NRiC §15.

Model depends on adjustable parameters.

Can be used for “constrained interpolation.”

Basic approach:

1. Choose *figure-of-merit* function (e.g., $\chi^2$).
3. Compute *goodness-of-fit*.
4. Compute *error estimates* for parameters.
Least Squares Fitting

Suppose we want to fit $N$ data points $(x_i, y_i)$ with a function that depends on $M$ parameters $a_j$ and that each data point has a standard deviation $\sigma_i$. The maximum likelihood estimate of the model parameters is obtained by minimizing:

$$\chi^2 \equiv \sum_{i=1}^{N} \left[ \frac{y_i - y(x_i; a_1 \ldots a_M)}{\sigma_i} \right]^2 .$$

Assuming the errors are normally distributed, a “good fit” has $\chi^2 \sim \nu$, where $\nu = N - M$.

NOTE: Assumption of normal errors means glitches or outliers in data may overbias the fit—see NRiC §15.7 for discussion of more robust methods.

Grossly overestimated (underestimated) $\sigma_i$’s may give incorrect impression that fit is very good (very bad).
If uncertain about reliability of goodness-of-fit measure, could do *Monte Carlo simulations* of fits to synthetic data.

Question: what to do if $\sigma_i$’s not known? Answer: choose an arbitrary constant $\sigma$, perform the fit, then estimate $\sigma$ from the fit:

$$\sigma^2 = \sum_{i=1}^{N} [y_i - y(x_i)]^2 / \nu$$

(note the denominator is what $\chi^2$ *should* approximately be equal to, if the fit is good).
Fitting Data to a Straight Line (Linear Regression)

For this case the model is simply:

\[ y(x) = y(x; a, b) = a + bx, \]

and

\[ \chi^2(a, b) = \sum_{i=1}^{N} \left( \frac{y_i - a - bx_i}{\sigma_i} \right)^2. \]

Derive formula for best-fit parameters by setting \( \partial \chi^2 / \partial a = 0 = \partial \chi^2 / \partial b \). See \textit{NRiC} §15.2 for the derivation (note: \texttt{sm} uses the same formulae for its \texttt{lsg} routine).
Derive uncertainties in $a$ and $b$ from propagation of errors:

$$
\sigma_f^2 = \sum_{i=1}^{N} \sigma_i^2 \left( \frac{\partial f}{\partial y_i} \right)^2,
$$

where $f = a(x_i, y_i, \sigma_i), b(x_i, y_i, \sigma_i)$ in this case (the $x_i$’s have no uncertainties).

Want probability that $\chi^2$ is bad by chance

$$
Q = \text{gammq}(\frac{N - 2}{2}, \frac{\chi^2}{2}) > 10^{-3} \text{ (here } \frac{N - 2}{2} \equiv \nu/2).$$
**General Linear Least Squares**

- Can generalize to any combination that is linear in $a_j$’s:

  $$y(x) = \sum_{j=1}^{M} a_j X_j(x),$$

  e.g., $y(x) = a_1 + a_2 x + a_3 x^2 + \ldots + a_M x^{M-1}$, or sines and cosines.

- Define $N \times M$ design matrix $A_{ij} = X_j(x_i)/\sigma_i$. Note $N \geq M$ for the fit to make sense.

- Also define vector $b$ of length $N$ where $b_i = y_i/\sigma_i$, and vector $a$ of length $M$ where $a_i = a_1, \ldots, a_M$.

- Then we wish to find $a$ that minimizes:

  $$\chi^2 = |A a - b|^2.$$

- This is what SVD solves!
Recall for SVD we had \( \mathbf{A} = \mathbf{UWV}^T \).

Rewriting the SVD solution we get:

\[
a = \sum_{j=1}^{M} \left( \frac{\mathbf{U}(j) \cdot \mathbf{b}}{w_j} \right) \mathbf{V}(j),
\]

where \( \mathbf{U}(j) \) (length \( N \)) and \( \mathbf{V}(j) \) (length \( M \)) denote columns of \( \mathbf{U} \) and \( \mathbf{V} \), respectively.

As before, if \( w_j \) is small (or zero), can set \( 1/w_j = 0 \).

Useful because least-squares problems are generally both overdetermined \( (N > M) \) and underdetermined (ambiguous combinations of parameters exist)!

Can also compute variances of estimated parameters:

\[
\sigma^2(a_j) = \sum_{i=1}^{M} (V_{ji}/w_i)^2.
\]

Can generalize to multidimensions.
Nonlinear Models

Suppose model depends *nonlinearly* on the \( a_j \)'s, e.g.,

\[
y(x) = a_1 \sin(a_2 x + a_3).
\]

Still minimize \( \chi^2 \), but must proceed iteratively:

- Use \( a_{\text{next}} = a_{\text{cur}} - \lambda \nabla \chi^2(a_{\text{cur}}) \) far from minimum (steepest descent), where \( \lambda \) is a constant.
- Use \( a_{\text{next}} = a_{\text{cur}} - D^{-1}[\nabla \chi^2(a_{\text{cur}})] \) close to minimum, where \( D \) is the *Hessian* matrix.
D comes from considering Taylor series expansion of $f(x)$ near a point $P$:

$$f(x) = f(P) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \ldots \approx c - b \cdot x + \frac{1}{2} x A x,$$

where $c \equiv f(P)$, $b \equiv -\nabla f|_P$, and $A_{ij} \equiv \frac{\partial^2 f}{\partial x_i \partial x_j}|_P$. Here $A$ is the Hessian matrix. Note that $\nabla f = A x - b$. 
Close to its minimum, $\chi^2$ can be approximated by the above quadratic form, and so an “exact” step can be taken to get to the point where $\nabla \chi^2 = 0$. The step is just $x' - x = -A^{-1}\nabla f|_P$.

In practice, terms involving the second derivatives of $y$ with respect to the fit parameters can be ignored, so the Hessian matrix is much simpler to compute (recall the $\chi^2$ function contains the model $y$).

The Levenberg-Marquardt method adjusts $\lambda$ to smooth the transition between these two regimes (vary between a diagonal matrix and inverse Hessian).

Cf. NRiC §15.5 for details of the L-M method.
Levenberg-Marquardt method in NRiC

NRiC provides two routines, `mrqmin()` and `mrqcof()`, that implement the L-M method.

The user must provide a function that computes $y(x_i)$ as well as all the partial derivatives $\frac{\partial y}{\partial a_j}$ evaluated at $x_i$.

The routine `mrqmin()` is called iteratively until a successful step (i.e., one in which $\lambda$ gets smaller) changes $\chi^2$ by less than a fractional amount, like 0.001 (no point in doing better).
Points to consider:

- The argument list for `mrqmin()` is *very* complicated. For example, you can request that some parameters be held fixed (via input array `ia`).
- You need to specify an initial guess for each $a_j$ (and set $\lambda < 0$).
- Estimated variances in the parameters are returned as the diagonal elements of the *covariance matrix* (`covar`), if you call `mrqmin()` with $\lambda = 0$.
- Also calls *NRiC* routines `covsrt()` and `gaussj()`.
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 of Chi^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 of the parameters a[1..ma], and Chi^2=chisq. ...
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 alambda<0 for initialization (which sets alambda=0.001). If a step succeeds chisq becomes smaller and alambda decreases by a factor of 10. If a step fails alambda grows by a factor of 10. You must call this routine repeatedly until convergence is achieved. Then, make a final call with alambda=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), float *alamda);
......
......
void fgauss(float x, float a[], float *y, float dyda[], int na)
//The dimensions of the arrays are a[1..na], dyda[1..na].
{
......
......