Maxima by Example:
Ch.14: Fitting a Model Function to Data *

Edwin L. (Ted) Woollett

January 15, 2017

Contents

*This version uses Maxima 5.36.1 for Windows. This is a live document. Check `http://www.csulb.edu/~woollett/` for the latest version of these notes. Send comments and suggestions to `woollett@charter.net`

This document is Ch. 14 of the series “Maxima by Example” and is made available via the author’s webpage **http://www.csulb.edu/~woollett/** to aid new users of the Maxima computer algebra system.

Ch.14 files used in the examples, available on the author’s webpage, include the chapter software file **fit.mac**, **qdraw.mac**, the data files **mbe14-fit1.mac** through **mbe14-fit8.mac**, and **coffee.dat**.

Most of the plots in Ch. 14 use our **qdraw.mac** software discussed in more detail in Ch. 13.

The interface XMaxima was used with the Windows XP operating system, with the startup file looking like **C:/Documents and Settings/Edwin Woollett/maxima/maxima-init.mac**.
If you are using Windows 7, the startup file path looks like
C:/Users/ted/maxima/maxima-init.mac.
See Chapter 1 for more information about setting up the startup file.

1 Introduction

Chapter 14 provides examples of the use of a new set of Maxima functions (defined in **fit.mac**). These new functions not only return the best-fit parameter values, but also the estimated parameter uncertainties and the χ^2 probability of the results. These new functions assume one independent variable and one corresponding dependent variable and are called **fit_line**, **fit_slope**, **fit_y_intercept**, **lfit**, and **nlfit**. These new functions also allow for the use, while finding best fit model parameters, of the estimated uncertainties of the measured dependent variable.

Before introducing these new functions, we remind the reader of the currently available Maxima function **lsquares_estimates**.

After introducing the syntax of the new **fit.mac** fitting functions, we work out nine examples in detail. Seven of these examples use experimental data from the text "Data Reduction and Error Analysis for the Physical Sciences," 3rd ed., Philip R. Bevington and D. Keith Robinson, McGraw-Hill (US), 2003.

A "New International Economy Edition" of this text, printed in India, can be found on the **amazon.com** website. This text is very valuable because of the in-depth approach and the many examples discussed in a physical context. We include links to online pdf copies of this text, as well as other suggested resources in the References section at the end of this chapter.

As a quick survey of using the available fitting functions, Example 1 uses our five fitting functions **fit_line**, **fit_slope**, **fit_y_intercept**, **lfit**, and **nlfit**, three of the auxiliary functions, and also **lsquares_estimates** to fit a simple set of data.

Data files included with Ch. 14 are **mbe_fit1.dat** through **mbe_fit8.dat**, in addition to **coffee.dat**. The latter data file was also used in Ch. 2 with a brief example of using **lsquares_estimates**.

Prior to Example 5 we show how to generate random numbers, histograms, and add Gaussian noise to a signal, using the standard Maxima packages **descriptive.mac** and **distrib.mac**.

In Example 7 we use Legendre polynomials as basis functions in defining a data model. We use the Maxima function **legendre_p(n,x)** which is defined in **orthopoly.lisp**. We set **orthopoly_returns_intervals** to **false** in **fit.mac** so that we get an ordinary number as the return value.

Most of the plots are created using our Ch. 13 software **qdraw.mac**, which provides a simple interface to the **draw2d** function. We have included **qdraw.mac** with the Ch. 14 files for convenience; it should be placed in your Maxima work folder along with the data files and **fit.mac**. The first few Examples provide enough guidance in the use of **qdraw** in the context of this chapter. In particular, in Example 1 we draw the same plot (approximately) using both **qdraw** and **draw2d** separately for comparison.

2 The Currently Available Maxima Function lsquares_estimates

The currently available Maxima least squares fit functions include **`lsquares_estimates`**, which is more general than the functions in **`fit.mac`**, in the sense that **`lsquares_estimates`** does not restrict the number of variables and does not assume one variable is the "independent variable." However, no estimates of the uncertainties of the fitted parameters is provided.

The Maxima manual has the description:

```
lsquares_estimates (D, x, e, a)
lsquares_estimates (D, x, e, a, initial = L, tol = t, iprint = [n1,n2])

Estimate parameters a to best fit the equation e in the variables x
 and a to the data D, as determined by the method of least squares.
 lsquares_estimates first seeks an exact solution, and if that fails,
 then seeks an approximate solution.

The return value is a list of lists of equations of the form [a = ..., b = ..., c = ...].
 Each element of the list is a distinct, equivalent minimum of the mean square error.

The data D must be a matrix. Each row is one datum (which may be called a 'record'
 or 'case' in some contexts), and each column contains the values of one variable across
 all data. The list of variables x gives a name for each column of D, even the columns
 which do not enter the analysis. The list of parameters a gives the names of the parameters
 for which estimates are sought. The equation e is an expression or equation in the variables
 x and a; if e is not an equation, it is treated the same as e = 0.

Additional arguments to lsquares_estimates are specified as equations and passed on
 verbatim to the function lbfgs which is called to find estimates by a numerical method
 when an exact result is not found.

If some exact solution can be found (via solve), the data D may contain non-numeric values.
 However, if no exact solution is found, each element of D must have a numeric value.
 This includes numeric constants such as %pi and %e as well as literal numbers (integers,
 rationals, ordinary floats, and bigfloats). Numerical calculations are carried out with ordinary
 floating-point arithmetic, so all other kinds of numbers are converted to ordinary floats for calculations.

load(lsquares) loads this function.
```

For information about the use of the option **`iprint = [n1,n2]`**, see the Maxima manual entry under the name **`lbfgs`**. For a long set of examples, see the comments at the top of **`.../share/lsquares/lsquares.mac`**. These latter examples also illustrate the use of the options **`initial = L`** and **`tol = t`**.

See Example 1 for an example of using **`lsquares_estimates`**.

3 Syntax of the fit.mac Functions

In the following functions, **Mdata** is a two column matrix, with the first column containing the values of the independent variable at which values of the dependent variable have been measured, and with the second column containing the corresponding values of the dependent variable. **`length(Mdata)`** will then produce the number of data points (the number of rows of the matrix **`Mdata`**).

sigL is a list of the estimated experimental errors of the dependent variable, with a separate number for each measured value; this list should have the same length as the number of data points. We ignore any measurement errors in the values of the independent variable x_i, assuming such possible errors are much smaller than those of the corresponding dependent variable y_i. If you have no estimate of the y_i experimental errors, you can still find a set of approximate numerical values of the parameters in your model by defining **`sigL`** to be a list of 1's. **`sigL : makelist (1,i,1,length (Mdata))`**.

Since we ignore the uncertainties of the independent variable values x_i, the uncertainty in the value found for the parameters depends only on the uncertainties σ_i of the dependent variable measured values y_i. We need some approximate estimate of these uncertainties to find reliable uncertainties in the fitted values of the model parameters as well as a reliable value of the χ^2 probability Q (the goodness-of-fit fractional probability). Q is the (fractional) probability that a value of χ^2 (pronounced "chi-square") greater than the value calculated from the data would be produced in a repetition of the same experiment.

Once you have found a set of fitted values of the model parameters, you can use the function **`y_gaussian_PE`** (see Example 1 for an example) to produce a value of the probable error of the dependent variable measurements, provided values drawn at a fixed value of the independent variable are drawn from a Gaussian distribution (more about this later), allowing a recalculation with a new **`sigL`** which will provide much better estimates of the model parameter uncertainties.

`ymodel` is an expression depending on some parameters and an independent variable, such as **`a + b*x`**, in which **`[a,b]`** are model parameters, and **`x`** is the independent variable, and the model is a two parameter fit to a straight line. If your measurements are taken at different temperatures **`T`**, for example, you would use for ymodel **`a + b*T`**. You can use any symbols for the unknown parameters, such as **`a1 + a2*x`**. A general linear fit model might be, for example **`a + b*cos(x) + c*exp(-x)`**, which contains three parameters, and this model expression is linear in each of the three parameters. The function **`lfit`** (or **`nlfit`**) should be used for such a general linear model. Both of the functions **`lfit`** and **`nlfit`** (the latter being the general non-linear fit function), can also fit a model containing terms which don't involve parameters, such as:
`cos(x)/x^3 + a*sin(x) + b*exp(-x)`. The non-linear fit function **`nlfit`** can be used with both linear and non-linear models. An example of a non-linear model (non-linear in at least one of the parameters) is **`a*exp(-b*x) + c*cos(d*x)`**, which has two linear parameters and two non-linear parameters.

The argument **`paramL`** is a list of the model parameters, such as **`[a,b]`** or **`[a1,a2]`**, etc. The functions detect the name being used for the independent variable from the expression used for **`ymodel`**.

The non-linear fitting function **`nlfit`** requires as its last argument the list **`param-initL`**; for example, if the fit is a two parameter fit, **`[1,-1]`** would be a list of the initial values of the two parameters. An example would be:
`nlfit (dataM, sL, a*exp(-b*x),[a,b],[1,-1])` .

The function **`y_gaussian_PE`** has as its second argument **`dof`**, the "number of degrees of freedom" of the fit, which is equal to the number of data points minus the number of parameters being fitted.

Each of our "fitting functions" searches for values of the parameters which produce a locally smallest numerical value of the non-negative number χ^2, defined as

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i, \mathbf{a})}{\sigma_i} \right)^2 , \tag{3.1}$$

in which y_i is the measured value at $x = x_i$, with an estimated uncertainty given by σ_i, $\mathbf{a}$ stands for the set of model parameters, and $y(x_i, \mathbf{a})$ is the model prediction for the value of the dependent variable at $x = x_i$. We ignore any measurement errors in the values of the x_i, assuming such possible errors are much smaller than those of the corresponding y_i.

A necessary condition for the existence of a local minimum of $\chi^2(\mathbf{a})$ is that the first derivative of χ^2 with respect to each of the parameters is equal to zero, a requirement that yields the same number of equations as the

number of parameters.

Each of our fitting functions returns the list **`[paramL, errorL,chi2,Q]`**, in which **`chi2`** stands for χ^2 (evaluated for the values of the fitted parameters), and **`Q`** stands for the "χ^2 fractional probability", the (fractional) probability a repetition of the data measuring experiment (starting with the same environment and initial conditions) would produce a value of χ^2 as large as the value found here. The "percent probability" is 100 times the value of Q. (See the details and derivations section for more background.) Each of our fitting functions also print to the screen the values of the ratio of χ^2 to the number of degrees of freedom, as well as the value of **`Q`**.

Given numerical values for the model parameters, you can independently calculate χ^2 using the function **`get_chi2`**, whose third and last argument **`yfit_expr`** might be **`1.2 - 3.4*x`** for a straight line fit **`a + b*x`**, in which the y-intercept **`a = 1.2`**, and the slope **`b = -3.4`**.

You can also independently reproduce the screen printouts of **`chi2/dof`** and **`Q`** using the **`chi2_prob`** function.

One can show (see details and derivations section) that if the "reduced chi-square"

$$\chi^2_\nu \equiv \frac{\chi^2}{\nu} \equiv \frac{\chi^2}{\text{dof}} \tag{3.2}$$

is of the order 1, then $Q \approx 0.5$, and both measures indicate a good fit to the data. For a straight line fit in which two parameters must be adjusted using N data points, the "number of degrees of freedom" $\nu = \text{dof} = N - 2$.

Quoting (loosely) Numerical Recipes (1992, Sec. 15.2)

> ...If Q is larger than, say, 0.1 (i.e., the goodness-of-fit probability is greater than 10%), then the goodness-of-fit is believable. If it is larger than, say, 0.001 (i.e., the goodness-of-fit probability is larger than 0.1%), then the fit *may* be acceptable if the errors are nonnormal [non-gaussian] or have been moderately underestimated. If Q is less than 0.001 then the model and/or estimation procedure can rightly be called into question.

For convience in checking syntax, we insert here the syntax of the most useful functions defined in the file **`fit.mac`**.

4 Example 1: Straight Line Fit of Data with No Error Information

As a quick survey of using the available fitting functions, Example 1 uses our five fitting functions **`fit_line`**, **`fit_slope`**, **`fit_y_intercept`**, **`lfit`**, and **`nlfit`**, as well as three of the auxiliary functions to fit a simple set of data.

In addition, we compare our fit results with Maxima's **`lsquares_estimates`** output, which can find the best fit parameters (ignoring relative weights of various data points) but does not return information about the estimated uncertainties of the parameters found. It is important to realize that if the estimated errors σ_i of the values of the measured quantity y_i at each x_i are assumed to be the same in magnitude and sign, then **`lsquares_estimates`** should return the same values for the model parameters as our fit functions. If the values of the σ_i are not all the same, as is the case for measurements y_i controlled by Poisson statistics for example, then the values returned for the parameters are not expected to be identical.

Example 1 is taken from Bevington (Data Reduction and Error Analysis for the Physical Sciences,1st. ed, 1969, p. 93-94). The data in our file **`mbe14-fit1.dat`** describes measurements of temperature T along a rod in degrees Celsius (column 2) at positions x in centimeters along the rod (column 1).

We will try to use a straight line fit $T = a + b\,x$, to the given data, in which a is the prediction of the temperature when $x = 0$, and b is the rate of change in temperature (degrees per cm.) along the rod. Generically, we call these respectively the "y-intercept" and the "slope" of the best fit line.

No errors in temperature measurement are available, so we define **`sigL`** to be a list of 1's when calling **`fit_line`**, **`fit_slope`**, **`fit_y_intercept`** (the three functions which are restricted to straight line fits),

or when calling **lfit** (general linear fit) or **nlfit** (general non-linear fit) for the first time.

In the absence of any additional information, we expect any temperature measurement (as a function of distance or time, etc) to be subject to both instrumental and random Gaussian errors, and we use **y_gaussian_PE** to estimate the size of temperature measurement errors based on both the given data as well as the best fit values of the straight line parameters found using the stop-gap device of assuming **sigL** to be a list of 1's.

We can then re-do our calculation of the best fit straight-line parameters, and the resulting estimates returned of the probable error size of those parameters should be more reasonable than ocurred in the first go. In addition, the values of the χ^2 probability Q and the value of the "reduced" χ^2 value can finally be taken seriously.

The author's work folder for this chapter is **c:/work9/**, and the various chapter data files are available for use there, as well as the file **fit.mac**. Because the Maxima startup file **maxima-init.mac** has been edited (or created) to let Maxima know the location of the current work folder (see the first chapter of Maxima by Example for a discussion of this issue), we can use **load(fit)** instead of **load("fit.mac")** or the even more onerous **load ("c:/work9/fit.mac")** to aquaint Maxima with the functions defined in that software file.

At the bottom of **fit.mac** appears a reset of some global variables which proves convenient for our work in this chapter:

```
ratprint : false$
orthopoly_returns_intervals : false$
display2d : false$
fpprintprec : 6$
```

The actual calculations are performed using 16 digit floating point arithmetic, as usual with Maxima in its default mode.

We prefer to use the interface XMaxima for routine work, and the setting **display2d : false** allows a denser display of information per screen. You are, of course, free to change this setting in **fit.mac** at any time. (And free also to change the definition of any of our functions!)

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/mbe14-fit1.dat"$
(%i3) printfile (fname)$
1.0 15.6
2.0 17.5
3.0 36.6
4.0 43.8
5.0 58.2
6.0 61.6
7.0 64.2
8.0 70.4
9.0 98.8
(%i4) Mdata : read_matrix (fname);
(%o4) matrix([1.0,15.6],[2.0,17.5],[3.0,36.6],[4.0,43.8],[5.0,58.2],
              [6.0,61.6],[7.0,64.2],[8.0,70.4],[9.0,98.8])
(%i5) ndata : length (Mdata);
(%o5) 9
```

The object **Mdata** (you can use any name for this) is a Maxima matrix whose first column contains values x_i of the "independent variable" x (distance along the rod), and the second column contains corresponding values T_i of the "dependent variable" T (the temperature), and each of the nine rows describes one "data point" (x_i, T_i).

4.1 Two Parameter Fit using fit_line

We first use **`fit_line(data_matrix, error_list)`**, a function that actually calls the general linear fit function **`lfit`**. Initially we don't have error estimates for the temperature values, so just set all the temperature errors equal to 1 for now.

```
(%i6) sigL1 : makelist(1,i,1,ndata);
(%o6) [1,1,1,1,1,1,1,1,1]
(%i7) out : fit_line (Mdata,sigL1);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
 num_data =  9
 num_param =  2
 dof =  7
 chi2/dof =  45.2369
 chi2_prob =  1.67021e-62 %
a =  4.81389  +/-  0.726483
b =  9.40833  +/-  0.129099
(%o7) [[a = 4.81389,b = 9.40833],[0.726483,0.129099],316.658,1.67021e-64]
(%i8) yfit : a + b*x, out[1];
(%o8) 9.40833*x+4.81389
```

Each of our fitting functions returns the list **`[paramL,errorL,chi2,Q]`**, in which **`paramL`** is a list of the best-fit parameter values (including parameter names), **`errorL`** is a list of the corresponding parameter uncertainties, **`chi2`** stands for χ^2 (evaluated for the values of the fitted parameters), and **`Q`** stands for the "χ^2 fractional probability", the (fractional) probability a repetition of the data measuring experiment (starting with the same environment and initial conditions) would produce a value of χ^2 as large as the value found here. The **percent** probability is 100 times the value of Q.

For this example, the number of degrees of freedom (dof) is the number of data points (9) minus the number of fitted parameters (2) which gives **`dof = 7`**. The value of the "reduced χ^2", **`chi2/dof`** should be of the order of 1 for a really good fit, and we see a poor fit.

The χ^2 probability Q is ridiculously small, again indicating a really poor fit. You can independently calculate these quantities using **`chi2_prob(chi2,dof)`**.

```
(%i9) chi2_prob (317,7);
 chi2/dof =  45.2857
 chi2_prob =  1.4115e-62 %
(%o9) done
```

We now use the returned values of the parameters to estimate the temperature "probable errors" if they have a gaussian distribution, using the function **`y_gaussian_PE (data-matrix, dof, yfit_expr)`**, define a new list of estimated errors, and call **`fit_line`** again.

```
(%i10) y_gaussian_PE (Mdata,7, yfit);
(%o10) 4.48389
(%i11) sigL2 : makelist(4.5,i,1,ndata);
(%o11) [4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5]
(%i12) out : fit_line (Mdata,sigL2);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
 num_data =  9
 num_param =  2
 dof =  7
 chi2/dof =  2.23392
 chi2_prob =  2.86433 %
a =  4.81389  +/-  3.26917
b =  9.40833  +/-  0.580948
(%o12) [[a = 4.81389,b = 9.40833],[3.26917,0.580948],15.6374,0.0286433]
```

```
(%i13) yfit : a + b*x, out[1];
(%o13) 9.40833*x+4.81389
```

The output indicates that the best fit straight line $T = a + b\,x$ has parameter estimates $a = 4.8\,^{\circ}\mathrm{C}$, $b = 9.4\,^{\circ}\mathrm{C/cm}$, with probable parameter uncertainties $\sigma_a = 4.9\,^{\circ}\mathrm{C}$, $\sigma_b = 0.87\,^{\circ}\mathrm{C/cm}$. The "probable error" of any one of the temperatures, based on this data set, is $\mathrm{P.E.} = 4.5\,^{\circ}\mathrm{C}$. Let σ_{pe} represent the probable error. Then if one takes **repeated** independent measurements of y_i for a fixed value of x_i, roughly 50% of the values will lie in the range $(\bar{y} - \sigma_{pe}, \bar{y} + \sigma_{pe})$, where $\bar{y}$ is the arithmetic mean of the y_i values taken for fixed x_i.

We can independently check the value of χ^2 using **`get_chi2`**:

```
(%i14) get_chi2(Mdata,sigL2,4.81 + 9.41*x);
(%o14) 15.6375
```

4.2 Plots

We prefer to use our **`qdraw.mac`** graphical package for simple plots. (Other Maxima options are **`plot2d`** and **`draw2d`**.) For more details about **`qdraw`**, see Maxima by Example, Ch. 13. We proceed to make a simple plot of the data points, the best fit straight line, and simple error bars based on the value reported for the probable errors (P.E.). So you can compare using **`qdraw`** with direct use of **`draw2d`**, we use both plot methods in this first example. To use the **`qdraw`** plotting interface, you must load both **`draw.lisp`** as well as **`qdraw.mac`**. To load the qdraw package file, you can just use **`load(qdraw)`** if you have the file in your work folder and have set up your file search paths as described in Chap. 1. Otherwise, if your work folder is (for example) **`c:\work9`**, and you have placed **`qdraw.mac`** in that folder, use **`load("c:/work9/qdraw.mac")`**.

```
(%i15) load(draw);
(%o15) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i16) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o16) "c:/work9/qdraw.mac"
(%i17) plist : read_nested_list (fname);
(%o17) [[1.0,15.6],[2.0,17.5],[3.0,36.6],[4.0,43.8],[5.0,58.2],[6.0,61.6],
        [7.0,64.2],[8.0,70.4],[9.0,98.8]]
(%i18) qdraw ( ex1 (yfit, x, 0,10),pts (plist,pc(black),ps(1)),
             key (bottom), errorbars (plist,4.5, lw(3), lc(black) ),
             yr (-10,110) )$
```

which produces the plot

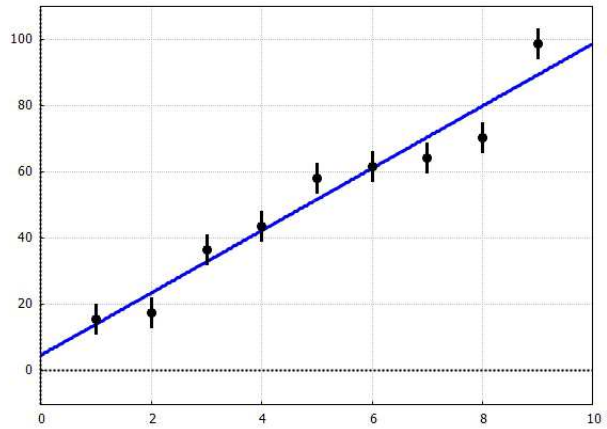

Figure 1: Data Points and Best Fit Line: T vs. x, using qdraw

We can also use **draw2d** directly, without the **qdraw** interface, making use of the **draw2d errors** element. We first need to append the probable error 4.5 to the end of each data point sublist in **plist** using Maxima's **lambda** function.

```
(%i19) pelist : map (lambda ([pL], append (pL,[4.5])), plist);
(%o19) [[1.0,15.6,4.5],[2.0,17.5,4.5],[3.0,36.6,4.5],[4.0,43.8,4.5],[5.0,58.2,4.5],
        [6.0,61.6,4.5],[7.0,64.2,4.5],[8.0,70.4,4.5],[9.0,98.8,4.5]]
(%i20) draw2d ( yrange = [-10,110], xaxis=true, xaxis_width=2,grid=true,
                  line_width=3,color=blue, explicit( yfit,x,0,10),
                 color=black, errors (pelist) )$
```

which produces a plot similar to that produced by **qdraw**:

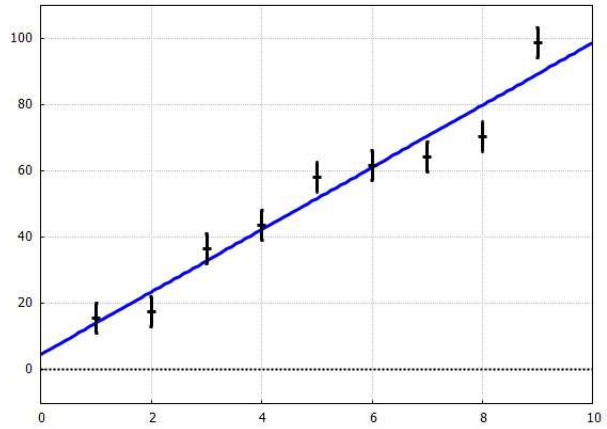

Figure 2: Data Points and Best Fit Line: T vs. x, using draw2d directly

4.3 Two Parameter Fit using lfit(dataM,sL,ymodel,pL)

The general linear fit function **lfit** can, of course, be called directly for this straight line fit.

```
(%i21) out : lfit (Mdata,sigL2,a + b*x,[a,b]);
 ivar =  x
 num_data =  9
 num_param =  2
 dof =  7
 chi2/dof =  2.23392
 chi2_prob =  2.86433 %
a =  4.81389  +/-  3.26917
b =  9.40833  +/-  0.580948
(%o21) [[a = 4.81389,b = 9.40833],[3.26917,0.580948],15.6374,0.0286433]
```

and the results are identical.

4.4 Two Parameter Fit using nlfit(dataM,sL,ymodel,pL,pgL)

We can also try out the general nonlinear fit function **nlfit**:

```
(%i22) nlfit (Mdata,sigL2,a + b*x,[a,b],[1,1]);
 Ndata =  9
 Nparam =  2
 dof =  7
 ivar =  x
start:   params:    [a = 1.0,b = 1.0]        chi2 =  1159.67
---------------------------------------------------------------
  n     lam
  1    0.001
```

```
              p_oldL =  [1.0,1.0]
              p_newL =  [4.99374,9.37156]    chi2_new =  15.6414
  2    1.0e-4
              p_oldL =  [4.99374,9.37156]
              p_newL =  [4.81577,9.408]    chi2_new =  15.6374
------------------------------------------------------------
 chi2/dof =  2.23392
 chi2_prob =  2.86433 %
------------------------------------------------------------
a =  4.81577  +/-  3.254
b =  9.408  +/-  0.578252
(%o22) [[a = 4.81577,b = 9.408],[3.254,0.578252],15.6374,0.0286433]
```

with parameter values the same to three decimal places.

4.5 Two Parameter Fit using lsquares_estimates

`lsquares_estimates` first tries to find an exact solution, using **`solve`**, and this is the result:

```
(%i23) load(lsquares);
(%o23) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/lsquares/lsquares.mac"
(%i24) lsquares_estimates(Mdata,[x,y], y = a + b*x,[a,b]);
(%o24) [[a = 1733/360,b = 1129/120]]
(%i25) float(%);
(%o25) [[a = 4.81389,b = 9.40833]]
```

which agrees with our **`fit_line`** and **`lfit`** values for the parameters.

4.6 One Parameter Fit using fit_slope

If we assume from the start that we want a fit with the y-intercept having the value 4.81, we can use **`fit_slope`**:

```
(%i26) fit_slope (Mdata,sigL2,4.81);
 fit model y(x) =  4.81  + b*x to given data
 ivar =  x
 num_param =  1
 num_data =  9
 dof =  8
 chi2/dof =  1.95468
 chi2_prob =  4.78735 %
b =  9.40895  +/-  0.266557
(%o26) [[b = 9.40895],[0.266557],15.6374,0.0478735]
```

4.7 One Parameter Fit using fit_y_intercept

If we want a fit in which the slope is required to be 9.41, then we can use **`fit_y_intercept`**:

```
(%i27) fit_y_intercept (Mdata,sigL2,9.41);
 fit model y(x) = a + ( 9.41 )*x to given data
 ivar =  x
 num_param =  1
 num_data =  9
 dof =  8
 chi2/dof =  1.95468
 chi2_prob =  4.78733 %
a =  4.80556  +/-  1.5
(%o27) [[a = 4.80556],[1.5],15.6374,0.0478733]
```

5 Example 2: Straight Line Fit of Data with Poisson Error Statistics

This example is taken from the first edition (1969) of Bevington's text "Data Reduction and Error Analysis for the Physical Sciences" (p. 95 - 97); see the References section at the end of this chapter.
Quoting Bevington:

> Consider a counting experiment in which we count the number of events recorded in a detector as a function of time. We have a source which is emitting radiation, and the number of counts per unit time from our detector is a measure of the rate at which this radiation is being emitted. We observe qualitatively that the rate of emission is decreasing approximately linearly with time and we wish to describe this quantitatively.
>
> We cannot determine the counting rate instantaneously because no counts will be detected in an infinitesimal time interval. But we can determine the number of counts C detected over a time interval Δt, and this should be representative of the average counting rate over that interval. ... it is customary and convenient to make the intervals equally spaced in time as well as equally long.
>
> In this example, the intervals are both equal $\Delta t_i = \Delta t$ and contiguous $\Delta t_i = t_{i+1} - t_i$; the times t_i at which the successive intervals start are given by $t_i = (i-1)\,\Delta t$, with time measured from the beginning of the first interval.

The data in **`mbe14-fit2.dat`** describes ten measurements of the number of counts C_i per 15 sec as a function of time. The first column is the time (in sec) of the beginning of each 15 sec interval.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/mbe14-fit2.dat"$
(%i3) printfile (fname)$
0 106
15 80
30 98
45 75
60 74
75 73
90 49
105 38
120 37
135 22
```

Quoting from Sec 15.02 of Nuclear Radiation Physics by Ralph Lapp and Howard Andrews (2nd. edition, 1954)

> **Statistical fluctuation**
> When radiation measurements are made, it is observed that all readings show fluctuations. This behavior is not always due to the instability of the measuring instrument but is inherent in the nature of radiation sources. Each nuclear disintegration is a completely random and independent process. Such a random process will obey the laws of statistics, which predict that, even though there is a definite average rate of disintegration, the number actually counted in a given time will show deviations from this average.
>
> The "true value" of a count can be obtained as the arithmetic mean of a very large number of observations, if proper care is taken to keep all experimental conditions constant. If the fluctuations of individual observations about the true value have a normal or Poisson distribution, the standard deviation σ of a single observation of N counts will be
>
> $$\sigma = \sqrt{N} \tag{5.1}$$

Since we have measurements of the counting rate as a function of time in this example, if we performed many repetitions of this experiment, with an identically prepared radioactive source, and kept track of the number of counts per 15 sec. starting after one minute of time for example, the numbers would approximately obey a Poisson distribution of values.

After creating a Maxima matrix **Mdata** from the data file, we can then obtain a list of just the dependent variable $y_i = C_i$ values (count-rate, counts per 15 sec.), using **list_matrix_entries** together with the **col** function. We can then take the square root of these values to get the estimated statistical errors σ_i of the individual count rate values, thus defining **sigL**.

```
(%i4) Mdata : read_matrix (fname);
(%o4) matrix([0,106],[15,80],[30,98],[45,75],[60,74],[75,73],[90,49],[105,38],
                      [120,37],[135,22])
(%i5) yL : list_matrix_entries (col (Mdata,2));
(%o5) [106,80,98,75,74,73,49,38,37,22]
(%i6) sigL : sqrt (yL),numer;
(%o6) [10.2956,8.94427,9.89949,8.66025,8.60233,8.544,7.0,6.16441,
                              6.08276,4.69042]
(%i7) out :  fit_line (Mdata, sigL);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
 num_data =  10
 num_param =  2
 dof =  8
 chi2/dof =  1.04017
 chi2_prob =  40.2721 %
a =  104.462  +/-  5.25106
b =  -0.593987  +/-  0.0536575
(%o7) [[a = 104.462,b = -0.593987],[5.25106,0.0536575],8.3214,0.402721]
(%i8) yfit : a + b*t, out[1];
(%o8) 104.462-0.593987*t
```

which indicates that $\chi^2/\nu \approx 1$ and there is a roughly 40% probability that a repetition of the same experiment (a similarly prepared radiation source, etc.) would produce a value of χ^2 greater than that found. Thus both measures of goodness-of-fit imply that a straight line model is a good fit to the given data.

We load **draw** and **qdraw** and make a simple plot of the raw data.

```
(%i9) load(draw);
(%o9) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i10) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o10) "c:/work9/qdraw.mac"
(%i11) ptsL : read_nested_list (fname);
(%o11) [[0,106],[15,80],[30,98],[45,75],[60,74],[75,73],[90,49],[105,38],
        [120,37],[135,22]]
(%i12) qdraw (pts (ptsL,pc(black),ps(1)),
         xr (-10,140), yr (0,120),
          more (xlabel = "t", ylabel = "C"))$
```

which produces the plot

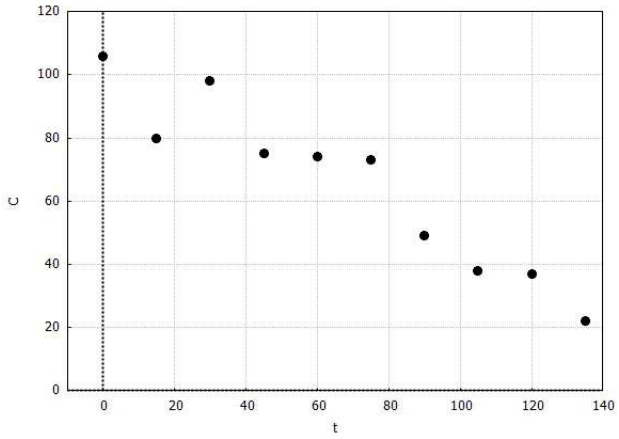

Figure 3: Counts per 15 sec. vs. Time (sec.) Raw Data

We can then add error bars

```
(%i13) qdraw (pts (ptsL,pc(black),ps(1)),
          xr (-10,140),yr(0,120),
          more (xlabel = "t", ylabel = "C"),
          errorbars (ptsL, sigL, lw(3),lc(blue)))$
```

which produces the plot

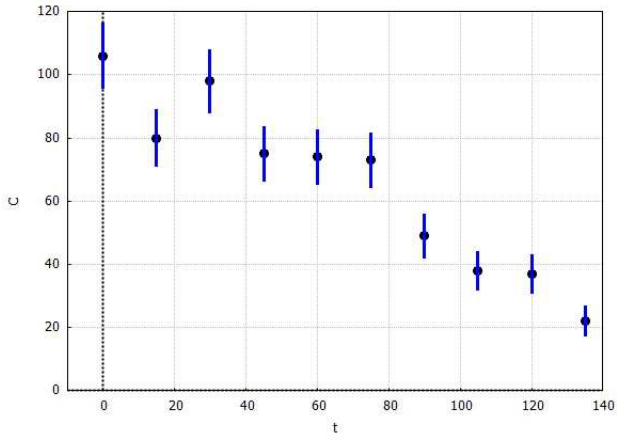

Figure 4: Counts vs. Time Raw Data with Statistical Error Bars

Finally, we can add the straight line fit to the data

```
(%i14) qdraw (pts (ptsL,pc(black),ps(1)),
          xr (-10,140),yr(0,120),
          more (xlabel = "t", ylabel = "C"),
          errorbars (ptsL, sigL, lw(3),lc(blue)),
          ex1 (yfit, t ,0, 140, lc(brown)))$
```

which produces the plot

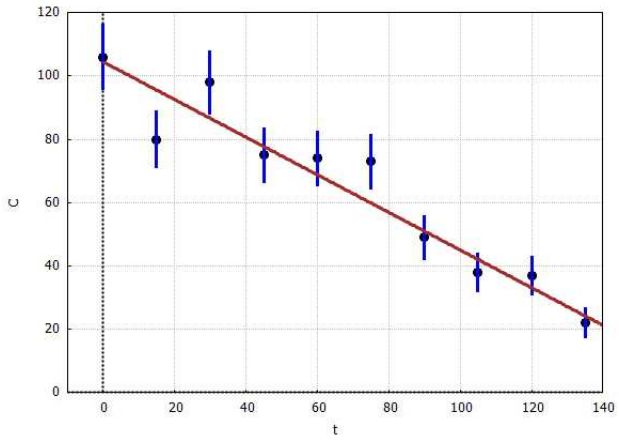

Figure 5: Counts vs. Time Data with Best Fit Line

We also try out the general nonlinear method function **`nlfit`** on this two parameter linear problem.

```
(%i15) nlfit (Mdata,sigL,a + b*x,[a,b],[1,1]);
 Ndata =  10
 Nparam =  2
 dof =  8
 ivar =  x
start:   params:    [a = 1.0,b = 1.0]        chi2 =  1153.75
------------------------------------------------------------
  n     lam
  1    0.001
               p_oldL =  [1.0,1.0]
               p_newL =  [103.184,-0.580648]    chi2_new =  8.3852
  2    1.0e-4
               p_oldL =  [103.184,-0.580648]
               p_newL =  [104.448,-0.593843]    chi2_new =  8.32141
  3    1.0e-5
               p_oldL =  [104.448,-0.593843]
               p_newL =  [104.462,-0.593985]    chi2_new =  8.3214
------------------------------------------------------------
 chi2/dof =  1.04017
 chi2_prob =  40.2721 %
------------------------------------------------------------
a =  104.462  +/-  5.22337
b =  -0.593985  +/-  0.0533745
(%o15) [[a = 104.462,b = -0.593985],[5.22337,0.0533745],8.3214,0.402721]
```

which produces results similar to **`fit_line`**.

6 Example 3: Straight Line Fit of Data with Uniform Instrumental Errors

The raw data from Table 6.1 from Bevington (3rd edition, pdf 114, see the References section at the end of this document) is contained in the file **`mbe14-fit3.dat`**, with column 1 the position (x in cm.) along a current carrying nickel-silver wire, and column 2 being the corresponding voltage (V in volts), with nine data points provided. From the nature of the voltage meter, we assume each voltage reading has a uniform uncertainty equal to 0.05 volts. We use **`fit_line`** to try fitting the straight line $V = a + b\,x$ to the given data.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/mbe14-fit3.dat"$
```

```
(%i3) printfile (fname)$
10.0 0.37
20.0 0.58
30.0 0.83
40.0 1.15
50.0 1.36
60.0 1.62
70.0 1.90
80.0 2.18
90.0 2.45
(%i4) Mdata : read_matrix (fname);
(%o4) matrix([10.0,0.37],[20.0,0.58],[30.0,0.83],[40.0,1.15],[50.0,1.36],
             [60.0,1.62],[70.0,1.9],[80.0,2.18],[90.0,2.45])
(%i4) ndata : length (Mdata);
(%o4) 9
(%i5) sigL : makelist (0.05,i,1,ndata);
(%o5) [0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05,0.05]
(%i6) out : fit_line (Mdata,sigL);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
 num_data =  9
 num_param =  2
 dof =  7
 chi2/dof =  0.278508
 chi2_prob =  96.2579 %
a =  0.0713889  +/-  0.0363242
b =  0.0262167  +/-  6.45497e-4
(%o6) [[a = 0.0713889,b = 0.0262167],[0.0363242,6.45497e-4],1.94956,0.962579]
(%i7) yfit : a + b*x, out[1];
(%o7) 0.0262167*x+0.0713889
```

We then plot the best fit straight line with the data points, including error bars, using **qdraw** as above:

```
(%i8) ptsL : read_nested_list (fname);
(%o8) [[10.0,0.37],[20.0,0.58],[30.0,0.83],[40.0,1.15],[50.0,1.36],
       [60.0,1.62],[70.0,1.9],[80.0,2.18],[90.0,2.45]]
(%i9) load(draw);
(%o9) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i10) load(qdraw);
(%o10) "c:/work9/qdraw.mac"
(%i11) qdraw (pts (ptsL,pc(black),ps(1)),
          xr (0,100),yr(0,3),
          more (xlabel = "x", ylabel = "V"),
          errorbars (ptsL, sigL, lw(3),lc(blue)),
          ex1 (yfit, x,0,100, lc(brown)))$
```

which produces the plot

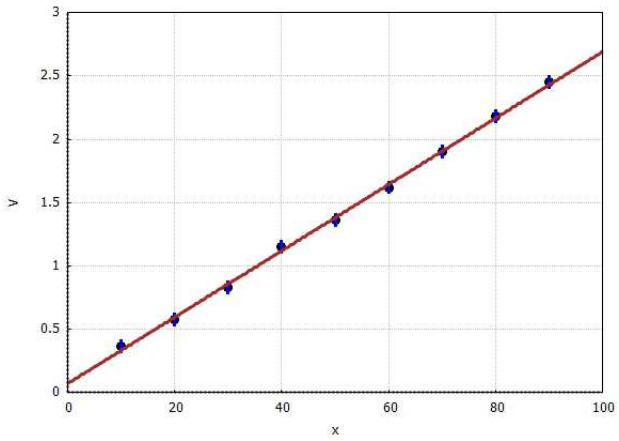

Figure 6: Voltage vs. Position Data with Best Fit Line

7 Example 4: Straight Line Fit of Inverse Square Law with Poisson Errors

The raw data from Table 6.2 from Bevington (3rd edition, pdf 114-115) is contained in the file **mbe14-fit4.dat**, with column 1 the distance (d in meters) from a radioactive source to a Geiger counter, and column 2 being the corresponding number of counts in 7.5 min intervals C, with ten data points provided. We seek to fit a general inverse square law model

$$C = a + b/d^2 = a + b\,x \tag{7.1}$$

with $x = 1/d^2$, the squared inverse distance with units $1/m^2$, to this data. Since the values of the dependent variable $y_i = C_i$ represent the number of counts recorded in a Geiger counter exposed to a radioactive source in some standard time interval, we ignore possible intrumental Geiger counter errors and include only the typical Poisson statistics errors associated with the source We use **fit_line** to try fitting the straight line $C = a + b\,x$ to the data. We follow the steps used in Example 2 which assumed Poisson errors. We need to convert the raw data (d, C) (in the data file) to the form $(x = d^{-2}, C)$.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/mbe14-fit4.dat"$
(%i3) printfile (fname)$
0.20 901
0.25 652
0.30 443
0.35 339
0.40 283
0.45 281
0.50 240
0.60 220
0.75 180
1.00 154
(%i4) dCM : read_matrix (fname);
(%o4) matrix([0.2,901],[0.25,652],[0.3,443],[0.35,339],[0.4,283],[0.45,281],
             [0.5,240],[0.6,220],[0.75,180],[1.0,154])
(%i5) dL : list_matrix_entries (col (dCM,1));
(%o5) [0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.6,0.75,1.0]
(%i6) CL : list_matrix_entries (col (dCM,2));
(%o6) [901,652,443,339,283,281,240,220,180,154]
(%i7) sigL : sqrt(CL),numer;
(%o7) [30.0167,25.5343,21.0476,18.412,16.8226,16.7631,15.4919,14.8324,13.4164,
       12.4097]
(%i8) xL : 1/dL^2;
(%o8) [25.0,16.0,11.1111,8.16327,6.25,4.93827,4.0,2.77778,1.77778,1.0]
(%i9) xCL : xyList (xL,CL);
(%o9) [[25.0,901],[16.0,652],[11.1111,443],[8.16327,339],[6.25,283],
       [4.93827,281],[4.0,240],[2.77778,220],[1.77778,180],[1.0,154]]
(%i10) xCM : apply ('matrix, xCL);
(%o10) matrix([25.0,901],[16.0,652],[11.1111,443],[8.16327,339],[6.25,283],
              [4.93827,281],[4.0,240],[2.77778,220],[1.77778,180],[1.0,154])
(%i11) out : fit_line (xCM,sigL);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
 num_data =  10
 num_param =  2
 dof =  8
 chi2/dof =  1.36831
 chi2_prob =  20.475 %
a =  119.497  +/-  7.5676
b =  30.6979  +/-  1.03408
(%i11) [[a = 119.497,b = 30.6979],[7.5676,1.03408],10.9465,0.20475]
(%i12) yfit : a + b*x, out[1];
(%o12) 30.6979*x+119.497
```

A linear fit to the data of the function $C = a + b\,x$ gives $a = 119 \pm 8$, $b = 31 \pm 1$, with $\chi^2 \approx 11$ for 8 degrees of freedom, $\chi^2_\nu = 1.4$. The χ^2 probability for the fit is about 20%.

We proceed to plot the data points and the straight line fit.

```
(%i13) load(draw);
(%o13) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i14) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o14) "c:/work9/qdraw.mac"
(%i15) qdraw (pts (xCL,pc(black),ps(1)),
          xr (0,30),yr(0,1000),
          more (xlabel = "x = 1/d^2", ylabel = "C"),
          errorbars (xCL, sigL, lw(3),lc(blue)),
          ex1 (yfit, x, 0, 30, lc(brown)))$
```

which produces the plot

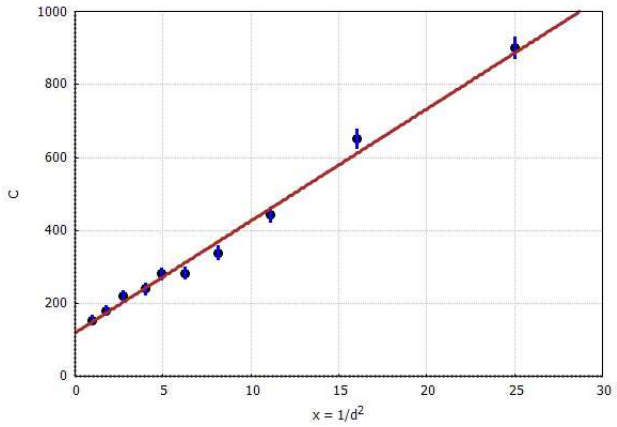

Figure 7: Counts vs. Inverse Distance Squared with Best Fit Line

8 Histograms, Random Numbers, and Gaussian Noise

8.1 Creating a Histogram from a List of Integers

We will use the Maxima package **`.../share/descriptive/descriptive.mac`** to make histograms via the use of the **`draw`** package. Here is an example similar to one from the Maxima help manual under the entry "histogram." The file **`pidigits.data`** is a list of the first 100 digits of the irrational number π. Each of these digits is an integer in the range $[0, 9]$ (ten possible values).

If the **`draw`** package is not already loaded, the loading of **`descriptive`** automatically loads **`draw`**. We place the integers found in the first 100 digits of π into 10 bins ("classes"), and let the leading edge of the first bin be located at -0.5, and the trailing edge of the last bin be located at 9.5. The functions **`fll`**, **`head`**, and **`tail`** are defined in **`fit.mac`**. The function **`fll(alist)`** returns the first element, the last element, and the length of the list.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) sL : read_list (file_search ("pidigits.data"))$
(%i3) fll (sL);
(%o3) [3,7,100]
(%i4) head (sL);
(%o4) [3,1,4,1,5,9]
(%i5) tail (sL);
(%o5) [1,1,7,0,6,7]
(%i6) load (descriptive);
(%o6) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/descriptive/descriptive.mac"
(%i7) histogram (sL, nclasses = [-0.5,9.5,10], title = "pi digits",
           xlabel = "digits", ylabel = "Absolute frequency",
           fill_color   = blue, fill_density = 0.6)$
```

which produces the histogram

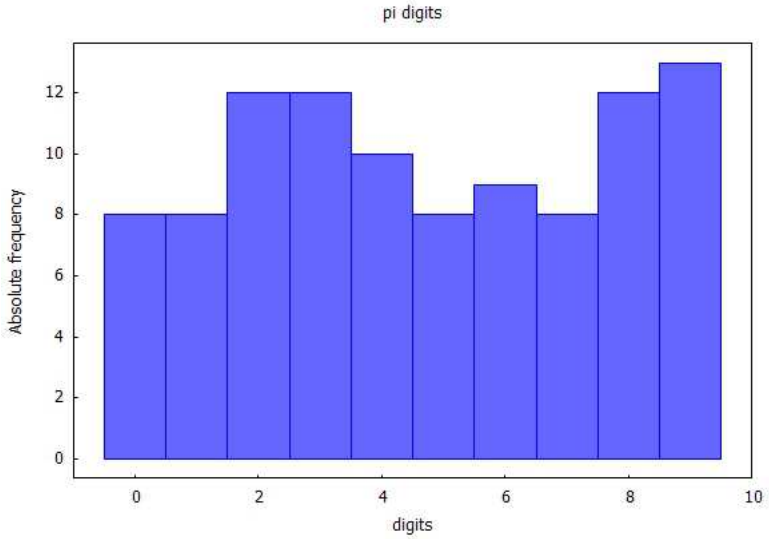

Figure 8: Histogram of the First 100 Digits of π

We can confirm the frequency of the various integer values in the list **sL** by defining a small function **integer_frequency(alist, an_integer)**.

```
(%i8) integer_frequency (xL,nv) :=
block ([val : 0],
    for j thru length (xL) do
         if xL[j] = nv then val : val + 1,
    val)$
(%i9) for k:0 thru 9 do
          print(" ",k," ",integer_frequency(sL,k))$
  0   8
  1   8
  2   12
  3   12
  4   10
  5   8
  6   9
  7   8
  8   12
  9   13
```

8.2 Creating a Histogram from a List of Pseudo-Random Floating Point Numbers

The Maxima function **random(x)** returns a non-negative floating point number in the open interval $[0.0, x]$ if **x** is a positive floating point number.

The returned number is drawn from a "uniform distribution," and is hence called a "uniform random variate" (or "uniform random deviate"), and is actually drawn from a "pseudo-random" sequence produced by code termed a "random number generator." The word "random" is properly reserved for the output of an intrinsically random physical process (see Numerical Recipes, Ch. 7, Random Numbers).

The command **set_random_state (make_random_state(an_integer))** uses the supplied integer to "seed" the random number generator. If you re-seed with the same seed, you will get the same sequence of "random" numbers.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
```

```
(%i2) set_random_state (make_random_state (654321))$
(%i3) rL : makelist(random(1.0),j,1,10000)$
(%i4) fll (rL);
(%o4) [0.226074,0.682233,10000]
(%i5) head (rL);
(%o5) [0.226074,0.677002,0.303875,0.953276,0.36083,0.538895]
(%i6) tail (rL);
(%o6) [0.315565,0.390841,0.890081,0.438075,0.00471975,0.682233]
(%i7) plot2d([discrete,rL],[x,0,10000],[y,0,1],[xlabel,""],
          [ylabel,""],[style,[points,1,5]])$
```

which produces the plot

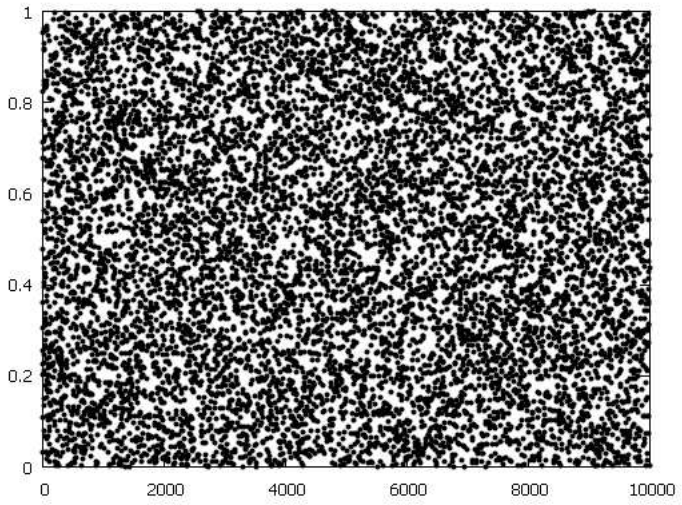

Figure 9: 10,000 Random Numbers in Range $0.0 < x < 1.0$

We next place the $10,000$ uniform random deviates in the list **`rL`** into 20 bins, using the function **`histogram`** defined in **`descriptive.mac`**. Since $10000/20 = 500$, we expect there to be roughly 500 numbers thrown into each of the 20 bins, if the numbers are drawn from a "uniform distribution."

```
(%i8) load (descriptive)$
(%i9) histogram (rL,nclasses = 20,title = "uniform variates",
          xlabel = "", ylabel = "",fill_color = blue,
          fill_density = 0.6)$
```

which produces the histogram

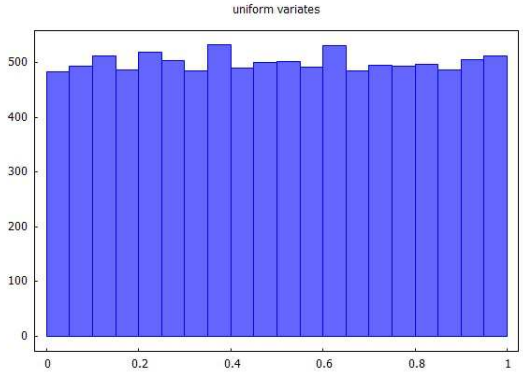

Figure 10: Histogram of $10,000$ Random Numbers in Range $0.0 < x < 1.0$

We could also have used the command **random_continuous_uniform(0,1,10000)**, using the **distrib** package, to generate a list of 10,000 uniform random deviates in the open interval $0.0 < x < 1.0$. One must first load the package **distrib.mac** to use the **random_continuous_uniform** function, and during the loading, the random number generator is automatically seeded with an integer which depends on your computer's clock time at the moment of loading. Hence your results here will differ in details. (You could then re-seed the random number generator to reproduce some previously generated sequence of random numbers, if so desired.)

```
(%i10) load(distrib);
(%o10) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/distrib/distrib.mac"
(%i11) rL2 : random_continuous_uniform(0,1,10000)$
(%i12) fll (rL2);
(%o12) [0.876421,0.34568,10000]
(%i13) head (rL2);
(%o13) [0.876421,0.851329,0.87345,0.141658,0.40555,0.592372]
(%i14) tail (rL2);
(%o14) [0.095683,0.412016,0.539407,0.571131,0.718902,0.34568]
(%i15) histogram (rL2,nclasses = 20,title = "uniform variates",
          xlabel = "", ylabel = "",fill_color = blue,
          fill_density = 0.6)$
```

which produces the histogram

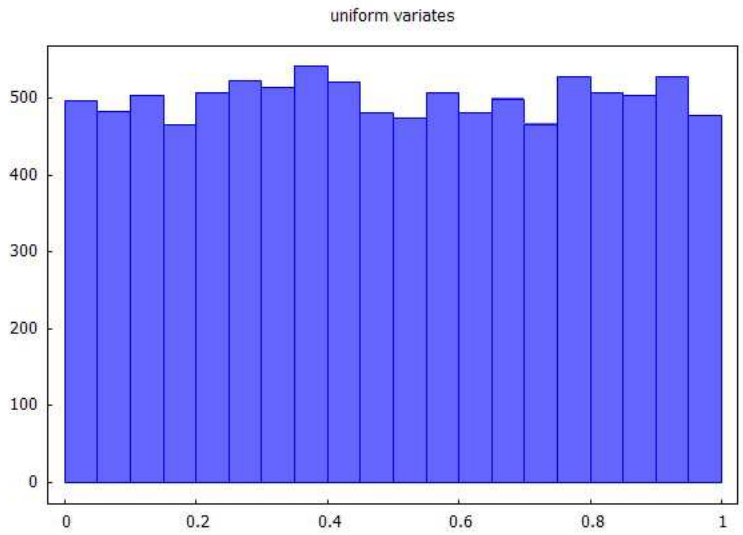

Figure 11: Histogram of a New Set of $10,000$ Random Numbers in Range $0.0 < x < 1.0$

8.3 Gaussian Noise

The Maxima package **.../share/distrib/distrib.mac** has the function **random_normal(m, s)** for one Gaussian deviate, or **random_normal(m, s, n)** for a list of **n** Gaussian deviates having a mean value equal to **m** and a standard deviation (the square root of the variance) equal to **s**. This package seeds Maxima's random number generator with a value depending on your computer's clock time while loading (as mentioned above), so you should get results which differ in details. We have started a new Maxima session here:

```
(%i1) load(distrib);
(%o1) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/distrib/distrib.mac"
(%i2) random_normal(0,10);
(%o2) 5.859170811329763
(%i3) fpprintprec:6$
```

```
(%i4) random_normal(0,10);
(%o4) -6.08762
(%i5) random_normal(0,10,5);
(%o5) [-7.74102,-12.3573,14.7975,5.13447,0.487853]
(%i6) plot2d([discrete,random_normal(0,1,500)],[x,0,500],[y,-4,4],[xlabel,""],
           [ylabel,""],[style,[points,1,5]])$
```

which produces the plot

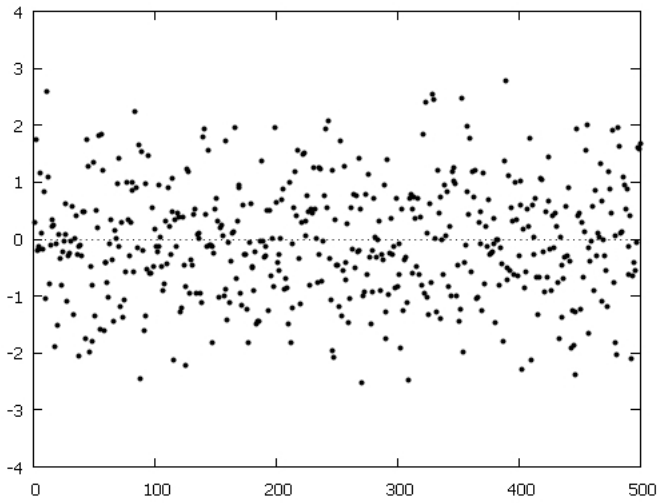

Figure 12: 500 Gaussian Deviates with Mean $= 0, s = 1$

We need to bin these random Gaussian deviates to see how they are distributed. As an example, we generate a new list of 10,000 Gaussian deviates which have a mean value 0 and a standard deviation 1 and place them into 20 bins using the function **`histogram`**.

```
(%i7) load (descriptive);
(%o7) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/descriptive/descriptive.mac"
(%i8) histogram ( random_normal(0,1,10000), nclasses = 20,
             title  = "uniform normal variates", xlabel       = "",
             ylabel  = "", fill_color   = blue, fill_density = 0.6)$
```

which produces the histogram

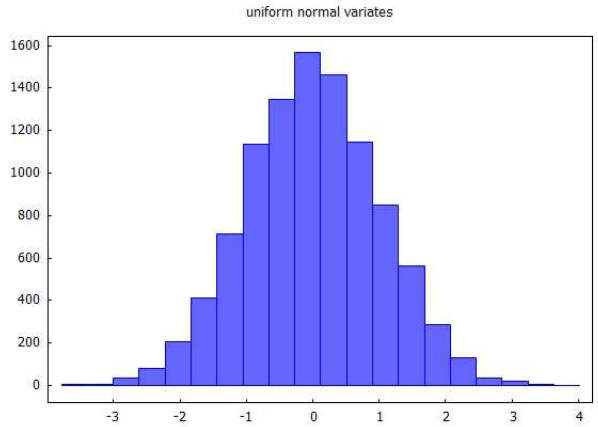

Figure 13: $10,000$ Gaussian Deviates with Mean $= 0, s = 1$

which is approaching the familiar bell-shaped curve.

We can use our **moment (dataL)** function defined in **fit.mac** to compare the mean, variance and standard deviation of a set of numbers produced by **random_normal (m,s,n)** in which **m** is the desired mean, **s** is the desired standard deviation, and the variance is then **s^2**. We again start a new Maxima session.

```
(%i1) load(distrib);
(%o1) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/distrib/distrib.mac"
(%i2) random_normal(0,1);
(%o2) 0.7792856737962007
(%i3) random_normal(0,1,10);
(%o3) [-0.1800590734570673,1.003713813350962,1.012049729950966,
       -1.188740041414326,-0.4503024282532589,0.214624288905816,
       2.240769783420878,0.6307828337928504,-1.355732037620963,
       -0.4398222598470207]
(%i4) moment (random_normal(0,1,10000));
 ndata =  10000
 mean =  4.5016e-4
 variance =  0.99186
 sigma =  0.99592
(%o4) [4.5016e-4,0.99186,0.99592]
```

A sample of 10,000 random normal deviates returned by **random_normal(0,1,10000)** has a mean of approximately 0 and a variance and standard deviation of approximately 1.

Likewise, a sample of 10,000 random normal deviates returned by **random_normal(0,0.5,10000)** has a mean of approximately 0 and a standard deviation of approximately 0.5 and a variance of approximately 0.25.

```
(%i5) moment (random_normal(0,0.5,10000));
 ndata =  10000
 mean =  0.0034911
 variance =  0.25136
 sigma =  0.50136
(%o5) [0.0034911,0.25136,0.50136]
```

Finally, a sample of 10,000 random normal deviates returned by **0.5*random_normal(0,1,10000)** has a mean of approximately 0, a standard deviation of approximately 0.5, and a variance of approximately 0.25.

```
(%i6) moment (0.5*random_normal(0,1,10000));
 ndata =  10000
 mean =  0.0066488
 variance =  0.25717
 sigma =  0.50712
(%o6) [0.0066488,0.25717,0.50712]
```

9 Example 5: Generating and Fitting Data with Gaussian Noise

We generate some noisy data by adding Gaussian noise to the signal $y = 1 - 2\,x$. We do not seed the random number generator (after loading **distrib.mac**) in the following example, allowing Maxima to use the default method. The **distrib** package seeds Maxima's random number generator with a value depending on your computer's clock time while loading. As a consequence, if you repeat this example, you will get a different set of random numbers, and slightly different results than shown here. The function **xyList(xL,yL)**, defined in our **fit.mac**, produces an xyList type of list having the form **[[x1,y1],[x2,y2],...[xN,yN]]**. Below we use **xyList(xL,yL)** to produce **xyL**, a nested data list which can later be converted to a data matrix we call **Mdata**.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
```

```
(%i2) load(draw);
(%o2) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i3) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o3) "c:/work9/qdraw.mac"
(%i4) load(distrib);
(%o4) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/distrib/distrib.mac"
(%i5) fpprintprec:5$
(%i6) xL : makelist (0.1*j,j,1,100)$
(%i7) fll (xL);
(%o7) [0.1,10.0,100]
(%i8) dyL : random_normal (0,0.25,100)$
(%i9) moment (dyL);
 ndata =  100
 mean =  0.0043537
 variance =  0.063962
 sigma =  0.25291
(%o9) [0.0043537,0.063962,0.25291]
(%i10) yL : -2*xL + 1 + dyL$
(%i11) xyL : xyList(xL,yL)$
(%i12) fll(xyL);
(%o12) [[0.1,0.79312],[10.0,-18.799],100]
(%i13) qdraw (ex1(1-2*x,x,0,10,lc(red)),pts(xyL,ps(1),pc(black),pt(1)),
               more (xlabel = "x", ylabel = "y"))$
```

which produces the plot

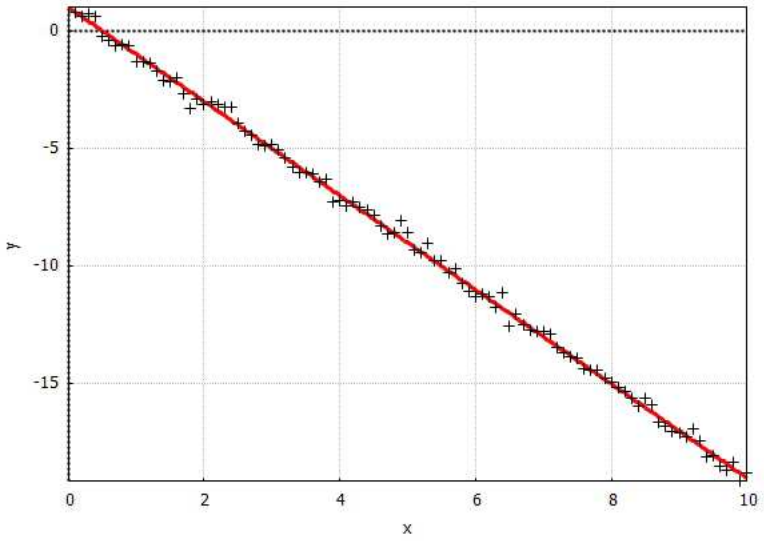

Figure 14: 100 Data Points with Gaussian Noise and the Line $1 - 2\,x$

We can try to fit a straight line model, using **`fit_line(Mdata, sigmaL)`**, to this noisy data.

```
(%i14) sigL : makelist(0.25291,j,1,100)$
(%i15) fll (sigL);
(%o15) [0.25291,0.25291,100]
(%i16) Mdata : apply ('matrix,xyL)$
(%i17) row (Mdata,1);
(%o17) matrix([0.1,0.79312])
(%i18) out : fit_line (Mdata, sigL);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
 num_data =  100
 num_param =  2
 dof =  98
 chi2/dof =  1.0084
 chi2_prob =  45.76 %
```

```
a =  0.98606  +/-  0.050964
b =  -1.9964  +/-  0.0087615
(%o18) [[a = 0.98606,b = -1.9964],[0.050964,0.0087615],98.827,0.4576]
```

The least squares best fit line thus has the parameters $a = 0.99 \pm 0.05$, $b = -2.00 \pm 0.01$, with $\chi^2/\text{dof} = 1$ and $Q = 0.46 = 46\%$.

10 Example 6: Fit to a Quadratic using lfit

This example is taken from Bevington(3rd), pdf 133, "Ex. 7.1." The data file **`mbe14-fit6.dat`** contains 21 data points for the voltage output (voltage in mV) of a thermocouple as a function of the temperature T (degrees Celsius) varying from 0 to 100. The model equation for the voltage V as a function of temperature T is taken to be

$$V(T) = a_1 + a_2\,T + a_3\,T^2. \tag{10.1}$$

Since this model expression is linear in the model parameters, we use **`lfit`** with this data set and model.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/mbe14-fit6.dat"$
(%i3) Mdata : read_matrix (fname);
(%o3) matrix([0,-0.849],[5,-0.738],[10,-0.537],[15,-0.354],[20,-0.196],
             [25,-0.019],[30,0.262],[35,0.413],[40,0.734],[45,0.882],
             [50,1.258],[55,1.305],[60,1.541],[65,1.768],[70,1.935],
             [75,2.147],[80,2.456],[85,2.676],[90,2.994],[95,3.2],[100,3.318])
(%i4) ndata : length (Mdata);
(%o4) 21
(%i5) sigL : makelist (0.05,i,1,ndata)$
(%i6) fll(sigL);
(%o6) [0.05,0.05,21]
(%i7) yexpr : a1 + a2*T + a3*T^2$
(%i8) out : lfit (Mdata, sigL, yexpr,[a1,a2,a3]);
 ivar =  T
 num_data =  21
 num_param =  3
 dof =  18
 chi2/dof =  1.47575
 chi2_prob =  8.75548 %
a1 =  -0.918104  +/-  0.0298453
a2 =  0.0376543  +/-  0.00138311
a3 =  5.49009e-5  +/-  1.33533e-5
(%o8) [[a1 = -0.918104,a2 = 0.0376543,a3 = 5.49009e-5],
       [0.0298453,0.00138311,1.33533e-5],26.5635,0.0875548]
(%i9) yfit : yexpr, out[1];
(%o9) 5.49009e-5*T^2+0.0376543*T-0.918104
```

We next make a plot of the data and the fit.

```
(%i10) load(draw);
(%o10) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i11) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o11) "c:/work9/qdraw.mac"
(%i12) TVL : read_nested_list (fname);
(%o12) [[0,-0.849],[5,-0.738],[10,-0.537],[15,-0.354],[20,-0.196],[25,-0.019],
        [30,0.262],[35,0.413],[40,0.734],[45,0.882],[50,1.258],[55,1.305],
        [60,1.541],[65,1.768],[70,1.935],[75,2.147],[80,2.456],[85,2.676],
        [90,2.994],[95,3.2],[100,3.318]]
(%i13) qdraw (xr(-5,105),yr(-2,4),
           more (xlabel = "T", ylabel = "V"),
              pts (TVL,pc(black),ps(1)),
            errorbars (TVL, sigL, lw(3),lc(red)),
           ex1 (yfit,T,0,100))$
```

which produces the plot

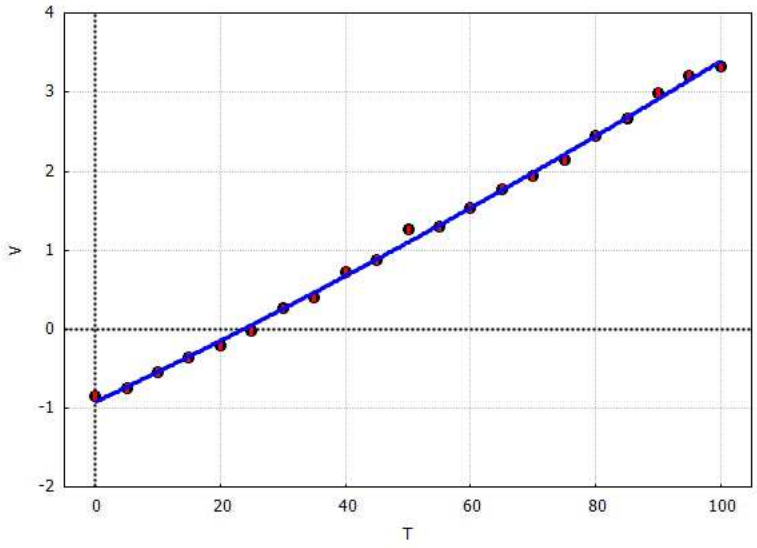

Figure 15: Voltage vs. Temperature Data with Best Fit Quadratic

11 Example 7: Using lfit with Legendre Polynomials as Basis Functions

11.1 Experimenting with the Maxima Function legendre_p

Maxima provides the function **`legendre_p(n,x)`** (defined in the **`orthopoly.lisp`** package) which produces the standard Legendre polynomial $P_n(x)$ for $n = 0, 1, 2, \ldots$. The **`orthopoly`** package automatically loads when you invoke **`legendre_p`**. You will need to apply **`ratsimp`** to get the simplified analytic form. Using the definitions of **`legendre_p`** provided by the package **`orthopoly.lisp`** (loads automatically when we try to use **`legendre_p`** interactively), we find that we need to set **`ratprint:false`** and **`orthopoly_returns_intervals : false`** to get ordinary floating point returns without the error interval additions and without the **`rat`** warnings. These settings are made in **`fit.mac`**.

As an experiment, we invoke **`legendre_p` << without >>** first loading **`fit.mac`** or **`orthopoly.lisp`**.

```
(%i1) legendre_p(4,x);
STYLE-WARNING: redefining MAXIMA::SIMP-UNIT-STEP in DEFUN
STYLE-WARNING: redefining MAXIMA::SIMP-POCHHAMMER in DEFUN
(%o1) (-10*(1-x))+(35*(1-x)^4)/8-(35*(1-x)^3)/2+(45*(1-x)^2)/2+1
(%i2) ratsimp(%);
(%o2) (35*x^4-30*x^2+3)/8
(%i3) P(nn,xx) := ratsimp (legendre_p(nn,xx))$
(%i4) P(0,x);
(%o4) 1
(%i5) P(1,x);
(%o5) x
(%i6) P(2,x);
(%o6) (3*x^2-1)/2
(%i7) P(3,x);
(%o7) (5*x^3-3*x)/2
(%i8) P(4,x);
(%o8) (35*x^4-30*x^2+3)/8
(%i9) P(4,1.2);
rat: replaced 4.046999999999998 by 4047/1000 = 4.047

rat: replaced 3.202738074747912e-14 by 5/156116419242106 = 3.20273807474791e-14
(%o9) interval(4047/1000,5/156116419242106)
```

```
(%i10) ratprint:false$
(%i11) P(4,1.2);
(%o11) interval(4047/1000,5/156116419242106)
(%i12) orthopoly_returns_intervals;
(%o12) true
(%i13) orthopoly_returns_intervals : false$
(%i14) P(4,1.2);
(%o14) 4047/1000
(%i15) float(%);
(%o15) 4.047
```

Information about the functions available via **`.../share/orthopoly/orthopoly.lisp`** can be found in the comment sections of that file.

The first four Legendre polynomials are

$$P_0(x) = 1 \quad P_2(x) = \frac{1}{2}\,(3\,x^2 - 1) \tag{11.1}$$

$$P_1(x) = x \quad P_3(x) = \frac{1}{2}\,(5\,x^3 - 3\,x) \tag{11.2}$$

$P_n(x)$ is an even function of x if n is an even integer, and is an odd function of x if n is an odd integer. The set of Legendre polynomials provides an "orthogonal set" of functions over the interval $-1 \le x \le 1$, where $x = \cos(\theta)$.

$$\int_{-1}^{1} P_n(x)\,P_m(x)\,dx = \frac{2}{2\,n+1}\,\delta_{nm}, \tag{11.3}$$

in which the Kronecker delta function δ_{nm} equals 1 if $n = m$, and otherwise equals 0.

Having defined **`P(n,x)`** (above) to return the simplified form of **`legendre_p(n,x`**, we can use this function in integrals and plots.

```
(%i16) integrate(P(2,x)^2,x,-1,1);
(%o16) 2/5
(%i17) integrate(P(2,x)*P(3,x),x,-1,1);
(%o17) 0
(%i18) plot2d([P(0,x),P(1,x),P(2,x),P(3,x),P(4,x)],[x,-1,1], [y,-1.5,2.5],
         [xlabel,  "x"], [ylabel , "P_n(x)"],[legend,"0","1","2","3","4"],
              [style,[lines,5]])$
```

which produces the plot

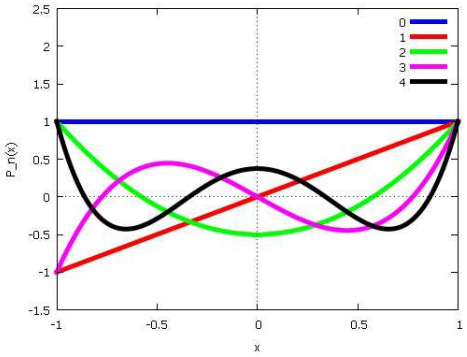

Figure 16: Legendre Polynomials for $n = 0, 1, 2, 3, 4$

11.2 Application of Legendre Polynomials to Example 7

This example follows Bevington(3rd): (Ex. 7.3, pdf 146).

Quoting Bevington (loosely)

> Let us consider an experiment in which Carbon-13 is bombarded by 4.5-MeV protons. In the subsequent reaction, some of the protons are captured by the Carbon-13 nucleus, producing a Nitrogen-14 nucleus in an excited state which then decays by gamma emission, producting gamma rays with energies up to 11 MeV. A measurement of the angular distribution of the emitted gamma rays gives information about the angular momentum states of the energy levels in the residual Nitrogen-14 nucleus.

The file **`mbe14-fit7.dat`** contains measurements of the number of gamma ray counts in a specified fixed time interval recorded at 17 different angles from 0 to 160 degrees. The first column is the angle in degrees, and the second column is the gamma ray count at that angle. We need to convert the angles in degrees to angles in radians before using Maxima's **`cos`** function. The measurement errors in the count rate are assumed to be wholly statistical, and the use of Poisson statistics implies the square root of the count rate should be used for the uncertainty of each count value.

11.3 Five Parameter Fit Using lfit with Legendre Polynomials

We use the first five Legendre polynomials as a model of the given data, with adjustable amplitudes to be found.

With $x = \cos\theta$, our model is

$$C = a_0\,P_0(x) + a_1\,P_1(x) + a_2\,P_2(x) + a_3\,P_3(x) + a_4\,P_4(x). \tag{11.4}$$

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/mbe14-fit7.dat"$
(%i3) Mdata : read_matrix (fname);
(%o3) matrix([0,1400],[10,1386],[20,1130],[30,1045],[40,971],[50,862],
             [60,819],[70,808],[80,862],[90,829],[100,824],[110,839],
             [120,819],[130,901],[140,925],[150,1044],[160,1224])
(%i4) DegreesL : list_matrix_entries (col (Mdata,1));
(%o4) [0,10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160]
(%i5) CountsL : list_matrix_entries (col (Mdata,2));
(%o5) [1400,1386,1130,1045,971,862,819,808,862,829,824,839,819,901,925,1044,1224]
(%i6) sigL : sqrt (CountsL), numer;
(%o6) [37.4166,37.229,33.6155,32.3265,31.1609,29.3598,28.6182,28.4253,29.3598,
       28.7924,28.7054,28.9655,28.6182,30.0167,30.4138,32.311,34.9857]
(%i7) RadiansL : DegreesL*%pi/180, numer;
(%o7) [0,0.174533,0.349066,0.523599,0.698132,0.872665,1.0472,1.22173,1.39626,
       1.5708,1.74533,1.91986,2.0944,2.26893,2.44346,2.61799,2.79253]
(%i8) xL : cos (RadiansL);
(%o8) [1,0.984808,0.939693,0.866025,0.766044,0.642788,0.5,0.34202,0.173648,
       6.12303e-17,-0.173648,-0.34202,-0.5,-0.642788,-0.766044,-0.866025,
       -0.939693]
(%i16) xCL : xyList (xL,CountsL);
(%o16) [[1,1400],[0.984808,1386],[0.939693,1130],[0.866025,1045],
        [0.766044,971],[0.642788,862],[0.5,819],[0.34202,808],[0.173648,862],
        [6.12303e-17,829],[-0.173648,824],[-0.34202,839],[-0.5,819],
        [-0.642788,901],[-0.766044,925],[-0.866025,1044],[-0.939693,1224]]
(%i9) dataM : apply ('matrix, xCL)$
(%i10) param_list : [a0,a1,a2,a3,a4]$
(%i11) P(nn,xx) := ratsimp (legendre_p (nn,xx))$
(%i12) y_expr_noun : sum (param_list[i+1] * 'P(i,x), i, 0, 4);
(%o12) 'P(4,x)*a4+'P(3,x)*a3+'P(2,x)*a2+'P(1,x)*a1+'P(0,x)*a0
```

```
(%i13) y_expr : %, nouns;
STYLE-WARNING: redefining MAXIMA::SIMP-UNIT-STEP in DEFUN
STYLE-WARNING: redefining MAXIMA::SIMP-POCHHAMMER in DEFUN
(%o13) (a4*(35*x^4-30*x^2+3))/8+(a3*(5*x^3-3*x))/2+(a2*(3*x^2-1))/2+a1*x+a0
(%i14) out : lfit (dataM, sigL, y_expr,param_list);
 ivar =  x
 num_data =  17
 num_param =  5
 dof =  12
 chi2/dof =  1.4342
 chi2_prob =  14.1852 %
a0 =  906.784  +/-  7.77408
a1 =  -1.01955  +/-  12.4292
a2 =  258.527  +/-  16.3188
a3 =  11.9971  +/-  19.4677
a4 =  189.521  +/-  21.6578
(%o14) [[a0 = 906.784,a1 = -1.01955,a2 = 258.527,a3 = 11.9971,a4 = 189.521],
        [7.77408,12.4292,16.3188,19.4677,21.6578],17.2104,0.141852]
(%i15) yfit5 : y_expr, out[1];
(%o15) 23.6901*(35*x^4-30*x^2+3)+5.99855*(5*x^3-3*x)+129.263*(3*x^2-1)
                                -1.01955*x+906.784
```

We can now make a plot of the five parameter fit using Legendre polynomials.

```
(%i16) load(draw);
(%o16) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i17) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o17) "c:/work9/qdraw.mac"
(%i18) qdraw (xr(-1.05,1.05),yr(0,1500),
           more (xlabel = "cos(th)", ylabel = "C"),
              pts (xCL, pc(black), ps(1)),
               errorbars (xCL, sigL, lw(3),lc(red)),
              ex1(yfit5,x,-1,1))$
```

which produces the plot

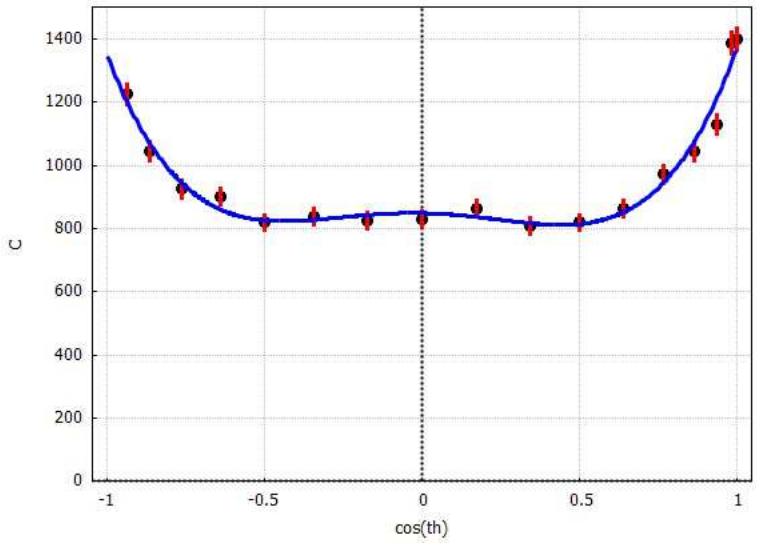

Figure 17: Counts vs. $\cos(\theta)$ Using All Terms through $P_4(\cos(\theta))$

11.4 Three Parameter Fit with Legendre Polynomials using lfit

Note that the values returned for the best fit values of a_1 and a_3 are much less (in magnitude) than the parameters which give the amplitudes of the Legendre polynomials which are even functions of their argument $x = \cos\theta$.

Retaining only the three (even) dominant terms of the five parameter fit, we seek a three parameter fit instead, using the model, again with $x = \cos\theta$,

$$C = a_0\, P_0(x) + a_2\, P_2(x) + a_4\, P_4(x). \tag{11.5}$$

```
(%i19) param_list : [a0,a2,a4]$
(%i20) y_expr_noun : a0*'P(0,x) + a2*'P(2,x) + a4*'P(4,x);
(%o20) 'P(4,x)*a4+'P(2,x)*a2+'P(0,x)*a0
(%i21) y_expr : %, nouns;
(%o21) (a4*(35*x^4-30*x^2+3))/8+(a2*(3*x^2-1))/2+a0
(%i22) out : lfit (dataM, sigL, y_expr,param_list);
 ivar =  x
 num_data =  17
 num_param =  3
 dof =  14
 chi2/dof =  1.25646
 chi2_prob =  22.6073 %
a0 =  907.175  +/-  7.7342
a2 =  260.479  +/-  15.8578
a4 =  193.667  +/-  20.0715
(%o22) [[a0 = 907.175,a2 = 260.479,a4 = 193.667],[7.7342,15.8578,20.0715],
        17.5905,0.226073]
(%i23) yfit3 : y_expr, out[1];
(%o23) 24.2083*(35*x^4-30*x^2+3)+130.24*(3*x^2-1)+907.175
(%i24) qdraw (xr(-1.05,1.05),yr(0,1500),
           more (xlabel = "cos(th)", ylabel = "C"),
              pts (xCL, pc(black), ps(1)),
               errorbars (xCL, sigL, lw(3),lc(red)),
              ex1(yfit3,x,-1,1))$
```

which produces the plot

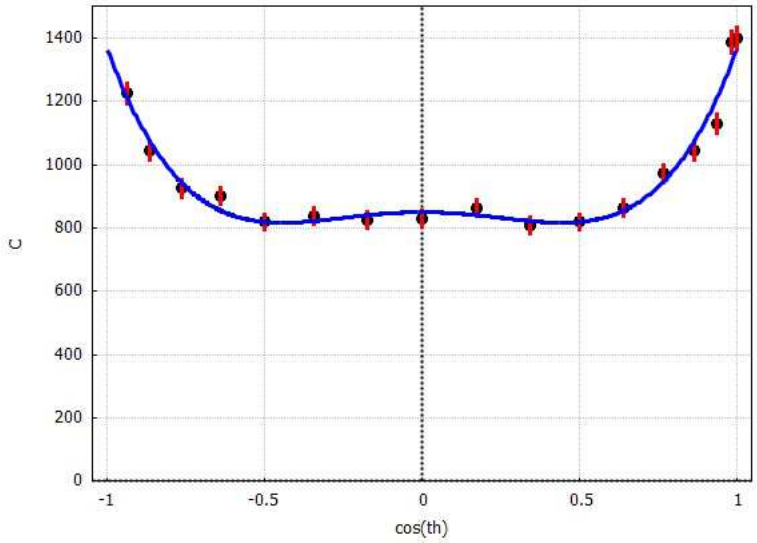

Figure 18: Counts vs. $\cos(\theta)$ Using Only Even Terms through $P_4(\cos(\theta))$

Note that the three parameter fit has a higher χ^2 probability than the five parameter fit, and is thus a better fit to the given data.

12 Nonlinear Least Squares Fit to Cooling Coffee Data

12.1 Massaging the Data

We use the coffee cooling experimental data in the file **coffee.dat**, data we used in Maxima by Example, Chapter 2. The first column is the time in minutes. The second column is the temperature (degrees Celsius) recorded for a cup of black coffee. The third column is the temperature recorded for a cup of coffee which has been cooled (at $t = 0$) by the addition of cream (white coffee).

We seek to fit the data for the black coffee, using a model based on an exponential decrease (Newton's law of cooling), and let t_c be the characteristic cooling time in units of minutes. Assuming the ambient room temperature is 17 deg Celsius, and the initial temperature of the black coffee is 82.3 deg Celsius, a one parameter model of the data could be

$$T = 17 + 65.3\, e^{-t/t_c}, \tag{12.1}$$

since for large enough times $t >> t_c$ the second term can be ignored compared with the first term.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/coffee.dat"$
(%i3) printfile(fname)$
0        82.3        68.8
2        78.5        64.8
4        74.3        62.1
6        70.7        59.9
8        67.6        57.7
10        65.0        55.9
12        62.5        53.9
14        60.1        52.3
16        58.1        50.8
18        56.1        49.5
20        54.3        48.1
22        52.8        46.8
24        51.2        45.9
26        49.9        44.8
28        48.6        43.7
30        47.2        42.6
32        46.1        41.7
34        45.0        40.8
36        43.9        39.9
38        43.0        39.3
40        41.9        38.6
42        41.0        37.7
44        40.1        37.0
46        39.5        36.4
(%i4) dataM : read_matrix (fname);
(%o4) matrix([0,82.3,68.8],[2,78.5,64.8],[4,74.3,62.1],[6,70.7,59.9],
             [8,67.6,57.7],[10,65.0,55.9],[12,62.5,53.9],[14,60.1,52.3],
             [16,58.1,50.8],[18,56.1,49.5],[20,54.3,48.1],[22,52.8,46.8],
             [24,51.2,45.9],[26,49.9,44.8],[28,48.6,43.7],[30,47.2,42.6],
             [32,46.1,41.7],[34,45.0,40.8],[36,43.9,39.9],[38,43.0,39.3],
             [40,41.9,38.6],[42,41.0,37.7],[44,40.1,37.0],[46,39.5,36.4])
(%i5) tL : list_matrix_entries (col(dataM,1));
(%o5) [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46]
(%i6) TbL : list_matrix_entries (col(dataM,2));
(%o6) [82.3,78.5,74.3,70.7,67.6,65.0,62.5,60.1,58.1,56.1,54.3,52.8,51.2,49.9,
       48.6,47.2,46.1,45.0,43.9,43.0,41.9,41.0,40.1,39.5]
(%i7) TcL : list_matrix_entries (col(dataM,3));
(%o7) [68.8,64.8,62.1,59.9,57.7,55.9,53.9,52.3,50.8,49.5,48.1,46.8,45.9,44.8,
       43.7,42.6,41.7,40.8,39.9,39.3,38.6,37.7,37.0,36.4]
(%i8) length(tL);
(%o8) 24
```

In the absence of information about measurement errors, we set the elements of **sigL** equal to unity.

```
(%i9) sigL : makelist (1,j,1,24);
(%o9) [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
```

```
(%i10) black_expr : 17 + 65.3*exp (-t/tc);
(%o10) 65.3*%e^-(t/tc)+17
(%i11) dataMB : apply ('matrix, xyList(tL,TbL));
(%o11) matrix([0,82.3],[2,78.5],[4,74.3],[6,70.7],[8,67.6],[10,65.0],[12,62.5],
              [14,60.1],[16,58.1],[18,56.1],[20,54.3],[22,52.8],[24,51.2],
              [26,49.9],[28,48.6],[30,47.2],[32,46.1],[34,45.0],[36,43.9],
              [38,43.0],[40,41.9],[42,41.0],[44,40.1],[46,39.5])
```

12.2 Using Vsearch (Visual Search) for One Parameter Fit

We first use **Vsearch** with **qdraw** to get a rough feel for an appropriate size of the characteristic cooling time t_c. Recall that the syntax is

```
Vsearch (data-matrix,sigmaL,y-expr,param-list, param-guess-list)
```

The model expression is a blue curve, the data is in black and red. After loading **draw** and **qdraw**, we start with $t_c = 1$.

```
(%i12) load(draw);
(%o12) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i13) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o13) "c:/work9/qdraw.mac"
(%i14) Vsearch(dataMB,sigL,black_expr,[tc],[1])$
  param_list =  [tc = 1]
  yfit_expr =  65.3*%e^-t+17
  chiSq =  32896.9
```

which produces the plot (the data is in black and red; the model expression curve is in blue)

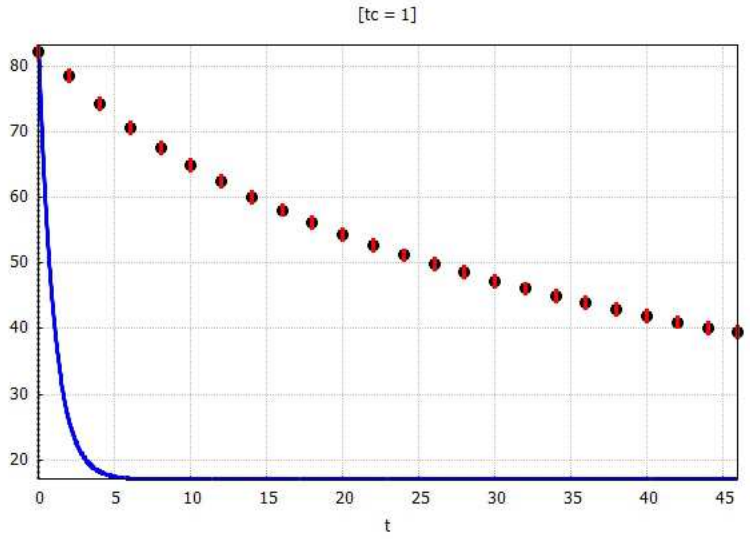

Figure 19: Temperature vs. time (t) for $t_c = 1$

Now increase the value of the characteristic time scale to $t_c = 10$ so the model temperature curve drops less rapidly to zero.

```
(%i15) Vsearch(dataMB,sigL,black_expr,[tc],[10])$
  param_list =  [tc = 10]
  yfit_expr =  65.3*%e^-(0.1*t)+17
  chiSq =  13937.1
```

which produces the plot

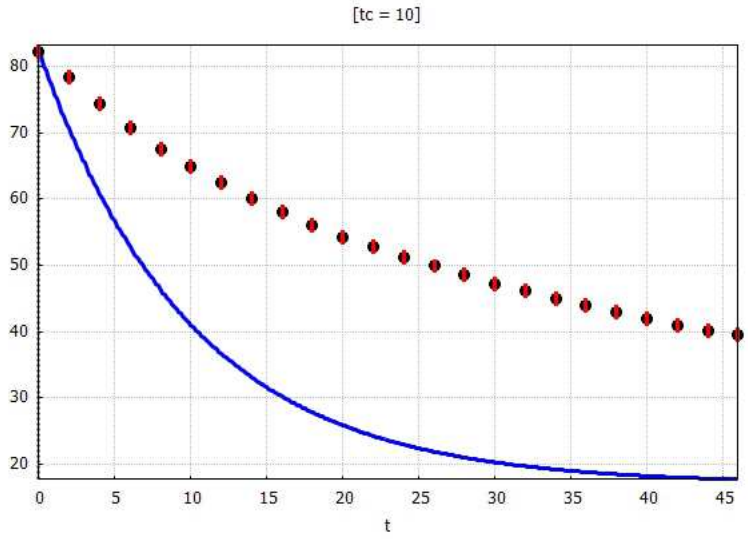

Figure 20: Temperature vs. time (t) for $t_c = 10$

which is still too small a characteristic cooling time. We next try $t_c = 100$.

```
(%i16) Vsearch(dataMB,sigL,black_expr,[tc],[100])$
  param_list =  [tc = 100]
  yfit_expr =  65.3*%e^-(0.01*t)+17
  chiSq =  5682.54
```

which produces the plot

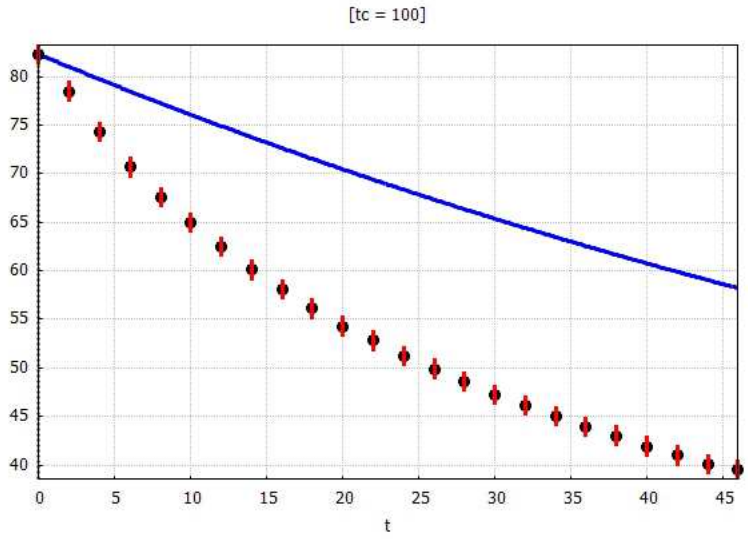

Figure 21: Temperature vs. time (t) for $t_c = 100$

which is too large a characteristic cooling time. We next try $t_c = 30$.

```
(%i17) Vsearch(dataMB,sigL,black_expr,[tc],[30])$
  param_list =  [tc = 30]
  yfit_expr =  65.3*%e^-(0.0333333*t)+17
  chiSq =  647.939
```

which produces the plot

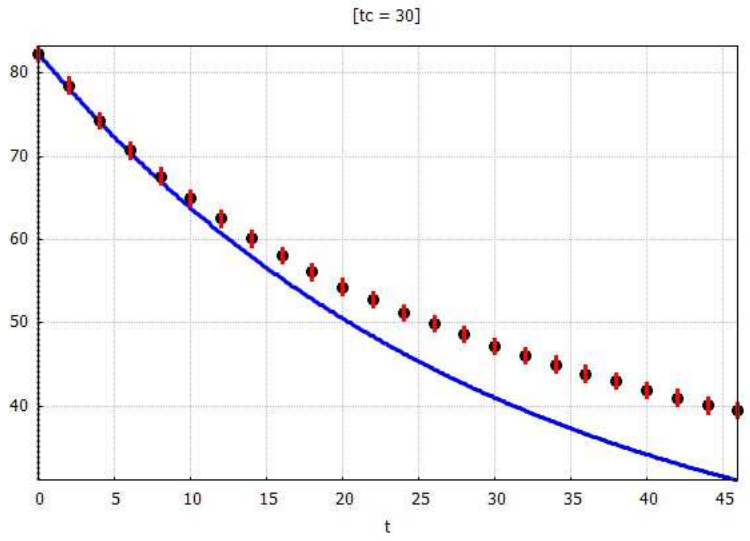

Figure 22: Temperature vs. time (t) for $t_c = 30$

which shows we are getting close to a reasonable characteristic cooling time t_c, and we try $t_c = 40$ next:

```
(%i18) Vsearch(dataMB,sigL,black_expr,[tc],[40])$
  param_list =  [tc = 40]
  yfit_expr =  65.3*%e^-(0.025*t)+17
  chiSq =  78.5918
```

which produces the plot

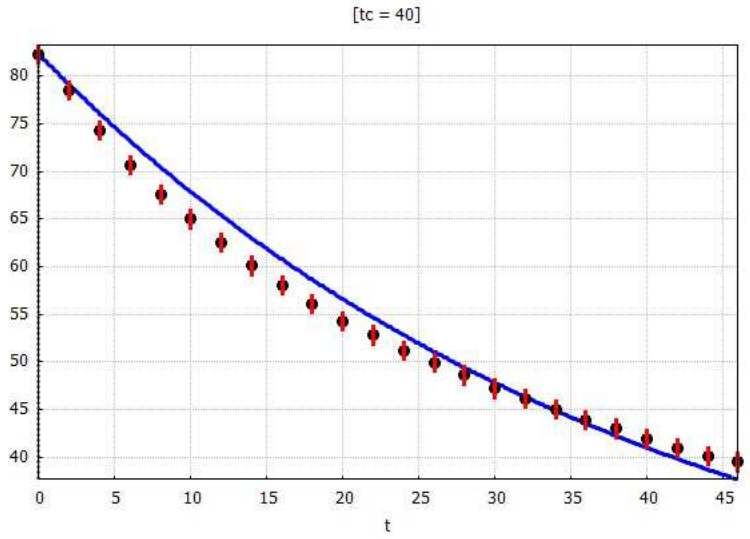

Figure 23: Temperature vs. time (t) for $t_c = 40$

12.3 Using grid_search for a Three Parameter Fit

We seek to use the three parameter expression

$$T = a_1 + a_2\, e^{-t/a_3} \tag{12.2}$$

to fit the black coffee cooling data in **coffee.dat**, using the **fit.mac** function

```
grid_search(data_matrix, sigma_list,ymodel_expr,paramL,param_startL,stepFactor).
```

The last argument **stepFactor** is used to help define the initial step size for each parameter, according to the code line

```
for i thru Nparam do  deltaA[i] : stepFactor*abs (ac[i]),
```

in which **ac[i]** is the starting value of the i'th parameter. If you get problem messages from the program **grid_search**, you should first try decreasing the value of **stepFactor** so that the program begins with smaller steps in parameter space.

Each step of the search adjusts the value of one of the parameters at a time, seeking for the approximate position of the valley bottom in values of χ^2, and prints out the last three values of χ^2, **chiSq1**, **chiSq2**, and **chiSq3**, which will give you an idea of how steep the ravine is for that parameter and its current step size.

If the grid search of parameter space is working correctly, the values of χ^2 should steadily decrease. Remember that χ^2 is inherently a non-negative number.

This type of grid search will give poor results if the values of the parameters (to achieve a minimum in χ^2) are strongly correlated.

grid_search returns a list of lists:

```
[ [a1, a2, ....], [da1, da2, ...]]
```

in which the parameter uncertainties **daj** are defined (assuming a local parabola fit) by how large a change in the parameter is required to cause the value of the non-negative number χ^2 to change by the value 1.

We continue with the **coffee.dat** data matrix **dataMB** and **sigL** defined above but defining a three parameter model **myexpr**.

Each "trial" adjusts separately the values of each of the three parameters, looking for a minimum in the value of χ^2. At the end of each trial, you are asked to either enter **c;** (to go on to the next trial) or **s;** (to stop the grid search).

```
(%i19)  myexpr : a1 + a2*exp (-t/a3)$
(%i20) grid_search(dataMB, sigL, myexpr,[a1,a2,a3],[17,65.3,40],0.5);
 ymodel =  a2*%e^-(t/a3)+a1
============================
 trial =   1  starting chiSqr =   78.5918
 starting parameter values and step sizes for this trial
  1   17   8.5
  2   65.3   32.65
  3   40   20.0
-------------------------------------------------
  parameter  1
 chiSq1 =  2219.68  chiSq2 =  78.5918  chiSq3 =  1405.5
 ac[j] =   16.0022  dac[j] =  0.204124  deltaA[j] =    0.288675
 chiSqr =    54.6987
-------------------------------------------------
  parameter  2
 chiSq1 =  10555.4  chiSq2 =  54.6987  chiSq3 =  9925.79
 ac[j] =   64.7955  dac[j] =  0.323507  deltaA[j] =    0.457508
 chiSqr =    52.2666
-------------------------------------------------
```

```
  parameter  3
 chiSq1 =  1128.49  chiSq2 =  52.2666  chiSq3 =  4715.61
  chiSqr-minimum is less than zero  using parabola fit
 compute using corresponding parameter value
 ac[j] =   46.2498  dac[j] =  0.373341  deltaA[j] =    0.527984
 chiSqr =    163.46
 chiSqr increased:   chiOld =  78.5918  new chiSqr =  163.46
Enter s; to stop trials, c; to continue
c;
============================
 trial =   2  starting chiSqr =   163.46
 starting parameter values and step sizes for this trial
  1   16.0022   0.288675
  2   64.7955   0.457508
  3   46.2498   0.527984
-------------------------------------------------
  parameter  1
 chiSq1 =  47.0253  chiSq2 =  46.3918  chiSq3 =  49.7582
 ac[j] =   13.7914  dac[j] =  0.204124  deltaA[j] =    0.288675
 chiSqr =    46.1584
-------------------------------------------------
  parameter  2
 chiSq1 =  47.9371  chiSq2 =  46.1584  chiSq3 =  48.7984
 ac[j] =   64.8401  dac[j] =  0.307798  deltaA[j] =    0.435292
 chiSqr =    46.1374
-------------------------------------------------
  parameter  3
 chiSq1 =  46.1374  chiSq2 =  45.3979  chiSq3 =  46.8094
 ac[j] =   45.8043  dac[j] =  0.509126  deltaA[j] =    0.720012
 chiSqr =    45.3717
Enter s; to stop trials, c; to continue
c;
============================
 trial =   3  starting chiSqr =   45.3717
 starting parameter values and step sizes for this trial
  1   13.7914   0.288675
  2   64.8401   0.435292
  3   45.8043   0.720012
-------------------------------------------------
  parameter  1
 chiSq1 =  45.4999  chiSq2 =  45.3717  chiSq3 =  49.2484
 ac[j] =   13.9265  dac[j] =  0.203997  deltaA[j] =    0.288496
 chiSqr =    44.9331
-------------------------------------------------
  parameter  2
 chiSq1 =  46.8037  chiSq2 =  44.9331  chiSq3 =  47.041
 ac[j] =   64.8531  dac[j] =  0.308633  deltaA[j] =    0.436474
 chiSqr =    44.9314
-------------------------------------------------
  parameter  3
 chiSq1 =  48.6985  chiSq2 =  44.9314  chiSq3 =  45.1024
 ac[j] =   45.4756  dac[j] =  0.513108  deltaA[j] =    0.725644
 chiSqr =    44.5209
Enter s; to stop trials, c; to continue
s;
====================================
(%o20) [[13.9265,64.8531,45.4756],[0.203997,0.308633,0.513108]]
```

With three trials completed, we chose to stop the grid search, having very rough values $a_1 \approx 14$, $a_2 \approx 65$, and $a_3 \approx 46$. We now proceed to use the non-linear fit function **`nlfit`**, using these rough values as starting guesses.

12.4 Three Parameter Fit Using nlfit

We try a three parameter fit using **nlfit**, using as starting values the final parameters found using **grid_search** above.

Note that we make the replacement $a_3 \rightarrow t_c$ in the definition of the model expression **myexpr**, and in the list of parameter names.

```
(%i21) myexpr : a1 + a2*exp(-t/tc);
(%o21) a2*%eˆ-(t/tc)+a1
(%i22) out : nlfit (dataMB,sigL,myexpr,[a1,a2,tc],[14,65,46]);
 Ndata =  24
 Nparam =  3
 dof =  21
 ivar =  t
start:   params:    [a1 = 14.0,a2 = 65.0,tc = 46.0]        chi2 =  46.8298
---------------------------------------------------------------
  n     lam
  1    0.001
               p_oldL =  [14.0,65.0,46.0]
               p_newL =  [30.1843,50.1806,26.8754]    chi2_new =  11.5832
  2    1.0e-4
               p_oldL =  [30.1843,50.1806,26.8754]
               p_newL =  [32.287,49.5529,24.5566]    chi2_new =  2.38017
  3    1.0e-5
               p_oldL =  [32.287,49.5529,24.5566]
               p_newL =  [32.2182,49.6507,24.6676]    chi2_new =  2.33385
  4    1.0e-6
               p_oldL =  [32.2182,49.6507,24.6676]
               p_newL =  [32.2146,49.6535,24.6717]    chi2_new =  2.33385
---------------------------------------------------------------
 chi2/dof =  0.111136
 chi2_prob =  100.0 %
---------------------------------------------------------------
a1 =  32.2146  +/-  1.59692
a2 =  49.6535  +/-  1.36525
tc =  24.6717  +/-  1.768
(%o22) [[a1 = 32.2146,a2 = 49.6535,tc = 24.6717],[1.59692,1.36525,1.768],
       2.33385,1.0]
(%i23) yfit : myexpr, out[1];
(%o23) 49.6535*%eˆ-(0.0405322*t)+32.2146
(%i24) y_gaussian_PE (dataMB, 21, yfit);
(%o24) 0.222247
```

If we use the assumption of Gaussian distribution errors for the temperature measurements, and the resulting estimate of the probable error thus returned by **y_gaussian_PE**, we can redefine the elements of **sigL** to be **0.22** and redo the fit using **nlfit**:

```
(%i25) sigL : makelist (0.22,j,1,24);
(%o25) [0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,
        0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22,0.22]
(%i26) out : nlfit (dataMB,sigL,myexpr,[a1,a2,tc],[14,65,46]);
 Ndata =  24
 Nparam =  3
 dof =  21
 ivar =  t
start:   params:    [a1 = 14.0,a2 = 65.0,tc = 46.0]        chi2 =  967.557
---------------------------------------------------------------
  n     lam
  1    0.001
               p_oldL =  [14.0,65.0,46.0]
               p_newL =  [30.1843,50.1806,26.8754]    chi2_new =  239.321
  2    1.0e-4
               p_oldL =  [30.1843,50.1806,26.8754]
               p_newL =  [32.287,49.5529,24.5566]    chi2_new =  49.1771
  3    1.0e-5
               p_oldL =  [32.287,49.5529,24.5566]
               p_newL =  [32.2182,49.6507,24.6676]    chi2_new =  48.2201
```

```
  4    1.0e-6
              p_oldL =  [32.2182,49.6507,24.6676]
              p_newL =  [32.2146,49.6535,24.6717]    chi2_new =  48.22
------------------------------------------------------------
 chi2/dof =  2.29619
 chi2_prob =  0.0641768 %
------------------------------------------------------------
a1 =  32.2146  +/-  0.351322
a2 =  49.6535  +/-  0.300356
tc =  24.6717  +/-  0.38896
(%o26) [[a1 = 32.2146,a2 = 49.6535,tc = 24.6717],[0.351322,0.300356,0.38896],
        48.22,6.41768e-4]
(%i27) yfit : myexpr, out[1];
(%o27) 49.6535*%e^-(0.0405322*t)+32.2146
(%i28) y_gaussian_PE (dataMB, 21, yfit);
(%o28) 0.222247
```

By changing the value of the elements of **sigL** in a trial and error method, each time invoking **nlfit** again, we can arrive at a combined instrumental error and random error estimate of the temperature measurement errors which finally gives reasonable values for the χ^2 probability and the reduced χ^2 value, χ^2/dof.

```
(%i29) sigL : makelist (0.3,j,1,24);
(%o29) [0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,
        0.3,0.3,0.3,0.3,0.3,0.3,0.3]
(%i30) out : nlfit (dataMB,sigL,myexpr,[a1,a2,tc],[14,65,46]);
 Ndata =  24
 Nparam =  3
 dof =  21
 ivar =  t
start:   params:    [a1 = 14.0,a2 = 65.0,tc = 46.0]        chi2 =  520.331
------------------------------------------------------------
  n     lam
  1    0.001
              p_oldL =  [14.0,65.0,46.0]
              p_newL =  [30.1843,50.1806,26.8754]    chi2_new =  128.702
  2    1.0e-4
              p_oldL =  [30.1843,50.1806,26.8754]
              p_newL =  [32.287,49.5529,24.5566]    chi2_new =  26.4464
  3    1.0e-5
              p_oldL =  [32.287,49.5529,24.5566]
              p_newL =  [32.2182,49.6507,24.6676]    chi2_new =  25.9317
  4    1.0e-6
              p_oldL =  [32.2182,49.6507,24.6676]
              p_newL =  [32.2146,49.6535,24.6717]    chi2_new =  25.9316
------------------------------------------------------------
 chi2/dof =  1.23484
 chi2_prob =  20.9068 %
------------------------------------------------------------
a1 =  32.2146  +/-  0.479076
a2 =  49.6535  +/-  0.409576
tc =  24.6717  +/-  0.5304
(%o30) [[a1 = 32.2146,a2 = 49.6535,tc = 24.6717],[0.479076,0.409576,0.5304],
        25.9316,0.209068]
```

We take the output of **nlfit** as the values of parameters for making a plot using **Vsearch**. We already loaded **draw** and **qdraw** above prior to using the graphics **Vsearch** method.

```
(%i31) Vsearch (dataMB,sigL,myexpr,[a1,a2,tc],[32.22,49.65,24.67])$
  param_list =  [a1 = 32.22,a2 = 49.65,tc = 24.67]
  yfit_expr =  49.65*%e^-(0.0405351*t)+32.22
  chiSq =  2.33405
```

which produces the plot

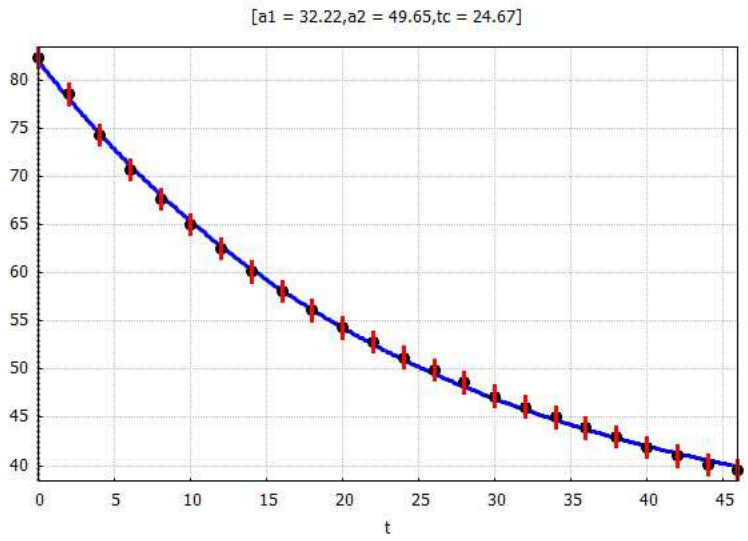

Figure 24: Temperature vs. time three parameter fit using nlfit

We see that the cooling coffee data can be fit closely if we use a three parameter fit. We would expect a poorer fit if we only allowed one or two parameters to be adjusted (you should try this).

13 Ex. 8: Nonlinear Fit of the Decay of Two Excited States Plus Background

Quoting Bevington (3rd), pdf 156, with some additions,

> In a popular undergraduate physics laboratory experiment, a real silver quarter is irradiated with thermal neutrons to create two short-lived isotopes of silver, Ag_{47}^{108} and Ag_{47}^{110}, that subsequently decay by beta emission. Students count the emitted beta particles in 15-s intervals for about 4 min to obtain a decay curve. Data collected from such an experiment are listed in Table 8.1 and plotted on a semi-logarithmic graph in Figure 8.1. The data are reported at the end of each 15-s interval, just as they were recorded by a scaler. The data points do not fall on a straight line because the probability function that describes the process is the sum of two exponential functions plus a constant background. We can represent the decay by the model
>
> $$C(t) = a_1 + a_2\, e^{-t/a_4} + a_3\, e^{-t/a_5} \tag{13.1}$$
>
> where the parameter a_1 corresponds to the constant background radiation, and a_2 and a_3 correspond to the amplitudes of the two excited states with mean lives a_4 and a_5, respectively. We assume that the second term proportional to e^{-t/a_4} is the contribution due to the short-lived excited state, and the third term proportional to e^{-t/a_5} is the contribution due to the long-lived excited state, so $a_4 << a_5$. $C(t)$ represents the number of beta particles recorded by the detector during the 15 sec prior to the time t. Clearly, Equation (13.1) in not linear in the parameters a_4 and a_5, although it is linear in the parameters a_1, a_2, and a_3.

13.1 Interactive Look at the Raw Data

The data for this example is in our file **mbe14-fit8.dat**.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) fname : "c:/work9/mbe14-fit8.dat"$
(%i3) printfile(fname)$
Radioactive decay
    59
    15   775   27.8
    30   479   21.9
    45   380   19.5
    60   302   17.4
    75   185   13.6
    90   157   12.5
   105   137   11.7
   120   119   10.9
   135   110   10.5
   150    89    9.4
   165    74    8.6
   180    61    7.8
   195    66    8.1
   210    68    8.2
   225    48    6.9
   240    54    7.3
   255    51    7.1
   270    46    6.8
   285    55    7.4
   300    29    5.4
   315    28    5.3
   330    37    6.1
   345    49    7.0
   360    26    5.1
   375    35    5.9
   390    29    5.4
   405    31    5.6
   420    24    4.9
   435    25    5.0
   450    35    5.9
   465    24    4.9
   480    30    5.5
   495    26    5.1
   510    28    5.3
   525    21    4.6
   540    18    4.2
   555    20    4.5
   570    27    5.2
   585    17    4.1
   600    17    4.1
   615    14    3.7
   630    17    4.1
   645    24    4.9
   660    11    3.3
   675    22    4.7
   690    17    4.1
   705    12    3.5
   720    10    3.2
   735    13    3.6
   750    16    4.0
   765     9    3.0
   780     9    3.0
   795    14    3.7
   810    21    4.6
   825    17    4.1
   840    13    3.6
   855    12    3.5
   870    18    4.2
   885    10    3.2
```

The first line of this data file is a title, the second line is the advertised number of data points. In each of the following lines we have the time (t in sec), the number of betas recorded in the prior 15 sec (C), and the square

root of the count ($\sqrt{C}$). Note that we cannot use **read_matrix** with this data file, since its use would result in the error message "matrix: all rows must be the same length." We can use **read_nested_list** to produce a (nested) list of the data file, use **fll** to look at the first element, last element, and the length, and then use **rest (alist, 2)** to remove the first two sub-lists (the first sub-list contains the title line and the second sub-list contains the advertised number of data points). We then have the option of using **apply ('matrix, a-nested-list)** to produce a matrix.

```
(%i4) data8L : read_nested_list(fname)$
(%i5) fll (data8L);
(%o5) [[Radioactive,decay],[885,10,3.2],61]
(%i6) head (data8L);
(%o6) [[Radioactive,decay],[59],[15,775,27.8],[30,479,21.9],[45,380,19.5],
       [60,302,17.4]]
(%i7) rest([a,b,c,d],2);
(%o7) [c,d]
(%i8) data8L : rest (data8L,2)$
(%i9) fll (data8L);
(%o9) [[15,775,27.8],[885,10,3.2],59]
(%i10) data8M : apply ('matrix,data8L)$
(%i11) row (data8M,1);
(%o11) matrix([15,775,27.8])
```

We let **tL** be the list of the times, **CL** be the list of the raw counts, and **sigL** be the list of the square-roots of the raw counts.

```
(%i12) tL : list_matrix_entries (col (data8M,1))$
(%i13) fll (tL);
(%o13) [15,885,59]
(%i14) CL : list_matrix_entries (col (data8M,2))$
(%i15) fll (CL);
(%o15) [775,10,59]
(%i16) sigL : list_matrix_entries (col (data8M,3))$
(%i17) fll (sigL);
(%o17) [27.8,3.2,59]
```

We first make a linear plot of the raw data, using our homemade **xyList** function:

```
(%i18) tCL : xyList (tL, CL)$
(%i19) fll (tCL);
(%o19) [[15,775],[885,10],59]
(%i20) load(draw)$
(%i21) load(qdraw)$
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%i22) qdraw (pts (tCL,pc(black),ps(1)),
         xr (0,900), yr (0,800),
          more (xlabel = "t", ylabel = "C"))$
```

which produces the plot

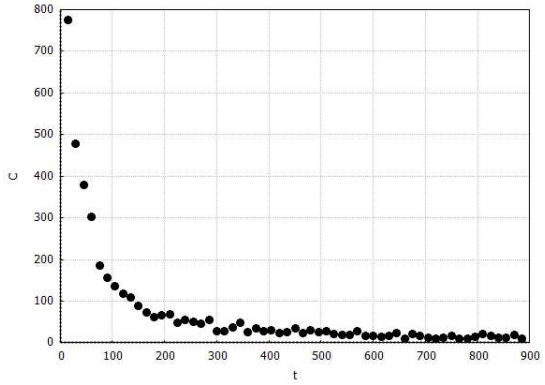

Figure 25: Linear Plot of Raw Data: Counts vs. time

We next make a semi-log plot, $\ln(C)$ vs. t of the raw data. Remember that in Maxima, **log** returns the natural logarithm. Note that if one of the counts was 0, **log(CL),numer** would return a **log(0)** error.

```
(%i23) lnCL : log (CL),numer$
(%i24) fll (lnCL);
(%o24) [6.65286,2.30259,59]
(%i25) t_lnCL : xyList (tL, lnCL)$
(%i26) fll (t_lnCL);
(%o26) [[15,6.65286],[885,2.30259],59]
(%i27) qdraw (pts (t_lnCL,pc(black),ps(1)),
         xr (0,900), yr (2,7),
          more (xlabel = "t", ylabel = "ln(C)"))$
```

which produces the plot

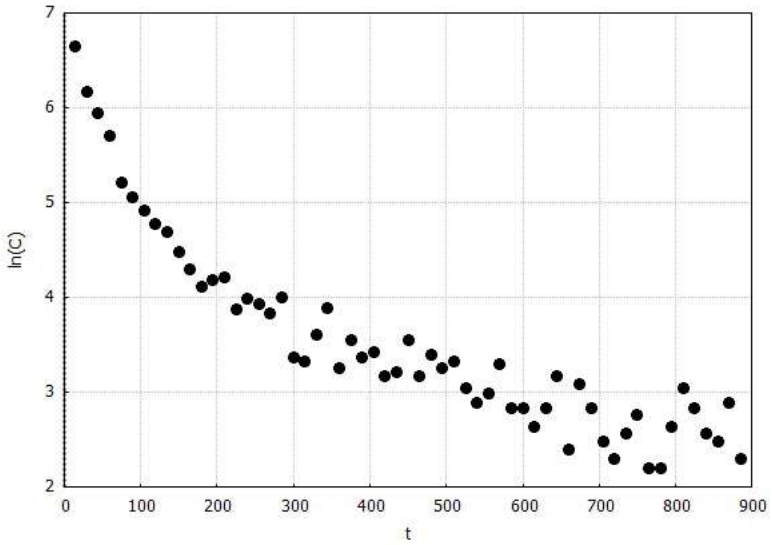

Figure 26: Raw Data: $\ln(C)$ vs. t

13.2 Estimates of the Mean Lifetime and Amplitude of each Excited State

After subtracting the constant background beta radiation counts from the raw data, we can consider separately the "early data", corresponding to $t < 200$ sec, in which the short-lived excited state dominates the measured counts, and the "late data", corresponding to $t > 200$ sec, in which the long-lived state dominates the measured counts.

We can estimate the mean lifetime and amplitude of the long-lived excited state by using only the corrected late data points, assuming the corrected late data points are approximately given by the third term of Eq. (13.1), and fitting a straight line to the natural log of the late counts vs. time.

We can then subtract the estimated contribution of the long-lived excited state from the early corrected data, and assume the twice-corrected early data points are approximately represented by the second term of 13.1, and fitting another straight line to the natural log of the twice-corrected early data points, which will yield estimates of the mean lifetime and amplitude of the short-lived state.

13.2.1 Subtraction of Background Beta Radiation Counts from Raw Data

A separate measurement of the beta particle background, before irradiation of the silver, produced the value $a_1 \approx 10$ beta particles per 15 sec. Let **Cmb** (C minus background) be the beta count numbers after subtracting

the approximate background.

```
(%i28) Cmb : CL - 10$
(%i29) fll (Cmb);
(%o29) [765,0,59]
```

13.2.2 Long-lived State Properties from Late Data Points

We can first work with only the large time (corrected) data points, for which $t > 200$. We assume they are mainly due to the long-lived state decay, represented by the term $a_3\, e^{-t/a_5}$.

To produce a list of times, corrected counts, and count errors corresponding to the requirement $t > 200$, we use our homemade function **`pos_GT(alist,anumber)`** which returns the first list element number of **`alist`** for which the list element is greater than **`anumber`**.

Applied to the list **`tL`**, we find that the fourteenth element of **`tL`** is the first element of **`tL`** which is greater than 200, and we can then define **`tL_late`** as the list of times produced by stripping away the first thirteen elements of **`tL`**. We can then strip away the first thirteen elements of **`Cmb`** and **`sigL`** to define the corresponding corrected late counts and count errors.

```
(%i30) pos_GT (tL,200);
(%o30) 14
(%i31) tL_late : rest(tL,13)$
(%i32) fll (tL_late);
(%o32) [210,885,46]
(%i33) Cmb_late : rest(Cmb,13)$
(%i34) fll (Cmb_late);
(%o34) [58,0,46]
(%i35) sigL_late : rest (sigL, 13)$
(%i36) fll (sigL_late);
(%o36) [8.2,3.2,46]
```

We are going to fit the natural logarithm of the late corrected counts to a straight line model in order to estimate the lifetime and amplitude of the long-lived decay contribution. We need to omit data points for which the late corrected counts are less than or equal to 0, since Maxima's **`log(x)`** function returns an error if $x \leq 0$.

We can use our homemade function **`positions_LE(alist,anumber)`** which returns the list of positions of elements which are less than or equal to the second argument. We can then use our homemade function **`Remove (L, nL)`** to return a list which omits the elements whose positions are in the list **`nL`**, thus defining **`tL_late_pos`** and corresponding lists for the counts and count errors. We finally take the natural log of the resulting count list, producing **`ln_Cmb_late_pos`**.

```
(%i37) pL : positions_LE (Cmb_late,0);
(%o37) [35,38,39,46]
(%i38) Cmb_late_pos : Remove (Cmb_late,pL)$
(%i39) fll (Cmb_late_pos);
(%o39) [58,8,42]
(%i40) tL_late_pos : Remove (tL_late,pL)$
(%i41) fll (tL_late_pos);
(%o41) [210,870,42]
(%i42) sigL_late_pos : Remove (sigL_late,pL)$
(%i43) fll (sigL_late_pos);
(%o43) [8.2,4.2,42]
(%i44) ln_Cmb_late_pos : log (Cmb_late_pos),numer$
(%i45) fll (ln_Cmb_late_pos);
(%o45) [4.06044,2.07944,42]
(%i46) dataM2 : apply ('matrix, xyList (tL_late_pos, ln_Cmb_late_pos))$
(%i47) row (dataM2,1);
(%o47) matrix([210,4.06044])
```

For a rough estimate of the straight line fit, we leave the values of **sigL_late_pos** alone (ie., no attempt to adapt to the switch to the natural log of the late corrected counts). We then use

$$\ln(A\,B) = \ln(A) + \ln(B), \quad \ln\left(e^{A}\right) = A, \quad e^{\ln(A)} = A. \tag{13.2}$$

to derive estimates **a3e** and **a5e** for a_3 and a_5 from the straight line fit.

```
(%i48) out1 : fit_line (dataM2, sigL_late_pos);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
 num_data =  42
 num_param =  2
 dof =  40
 chi2/dof =  0.0184701
 chi2_prob =  100.0 %
a =  4.44145  +/-  2.49489
b =  -0.00396122  +/-  0.0040118
(%o48) [[a = 4.44145,b = -0.00396122],[2.49489,0.0040118],0.738804,1.0]
(%i49) [av, bv] : map ('rhs, out1[1]);
(%o49) [4.44145,-0.00396122]
(%i50) a5e : -1/bv;
(%o50) 252.448
(%i51) a3e : exp(av);
(%o51) 84.8976
```

13.2.3 Short-lived State Properties from Early Data Points

We now concentrate on the early data points $t < 200$.

```
(%i52) n200 : pos_GT (tL,200);
(%o52) 14
(%i53) tL_early : rest (tL,-(length(tL) - n200 + 1))$
(%i54) fll (tL_early);
(%o54) [15,195,13]
(%i55) Cmb_early : rest (Cmb,-(length(tL) - n200 + 1))$
(%i56) fll (Cmb_early);
(%o56) [765,56,13]
(%i57) sigL_early : rest (sigL,-(length(tL) - n200 + 1))$
(%i58) fll (sigL_early);
(%o58) [27.8,8.1,13]
```

We use our estimates **a3e** and **a53** to estimate the contribution of the long-lived state at early times to define the list **long_lived_early**, and subtract these values from the corrected early counts to find a list of counts approximately due just to the short-lived state.

We use the natural log of the early corrected counts due to the short-lived state to fit another straight line, thus obtaining estimates **a2e** for a_2 and **a4e** for a_4.

```
(%i59) long_lived_early : map ('lambda ([t],a3e*exp (-t/a5e)), tL_early)$
(%i60) fll (long_lived_early);
(%o60) [80.0001,39.213,13]
(%i61) C_short_early : Cmb_early - long_lived_early$
(%i62) fll (C_short_early);
(%o62) [685.0,16.787,13]
(%i63) ln_C_short_early : log (C_short_early)$
(%i64) fll (ln_C_short_early);
(%o64) [6.52942,2.8206,13]
(%i65) dataM1 : apply ('matrix, xyList (tL_early, ln_C_short_early))$
(%i66) row (dataM1,1);
(%o66) matrix([15,6.52942])
(%i67) out1 : fit_line (dataM1, sigL_early);
 fit model y(x) = a + b*x to given data
 a = y-intercept,  b = slope
 ivar =  x
```

```
 num_data =  13
 num_param =  2
 dof =  11
 chi2/dof =  7.18198e-4
 chi2_prob =  100.0 %
a =  6.5285  +/-  9.80892
b =  -0.0211175  +/-  0.0668757
(%o67) [[a = 6.5285,b = -0.0211175],[9.80892,0.0668757],0.00790018,1.0]
(%i68) [av, bv] : map (’rhs, out1[1]);
(%o68) [6.5285,-0.0211175]
(%i69) a4e : -1/bv;
(%o69) 47.3541
(%i70) a2e : exp(av);
(%o70) 684.368
```

13.3 Five Parameter Fit using nlfit

We can now find a five parameter fit using **`nlfit`**. We define the raw data (for all times) matrix **`dataM`** using the nested list **`tCL`** of the raw data of times and counts.

```
(%i71) dataM : apply (’matrix, tCL)$
(%i72) row (dataM,1);
(%o72) matrix([15,775])
(%i73) myexpr : a1 + a2*exp(-t/a4) + a3*exp(-t/a5);
(%o73) a3*%eˆ-(t/a5)+a2*%eˆ-(t/a4)+a1
(%i74) a1e : 10;
(%o74) 10
(%i75) out1 : nlfit (dataM,sigL,myexpr,[a1,a2,a3,a4,a5],[a1e,a2e,a3e,a4e,a5e]);
 Ndata =  59
 Nparam =  5
 dof =  54
 ivar =  t
start:   params:
                  [a1 = 10.0,a2 = 684.368,a3 = 84.8976,a4 = 47.3541,
                   a5 = 252.448]        chi2 =  116.335
------------------------------------------------------------
  n     lam
  1    0.001
              p_oldL =  [10.0,684.368,84.8976,47.3541,252.448]
              p_newL =  [10.6205,889.143,133.829,31.3675,177.516]
                          chi2_new =  120.391
 increase
  2    0.01
              p_oldL =  [10.0,684.368,84.8976,47.3541,252.448]
              p_newL =  [8.80802,884.114,113.44,34.9627,233.081]
                          chi2_new =  79.2254
  3    0.001
              p_oldL =  [8.80802,884.114,113.44,34.9627,233.081]
              p_newL =  [10.0939,956.901,125.39,34.47,210.619]    chi2_new =
                        66.2584
  4    1.0e-4
              p_oldL =  [10.0939,956.901,125.39,34.47,210.619]
              p_newL =  [10.1581,957.652,128.391,34.2386,209.419]
                          chi2_new =  66.0562
  5    1.0e-5
              p_oldL =  [10.1581,957.652,128.391,34.2386,209.419]
              p_newL =  [10.1732,957.981,128.636,34.2055,209.114]
                          chi2_new =  66.056
------------------------------------------------------------
 chi2/dof =  1.22326
 chi2_prob =  12.5774 %
------------------------------------------------------------
a1 =  10.1732  +/-  1.8923
a2 =  957.981  +/-  49.5375
a3 =  128.636  +/-  21.2337
a4 =  34.2055  +/-  2.51918
a5 =  209.114  +/-  31.5955
(%o75) [[a1 = 10.1732,a2 = 957.981,a3 = 128.636,a4 = 34.2055,a5 = 209.114],
       [1.8923,49.5375,21.2337,2.51918,31.5955],66.056,0.125774]
```

```
(%i76) yfit : myexpr, out1[1];
(%o76) 128.636*%e^-(0.00478209*t)+957.981*%e^-(0.029235*t)+10.1732
```

13.4 Linear Plots for Early and Late Times

Linear plot of the five parameter fit and the data points for **late times** $t = 200 - 900$ sec.

```
(%i77) load(draw);
(%o77) "C:/Program Files/Maxima-sbcl-5.36.1/share/maxima/5.36.1/share/draw/draw.lisp"
(%i78) load(qdraw);
" qdraw(...), qdensity(...), qdensity1(...), syntax: type qdraw(); "
(%o78) "c:/work9/qdraw.mac"
(%i79) CL_late : rest (CL,n200 -1)$
(%i80) fll (CL_late);
(%o80) [68,10,46]
(%i81) ptsL : xyList(tL_late,CL_late)$
(%i82) qdraw (pts (ptsL,pc(black),ps(1)),
            errorbars (ptsL, sigL_late, lw(3),lc(blue)),
                        ex1 (yfit, t ,200, 900, lc(brown)),
            xr (200,900), yr (0,80),
          more (xlabel = "t", ylabel = "C"))$
```

which produces the plot

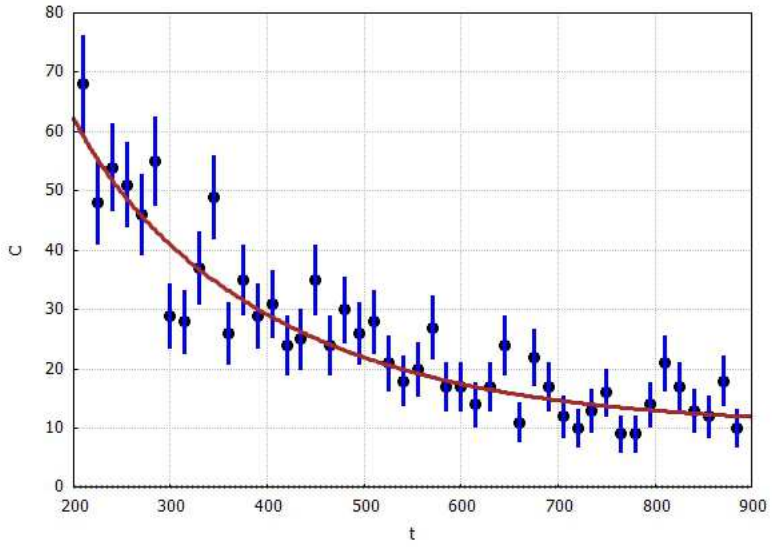

Figure 27: Linear Plot of Late Data and Five Parameter Fit

Linear plot of the five parameter fit and the data points for **early times** $t = 0 - 200$ sec.

```
(%i83) CL_early : rest (CL,-(length(tL) - n200 + 1));
(%o83) [775,479,380,302,185,157,137,119,110,89,74,61,66]
(%i84) length (CL_early);
(%o84) 13
(%i85) ptsL : xyList(tL_early,CL_early);
(%o85) [[15,775],[30,479],[45,380],[60,302],[75,185],[90,157],[105,137],
        [120,119],[135,110],[150,89],[165,74],[180,61],[195,66]]
(%i86) qdraw (pts (ptsL,pc(black),ps(1)),
            errorbars (ptsL, sigL_early, lw(3),lc(blue)),
                        ex1 (yfit, t ,0, 200, lc(brown)),
            xr (0,200), yr (0,800),
          more (xlabel = "t", ylabel = "C"))$
```

which produces the plot

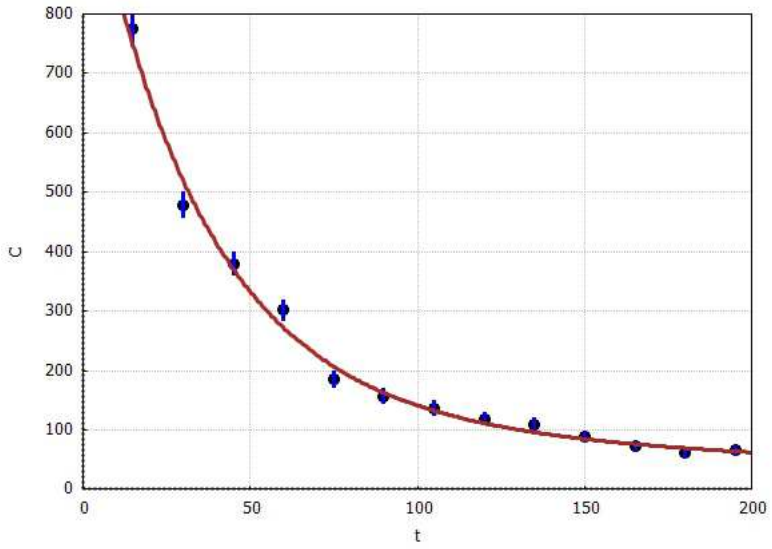

Figure 28: Linear Plot of Early Data and Five Parameter Fit

13.5 Four Parameter Fit Using nlfit

Try a four parameter fit, enforcing the measured background value.

```
(%i87) ymodel_4param : 10 + a2*exp(-t/a4) + a3*exp(-t/a5);
(%o87) a3*%eˆ-(t/a5)+a2*%eˆ-(t/a4)+10
(%i88) out1 : nlfit (dataM,sigL,ymodel_4param,[a2,a3,a4,a5],[a2e,a3e,a4e,a5e]);
 Ndata =  59
 Nparam =  4
 dof =  55
 ivar =  t
start:   params:    [a2 = 684.368,a3 = 84.8976,a4 = 47.3541,a5 = 252.448]
                        chi2 =  116.335
-----------------------------------------------------------------
  n     lam
  1    0.001
              p_oldL =  [684.368,84.8976,47.3541,252.448]
              p_newL =  [891.203,129.935,31.8702,191.623]    chi2_new =
                       104.811
  2    1.0e-4
              p_oldL =  [891.203,129.935,31.8702,191.623]
              p_newL =  [958.954,127.742,34.3479,211.159]    chi2_new =
                       66.0768
  3    1.0e-5
              p_oldL =  [958.954,127.742,34.3479,211.159]
              p_newL =  [957.992,127.205,34.3338,211.748]    chi2_new =
                       66.0637
  4    1.0e-6
              p_oldL =  [957.992,127.205,34.3338,211.748]
              p_newL =  [958.007,127.202,34.3335,211.756]    chi2_new =
                       66.0637
-----------------------------------------------------------------
 chi2/dof =  1.20116
 chi2_prob =  14.5928 %
-----------------------------------------------------------------
a2 =  958.007  +/-  49.2896
a3 =  127.202  +/-  14.7266
a4 =  34.3335  +/-  2.1612
a5 =  211.756  +/-  14.9265
(%o88) [[a2 = 958.007,a3 = 127.202,a4 = 34.3335,a5 = 211.756],
        [49.2896,14.7266,2.1612,14.9265],66.0637,0.145928]
```

We show the late data and fit on a linear plot.

```
(%i89) yfit : ymodel_4param, out1[1];
(%o89) 127.202*%eˆ-(0.00472242*t)+958.007*%eˆ-(0.029126*t)+10.0
(%i90) ptsL : xyList(tL_late,CL_late)$
(%i91) qdraw (pts (ptsL,pc(black),ps(1)),
            errorbars (ptsL, sigL_late, lw(3),lc(blue)),
                      ex1 (yfit, t ,200, 900, lc(brown)),
            xr (200,900), yr (0,80),
          more (xlabel = "t", ylabel = "C"))$
```

which produces the plot

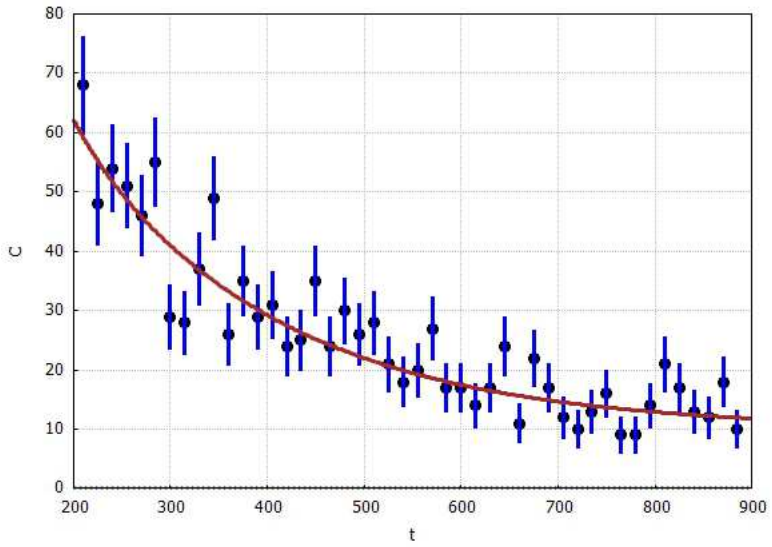

Figure 29: Linear Plot of Late Data and Four Parameter Fit

which shows that a four parameter fit, enforcing the measured value of the background, results in a poorer fit than the five parameter fit, in which we let all five parameters be adjustable.

14 General Model Fitting Background

This section presents some context about the least squares approach to fitting a model to a set of data.

Quoting Numerical Recipes (1992), Sec. 15.0,

> The basic approach in all cases is usually the same: You choose or design a *figure-of-merit function* ("merit function," for short) that measures the agreement between the data and the model with a particular choice of parameters. The merit function is conventionally arranged so that small values represent close agreement. The parameters of the model are then adjusted to achieve a minimum in the merit function, yielding "best-fit parameters." The adjustment process is thus a problem in minimization in many dimensions. ...however, there exist special, more efficient, methods that are specific to modeling, and we will discuss these in this chapter.
>
> There are important issues that go beyond the mere finding of best-fit parameters. Data are generally not exact. They are subject to measurement errors (called noise in the context of signal-processing). Thus, typical data never exactly fit the model that is being used, even when that model is correct. We need the means to assess whether or not the model is appropriate, that is, we

> need to test the goodness-of-fit against some useful statistical standard. We usually also need to know the accuracy with which parameters are determined by the data set. In other words, we need to know the likely errors of the best-fit parameters. Finally, it is not uncommon in fitting data to discover that the merit function is not unimodal, with a single minimum. In some cases, we may be interested in global rather than local questions. Not, "how good is this fit?" but rather, "how sure am I that there is not a very much better fit in some corner of parameter space?"
>
> . . . To be genuinely useful, a fitting procedure should provide (i) parameters, (ii) error estimates on the parameters, and (iii) a statistical measure of goodness-of-fit. When the third item suggests that the model is an unlikely match to the data, then items (i) and (ii) are probably worthless.

Quoting the forward of Statistical Methods for Experimental Physics, Frederick James, 2nd. ed., 2006

> A very common tacit assumption in the everyday use of statistics is that the set of data is large enough for asymptotic conditions to apply.

When we define the number of degrees of freedom in a model fit to data, in which the model has m unknown parameters and we have N data points, as dof $= \nu = N - m$, this may be true asymptotically, but not for smaller amounts of data.

On the use of "language" in statistics, we quote James (Ch. 1, Sec.2):

> Statistics, like any other branch of learning, has its own terminology which one has to become accustomed to. Certain confusion may, however, arise when the same term has a different meaning in statistics and in physics, or when the same concept has different names. In the former case we usually imply the statistical meaning (obliging the physicist to recognize and learn the difference); in the second case we often choose the physical term.
>
> An example of the first kind [same term, different meaning] is the following:

Physicists say	Statisticians say
Determine	Estimate
Estimate	Guess

> Thus the word "estimate" has different meaning in physics and in statistics. We use it as statisticians do. (We use three chapters to explain what statisticians mean thereby).
>
> An example of the second kind is "the demographic approach" to experimental physics. Much of statistics has been developed in connection with population studies (sociology, medicine, agriculture) and at the production line (industrial quality control). Then one is not able to study the whole population, so one "draws a sample". And the population exists in a real sense.
>
> In experimental physics, the set of all measurements under study corresponds to the "sample". Increasing the number of measurements, the physicist increases the "size of the sample", but he never attains the "population". Thus the "population" is an underlying abstraction which does not exist in any real sense. These "demographic" terms are therefore to some extent inappropriate and unnecessary, and we try to avoid some of them:

For the "demographic" term	we use the physics term
Sample	Data (set)
Draw a sample	Observe, measure
Sample of size N	N observations
Population	Observable space

Still, one has to be able to distinguish between, say, the mean of the data at hand, and the mean if the data set were infinite. When this distinction is necessary, we use **sample mean**, **sample variance**, etc. as contrasted to **parent mean**, **parent variance**, etc., or mean and variance of the underlying distribution. Thus

```
Parent mean = Mean of the underlying distribution = Population mean
```

We avoid the physical term "error", which is misleading, and use instead "variance of estimate", "confidence interval", or "interval estimate". We also try to avoid the words "precision" and "accuracy", because they are not well defined. In many books on statistics one finds whole chapters dealing with the "propagation of errors". Such a term, in our minds, is confusing. The corresponding notion here is "change of variables". Other topics which may seem to have got lost, may also sometimes be refound under other names. For instance, the term "regression analysis" is never used, but the techniques are treated under least-squares fits of linear models.

Despite James' avoidance of the term "errors" in his book, we continue to use language such as "propagation of errors" and "likely errors", because that language is so widespread in physics.

Quoting from Bevington, ch. 1

> *Error* is defined by Webster as "the difference between an observed or calculated value and the true value." Usually we do not know the "true" value; otherwise there would be no reason for performing the experiment. We may know approximately what it should be, however, either from earlier experiments or from theoretical predictions. Such approximations can serve as a guide but we must always determine in a systematic way from the data and the experimental conditions themselves how much confidence we can have in our experimental results.
>
> There is one class of error that we can deal with immediately: errors that originate from mistakes or blunders in measurement or computations. Fortunately, these errors are usually apparent either as obviously incorrect data points or as results that are not reasonably close to expected values. They can be classified as *illegitimate errors* and generally can be corrected by carefully repeating the operations.
>
> Our interest is in *uncertainties* introduced by random fluctuations in our measurements, and *systematic errors* that limit the precision and accuracy of our results in more or less well-defined ways. Generally, we refer to the uncertainties as the *errors* in our results, and the procedure for estimating them as *error analysis.*

Accuracy Versus Precision

It is important to distinguish between the terms *accuracy* and *precision.* The accuracy of an experiment is a measure of how close the result of the experiment is to the true value; the precision is a measure of how well the result has been determined, without reference to its agreement with the true value. The precision is also a measure of the reproducibility of the result in a given experiment.

And also quoting the Univ. of North Carolina measurement manual

When analyzing experimental data, it is important that you understand the difference between precision and accuracy. **Precision** indicates the quality of the measurement, without any guarantee that the measurement is "correct." **Accuracy**, on the other hand, assumes there is an ideal value, and tells you how far your answer is from that ideal, "right" answer. These concepts are directly related to **random** and **systematic** measurement errors.

Measurement errors may be classified as either **random** or **systematic**, depending on how the measurement was obtained (an intrument could cause a random error in one situation and a systematic error in another).

Random errors are statistical fluctuations (in either direction) in the measured data due to the precision limitations of the measurement device. Random errors can be evaluated through statistical analysis and can be reduced by averaging over a large number of observations (see standard error).

Systematic errors are reproducible inaccuracies that are consistently in the same direction. These errors and difficult to detect and cannot be analyzed statistically. If a systematic error is identifed when calibrating against a standard, applying a correction or correction factor to compensate for the effect can reduce the bias. Unlike random errors, systematic errors cannot be detected or reduced by increasing the number of observations.

. . . Gross personal errors, sometimes called **mistakes** or **blunders**, should be avoided and corrected if discovered. As a rule, personal errors are excluded from the error analysis discussion because it is generally assumed that the experimental result was obtained by following correct procedures. The term "human error" should also be avoided in error analysis discussions because it is too general to be useful.

14.1 Propagation of Errors

Suppose the quantity y is some function of measured parameters a and b, which each have some estimated uncertainty $\delta a \equiv \sigma_a$, $\delta b \equiv \sigma_b$, and

$$y = f(a, b) \tag{14.1}$$

The uncertainty in y if b were exactly known would be

$$\delta y_a = \frac{\partial f(a,b)}{\partial a}\,\sigma_a \tag{14.2}$$

and likewise the uncertainty in y if a were exactly known would be

$$\delta y_b = \frac{\partial f(a,b)}{\partial b}\,\sigma_b \tag{14.3}$$

If both a and b have uncertainties then we assume that the uncertainties add in quadrature in the sense

$$(\delta y)^2 \equiv \sigma_y^2 = (\delta y_a)^2 + (\delta y_b)^2 \tag{14.4}$$

or

$$\sigma_y^2 = \left[\frac{\partial f(a,b)}{\partial a}\right]^2 (\sigma_a)^2 + \left[\frac{\partial f(a,b)}{\partial b}\right]^2 (\sigma_b)^2 \tag{14.5}$$

Thus if $y = a - b$ then $\sigma_y^2 = \sigma_a^2 + \sigma_b^2$.

If y depends on more than two measured or observed quantities, the above approach can be easily extended to obtain σ_y^2 appropriate to the situation, as we will see in the next section.

14.2 Moments of a Distribution: Mean, Variance, Standard Deviation

This section in included so we can remind the reader about the differences between the "**measured mean**" and the "**true mean**" and likewise for the variance. We also review the usefulness of the Gaussian (Normal) statistical distribution for error analysis.

If there are N elements $\{x_1, \ldots, x_N\}$ in the set of data, and we assume each element is drawn from the same parent distribution, then each x_i has the same uncertainty

$$\delta x_i \equiv \sigma_i = \hat{\sigma} \text{ for all } \quad i \tag{14.6}$$

The **sample mean**, or the **measured mean** of the values x_i is the "arithmetic mean"

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{14.7}$$

In the same way you can calculate the mean value of any function$f(x)$:

$$\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{14.8}$$

We can simplify our notation by omitting mention of the index when we sum over N measurements:

$$\sum x_i = \sum_{i=1}^{N} x_i \tag{14.9}$$

The value of the **parent mean** or the **true mean** μ corresponding to $\bar{x}$ is defined by

$$\mu = \lim_{N\to\infty}\left(\frac{1}{N}\sum x_i\right) \tag{14.10}$$

Using (14.5) we can calculate the statistical error $\sigma_{\bar{x}}$ of the sample mean $\bar{x}$. Taking into account that

$$\frac{\partial x_i}{\partial x_j} = \delta_{ij} \tag{14.11}$$

the Kronecker delta symbol, which equals 1 if $i = j$ and equals 0 otherwise, and hence

$$\frac{\partial}{\partial x_j}\sum_i x_i = \sum_i \delta_{ij} = 1 \tag{14.12}$$

Then, using

$$\frac{\partial \bar{x}}{\partial x_j} = \frac{1}{N}, \tag{14.13}$$

we get

$$\sigma_{\bar{x}}^2 = \left(\frac{1}{N}\right)^2 \sigma_{x_1}^2 + \ldots + \left(\frac{1}{N}\right)^2 \sigma_{x_N}^2 = \left(\frac{1}{N}\right)^2 N\hat{\sigma}^2 = \frac{\hat{\sigma}^2}{N} \tag{14.14}$$

so

$$\delta\bar{x} \equiv \sigma_{\bar{x}} = \frac{\hat{\sigma}}{\sqrt{N}}. \tag{14.15}$$

As a numerical example, suppose $\hat{\sigma} = 0.1$ and $N = 100$. Then you can quote the sample mean as (ignoring units here)

$$\bar{x} = 9.84 \pm 0.01 \tag{14.16}$$

According to (14.15) you can decrease the statistical error of the sample mean by increasing the number of independent measurements, but if one increases the number of measurements by a factor of 4, the statistical error of the sample mean is only decreased by a factor of $\frac{1}{2}$ (all other things being equal).

When measured values are quoted with an error estimate, that error estimate is a "Gaussian standard deviation." If you say the length is (9.84 ± 0.01) cm, you mean that you have used a measuring instrument which gives answers that differ from the true value by within ± 0.01 cm 68% of the time, within ± 0.02 cm 95% of the time, and ± 0.03 cm 99.7% of the time. Errors on measurements and average results are generally well described by the Gaussian distribution, which is, of course, why it is also known as the "normal distribution." Thus a measurement reported as

$$\text{Measurement} = (\text{measured value } \pm \text{ standard uncertainty }) \text{ unit of measurement} \tag{14.17}$$

where the $\pm$standard uncertainty indicates approximately a 68% **confidence interval**.

The last digit retained in the estimate of the mean should be in the same decimal place as the first digit of the standard error. The resultant number of significant figures in the reported mean indicates the precision of the experiment.

14.3 Gaussian (Normal) Distribution

Quoting Bevington, Ch.2, somewhat loosely,

> The Gaussian probability density is defined as
>
> $$p_G = \frac{1}{\sigma\sqrt{2\,\pi}}\, exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{14.18}$$
>
> This is a continuous function describing the probability of obtaining the value x in a random observation from a parent distribution with parameters μ and σ, corresponding to the mean and standard deviation, respectively. Because the distribution is continuous, we must define an interval in which the value of the observation x must fall. ...the probability $dP_G(x;\mu,\sigma)$ that the value of a random observation will fall within an infinitesimal interval dx around x is given by
>
> $$dP_G(x;\mu,\sigma) = p_G(x;\mu,\sigma)\, dx \tag{14.19}$$

The probability density function is normalized such that there is a 100% probability that the value of a random observation will lie in the "interval" $-\infty < x < +\infty$:

$$\int_{-\infty}^{+\infty} p_G(x;\mu,\sigma)\,dx = 1 \tag{14.20}$$

"The curve has unit area." The peak of the curve is at $x = \mu$, and the width of the curve is determined by the value of σ such that for $x = \mu + \sigma$, the height of the curve is reduced to $e^{-1/2} = 0.606531$ of its value at the peak

$$p_G(\mu \pm \sigma;\mu,\sigma) = e^{-1/2} p_G(\mu;\mu,\sigma) \tag{14.21}$$

The Gaussian distribution curve has a characteristic bell shape and is symmetric about the mean μ.

We can characterize a distribution by its *full-width at half maximum* Γ, often referred to as the *half-width,*defined as the range of x between values at which the probability density is half its maximum value:

$$p_G(\mu \pm \frac{1}{2}\Gamma;\mu,\sigma) = \frac{1}{2} p_G(\mu;\mu,\sigma) \tag{14.22}$$

which implies the value (see below)

$$\Gamma = 2.3548\,\sigma \tag{14.23}$$

The Gaussian distribution moments can be summarized by calculating the expected value of x, making use of Maxima's **`integrate`** function, (note that we use a Maxima "expression" for the probability density, instead of a Maxima function),

$$< x >= \int_{-\infty}^{+\infty} x\, p_G(x;\mu,\sigma)\,dx = \mu \tag{14.24}$$

```
(%i1) rho : exp(-(x-mu)^2/2/sig^2)/sig/sqrt(2*%pi)$
(%i2) assume(sig>0,mu>0);
(%o2) [sig > 0,mu > 0]
(%i3) xbar : integrate(x*rho,x,-inf,inf);
(%o3) mu
```

and the "variance",

$$< (x- < x >)^2 >= \int_{-\infty}^{+\infty} (x-\mu)^2\, p_G(x;\mu,\sigma)\,dx = \sigma^2 \tag{14.25}$$

```
(%i4) variance : integrate((x-mu)^2*rho,x,-inf,inf);
(%o4) sig^2
```

We make a simple plot of the Gaussian probability density for the cases $\mu = 0$, and $\sigma = 1, 2$.

```
(%i5) rho1 : rho,mu=0,sig=1;
(%o5) %e^-(x^2/2)/(sqrt(2)*sqrt(%pi))
(%i6) rho2 : rho,mu=0,sig=2;
(%o6) %e^-(x^2/8)/(2^(3/2)*sqrt(%pi))
(%i7) plot2d([rho1,rho2],[x,-6,6],[legend,"sig=1","sig=2"],
            [style,[lines,2]], [ylabel,"rho"])$
```

which produces the plot

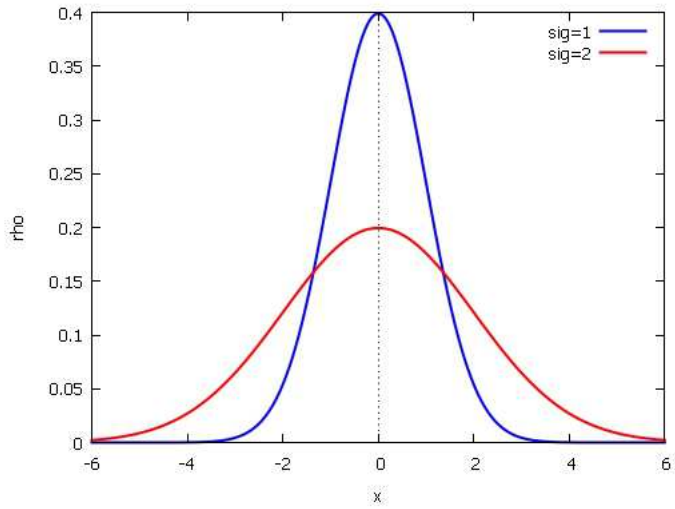

Figure 30: Gaussian Probability Density for $\mu = 0$, $\sigma = 1, 2$

Integral Probability

The probability that any random value of x will deviate from the mean by less than $\pm\Delta x$ is

$$P_G(\Delta x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\,\pi}} \int_{\mu-\Delta x}^{\mu+\Delta x} exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] = P_G(\Delta z) = \frac{1}{\sqrt{2\,\pi}} \int_{-\Delta z}^{\Delta z} e^{-\frac{z^2}{2}}\, dz \tag{14.26}$$

where dimensionless z is defined by

$$z = \frac{x-\mu}{\sigma}, \quad \Delta z = \frac{\Delta x}{\sigma} \tag{14.27}$$

Thus Δz measures the deviation from the mean in units of the standard deviation σ. Values of the "Normal Probability Integral" $P_G(\Delta z)$ can be found tabulated in various places. We can easily write a Maxima function **`norm_prob(delz)`** which returns values of $P_G(\Delta z)$ after a little interactive experimentation.

```
(%i8) np : integrate(exp(-z^2/2),z,-dz,dz)/sqrt(2*%pi);
(%o8) erf(dz/sqrt(2))
(%i9) np,dz = 1,numer;
(%o9) 0.6826894921370859
(%i10) np,dz = 2,numer;
(%o10) 0.9544997361036416
(%i11) np,dz = 3,numer;
(%o11) 0.9973002039367398
(%i212) norm_prob(delz) := float (erf(delz/sqrt(2)))$
(%i13) norm_prob(1);
(%o13) 0.6826894921370859
(%i14) norm_prob(2);
(%o14) 0.9544997361036416
```

These results mean that roughly 68% of random values of x drawn from a parent Gaussian distribution having mean μ and standard deviation σ will have values in the range $\mu \pm \sigma$, and roughly 95% will have values in the range $\mu \pm 2\sigma$, and roughly 99.7% will have values in the range $\mu \pm 3\sigma$.

The "probable error"

$$\text{P.E.} = \sigma_{pe} = 0.6745\,\sigma \approx \frac{2}{3}\,\sigma \tag{14.28}$$

defines an interval $\mu - \sigma_{pe} \le x \le \mu + \sigma_{pe}$ within which half (50%) of the values of x measured will lie.

```
(%i15) norm_prob (0.6745);
(%o15) 0.5000065142726016
(%i16) find_root (norm_prob(dz) - 0.5, dz, 0.6, 0.7);
(%o16) 0.6744897501960818
```

Working with the dimensionless variable z and the probability density function

$$p(z) = \frac{1}{\sqrt{2\,\pi}}\, e^{-z^2/2} \tag{14.29}$$

we can define the half-width Γ as $2\,z_0$, where

$$p(z_0) = p(0)/2 \tag{14.30}$$

which implies, using $\ln e^A = A$,

$$\Gamma = 2\,z_0 = 2\,\sqrt{2\,\ln 2} = 2.3548 \tag{14.31}$$

```
(%i17) 2*sqrt(2*log(2));
(%o17) 2^(3/2)*sqrt(log(2))
(%i18) float(%);
(%o18) 2.35482004503095
```

Quoting Lyons, p. 15

> One feature which helps to make the Gaussian distribution of such widespread relevance is the central limit theorem. One statement of this is that if x_i is a set of N independent variables of mean μ and variance σ^2, then for large N
>
> $$y = \frac{1}{N}\sum x_i \tag{14.32}$$
>
> tends to a Gaussian distribution of mean μ and variance σ^2/N. The distribution of the individual x_i is irrelevant. Furthermore, the x_i can even come from different distributions with different means μ_i and variances σ_i^2 in which case y tends to a Gaussian of mean $(1/N)\sum \mu_i$ and variance $\sum \sigma_i^2/N$. If the x_i are already Gaussian distributed, then the distribution of (14.32) is already Gaussian for all values of N from 1 upwards. But even if x_i is, say, uniformly distributed over a finite range, then the sum of a few x_i will already look Gaussian. ...Thus whatever the initial distributions, a linear combination of a few variables almost always degenerates into a Gaussian distribution

Sample Variance and Standard Deviation

The sample mean $\bar{x}$ describes all your data with just one number, but doesn't give any information about how spread out the data values are. We need a number to express the spread or dispersion of the data about the mean. The average squared deviation from the mean is a sensible measure of the spread of the data. It is called the **sample variance** or the **measured variance** $V(x)$

$$V(x) = \frac{1}{N}\sum (x_i - \bar{x})^2 = \frac{1}{N}\sum x_i^2 - \left(\frac{1}{N}\sum x_i\right)^2 = \overline{x^2} - \bar{x}^2 \tag{14.33}$$

Thus the sample variance is the mean square minus the squared mean.

The root mean squared deviation is called the **sample standard deviation** and given the symbol σ. It is just the square root of the sample variance and can be expressed in various forms

$$\sigma = \sqrt{V(x)} = \sqrt{\overline{x^2} - \bar{x}^2} = \sqrt{\frac{1}{N}\sum (x_i - \bar{x})^2} \tag{14.34}$$

Quoting Barlow (see References section at the end), Sec. 2.4.2 and 2.4.3,

σ represents a reasonable amount for a particular data point to differ from the mean. The exact numerical details depend on the case, but usually one is not surprised by data points one or two standard deviations from the mean, whereas a data point three or more σ away would cause a few raised eyebrows.

The definition of σ is a minefield of alternatives, and to call it the 'standard' deviation is something of a sick joke. It is important to face up to this, for when people are unaware of the differences between the definitions they get confused and dismayed by factors of $\sqrt{N/(N-1)}$ that appear apparently out of nowhere. This leads to a tendency to insert such factors at random and generally incorrect moments. (14.34) defined the standard deviation of a data sample as

$$\sigma = \sqrt{\frac{1}{N}\sum (x_i - \bar{x})^2} \tag{14.35}$$

So far so good. However, our data are presumably taken as a sample from a parent distribution, which has a mean and a standard deviation, denoted μ and σ. In terms of expectation values:

$$\mu = \langle x \rangle, \quad \sigma = \sqrt{\langle x^2 \rangle - \langle x \rangle^2} = \sqrt{\langle x^2 \rangle - \mu^2} \tag{14.36}$$

There is thus a clear distinction between $\bar{x}$, the mean of the sample, and μ, that of the parent, and complete confusion between σ, the standard deviation of the sample, and σ, that of the parent. This is not really too bad, as it is generally clear which is meant. However, it gets worse. Some authors define the term 'standard deviation' as the r.m.s. deviation of the data points from the 'true' mean μ, rather than the sample mean $\bar{x}$:

$$\sqrt{\frac{1}{N}\sum (x_i - \mu)^2} \tag{14.37}$$

This is felt to be a more fundamental and 'truer' quantity than that defined in (14.34), but it is not much use if you do not know the value of μ. However, an estimate of this, which (when squared) gives an unbiased estimate of σ^2 of the parent, is given by

$$s = \sqrt{\frac{1}{N-1}\sum (x_i - \bar{x})^2} \tag{14.38}$$

. . . it is not a matter of 'right' and 'wrong' definitions: you can use whichever definition of standard deviation you please, provided you make it clear to other people what that is, and when using other people's results and formulae involving s or σ you check what they mean by it. Some authors helpfully use the name "sample standard deviation" explicitly for the quantity defined in (14.38). Unfortunately others use it for the quantity defined in (14.35). Definitions of variance, and *sample variance*, are similarly confused. In this book we will consistently use σ as defined in (14.35) and s for the quantity defined by (14.38). This is not universal, and different authors use either symbol for either quantity – you have been warned. Some authors use Greek symbols for quantities from distributions and the Roman alphabet for those of data samples, but the usage of σ is so entrenched that this has no chance of universal adoption, and anyway this still leaves the ambiguity between (14.35), (14.37), and (14.38). If necessary, the distinction can be made completely clear and explicit by denoting the quantity defined by (14.35) as σ_N and that of (14.38) as σ_{N-1}, though this involves extra subscripts which lead to messy-looking formulae.

A low value of the standard deviation indicates a high precision – the data points are closely clustered, with low scatter. Hence, the smaller the standard error, the more precise are the set of measurements, and the more reproducible are the results.

Our preference for using s (14.38) rather than σ (14.35) can be illustrated by assuming we have only one measurement of x which we call x_1, and with $N = 1$, our rule for calculating $\bar{x}$ (14.7), which we repeat here:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{14.39}$$

says that $\bar{x} = x_1$. If we then use (14.35), we get $\sigma = \sqrt{(x_1 - \bar{x})^2} = 0$, which is an unacceptably low estimate of the standard deviation. If we instead use (14.38), we get $s = \sqrt{\frac{0}{0}}$ which is an indeterminate result, and forces us to use at least two measured values ($N = 2$) to get acceptable values for both the mean and the standard deviation. The denominator $N - 1$ is called the "number of degrees of freedom" $\nu = \text{dof}$. Of course, if N is large, there will be no practical difference between using σ or s as a measure of the standard deviation of the random values of x.

The **accuracy** of a measurement refers to how closely a measurement compares with a known "standard" or "accepted" or "theoretical" value. Sometimes, measurements with a high precision may cluster very closely around an inaccurate mean value, usually due to the presence of systematic errors.

If you have a large data set (large N), the data can be "binned" into small sample "classes" and the means of the individual bins ("classes") can be used to plot a histogram of the data set. With only random errors present, the plot of the histogram ("plot of the frequency distribution") will be a characteristic bell-shaped curve that is symmetric about the mean of the data set $\bar{x}$. If the "normal distribution" is instead asymmetric and the peak of the histogram plot does not coincide with the position of $\bar{x}$, but is shifted either right or left, one should investigate the possibility of systematic measurement errors (in addition to random errors).

Our estimate of the statistical standard error on the mean from (14.15) for a situation in which we knew that the uncertainty of each individual measurement was approximately $\sigma_i = \hat{\sigma}$ for all i was

$$\delta\bar{x} \equiv \sigma_{\bar{x}} = \frac{\hat{\sigma}}{\sqrt{N}}. \tag{14.40}$$

If we don't know the uncertainty of each repeated measurement, or want to check if our estimate was realistic, we can use our data and (14.38) to calculate s, the square root of the unbiased variance, and then use s instead of the unknown or suspect value of $\hat{\sigma}$ to calculate the standard statistical error of $\bar{x}$.

$$\delta\bar{x} \equiv \sigma_{\bar{x}} = \frac{s}{\sqrt{N}}. \tag{14.41}$$

14.4 The χ^2 Goodness-of-Fit Test

The χ^2 (chi-square) "goodness-of-fit" test involves using what Numerical Recipes calls the "incomplete gamma function" $Q(a, x)$ defined by

$$Q(a, x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t}\, t^{a-1}\, dt \tag{14.42}$$

where $\Gamma(z)$ is the gamma function

$$\Gamma(z) = \int_0^\infty t^{z-1}\, e^{-t}\, dt \tag{14.43}$$

When the argument z is an integer

$$\Gamma(n) = (n-1)! \tag{14.44}$$

so $\Gamma(4) = 3!$, $\Gamma(5) = 4!$, etc.

```
(%i1) gamma(1);
(%o1) 1
(%i2) 0!;
(%o2) 1
(%i3) gamma(4);
(%o3) 6
(%i4) gamma(5);
(%o4) 24
```

This version of the "incomplete gamma function" $Q(a,x)$ has the limiting values

$$Q(a,0) = 1 \quad \text{and} \quad Q(a,\infty) = 0 \tag{14.45}$$

To compute $Q(a,x)$ using Maxima, we use

```
Q(a,x)  <==>  gamma_incomplete(a,x) / gamma(a)
```

With the above notation, we can estimate the "goodness-of-fit" of the data to the model in terms of a number we call Q. The quantitiy Q, defined using the current data and fit value of $\chi^2(a,b)$ defined in (3.1) in terms of the function $Q(a,x)$, is (we justify this definition below):

$$Q = Q\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right) \tag{14.46}$$

and is the (fractional) "chi-square probability" that a repetition of the same experiment (same number of data points and same number of degrees of freedom (dof) $\nu = N - m$ and same model) would produce a value of χ^2 greater than the value found. (m is the number of model parameters fitted, which is 2 in our straight line model.) From the limiting values (14.45) we see that the chi-square probability that a ν degree of freedom fit results in a value $\chi^2 > 0$ is 100%.

If the "reduced chi-square" $\chi^2_\nu = \chi^2/\nu$ is reasonably close to 1, then Q is reasonably close to 0.5 (50% probability). Equivalently, we can say that if the value of χ^2 predicted by the data and fit is approximately equal to the number of degrees of freedom (dof $= \nu = N - 2$ for a straight line fit) , then Q is reasonably close to 0.5 (50% probability).

We can use our **`fit.mac`** function **`chi2_prob(chi2, dof)`** to illustrate this for the case $\nu = 8$ and $\chi^2 = 7.35$:

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) chi2_prob(7.35,8)$
 chi2/dof =  0.91875
 chi2_prob =  49.9383 %
```

If we use the underlying χ^2 distribution, which governs the value of the mean $\overline{\chi^2}$ and variance $\sigma^2(\chi^2)$, justified if the experimentally observed values y_i are Gaussian distributed with mean $a + b\,x_i$ and with variance σ_i^2, the sum (3.1) is distributed as predicted by the "χ^2 distribution", decribed by the probability distribution function (pdf)

$$p(\chi^2,\nu)\,d\chi^2 = \frac{1}{2^{\nu/2}\,\Gamma(\nu/2)}\,[\chi^2]^{\frac{\nu}{2}-1}\,e^{-\frac{\chi^2}{2}}\,d\chi^2 \tag{14.47}$$

for

$$0 \le \chi^2 < \infty \tag{14.48}$$

and one can show that

$$\overline{\chi^2} = \nu, \quad \sigma^2(\chi^2) = 2\,\nu \tag{14.49}$$

For example, with **`z`** standing for the integration variable χ^2 (these definitions are also in **`fit.mac`**),

```
(%i1) p(chi2,nu) := chi2^(nu/2 -1)*exp(-chi2/2)/2^(nu/2) / gamma(nu/2)$
(%i2) chi2_moment (m,nu) := integrate(z^m*p(z,nu),z,0,inf)$
(%i3) chi2_norm(nu) := chi2_moment (0,nu)$
(%i4) chi2_mean (nu) := chi2_moment (1,nu)$
(%i5) chi2_variance (nu) := (chi2_moment (2,nu) - chi2_mean (nu)^2)$
(%i6) chi2_norm(8);
(%o6) 1
(%i7) map ('chi2_norm,[8,9]);
(%o7) [1,1]
(%i8) map ('chi2_mean,[8,9]);
(%o8) [8,9]
(%i9) map ('chi2_variance,[8,9]);
(%o9) [16,18]
(%i10) qdraw (ex1(p(z,6),z,0.2,20,lc(blue),lk("6")),
            ex1(p(z,10),z,0.2,20,lc(red),lk("10")),
              more (xlabel = "chi2"))$
```

which produces the plot

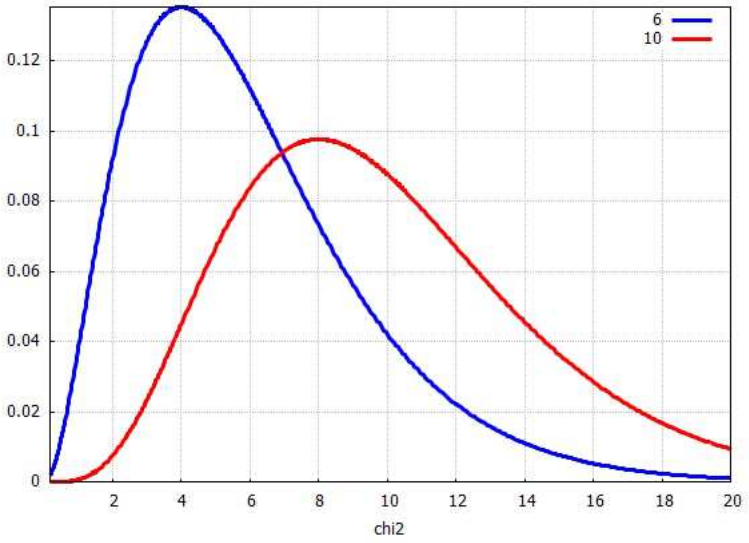

Figure 31: χ^2 prob. d.f. for two values of ν

This means that "large" values of χ^2 are unlikely, and very small values of χ^2 are also unlikely. Thus very large or very small values of χ^2 probably indicate that the data cannot be modelled well with a straight line fit (or else the experimental uncertainties in the data y_i have not been accurately estimated).

Returning to the χ^2 goodness-of-fit value Q, quoting Numerical Recipes (1992, Sec. 15.2)

> ...If Q is larger than, say, 0.1 (i.e., the chi-square probability is greater than 10%), then the goodness-of-fit is believable. If it is larger than, say, 0.001 (i.e., the chi-square probability is larger than 0.1%) , then the fit *may* be acceptable if the errors are nonnormal [non-gaussian] or have been moderately underestimated. If Q is less than 0.001 then the model and/or estimation procedure can rightly be called into question.

Now that we have introduced the definition of the χ^2 probability distribution function (pdf) **`p(chi2,dof)`**, we can use that function together with Maxima's **`integrate`** to calculate the probabilty of finding a new value of χ^2 (in a repetition of the experiment and fit) greater than some already "observed" value of $\chi^2(a,b)$. Repeating the example above (by doing the implied integral directly), we assume the observed $\chi^2 = 7.35$ and dof $= \nu = 8$.

```
(%i1) load(fit);
(%o1) "c:/work9/fit.mac"
(%i2) integrate(p(z,8),z,7.35,inf);
(%o2) 0.499383
(%i3) chi2_prob(7.35,8)$
 chi2/dof =  0.91875
 chi2_prob =  49.9383 %
```

By using **integrate** with symbolic values for the lower limit of the integral and for the number of degrees of freedom ν, we arrive at the same definition as we used above to define the goodness-of-fit number Q.

```
(%i4) integrate(p(z,nu),z,z0,inf);
Is nu positive, negative or zero?
p;
(%o4) gamma_incomplete(nu/2,z0/2)/gamma(nu/2)
```

15 General Linear Fit Matrix Solution Derivation

We assume the mathematical model in terms of M unknown parameters a_k is

$$y(x;\mathbf{a}) = f(x) + \sum_{k=1}^{M} a_k\, X_k(x), \tag{15.1}$$

in which the $X_k(x)$ are arbitrary given functions of the independent variable x.

Given a data set (x_i, y_i, σ_i) having N data points, with σ_i being the estimated uncertainty of each measured y_i, and given a mathematical model, χ^2 is defined as

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i;\mathbf{a})}{\sigma_i} \right)^2. \tag{15.2}$$

We now assume that the a_k values which satisfy the M equations

$$\frac{\partial \chi^2}{\partial a_k} = 0 \tag{15.3}$$

will yield a good fit to the data. The resulting solution should be checked visually with a simple plot of the data and the model together. We use

$$\frac{\partial}{\partial a_m} y(x_i;\mathbf{a}) = X_m(x_i) \tag{15.4}$$

to simplify this set of equations for the a_k.

In our matrix notation, the transpose of a matrix A is denoted A^{T} and we will use boldfaced lower case letters, such as $\mathbf{d}$, to denote matrix column vectors.

Let

$$e_i = \frac{(y_i - f(x_i))}{\sigma_i}, \tag{15.5}$$

and

$$A_{ik} = \frac{X_k(x_i)}{\sigma_i}. \tag{15.6}$$

Then the M equations (15.3) become

$$0 = \sum_{i=1}^{N} e_i \, A_{ik} - \sum_{i=1}^{N} \sum_{m=1}^{M} a_m \, A_{im} \, A_{ik}. \tag{15.7}$$

In the first term, we define

$$d_k = \sum_{i=1}^{N} e_i \, A_{ik} = \sum_{i=1}^{N} \left(A^{\mathrm{T}}\right)_{ki} \, e_i, \tag{15.8}$$

or, using matrix notation

$$\mathbf{d} = A^{\mathrm{T}} \, \mathbf{e}. \tag{15.9}$$

In the second term, we interchange the order of summation and get

$$-\sum_{m=1}^{M} \left(\sum_{i=1}^{N} \left(A^{\mathrm{T}}\right)_{ki} \, A_{im} \right) a_m = -\sum_{m=1}^{M} B_{km} \, a_m = -\left(B \, \mathbf{a}\right)_k, \tag{15.10}$$

thus defining

$$B = A^{\mathrm{T}} A. \tag{15.11}$$

Thus the M equations (15.3) reduce to the single matrix equation

$$B \, \mathbf{a} = \mathbf{d}, \tag{15.12}$$

from which we get

$$\mathbf{a} = B^{-1} \, \mathbf{d} = \left(A^{\mathrm{T}} \, A\right)^{-1} \, \left(A^{\mathrm{T}} \, \mathbf{e}\right). \tag{15.13}$$

Errors in a_k arise only from errors in the measured data values y_j, since we are assuming the corresponding x_j are known exactly or at least with much less error. To calculate the estimated error $\sigma(a_k)$ in the parameter a_k, we add contributions in quadrature, as usual,

$$\sigma^2(a_k) = \sum_{j=1}^{N} \left(\frac{\partial a_k}{\partial y_j} \right)^2 \sigma_j^2, \tag{15.14}$$

and it is convenient to define

$$C = B^{-1}, \tag{15.15}$$

so that

$$\mathbf{a} = C \, \mathbf{d}. \tag{15.16}$$

Now B and hence C do not depend on the data values y_i, and can be treated as constants in finding the variation of a_k due to the estimated errors of the y_i. Thus

$$\frac{\partial a_k}{\partial y_j} = \sum_{m=1}^{M} C_{km} \frac{\partial d_m}{\partial y_j}, \tag{15.17}$$

and

$$\frac{\partial d_m}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_{i=1}^{N} A_{im} \, e_i = \sum_{i=1}^{N} A_{im} \, \frac{\partial}{\partial y_j} e_i = \sum_{i=1}^{N} A_{im} \frac{\delta_{ij}}{\sigma_i} = \frac{A_{jm}}{\sigma_j}. \tag{15.18}$$

We have used the Kronecker delta symbol δ_{ij} which is equal to unity if i equals j, and is otherwise equal to zero. For example, since y_i and y_j are independent numbers for $i \neq j$,

$$\frac{\partial y_i}{\partial y_j} = \delta_{ij}. \tag{15.19}$$

We then have

$$\frac{\partial a_k}{\partial y_j} = \frac{1}{\sigma_j^2} \sum_{m=1}^{M} C_{km}\, X_m(x_j) \tag{15.20}$$

for $j = 1, \ldots, N$ and $k = 1, \ldots, M$. Using (15.20) in (15.14), interchanging the order of summations, and using

$$\sum_{l=1}^{M} C_{kl} B_{lm} = (CB)_{km} = \left(B^{-1}B\right)_{km} = \delta_{km}, \tag{15.21}$$

we get

$$\begin{aligned}
\sigma^2(a_k) &= \sum_{j=1}^{N} \sigma_j^2 \left(\sum_{l=1}^{M} \frac{C_{kl} X_l(x_j)}{\sigma_j^2} \sum_{m=1}^{M} \frac{C_{km} X_m(x_j)}{\sigma_j^2} \right) \\
&= \sum_{l=1}^{M} \sum_{m=1}^{M} C_{kl} C_{km} \sum_{j=1}^{N} \frac{X_l(x_j) X_m(x_j)}{\sigma_j^2} \\
&= \sum_{l=1}^{M} \sum_{m=1}^{M} C_{kl} C_{km} \sum_{j=1}^{N} A_{jl} A_{jm} \\
&= \sum_{l=1}^{M} \sum_{m=1}^{M} C_{kl} C_{km} B_{lm} \\
&= \sum_{m=1}^{M} C_{km} \delta_{km} \\
&= C_{kk}.
\end{aligned}$$

Hence the uncertainty in a_k is given by

$$\sigma(a_k) = \sqrt{C_{kk}}. \tag{15.22}$$

With some changes in notation, the above matrix method is the basis of the code for the functions which are called by the general linear fit method **`lfit`** in **`fit.mac`**.

16 General Nonlinear Fit Search Method

The method used for **`nlfit`** is called the Levenberg-Marquardt method (or just the Marquardt method). This method is summarized by Bevington (3rd ed), pdf 162, and also by Numerical Recipes (Fortran 77, 2nd ed., 1992, p. 678).

A dimensionless parameter (fudge factor) λ is used to combine the advantages of two different search methods, adjusting the value of λ in response to whether the value of χ^2 increases or decreases as one change the values of the adjustable model parameters.

Quoting Numerical Recipes: “This . . . method . . . works very well in practice, and has become the standard of nonlinear least-squares routines.”

17 References

1. Data Reduction and Error Analysis for the Physical Sciences, 3rd Ed, 2003,
by Philip R. Bevington and D. Keith Robinson, McGraw Hill;
New copies available via Amazon.com, listed as "New International
Economy Edition," printed in India by McGraw-Hill, $13.84.

Also available as a pdf on the web at:
http://advancedlab.berkeley.edu/mediawiki/images/c/c3/Bevington.pdf

or click on link in blue under:
"References

1. P. Bevington, ["Data Reduction and Error Analysis for the Physical Sciences", McGraw-Hill].
An old standard that is pretty dry but straightforward. "

located on the webpage:
http://advancedlab.berkeley.edu/mediawiki/index.php/Error_Analysis_Exercise

which is part of the Physics 111 web page:
http://advancedlab.berkeley.edu/mediawiki/index.php/Main_Page

Also the same pdf of Bevington and Robinson can be found on the
Cornell Univ. Astronomy 3310, Planetary Image Processing, web page
http://astro.cornell.edu/academics/courses/astro3310/

---

2. Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences,
Roger Barlow, 1989, John Wiley, (paperback, 1993).

---

3. Numerical Recipes in Fortran 77, Second Edition (1992),
by Press, Teukolsky, Vetterling, and Flannery,
Cambridge Univ. Press, now "obsolete", but available as a used book
(eg. Amazon $9) or via an online version at:
http://apps.nrbook.com/fortran/index.html

See, especially, Ch. 15: Modelling of Data

Numerical Reciples Home page: http://numerical.recipes/

---

4. Nuclear Radiation Physics by Ralph E. Lapp and Howard L. Andrews,
2nd. edition, 1954, Prentice-Hall

---

5. Statistics for Nuclear and Particle Physicists, Louis Lyons, 1986, Cambridge Univ. Press
(paperback edition of same in 1989)

---

6. Statistical Methods in Experimental Physics, 2nd. ed., Frederick James,
2006, World Scientific

---

7. Univ. N. Carolina Measurement manual
http://www.webassign.net/question_assets/unccolphysmechl1/measurements/manual.pdf