Least Squares Fitting to a Straight Line
Physics 238

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 fitting a straight line.

Fitting a Straight Line

Data

First, here is some sample data of {mass, angle} pairs from the torsional oscillator experiment. Masses are listed as negative when they apply a negative torque.

Typing the data using the Classroom Assistant:

From the Palettes -> Classroom Assistant pallete, 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, and Ctrl-, to add a new column.)

In[10]:= $$\texttt{data} = \begin{pmatrix} -400 & 1.505 \\ -350 & 1.645 \\ -300 & 1.800 \\ -250 & 1.980 \\ -200 & 2.180 \\ -150 & 2.380 \\ -100 & 2.600 \\ -50 & 2.800 \\ 0 & 3.005 \\ 50 & 3.205 \\ 100 & 3.405 \\ 150 & 3.620 \\ 200 & 3.830 \\ 250 & 4.025 \\ 300 & 4.235 \\ 350 & 4.460 \\ 400 & 4.645 \\ 0 & 3.005 \end{pmatrix};$$

Typing the data directly:

You can also type in 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[11]:=
```
data = {
   {-400, 1.505}, {-350, 1.645}, {-300, 1.800}, {-250, 1.980},
   {-200, 2.180}, {-150, 2.380}, {-100, 2.600}, {-50, 2.800},
   {0, 3.005},
   {50, 3.205}, {100, 3.405}, {150, 3.620}, {200, 3.830},
   {250, 4.025}, {300, 4.235}, {350, 4.460}, {400, 4.645},
   {0, 3.005}
  };
```

In[12]:=
```
npts = Length[data]
```

Out[12]=

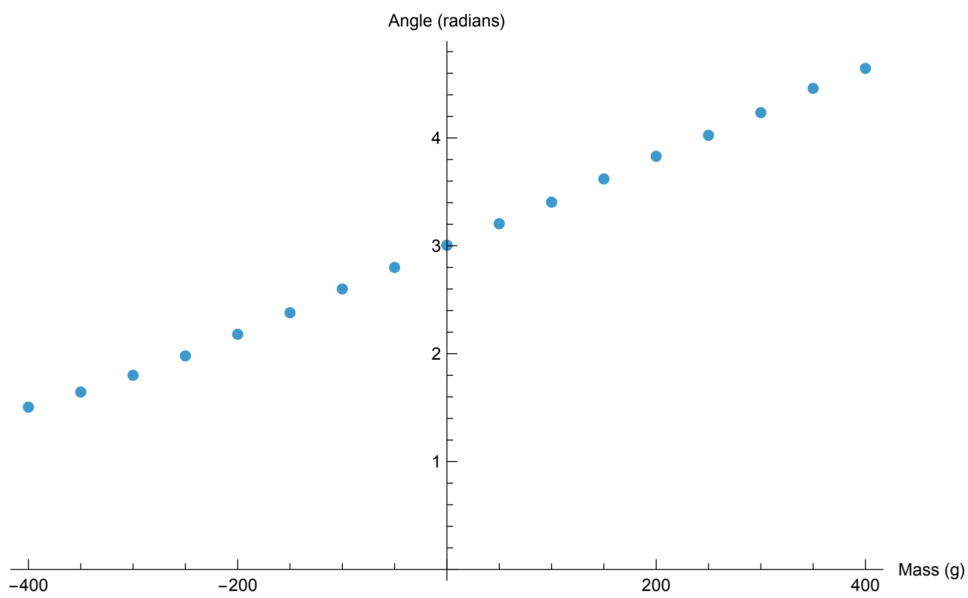
18

Plot the data.

Be sure to give your axes meaningful labels.

In[13]:=
```
ListPlot[data, AxesLabel → {"Mass (g)", "Angle (radians)"}]
```

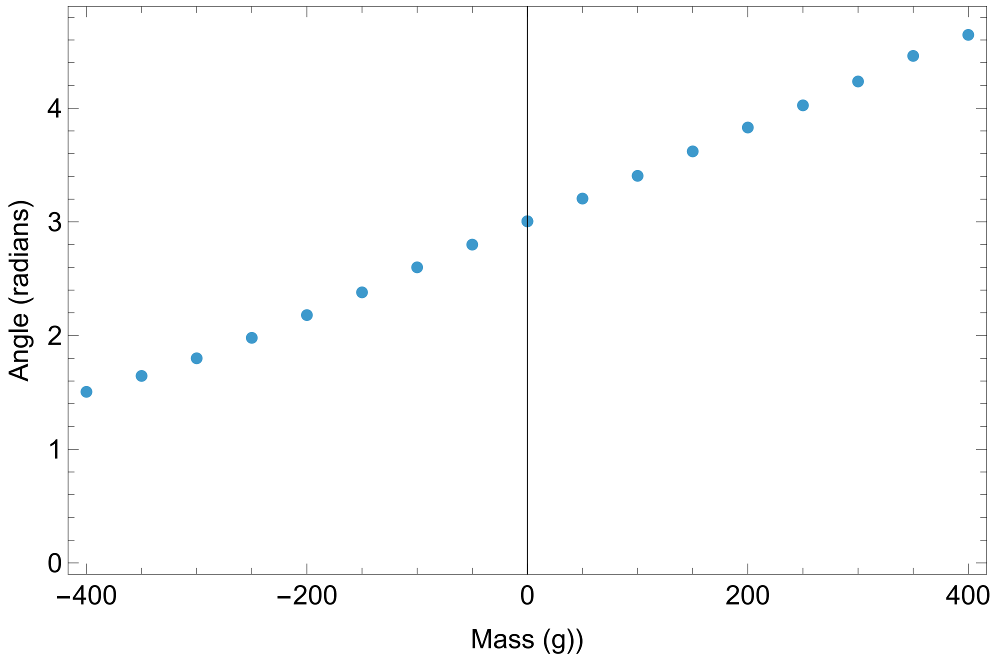
Out[13]=



The default plot isn't particularly nice, especially if you want to display it or include it in another document. There are lots of options to tweak; the following generally work fairly well in practice. It also shows how to assign a name to that plot, to make it easy to re-use it and layer the fit on top.

In[14]:=
```
dataPlot = ListPlot[data, ImageSize → Scaled[0.8],
  Frame → True, FrameLabel → {"Mass (g))", "Angle (radians)"},
  LabelStyle → Larger, PlotStyle → Thick]
```

Out[14]=



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[15]:=
```
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 fuction, 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 Pallete 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[16]:=
```
calculateChisq[data_, a0_, a1_] :=
```

$$\frac{1}{\text{Length[data]} - 2} \sum_{i=1}^{\text{Length[data]}} (\text{yfit[data⟦i, 1⟧, a0, a1]} - \text{data⟦i, 2⟧})^2$$

In[19]:=
```
calculateChisq[data_, a0_, a1_] := 1/(Length[data] - 2)
  Sum[(yfit[data⟦i, 1⟧, a0, a1] - data⟦i, 2⟧)^2, {i, 1, Length[data]}]
```

Initial Explorations

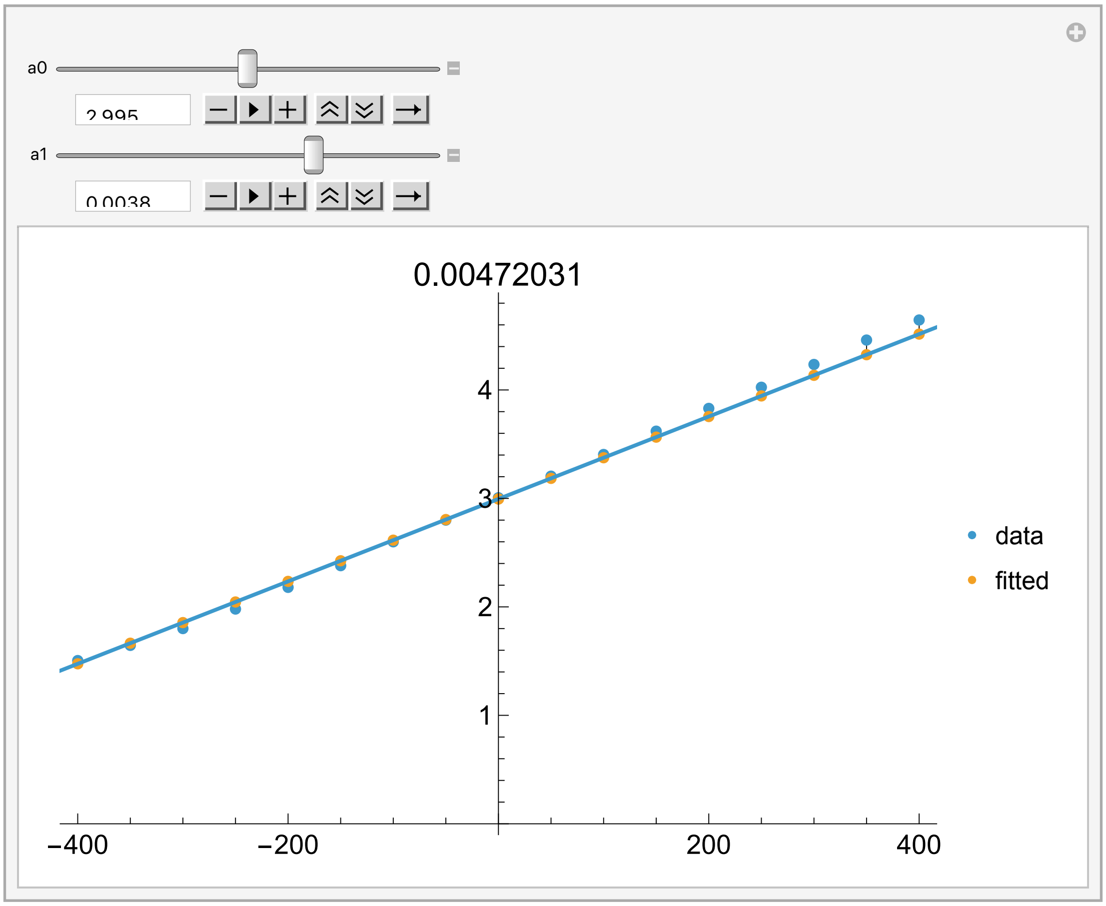
This command builds an interactive window showing the data, the current fit, and the chi squared value (as the plot title). It draws vertical lines from each data point to the fit line. Move the a0 and a1 sliders to minimize χ^2. Mathematica's "Filling" option gets confused if the data isn't sorted,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

In[20]:=
```
data = SortBy[data, First]
```

Out[20]=
```
{{-400, 1.505}, {-350, 1.645}, {-300, 1.8}, {-250, 1.98}, {-200, 2.18}, {-150, 2.38},
 {-100, 2.6}, {-50, 2.8}, {0, 3.005}, {0, 3.005}, {50, 3.205}, {100, 3.405},
 {150, 3.62}, {200, 3.83}, {250, 4.025}, {300, 4.235}, {350, 4.46}, {400, 4.645}}
```

In[22]:=
```
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, -450, 450}]},
  PlotLabel → calculateChisq[data, a0, a1],
  LabelStyle → Larger, ImageSize → Scaled[0.8]] ,
 {{a0, 3}, 1.5, 4.5, 0.005,  Appearance → "Open"},
 {{a1, 0}, -0.01, 0.01, 0.0002, Appearance → "Open"}
]
```

Out[22]=

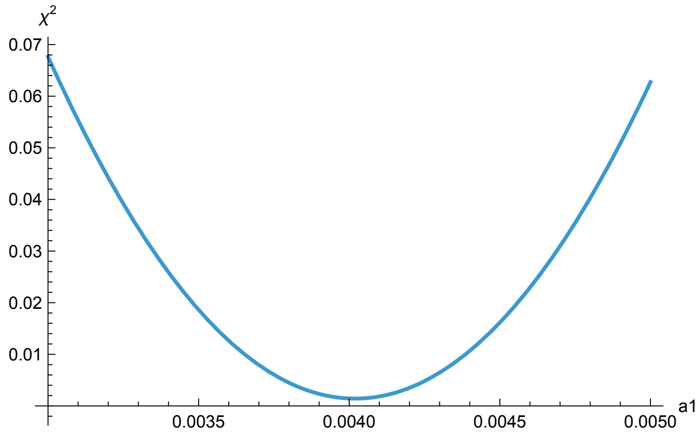


Minimizing χ^2

Picking the intercept a0 = 3 (close to the expected value), look at how χ^2 varies with a1.

In[23]:=
```
Plot[calculateChisq[data, 3.0, a1], {a1, 0.003, 0.005}, AxesLabel → {"a1", "χ²"}]
```

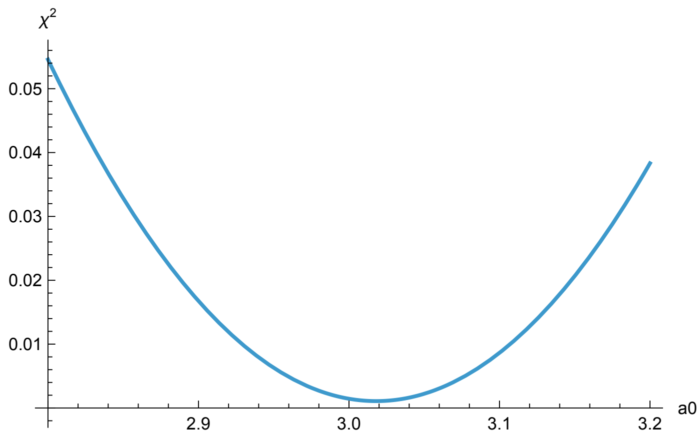
Out[23]=



Picking a1 = 0.004 (again, close to the expected value), look at how χ^2 varies with a0.

In[24]:=
```
Plot[calculateChisq[data, a0, 0.004], {a0, 2.8, 3.2}, AxesLabel → {"a0", "χ²"}]
```

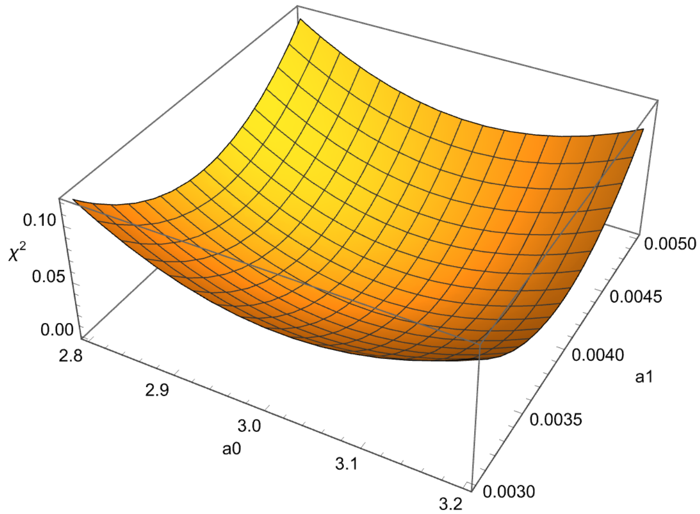
Out[24]=



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.

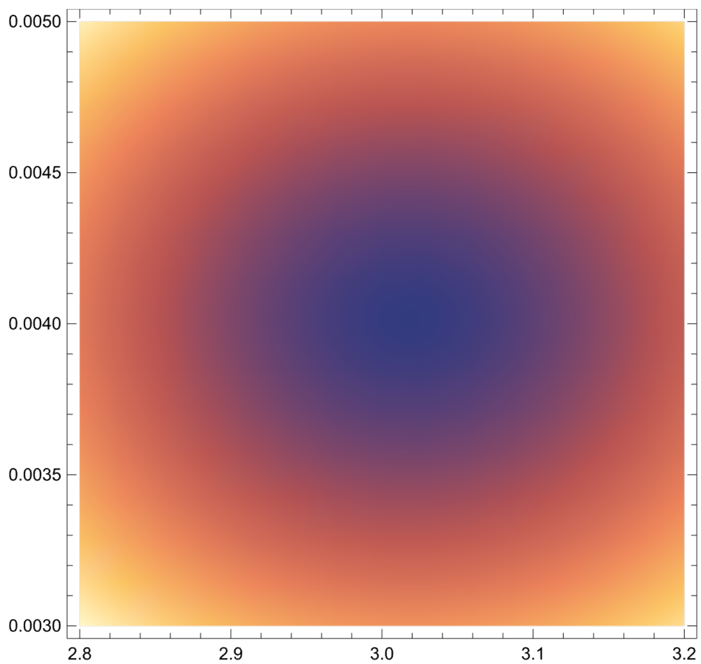
```
In[25]:= Plot3D[calculateChisq[data, a0, a1], {a0, 2.8, 3.2},
          {a1, 0.003, 0.005}, AxesLabel → {"a0", "a1", "χ²"}]
```

Out[25]=




```
In[26]:= DensityPlot[calculateChisq[data, a0, a1], {a0, 2.8, 3.2}, {a1, 0.003, 0.005}]
```

Out[26]=



Mathematica can find the minimum here as well.

```
In[27]:= Clear[a0, a1]
         result = FindMinimum[calculateChisq[data, a0, a1], {a0, a1}]
```

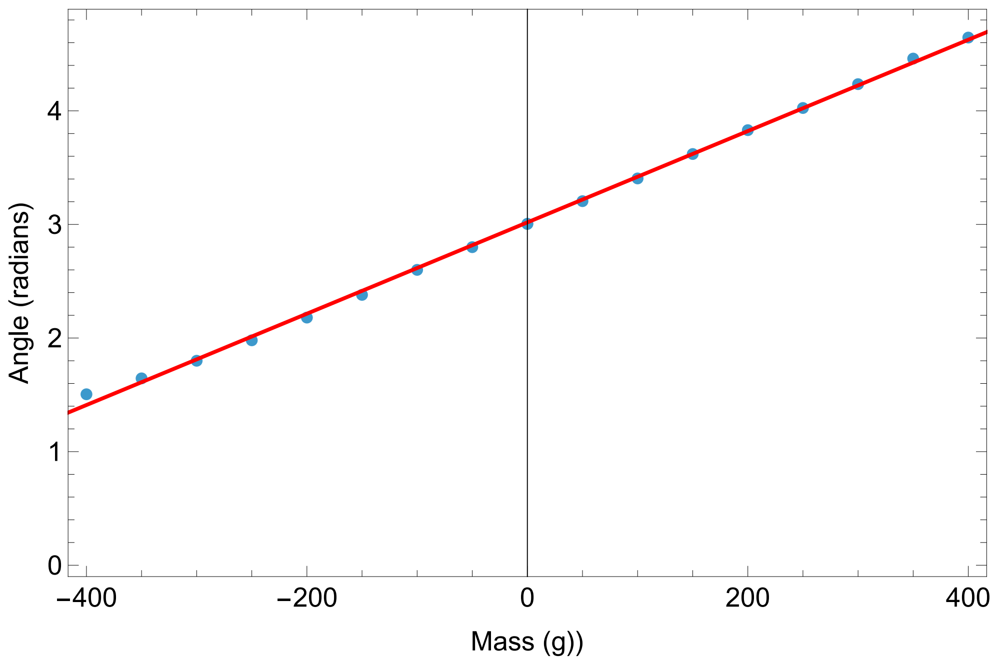
Out[28]=

```
{0.00105466, {a0 → 3.01806, a1 → 0.00401936}}
```

Finally, here is the best fit curve.

In[29]:=
```
Show[{
  dataPlot,
  Plot[yfit[x, a0, a1] /. result[[2]], {x, -450, 450}, PlotStyle → Red]
 }]
```

Out[29]=



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[30]:=
```
fit = LinearModelFit[data, {1, x}, x]
```

Out[30]=

FittedModel[3.02 + 0.00402 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[31]:=
```
fit["BestFit"]
```
Out[31]=
```
3.01806 + 0.00401936 x
```

In[32]:=
```
fit["BestFitParameters"]
```
Out[32]=
```
{3.01806, 0.00401936}
```

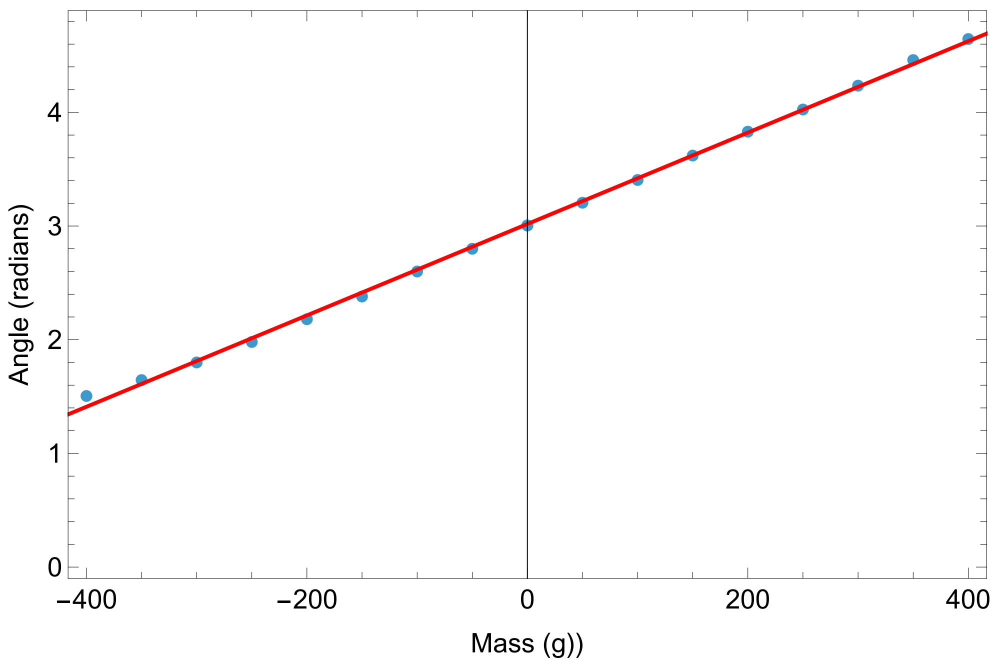
In[33]:=
```
fit["ParameterConfidenceIntervalTable"]
```
Out[33]=

	Estimate	Standard Error	Confidence Interval
1	3.01806	0.00765455	{3.00183, 3.03428 }
x	0.00401936	0.0000321555	{0.0039512 , 0.00408753 }

In[34]:=
```
fit["CorrelationMatrix"] // MatrixForm
```
Out[34]//MatrixForm=

$$\begin{pmatrix} 1. & 1.86783\times10^{-32} \\ 2.51254\times10^{-32} & 1. \end{pmatrix}$$

In[35]:=
```
Show[{dataPlot, Plot[fit[x] , {x, -450, 450}, PlotStyle → Red]}]
```
Out[35]=



In[36]:=
```
fit["EstimatedVariance"]
```
Out[36]=
```
0.00105466
```

This is the same as our χ^2.

In[37]:=
```
{a0, a1} = fit["BestFitParameters"]
```
Out[37]=
```
{3.01806, 0.00401936}
```

In[38]:=
```
calculateChisq[data, a0, a1]
```
Out[38]=
```
0.00105466
```

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[39]:=
```
fit["FitResiduals"]
```
Out[39]=
```
{0.0946895, 0.0337214, -0.0122467, -0.0332149, -0.034183, -0.0351511,
 -0.0161193, -0.0170874, -0.0130556, -0.0130556, -0.0140237, -0.0149918,
 -0.000959967, 0.0080719, 0.00210376, 0.0111356, 0.0351675, 0.0191993}
```

In[40]:=
```
Total[fit["FitResiduals"]^2] / (Length[fit["FitResiduals"]] - 2)
```
Out[40]=
```
0.00105466
```

In[41]:=
```
fit["EstimatedVariance"]
```
Out[41]=
```
0.00105466
```

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[42]:=
```
Sqrt[fit["EstimatedVariance"]]
```
Out[42]=
```
0.0324755
```

Are differences of 0.03 radians meaningful? For this experiment, you can read the scale to at lead ± 0.02 radians, and you can likely interpolate to 0.01 or even 0.005 radians, so variances of 0.03 radians seem larger than can be explained by simple random measurement error. Looking carefully at the graph, it seems likely that we twisted the wire a little beyond its linear range, and are starting to see a slightly nonlinear response.

Residual Plot

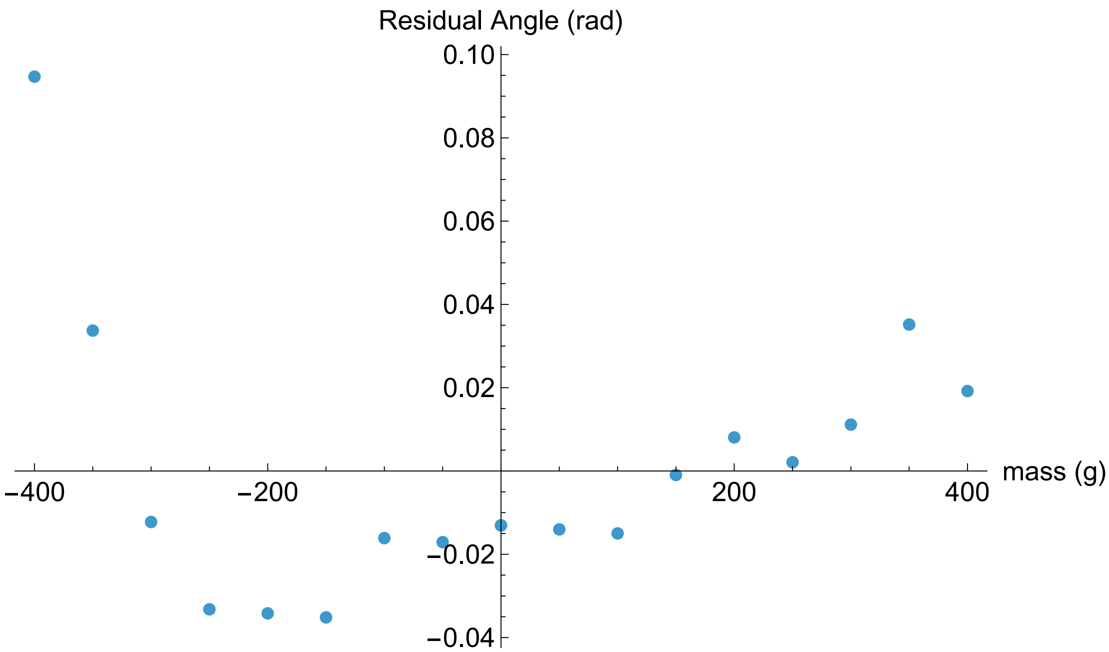
Sometimes, it is helpful to look at the residuals in more detail to try to understand why the fit and data might not agree. A residual plot can sometimes highlight trends within the data that might not be evident in the initial fit plot and that might warrant additional investigation.

In[43]:=
```
residuals =
 Table[{data⟦i, 1⟧, (data⟦i, 2⟧ - fit[data⟦i, 1⟧])}, {i, 1, Length[data]}]
```

Out[43]=
```
{{-400, 0.0946895}, {-350, 0.0337214}, {-300, -0.0122467},
 {-250, -0.0332149}, {-200, -0.034183}, {-150, -0.0351511},
 {-100, -0.0161193}, {-50, -0.0170874}, {0, -0.0130556}, {0, -0.0130556},
 {50, -0.0140237}, {100, -0.0149918}, {150, -0.000959967}, {200, 0.0080719},
 {250, 0.00210376}, {300, 0.0111356}, {350, 0.0351675}, {400, 0.0191993}}
```

In[44]:=
```
ListPlot[residuals, AxesLabel → {"mass (g)", "Residual Angle (rad)"},
 LabelStyle → Larger, ImageSize → Large]
```

Out[44]=



Other Fit Properties.

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

In[45]:=
```
fit["Properties"]
```

Out[45]=
```
{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[ ]:=
```
fit["RSquared"]
```

Out[ ]=
```
0.998977
```

This is close to 1.0. Does that mean you have a good fit? Not necessarily. Consider the following parabolic data set:

In[46]:=
```
parab = Table[{x, 1.0 + 0.5 x + 0.3 x^2}, {x, 1, 20, 2}]
```

Out[46]=
```
{{1, 1.8}, {3, 5.2}, {5, 11.}, {7, 19.2}, {9, 29.8},
 {11, 42.8}, {13, 58.2}, {15, 76.}, {17, 96.2}, {19, 118.8}}
```

Let’s try fitting it with a straight line

In[47]:=
```
pfit = LinearModelFit[parab, x, x]
```

Out[47]=
```
FittedModel[ -19.1 + 6.5 x ]
```

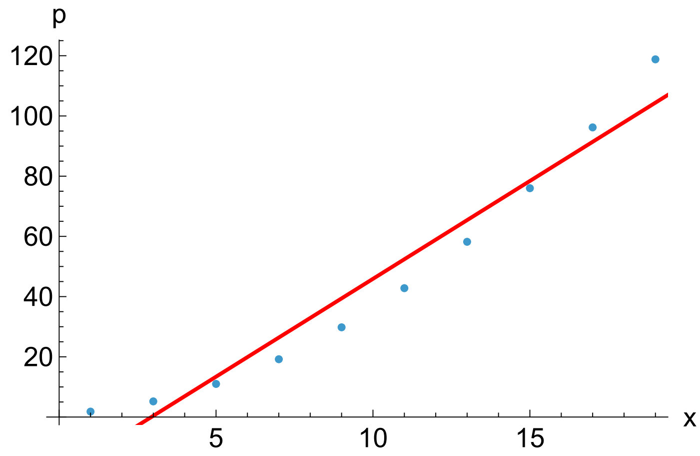
In[48]:=
```
pfit["ParameterConfidenceIntervalTable"]
```

Out[48]=

	Estimate	Standard Error	Confidence Interval
1	−19.1	6.18902	{−33.3719, −4.82809}
x	6.5	0.536656	{5.26247, 7.73753}

In[53]:=
```
Show[{ListPlot[parab, PlotStyle → Thick],
  Plot[pfit[x] , {x, 0, 20}, PlotStyle → Red]},
 LabelStyle → Larger, AxesLabel → {"x", "p"}]
```

Out[53]=



In[50]:=
```
pfit["RSquared"]
```

Out[50]=

0.948287

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:

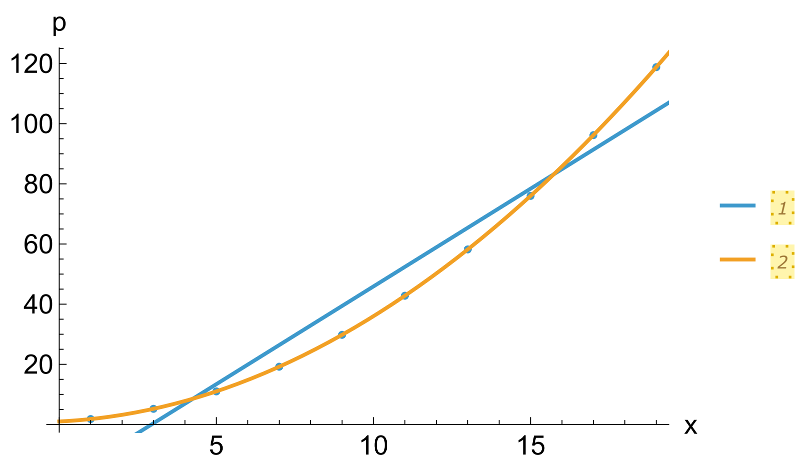
In[51]:=
```
pfit2 = LinearModelFit[parab, {x, x^2}, x]
```

Out[51]=

FittedModel[$1. + 0.5\,x + 0.3\,x^2$]

In[54]:=
```
Show[{ListPlot[parab],
  Plot[{pfit[x], pfit2[x]} , {x, 0, 20}, PlotLegends → Automatic]},
 LabelStyle → Larger, AxesLabel → {"x", "p"}]
```

Out[54]=



Summary Presentation

Given the data set:

In[ ]:=
```
data
```
Out[ ]=
```
{{-400, 1.505}, {-350, 1.645}, {-300, 1.8}, {-250, 1.98}, {-200, 2.18}, {-150, 2.38},
 {-100, 2.6}, {-50, 2.8}, {0, 3.005}, {0, 3.005}, {50, 3.205}, {100, 3.405},
 {150, 3.62}, {200, 3.83}, {250, 4.025}, {300, 4.235}, {350, 4.46}, {400, 4.645}}
```

Fit to a straight line

In[ ]:=
```
fit = LinearModelFit[data, {1, x}, x]
```
Out[ ]=
```
FittedModel[ 3.01806 + 0.00401936 x ]
```

Find the values for the slope and intercept, along with their uncertainties.

In[ ]:=
```
fit["BestFitParameters"]
fit["ParameterConfidenceIntervalTable"]
```
Out[ ]=
```
{3.01806, 0.00401936}
```
Out[ ]=

	Estimate	Standard Error	Confidence Interval
1	3.01806	0.00765455	{3.00183, 3.03428 }
x	0.00401936	0.0000321555	{0.0039512 , 0.00408753 }

Look at how far off (in radians) a typical data point is from the best-fit line. Think about this value. Is it reasonable?

In[ ]:=
```
Sqrt[fit["EstimatedVariance"]]
```
Out[ ]=
```
0.0324755
```

Show the data and the fit on the same graph.

In[ ]:=
```
Show[{ListPlot[data], Plot[fit[x] , {x, -450, 450}, PlotStyle → Red]},
  Frame → True, LabelStyle → Larger, FrameLabel → {"Mass (g)", "Angle (radians)"}]
```

Out[ ]=