Fitting Data with *Mathematica* for a Nonlinear Model

In a number of experiments this semester, you will be asked to fit experimental data to a theoretical curve where a nonlinear fit is needed. This notebook walks through an example of calculating chi-squared in such a case where LinearModelFit won't work.

For this notebook, we will look at fitting a simple oscillating function to some data.

In[14]:=
```
Clear["Global`*"]; DateString[]
```
Out[14]=
```
Mon 17 Feb 2025 11:32:53
```

Entering Data

Importing from a file

In[15]:=
```
SetDirectory[NotebookDirectory[]]
```
Out[15]=
```
/Users/doughera/238/2025/lectures-dev/Ch08
```

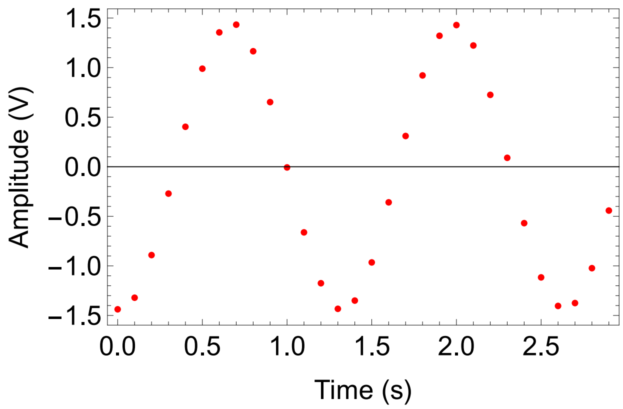
In[16]:=
```
FilePrint["nonlinear-modelfit-1-data.txt", 5] (* Print out the first 5 lines *)
```
```
0.00   -1.438
0.10   -1.321
0.20   -0.891
0.30   -0.271
0.40    0.403
```

This data file is just two columns of numbers separated by spaces. Mathematica can import this as a simple table.

In[17]:=
```
data = Import["nonlinear-modelfit-1-data.txt", "Table"]
```
Out[17]=
```
{{0., -1.438}, {0.1, -1.321}, {0.2, -0.891}, {0.3, -0.271}, {0.4, 0.403},
 {0.5, 0.989}, {0.6, 1.355}, {0.7, 1.433}, {0.8, 1.165}, {0.9, 0.652},
 {1., -0.007}, {1.1, -0.662}, {1.2, -1.175}, {1.3, -1.433}, {1.4, -1.35},
 {1.5, -0.965}, {1.6, -0.359}, {1.7, 0.31}, {1.8, 0.921}, {1.9, 1.321},
 {2., 1.429}, {2.1, 1.223}, {2.2, 0.725}, {2.3, 0.09}, {2.4, -0.569},
 {2.5, -1.116}, {2.6, -1.404}, {2.7, -1.375}, {2.8, -1.023}, {2.9, -0.442}}
```

In[18]:=
```
dataplot = ListPlot[data, PlotRange → All,
  PlotStyle → Red, Frame → True, LabelStyle → Larger,
  FrameLabel → {"Time (s)", "Amplitude (V)"}, ImageSize → Scaled[0.5]]
```

Out[18]=



Fitting a Model to the Data

For this exercise, we will try to fit this to a cosine function, allowing for the possibility that the average value might not be zero, due to some offset in the sensor. This function has 4 free parameters: A, ω, ϕ, and xoff.

In[19]:=
```
x[A_, ω_, ϕ_, t_, xoff_] := A Cos[ω t + ϕ] + xoff
```

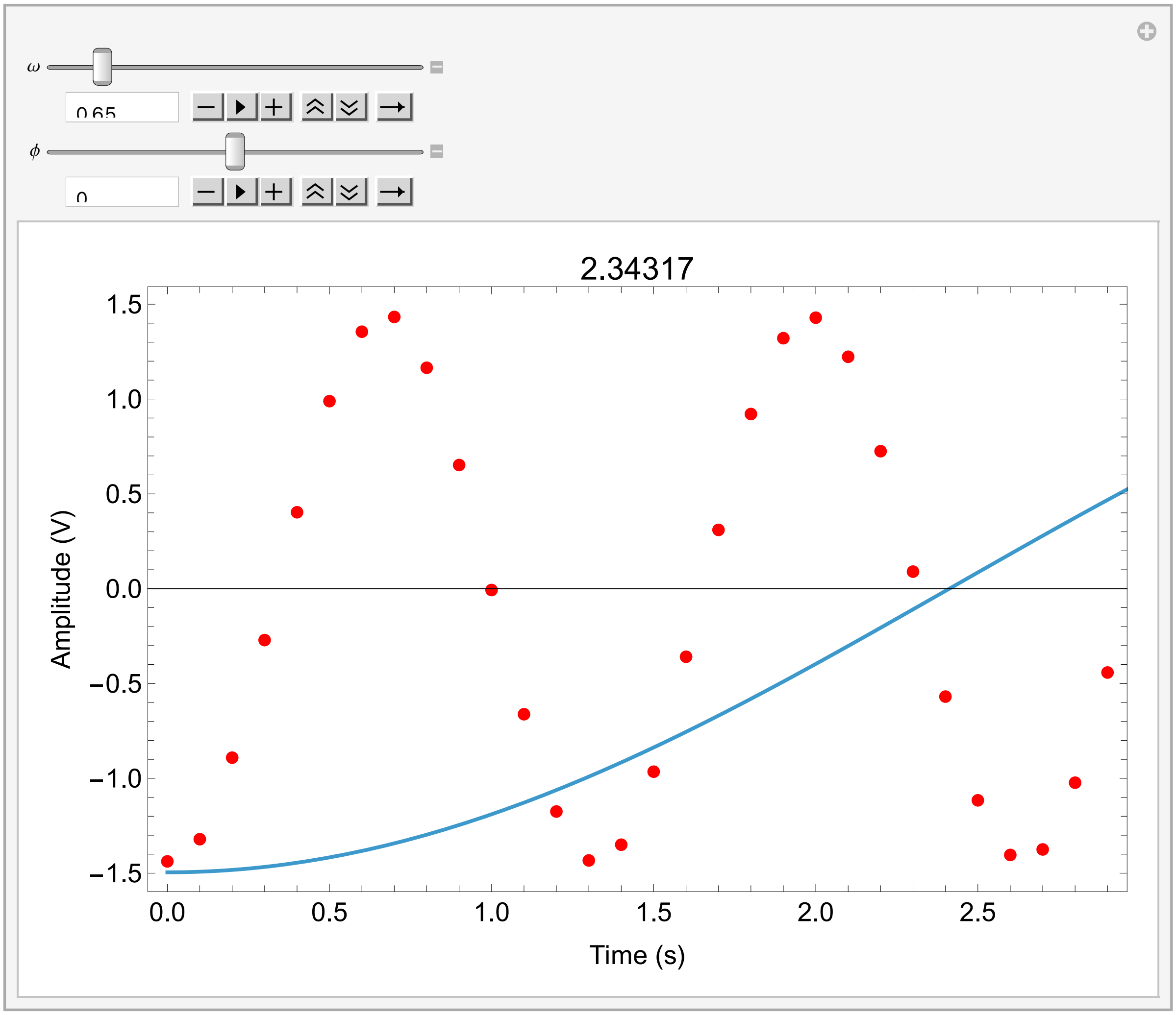
In[20]:=
```
Clear[chisq]
chisq[data_, A_, ω_, ϕ_, xoff_] :=
 Sum[(data[[i, 2]] - x[A, ω, ϕ, data[[i, 1]], xoff])^2, {i, 1, Length[data]}] /
  (Length[data] - 4)
```

For convenience of our chisq visualization, pick Amplitude and offset values that seem plausible. Look at varying ω and ϕ.

In[22]:=
```
myxoff = 0.0044; myA = -1.5;
```

In[23]:=
```
Manipulate[
 Show[{
   dataplot,
   Plot[x[myA, ω, ϕ, t, myxoff], {t, 0, 3}]
  }, ImageSize → Large, PlotLabel → chisq[data, myA, ω, ϕ, myxoff]] ,
 {{ω, 0.65}, 0, 6, 0.05, Appearance → "Open"},
 {{ϕ, 0}, -π, π, 0.1, Appearance → "Open"}
]
```

Out[23]=



Minimizing χ^2

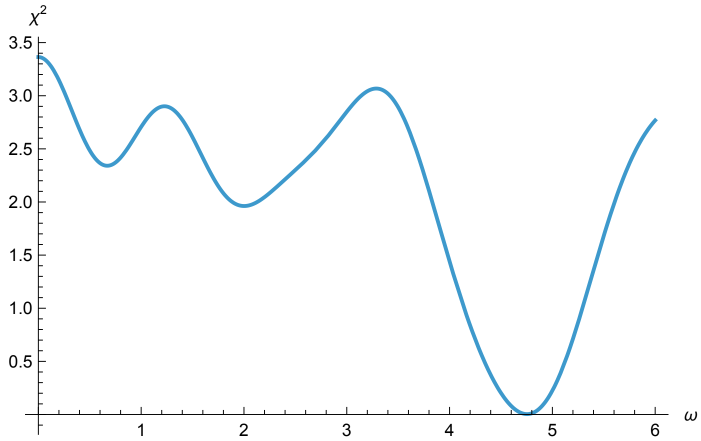
One-dimensional minimization

Picking A = -1.5 and a phase of 0, look at how χ^2 varies with ω.

In[24]:=
```
myA = -1.5; myϕ = 0;
```

In[25]:=
```
Plot[chisq[data, myA, ω, myϕ, myxoff], {ω, 0, 6 }, AxesLabel → {"ω", "χ^2"}]
```

Out[25]=



There clearly is a global minimum near 4.8, but there are also local minima near 0.7 and 2.0. Without help, *Mathematica* might get trapped in a local minimum.

In[26]:=
```
FindMinimum[chisq[data, myA, ω, myϕ, myxoff], ω]
(* Mathematica tries starting at ω = 1 *)
```

Out[26]=
```
{2.34117, {ω → 0.670019}}
```

In[27]:=
```
FindMinimum[chisq[data, myA, ω, myϕ, myxoff], {ω, 3}] (* Tell it to start at 3 *)
```

Out[27]=
```
{1.96213, {ω → 2.00135}}
```

This finds the second minimum. An even better initial guess will find the global minimum.

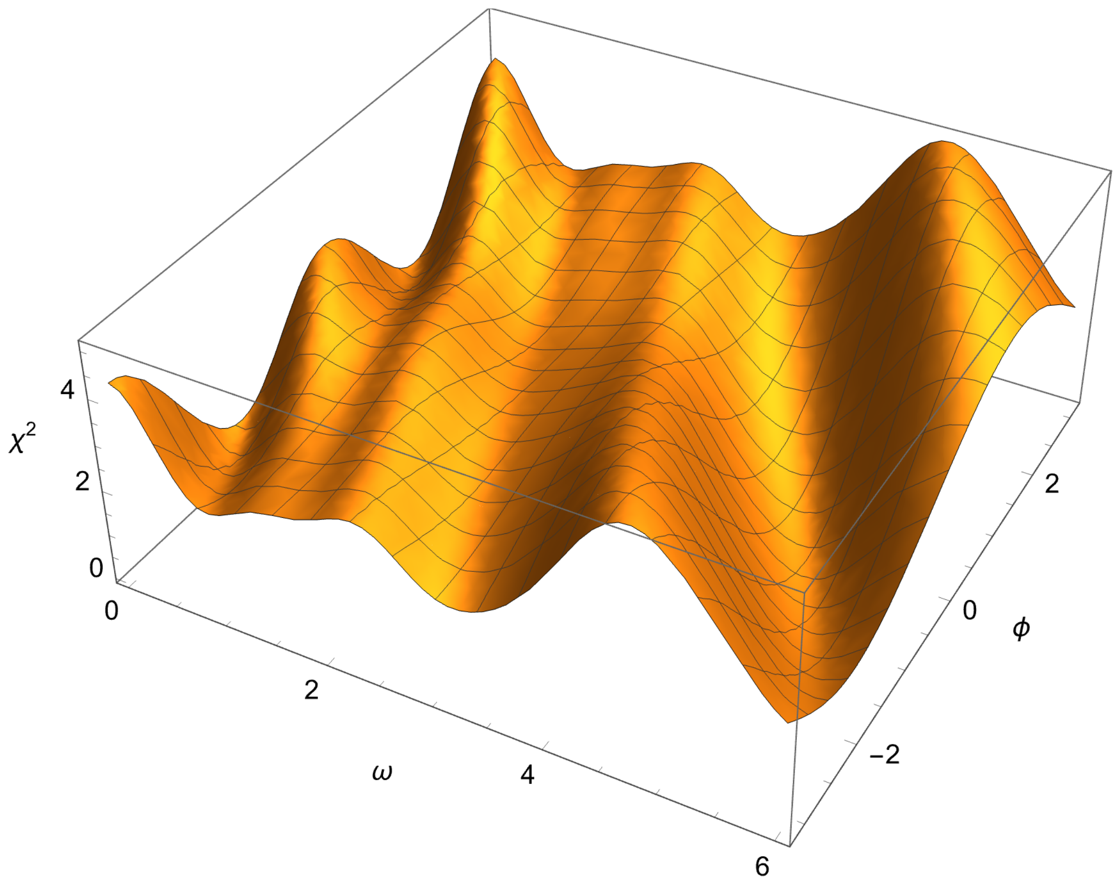
In[28]:=
```
FindMinimum[chisq[data, myA, ω, myϕ, myxoff], {ω, 4}] (* Tell it to start near 4 *)
```

Out[28]=
```
{0.0036504, {ω → 4.75364}}
```

Two-dimensional minimization

In[29]:=
```
Plot3D[chisq[data, myA, ω, ϕ, myxoff], {ω, 0, 6}, {ϕ, -π, π}, PlotRange → All,
 AxesLabel → {"ω", "ϕ", "χ^2"}, LabelStyle → Larger, ImageSize → Large]
```

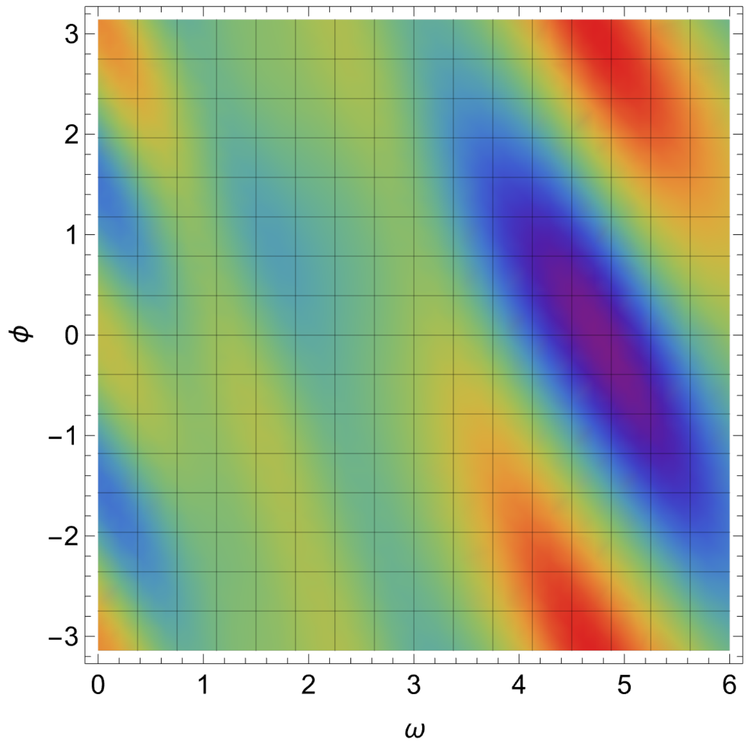
Out[29]=



Sometimes it helps to visualize this sort of thing as a 2d density plot.

In[30]:=
```
DensityPlot[chisq[data, myA, ω, ϕ, myxoff], {ω, 0, 6}, {ϕ, -π, π},
 PlotRange → All, FrameLabel → {"ω", "ϕ"}, LabelStyle → Larger,
 Mesh → True, ColorFunction → "Rainbow", ImageSize → Scaled[0.6]]
```

Out[30]=



In[31]:=
```
result = FindMinimum[chisq[data, myA, ω, ϕ, myxoff], {ω, ϕ}]
```

Out[31]=
```
{1.75164, {ω → 1.68242, ϕ → 0.87468}}
```

Mathematica chooses initial starting values of ‘1’. This gets caught in the local minimum on the left hand side. However, if you give it a good starting point for at least one of the parameters, *Mathematica* finds the minimum correctly. Here we will let it find all of the parameters, but give a good starting hint.

In[32]:=
```
result = FindMinimum[chisq[data, A, ω, ϕ, off], {A, {ω, 4}, ϕ, off}]
```

Out[32]=
```
{0.0000598132, {A → -1.43493, ω → 4.7841, ϕ → -0.0628613, off → 0.00441981}}
```

NonlinearModelFit

NonlinearModelFit[ ] performs this minimization automatically. Though it has certain algorithms to attempt to avoid local minima, you still sometimes need to give it hints.

In[33]:=
```
fit = NonlinearModelFit[data, x[A, ω, ϕ, t, xoff], {A, ω, ϕ, xoff}, t]
```

NonlinearModelFit : Failed to converge to the requested accuracy or precision within 100 iterations.

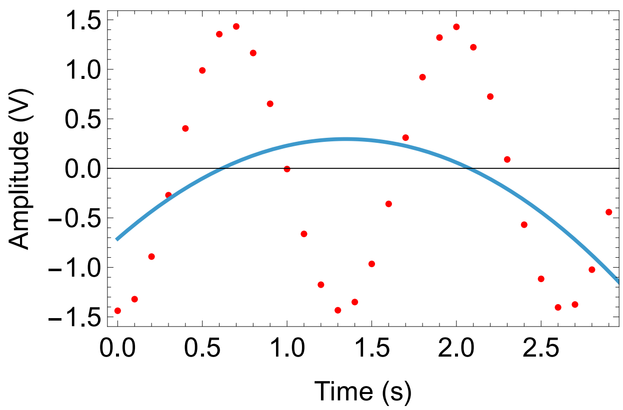
Out[33]=
```
FittedModel[ -879. + 879. Cos[0.0479 - 0.0355 t] ]
```

In[34]:=
```
fit["BestFitParameters"]
```

Out[34]=
```
{A → 879.285, ω → 0.0355321, ϕ → -0.0478572, xoff → -878.989}
```

In[35]:=
```
fitplot = Plot[fit[t], {t, 0, 3}, PlotRange → All];
Show[dataplot, fitplot]
```

Out[36]=



In[37]:=
```
Sqrt[fit["EstimatedVariance"]] (* The typical error is about 1 volt. *)
```

Out[37]=
```
1.01463
```

This fit is quite poor. Give it a hint for ω, and it does much better.

In[38]:=
```
fit = NonlinearModelFit[data, x[A, ω, ϕ, t, xoff], {A, {ω, 4}, ϕ, xoff}, t];
fit["BestFitParameters"]
```

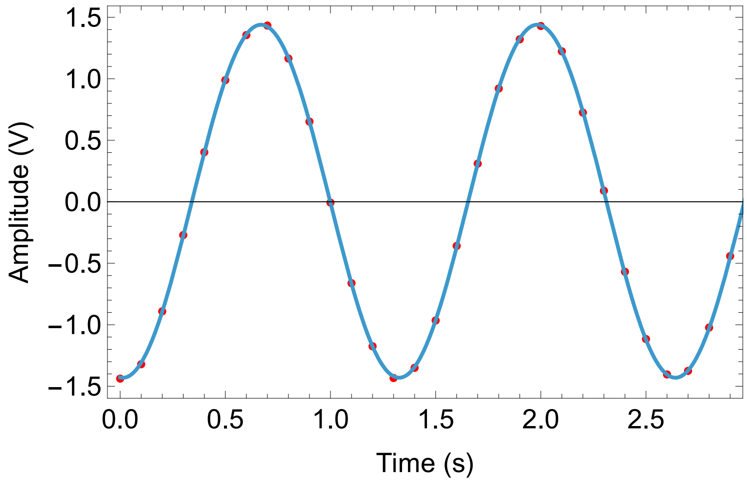
Out[39]=
```
{A → -1.43493, ω → 4.7841, ϕ → -0.0628613, xoff → 0.00441981}
```

In[40]:=
```
Sqrt[fit["EstimatedVariance"]] (* The typical error is now about 8 mV. *)
```

Out[40]=
```
0.0077339
```

In[41]:=
```
fitplot = Plot[fit[t], {t, 0, 3}];
Show[{dataplot, fitplot}, ImageSize → Scaled[0.6]]
```

Out[42]=



Fit Diagnostics

NonlinearModelFit returns a variety of measures for how "good" the fit is. The online help has more details, but four particularly useful ones are shown here.

Best Fit Parameters

In[43]:=
```
fit["BestFitParameters"]
```

Out[43]=
```
{A → -1.43493, ω → 4.7841, ϕ → -0.0628613, xoff → 0.00441981}
```

The best fit parameters are given as a set of replacement rules. You can assign a value to a variable with the ./ notation. For example, to extract the phase you could do

In[63]:=
```
phase = ϕ /. fit["BestFitParameters"]
```

Out[63]=
```
-0.0628613
```

Parameter Confidence Interval

This item gives both the uncertainty and the 95% confidence-level interval.

In[44]:=
```
fit["ParameterConfidenceIntervalTable"]
```

Out[44]=

	Estimate	Standard Error	Confidence Interval
A	−1.43493	0.00199071	{−1.43902, −1.43084 }
ω	4.7841	0.00169737	{4.78062, 4.78759 }
ϕ	−0.0628613	0.00289789	{−0.068818, −0.0569046 }
xoff	0.00441981	0.00144711	{0.00144523 , 0.00739438 }

The table entries are available in the raw form, and you can extract the ones you want with the Part [[ ]] notation. For example, the ω values are in the 2nd row, 1st and 2nd columns

In[45]:=
```
fit["ParameterConfidenceIntervalTableEntries"]
```
Out[45]=
```
{{-1.43493, 0.00199071, {-1.43902, -1.43084}},
 {4.7841, 0.00169737, {4.78062, 4.78759}},
 {-0.0628613, 0.00289789, {-0.068818, -0.0569046}},
 {0.00441981, 0.00144711, {0.00144523, 0.00739438}}}
```

In[46]:=
```
{bestω, δω} = fit["ParameterConfidenceIntervalTableEntries"][[2, {1, 2}]]
```
Out[46]=
```
{4.7841, 0.00169737}
```

Correlation Matrix

The "CorrelationMatrix" tells you about the correlations among the fit parameters. Low values mean the parameters are poorly correlated. This is good -- it means they are mostly independent. On the other hand, high values mean the parameters are highly correlated. This is often a sign of trouble in the proposed fit. For example, if you are trying to fit the parameters A and B in the function

$$f[x] = A\frac{x}{B}$$

then the fitted parameters for A and B will be highly correlated. You could double A and also double B, and still get the same answer.

In[47]:=
```
fit["CorrelationMatrix"] // MatrixForm
```
Out[47]//MatrixForm=

1.	-0.0313374	0.000416606	-0.125407
-0.0313374	1.	-0.872054	0.16148
0.000416606	-0.872054	1.	-0.17947
-0.125407	0.16148	-0.17947	1.

For example, the 3rd element on the first row (0.000416606) means that changes in the first parameter (A) are quite weakly correlated with changes in the third parameter (ϕ). The 1's along the diagonals mean each parameter is perfectly correlated with itself, as expected. Checking the correlation matrix is a good way to check whether your proposed fitting function really has the independent parameters you thought it did.

Here, the only tricky one is the -0.87 correlation between ω and the phase ϕ. Because there are only two full oscillations in the data, you can (partially) compensate for increasing ϕ by decreasing ω. This shows up clearly in the density plot of χ^2, where I have fixed A and xoff to approximately the right values, but allowed ω and ϕ to vary. The deep purple diagonal valley at the center shows how you can increase ω but decrease ϕ without changing χ^2much. (You can also fiddle with the Manipulate box above to see this.)

RMS Error

The RMS error is the typical amount the fit misses a data point. Here it is about 0.0077, which is about 0.5% of the amplitude. Depending on the specific sensor used, this is probably a plausible uncertainty.

In[48]:=
```
Sqrt[fit["EstimatedVariance"]]
```
Out[48]=
```
0.0077339
```

In[49]:=
```
(A /. fit["BestFitParameters"])
```
Out[49]=
```
-1.43493
```

In[50]:=
```
(100 * Sqrt[fit["EstimatedVariance"]]) / (A /. fit["BestFitParameters"])
```
Out[50]=
```
-0.538974
```

The resulting fit object has many other "Properties" you can query. Here is a list of all of them:

In[51]:=
```
fit["Properties"]
```
Out[51]=
```
{AdjustedRSquared, AIC, AICc, ANOVA, BestFit, BestFitParameters,
 BIC, CorrelationMatrix, CovarianceMatrix, CurvatureConfidenceRegion,
 Data, Weights, EstimatedVariance, FitCurvature, FitResiduals, Function,
 HatDiagonal, MaxIntrinsicCurvature, MaxParameterEffectsCurvature,
 MeanPredictions, MeanPredictionBands, ParameterEstimates,
 ParameterConfidenceRegion, ParameterBias, PredictedResponse, Properties,
 Response, RSquared, SingleDeletionVariances, SinglePredictions,
 SinglePredictionBands, StandardizedResiduals, StudentizedResiduals}
```

Summary of Usage

Import Data

In[52]:=
```
SetDirectory[NotebookDirectory[]];
```

In[53]:=
```
data = Import["nonlinear-modelfit-1-data.txt", "Table"];
dataplot = ListPlot[data, PlotRange → All,
   PlotStyle → Red, Frame → True, LabelStyle → Larger,
   FrameLabel → {"Time (s)", "Amplitude (V)"}, ImageSize → Scaled[0.5]];
```

Construct Model

In[55]:=
```
x[A_, ω_, ϕ_, t_, xoff_] := A Cos[ω t + ϕ] + xoff
```

Run NonlinearModel Fit, giving hints for parameters as needed

In[56]:=
```
fit = NonlinearModelFit[data, x[A, ω, ϕ, t, xoff],
   {A, {ω, 4}, ϕ, xoff}, t]; (* Here we give an initial guess for ω *)
fit["ParameterConfidenceIntervalTable"]
```

Out[57]=

	Estimate	Standard Error	Confidence Interval
A	−1.43493	0.00199071	{−1.43902, −1.43084 }
ω	4.7841	0.00169737	{4.78062, 4.78759 }
ϕ	−0.0628613	0.00289789	{−0.068818, −0.0569046 }
xoff	0.00441981	0.00144711	{0.00144523 , 0.00739438 }

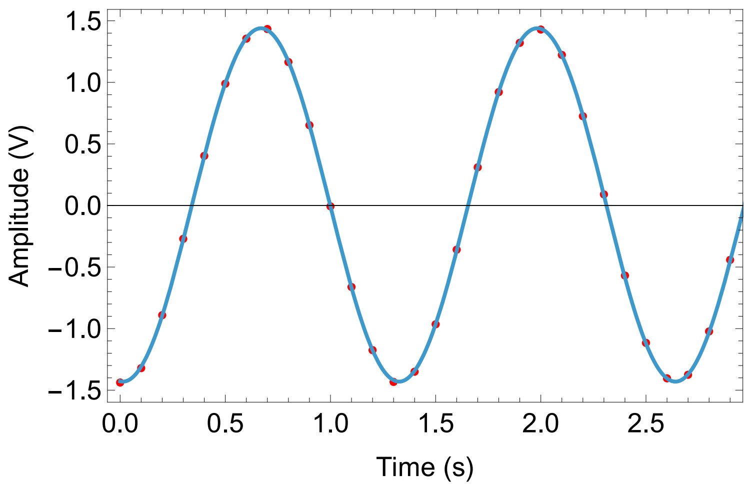
In[58]:=
```
Sqrt[fit["EstimatedVariance"]] (* Find typical error. *)
```

Out[58]=

```
0.0077339
```

In[59]:=
```
fitplot = Plot[fit[t], {t, 0, 3}];
Show[{dataplot, fitplot}, ImageSize → Scaled[0.6]]
```

Out[60]=



Extracting values (with uncertainty) from the fit

In[66]:=
```
phase = ϕ /. fit["BestFitParameters"]
```

Out[66]=

```
-0.0628613
```

In[67]:=
```
{bestω, δω} = fit["ParameterConfidenceIntervalTableEntries"][[2, {1, 2}]]
```

Out[67]=

```
{4.7841, 0.00169737}
```