In this notebook we will examine the Eyring equation.Specifically, how to fit data to the Eyring equation properly. The Eyring equation is $k = \frac{k_b}{h} T e^{\frac{-\Delta G}{RT}}$, where $\Delta G = \Delta H - T \Delta S$

Let' s take a look at a system with theoretical parameters ΔH= 75 kJ/mol and ΔS= -15 J/(mol K).We' ll do a couple simulated "experiments" (with error) and see how different fitting strategies work.While looking at this, remember that a good fitting strategy will produce fitting parameters close to the theoretical values.

In[69]:=
```
Needs["PhysicalConstants`"];
```

In[70]:=
```
h = N[PlanckConstant][[1]];
kb = N[BoltzmannConstant][[1]];
R = N[MolarGasConstant][[1]];
```

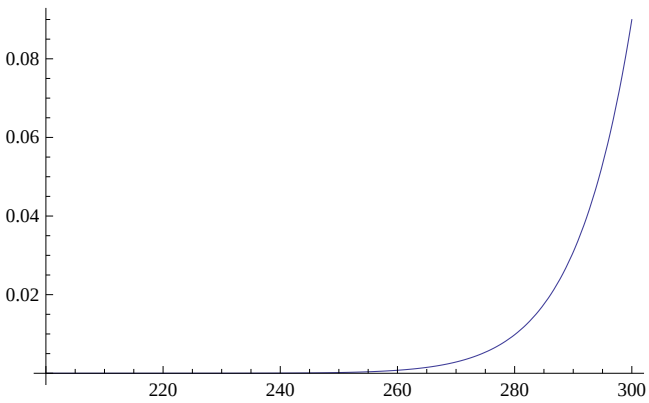
In[73]:=
```
theoΔ H= 75 * 10^3;
theoΔ S= -15;
theoParams = {Δ H→ theoΔ H, Δ S→ theoΔ S};
```

In[76]:=
```
model = kb/h T * Exp[-(Δ H- T Δ S)/(R T)];
modelParams = {Δ H, Δ S};
```

This is what the theoretical plot should look like. The log plot will actually be more informative, so we' ll stick with that.

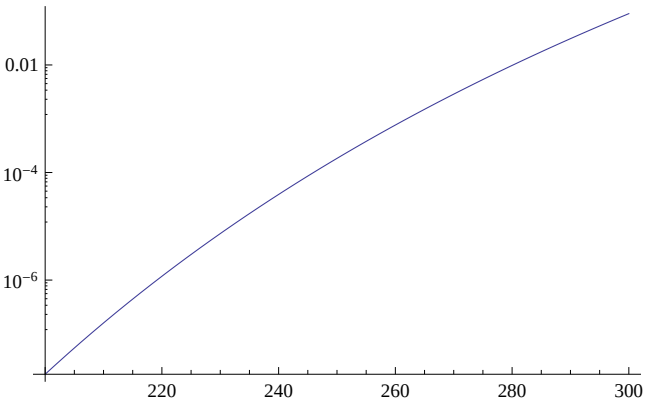
In[78]:=
```
theoPlot = Plot[model /. theoParams, {T, 200, 300}, PlotRange → All]
```

Out[78]=



In[79]:=
```
theoLogPlot = LogPlot[model /. theoParams, {T, 200, 300}]
```

Out[79]=



This is how we' ll run our experiment. Each data point will be randomly distributed within 20 % of the exact theoretical value.

In[80]:=
```
uniformRelativeError[x_] :=
 Module[{T, err, k},
  T = Range[200, 300, 10];
  k = Table[ RandomReal[{1 - x, 1 + x}] * kb/h * t * Exp[- theoΔ H - t * theoΔ S / R * t], {t, T}];
  {T, k}]
```

In[81]:=
```
{exp1T, exp1k} = uniformRelativeError[0.2];
```

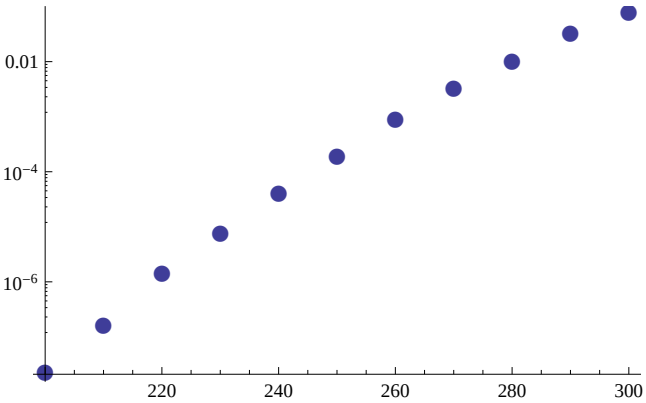
In[82]:=
```
exp1 = Transpose[{exp1T, exp1k}];
MatrixForm[exp1]
```

Out[83]//MatrixForm=

200	2.09494×10^{-8}
210	1.49817×10^{-7}
220	1.31898×10^{-6}
230	7.1939×10^{-6}
240	0.0000376397
250	0.0001751
260	0.000820417
270	0.00309937
280	0.00936587
290	0.0304909
300	0.0744224

In[84]:=
```
exp1ListLogPlot = ListLogPlot[exp1, PlotMarkers → {Automatic, Medium}]
```

Out[84]=



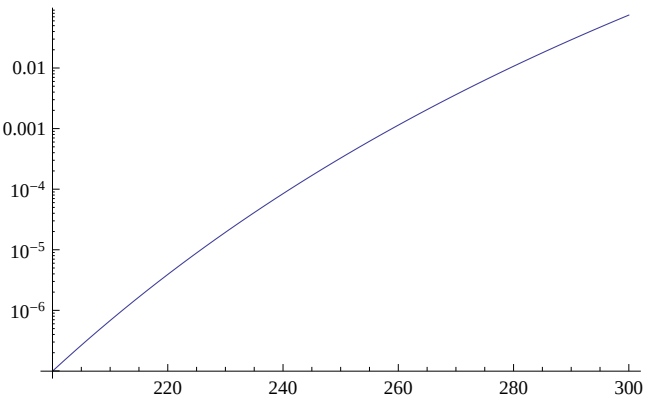
All right, we' ve got our data. Let' s start fitting. Mathematica has a non - linear fitting routine built - in, called FindFit.

In[85]:=
```
exp1Params = FindFit[exp1, model, modelParams, T, Method → Gradient]
```

Out[85]=
```
{△ H→ 65 485.4, △ S→ -48.2606}
```

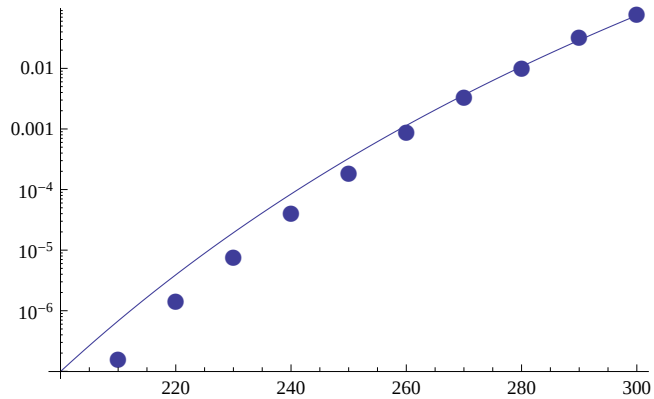
In[86]:=
```
exp1LogPlot = LogPlot[model /. exp1Params, {T, 200, 300}]
```

Out[86]=



In[87]:=
```
Show[exp1LogPlot, exp1ListLogPlot]
```

Out[87]=



As we can see, the fit isn' t great. The parameters we got are not very cloes to the theoretical values. However, Mathematica' s curve-fitting is not to blame. It is actually our whole curve-fitting strategy that is at fault. To see why, let's take a look at how we're currently approaching curve-fitting.

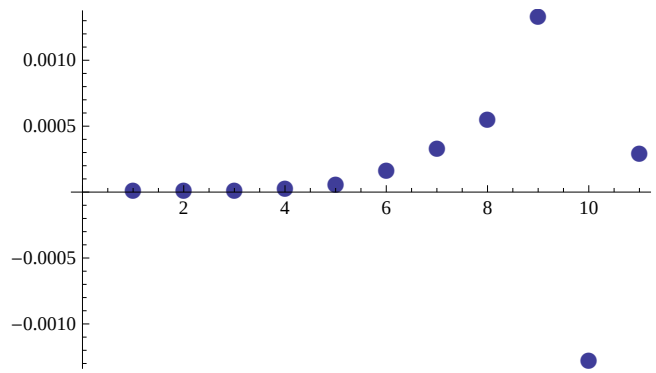
Curve-fitting is the process of getting "the best fit" to the data. In general, "the best fit" to a set of data is defined as the fit which minimizes the squared error. Let's take a look at the residuals of our fit to get a better idea of what that means.

In[88]:=
```
exp1Residuals = Table[model /. exp1Params, {T, 200, 300, 10}] - exp1[[All, 2]]
```

Out[88]=
$$\{7.82161 \times 10^{-8},\ 5.29329 \times 10^{-7},\ 2.59431 \times 10^{-6},\ 0.0000122079,\ 0.0000466913,$$
$$0.000151341,\ 0.00032003,\ 0.000537443,\ 0.00132387,\ -0.00128602,\ 0.000281554\}$$

In[89]:=
```
exp1ResidualPlot = ListPlot[exp1Residuals, PlotMarkers → {Automatic, Medium}]
```

Out[89]=



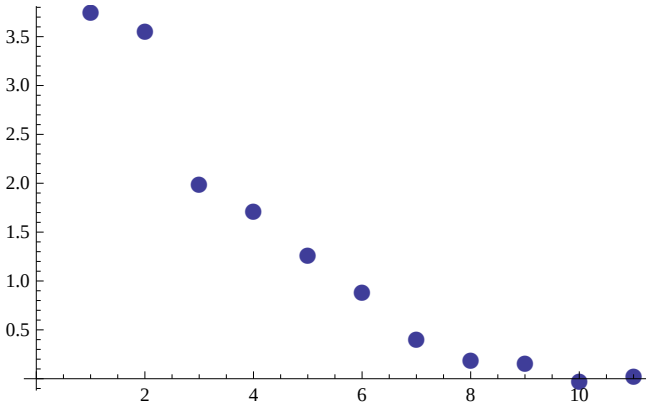
Here we can that the errors are, in general, very small (none of them are larger than 0.001 or so). The "squared error" that we mentioned previously is just the sum of the squares of the residuals (i.e., $\Sigma\ r^2$). Mathematica picked out Δ H and Δ S such that the squared error was as small as possible.

That is not what we want.

Our experiment had *relative* error. That is, each data point was within 20% of the correct value. Let's take a look at the *relative* error of the fit that Mathematica gave us.

In[90]:=
```
ListPlot[exp1Residuals / exp1[[All, 2]], PlotMarkers → {Automatic, Medium}]
```

Out[90]=



The relative error is *terrible*. Because Mathematica tried to minimize the *absolute* error, it cared much more about getting the later points close. An 80% error at low temperatures is barely even registered as important (but it obviously should be).

If we want to get a good fit for our experiment, we really want to minimize the squared relative error instead of the squared absolute error. Luckily, Mathematica has an easy way to do that. We can just give it a different function to try and minimize.

In[91]:=
```
weightedNorm[residuals_] := Norm[residuals / exp1k]
```

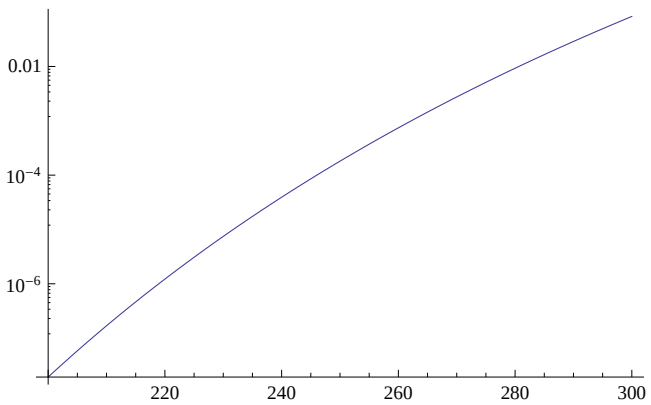
With this new weighted function, let's try doing the fit again.

In[92]:=
```
exp1WeightedParams =
 FindFit[exp1, model, modelParams, T, NormFunction → weightedNorm, Method → Gradient]
```

Out[92]=
```
{△ H→ 74 222.8, △ S→ -18.2268}
```

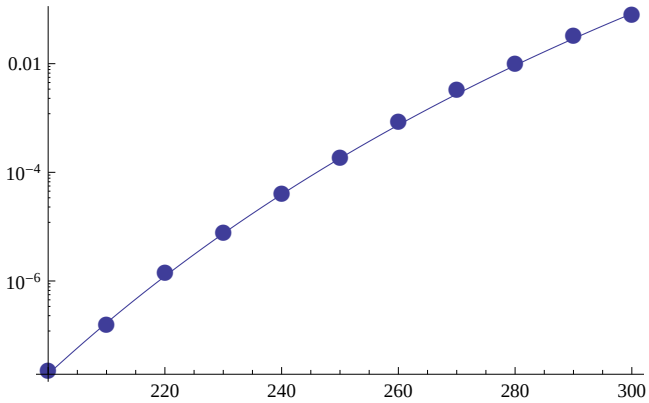
In[93]:=
```
exp1WeightedLogPlot = LogPlot[model /. exp1WeightedParams, {T, 200, 300}]
```

Out[93]=



In[94]:=
```
Show[exp1WeightedLogPlot, exp1ListLogPlot]
```

Out[94]=



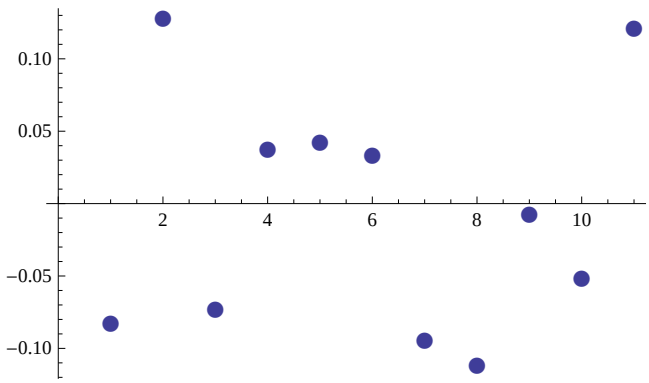
Much better. The fit looks great and the parameters we got are very close to the expected values. Let's take a look at the percent error again.

In[95]:=
```
exp1WeightedResiduals = Table[model /. exp1WeightedParams, {T, 200, 300, 10}] - exp1k
```

Out[95]=
$$\{-1.75357\times10^{-9},\ 1.90218\times10^{-8},\ -9.76426\times10^{-8},\ 2.59092\times10^{-7},\ 1.54841\times10^{-6},\ 5.63254\times10^{-6},\ -0.0000782189,\ -0.000350352,\ -0.0000806477,\ -0.00161831,\ 0.00891345\}$$

In[96]:=
```
ListPlot[exp1WeightedResiduals / exp1[[All, 2]],
 PlotMarkers → {Automatic, Medium}, PlotRange → All]
```

Out[96]=



Roughly randomly distributed, and almost all of them are within 20 %. As we have seen, choosing the correct curve-fitting strategy, based on the type of experiment that was run, is very important. Even good data will yield poor results if it' s not analyzed correctly.

As an historical aside, note that the amount of computation required to do this non-linear fitting was completely out of reach before the 1960s or so. Luckily for everyone involved, most commonly used equations in chemistry can be linearized. For instance, let's take the log of both sides of the Eyring equation:

$$\log k = \log\left[\frac{k_b}{h}\right] + \log[T] - \frac{\Delta H - T\Delta S}{R\,T},$$

which implies that

$$\log\left[\frac{k}{T}\right] = -\frac{\Delta H}{R}\frac{1}{T} - \frac{\Delta S}{R} + \log\left[\frac{k_b}{h}\right],$$

which is wonderful. We can calculate the stuff on the left (as we have both *k* and *T*) and then we can plot it versus $\frac{1}{T}$, leading to a simple *linear* curve-fitting problem.

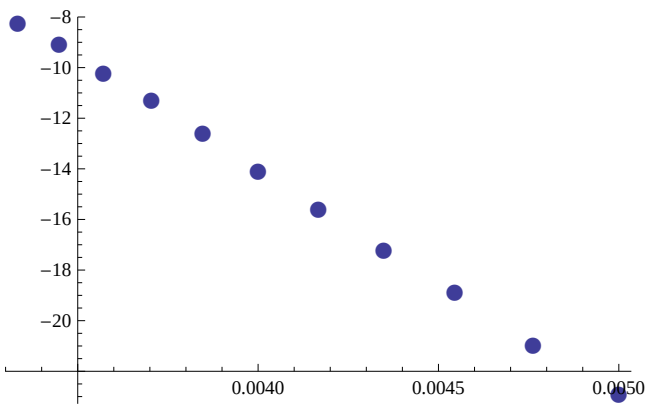
In[97]:=
```
exp1Linear = Transpose[{1 / exp1T, Log[exp1k / exp1T]}];
MatrixForm[exp1Linear]
```

Out[98]//MatrixForm=

$$\begin{pmatrix} \frac{1}{200} & -22.9795 \\ \frac{1}{210} & -21.061 \\ \frac{1}{220} & -18.9323 \\ \frac{1}{230} & -17.2804 \\ \frac{1}{240} & -15.6681 \\ \frac{1}{250} & -14.1716 \\ \frac{1}{260} & -12.6664 \\ \frac{1}{270} & -11.375 \\ \frac{1}{280} & -10.3055 \\ \frac{1}{290} & -9.16021 \\ \frac{1}{300} & -8.30178 \end{pmatrix}$$

In[99]:=
```
ListPlot[exp1Linear, PlotMarkers → {Automatic, Medium}]
```

Out[99]=



Now that we' ve linearized the data, we can do a much easier *linear* curve-fit using Mathematica's linear fitting routine, Fit. We're looking for the constant coefficient (which will be $\frac{\Delta S}{R} + \log\left[\frac{k_b}{h}\right]$) and the linear coefficient (which will be $-\frac{\Delta H}{R}$).

In[100]:=
```
exp1LinearFit = Fit[exp1Linear, {1, x}, x]
```

Out[100]= 21.5738 − 8926. x

In[101]:=
```
{constCoeff, linearCoeff} = CoefficientList[exp1LinearFit, x]
```

Out[101]= {21.5738, −8926.}

In[102]:=
```
Solve[{ΔS/R + Log[kb / h] == constCoeff, -ΔH/R == linearCoeff}, {ΔH, ΔS}]
```

Out[102]= {{ΔS → −18.1772, ΔH → 74215.}}

Interestingly, the linear method gives very good results, with significantly less computational effort (this type of linear fit could be done on a handheld calculator).