The Histogram

Histograms were invented for human beings, so that we can visualize the observations as a density (estimated values of a pdf).

The binning most convenient for visualization is not necessarily the best for fitting.

We have already looked in detail at the problem of choosing the optimal binning for estimation and testing using Pearson's Chi-square statistic.

For other methods of fitting, it is better to have very narrow bins with very few events per bin, since putting events into bins causes us to lose the information about exactly where the event was observed.

For maximum likelihood fitting, there is no binning at all.

How to fit a histogram

Fitting always means minimizing something.
The "something" to be minimized is called the objective function
(that is the language of function minimization).

The function may however take many forms. The most important are:

Pearson's Chi-square : The common least squares sum, with sigma taken as the square root of the expected number of events.

Neyman's Chi-square : The same least squares sum, except that sigma is the square root of the observed number of events.

Binned Likelihood : $-2 \ln \lambda$, where λ is the binned likelihood ratio of Baker and Cousins.

Likelihood : $-2 \ln L$, where L is the usual (unbinned) likelihood of the observed individual events. The histogram has disappeared.

Pearson's Chi-square to fit a histogram

The Pearson test statistic T is the residual sum of weighted squares of differences between the observed number of events and the expected number:

$$T(\theta) = \sum_{bins} \frac{(n_i - Np_i(\theta))^2}{Np_i(\theta)},$$

where Np_i, the expected number of events in bin i, is expected on the basis of the model function, the p.d.f. $f(x|\theta)$

$$p_i(\theta) = \int_{bin\ i} f(x|\theta)dx$$

We nearly always approximate the integral over the bin by the value of f in the middle of the bin $f(x_i)$, times the bin width:

$$p_i(\theta) = \int_{bin\ i} f(x|\theta)dx \approx f(x_i|\theta)\Delta x$$

Pearson's Chi-square: the integral over the bin

The approximation

$$p_i(\theta) \approx f(x_i|\theta)\Delta x$$

is in fact a very good approximation, even if f is rising or falling steeply, since the leading term in the error is proportional only to the second derivative of f.

All other methods with binning also use this approximation.
You must be careful only when the bins are so wide that there is significant curvature of f inside the bin.

However, you should be sure that x_i is in the middle of the i^{th} bin.

If the bins are all of the same width, then the bin width Δx is just an overall normalization factor which can usually be absorbed into one of the fit parameters θ, in which case it can simply be dropped.

Pearson's Chi-square: dependence on the parameters

Pearson proposed his T-statistic for GOF testing, with no parameters θ. For fitting points y to a curve $f(x)$, it is generally written:

$$T(\theta) = \sum_{points} \frac{(y_i - f(x_i, \theta))^2}{\sigma_i^2},$$

and it is assumed that the variances σ_i^2 are known and do not depend on the parameters θ. If f is a linear function of the θ, the problem is linear and can be solved by matrix inversion without iteration.

However, when we fit histograms by minimizing

$$T(\theta) = \sum_{bins} \frac{(n_i - Np_i(\theta))^2}{Np_i(\theta)},$$

there is an additional dependence on θ in the denominator which makes a linear problem non-linear and has the additional effect of making the minimum of T occur for larger values of $\sigma_i^2 = Np_i(\theta)$.
That means that it biases the fit toward larger values of f.

Pearson's Chi-square: dependence on the parameters

The bias can be avoided by removing the dependence on θ in the denominator during the fit. One way to do that is:

1. Fit the histogram using T defined as above, with the unwanted dependence on θ.
2. Remember the values of Np_i for each bin at the minimum of T.
3. Perform the fit again, this time replacing in the objective function $Np_i(\theta)$ in the denominator for each bin, by the value obtained in step 2 above.
4. If necessary, repeat steps 2 and 3 until the fit is stable.

In this way, the denominator is Np_i, the expected number of events based on $f(x)$, but its derivative with respect to θ is zero during the (last) fit.

Neyman's Chi-square

Another way to remove the dependence of the denominator on θ, is to use not the expected number of events, but the observed number, which is of course constant.

This variant is called Neyman's Chi-square:

$$T_N(\theta) = \sum_{bins} \frac{(n_i - N p_i(\theta))^2}{n_i},$$

This is probably the most common form for fitting histograms, because it is the easiest to program, and it can be shown to be asymptotically equivalent to Pearson's Chi-square.

But it is obviously a poor approximation when n_i is small, since it then gives an exceptionally high weight to that bin.

Neyman's Chi-square: the bias for small n_i

There are two problems that arise when n_i is small:

1. When n_i is small, it is often because of a negative fluctuation from a larger expected value, in which case this large fluctuation will be given a larger weight than it should have, and cause a large bias toward smaller values of f.

   This may usually be fixed by taking, instead of $\sigma_i = n_i$,
   $\sigma_i =$ the average of n_i over three bins:

$$T_N(\theta) = \sum_{bins} \frac{(n_i - Np_i(\theta))^2}{(n_{i-1} + n_i + n_{i+1})/3} \,,$$

2. When $n_i = 0$, the value of T diverges, so $n_i = 0$ must be avoided absolutely. So, if after having used the trick above, there are still some bins with $\sigma_i = 0$ or 1, join those bins with neighbouring bins until σ_i exceeds one.

Binned Likelihood

The best way to avoid all the problems that arise because of binning with Chi-square, is to use either binned likelihood, which is insensitive to binning, or likelihood, which requires no binning at all.

Binned Likelihood treats the bin contents as either Poisson-distributed or Multinomial-distributed, whereas the Chi-square statistic implies a Gaussian approximation.

We use here the Poisson form, the most commonly used.

This has been introduced in this course in chapters 2 and 5, but it is treated only briefly in the book. (Statistics books often do not treat this at all, since it is just a special case of maximum likelihood, but it is important for those who work a lot with histograms.)

Binned Likelihood

The statistic to be minimized is what Baker and Cousins call the likelihood chi-square because it is a likelihood ratio but is distributed under H_0 as a $\chi^2(nbins-1)$:

$$\chi^2_\lambda(\theta) = -2 \ln \lambda(\theta) = 2 \sum_i [\mu_i(\theta) - n_i + n_i \ln(n_i/\mu_i(\theta))]$$

where the sum is over bins and $\mu_i(\theta) = N p_i(\theta)$.

This solves the problems of binning encountered when using Chi-square, since there is no minimum number of events per bin.
(The last term in the expression for χ^2_λ is taken to be zero when $n_i = 0$).

However, not much seems to be known about optimal binning for GOF, but clearly some bins (most bins?) must have more than one event.

This method has the area-conserving property: at the minimum, the normalization is automatically correct: $\sum_i \mu_i(\hat{\theta}) = \sum_i n_i$.

Unbinned Likelihood

We have seen in point estimation that under very general conditions, the optimal estimator is the Maximum Likelihood estimator, and that putting events into bins, as required for Chi-square estimation, always loses some information.

So why doesn't everyone always use M.L. fitting? Basically, because:

1. The value of the likelihood is not a good GOF test statistic (as is the Chi-square).
2. Normalization of the pdf is often a problem. The likelihood is a product over observations of the pdf of the model (the fitting function). By definition, the pdf should always be normalized, but in practice it is not always so easy.

   In fact, absolute normalization is not necessary, since the value of the likelihood function is only defined up to an arbitrary multiplicative constant, but the integral of the pdf over all the data space must be independent of the parameters θ.

Calculating the Likelihood

Since the likelihood is only defined up to an arbitrary multiplicative constant (one can think of this as being the units of L), the values of L may easily overflow or underflow the capacity of the computer arithmetic. Since we are going to use $-2 \ln L$ anyway, we therefore accumulate $\ln L$ by summing the logarithms of the pdf. Then the multiplicative constant becomes an arbitrary additive constant which is easier to handle arithmetically.

A more serious problem is the normalization of the pdf, which has to remain constant as the parameters are varied. Sometimes it is possible to work with correctly normalized functions, but more often we must numerically integrate and re-normalize the pdf at each function call.

If the observable space is one-dimensional, the integration can be done to high accuracy using numerical quadrature, so it is not a real problem, just a lot of extra work to make sure it works.

Calculating the Likelihood

But when fitting in higher dimensions, the numerical integration is more complicated.

Already in two or three dimensions, you have to know what you are doing, and convergence to high accuracy can be rather slow.

In higher dimensions, the integration may have to be done by Monte Carlo, which converges slowly and is usually not accurate enough for the fitting program to calculate numerical derivatives of $-2 \ln L$ with respect to the parameters. This in turn means that you may have considerable difficulty finding an accurate M.L. solution.

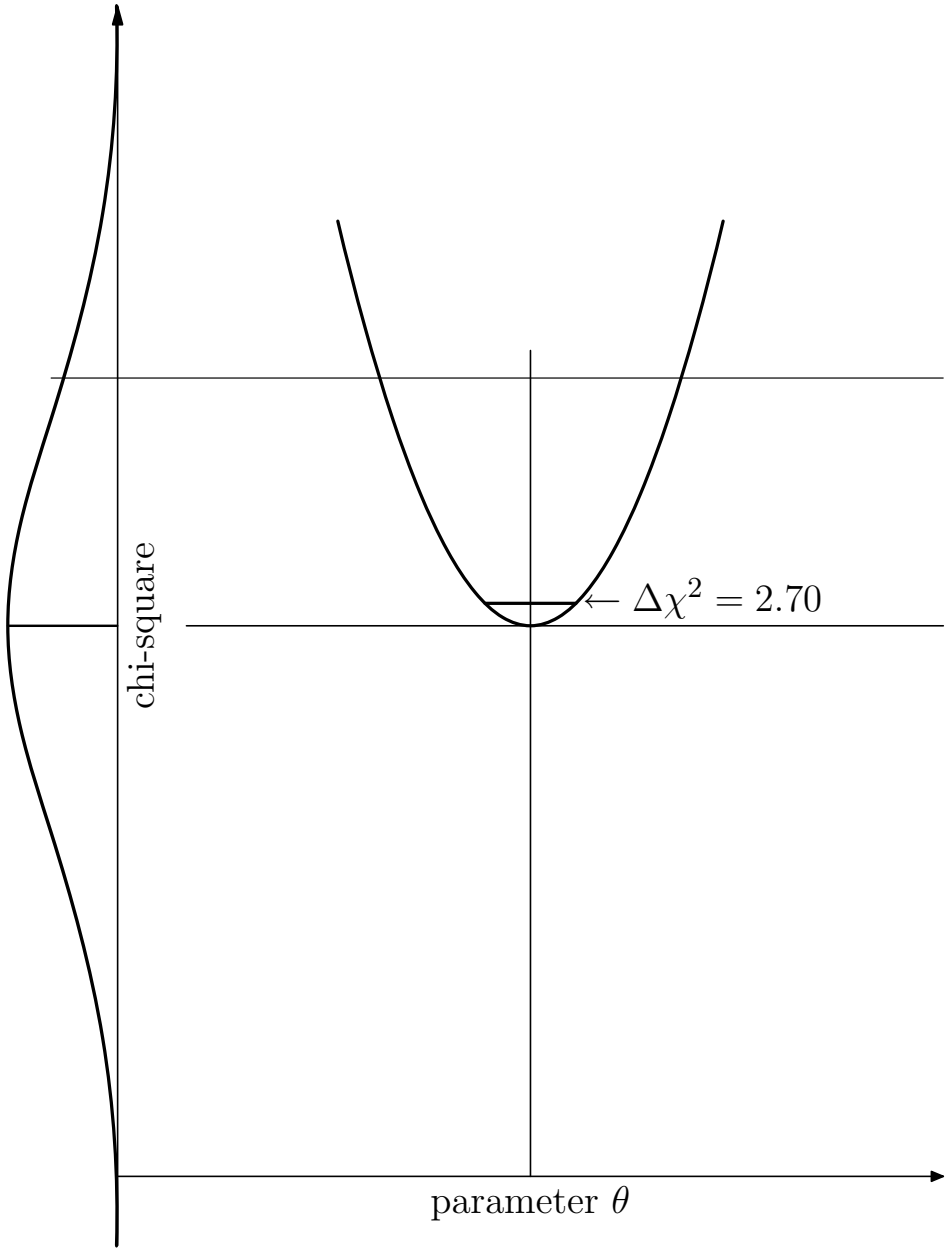
Confidence Intervals and Goodness-of-fit

When we introduced the "Five Classes of Problems", we said that some problems could be approached using the methods of different classes, giving different results.

Here is a good example of this effect, looking at a confidence interval on a fitted parameter as determined by:

1. The method of interval estimation: The ensemble of parameter values that includes the true value with probability 90%.
2. The method of goodness-of-fit: The ensemble of parameter values that gives a good fit to the data, with a P-value greater than 0.10.

In this example, we we use the chi-square method to fit a histogram with 51 bins and one unknown parameter θ.



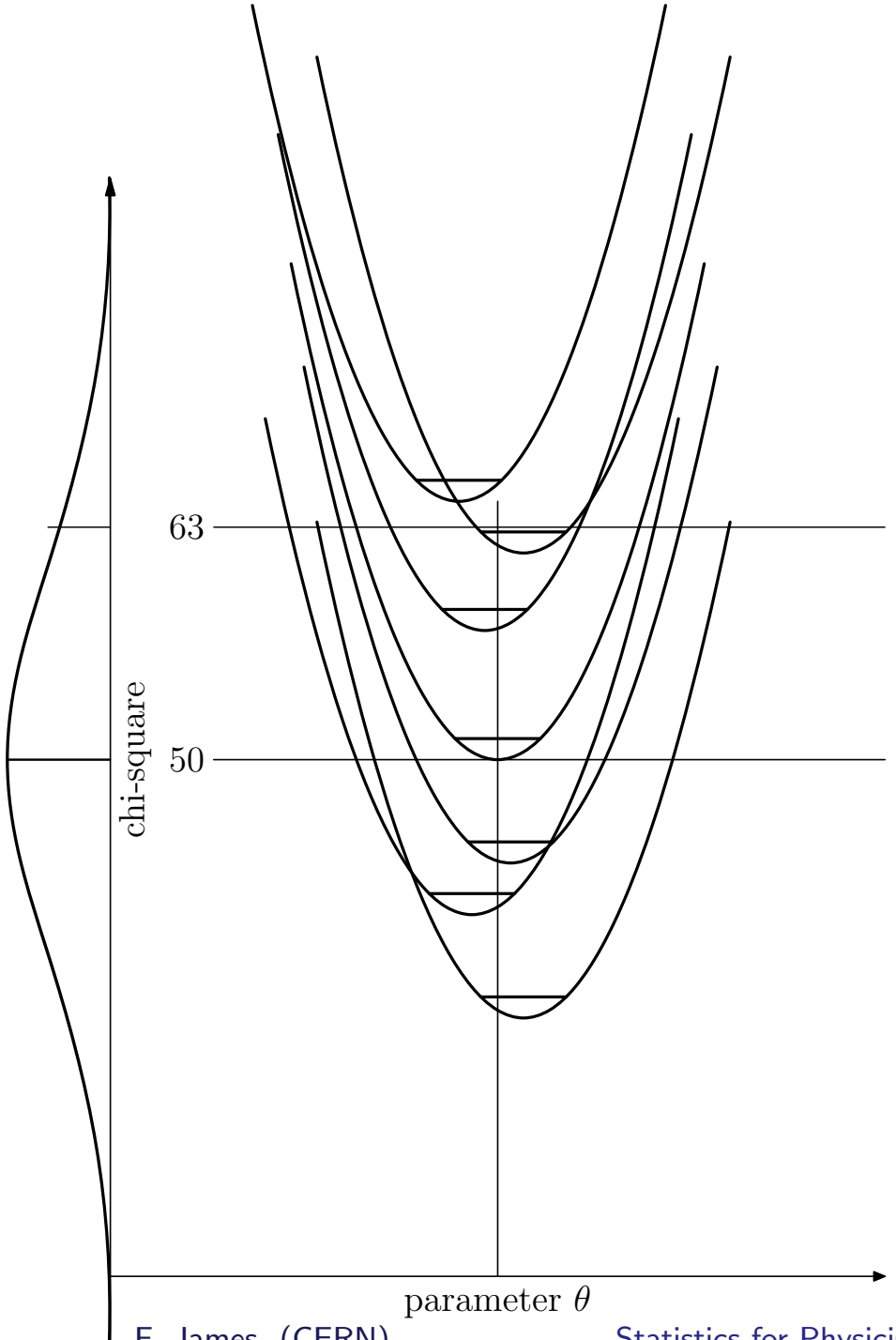
Chi-square as a function of the parameter θ.

At the 90% level, a good fit is defined by $\chi^2 < 63$ because there are 51 bins and one parameter θ.

But the 90% confidence interval for θ is given by $\chi^2 = \chi^2_{min} + 2.70$

How can we understand this?

Repeat the experiment!



Each time we repeat the experiment, we will get a parabola as shown here.

The confidence interval will always be (almost) the same width,
but the GOF intervals will vary wildly, sometimes even be empty.

However both methods will have 90 % coverage.

What is the Error Matrix?

We normally avoid using the word "error" because it is vague.
We prefer more meaningful names for the uncertainty
such as variance of the estimate, or confidence interval.

The error matrix is however a well-defined entity because it is just
the physicist's name for the variance matrix of the estimates
in a system where n parameters are being estimated.
The error matrix, a symmetric $n \times n$ matrix,
summarizes the parameter uncertainties with $n(n+1)/2$ numbers.

We recall the three levels of frequentist interval (region) estimation.

1. Intervals with exact coverage, using the Neyman construction.
2. Approximate intervals using likelihood contours (MINOS).
3. Normal Theory intervals for Gaussian-distributed estimates.

Different names for the error matrix and its inverse

according to	$\underset{\sim}{V}$	$\underset{\sim}{V}^{-1}$
statisticians	variance matrix	Fisher information matrix
physicists	error matrix	second derivative matrix
numerical analysts	inverse Hessian	Hessian
general relativists	contravariant metric tensor	covariant metric tensor

Statistical definition of the error matrix

Recall from point estimation that the variance of the estimates cannot be smaller than V_{min}, given by the Cramér-Rao lower bound.

$$V(\hat{\theta}) \underset{N\to\infty}{\longrightarrow} \left\{ E\left[\left(\frac{\partial \ln L}{\partial \theta} \right)^2 \right] \right\}^{-1} .$$

which is the inverse of the Fisher information. For several parameters, this can be written:

$$\left[\mathcal{I}_X(\boldsymbol{\theta}) \right]_{ij} = -E\left[\frac{\partial^2}{\partial\theta_i \partial\theta_j} \ln L(X|\boldsymbol{\theta}) \right] .$$

This limit is achieved when there are sufficient statistics, which as we have seen is the case when the model is of the exponential class (for example, Gaussian). In practice, we assume that this is the case whenever the likelihood (or chi-square) contours are ellipses, indicating a parabolic log-likelihood. Since the second derivative is constant, we don't have to calculate its expectation, we can just evaluate it anywhere.

Steepest Descent Method for minimizing F

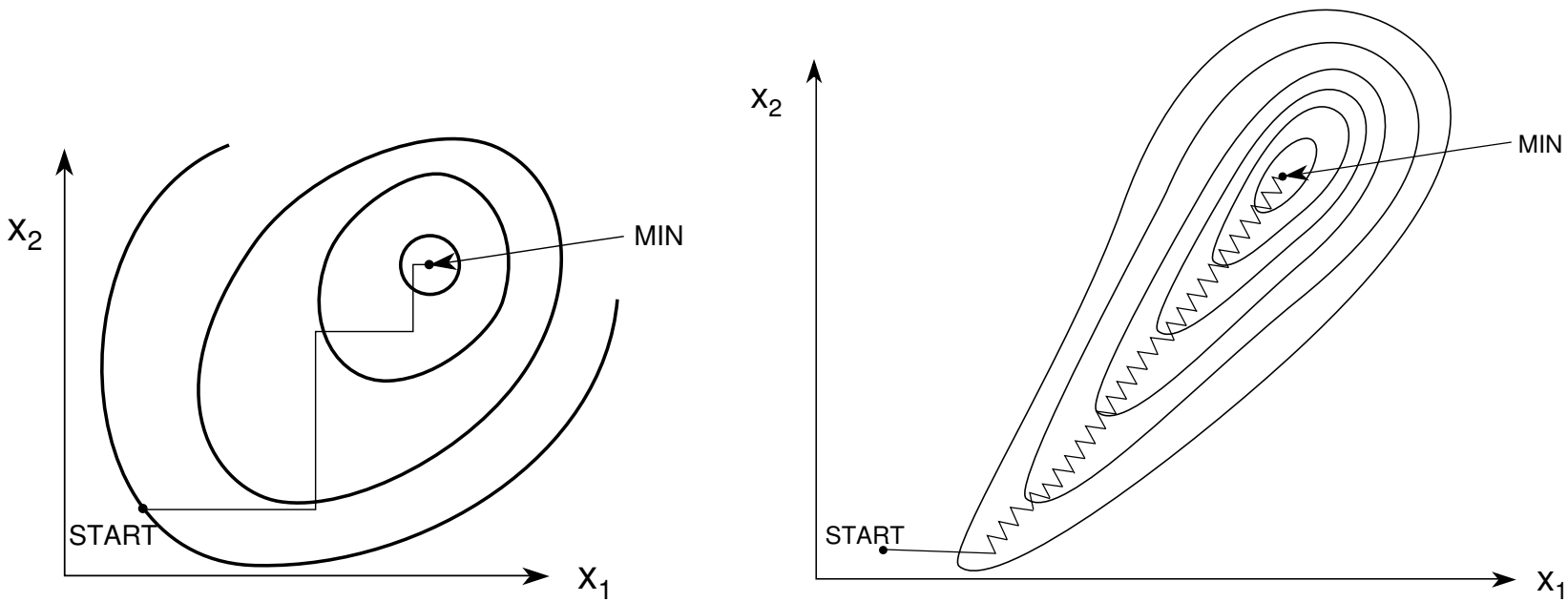
Since the Minuit minimization method MIGRAD produces an error matrix as a by-product of the minimization, we take some time here to discuss the general problem of minimization.

The obvious way to find the minimum of a general n-dimensional function $F(\boldsymbol{\theta})$, starting from some point $\boldsymbol{\theta}_{(0)}$, is to take a step in the direction of steepest descent, that is along the vector $-\mathbf{g} = -\partial F/\partial \boldsymbol{\theta}$.

This method (steepest descent) specifies only the direction, not the length of the step, so it requires then a line search, a one-dimensional minimization along the vector $-\mathbf{g}$. Thus a many-dimensional minimization is reduced to a one-dimensional one.

Steepest Descent Method for minimizing F

However, it is well known that, although this method converges under rather general conditions, the convergence can be arbitrarily slow, even for an exactly quadratic function, if the parameters are strongly correlated.



Newton's Method was the first to overcome this problem.

Newton's Method for minimizing F

At step i, where the parameter values are $\boldsymbol{\theta}_{(i)}$, Newton's Method:

1. evaluates the function $F(\boldsymbol{\theta}_{(i)})$
2. evaluates the first derivatives $\mathbf{g}_{(i)} = \partial F/\partial \boldsymbol{\theta}$
3. evaluates the second derivatives of F and inverts the second derivative matrix to obtain $\underset{\sim}{V} = \left[\frac{\partial^2(-2\ln\ L(X|\boldsymbol{\theta}))}{\partial\theta_i \partial\theta_j}\right]^{-1}$
4. takes the step $\boldsymbol{\delta}_{(i)} = \boldsymbol{\theta}_{(i+1)} - \boldsymbol{\theta}_{(i)} = -\mathbf{g}\underset{\sim}{V}$

If the problem is linear (F is parabolic), one step is enough, otherwise it is necessary to iterate to attain the minimum.

Problems with Newton's Method:

- requires evaluating a matrix of second derivatives at each iteration.
- requires inverting the matrix at each iteration.
- if $\underset{\sim}{V}$ is not positive-definite, Newton's step is not in general in a descending direction (it can be unstable and diverge).
- even if it does not diverge, it may oscillate.

How does Minuit calculate the error matrix?

In Minuit, the minimization is usually done with Migrad, which uses a variation of Newton's method called variable metric minimization.

In this method, the metric is $\underset{\sim}{V} = \left[\frac{\partial^2(-2 \ln\ L(X|\boldsymbol{\theta}))}{\partial\theta_i \partial\theta_j}\right]^{-1}$

At step i, where the parameter values are $\boldsymbol{\theta}_{(i)}$, the method:

1. evaluates the function $F(\boldsymbol{\theta}_{(i)})$
2. evaluates the first derivatives $\mathbf{g}_{(i)} = \partial F/\partial\boldsymbol{\theta}$

Using the current approximation to $\underset{\sim}{V}$, ($\underset{\sim}{V}_{(0)}$=$\underset{\sim}{I}$) calculate:

1. The next step: $\boldsymbol{\delta}_{(i)} = \boldsymbol{\theta}_{(i+1)} - \boldsymbol{\theta}_{(i)} = -\mathbf{g}\underset{\sim}{V}$ (Newton's step)
2. The EDM $= \mathbf{g}^T\ \underset{\sim}{V}\ \mathbf{g}/2$
3. The updated error matrix $\underset{\sim}{V}_{(i)}$=$\underset{\sim}{V}_{(i-1)} + dV$

where the updating formula dV assures that $\underset{\sim}{V}$ converges to the inverse of the second derivative matrix for a quadratic function F.

The Migrad Error Matrix

If EDM is less than the minimization tolerance, the process is finished. Otherwise, it continues to iterate: $\boldsymbol{\theta}$ converges toward the values for which F is a minimum, and $\underset{\sim}{V}$ converges toward the exact error matrix.

The minimization tolerance is by default $= 10^{-3} \times$ UP, where UP is the user-specified log-likelihood ratio that defines the parameter errors. For a one-parameter 68% confidence interval, UP $= 1.0$ (its default value).

It can happen that Migrad converges to the minimum of F before it converges to an accurate estimate of $\underset{\sim}{V}$. In that case, it automatically calculates $\underset{\sim}{V}$ by calculating the second derivative matrix and inverting it. The user can always request such a calculation by executing the command HESSE. This also allows the user to compare the $\underset{\sim}{V}$ estimated by Migrad with the $\underset{\sim}{V}$ calculated by matrix inversion.

The Error Matrix as a Metric Tensor

The idea of $\underset{\sim}{V}$ as a metric is appealing to a physicist, because it allows us to establish an analogy with General Relativity, where everything is reduced to geometry through the covariant metric tensor $g_{\mu\nu}$ which defines the element of distance in the (curved) space:

$$ds^2 = g_{\mu\nu} dx^\mu dx^\nu = \sum_{\mu,\nu} g_{\mu\nu} dx^\mu dx^\nu \qquad \begin{pmatrix}\text{summation}\\ \text{convention}\end{pmatrix}$$

Identifying $\underset{\sim}{V}$ with the contravariant metric tensor (because it transforms like coordinates), and $\underset{\sim}{V}^{-1}$ with the covariant metric tensor, we can form two invariants (Recall that $\mathbf{g}_{(i)} = \partial F/\partial \boldsymbol{\theta}$ and $\boldsymbol{\delta}_{(i)} = \boldsymbol{\theta}_{(i+1)} - \boldsymbol{\theta}_{(i)}$):

$$s_1 = \mathbf{g}^T \, \underset{\sim}{V} \, \mathbf{g} \qquad \text{and} \qquad s_2 = \boldsymbol{\delta}^T \, \underset{\sim}{V}^{-1} \, \boldsymbol{\delta}$$

The vertical distance s_1 is just what we called EDM, and the horizontal distance s_2 is the distance from $\boldsymbol{\theta}$ to $\boldsymbol{\theta}+\boldsymbol{\delta}$ measured in standard deviations.

The Extraordinary Accuracy of EDM

If the problem is linear ($f(X \mid \theta)$ is a linear function of θ)
then $\ln L$ is a multidimensional parabola
and EDM is the exact vertical distance to the minimum.

But we can always make a (non-linear) transformation of the parameters θ
that will make $\ln L$ parabolic, and since EDM is an invariant,
it should be correct even if $\ln L$ is not parabolic,
and we don't even have to find the transformation.

This reasoning is not mathematically rigorous for the statistical problem,
but it worked very well when we used it for
likelihood-based confidence intervals.

In practice it works even better for EDM !

The Error Matrix during a Migrad minimization

When Migrad has converged to the minimum of F, its "working value" of $\underset{\sim}{V}$ should converge to an accurate value of the Error Matrix, the inverse of the second derivative matrix, and it should be positive-definite, or at least non-negative, at the minimum.

If the problem is underdetermined, then $\underset{\sim}{V}$ is singular (not of full rank) and cannot be used as an Error Matrix.

During the minimization, the metric is variable, reflecting the curvature of the "space" F. The "working value" of $\underset{\sim}{V}$ used by Migrad for minimization must remain positive-definite in order to assure that each Newton step is a descent step, even if the true $\underset{\sim}{V}$ is not positive in some regions.

Migrad usually starts with $\underset{\sim}{V}=\underset{\sim}{I}$, the unit matrix, which is of course positive. Therefore the first step is a steepest descent step and $\underset{\sim}{V}$ is positive-definite at the start.

Keeping $\underset{\sim}{V}$ positive during a Migrad minimization

The matrix updating formula used by Migrad is supposed to assure that the working value of $\underset{\sim}{V}$ remains positive after each step, but because of numerical roundoff and other arithmetic mysteries, Migrad's value of $\underset{\sim}{V}$ can sometimes go negative during minimization.

If this happens, Migrad must force $\underset{\sim}{V}$ positive by adding some positive quantity κ to each diagonal element, so we will have $\underset{\sim}{V} \rightarrow \underset{\sim}{V} + \kappa \underset{\sim}{I}$

The quantity κ is determined by the conditions:

- If κ is too big, the next step becomes nearly a steepest descent step, and $\underset{\sim}{V}$ becomes inaccurate.
- If κ is too small, $\underset{\sim}{V}$ will remain negative.

The optimal κ is just a little bigger than $|\lambda_{\mathsf{neg}}|$, where λ_{neg} is the largest negative eigenvalue of $\underset{\sim}{V}$. Minuit therefore calculates the most negative eigenvalue of $\underset{\sim}{V}$ and adds $\kappa = 1.1|\lambda_{\mathsf{neg}}|$ along the diagonal.

How to use the Error Matrix

If the contours of F (either Chi-square or -2 log Likelihood) are ellipses, the parameter uncertainties can be summarized by the Error Matrix from which we can calculate uncertainties on derived quantities, using a procedure with the unfortunate name propagation of errors.

Given that some parameters $\boldsymbol{\theta}$ have been estimated with a covariance (error) matrix $\underset{\sim}{V}$, we can calculate the resulting uncertainty ΔZ in some quantity $Z(\boldsymbol{\theta})$ using linear error propagation:

$$(\Delta Z)^2 = \sum_i \sum_j \frac{\partial Z}{\partial \theta_i} \, \underset{\sim}{V}_{ij} \, \frac{\partial Z}{\partial \theta_j} \, .$$

The above formula is accurate in the limit that the uncertainties σ_i are small compared with the nonlinearities in the model. However, even in this case, remember that the correlations are often still important, so you must use the full error matrix including the off-diagonal elements. (except when they are known to be zero, as in the simple exercises on the next slide)

simple examples of propagation of errors

$$(\Delta Z)^2 = \sum_i \sum_j \frac{\partial Z}{\partial \theta_i} \underset{\sim}{V}_{ij} \frac{\partial Z}{\partial \theta_j} .$$

Short Exercises

1. You have a cube, and you know that all the edges are exactly the same length. You measure one edge s and find $s = 10.0 \pm 0.1$ cm. What is the volume of the cube?
2. You have a second cube that looks like the first one, except that you don't know if all the edges are the same length,
   so you measure three mutually orthogonal edges s_1, s_2, s_3, and you find:

   $$s_1 = 10.0 \pm 0.1 \ cm.$$
   $$s_2 = 10.0 \pm 0.1 \ cm.$$
   $$s_3 = 10.0 \pm 0.1 \ cm.$$

   What is the volume of the second cube?

Solution to Short Exercise above

$$(\Delta Z)^2 = \sum_i \sum_j \frac{\partial Z}{\partial \theta_i} \underset{\sim}{V}_{ij} \frac{\partial Z}{\partial \theta_j} .$$

Volume of first cube, with all edges equal: $Z = s^3$

$$(\Delta Z)^2 = \left(\frac{\partial Z}{\partial s}\right)^2 \sigma_s^2$$

$$\Delta Z = \frac{\partial Z}{\partial s} \sigma_s = 3s^2 \sigma_s = 30$$

Volume of second cube: $Z = s_1 s_2 s_3$

$$(\Delta Z)^2 = \left(\frac{\partial Z}{\partial s_1}\right)^2 \sigma_1^2 + \left(\frac{\partial Z}{\partial s_2}\right)^2 \sigma_2^2 + \left(\frac{\partial Z}{\partial s_3}\right)^2 \sigma_3^2$$

$$\Delta Z = 10\sqrt{3}$$

Using the Error Matrix to understand correlations

For a 2×2 error matrix, the correlations are easy to understand. There is just one correlation coefficient $-1 \leq \rho \leq 1$. When ρ is positive, the two parameters are positively correlated, which means:

> *a given fluctuation for one parameter is more likely to be accompanied by a fluctuation of the same sign for the other parameter*

and when ρ is negative, the signs of fluctuations are more likely to be opposite.

For more than two parameters, the correlations are much more complicated. In particular, two parameters i and j may be correlated either because ρ_{ij} is large, or because both ρ_{ik} and ρ_{kj} are large for some other parameter k.

In addition, the condition that $\underset{\sim}{V}$ must be positive-definite imposes additional constraints on the ρ_{ij}, in addition to the constraint $-1 \leq \rho \leq 1$.

Correlations between many parameters

Thus it can happen that, for many parameters, even if the error matrix has small correlation coefficients, some parameter(s) may be strongly correlated with some linear combination of the others.

For this reason, the global correlation coefficients are very useful.

> *The global correlation coefficient ρ_k is the correlation between parameter k and that linear combination of all the other parameters that is most strongly correlated with parameter k.*

The global correlation coefficient can be found from the diagonal elements V_{kk} of the error matrix, and the diagonal elements $(V^{-1})_{kk}$ of its inverse,

$$\rho_k = \sqrt{1 - [V_{kk} \cdot (V^{-1})_{kk}]^{-1}}\,.$$

Beyond Minuit

Minuit is a general tool for fitting and error analysis, but there are still many problems you cannot (or should not) solve with it.

1. Problems with general constraints on the parameters.
2. Problems with very large numbers of parameters.
3. Repeated solution of the same problem (for example, fitting tracks in a detector).
4. Minimizing functions that are not parabolic at the minimum.

We will now look at better approaches to solving the above problems.

Constraints on parameters

Constraints pose serious problems in minimization and error analysis, and that is why Minuit can handle only the simplest kind of constraints. The different problems that arise depend on the kind of constraints we wish to impose. We consider first the kinds of constraints that Minuit can handle.

Rectangular Constraints (independent limits on parameters)

To handle rectangular constraints correctly, Minuit would have to verify before each call to the objective function, whether the proposed point is inside the allowed region, and if it is outside, it would have to take a different decision what to do depending on what it was trying to do when the illegal point was calculated.

Instead, Minuit uses a transformation of coordinates between

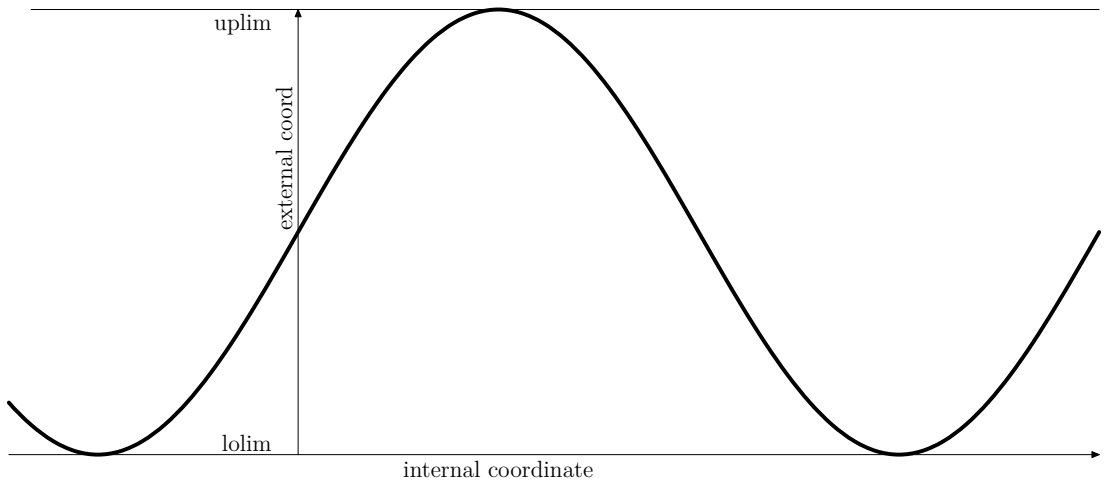
internal coordinates seen by Minuit, which may take on any value, and

external coordinates seen by the objective function FCN, which always stay inside the imposed limits

Limits on Parameters in Minuit

If the user specifies limits (a, b) on a parameter, then Minuit to converts between internal and external parameter values using :

$$P_{\text{int}} = \arcsin\left(2\,\frac{P_{\text{ext}} - a}{b - a} - 1\right), \qquad P_{\text{ext}} = a + \frac{b - a}{2}\left(\sin P_{\text{int}} + 1\right)$$



With this method, the minimizing routines like Migrad never have to worry about parameter limits, and FCN sees only allowed values.

However, when the constraint becomes active (parameter is at a limit), the error matrix becomes singular because the dimensionality of the parameter space is reduced.

Constraints by User-defined transformations

Often, a constraint can be expressed elegantly by a particular transformation. For example, suppose we have 2 parameters, (x, y) and wish to impose the constraint

$$x^2 + y^2 = R^2$$

Then obviously we can define a single free parameter ϕ for Minuit, and internally in FCN we calculate:

$$x = R\sin(\phi)\,, \qquad y = R\cos(\phi)$$

This can be extended to impose the sum of the squares of more parameters using angles to define a point on the surface of a hypersphere.

Constraints by User-defined transformations

A different transformation can be used to constrain the sum of several parameters to be equal to a constant. Suppose we wish to impose

$$0 \le x_i \le 1\,, \quad i = 1, 2, \ldots n \qquad \text{and} \qquad \sum_{i=1}^{n} x_i = 1$$

then we can define $n-1$ parameters $a_1, a_2, \ldots a_{n-1}$ for Minuit, and inside FCN calculate

$$\begin{aligned} x_1 &= a_1 \\ x_2 &= (1-a_1)a_2 \\ x_3 &= (1-a_1)(1-a_2)a_3 \\ &\vdots \\ x_n &= (1-a_1)(1-a_2)\ldots(1-a_{n-1}) \end{aligned}$$

If we tell Minuit to impose rectangular constraints on the a_i,

$$0 \le a_i \le 1\,, \qquad i = 1, 2, \ldots n-1\,,$$

then the x_i will be constrained as required.

Penalty functions

Suppose we have some parameters $\boldsymbol{\theta}$ on which we wish to impose a general constraint of the form $C(\boldsymbol{\theta}) = 0$, where C is any function of the parameters.

We can force Minuit to stay close to the surface $C(\boldsymbol{\theta}) = 0$ by adding to the objective function F another function which increases strongly as the constraints are violated:

$$F \to F + \beta C^2$$

where β (for βig) is a large positive constant. In practice, you may have to play with the size of β to have it big enough that the constraints are satisfied as accurately as you need them, but so that F is still minimized.

The error matrix coming out of a fit with penalty functions is essentially meaningless, since Minuit does not understand what you are trying to do.

The right way to handle general constraints

The penalty function method may work, but it is at best *bricolage*. The right way is to use a method (a program) designed for constraints which understands that the parameter space is going to have fewer dimensions.

If the constraint equations are linear, the problem is not difficult and there are excellent programs that handle that situation.

If the problem is linear least squares with linear constraints, the problem is very easy and is solved exactly with a few matrix manipulations.

Inequality Constraints of the form $C(\boldsymbol{\theta}) \geq 0$ are more complicated, because at any time during the minimization they may be either active (when $C(\boldsymbol{\theta}) = 0$) or passive (when $C(\boldsymbol{\theta}) > 0$).
When they are passive, we can forget about them, and when active they must be treated like equality constraints.

Fitting very many parameters

Minuit assumes that the user is interested in the error analysis for his or her fit, so it always carries around a $n \times n$ error matrix. This obviously doesn't work if you have hundreds or thousands of free parameters to fit.

When there are many parameters, the error matrix is useful only if it is sparse, that is, most of the off-diagonal elements are zero. There are specialized techniques and programs for storing and manipulating such matrices and fitting programs that use them.

The program LVMINI has been developed by Volker Blobel of Hamburg and works well when there are many parameters to fit. He has used it successfully to fit over 20 000 alignment parameters for the CMS detector.

Don't try that with Minuit!

Track-finding and track fitting

Track fitting is a problem for which Minuit is quite well suited. However, it is so important in HEP that specialized programs have been developed which are much faster and more efficient,often using specialized techniques for particular detectors.

Track-finding, that is deciding which points belong to which tracks, uses a large collection of methods, among which is track fitting, since the χ^2 of the fit can be used to determine whether a point belongs to a track. For this purpose, one should use the Kalman filter which allows adding a point to a track already fitted without re-doing the whole fit.

So Minuit can be used profitably during the development and testing of reconstruction software, but it is never as good as the specialized programs for final production.

Functions with discontinuous derivatives

The kind of functions we usually minimize (chi-square and log likelihood) are approximately parabolic. The MIGRAD method in Minuit assumes this behaviour, and simply doesn't work (or works very very very slowly) on functions with discontinuous derivatives.

There is a large class of problems, encountered mostly in business and industry, where the objective function is piecewise linear. For example, the cost to produce some item is generally a linear function of the amount of different components required. Minimizing such functions requires a different kind of technique called linear programming.

The method most used in linear programming is called the simplex method. It has nothing to do with the simplex method that appears in Minuit.

The SIMPLEX method in Minuit does not use derivatives and does not assume any smooth behaviour of the function. It is very slow and doesn't converge accurately as MIGRAD does.