I. Example of fitting to a quadratic.

II. Use of fit as calibration to determine unknown.

In[59]:=

```
<< "ErrorBarPlots`"
```

Here is a definition of a linear function f :

In[117]:=

```
a = {0.25, 2.3, -.03};
f[x_, a_] := a[[1]] + a[[2]] * x
```

Now generate a data set in list named "data."

The "x" values are in data[[i,1]] (column 1 below), the "measured" y values are in data[[i,2]] (column 2 below), and the uncertainties in the y values are in data[[i,3]] (column 3 below).

Convert one of the next three cells from "text" format to "input" format to fill the list named "data." The first cell generates a data set based on a quadratic function with random noise; the second cell imports the data from a file named "sample.dat"; the third uses the sample data from from the writeup.

data = Table[{x = i,f[x,a]-0.3 x^2 +Random[NormalDistribution[0,.1]],.1},{i,0,10}];

data=Import["sample.dat"]

In[62]:=

```
data = {{0, 0.18804, 0.1}, {1, 2.40351, 0.1}, {2, 4.76703, 0.1},
   {3, 6.80721, 0.1}, {4, 8.95863, 0.1}, {5, 10.9312, 0.1}, {6, 13.0096, 0.1},
   {7, 14.6587, 0.1}, {8, 16.8880, 0.1}, {9, 18.3569, 0.1}, {10, 20.3084, 0.1}};
```

In[119]:=

```
MatrixForm[data]
```

Out[119]//MatrixForm=

$$\begin{pmatrix} 0 & 0.18804 & 0.1 \\ 1 & 2.40351 & 0.1 \\ 2 & 4.76703 & 0.1 \\ 3 & 6.80721 & 0.1 \\ 4 & 8.95863 & 0.1 \\ 5 & 10.9312 & 0.1 \\ 6 & 13.0096 & 0.1 \\ 7 & 14.6587 & 0.1 \\ 8 & 16.888 & 0.1 \\ 9 & 18.3569 & 0.1 \\ 10 & 20.3084 & 0.1 \end{pmatrix}$$

Separate the list "data" into a list that has x-y pairs only ("xydata"), and a list that has the weights for each point ("w"). This is convenient for the Mathematica syntax in the function LinearModelFit below.

In[120]:=

```
xydata = Table[{data[[i, 1]], data[[i, 2]]}, {i, 1, Length[data]}];
w = Table[1 / data[[i, 3]]^2, {i, 1, Length[data]}];
```

Export["example3.dat",testdata]; (* Export data for gnuplot comparison if desired *)

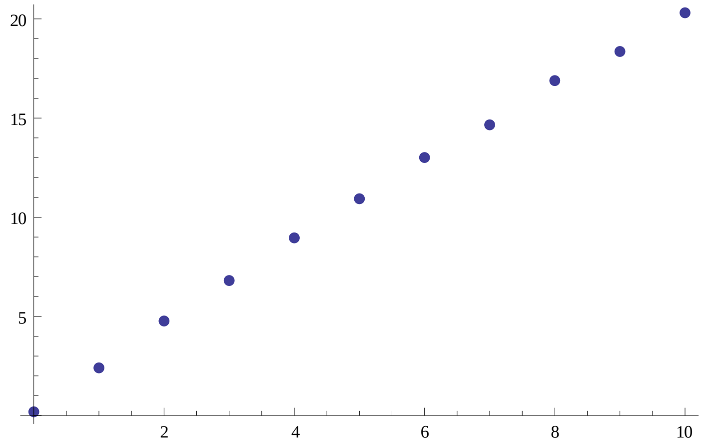
Define functions that calculate $\chi 2$. The function $\chi 2$ includes weights; the function χ2nw doesn't.

In[122]:=

```
χ2[data_, p_] :=
 Sum[(f[data[[i, 1]], p] - data[[i, 2]])^2 / data[[i, 3]]^2, {i, 1, Length[data]}]
χ2nw[data_, p_] := Sum[(f[data[[i, 1]], p] - data[[i, 2]])^2, {i, 1, Length[data]}]
```

In[124]:=
```
dataplot =
 ErrorListPlot[{Table[{{data⟦i, 1⟧, data⟦i, 2⟧}, ErrorBar[data⟦i, 3⟧]}, {i, 1, Length[data]}]},
  PlotStyle → PointSize[0.016]]
```

Out[124]=



- **Minimize $\chi 2$ with weighted LINEAR fitting**

In[125]:=
```
Clear[x];
lm = LinearModelFit[xydata, {1, x}, x, Weights → w]
```

Out[126]= FittedModel[$0.637374 + 2.00484\ x$]

You can get other statistical information about the fit from *Mathematica*. Note that the CovarianceMatrix of *Mathematica* is the curvature matrix, α, TIMES the reduced $\chi 2$ of the original data set.. (See my notes and error_test.nb)

In[127]:=
```
lm["BestFitParameters"]
```

Out[127]= {0.637374, 2.00484}

In[128]:=
```
lm["ParameterErrors"]
```

Out[128]= {0.175528, 0.0296696}

In[129]:=
```
lm["ParameterTable"]
```

Out[129]=

	Estimate	Standard Error	t-Statistic	P-Value
1	0.637374	0.175528	3.63118	0.00547555
x	2.00484	0.0296696	67.572	1.71976×10^{-13}

In[130]:=
```
MatrixForm[lm["CovarianceMatrix"]]
```

Out[130]//MatrixForm=

$$\begin{pmatrix} 0.0308101 & -0.00440144 \\ -0.00440144 & 0.000880288 \end{pmatrix}$$

In[131]:=
```
MatrixForm[lm["CorrelationMatrix"]]
```

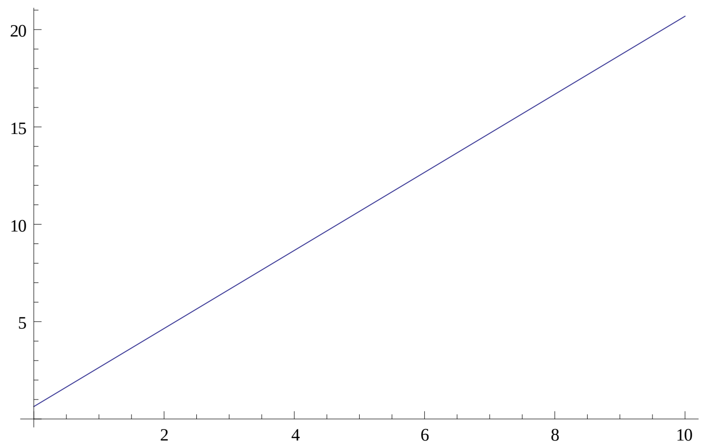
Out[131]//MatrixForm=

$$\begin{pmatrix} 1. & -0.845154 \\ -0.845154 & 1. \end{pmatrix}$$

To use information returned by Mathematica I give appropriate names to the parameters

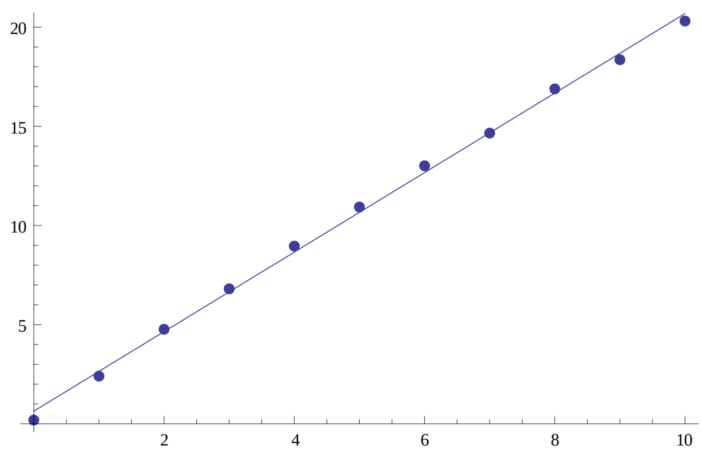
```
In[132]:= {b, m} = lm["BestFitParameters"]
          χ2min = χ2[data, {b, m}]
          rχ2 = χ2min / (Length[data] - 2)

Out[132]= {0.637374, 2.00484}

Out[133]= 87.1485

Out[134]= 9.68317

In[135]:= funplot = Plot[{lm[x]}, {x, 0, 10}]
```

Out[135]=




```
In[136]:= Show[dataplot, funplot]
```

Out[136]=

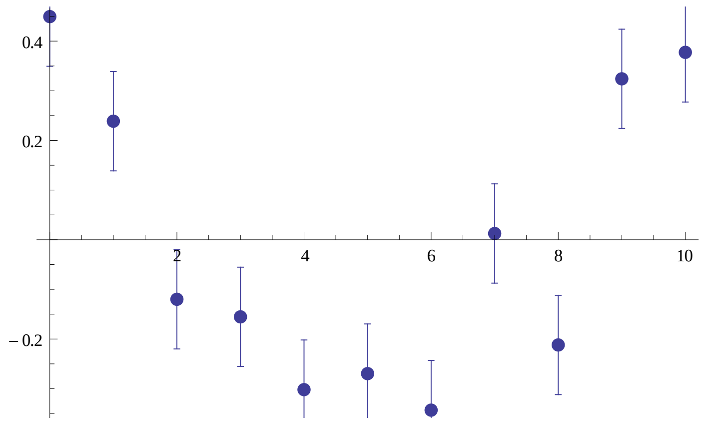


In[81]:=

This plot looks linear, but the value of the reduced $\chi 2$ is pretty high. Let's plot residuals:

In[137]:=
```
ErrorListPlot[Table[{{data[[i, 1]], lm[data[[i, 1]]] - data[[i, 2]]}, ErrorBar[0.1`]},
  {i, 1, Length[data]}], PlotStyle → PointSize[0.02]]
```

Out[137]=



This suggests that there a quadratic term might be a good thing in our fitting function.

Redo with weighted QUADRATIC fitting

Add a term to the function f to make it a quadratic:

In[138]:=
```
f[x_, a_] := a[[1]] + a[[2]] x + a[[3]] x^2
```

In[139]:=
```
Clear[x];
lm = LinearModelFit[xydata, {1, x, x^2}, x, Weights → w]
```

Out[140]= FittedModel[$0.192138 + 2.30166\ x - 0.0296824\ x^2$]

To use this information later:

In[141]:=
```
{a1, a2, a3} = lm["BestFitParameters"]
```

Out[141]= {0.192138, 2.30166, -0.0296824}

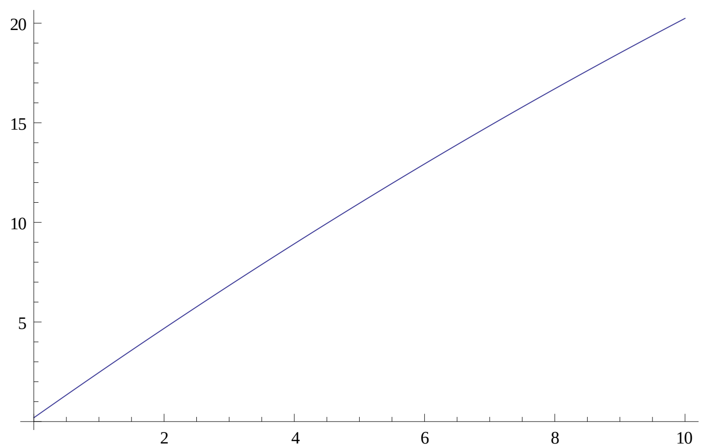
In[142]:=
```
χ2min = χ2[data, {a1, a2, a3}]
χ2min / 9
```

Out[142]= 11.5548

Out[143]= 1.28387

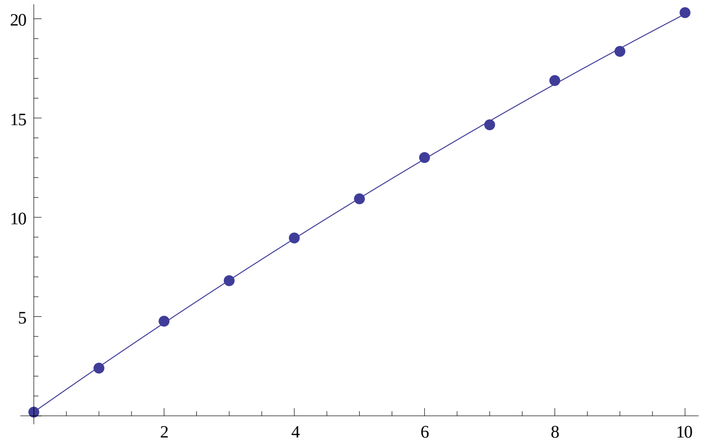
In[144]:=
```
funplot = Plot[{f[x, {a1, a2, a3}]}, {x, 0, 10}]
```

Out[144]=



In[145]:=
```
Show[dataplot, funplot]
```

Out[145]=

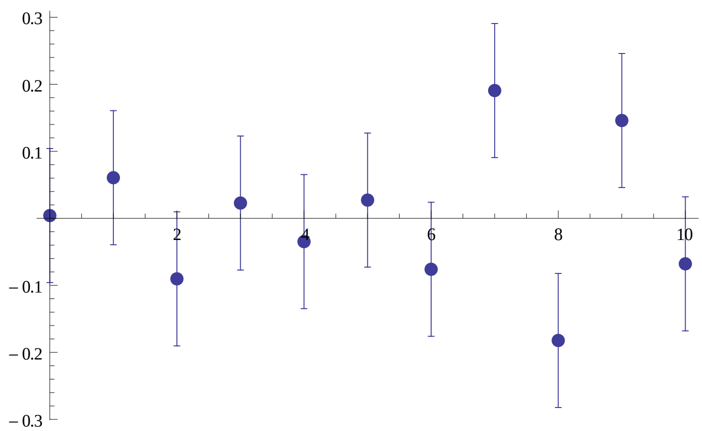


In[93]:=

Plot residuals:

In[147]:=
```
ErrorListPlot[{Table[{{data[[i, 1]], f[data[[i, 1]], {a1, a2, a3}] - data[[i, 2]]}, ErrorBar[0.1`]},
    {i, 1, Length[data]}]}, PlotStyle → PointSize[0.02], PlotRange → All]
```

Out[147]=



MUCH BETTER!

■ Using Calibration

In[148]:=
```
lm = LinearModelFit[xydata, {1, x, x^2}, x, Weights → w]
```

Out[148]= FittedModel[$0.192138 + 2.30166\ x - 0.0296824\ x^2$]

In[149]:=
```
{a1, a2, a3} = lm["BestFitParameters"]
```

Out[149]= {0.192138, 2.30166, -0.0296824}

Assume that you measure some value of y, and call this value Y, with uncertainty ΔY.

In[150]:=
```
Y = 20;
ΔY = .1 ;
```

The following gives the best X(you could use quadratic formula to find Xin this case):

In[152]:=
```
FindRoot[f[x, a] == Y, {x, 11.}]
```

Out[152]= {x → 9.85332}

In[153]:=
```
data
```

Out[153]=
```
{{0, 0.18804, 0.1}, {1, 2.40351, 0.1}, {2, 4.76703, 0.1},
 {3, 6.80721, 0.1}, {4, 8.95863, 0.1}, {5, 10.9312, 0.1}, {6, 13.0096, 0.1},
 {7, 14.6587, 0.1}, {8, 16.888, 0.1}, {9, 18.3569, 0.1}, {10, 20.3084, 0.1}}
```

Generate hypothetical data sets

In[155]:=
```
pickY[x_, σ_] := Random[NormalDistribution[x, σ]]
```

In[156]:=
```
npt = 1000;
params = Table[{0, 0, 0, 0}, {i, 1, npt}];
For[i = 1, i ≤ npt, ++i,
 {hdata = Table[{x = data[[i, 1]], pickY[f[x, {a1, a2, a3}], data[[i, 3]]], data[[i, 3]]},
    {i, 1, Length[data]}],
  hxydata = Table[{hdata[[i, 1]], hdata[[i, 2]]}, {i, 1, Length[data]}],
  hY = Random[NormalDistribution[Y, ΔY]];
  hw = Table[1 / hdata[[i, 3]]^2, {i, 1, Length[hdata]}],
  Clear[x],
  hbest = LinearModelFit[hxydata, {1, x, x^2}, x, Weights → hw] ,
  params[[i, 1]] = ha1 = hbest["BestFitParameters"][[1]],
  params[[i, 2]] = ha2 = hbest["BestFitParameters"][[2]],
  params[[i, 3]] = ha3 = hbest["BestFitParameters"][[3]],
  params[[i, 4]] = x /. FindRoot[f[x, {ha1, ha2, ha3}] == hY, {x, 9.8}]}]
```

In[104]:=
```
MatrixForm[params];
```

In[159]:=
```
a1bar = Mean[Table[params[[i, 1]], {i, 1, Length[params]}]]
a1dev = StandardDeviation[Table[params[[i, 1]], {i, 1, Length[params]}]]
a2bar = Mean[Table[params[[i, 2]], {i, 1, Length[params]}]]
a2dev = StandardDeviation[Table[params[[i, 2]], {i, 1, Length[params]}]]
a3bar = Mean[Table[params[[i, 3]], {i, 1, Length[params]}]]
a3dev = StandardDeviation[Table[params[[i, 3]], {i, 1, Length[params]}]]
Xbar = Mean[Table[params[[i, 4]], {i, 1, Length[params]}]]
Xdev = StandardDeviation[Table[params[[i, 4]], {i, 1, Length[params]}]]
```

Out[159]= 0.191733

Out[160]= 0.078764

Out[161]= 2.30221

Out[162]= 0.0363898

Out[163]= −0.02978

Out[164]= 0.00346928

Out[165]= 9.86015

Out[166]= 0.0737788

NOTE: The final uncertainty Xdev is due to uncertainty in parameters and uncertainty ΔY. You can check relative contributions by fixing hY = Y and recalculating to give uncertainty due to parameters, and using linear approx to get uncertainty due to ΔY:

(ΔX)_{parameter unnertainty} = 0.041

(ΔX)_{Y uncertainty} = ΔY/ (local slope) = .1/ (2.3 - .06 \ times X) = 1.71 = 0.058

(ΔX)_total = Sqrt[.0041^2 + 0.058^2] = 0.071