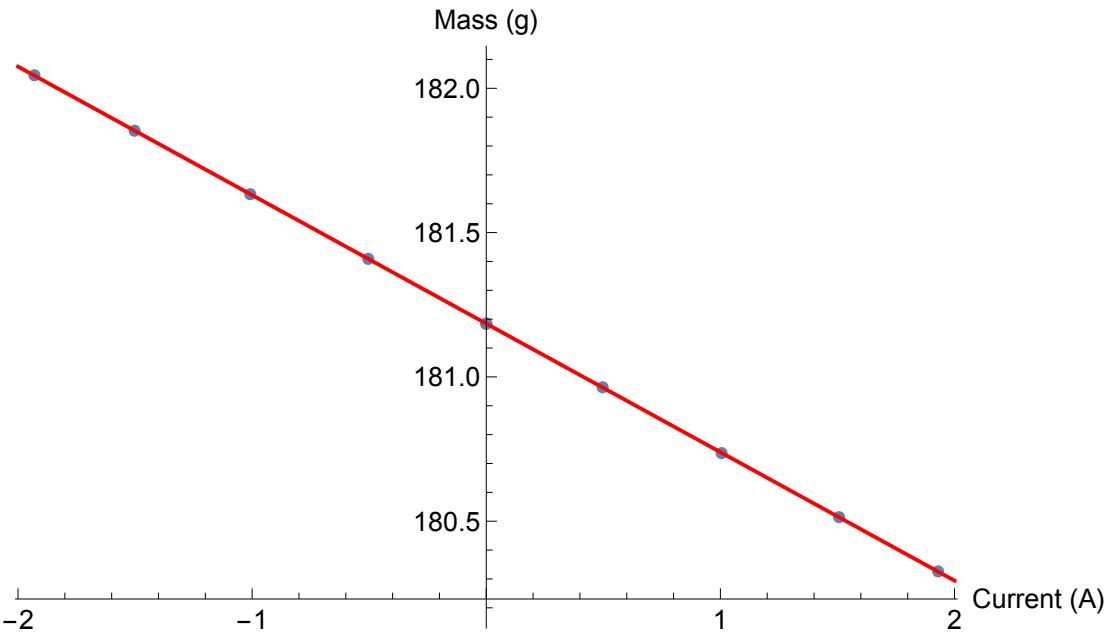
Out[84]=

$$\{5.27447 \times 10^{-7}, \{181.185 \to 181.185\}\}$$

Finally, here is the best fit curve.

In[85]:=
```
Show[{
  dataplot,
  Plot[yfit[x, a0, a1] /. result[[2]], {x, -2, 2}, PlotStyle → Red]
 }]
```

Out[85]=



Linear Model Fit

Mathematica can do this minimization automatically. The LinearModelFit[ ] function searches the "parameter space" for the minimum value of χ^2. It reports a "Confidence Interval" that reflects the curvature of χ^2 -- how much you can vary either parameter without making χ^2 too large.

One other interesting thing to note: Note how our χ^2 space has a "valley" (dark purple in the DensityPlot) where you can explore what happens if you change a0 but don't change a1, and vice-versa. This is reflected in the LinearModelFit[ ] report. Look at the "Correlation Matrix." It tells you, in essence, how well correlated each parameter is with the other. The 1's along the diagonal mean a0 is perfectly correlated with a0 (obviously) and a1 is perfectly correlated with a1. The off-diagonal elements tell you that a0 and a1 are mostly uncorrelated -- increasing one doesn't really affect the other. This isn't always true for fits. Sometimes, a change in one parameter can be partially compensated for by a change in another. In those cases, they are not completely independent parameters, and the Correlation Matrix elements are further from zero.

In[86]:=
```
fit = LinearModelFit[data, x, x]
```

Out[86]=

FittedModel[181. − 0.445 x]

See the on-line help for more information on dealing with the results from LinearModelFit. Here are some examples of things you can do with it.

In[87]:=
```
fit["BestFit"]
```

Out[87]=
$$181.185 - 0.445241\, x$$

In[88]:=
```
fit["BestFitParameters"]
```

Out[88]=
```
{181.185, -0.445241}
```

In[89]:=
```
fit["ParameterConfidenceIntervalTable"]
```

Out[89]=

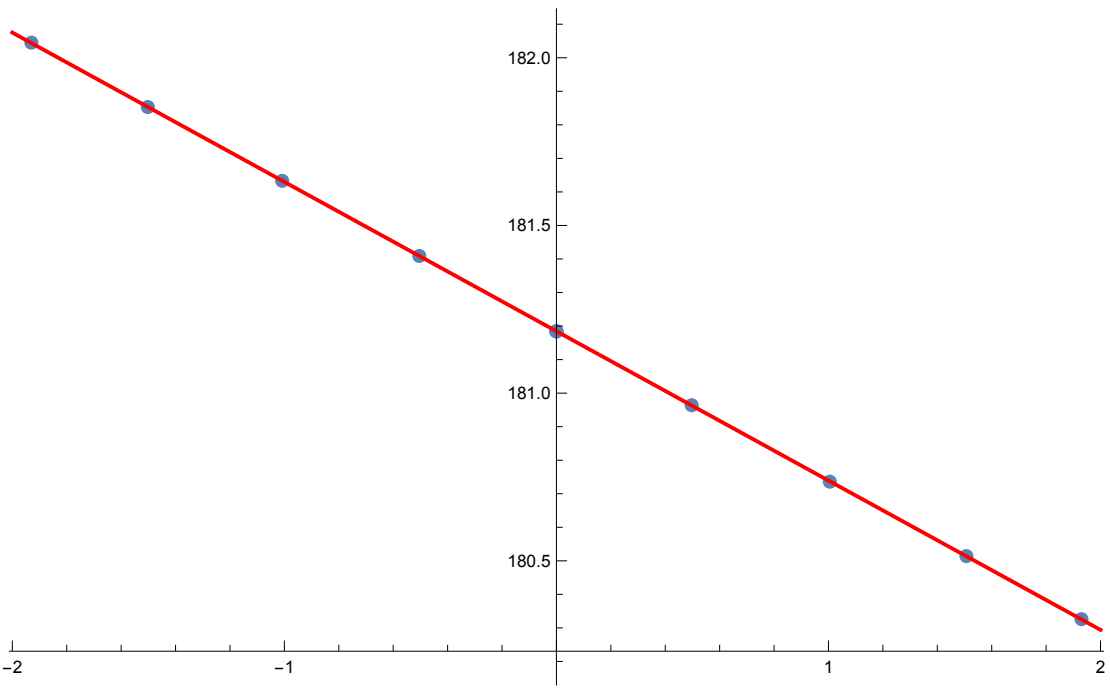
	Estimate	Standard Error	Confidence Interval
1	181.185	0.000218974	{181.184, 181.185 }
x	−0.445241	0.000190698	{−0.445673 , −0.44481 }

In[90]:=
```
fit["CorrelationMatrix"] // MatrixForm
```

Out[90]//MatrixForm=
$$\begin{pmatrix} 1. & 0.00039585 \\ 0.00039585 & 1. \end{pmatrix}$$

In[91]:=
```
Show[{ListPlot[data], Plot[fit[x] , {x, -2, 2},
   PlotStyle → Red, LabelStyle → Larger]}, ImageSize → Large]
```

Out[91]=



In[92]:=
```
fit["EstimatedVariance"]
```

Out[92]=
$$5.27447 \times 10^{-7}$$

This is the same as our χ^2.

In[93]:=
```
{a0, a1} = fit["BestFitParameters"]
```

Out[93]=
```
{181.185, -0.445241}
```

In[94]:=
```
calculateChisq[data, a0, a1]
```

Out[94]=
```
5.27447 × 10^-7
```

Interpreting the Uncertainties.

Mathematica gives you the uncertainties in the parameters, but you should also check whether the fit is reasonable. In particular, do the differences between the data points and the fit make sense? Is the size believable? Are there systematic trends?

Mathematica will report the "FitResiduals", which are the differences between each data point and the fitted curve. The average square of the residuals is called the "EstimatedVariance". (Actually you divide by N-2, since there are 2 degrees of freedom used up by the two fit parameters, a0 and a1.

In[95]:=
```
fit["FitResiduals"]
```

Out[95]=
```
{0.00106825, -0.000368418, -0.000419155, -0.0000174771, -0.000615799,
 -0.000615799, 0.000384201, 0.000669188, -0.00114817, 0.000363027, 0.000700149}
```

In[96]:=
```
Total[fit["FitResiduals"]^2] / (Length[fit["FitResiduals"]] - 2)
```

Out[96]=
```
5.27447 × 10^-7
```

In[97]:=
```
fit["EstimatedVariance"]
```

Out[97]=
```
5.27447 × 10^-7
```

The square root of the estimated variance is thus the RMS (root mean square) error -- the "typical" amount by which the fitted line misses the data. It has the same units as the original y values, and should be compared to the y uncertainties.

In[98]:=
```
Sqrt[fit["EstimatedVariance"]]
```

Out[98]=
```
0.000726255
```

Are differences of 0.0007 grams meaningful? For this experiment, you can read the scale to 0.001 g, and the zero current reading in this case varied between 181.183 and 181.184. Similarly, you probably saw fluctuations on the order of 0.001g routinely as you did the experiment. Accordingly, a variance of 0.0007 grams seems quite plausible.

Other Fit Properties.

This is a complete list of all available properties of the fit.

In[99]:=
```
fit["Properties"]
```

Out[99]=
```
{AdjustedRSquared, AIC, AICc, ANOVA, BasisFunctions, BetaDifferences,
 BestFit, BestFitParameters, BIC, CatcherMatrix, CoefficientOfVariation,
 CookDistances, CorrelationMatrix, CovarianceMatrix, CovarianceRatios, Data,
 Weights, DesignMatrix, DurbinWatsonD, Eigenstructure, EstimatedVariance,
 FitDifferences, FitResiduals, Function, FVarianceRatios, HatDiagonal,
 MeanPredictions, MeanPredictionBands, ParameterEstimates, PartialSumOfSquares,
 PredictedResponse, Properties, Response, RSquared, SequentialSumOfSquares,
 SingleDeletionVariances, SinglePredictions, SinglePredictionBands,
 StandardizedResiduals, StudentizedResiduals, VarianceInflationFactors}
```

A note about the “Rsquared” value

In[100]:=
```
fit["RSquared"]
```

Out[100]=
```
0.999998
```

This is close to 1.0. Does that mean you have a good fit? Not necessarily. Consider the following parabolic data set, where each point has a “random noise” added to each point (the noise has a mean of 0 and a standard deviation of 1).

In[101]:=
```
RandomReal[NormalDistribution[0, 1]]
```

Out[101]=
```
0.265018
```

In[102]:=
```
parab = Table[
  {x, 1.0 + 0.5 x + 0.3 x^2 + RandomReal[NormalDistribution[0, 1]]}, {x, 1, 20, 2}]
```

Out[102]=
```
{{1, 1.35556}, {3, 4.95557}, {5, 10.4837}, {7, 17.4556}, {9, 30.5001},
 {11, 41.9389}, {13, 58.0746}, {15, 75.7125}, {17, 92.608}, {19, 118.617}}
```

Let’s try fitting it with a straight line

In[103]:=
```
pfit = LinearModelFit[parab, x, x]
```

Out[103]=
```
FittedModel[ -19.3 + 6.45 x ]
```

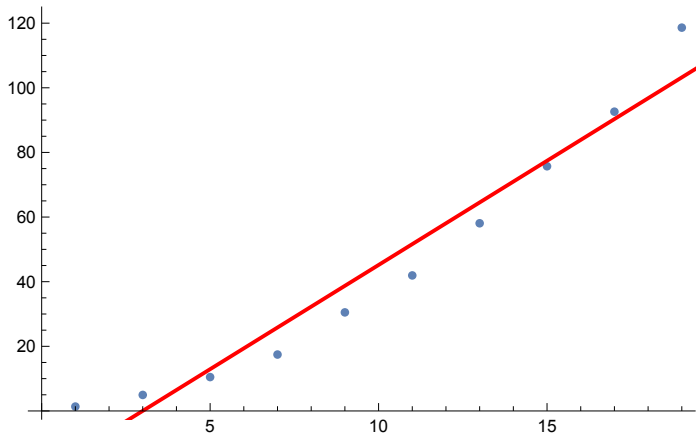
In[104]:=
```
pfit["ParameterConfidenceIntervalTable"]
```

Out[104]=

	Estimate	Standard Error	Confidence Interval
1	−19.3256	6.15082	{−33.5094, −5.1418}
x	6.44958	0.533343	{5.21968, 7.67947}

In[105]:=
```
Show[{ListPlot[parab],
  Plot[pfit[x] , {x, 0, 20}, PlotStyle → Red, LabelStyle → Larger]}]
```

Out[105]=



In[106]:=
```
pfit["RSquared"]
```

Out[106]=
```
0.948131
```

This is still close to 1.0, even though we're missing a systematic trend in the data. The R^2 value does not distinguish between data points that scatter about a line and those that deviate systematically from the line. You need to look at the actual graph. Obviously, a parabola is the right fit here:

In[107]:=
```
Sqrt[pfit["EstimatedVariance"]]
```

Out[107]=
```
9.68867
```

A typical variance of 9.4 seems unreasonable given that our data should have noise on the order of 1.

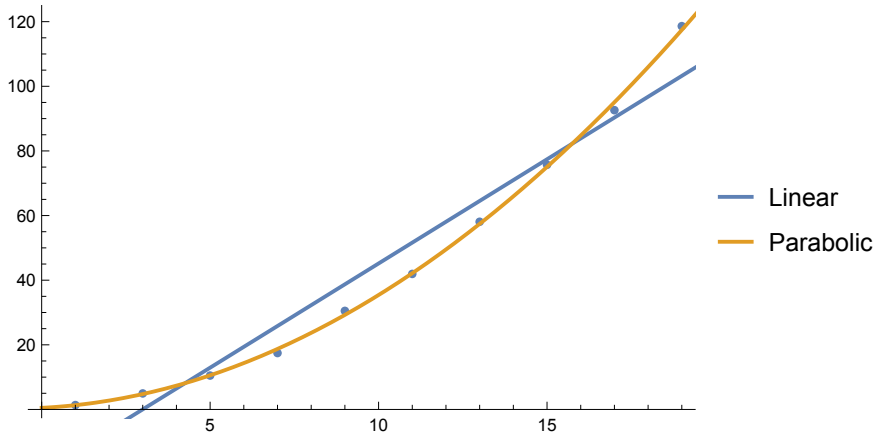
In[108]:=
```
pfit2 = LinearModelFit[parab, {x, x^2}, x]
```

Out[108]=

$$\text{FittedModel}\left[0.496 + 0.533\,x + 0.296\,x^2\right]$$

In[109]:=
```
Show[{ListPlot[parab],
  Plot[{pfit[x], pfit2[x]} , {x, 0, 20}, PlotLegends → {"Linear", "Parabolic"}]}]
```

Out[109]=



In[110]:=
```
Sqrt[pfit2["EstimatedVariance"]]
```

Out[110]=
```
1.28291
```

A typical mismatch of roughly 1 between the data and the fit makes sense, given that we added noise with a typical amplitude of about 1.