Lawson, C.L., and Hanson, R. 1974, *Solving Least Squares Problems* (Englewood Cliffs, NJ: Prentice-Hall).

Forsythe, G.E., Malcolm, M.A., and Moler, C.B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 9.

15.5 Nonlinear Models

We now consider fitting when the model depends *nonlinearly* on the set of M unknown parameters $a_k, k = 1, 2, \ldots, M$. We use the same approach as in previous sections, namely to define a χ^2 merit function and determine best-fit parameters by its minimization. With nonlinear dependences, however, the minimization must proceed iteratively. Given trial values for the parameters, we develop a procedure that improves the trial solution. The procedure is then repeated until χ^2 stops (or effectively stops) decreasing.

How is this problem different from the general nonlinear function minimization problem already dealt with in Chapter 10? Superficially, not at all: Sufficiently close to the minimum, we expect the χ^2 function to be well approximated by a quadratic form, which we can write as

$$\chi^2(\mathbf{a}) \approx \gamma - \mathbf{d} \cdot \mathbf{a} + \frac{1}{2}\mathbf{a} \cdot \mathbf{D} \cdot \mathbf{a} \tag{15.5.1}$$

where $\mathbf{d}$ is an M-vector and $\mathbf{D}$ is an $M \times M$ matrix. (Compare equation 10.6.1.) If the approximation is a good one, we know how to jump from the current trial parameters $\mathbf{a}_{\text{cur}}$ to the minimizing ones $\mathbf{a}_{\min}$ in a single leap, namely

$$\mathbf{a}_{\min} = \mathbf{a}_{\text{cur}} + \mathbf{D}^{-1} \cdot \left[-\nabla\chi^2(\mathbf{a}_{\text{cur}})\right] \tag{15.5.2}$$

(Compare equation 10.7.4.)

On the other hand, (15.5.1) might be a poor local approximation to the shape of the function that we are trying to minimize at $\mathbf{a}_{\text{cur}}$. In that case, about all we can do is take a step down the gradient, as in the steepest descent method (§10.6). In other words,

$$\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{cur}} - \text{constant} \times \nabla\chi^2(\mathbf{a}_{\text{cur}}) \tag{15.5.3}$$

where the constant is small enough not to exhaust the downhill direction.

To use (15.5.2) or (15.5.3), we must be able to compute the gradient of the χ^2 function at any set of parameters $\mathbf{a}$. To use (15.5.2) we also need the matrix $\mathbf{D}$, which is the second derivative matrix (Hessian matrix) of the χ^2 merit function, at any $\mathbf{a}$.

Now, this is the crucial difference from Chapter 10: There, we had no way of directly evaluating the Hessian matrix. We were given only the ability to evaluate the function to be minimized and (in some cases) its gradient. Therefore, we had to resort to iterative methods *not just* because our function was nonlinear, *but also* in order to build up information about the Hessian matrix. Sections 10.7 and 10.6 concerned themselves with two different techniques for building up this information.

Here, life is much simpler. We *know* exactly the form of χ^2, since it is based on a model function that we ourselves have specified. Therefore the Hessian matrix is known to us. Thus we are free to use (15.5.2) whenever we care to do so. The only reason to use (15.5.3) will be failure of (15.5.2) to improve the fit, signaling failure of (15.5.1) as a good local approximation.

Calculation of the Gradient and Hessian

The model to be fitted is

$$y = y(x; \mathbf{a}) \tag{15.5.4}$$

and the χ^2 merit function is

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{N} \left[\frac{y_i - y(x_i; \mathbf{a})}{\sigma_i} \right]^2 \tag{15.5.5}$$

The gradient of χ^2 with respect to the parameters $\mathbf{a}$, which will be zero at the χ^2 minimum, has components

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_{i=1}^{N} \frac{[y_i - y(x_i; \mathbf{a})]}{\sigma_i^2} \frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \qquad k = 1, 2, \ldots, M \tag{15.5.6}$$

Taking an additional partial derivative gives

$$\frac{\partial^2 \chi^2}{\partial a_k \partial a_l} = 2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \frac{\partial y(x_i; \mathbf{a})}{\partial a_l} - [y_i - y(x_i; \mathbf{a})] \frac{\partial^2 y(x_i; \mathbf{a})}{\partial a_l \partial a_k} \right] \tag{15.5.7}$$

It is conventional to remove the factors of 2 by defining

$$\beta_k \equiv -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k} \qquad \alpha_{kl} \equiv \frac{1}{2} \frac{\partial^2 \chi^2}{\partial a_k \partial a_l} \tag{15.5.8}$$

making $[\alpha] = \frac{1}{2}\mathbf{D}$ in equation (15.5.2), in terms of which that equation can be rewritten as the set of linear equations

$$\sum_{l=1}^{M} \alpha_{kl}\, \delta a_l = \beta_k \tag{15.5.9}$$

This set is solved for the increments δa_l that, added to the current approximation, give the next approximation. In the context of least-squares, the matrix $[\alpha]$, equal to one-half times the Hessian matrix, is usually called the *curvature matrix*.

Equation (15.5.3), the steepest descent formula, translates to

$$\delta a_l = \text{constant} \times \beta_l \tag{15.5.10}$$

Note that the components α_{kl} of the Hessian matrix (15.5.7) depend both on the first derivatives and on the second derivatives of the basis functions with respect to their parameters. Some treatments proceed to ignore the second derivative without comment. We will ignore it also, but only *after* a few comments.

Second derivatives occur because the gradient (15.5.6) already has a dependence on $\partial y/\partial a_k$, so the next derivative simply must contain terms involving $\partial^2 y/\partial a_l \partial a_k$. The second derivative term can be dismissed when it is zero (as in the linear case of equation 15.4.8), or small enough to be negligible when compared to the term involving the first derivative. It also has an additional possibility of being ignorably small in practice: The term multiplying the second derivative in equation (15.5.7) is $[y_i - y(x_i; \mathbf{a})]$. For a successful model, this term should just be the random measurement error of each point. This error can have either sign, and should in general be uncorrelated with the model. Therefore, the second derivative terms tend to cancel out when summed over i.

Inclusion of the second-derivative term can in fact be destabilizing if the model fits badly or is contaminated by outlier points that are unlikely to be offset by compensating points of opposite sign. From this point on, we will always use as the definition of α_{kl} the formula

$$\alpha_{kl} = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \frac{\partial y(x_i; \mathbf{a})}{\partial a_l} \right] \tag{15.5.11}$$

This expression more closely resembles its linear cousin (15.4.8). You should understand that minor (or even major) fiddling with $[\alpha]$ has no effect at all on what final set of parameters $\mathbf{a}$ is reached, but affects only the iterative route that is taken in getting there. The condition at the χ^2 minimum, that $\beta_k = 0$ for all k, is independent of how $[\alpha]$ is defined.

Levenberg-Marquardt Method

Marquardt [1] has put forth an elegant method, related to an earlier suggestion of Levenberg, for varying smoothly between the extremes of the inverse-Hessian method (15.5.9) and the steepest descent method (15.5.10). The latter method is used far from the minimum, switching continuously to the former as the minimum is approached. This *Levenberg-Marquardt method* (also called *Marquardt method*) works very well in practice and has become the standard of nonlinear least-squares routines.

The method is based on two elementary, but important, insights. Consider the "constant" in equation (15.5.10). What should it be, even in order of magnitude? What sets its scale? There is no information about the answer in the gradient. That tells only the slope, not how far that slope extends. Marquardt's first insight is that the components of the Hessian matrix, even if they are not usable in any precise fashion, give *some* information about the order-of-magnitude scale of the problem.

The quantity χ^2 is nondimensional, i.e., is a pure number; this is evident from its definition (15.5.5). On the other hand, β_k has the dimensions of $1/a_k$, which may well be dimensional, i.e., have units like cm^{-1}, or kilowatt-hours, or whatever. (In fact, each component of β_k can have different dimensions!) The constant of proportionality between β_k and δa_k must therefore have the dimensions of a_k^2. Scan the components of $[\alpha]$ and you see that there is only one obvious quantity with these dimensions, and that is $1/\alpha_{kk}$, the reciprocal of the diagonal element. So that must set the scale of the constant. But that scale might itself be too big. So let's divide the constant by some (nondimensional) fudge factor λ, with the possibility of setting $\lambda \gg 1$ to cut down the step. In other words, replace equation (15.5.10) by

$$\delta a_l = \frac{1}{\lambda \alpha_{ll}} \beta_l \qquad \text{or} \qquad \lambda\, \alpha_{ll}\, \delta a_l = \beta_l \tag{15.5.12}$$

It is necessary that a_{ll} be positive, but this is guaranteed by definition (15.5.11) — another reason for adopting that equation.

Marquardt's second insight is that equations (15.5.12) and (15.5.9) can be combined if we define a new matrix α' by the following prescription

$$\begin{aligned} \alpha'_{jj} &\equiv \alpha_{jj}(1+\lambda) \\ \alpha'_{jk} &\equiv \alpha_{jk} \qquad (j \neq k) \end{aligned} \tag{15.5.13}$$

and then replace both (15.5.12) and (15.5.9) by

$$\sum_{l=1}^{M} \alpha'_{kl}\, \delta a_l = \beta_k \tag{15.5.14}$$

When λ is very large, the matrix α' is forced into being *diagonally dominant*, so equation (15.5.14) goes over to be identical to (15.5.12). On the other hand, as λ approaches zero, equation (15.5.14) goes over to (15.5.9).

Given an initial guess for the set of fitted parameters **a**, the recommended Marquardt recipe is as follows:

- Compute $\chi^2(\mathbf{a})$.
- Pick a modest value for λ, say $\lambda = 0.001$.
- (†) Solve the linear equations (15.5.14) for $\delta\mathbf{a}$ and evaluate $\chi^2(\mathbf{a} + \delta\mathbf{a})$.
- If $\chi^2(\mathbf{a} + \delta\mathbf{a}) \geq \chi^2(\mathbf{a})$, *increase* λ by a factor of 10 (or any other substantial factor) and go back to (†).
- If $\chi^2(\mathbf{a} + \delta\mathbf{a}) < \chi^2(\mathbf{a})$, *decrease* λ by a factor of 10, update the trial solution $\mathbf{a} \leftarrow \mathbf{a} + \delta\mathbf{a}$, and go back to (†).

Also necessary is a condition for stopping. Iterating to convergence (to machine accuracy or to the roundoff limit) is generally wasteful and unnecessary since the minimum is at best only a statistical estimate of the parameters **a**. As we will see in §15.6, a change in the parameters that changes χ^2 by an amount $\ll 1$ is *never* statistically meaningful.

Furthermore, it is not uncommon to find the parameters wandering around near the minimum in a flat valley of complicated topography. The reason is that Marquardt's method generalizes the method of normal equations (§15.4), hence has the same problem as that method with regard to near-degeneracy of the minimum. Outright failure by a zero pivot is possible, but unlikely. More often, a small pivot will generate a large correction which is then rejected, the value of λ being then increased. For sufficiently large λ the matrix $[\alpha']$ is positive definite and can have no small pivots. Thus the method does tend to stay away from zero pivots, but at the cost of a tendency to wander around doing steepest descent in very un-steep degenerate valleys.

These considerations suggest that, in practice, one might as well stop iterating on the first or second occasion that χ^2 decreases by a negligible amount, say either less than 0.01 absolutely or (in case roundoff prevents that being reached) some fractional amount like 10^{-3}. Don't stop after a step where χ^2 *increases*: That only shows that λ has not yet adjusted itself optimally.

Once the acceptable minimum has been found, one wants to set $\lambda = 0$ and compute the matrix

$$[C] \equiv [\alpha]^{-1} \tag{15.5.15}$$

which, as before, is the estimated covariance matrix of the standard errors in the fitted parameters **a** (see next section).

The following pair of functions encodes Marquardt's method for nonlinear parameter estimation. Much of the organization matches that used in `lfit` of §15.4. In particular the array `ia[1..ma]` must be input with components one or zero corresponding to whether the respective parameter values `a[1..ma]` are to be fitted for or held fixed at their input values, respectively.

The routine `mrqmin` performs one iteration of Marquardt's method. It is first called (once) with `alamda` < 0, which signals the routine to initialize. `alamda` is set on the first and all subsequent calls to the suggested value of λ for the next iteration; `a` and `chisq` are always given back as the best parameters found so far and their χ^2. When convergence is deemed satisfactory, set `alamda` to zero before a final call. The matrices `alpha` and `covar` (which were used as workspace in all previous calls) will then be set to the curvature and covariance matrices for the converged parameter values. The arguments `alpha`, `a`, and `chisq` must not be modified between calls, nor should `alamda` be, except to set it to zero for the final call. When an uphill step is taken, `chisq` and `a` are given back with their input (best) values, but `alamda` is set to an increased value.

The routine `mrqmin` calls the routine `mrqcof` for the computation of the matrix $[\alpha]$ (equation 15.5.11) and vector β (equations 15.5.6 and 15.5.8). In turn `mrqcof` calls the user-supplied routine `funcs(x,a,y,dyda)`, which for input values `x` $\equiv x_i$ and `a` $\equiv$ **a** calculates the model function `y` $\equiv y(x_i; \mathbf{a})$ and the vector of derivatives `dyda` $\equiv \partial y/\partial a_k$.

```
#include "nrutil.h"

void mrqmin(float x[], float y[], float sig[], int ndata, float a[], int ia[],
    int ma, float **covar, float **alpha, float *chisq,
    void (*funcs)(float, float [], float *, float [], int), float *alamda)
```

Levenberg-Marquardt method, attempting to reduce the value χ^2 of a fit between a set of data points `x[1..ndata]`, `y[1..ndata]` with individual standard deviations `sig[1..ndata]`, and a nonlinear function dependent on `ma` coefficients `a[1..ma]`. The input array `ia[1..ma]` indicates by nonzero entries those components of `a` that should be fitted for, and by zero entries those components that should be held fixed at their input values. The program returns current best-fit values for the parameters `a[1..ma]`, and χ^2 = `chisq`. The arrays `covar[1..ma][1..ma]`, `alpha[1..ma][1..ma]` are used as working space during most iterations. Supply a routine `funcs(x,a,yfit,dyda,ma)` that evaluates the fitting function `yfit`, and its derivatives `dyda[1..ma]` with respect to the fitting parameters `a` at `x`. On the first call provide an initial guess for the parameters `a`, and set `alamda<0` for initialization (which then sets `alamda=.001`). If a step succeeds `chisq` becomes smaller and `alamda` decreases by a factor of 10. If a step fails `alamda` grows by a factor of 10. You must call this routine repeatedly until convergence is achieved. Then, make one final call with alamda=0, so that covar[1..ma][1..ma] returns the covariance matrix, and alpha the curvature matrix. (Parameters held fixed will return zero covariances.)

```
{
    void covsrt(float **covar, int ma, int ia[], int mfit);
    void gaussj(float **a, int n, float **b, int m);
    void mrqcof(float x[], float y[], float sig[], int ndata, float a[],
        int ia[], int ma, float **alpha, float beta[], float *chisq,
        void (*funcs)(float, float [], float *, float [], int));
    int j,k,l;
    static int mfit;
    static float ochisq,*atry,*beta,*da,**oneda;

    if (*alamda < 0.0) {                Initialization.
        atry=vector(1,ma);
        beta=vector(1,ma);
        da=vector(1,ma);
        for (mfit=0,j=1;j<=ma;j++)
            if (ia[j]) mfit++;
        oneda=matrix(1,mfit,1,1);
        *alamda=0.001;
        mrqcof(x,y,sig,ndata,a,ia,ma,alpha,beta,chisq,funcs);
        ochisq=(*chisq);
        for (j=1;j<=ma;j++) atry[j]=a[j];
    }
    for (j=1;j<=mfit;j++) {             Alter linearized fitting matrix, by augmenting di-
        for (k=1;k<=mfit;k++) covar[j][k]=alpha[j][k];       agonal elements.
        covar[j][j]=alpha[j][j]*(1.0+(*alamda));
        oneda[j][1]=beta[j];
    }
    gaussj(covar,mfit,oneda,1);         Matrix solution.
    for (j=1;j<=mfit;j++) da[j]=oneda[j][1];
    if (*alamda == 0.0) {               Once converged, evaluate covariance matrix.
        covsrt(covar,ma,ia,mfit);
        covsrt(alpha,ma,ia,mfit);       Spread out alpha to its full size too.
        free_matrix(oneda,1,mfit,1,1);
        free_vector(da,1,ma);
        free_vector(beta,1,ma);
        free_vector(atry,1,ma);
        return;
    }
    for (j=0,l=1;l<=ma;l++)             Did the trial succeed?
        if (ia[l]) atry[l]=a[l]+da[++j];
    mrqcof(x,y,sig,ndata,atry,ia,ma,covar,da,chisq,funcs);
    if (*chisq < ochisq) {              Success, accept the new solution.
        *alamda *= 0.1;
        ochisq=(*chisq);
        for (j=1;j<=mfit;j++) {
            for (k=1;k<=mfit;k++) alpha[j][k]=covar[j][k];
            beta[j]=da[j];
        }
        for (l=1;l<=ma;l++) a[l]=atry[l];
    } else {                            Failure, increase alamda and return.
        *alamda *= 10.0;
        *chisq=ochisq;
    }
}
```

Notice the use of the routine covsrt from §15.4. This is merely for rearranging the covariance matrix covar into the order of all ma parameters. The above routine also makes use of

```
#include "nrutil.h"

void mrqcof(float x[], float y[], float sig[], int ndata, float a[], int ia[],
    int ma, float **alpha, float beta[], float *chisq,
    void (*funcs)(float, float [], float *, float [], int))
Used by mrqmin to evaluate the linearized fitting matrix alpha, and vector beta as in (15.5.8),
and calculate χ².
{
    int i,j,k,l,m,mfit=0;
    float ymod,wt,sig2i,dy,*dyda;

    dyda=vector(1,ma);
    for (j=1;j<=ma;j++)
        if (ia[j]) mfit++;
    for (j=1;j<=mfit;j++) {                 Initialize (symmetric) alpha, beta.
        for (k=1;k<=j;k++) alpha[j][k]=0.0;
        beta[j]=0.0;
    }
    *chisq=0.0;
    for (i=1;i<=ndata;i++) {                Summation loop over all data.
        (*funcs)(x[i],a,&ymod,dyda,ma);
        sig2i=1.0/(sig[i]*sig[i]);
        dy=y[i]-ymod;
        for (j=0,l=1;l<=ma;l++) {
            if (ia[l]) {
                wt=dyda[l]*sig2i;
                for (j++,k=0,m=1;m<=l;m++)
                    if (ia[m]) alpha[j][++k] += wt*dyda[m];
                beta[j] += dy*wt;
            }
        }
        *chisq += dy*dy*sig2i;              And find χ².
    }
    for (j=2;j<=mfit;j++)                   Fill in the symmetric side.
        for (k=1;k<j;k++) alpha[k][j]=alpha[j][k];
    free_vector(dyda,1,ma);
}
```

Example

The following function `fgauss` is an example of a user-supplied function `funcs`. Used with the above routine `mrqmin` (in turn using `mrqcof`, `covsrt`, and `gaussj`), it fits for the model

$$y(x) = \sum_{k=1}^{K} B_k \exp\left[-\left(\frac{x - E_k}{G_k}\right)^2\right] \tag{15.5.16}$$

which is a sum of K Gaussians, each having a variable position, amplitude, and width. We store the parameters in the order $B_1, E_1, G_1, B_2, E_2, G_2, \ldots, B_K, E_K, G_K$.

```
#include <math.h>

void fgauss(float x, float a[], float *y, float dyda[], int na)
```
$y(x;a)$ is the sum of na/3 Gaussians (15.5.16). The amplitude, center, and width of the Gaussians are stored in consecutive locations of a: a[i] $= B_k$, a[i+1] $= E_k$, a[i+2] $= G_k$, $k = 1, ..., $na/3. The dimensions of the arrays are a[1..na], dyda[1..na].
```
{
    int i;
    float fac,ex,arg;

    *y=0.0;
    for (i=1;i<=na-1;i+=3) {
        arg=(x-a[i+1])/a[i+2];
        ex=exp(-arg*arg);
        fac=a[i]*ex*2.0*arg;
        *y += a[i]*ex;
        dyda[i]=ex;
        dyda[i+1]=fac/a[i+2];
        dyda[i+2]=fac*arg/a[i+2];
    }
}
```

More Advanced Methods for Nonlinear Least Squares

The Levenberg-Marquardt algorithm can be implemented as a model-trust region method for minimization (see §9.7 and ref. [2]) applied to the special case of a least squares function. A code of this kind due to Moré [3] can be found in MINPACK [4]. Another algorithm for nonlinear least-squares keeps the second-derivative term we dropped in the Levenberg-Marquardt method whenever it would be better to do so. These methods are called "full Newton-type" methods and are reputed to be more robust than Levenberg-Marquardt, but more complex. One implementation is the code NL2SOL [5].

CITED REFERENCES AND FURTHER READING:

Bevington, P.R. 1969, *Data Reduction and Error Analysis for the Physical Sciences* (New York: McGraw-Hill), Chapter 11.

Marquardt, D.W. 1963, *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, pp. 431–441. [1]

Jacobs, D.A.H. (ed.) 1977, *The State of the Art in Numerical Analysis* (London: Academic Press), Chapter III.2 (by J.E. Dennis).

Dennis, J.E., and Schnabel, R.B. 1983, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (Englewood Cliffs, NJ: Prentice-Hall). [2]

Moré, J.J. 1977, in *Numerical Analysis*, Lecture Notes in Mathematics, vol. 630, G.A. Watson, ed. (Berlin: Springer-Verlag), pp. 105–116. [3]

Moré, J.J., Garbow, B.S., and Hillstrom, K.E. 1980, *User Guide for MINPACK-1*, Argonne National Laboratory Report ANL-80-74. [4]

Dennis, J.E., Gay, D.M, and Welsch, R.E. 1981, *ACM Transactions on Mathematical Software*, vol. 7, pp. 348–368; *op. cit.*, pp. 369–383. [5].

15.6 Confidence Limits on Estimated Model Parameters

Several times already in this chapter we have made statements about the standard errors, or uncertainties, in a set of M estimated parameters $\mathbf{a}$. We have given some formulas for computing standard deviations or variances of individual parameters (equations 15.2.9, 15.4.15, 15.4.19), as well as some formulas for covariances between pairs of parameters (equation 15.2.10; remark following equation 15.4.15; equation 15.4.20; equation 15.5.15).

In this section, we want to be more explicit regarding the precise meaning of these quantitative uncertainties, and to give further information about how quantitative confidence limits on fitted parameters can be estimated. The subject can get somewhat technical, and even somewhat confusing, so we will try to make precise statements, even when they must be offered without proof.

Figure 15.6.1 shows the conceptual scheme of an experiment that "measures" a set of parameters. There is some underlying true set of parameters $\mathbf{a}_{\text{true}}$ that are known to Mother Nature but hidden from the experimenter. These true parameters are statistically realized, along with random measurement errors, as a measured data set, which we will symbolize as $\mathcal{D}_{(0)}$. The data set $\mathcal{D}_{(0)}$ *is* known to the experimenter. He or she fits the data to a model by χ^2 minimization or some other technique, and obtains measured, i.e., fitted, values for the parameters, which we here denote $\mathbf{a}_{(0)}$.

Because measurement errors have a random component, $\mathcal{D}_{(0)}$ is not a unique realization of the true parameters $\mathbf{a}_{\text{true}}$. Rather, there are infinitely many other realizations of the true parameters as "hypothetical data sets" each of which *could* have been the one measured, but happened not to be. Let us symbolize these by $\mathcal{D}_{(1)}, \mathcal{D}_{(2)}, \ldots$. Each one, had it been realized, would have given a slightly different set of fitted parameters, $\mathbf{a}_{(1)}, \mathbf{a}_{(2)}, \ldots$, respectively. These parameter sets $\mathbf{a}_{(i)}$ therefore occur with some probability distribution in the M-dimensional space of all possible parameter sets $\mathbf{a}$. The actual measured set $\mathbf{a}_{(0)}$ is one member drawn from this distribution.

Even more interesting than the probability distribution of $\mathbf{a}_{(i)}$ would be the distribution of the difference $\mathbf{a}_{(i)} - \mathbf{a}_{\text{true}}$. This distribution differs from the former one by a translation that puts Mother Nature's true value at the origin. If we knew *this* distribution, we would know everything that there is to know about the quantitative uncertainties in our experimental measurement $\mathbf{a}_{(0)}$.

So the name of the game is to find some way of estimating or approximating the probability distribution of $\mathbf{a}_{(i)} - \mathbf{a}_{\text{true}}$ without knowing $\mathbf{a}_{\text{true}}$ and without having available to us an infinite universe of hypothetical data sets.

Monte Carlo Simulation of Synthetic Data Sets

Although the measured parameter set $\mathbf{a}_{(0)}$ is not the true one, let us consider a fictitious world in which it *was* the true one. Since we hope that our measured parameters are not *too* wrong, we hope that that fictitious world is not too different from the actual world with parameters $\mathbf{a}_{\text{true}}$. In particular, let us hope — no, let us *assume* — that the shape of the probability distribution $\mathbf{a}_{(i)} - \mathbf{a}_{(0)}$ in the fictitious world is the same, or very nearly the same, as the shape of the probability distribution