Chapter 1

Model fitting by least squares

The first level of computer use in science and engineering is "**modeling**." Beginning from physical principles and design ideas, the computer mimics nature. After this, the worker looks at the result and thinks a while, then alters the modeling program and tries again. The next, deeper level of computer use is that the computer itself examines the results of modeling and reruns the modeling job. This deeper level is variously called "**fitting**" or "**inversion**." We inspect the conjugate gradient (CG) method of fitting and write a subroutine for it that will be used in most of the examples in this monograph.

1.1 HOW TO DIVIDE NOISY SIGNALS

A single parameter inversion problem arises in Fourier analysis, where we seek a "best answer" at each frequency, then combine all the frequencies to get a best signal. Thus emerges a wide family of interesting and useful applications. However, Fourier analysis first requires us to introduce complex numbers into statistical estimation.

Multiplication in the Fourier domain is convolution in the time domain. Fourier-domain division is time-domain deconvolution. This division is challenging when F has observational error. Failure erupts if zero division occurs. More insidious are the poor results we obtain when zero division is avoided by a near miss.

1.1.1 Dividing by zero smoothly

Think of any real numbers x, y, and f and any program containing $x = y/f$. How can we change the program so that it never divides by zero? A popular answer is to change $x = y/f$ to $x = yf/(f^2 + \epsilon^2)$, where ϵ is any tiny value. When $|f| >> |\epsilon|$, then x is approximately y/f as expected. But when the divisor f vanishes, the result is safely zero instead of infinity. The transition is smooth, but some criterion is needed to choose the value of ϵ. This method may not be the only way or the best way to cope with **zero division**, but it is a good way, and it permeates the subject of signal analysis.

To apply this method in the Fourier domain, suppose X, Y, and F are complex numbers. What do we do then with $X = Y/F$? We multiply the top and bottom by the complex conjugate $\overline{F}$, and again add ϵ^2 to the denominator. Thus,

$$X(\omega) \quad = \quad \frac{\overline{F(\omega)}\; Y(\omega)}{\overline{F(\omega)} F(\omega) \ + \ \epsilon^2} \tag{1.1}$$

Now the denominator must always be a positive number greater than zero, so division is always safe. Equation (1.1) ranges continuously from **inverse filter**ing with $X = Y/F$ to filtering with $X = \overline{F}Y$, which is called "**matched filter**ing." Notice that for any complex number F, the phase of $1/F$ equals the phase of $\overline{F}$, so all these filters have the same phase.

The filter $\overline{F}$ is called the "matched filter." If nature created Y by random bursts of energy into F, then building X from Y and F in the limit $\epsilon \rightarrow \infty$ in equation (1.1) amounts to $X = Y\overline{F}$ which **crosscorrelate**s F with the randomly placed copies of F that are in Y.

1.1.2 Damped solution

Equation (1.1) is the solution to an optimization problem that arises in many applications. Now that we know the solution, let us formally define the problem. First, we will solve a simpler problem with real values: we will choose to minimize the **quadratic function** of x:

$$Q(x) \quad = \quad (fx - y)^2 + \epsilon^2 x^2 \tag{1.2}$$

The second term is called a "**damping** factor" because it prevents x from going to $\pm\infty$ when $f \rightarrow 0$. Set $dQ/dx = 0$, which gives

$$0 \quad = \quad f(fx - y) + \epsilon^2 x \tag{1.3}$$

This yields the earlier answer $x = fy/(f^2 + \epsilon^2)$.

With Fourier transforms, the signal X is a complex number at each frequency ω. So we generalize equation (1.2) to

$$Q(\bar{X}, X) \quad = \quad (\overline{FX - Y})(FX - Y) + \epsilon^2 \bar{X} X \quad = \quad (\bar{X}\bar{F} - \bar{Y})(FX - Y) + \epsilon^2 \bar{X} X \tag{1.4}$$

To minimize Q we could use a real-values approach, where we express $X = u + iv$ in terms of two real values u and v and then set $\partial Q/\partial u = 0$ and $\partial Q/\partial v = 0$. The approach we will take, however, is to use complex values, where we set $\partial Q/\partial X = 0$ and $\partial Q/\partial \bar{X} = 0$. Let us examine $\partial Q/\partial \bar{X}$:

$$\frac{\partial Q(\bar{X}, X)}{\partial \bar{X}} \quad = \quad \bar{F}(FX - Y) + \epsilon^2 X \quad = \quad 0 \tag{1.5}$$

The derivative $\partial Q/\partial X$ is the complex conjugate of $\partial Q/\partial \bar{X}$. So if either is zero, the other is too. Thus we do not need to specify both $\partial Q/\partial X = 0$ and $\partial Q/\partial \bar{X} = 0$. I usually set $\partial Q/\partial \bar{X}$ equal to zero. Solving equation (1.5) for X gives equation (1.1).

Equation (1.1) solves $Y = XF$ for X, giving the solution for what is called "the **deconvolution** problem with a known wavelet F." Analogously we can use $Y = XF$ when the filter F is unknown, but the input X and output Y are given. Simply interchange X and F in the derivation and result.

1.2 MULTIVARIATE LEAST SQUARES

1.2.1 Inside an abstract vector

In engineering, a vector has three scalar components which correspond to the three dimensions of the space in which we live. In least-squares data analysis, a vector is a one-dimensional array that can contain many different things. Such an array is an "**abstract vector**." For example, in earthquake studies, the vector might contain the time an earthquake began as well as its latitude, longitude, and depth. Alternately, the abstract vector might contain as many components as there are seismometers, and each component might be the arrival time of an earthquake wave. In signal analysis, the vector might contain the values of a signal at successive instants in time or, alternately, a collection of signals. These signals might be "**multiplex**ed" (interlaced) or "demultiplexed" (all of each signal preceding the next). In image analysis, the one-dimensional array might contain an image, which could itself be thought of as an array of signals. Vectors, including abstract vectors, are usually denoted by **boldface letters** such as $\mathbf{p}$ and $\mathbf{s}$. Like physical vectors, abstract vectors are **orthogonal** when their dot product vanishes: $\mathbf{p} \cdot \mathbf{s} = 0$. Orthogonal vectors are well known in physical space; we will also encounter them in abstract vector space.

We consider first a hypothetical application with one data vector $\mathbf{d}$ and two fitting vectors $\mathbf{f}_1$ and $\mathbf{f}_2$. Each fitting vector is also known as a "**regressor**." Our first task is to approximate the data vector $\mathbf{d}$ by a scaled combination of the two regressor vectors. The scale factors x_1 and x_2 should be chosen so that the model matches the data, i.e.,

$$\mathbf{d} \quad \approx \quad \mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 \tag{1.6}$$

Notice that we could take the partial derivative of the data in (1.6) with respect to an unknown, say x_1, and the result is the regressor $\mathbf{f}_1$.

> The **partial derivative** of *all* data with respect to *any* model parameter gives a **regressor**. A **regressor** is a column in the matrix of partial-derivatives, $\partial d_i/\partial m_j$.

The regression (1.6) is often expressed in the more compact mathematical matrix notation $\mathbf{d} \approx \mathbf{F}\mathbf{x}$, but in our derivation here we will keep track of each component explicitly and use mathematical matrix notation to summarize the final result. Fitting the observed data $\mathbf{d} = \mathbf{d}^{\mathrm{obs}}$ to its two theoretical parts $\mathbf{f}_1 x_1$ and $\mathbf{f}_2 x_2$ can be expressed as minimizing the length of the residual vector $\mathbf{r}$, where

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \mathbf{d}^{\mathrm{theor}} - \mathbf{d}^{\mathrm{obs}} \tag{1.7}$$

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 \; - \; \mathbf{d} \tag{1.8}$$

So we construct a sum of squares (also called a "**quadratic form**") of the components of the residual vector by using a dot product:

$$Q(x_1, x_2) \quad = \quad \mathbf{r} \cdot \mathbf{r} \tag{1.9}$$

$$= \quad (\mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 - \mathbf{d}) \cdot (\mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 - \mathbf{d}) \tag{1.10}$$

The gradient of $Q(x_1, x_2)$ is defined by its two components:

$$\frac{\partial Q}{\partial x_1} \quad = \quad \mathbf{f}_1 \cdot (\mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 - \mathbf{d}) + (\mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 - \mathbf{d}) \cdot \mathbf{f}_1 \tag{1.11}$$

$$\frac{\partial Q}{\partial x_2} \quad = \quad \mathbf{f}_2 \cdot (\mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 - \mathbf{d}) + (\mathbf{f}_1 x_1 + \mathbf{f}_2 x_2 - \mathbf{d}) \cdot \mathbf{f}_2 \tag{1.12}$$

Setting these derivatives to zero and using $(\mathbf{f}_1 \cdot \mathbf{f}_2) = (\mathbf{f}_2 \cdot \mathbf{f}_1)$ etc., we get

$$(\mathbf{f}_1 \cdot \mathbf{d}) \quad = \quad (\mathbf{f}_1 \cdot \mathbf{f}_1) x_1 + (\mathbf{f}_1 \cdot \mathbf{f}_2) x_2 \tag{1.13}$$

$$(\mathbf{f}_2 \cdot \mathbf{d}) \quad = \quad (\mathbf{f}_2 \cdot \mathbf{f}_1) x_1 + (\mathbf{f}_2 \cdot \mathbf{f}_2) x_2 \tag{1.14}$$

which two equations we can use to solve for the two unknowns x_1 and x_2. Writing this expression in matrix notation, we have

$$\begin{bmatrix} (\mathbf{f}_1 \cdot \mathbf{d}) \\ (\mathbf{f}_2 \cdot \mathbf{d}) \end{bmatrix} \quad = \quad \begin{bmatrix} (\mathbf{f}_1 \cdot \mathbf{f}_1) & (\mathbf{f}_1 \cdot \mathbf{f}_2) \\ (\mathbf{f}_2 \cdot \mathbf{f}_1) & (\mathbf{f}_2 \cdot \mathbf{f}_2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{1.15}$$

It is customary to use matrix notation without dot products. For this we need some additional definitions. To clarify these definitions, I choose the number of components in the vectors $\mathbf{f}_1$, $\mathbf{f}_2$, and $\mathbf{d}$ to be three. Thus I can explicitly write a matrix $\mathbf{F}$ in full as

$$\mathbf{F} \quad = \quad [\mathbf{f}_1 \quad \mathbf{f}_2] \quad = \quad \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \\ f_{31} & f_{32} \end{bmatrix} \tag{1.16}$$

Likewise, the *transposed* matrix $\mathbf{F}'$ is defined by

$$\mathbf{F}' \quad = \quad \begin{bmatrix} f_{11} & f_{21} & f_{31} \\ f_{12} & f_{22} & f_{32} \end{bmatrix} \tag{1.17}$$

The matrix in equation (1.15) contains dot products. Matrix multiplication is an abstract way of representing the dot products:

$$\begin{bmatrix} (\mathbf{f}_1 \cdot \mathbf{f}_1) & (\mathbf{f}_1 \cdot \mathbf{f}_2) \\ (\mathbf{f}_2 \cdot \mathbf{f}_1) & (\mathbf{f}_2 \cdot \mathbf{f}_2) \end{bmatrix} \quad = \quad \begin{bmatrix} f_{11} & f_{21} & f_{31} \\ f_{12} & f_{22} & f_{32} \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \\ f_{31} & f_{32} \end{bmatrix} \tag{1.18}$$

Thus, equation (1.15) without dot products is

$$\begin{bmatrix} f_{11} & f_{21} & f_{31} \\ f_{12} & f_{22} & f_{32} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \quad = \quad \begin{bmatrix} f_{11} & f_{21} & f_{31} \\ f_{12} & f_{22} & f_{32} \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \\ f_{31} & f_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{1.19}$$

which has the matrix abbreviation

$$\mathbf{F}'\mathbf{d} \quad = \quad (\mathbf{F}'\,\mathbf{F})\mathbf{x} \tag{1.20}$$

Equation (1.20) is the classic result of least-squares fitting of data to a collection of regressors. Obviously, the same matrix form applies when there are more than two regressors and each vector has more than three components. Equation (1.20) leads to an **analytic solution** for $\mathbf{x}$ using an inverse matrix. To solve formally for the unknown $\mathbf{x}$, we premultiply by the inverse matrix $(\mathbf{F}'\,\mathbf{F})^{-1}$:

$$\mathbf{x} \quad = \quad (\mathbf{F}'\,\mathbf{F})^{-1}\,\mathbf{F}'\mathbf{d} \tag{1.21}$$

Equation (1.21) is the central result of **least-squares** theory. We see it everywhere.

In our first manipulation of matrix algebra, we move around some parentheses in (1.20):

$$\mathbf{F}'\mathbf{d} \quad = \quad \mathbf{F}'\,(\mathbf{F}\mathbf{x}) \tag{1.22}$$

Moving the parentheses implies a regrouping of terms or a reordering of a computation. You can verify the validity of moving the parentheses by writing (1.22) in full as the set of two equations it represents. Equation (1.20) led to the "analytic" solution (1.21). In a later section on conjugate gradients, we will see that equation (1.22) expresses better than (1.21) the philosophy of iterative methods.

Notice how equation (1.22) invites us to cancel the matrix $\mathbf{F}'$ from each side. We cannot do that of course, because $\mathbf{F}'$ is not a number, nor is it a square matrix with an inverse. If you really want to cancel the matrix $\mathbf{F}'$, you may, but the equation is then only an approximation that restates our original goal (1.6):

$$\mathbf{d} \quad \approx \quad \mathbf{F}\mathbf{x} \tag{1.23}$$

A speedy problem solver might ignore the mathematics covering the previous page, study his or her application until he or she is able to write the **statement of wishes**

(1.23) = (1.6), premultiply by $\mathbf{F}'$, replace $\approx$ by $=$, getting (1.20), and take (1.20) to a simultaneous equation-solving program to get $\mathbf{x}$.

The formal literature does not speak of "statement of wishes" but of "**regression**," which is the same concept. In a regression, there is an abstract vector called the residual $\mathbf{r} = \mathbf{F}\mathbf{x} - \mathbf{d}$ whose components should all be small. Formally this is often written as:

$$\min_{\mathbf{x}} \quad ||\mathbf{F}\mathbf{x} - \mathbf{d}|| \tag{1.24}$$

The notation above with two pairs of vertical lines looks like double absolute value, but we can understand it as a reminder to square and sum all the components. This formal notation is more explicit about what is being minimized and what is being varied, but I often find myself sketching out applications in the form of a "statement of wishes," which I call the "**regression**."

1.2.2 Normal equations

An important concept is that when energy is minimum, the residual is orthogonal to the fitting functions. The fitting functions are the column vectors $\mathbf{f}_1$, $\mathbf{f}_2$, and $\mathbf{f}_3$. Let us verify only that the dot product $\mathbf{r} \cdot \mathbf{f}_2$ vanishes by showing that those two vectors are orthogonal. Energy minimum is found by

$$0 \quad = \quad \frac{\partial}{\partial x_2} \mathbf{r} \cdot \mathbf{r} \quad = \quad 2 \, \frac{\partial \mathbf{r}}{\partial x_2} \cdot \mathbf{r} \quad = \quad 2 \, \mathbf{f}_2 \cdot \mathbf{r} \tag{1.25}$$

(To compute the derivative refer to equation (1.8).) Equation (1.25) shows that the residual is orthogonal to the fitting function.

The basic least-squares equations are often called the "**normal**" equations. The word "normal" means perpendicular. We can rewrite equation (1.22) to emphasize the perpendicularity. Bring both terms to the left, and recall the definition of the residual $\mathbf{r}$ from equation (1.8):

$$\begin{aligned} \mathbf{F}'(\mathbf{F}\mathbf{x} - \mathbf{d}) &= \mathbf{0} &\quad (1.26) \\ \mathbf{F}'\mathbf{r} &= \mathbf{0} &\quad (1.27) \end{aligned}$$

Equation (1.27) says that the **residual** vector $\mathbf{r}$ is perpendicular to each row in the $\mathbf{F}'$ matrix. These rows are the **fitting functions**. Therefore, the residual, after it has been minimized, is perpendicular to *all* the fitting functions.

1.2.3 Differentiation by a complex vector

Complex numbers frequently arise in physical problems, particularly with Fourier series. Let us extend the multivariable least-squares theory to the use of complex-valued unknowns $\mathbf{x}$. First recall how complex numbers were handled with single-variable least squares, i.e., as in the discussion leading up to equation (1.5). Use prime, such as $\mathbf{x}'$, to denote the complex conjugate of the transposed vector $\mathbf{x}$. Now write the positive **quadratic form** as

$$Q(\mathbf{x}',\mathbf{x}) \quad = \quad (\mathbf{F}\mathbf{x}-\mathbf{d})'(\mathbf{F}\mathbf{x}-\mathbf{d}) \quad = \quad (\mathbf{x}'\mathbf{F}'-\mathbf{d}')(\mathbf{F}\mathbf{x}-\mathbf{d}) \tag{1.28}$$

After equation (1.4), we minimized a quadratic form $Q(\bar{X},X)$ by setting to zero both $\partial Q/\partial\bar{X}$ and $\partial Q/\partial X$. We noted that only one of $\partial Q/\partial\bar{X}$ and $\partial Q/\partial X$ is necessary because they are conjugates of each other. Now take the derivative of Q with respect to the (possibly complex, row) vector $\mathbf{x}'$. Notice that $\partial Q/\partial\mathbf{x}'$ is the complex conjugate transpose of $\partial Q/\partial\mathbf{x}$. Thus, setting one to zero sets the other also to zero. Setting $\partial Q/\partial\mathbf{x}'=\mathbf{0}$ gives the normal equations:

$$\mathbf{0} \quad = \quad \frac{\partial Q}{\partial \mathbf{x}'} \quad = \quad \mathbf{F}'(\mathbf{F}\mathbf{x}-\mathbf{d}) \tag{1.29}$$

The result is merely the complex form of our earlier result (1.20). Therefore, differentiating by a complex vector is an abstract concept, but it gives the same set of equations as differentiating by each scalar component, and it saves much clutter.

1.2.4 From the frequency domain to the time domain

Equation (1.4) is a frequency-domain quadratic form that we minimized by varying a single parameter, a Fourier coefficient. Now we will look at the same problem in the time domain. The time domain offers new flexibility with boundary conditions, constraints, and weighting functions. The notation will be that a filter f_t has input x_t and output y_t. In Fourier space this is $Y=XF$. There are two problems to look at, unknown filter F and unknown input X.

Unknown filter

Given inputs and outputs, the problem of finding an unknown filter appears to be overdetermined, so we write $\mathbf{y}\approx\mathbf{X}\mathbf{f}$ where the matrix $\mathbf{X}$ is a matrix of downshifted columns like (??). Thus the quadratic form to be minimized is a restatement of equation (1.28) using filter definitions:

$$Q(\mathbf{f}',\mathbf{f}) \quad = \quad (\mathbf{X}\mathbf{f}-\mathbf{y})'(\mathbf{X}\mathbf{f}-\mathbf{y}) \tag{1.30}$$

The solution $\mathbf{f}$ is found just as we found (1.29), and it is the set of simultaneous equations $\mathbf{0}=\mathbf{X}'(\mathbf{X}\mathbf{f}-\mathbf{y})$.

Unknown input: deconvolution with a known filter

For the unknown input problem we put the known filter f_t in a matrix of downshifted columns $\mathbf{F}$. Our statement of wishes is now to find x_t so that $\mathbf{y}\approx\mathbf{F}\mathbf{x}$. We can expect to have trouble finding unknown filter inputs x_t when we are dealing with certain kinds of filters, such as bandpass filters. If the output is zero in a frequency band, we will never be able to find the input in that band and will need to prevent x_t from diverging there. We do this by the statement that we wish $\mathbf{0} \approx \epsilon\,\mathbf{x}$, where ϵ is a parameter that is small and whose exact size will be chosen by experimentation. Putting both wishes into a single, partitioned matrix equation gives

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad \approx \quad \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} \quad = \quad \begin{bmatrix} \mathbf{F} \\ \epsilon\,\mathbf{I} \end{bmatrix} \mathbf{x} \quad - \quad \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} \tag{1.31}$$

To minimize the residuals $\mathbf{r}_1$ and $\mathbf{r}_2$, we can minimize the scalar $\mathbf{r}'\mathbf{r} = \mathbf{r}'_1\mathbf{r}_1 + \mathbf{r}'_2\mathbf{r}_2$. This is

$$\begin{aligned} Q(\mathbf{x}', \mathbf{x}) &= (\mathbf{F}\mathbf{x} - \mathbf{y})'(\mathbf{F}\mathbf{x} - \mathbf{y}) + \epsilon^2 \mathbf{x}'\mathbf{x} \\ &= (\mathbf{x}'\mathbf{F}' - \mathbf{y}')(\mathbf{F}\mathbf{x} - \mathbf{y}) + \epsilon^2 \mathbf{x}'\mathbf{x} \end{aligned} \tag{1.32}$$

We solved this minimization in the frequency domain (beginning from equation (1.4)).

Formally the solution is found just as with equation (1.29), but this solution looks unappealing in practice because there are so many unknowns and because the problem can be solved much more quickly in the Fourier domain. To motivate ourselves to solve this problem in the time domain, we need either to find an approximate solution method that is much faster, or to discover that constraints or time-variable weighting functions are required in some applications.

1.3 ITERATIVE METHODS

The **solution time** for simultaneous **linear equations** grows cubically with the number of unknowns. There are three regimes for solution; which one is applicable depends on the number of unknowns n. For n three or less, we use analytical methods. We also sometimes use analytical methods on matrices of size 4×4 if the matrix contains many zeros. For $n < 500$ we use exact numerical methods such as **Gauss reduction**. A 1988 vintage workstation solves a 100×100 system in a minute, but a 1000×1000 system requires a week. At around $n = 500$, exact numerical methods must be abandoned and **iterative method**s must be used.

An example of a geophysical problem with $n > 1000$ is a missing seismogram. Deciding how to handle a missing seismogram may at first seem like a question of missing *data*, not excess numbers of *model* points. In fitting wave-field data to a consistent model, however, the missing data is seen to be just more unknowns. In real life we generally have not one missing seismogram, but many. Theory in 2-D requires that seismograms be collected along an infinite line. Since any data-collection activity has a start and an end, however, practical analysis must choose between falsely asserting zero data values where data was not collected, or implicitly determining values for unrecorded data at the ends of a survey.

A numerical technique known as the "conjugate-gradient method" (**CG**) works well for all values of n and is our subject here. As with most simultaneous equation solvers, an exact answer (assuming exact arithmetic) is attained in a finite number of steps. And if n is too large to allow n^3 computations, the CG method can be interrupted at any stage, the partial result often proving useful. Whether or not a partial result actually is useful is the subject of much research; naturally, the results vary from one application to the next.

1.3.1 Sign convention

On the last day of the survey, a storm blew up, the sea got rough, the receivers drifted further downwind. The data recorded that day had a larger than usual difference from that predicted by the final model. We could call $(\mathbf{d} - \mathbf{F}\mathbf{m})$ the *experimental* error. (Here $\mathbf{d}$ is data, $\mathbf{m}$ is model parameters, and $\mathbf{F}$ is their linear relation).

The alternate view is that our theory was too simple. It lacked model parameters for the waves and the drifting cables. Because of this model oversimplification we had a *modeling* error of the opposite polarity $(\mathbf{F}\mathbf{m} - \mathbf{d})$.

A strong experimentalist prefers to think of the error as experimental error, something for him or her to work out. Likewise a strong analyst likes to think of the error as a theoretical problem. (Weaker investigators might be inclined to take the opposite view.)

Regardless of the above, and opposite to common practice, I define the **sign convention** for the error (or residual) as $(\mathbf{F}\mathbf{m} - \mathbf{d})$. Choosing this sign convention we will make fewer computational errors because $\mathbf{F}$ is often complicated and formed by combining many parts. So in this book we see positive signs on operators and we see residuals initialized by the negative of the data, often with subroutine `negcopy()` (page 9).

```
subroutine negcopy( n, xx, yy)
integer i, n;   real xx(n), yy(n)
do i= 1, n
        yy(i) = - xx(i)
return; end
```

1.3.2 Method of random directions and steepest descent

Let us minimize the sum of the squares of the components of the **residual** vector given by

$$\text{residual} \quad = \quad \text{transform} \quad \text{model space} \quad - \quad \text{data space} \tag{1.33}$$

$$\begin{bmatrix} \\ \mathbf{r} \\ \\ \end{bmatrix} = \begin{bmatrix} & & \\ & \mathbf{F} & \\ & & \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} - \begin{bmatrix} \\ \mathbf{d} \\ \\ \end{bmatrix} \tag{1.34}$$

A **contour** plot is based on an altitude function of space. The altitude is the **dot product** $\mathbf{r}\cdot\mathbf{r}$. By finding the lowest altitude we are driving the residual vector $\mathbf{r}$ as close as we can to zero. If the residual vector $\mathbf{r}$ reaches zero, then we have solved the simultaneous equations $\mathbf{d} = \mathbf{F}\mathbf{x}$. In a two-dimensional world the vector $\mathbf{x}$ has two components, (x_1, x_2). A contour is a curve of constant $\mathbf{r}\cdot\mathbf{r}$ in (x_1, x_2)-space. These contours have a statistical interpretation as contours of uncertainty in (x_1, x_2), given measurement errors in $\mathbf{d}$.

Let us see how a random search direction can be used to reduce the residual $\mathbf{r} = \mathbf{F}\mathbf{x} - \mathbf{d}$. Let $\Delta\mathbf{x}$ be an abstract vector with the same number of components as the solution $\mathbf{x}$, and let $\Delta\mathbf{x}$ contain arbitrary or random numbers. We add an unknown quantity α of vector $\Delta\mathbf{x}$ to the vector $\mathbf{x}$, thereby creating $\mathbf{x}_{\rm new}$

$$\mathbf{x}_{\rm new} \quad = \quad \mathbf{x} + \alpha\Delta\mathbf{x} \tag{1.35}$$

This gives a new residual:

$$\begin{aligned}
\mathbf{r}_{\rm new} &= \mathbf{F}\ \mathbf{x}_{\rm new} - \mathbf{d} && (1.36)\\
\mathbf{r} + \alpha\mathbf{F}\Delta\mathbf{x} &= \mathbf{F}(\mathbf{x} + \alpha\Delta\mathbf{x}) - \mathbf{d} && (1.37)\\
\mathbf{r}_{\rm new} \quad = \quad \mathbf{r} + \alpha\Delta\mathbf{r} &= \mathbf{F}\mathbf{x} - \mathbf{d} + \alpha\mathbf{F}\Delta\mathbf{x} && (1.38)\\
\mathbf{r}_{\rm new} &= \mathbf{r} + \alpha\mathbf{F}\Delta\mathbf{x} && (1.39)
\end{aligned}$$

which defines $\Delta\mathbf{r} = \mathbf{F}\Delta\mathbf{x}$.

Next we adjust α to minimize the dot product $\mathbf{r}_{\rm new}\cdot\mathbf{r}_{\rm new}$

$$(\mathbf{r} + \alpha\Delta\mathbf{r})\cdot(\mathbf{r} + \alpha\Delta\mathbf{r}) \quad = \quad \mathbf{r}\cdot\mathbf{r} + 2\alpha(\mathbf{r}\cdot\Delta\mathbf{r}) \ + \ \alpha^2\Delta\mathbf{r}\cdot\Delta\mathbf{r} \tag{1.40}$$

by setting to zero its derivative with respect to α getting

$$\alpha \quad = \quad -\,\frac{(\mathbf{r}\cdot\Delta\mathbf{r})}{(\Delta\mathbf{r}\cdot\Delta\mathbf{r})} \tag{1.41}$$

Geometrically and algebraically the new residual $\mathbf{r}_{\rm new} = \mathbf{r} + \alpha\Delta\mathbf{r}$ is perpendicular to the "fitting function" $\Delta\mathbf{r}$. (You can confirm this by substitution leading to $\mathbf{r}_{\rm new}\cdot\Delta\mathbf{r} = 0$.)

A "computation **template**" for the method of random directions is

$$\begin{array}{l}
\mathbf{r} \quad \longleftarrow \quad \mathbf{F}\mathbf{x} - \mathbf{d} \\
\text{iterate } \{ \\
\qquad \begin{array}{lcl}
\Delta\mathbf{x} & \longleftarrow & \text{random numbers} \\
\Delta\mathbf{r} & \longleftarrow & \mathbf{F}\ \Delta\mathbf{x} \\
\alpha & \longleftarrow & -(\mathbf{r}\cdot\Delta\mathbf{r})/(\Delta\mathbf{r}\cdot\Delta\mathbf{r}) \\
\mathbf{x} & \longleftarrow & \mathbf{x} + \alpha\Delta\mathbf{x} \\
\mathbf{r} & \longleftarrow & \mathbf{r}\ + \alpha\Delta\mathbf{r} \\
\}
\end{array}
\end{array}$$

A nice thing about the method of random directions is that you do not need to know the adjoint operator $\mathbf{F}'$.

In practice, random directions are rarely used. It is more common to use the **gradient** direction than a random direction. Notice a vector of the size of $\Delta\mathbf{x}$ is

$$\mathbf{g} \quad = \quad \mathbf{F}'\mathbf{r} \tag{1.42}$$

Notice also that this vector can be found by taking the gradient of the size of the residuals:

$$\frac{\partial}{\partial \mathbf{x}'}\, \mathbf{r}\cdot\mathbf{r} \quad = \quad \frac{\partial}{\partial \mathbf{x}'}\, (\mathbf{x}'\,\mathbf{F}' \;-\; \mathbf{d}')\,(\mathbf{F}\,\mathbf{x} \;-\; \mathbf{d}) \quad = \quad \mathbf{F}'\,\mathbf{r} \tag{1.43}$$

Choosing $\Delta\mathbf{x}$ to be the gradient vector $\Delta\mathbf{x} = \mathbf{g} = \mathbf{F}'\mathbf{r}$ is called "the method of **steepest descent**."

Starting from a model $\mathbf{x} = \mathbf{m}$ (which may be zero), below is a **template** of pseudocode for minimizing the residual $\mathbf{0} \approx \mathbf{r} = \mathbf{F}\mathbf{x} - \mathbf{d}$ by the steepest-descent method:

$$
\begin{array}{lll}
\mathbf{r} & \longleftarrow & \mathbf{F}\mathbf{x} - \mathbf{d} \\
\text{iterate } \{ & & \\
\quad \Delta\mathbf{x} & \longleftarrow & \mathbf{F}'\,\mathbf{r} \\
\quad \Delta\mathbf{r} & \longleftarrow & \mathbf{F}\,\Delta\mathbf{x} \\
\quad \alpha & \longleftarrow & -(\mathbf{r}\cdot\Delta\mathbf{r})/(\Delta\mathbf{r}\cdot\Delta\mathbf{r}) \\
\quad \mathbf{x} & \longleftarrow & \mathbf{x} + \alpha\Delta\mathbf{x} \\
\quad \mathbf{r} & \longleftarrow & \mathbf{r} + \alpha\Delta\mathbf{r} \\
\quad \} & &
\end{array}
$$

1.3.3 Null space and gradient methods

In some applications, while fitting a regression $\mathbf{d} \approx \mathbf{F}\mathbf{x}$ there exists a vector (or a family of vectors) defined by the condition $\mathbf{0} = \mathbf{F}\mathbf{x}_{\text{null}}$. This family is called a **null space**. For example, if the operator $\mathbf{F}$ is a time derivative, then the null space is the constant function; if the operator is a second derivative, then the null space has two components, a constant function and a linear function, or combinations of them. The null space is a family of model components that have no effect on the data.

In the steepest descent method we iteratively find solutions by this updating

$$\mathbf{x}_{i+1} \quad = \quad \mathbf{x}_i + \alpha\Delta\mathbf{x} \tag{1.44}$$

$$\mathbf{x}_{i+1} \quad = \quad \mathbf{x}_i + \alpha\mathbf{F}'\mathbf{r} \tag{1.45}$$

$$\mathbf{x}_{i+1} \quad = \quad \mathbf{x}_i + \alpha\mathbf{F}'(\mathbf{F}\mathbf{x} - \mathbf{d}) \tag{1.46}$$

After we have iterated to convergence, the gradient $\Delta\mathbf{x}$ vanishes as does $\mathbf{F}'(\mathbf{F}\mathbf{x} - \mathbf{d})$. Suppose we now change $\mathbf{x}$ by adding a huge amount of $\mathbf{x}_{\text{null}}$ and continue iterating. Notice that $\Delta\mathbf{x}$ remains zero because $\mathbf{F}\mathbf{x}_{\text{null}}$ vanishes. Thus we conclude that any null space in the initial guess $\mathbf{x}_0$ will remain there unaffected by the gradient descent process.

> A practical way to learn about the existence of null spaces and their general appearance is simply to try gradient descent methods beginning from various different starting guesses.

1.3.4 Why steepest descent is so slow

Before we can understand why the **conjugate-gradient method** is so fast, we need to see why the **steepest-descent method** is so slow. The process of selecting α is called "**line search**," but for a linear problem like the one we have chosen here, we hardly recognize choosing α as searching a line. A more graphic understanding of the whole process is possible in a two-dimensional space where the vector of unknowns $\mathbf{x}$ has just two components, x_1 and x_2. Then the size of the residual vector $\mathbf{r} \cdot \mathbf{r}$ can be displayed with a contour plot in the plane of (x_1, x_2). Visualize a contour map of a mountainous terrain. The gradient is perpendicular to the contours. Contours and gradients are *curved lines*. In the steepest-descent method we start at a point and compute the gradient direction at that point. Then we begin a *straight-line* descent in that direction. The gradient direction curves away from our direction of travel, but we continue on our straight line until we have stopped descending and are about to ascend. There we stop, compute another gradient vector, turn in that direction, and descend along a new straight line. The process repeats until we get to the bottom, or until we get tired.

What could be wrong with such a direct strategy? The difficulty is at the stopping locations. These occur where the descent direction becomes *parallel* to the contour lines. (There the path becomes level.) So after each stop, we turn 90°, from parallel to perpendicular to the local contour line for the next descent. What if the final goal is at a 45° angle to our path? A 45° turn cannot be made. Instead of moving like a rain drop down the centerline of a rain gutter, we move along a fine-toothed zigzag path, crossing and recrossing the centerline. The gentler the slope of the rain gutter, the finer the teeth on the zigzag path.

1.3.5 Conjugate gradient

In the **conjugate-gradient method**, not a line, but rather a plane, is searched. A plane is made from an arbitrary linear combination of two vectors. One vector will be chosen to be the gradient vector, say $\mathbf{g}$. The other vector will be chosen to be the previous descent step vector, say $\mathbf{s} = \mathbf{x}_j - \mathbf{x}_{j-1}$. Instead of $\alpha\,\mathbf{g}$ we need a linear combination, say $\alpha\mathbf{g} + \beta\mathbf{s}$. For minimizing quadratic functions the plane search requires only the solution of a two-by-two set of linear equations for α and β. The equations will be specified here along with the program. (For *nonquadratic* functions a plane search is considered intractable, whereas a line search proceeds by bisection.)

> The simple form of the CG algorithm covered here is a sequence of steps. In each step the minimum is found in the plane given by two vectors: the gradient vector and the vector of the previous step.

1.3.6 Magic

Some properties of the conjugate-gradient approach are well known but hard to explain. D. G. **Luenberger**'s book, *Introduction to Linear and Nonlinear Programming*, is a good place to look for formal explanations of this magic. (His book also provides other forms of the conjugate-gradient algorithm.) Another helpful book is **Strang**'s *Introduction to Applied Mathematics*. Known properties follow:

1. The **conjugate-gradient method** gets the exact answer (assuming exact arithmetic) in n descent steps (or less), where n is the number of unknowns.
2. Since it is helpful to use the previous step, you might wonder why not use the previous two steps, since it is not hard to solve a three-by-three set of simultaneous linear equations. It turns out that the third direction does not help: the distance moved in the extra direction is zero.

1.3.7 Conjugate-gradient theory for programmers

Fourier-transformed variables are often capitalized. This convention will be helpful here, so in this subsection only, we capitalize vectors transformed by the $\mathbf{F}$ matrix. As everywhere, a matrix such as $\mathbf{F}$ is printed in **boldface** type but in this subsection, vectors are *not* printed in boldface print. Thus we define the solution, the solution step (from one iteration to the next), and the gradient by

$$X = \mathbf{F}\, x \qquad \text{solution} \tag{1.47}$$

$$S_j = \mathbf{F}\, s_j \qquad \text{solution step} \tag{1.48}$$

$$G_j = \mathbf{F}\, g_j \qquad \text{solution gradient} \tag{1.49}$$

A linear combination in solution space, say $s+g$, corresponds to $S+G$ in the conjugate space, because $S+G = \mathbf{F}s+\mathbf{F}g = \mathbf{F}(s+g)$. According to equation (1.34), the residual is

$$R = \mathbf{F}x - D = X - D \tag{1.50}$$

The solution x is obtained by a succession of steps s_j, say

$$x = s_1 + s_2 + s_3 + \cdots \tag{1.51}$$

The last stage of each iteration is to update the solution and the residual:

$$\text{solution update :} \qquad x \leftarrow x \quad + s \tag{1.52}$$

$$\text{residual update :} \qquad R \leftarrow R \quad + S \tag{1.53}$$

The *gradient* vector g is a vector with the same number of components as the solution vector x. A vector with this number of components is

$$g \quad = \quad \mathbf{F}' \, R \quad = \quad \text{gradient} \tag{1.54}$$

$$G \quad = \quad \mathbf{F} \, g \quad = \quad \text{conjugate gradient} \tag{1.55}$$

The gradient g in the transformed space is G, also known as the **conjugate gradient**.

The minimization (1.40) is now generalized to scan not only the line with α, but simultaneously another line with β. The combination of the two lines is a plane:

$$Q(\alpha, \beta) \quad = \quad (R + \alpha G + \beta S) \cdot (R + \alpha G + \beta S) \tag{1.56}$$

The minimum is found at $\partial Q / \partial \alpha = 0$ and $\partial Q / \partial \beta = 0$, namely,

$$0 \quad = \quad G \cdot (R + \alpha G + \beta S) \tag{1.57}$$

$$0 \quad = \quad S \cdot (R + \alpha G + \beta S) \tag{1.58}$$

The solution is

$$\left[\begin{array}{c} \alpha \\ \beta \end{array} \right] \quad = \quad \frac{-1}{(G \cdot G)(S \cdot S) - (G \cdot S)^2} \left[\begin{array}{cc} (S \cdot S) & -(S \cdot G) \\ -(G \cdot S) & (G \cdot G) \end{array} \right] \left[\begin{array}{c} (G \cdot R) \\ (S \cdot R) \end{array} \right] \tag{1.59}$$

The many applications in this book all need to find α and β with (1.59) and then update the solution with (1.52) and update the residual with (1.53). Thus we package these activities in a subroutine named `cgplus()` (page 15). To use that subroutine we will have a computation **template** like we had for steepest descents, except that we will have the repetitive work done by subroutine `cgplus()`. This template (or pseudocode) for minimizing the residual $\mathbf{0} \approx \mathbf{r} = \mathbf{F}\mathbf{x} - \mathbf{d}$ by the conjugate-gradient method is

$$
\begin{array}{lll}
\mathbf{r} & \longleftarrow & \mathbf{F}\mathbf{x} - \mathbf{d} \\
\text{iterate } \{ & & \\
& \Delta\mathbf{x} \quad \longleftarrow & \mathbf{F}' \, \mathbf{r} \\
& \Delta\mathbf{r} \quad \longleftarrow & \mathbf{F} \, \Delta\mathbf{x} \\
& (\mathbf{x}, \mathbf{r}) \quad \longleftarrow & \text{cgplus}(\mathbf{x}, \mathbf{r}, \Delta\mathbf{x}, \Delta\mathbf{r}) \\
& \} &
\end{array}
$$

where the subroutine `cgplus()` remembers the previous iteration and works out the step size and adds in the proper proportion of the $\Delta\mathbf{x}$ of the previous step.

Most of the least-squares solver subroutines in this book follow the above **template**. They may look less complicated because they start from $\mathbf{x} = \mathbf{0}$ or more complicated because $\mathbf{F}$ has several parts. $\mathbf{F}$ is often a partitioned matrix of operators and the code for applying it will have subtle differences from the code for applying its adjoint $\mathbf{F}'$. Often we fit two regressions simultaneously, $\mathbf{0} \approx \mathbf{G}\mathbf{x} - \mathbf{d}$ and $\mathbf{0} \approx \mathbf{A}\mathbf{x}$, and then $\mathbf{F}$ is the column matrix

$$\mathbf{F} \quad = \quad \begin{bmatrix} \mathbf{G} \\ \mathbf{A} \end{bmatrix} \tag{1.60}$$

1.3.8 Routine for a step of conjugate gradients

The **conjugate-gradient program** can be divided into two parts: an inner part that is used almost without change over a wide variety of applications, and an outer part containing memory allocations, operator invocations, and initializations. Since **Fortran** does not recognize the difference between upper- and lower-case letters, the conjugate vectors G and S in the program are denoted by `gg` and `ss`. The inner part of the conjugate-gradient task is in subroutine `cgplus()`.

```
# A step of conjugate-gradient descent.
#
subroutine cgplus( iter,   n, x, g, s,   m, rr, gg, ss)
integer i,         iter,   n,            m
real    x(n), rr(m),    # solution, residual
        g(n), gg(m),    # gradient, conjugate gradient
        s(n), ss(m)     # step,     conjugate step
double precision ddot, sds, gdg, gds, determ, gdr, sdr, alfa, beta
if( iter < 1 ) {                    # steepest descent on first iteration.
        do i= 1, n
                s(i) = 0.
        do i= 1, m
                ss(i) = 0.
        if( ddot(m,gg,gg)==0 )  call erexit('cgplus: first grad vanishes identically')
        alfa =  - ddot(m,gg,rr) / ddot(m,gg,gg)
        beta = 0.
        }
else {                              # search plane by solving 2-by-2
        gdg = ddot(m,gg,gg)         #  G . (R + G*alfa + S*beta) = 0
        sds = ddot(m,ss,ss)         #  S . (R + G*alfa + S*beta) = 0
        gds = ddot(m,gg,ss)
        if    ( gdg == 0.)          call erexit('cgplus: grad vanishes identically')
        else if( sds == 0.)   { alfa =  - ddot(m,gg,rr) / ddot(m,gg,gg);  beta=0. }
        else { determ = gdg * sds * dmax1( 1.d0 - (gds/gdg)*(gds/sds), 1.0d-12 )
                gdr = - ddot(m,gg,rr)
                sdr = - ddot(m,ss,rr)
                alfa = ( sds * gdr - gds * sdr ) / determ
                beta = (-gds * gdr + gdg * sdr ) / determ
                }
        }
do i= 1, n                                                # s = model step
            s(i)  = alfa * g(i)  + beta * s(i)
do i= 1, m                                                # ss = conjugate
            ss(i) = alfa * gg(i) + beta * ss(i)
do i= 1, n                                                # update solution
            x(i)  =  x(i) +  s(i)
do i= 1, m                                                # update residual
            rr(i) = rr(i) + ss(i)
return; end
```

```
double precision function ddot( n, x, y )
integer i, n;           real  x(n), y(n);       double precision val
val = 0.;       do i=1,n  { val = val + x(i) * y(i) }
ddot = val;     return;         end
```

Now we assemble a subroutine cgmeth() (page 16) for solving simultaneous equations. Starting with the conjugate-gradient solver template on page 14 we insert the subroutine matmult() (page ??) as the linear operator.

```
# setup of conjugate gradient descent, minimize  SUM rr(i)**2
#                     nx
# rr(i)  = yy(i) - sum fff(i,j) * x(j)
#                     j=1
subroutine cgmeth( nx,x, nr,yy,rr, fff, niter)
integer i, iter,   nx,   nr,            niter
real    x(nx), yy(nr), rr(nr), fff(nr,nx)
temporary real dx(nx), sx(nx), dr(nr), sr(nr)
do i= 1, nx
        x(i) = 0.
do i= 1, nr
        rr(i) = - yy(i)
do iter= 0, niter {
        call matmult( 1, 0, fff, dx,nx, rr,nr)          # dx= dx(fff,rr)
        call matmult( 0, 0, fff, dx,nx, dr,nr)          # dr= dr(fff,dx)
        call cgplus( iter, nx, x,dx,sx,
                           nr,rr,dr,sr)                 # x=x+s; rr=rr+ss
        }
return; end
```

Below shows the solution to 5×4 set of simultaneous equations. Observe that the "exact" solution is obtained in the last step. Since the data and answers are integers, it is quick to check the result manually.

```
 d transpose
     3.00      3.00      5.00      7.00      9.00

 F transpose
     1.00      1.00      1.00      1.00      1.00
     1.00      2.00      3.00      4.00      5.00
     1.00      0.00      1.00      0.00      1.00
     0.00      0.00      0.00      1.00      1.00

for iter = 0, 4
x    0.43457383  1.56124675  0.27362058  0.25752524
res -0.73055887  0.55706739  0.39193487 -0.06291389 -0.22804642
x    0.51313990  1.38677299  0.87905121  0.56870615
res -0.22103602  0.28668585  0.55251014 -0.37106210 -0.10523783
x    0.39144871  1.24044561  1.08974111  1.46199656
res -0.27836466 -0.12766013  0.20252672 -0.18477242  0.14541438
x    1.00001287  1.00004792  1.00000811  2.00000739
res  0.00006878  0.00010860  0.00016473  0.00021179  0.00026788
x    1.00000024  0.99999994  0.99999994  2.00000024
res -0.00000001 -0.00000001  0.00000001  0.00000002 -0.00000001
```

1.4 INVERSE NMO STACK

To illustrate an example of solving a huge set of simultaneous equations without ever writing down the matrix of coefficients we consider how **back projection** can be upgraded towards **inversion** in the application called moveout and stack.

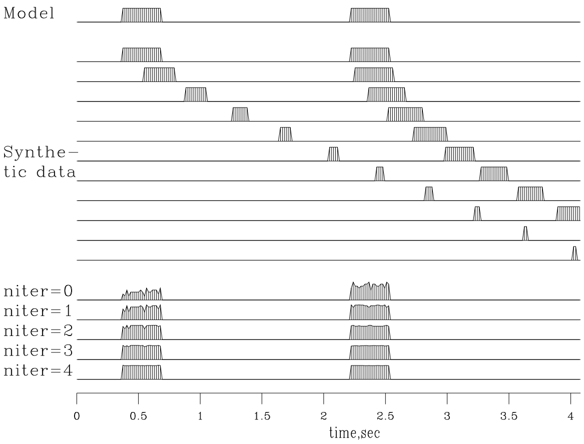

Figure 1.1: Top is a model trace $\mathbf{m}$. Next are the synthetic data traces, $\mathbf{d} = \mathbf{Mm}$. Then, labeled `niter=0` is the **stack**, a result of processing by adjoint modeling. Increasing values of `niter` show $\mathbf{x}$ as a function of iteration count in the regression $\mathbf{d} \approx \mathbf{Mm}$. (Carlos Cunha-Filho) $\boxed{\text{lsq-invstack}}$ [NR]

The seismograms at the bottom of Figure 1.1 show the first four iterations of conjugate-gradient inversion. You see the original rectangle-shaped waveform returning as the iterations proceed. Notice also on the **stack** that the early and late events have unequal amplitudes, but after iteration they are equal, as they began. Mathematically, we can denote the top trace as the model $\mathbf{m}$, the synthetic data signals as $\mathbf{d} = \mathbf{Mm}$, and the stack as $\mathbf{M}'\mathbf{d}$. The conjugate-gradient algorithm solves the regression $\mathbf{d} \approx \mathbf{Mx}$ by variation of $\mathbf{x}$, and the figure shows $\mathbf{x}$ converging to $\mathbf{m}$. Since there are 256 unknowns in $\mathbf{m}$, it is gratifying to see good convergence occurring after the first four iterations. The CG subroutine used is `invstack()`, which is just like `cgmeth()` (page 16) except that the matrix-multiplication operator `matmult()` (page ??) has been replaced by `imospray()` (page ??). Studying the program, you can deduce that, except for a scale factor, the output at `niter=0` is identical to the stack $\mathbf{M}'\mathbf{d}$. All the signals in Figure 1.1 are intrinsically the same scale.

```
# NMO stack by inverse of forward modeling
#
subroutine invstack(  nt,model,nx,gather,rr,t0,x0,dt,dx,slow,niter)
integer it, ix, iter, nt,      nx,                           niter
real    t0,x0,dt,dx,slow, gather(nt,nx), rr(nt,nx), model(nt)
temporary real  dmodel(nt), smodel(nt), dr(nt,nx), sr(nt,nx)
do it= 1, nt
        model(it) = 0.0
do it= 1, nt
        do ix= 1, nx
                rr(it,ix) = - gather(it,ix)
do iter = 0, niter {
        call imospray( 1,0,slow,x0,dx,t0,dt,nx,nt,dmodel,rr)    # nmo-stack
        call imospray( 0,0,slow,x0,dx,t0,dt,nx,nt,dmodel,dr)    # modeling
        call cgplus(iter, nt, model, dmodel, smodel, nt*nx, rr, dr, sr)
        }
return; end
```

This simple inversion is inexpensive. Has anything been gained over conventional stack? First, though we used **nearest-neighbor** interpolation, we managed to preserve the spectrum of the input, apparently all the way to the Nyquist frequency. Second, we preserved the true amplitude scale without ever bothering to think about (1) dividing by the number of contributing traces, (2) the amplitude effect of NMO stretch, or (3) event truncation.

With depth-dependent velocity, wave fields become much more complex at wide offset. NMO soon fails, but wave-equation forward modeling offers interesting opportunities for inversion.

1.5 THE WORLD OF CONJUGATE GRADIENTS

1.5.1 Physical nonlinearity

Nonlinearity arises two ways: First, theoretical data might be a nonlinear function of the model parameters. Second, observed data could contain imperfections that force us to use nonlinear methods of statistical estimation.

Standard methods of physics relate theoretical data $\mathbf{d}_{\mathrm{theor}}$ to model parameters $\mathbf{m}$, often with a nonlinear relation, say $\mathbf{d}_{\mathrm{theor}} = \mathbf{f}(\mathbf{m})$. The power-series approach then leads to representing theoretical data as

$$\mathbf{d}_{\mathrm{theor}} \quad = \quad \mathbf{f}(\mathbf{m}_0 + \Delta\mathbf{m}) \quad \approx \quad \mathbf{f}(\mathbf{m}_0) + \mathbf{F}\Delta\mathbf{m} \tag{1.61}$$

where $\mathbf{F}$ is the matrix of partial derivatives of data values by model parameters, say $\partial d_i / \partial m_j$, evaluated at $\mathbf{m}_0$. The theoretical data $\mathbf{d}_{\mathrm{theor}}$ minus the observed data $\mathbf{d}_{\mathrm{obs}}$ is the residual we minimize.

$$\mathbf{0} \quad \approx \quad \mathbf{d}_{\mathrm{theor}} - \mathbf{d}_{\mathrm{obs}} \quad = \quad \mathbf{F}\boldsymbol{\Delta}\mathbf{m} + [\mathbf{f}(\mathbf{m}_0) - \mathbf{d}_{\mathrm{obs}}] \tag{1.62}$$

$$\mathbf{r}_{\mathrm{new}} \quad = \quad \mathbf{F}\boldsymbol{\Delta}\mathbf{m} + \mathbf{r}_{\mathrm{old}} \tag{1.63}$$

It is worth noticing that the residual updating (1.63) in a nonlinear problem is the same as in a linear problem updating (1.39).

The pathway of inversion with physical nonlinearity is well developed in the academic literature and Bill **Symes** at Rice University has a particularly active group.

1.5.2 Statistical nonlinearity

The data itself often has noise bursts or gaps, and we will see that this leads us to readjusting the weighting function. In principle, we should fix the weighting function and solve the problem. Then we should revise the weighting function and solve the problem again. In practice we find it convenient to change the weighting function during the optimization descent. There are some pitfalls if the weighting function is changed too rapidly or drastically. I am hoping this form of nonlinearity does not lie outside the scope of Optimization Theory (OT). Perhaps it does, and we will never be able to relegate our problems to a professional nonlinear solver, but must instead create our own mechanisms for guiding regressions towards the best model.

1.5.3 Coding nonlinear regressions

None of us is an expert in both Geophysics and in OT yet we need to handle both. It is particularly difficult to put the two fields together with Fortran. We would like to have each group write its own code with a relatively easy interface. The problem is that the OT codes must invoke the physical operators yet the OT codes should not need to deal with all the data and parameters needed by the physical operators.

In this book I continue to use the book3 CG method because (given the constraint of a Fortran framework) I find it the most practical way to isolate features of geophysical analysis from details of OT. We can solve nonlinear least-squares problems about the same way as iteratively reweighted ones. A simple adaptation of the book3 method, (moving the "`do iter=0,niter`" line above the residual initialization) gives us a nonlinear solver. Omitting the weighting function (for simplicity) the **template** is:

$$
\begin{array}{ll}
\text{iterate } \{ & \\
 & \mathbf{r} \quad \longleftarrow \quad \mathbf{f}(\mathbf{m}) - \mathbf{d} \\
 & \text{Define } \mathbf{F} = \partial \mathbf{d} / \partial \mathbf{m}. \\
 & \Delta\mathbf{m} \quad \longleftarrow \quad \mathbf{F}'\,\mathbf{r} \\
 & \Delta\mathbf{r} \quad \longleftarrow \quad \mathbf{F}\,\Delta\mathbf{m} \\
 & (\mathbf{m}, \mathbf{r}) \quad \longleftarrow \quad \text{cgplus}(\mathbf{m}, \mathbf{r}, \Delta\mathbf{m}, \Delta\mathbf{r}) \\
 & \}
\end{array}
$$

A formal theory for the optimization exists, but we are not using it here. The assumption we make is that the step size will be small so familiar line-search and plane-search approximations will succeed in reducing the residual. Unfortunately this assumption is not reliable. What we should do is test that the residual really does decrease, and if it does not we should revert to steepest descent with a smaller step size. I am about to test an incremental variation on the status quo where inside `cgplus()` (page 15) we check to see if the residual diminished in the *previous* step, and if it did not, choose the *current* step to be steepest descent instead of CG. Proper handling in Fortran requires deeper integration of OT and `cgplus()` (page 15) with the application dependent solver. I am not prepared either to drop Fortran or to integrate solvers with application code.

Experience shows that nonlinear problems have many pitfalls. Start with a linear problem, add a minor physical improvement or unnormal noise, and the problem becomes nonlinear and probably has another solution far from anything reasonable. When solving such a nonlinear problem, we cannot arbitrarily begin from zero as we do with linear problems. We must choose a reasonable starting guess, and then move in a stable and controlled manner. Look back to `cgplus()` (page 15) and notice that if you invoke `cgplus()` with "`do iter=-n,n`" it will begin with `n` steps of steepest descent and then switch over to do `n` more steps of CG.

1.5.4 Standard methods

The conjugate-gradient method is really a family of methods. Mathematically these algorithms all converge to the answer in n (or fewer) steps where there are n unknowns. The various methods differ in numerical accuracy, treatment of underdetermined systems, accuracy in treating ill-conditioned systems, space requirements, and numbers of dot products. Technically, the method of CG used in this book is not the conjugate-gradient method itself, but is equivalent to it. The method of this book is more properly called the conjugate-direction method with a memory of one step. I chose this method for its clarity and flexibility. If you would like a free introduction and summary of conjugate-gradient methods, I can recommend *An Introduction to Conjugate Gradient Method Without Agonizing Pain* by Jonathan Shewchuk, which you can download.[1]

I suggest you skip over the remainder of this section and return after you have seen many examples and have developed some expertise, and have some technical problems.

The **conjugate-gradient method** was introduced by **Hestenes** and **Stiefel** in 1952. To read the standard literature and relate it to this book, you should first realize that when I write regressions like

$$0 \quad \approx \quad \mathbf{W}(\mathbf{Fm} - \mathbf{d}) \tag{1.64}$$

$$0 \quad \approx \quad \mathbf{Am}, \tag{1.65}$$

they are equivalent to minimizing the quadratic form:

$$\mathbf{m}: \qquad \min_{\mathbf{m}} Q(\mathbf{m}) \quad = \quad (\mathbf{m}'\mathbf{F}' - \mathbf{d}')\mathbf{W}'\mathbf{W}(\mathbf{Fm} - \mathbf{d}) \; + \; \mathbf{m}'\mathbf{A}'\mathbf{Am} \tag{1.66}$$

The optimization theory (OT) literature starts from a minimization of

$$\mathbf{x}: \qquad \min_{\mathbf{x}} Q(\mathbf{x}) \quad = \quad \mathbf{x}'\mathbf{Hx} - \mathbf{b}'\mathbf{x} \tag{1.67}$$

To relate equation (1.66) to (1.67) we expand the parentheses in (1.66) and abandon the constant term $\mathbf{d}'\mathbf{d}$. Then gather the quadratic term in $\mathbf{m}$ and the linear term in $\mathbf{m}$. There are two terms linear in $\mathbf{m}$ that are transposes of each other. They are scalars so they are equal. Thus, to invoke "standard methods" you take your problem-formulation operators $\mathbf{F}$, $\mathbf{W}$, $\mathbf{A}$ and create two subroutines that apply

$$\mathbf{H} \quad = \quad \mathbf{F}'\mathbf{W}'\mathbf{WF} + \mathbf{A}'\mathbf{A} \tag{1.68}$$

$$\mathbf{b}' \quad = \quad 2(\mathbf{F}'\mathbf{W}'\mathbf{Wd})' \tag{1.69}$$

[1] or http://www.cs.cmu.edu/afs/cs/project/quake/public/papers/painless-conjugate-gradient.ps

The operators **H** and **b** operate on model space. Standard procedures do not require their adjoints because **H** is its own adjoint and **b** reduces model space to a scalar. You can see that computing **H** and **b** requires one temporary space the size of data space (whereas my book3 method requires two).

When people have trouble with CG, I always refer them to the Paige and Saunders algorithm `LSQR`. Methods that form **H** explicitly or implicitly (including both the standard literature and the book3 method) square the condition number, that is, they are twice as susceptible to rounding error as is `LSQR`. The Paige and Saunders method is reviewed by Nolet in a geophysical context, and a brief explanation on how to use the method (without explaining why it works) is found in PVI.

REFERENCES

Gill, P.E., Murray, W., and Wright, M.H., 1981, Practical optimization: Academic Press.

Gorlen, K.E., Orlow, S.M., and Plexico, P.S., 1990, Data abstraction and object-oriented programming in C++: J. Wiley.

Hestenes, M.R., and Stiefel, E., 1952, Methods of conjugate gradients for solving linear systems: J. Res. Natl. Bur. Stand., **49**, 409-436.

Luenberger, D.G., 1973, Introduction to linear and nonlinear programming: Addison-Wesley.

Nolet, G., 1985, Solving or resolving inadequate and noisy tomographic systems: J. Comp. Phys., **61**, 463-482.

Paige, C.C., and Saunders, M.A., 1982a, LSQR: an algorithm for sparse linear equations and sparse least squares: Assn. Comp. Mach. Trans. Mathematical Software, **8,** 43-71.

Paige, C.C., and Saunders, M.A., 1982b, Algorithm 583, LSQR: sparse linear equations and least squares problems: Assn. Comp. Mach. Trans. Mathematical Software, **8,** 195-209.

Press, W.H. et al., 1989, Numerical recipes: the art of scientific computing: Cambridge University Press.

Strang, G., 1986, Introduction to applied mathematics: Wellesley-Cambridge Press.

EXERCISES:

1 One way to remove a mean value m from signal $s(t) = \mathbf{s}$ is with the regession $\mathbf{0} \approx \mathbf{s} - m$. What operator matrix is involved?

2 What linear operator subroutine from chapter **??** can be used for finding the mean?

3 How many CG iterations should be required to get the exact mean value?

4 Write a mathematical expression for finding the mean by the CG method.