Chapter 1

Inverse interpolation and nasty noise

Here we look at many simple applications of the theory in chapters ?? and ??. I wish I could postpone irritating practical details like irregularly located data and ill-mannered noises, but these difficulties arise so commonly that we must begin with them or find ourselves handicapped in most genuine applications until we do. As we dig into the topics of interpolation and noises, we soon find a wealth of important applications, fascinating theory, and some conceptual gaps I cannot yet cross. Here we first look at missing data, then irregularly spaced data, then nasty noises, and finally, anticipating higher-dimensional, computer-intensive problems, and we examine some preconditioning strategies.

1.1 MISSING DATA IN ONE DIMENSION

A method for restoring **missing data** is to ensure that the restored data, after specified filtering, has minimum energy. Specifying the filter chooses the interpolation philosophy. Generally the filter is a **roughening** filter. When a roughening filter goes off the end of smooth data, it typically produces a big end transient. Minimizing energy implies a choice for unknown data values at the end, to minimize the transient. We will examine five cases and then make some generalizations.

> A method for restoring missing data is to ensure that the restored data, after specified filtering, has **minimum energy**.

Let m denote a missing value. The dataset on which the examples are based is $(\cdots, m, m, \ 1, m, 2, 1, 2, m, m, \cdots)$. Using subroutine `mis1()` (page 6), values were found to replace the missing m values so that the power in the filtered data is minimized. Figure 1.1 shows interpolation of the dataset with $(1, -1)$ as a roughening filter. The interpolated data matches the given data where they overlap.

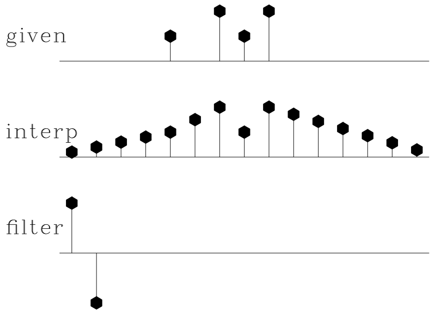


Figure 1.1: Top is given data. Middle is given data with interpolated values. Missing values seem to be interpolated by straight lines. Bottom shows the filter $(1, -1)$, whose output has minimum power. iin-mlines [NR]

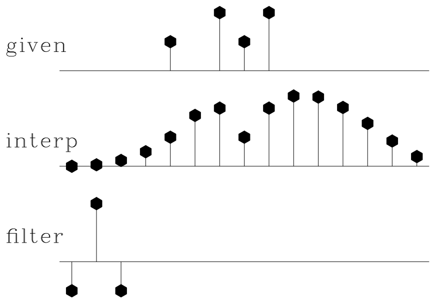


Figure 1.2: Top is the same input data as in Figure 1.1. Middle is interpolated. Bottom shows the filter $(-1, 2, -1)$. The missing data seems to be interpolated by parabolas. iin-mparab [NR]

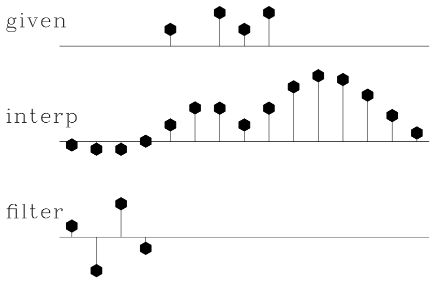


Figure 1.3: Top is the same input. Middle is interpolated. Bottom shows the filter $(1, -3, 3, -1)$. The missing data is very smooth. It shoots upward high off the right end of the observations, apparently to match the data slope there. iin-mseis [NR]

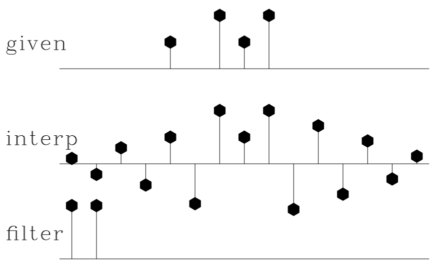


Figure 1.4: Bottom shows the filter $(1, 1)$. The interpolation is rough. Like the given data itself, the interpolation has much energy at the Nyquist frequency. But unlike the given data, it has little zero-frequency energy. iin-moscil [NR]

Figures 1.1–1.4 illustrate that the rougher the filter, the smoother the interpolated data, and vice versa. Let us switch our attention from the residual spectrum to the residual itself. The residual for Figure 1.1 is the *slope* of the signal (because the filter $(1, -1)$ is a *first derivative*), and the slope is constant (uniformly distributed) along the straight lines where the least-squares procedure is choosing signal values. So these examples confirm the idea that the **least-squares method** abhors large values (because they are squared). Thus, least squares tends to distribute residuals uniformly in both time and frequency to the extent the **constraint**s allow.

This idea helps us answer the question, what is the best filter to use? It suggests choosing the filter to have an amplitude spectrum that is inverse to the spectrum we want for the interpolated data. A systematic approach is given in the next section, but I will offer a simple subjective analysis here. Looking at the data, I see that all points are positive. It seems, therefore, that the data is rich in low frequencies; thus the filter should contain something like $(1, -1)$, which vanishes at zero frequency. Likewise, the data seems to contain Nyquist frequency, so the filter should contain $(1, 1)$. The result of using the filter $(1, -1) * (1, 1) = (1, 0, -1)$ is shown in Figure 1.5. This is my best subjective interpolation based on the idea that the missing data should look like the given data. The **interpolation** and **extrapolation**s are so good that you can hardly guess which data values are given and which are interpolated.

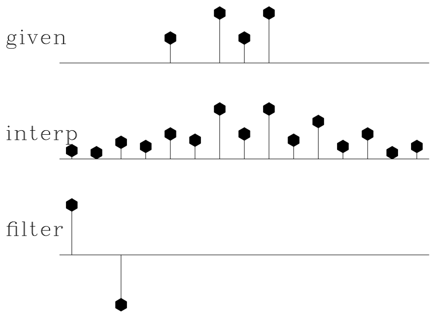


Figure 1.5: Top is the same as in Figures 1.1 to 1.4. Middle is interpolated. Bottom shows the filter $(1, 0, -1)$, which comes from the coefficients of $(1, -1) * (1, 1)$. Both the given data and the interpolated data have significant energy at both zero and Nyquist frequencies. iin-mbest [NR]

1.1.1 Missing-data program

Now let us see how Figures 1.1-1.5 could have been calculated and how they were calculated. They could have been calculated using matrices, pulling matrices apart according to subscripts of known or missing data, instead I computed them using operators, applying only operators and their adjoints. First we inspect the matrix approach because it is more conventional.

Matrix approach to missing data

There are 15 data points in Figures 1.1-1.5. Of the 15, 4 are known and 11 are missing. Denote the known by k and the missing by m. Then the sequence of missing and known is $(m, m, m, m, k, m, k, k, k, m, m, m, m, m, m)$. Since I will have trouble printing 15×15 matrices, please allow me to describe instead a data space of 6 values $(d_1, d_2, d_3, d_4, d_5, d_6)$ with known values only d_2 and d_3, that is arranged like (m, k, k, m, m, m). To minimize the power in the residual, which is the filtered data, we state the regression $\mathbf{0} \approx \mathbf{F}\mathbf{d}$ as

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \approx \quad \mathbf{r} \quad = \quad \begin{bmatrix} f_1 & 0 & 0 & 0 & 0 & 0 \\ f_2 & f_1 & 0 & 0 & 0 & 0 \\ f_3 & f_2 & f_1 & 0 & 0 & 0 \\ 0 & f_3 & f_2 & f_1 & 0 & 0 \\ 0 & 0 & f_3 & f_2 & f_1 & 0 \\ 0 & 0 & 0 & f_3 & f_2 & f_1 \\ 0 & 0 & 0 & 0 & f_3 & f_2 \\ 0 & 0 & 0 & 0 & 0 & f_3 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \tag{1.1}$$

Rearrange the above regression bringing the columns multiplying known data values, d_2 and d_3 to the left getting $\mathbf{y} = -\mathbf{F}_k\mathbf{d}_k \approx \mathbf{F}_m\mathbf{d}_m$.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} \quad = \quad - \begin{bmatrix} 0 & 0 \\ f_1 & 0 \\ f_2 & f_1 \\ f_3 & f_2 \\ 0 & f_3 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} d_2 \\ d_3 \end{bmatrix} \quad \approx \quad \begin{bmatrix} f_1 & 0 & 0 & 0 \\ f_2 & 0 & 0 & 0 \\ f_3 & 0 & 0 & 0 \\ 0 & f_1 & 0 & 0 \\ 0 & f_2 & f_1 & 0 \\ 0 & f_3 & f_2 & f_1 \\ 0 & 0 & f_3 & f_2 \\ 0 & 0 & 0 & f_3 \end{bmatrix} \begin{bmatrix} d_1 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \tag{1.2}$$

which is the familiar form of an overdetermined system of equations $\mathbf{y} \approx \mathbf{F}_m\mathbf{d}_m$ which we could solve for $\mathbf{d}_m$ as illustrated earlier by conjugate gradients, or by a wide variety of well-known methods.

The trouble with this matrix approach is that it is awkward to program the partitioning of the operator into the known and missing parts, particularly if the application of the operator uses arcane techniques, such as those used by the fast Fourier transform operator or various numerical approximations to differential or partial differential operators that depend on regular data sampling. Even for the modest convolution operator, we already have a library of convolution programs that handle a variety of end effects, and it would be much nicer to use the library as it is rather than recode it for all possible geometrical arrangements of missing data values. Here I take the main goal to be the clarity of the code, not the efficiency or accuracy of the solution. (So, if your problem consumes too many resources, and if you have many more known points than missing ones, maybe you should solve $\mathbf{y} \approx \mathbf{F}_m\mathbf{x}$ and ignore what I suggest below.)

Operator approach to missing data

For the operator approach to the regression $-\mathbf{F}_k\mathbf{d}_k \approx \mathbf{F}_m\mathbf{d}_m$ we rewrite it as $-\mathbf{F}_k\mathbf{d}_k \approx \mathbf{F}\mathbf{J}\mathbf{d}$ where

$$
-\mathbf{F}_{\mathrm{k}}\mathbf{d}_{\mathrm{k}} \quad \approx \quad \begin{bmatrix} f_1 & 0 & 0 & 0 & 0 & 0 \\ f_2 & f_1 & 0 & 0 & 0 & 0 \\ f_3 & f_2 & f_1 & 0 & 0 & 0 \\ 0 & f_3 & f_2 & f_1 & 0 & 0 \\ 0 & 0 & f_3 & f_2 & f_1 & 0 \\ 0 & 0 & 0 & f_3 & f_2 & f_1 \\ 0 & 0 & 0 & 0 & f_3 & f_2 \\ 0 & 0 & 0 & 0 & 0 & f_3 \end{bmatrix} \begin{bmatrix} 1 & . & . & . & . & . \\ . & 0 & . & . & . & . \\ . & . & 0 & . & . & . \\ . & . & . & 1 & . & . \\ . & . & . & . & 1 & . \\ . & . & . & . & . & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{bmatrix} \quad = \quad \mathbf{FJd} \tag{1.3}
$$

Notice the introduction of the new diagonal matrix $\mathbf{J}$ that I will call a **mask**ing matrix or a **free-mask** matrix because it multiplies freely adjustable variables by unity and constrained variables by zero. In this new formulation, the whole data space seems freely adjustable, both the missing data values and known values, but we will soon see that the CG method will not be changing the known values. In general, we derive the regression (1.3) by

$$
\begin{aligned}
\mathbf{0} &\approx \mathbf{Fd} && (1.4)\\
\mathbf{0} &\approx \mathbf{F}(\mathbf{J}+(\mathbf{I}-\mathbf{J}))\mathbf{d} && (1.5)\\
\mathbf{0} &\approx \mathbf{FJd}+\mathbf{F}(\mathbf{I}-\mathbf{J})\mathbf{d} && (1.6)\\
\mathbf{0} &\approx \mathbf{FJd}+\mathbf{Fd}_{\mathrm{known}} && (1.7)\\
\mathbf{0} \approx \mathbf{r} &= \mathbf{FJd}+\mathbf{r}_0 && (1.8)
\end{aligned}
$$

To implement this calculation we begin with the known data values where missing data values are replaced by zeros, namely $(\mathbf{I}-\mathbf{J})\mathbf{d}$. Filter this data getting $\mathbf{F}(\mathbf{I}-\mathbf{J})\mathbf{d}$ and load it into the residual $\mathbf{r}_0$. With initialization completed, we begin an iteration loop. First we pass the residual through the adjoint to get suggested data changes

$$
\Delta\mathbf{d} \quad \longleftarrow \quad \mathbf{J}'\mathbf{F}'\mathbf{r} \tag{1.9}
$$

$\mathbf{F}'$ applies a *crosscorrelation* of the filter to the residual and then $\mathbf{J}'$ sets to zero any changes proposed to known data values. Next, compute the change in residual $\Delta\mathbf{r}$ from the proposed change in the data $\Delta\mathbf{d}$.

$$
\Delta\mathbf{r} \quad \longleftarrow \quad \mathbf{FJ}\Delta\mathbf{d} \tag{1.10}
$$

This applies the filtering again. Then use the method of steepest descent (or conjugate gradient) to choose the appropriate scaling (or inclusion of previous step) of $\Delta\mathbf{d}$ and $\Delta\mathbf{r}$ and update $\mathbf{d}$ and $\mathbf{r}$ accordingly and iterate.

In the missing-data subroutine `mis1()` (page 6), the data space $\mathbf{d}$ is in `xx(ix)`. The diagonal of the $\mathbf{J}$ matrix is in the vector `xfree(ix)==1`. Zero-padded known data $(\mathbf{I}-\mathbf{J})\mathbf{d}$ is `(1-xfree(ix))*xx(ix)`. Applying the filtering operator $\mathbf{F}$ with the transient convolution subroutine `tcai1()` (page ??) gives the initial residual $\mathbf{r}_0$. Data revisions should reduce this residual

$$
\mathbf{r} \quad \longleftarrow \quad \mathbf{FJ}\Delta d+\mathbf{r} \tag{1.11}
$$

I could have coded a new operator **FJ** from **F** and **J** but it seems easier to simply apply $\mathbf{J} = \mathbf{J}\mathbf{J}'$ after $\mathbf{F}'$ and before $\mathbf{F}$.

```
# fill in missing data on 1-axis by minimizing power out of a given filter.
#
subroutine mis1( xfree, xx, nx, niter, aa,na)
integer iter, ix, nr,       nx, niter,    na
real    xx(nx)          # known data with zeros for missing values.
real    xfree(nx)       # xfree(i)=0 where data known, elsewhere xfree(i)=1.
real    aa(na)          # input roughening filter
temporary real  xk(nx),dx(nx),sx(nx),  rr(nx+na),dr(nx+na),sr(nx+na)
                                     nr = nx+na
        do ix=1,nx
                xk(ix) = (1. - xfree(ix)) * xx(ix)      # zap the unknown data
        call tcai1(    0, 0, aa,na, xk,nx, rr)  # residual = filtered data
do iter= 0, niter {                             # niter= number missing or less
        call tcai1(    1, 0, aa,na, dx,nx, rr)  # dx(a,r)   correlation
        do ix= 1, nx
                dx(ix) = xfree(ix) * dx(ix)     # cannot change known data.
        call tcai1(    0, 0, aa,na, dx,nx, dr)  # dr=a*dx   convolution
        call cgplus( iter, nx,xx,dx,sx,
                           nr,rr,dr,sr )        # x=x+dx; r=r+dr
        }
return; end
```

EXERCISES:

1 Figure 1.1–1.4 seem to extrapolate to vanishing signals at the side boundaries. Why is that so, and what could be done to leave the sides unconstrained in that way?

2 Show that the interpolation curve in Figure 1.2 is not parabolic as it appears, but cubic. (HINT: First show that $(\nabla^2)'\nabla^2 u = 0$.)

3 Verify by a program example that the number of iterations required with simple constraints is the number of free parameters.

4 A signal on a uniform mesh has missing values. How should we estimate the mean?

1.2 INVERSE LINEAR INTERPOLATION

In chapter ?? we defined **linear interpolation** as extracting values from between mesh points. In a typical setup (occasionally the role of data and model are swapped), a model is given on a uniform mesh and we solve the easy problem of extracting values between the mesh points with subroutine `lint1()` (page ??). The genuine problem is the inverse problem which we attack here. Data values are sprinkled all around, and we wish to find a function on a uniform mesh from which we can extract that data by **linear interpolation**. The adjoint operator for subroutine `lint1()` simply piles data back into its proper location in model space without regard to how many data values land in each region. Thus some model values may have many data points added to them while other model values get none. We could interpolate by minimizing the power in the gradient, or that in the second derivative of the model, or that in the output of any other roughening filter applied to the model.

Formalizing now our wish that data $\mathbf{d}$ be extractable by **linear interpolation** $\mathbf{L}$, from a model $\mathbf{m}$, and our wish that application of a roughening filter with an operator $\mathbf{A}$ have minimum power, we write the regressions:

$$\begin{array}{lll} \mathbf{0} & \approx & \mathbf{Lm} - \mathbf{d} \\ \mathbf{0} & \approx & \mathbf{Am} \end{array} \tag{1.12}$$

Suppose we take the roughening filter to be the second difference operator $(1, -2, 1)$ scaled by a constant ϵ, and suppose we have a data point near each end of the model and a third data point exactly in the middle. Then for a model space 7 points long the regression could look like:

$$\left[\begin{array}{cccccc} .8 & .2 & . & . & . & . \\ . & . & 1 & . & . & . \\ . & . & . & . & .5 & .5 \\ \hline \epsilon & . & . & . & . & . \\ -2\epsilon & \epsilon & . & . & . & . \\ \epsilon & -2\epsilon & \epsilon & . & . & . \\ . & \epsilon & -2\epsilon & \epsilon & . & . \\ . & . & \epsilon & -2\epsilon & \epsilon & . \\ . & . & . & \epsilon & -2\epsilon & \epsilon \\ . & . & . & . & \epsilon & -2\epsilon \\ . & . & . & . & . & \epsilon \\ . & . & . & . & . & . \end{array} \right] \left[\begin{array}{c} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{array} \right] - \left[\begin{array}{c} d_0 \\ d_1 \\ d_2 \\ \hline 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \approx \mathbf{0} \tag{1.13}$$

The residual vector has two parts, a data part r_d on top and a model part r_m on the bottom. The data residual should vanish except where contradictory data values happen to lie in the same place. The model residual is the roughened model.

The subroutine for this job does not initialize the original model. The caller might want to initialize the model with $\mathbf{m} = \mathbf{0}$ or with some other model $\mathbf{m}_0$. First we load the data part of the residual $\mathbf{r}_d = \mathbf{Lm} - \mathbf{d}$ and the model part of the residual $\mathbf{r}_m = \mathbf{Am}$. After the initialization, subroutine `invint1()` (page 8) begins an iteration loop by first computing the proposed model perturbation $\Delta\mathbf{m}$ with the adjoint operator

$$\Delta\mathbf{m} \quad \longleftarrow \quad \left[\begin{array}{cc} \mathbf{L}' & \mathbf{A}' \end{array} \right] \left[\begin{array}{c} \mathbf{r}_d \\ \mathbf{r}_m \end{array} \right] \tag{1.14}$$

I chose to implement the model roughening operator $\mathbf{A}$ with the convolution subroutine `tcai1()` (page ??) which has transient end effects (and an output length equal the input length plus the filter length). The adjoint of subroutine `tcai1()` suggests perturbations in the convolution input (not the filter). Using this value of $\Delta\mathbf{m}$, `invint1()` finds the implied change in residual $\Delta\mathbf{r}$ as

$$\Delta \left[\begin{array}{c} \mathbf{r}_d \\ \mathbf{r}_m \end{array} \right] \quad \longleftarrow \quad \left[\begin{array}{c} \mathbf{L} \\ \mathbf{A} \end{array} \right] \Delta\mathbf{m} \tag{1.15}$$

and the last thing in the loop is to use the conjugate-gradient subroutine `cgplus()` (page ??) to choose the length of the step size and to choose how much of the previous step to include.

```
# invint1 -- INVerse INTerpolation in 1-D.
#
subroutine invint1( aa,a1,  ord,     dd,nd,  o1,d1, mm,m1, niter)
integer iter,          a1,             nd,              m1, niter,    nr, rd, rm
real                aa(a1), ord(nd), dd(nd), o1,d1, mm(m1)  # in=a,ord,d.  out=m.
temporary real  dm(m1), sm(m1), rr( nd+m1+a1), dr( nd+m1+a1), sr( nd+m1+a1)
                                nr= nd+m1+a1;   rd=1;  rm=1+nd
        call negcopy(     nd,         dd,    rr(rd))       # r_d =     - d
        call lint1( 0, 1, o1,d1, ord, mm,m1, rr(rd), nd)   # r_d = L m - d
        call tcai1( 0, 0, aa,a1,      mm,m1, rr(rm))       # r_m = A m
do iter= 0, niter {
        call lint1( 1, 0, o1,d1, ord, dm,m1, rr(rd), nd)   #     L' r_d
        call tcai1( 1, 1, aa,a1,      dm,m1, rr(rm))       # Dm= L' r_d + A' r_m
        #
        call tcai1( 0, 0, aa,a1,      dm,m1, dr(rm))       # Dr_m = A Dm
        call lint1( 0, 0, o1,d1, ord, dm,m1, dr(rd), nd)   # Dr_d = L Dm
        call cgplus( iter, m1, mm, dm, sm,
                           nr, rr, dr, sr )
        }
return; end
```

Figure 1.6 shows an example for a $(1, -2, 1)$ filter with $\epsilon = 1$. The continuous curve representing the model $\mathbf{m}$ passes through the data points. Since the models are computed with transient convolution end effects, the models tend to damp linearly to zero outside the region where signal samples are given.

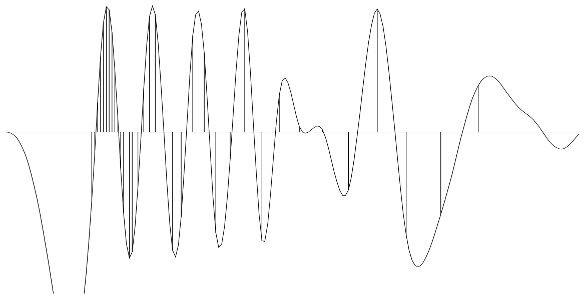


Figure 1.6: Sample points and estimation of a continuous function through them. iin-im1-2+1 [NR]

To show an example where the result is clearly a theoretical answer, I prepared another figure with the simpler filter $(1, -1)$. When we minimize energy in the first derivative of the waveform, the residual distributes itself uniformly between data points so the solution there is a straight line. Theoretically it should be a straight line because a straight line has a vanishing second derivative, and that condition arises by differentiating by $\mathbf{x}'$, the minimized quadratic form $\mathbf{x}'\mathbf{A}'\mathbf{A}\mathbf{x}$, getting $\mathbf{A}'\mathbf{A}\mathbf{x} = \mathbf{0}$. (By this logic, the curves between data points in Figure 1.6 must be cubics.) The $(1, -1)$ result is shown in Figure 1.7.

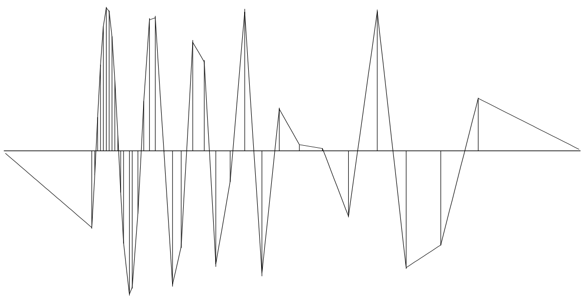


Figure 1.7: The same data samples and a function through them that minimizes the power in the first derivative. iin-im1-1a [NR]

1.2.1 Wells not matching the seismic map

Accurate knowledge comes from a **well**, but wells are expensive and far apart. Less accurate knowledge comes from surface seismology, but this knowledge is available densely in space and can indicate significant **trend**s between the wells. For example a prospective area may contain 15 wells but 600 or more seismic stations. To choose future well locations, it is helpful to match the known well data with the seismic data. Although the seismic data is delightfully dense in space, it often mismatches the wells because there are systematic differences in the nature of the measurements. These discrepancies are sometimes attributed to velocity **anisotropy**. We do not need to track down the physical model, we need only to merge the information somehow so we can appropriately **map** the trends between wells and make a proposal for the next drill site. Such empirical corrections are often called "**fudge factor**s".

Here we consider only a scalar value at each location. Take $\mathbf{w}$ to be a vector of 15 components, each component being the seismic travel time to some fixed depth in a well. Likewise let $\mathbf{s}$ be a 300 component vector each with the seismic travel time to that fixed depth as estimated wholly from surface seismology. We will take our map $\mathbf{m}$ to overlay the seismic data $\mathbf{s}$ which we can assume to lie on a uniform mesh. The regression (1.12) is almost what we need, but we need a little more, namely

$$
\begin{array}{lcl}
\mathbf{0} & \approx & \mathbf{Lm} - \mathbf{w} \\
\mathbf{0} & \approx & \mathbf{A}(\mathbf{m} - \mathbf{s})
\end{array}
\tag{1.16}
$$

where $\mathbf{A}$ is a roughening operator like ∇ or ∇^2. As before, the first regression $\mathbf{0} \approx \mathbf{Lm} - \mathbf{w}$ says when we linearly interpolate from the map, we should get the well data. The second regression $\mathbf{0} \approx \mathbf{A}(\mathbf{m} - \mathbf{s})$ says the map $\mathbf{m}$ should match the seismic data $\mathbf{s}$ at high frequencies but need not do so at low frequencies. Naturally, there are subjective questions such as what roughening operator is a good one, and at what frequency should the regressions be in balance? Later in this book we address issues like these and see how to solve this problem when the data is distributed in a plane instead of simply being on a line like the regression (1.13).

EXERCISES:

1 Write out the regression (1.16) in full like regression (1.13). What is the new ingredient?

2 Show how to modify subroutine `invint1()` (page 8) to include the new feature in the regression (1.16).

3 What regression is required if the seismic data is not uniformly spaced on the spatial axis? (Assume that the number of components in $\mathbf{m}$ exceeds that in $\mathbf{s}$.)

1.3 NOISE BURSTS

Sometimes noise comes in isolated **spikes**. Sometimes it comes in **bursts** or bunches (like grapes). Figure 1.8 is a simple one-dimensional example of a periodic signal plus spikes and bursts. Three processes are applied to this data, **despike** and two flavors of **deburst**. Here we will examine the processes used. (For even better results, see chapter ??, Figure ??.)

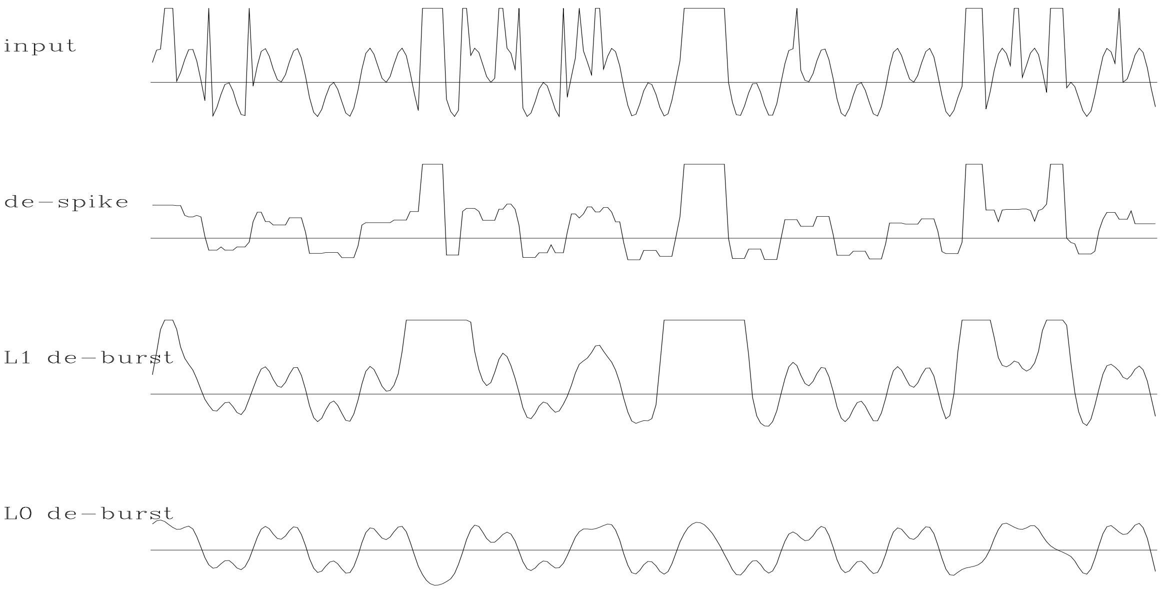


Figure 1.8: Top is synthetic data with noise spikes and bursts. (Most bursts are a hundred times larger than shown.) Next is after running medians. Next is an L_1 guided deburst processing. Bottom is L_0 guided deburst. iin-burst [NR]

1.3.1 Means, medians, and modes

Means, **medians**, and **modes** are different averages. Given some data values d_i for $i = 1, 2, ..., N$, the arithmetic mean value m_2 is

$$m_2 \quad = \quad \frac{1}{N} \sum_{i=1}^{N} d_i \tag{1.17}$$

It is useful to notice that this m_2 is the solution of the simple regression $d_i \approx m_2$ or $\mathbf{d} \approx m_2$, in other words, $\min_{m_2} \sum (m_2 - d_i)^2$ or

$$0 \quad = \quad \frac{d}{dm_2} \sum_{i=1}^{N} (m_2 - d_i)^2 \tag{1.18}$$

The median of the d_i values is found by sorting the values from smallest to largest and then selecting the value in the middle. This is equivalent to minimizing

$$\min_{m_1} \sum_{i=1}^{N} |m_1 - d_i| \tag{1.19}$$

To see why, notice that the derivative of the absolute value function is the signum function,

$$\operatorname{sgn}(x) \quad = \quad \lim_{\epsilon \longrightarrow 0} \frac{x}{|x| + \epsilon} \tag{1.20}$$

So the unfamiliar minimization

$$0 \quad = \quad \frac{d}{dm_1} \sum_{i=1}^{N} |m_1 - d_i| \tag{1.21}$$

$$0 \quad = \quad \frac{d}{dm_1} \sum_{i=1}^{N} \operatorname{sgn}(m_1 - d_i) \tag{1.22}$$

means that m_1 is greater than half the data values and less than the other half which is the definition of a median. The mean is said to minimize the L_2 norm of the residual and the median is said to minimize its L_1 norm.

Yet another average is the "mode" which is the most commonly occurring value. For example, in the number sequence $(1, 2, 3, 5, 5)$ the mode is 5 because it occurs the most times. Mathematically, the mode minimizes the zero norm of the residual, namely $L_0 = |m_0 - d_i|^0$. To see why, notice that when we raise a residual to the zero power, the result is 0 if $d_i = m_0$, and it is 1 if $d_i \neq m_0$. Thus, the L_0 sum of the residuals is the total number of residuals less those for which d_i matches m_0. The minimum of $L_0(m)$ is the mode $m = m_0$. The zero power function is nondifferentiable at the place of interest so we do not look at the gradient.

$L_2(m)$ and $L_1(m)$ are convex functions of m (positive second derivative for all m) which leads to the triangle inequalities $L_p(a) + L_p(b) \geq L_p(a + b)$ for $p \geq 1$ and

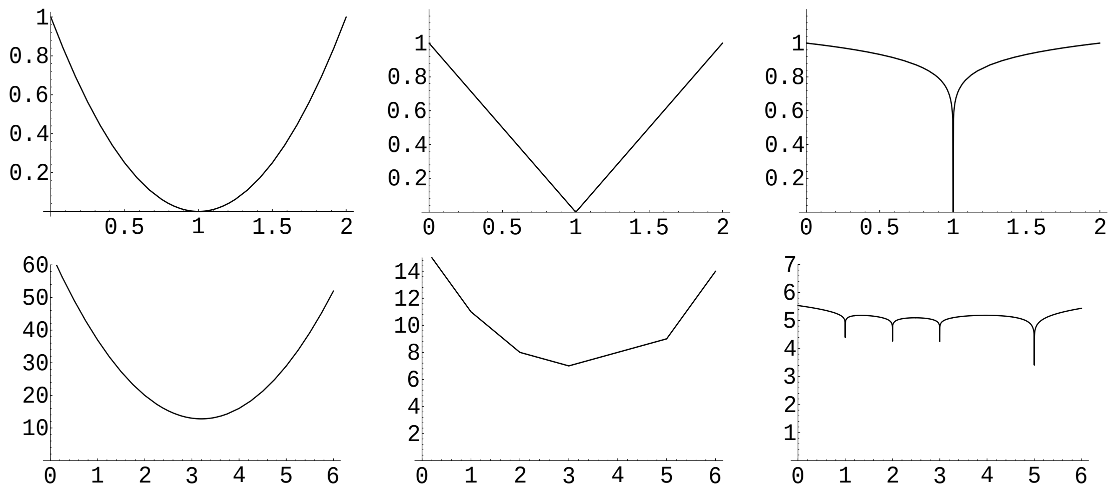


Figure 1.9: The ordinates are m. Top is the L_2, L_1, and $L_{1/10} \approx L_0$ measures of $m-1$. Bottom is the same measures of the data set $(1, 2, 3, 5, 5)$. iin-norms [NR]

assures slopes lead to a unique (if $p > 1$) bottom. Technically L_0 should not be called a "norm"; it should be called a "measure".

Since the most values are at the mode, the mode is where a probability function is maximum. The mode occurs with the maximum liklihood. It is awkward to think of the mode for floating-point values where the probability is miniscule (and irrelevant) that any two values are identical.

Although the mode is difficult to deal with theoretically and there is little literature about it, the L_0-guided approach is evidently better than the L_1-guided approach in the debursting task in Figure 1.8. The mode seems to say to throw away bad values while the median says to handle their polarity as the polarities of all other values.

1.3.2 Percentiles and Hoare's algorithm

The median is the 50-th **percentile**. After residuals are ordered from smallest to largest, the 90-th percentile is the value with 10% of the values above and 90% below. At SEP the default value for clipping plots of field data is at the 98th percentile. In other words, magnitudes above the 98-th percentile are plotted at the 98-th percentile. Any percentile is most easily defined if the population of values a_i, for $i = 1, 2, ..., n$ has been sorted into order so that $a_i \leq a_{i+1}$ for all i. Then the 90-th percentile is a_k where $k = (90n)/100$.

We can save much work by using Hoare's algorithm. Do not order the whole list. Instead, divide it into two groups, a group below a_k and another group above a_k. This reordering can be done "in place". Start one pointer at the top of the list and another at the bottom. Grab an arbitrary value from the list (such as the current value of a_k). March the two pointers towards each other until you have an upper value out of order with a_k and a lower value out of order with a_k. Swap the upper and lower value. Continue until the pointers merge somewhere midlist. Now you have divided the list into two sublists, one above your (random) value a_k and the other below. By comparing the place the pointers merged to the value of k, you can abandon one of these two lists. Continuing recursively, process the other sublist as described above, abandoning another sublist at each stage. Theoretically about $3n$ comparisons are expected to find the median of a list of n values. A subroutine (that I adapted from **Luis Canales**) for this task is `quantile()` (page 13).

```
# Quantile finders:
#
# value = value of bb(k) if bb(1...n) were sorted into increasing order.
subroutine quantile( k, n, bb, value)
                        integer   i, k, n;      real bb(n), value
                        temporary real aa(n)
                        do i= 1, n
                                aa(i) = bb(i)
                        call quantinternal( k, n, aa); value = aa(k); return; end
# value = value of abs(bb(k)) if abs(bb(1...n)) were sorted to increasing order.
subroutine quantabs( k, n, bb, value)
                        integer   i, k, n;      real bb(n), value
                        temporary real aa(n)
                        do i= 1, n
                                aa(i) = abs( bb(i))
                        call quantinternal( k, n, aa); value = aa(k); return; end
subroutine quantinternal( k, n, a)
integer k, n, i, j, low, hi;            real a(n), ak, aa
if( k<1 | n<k )  call erexit("quant: inputs not in range  1 <= k <= n ")
low = 1;        hi = n
while( low < hi) {
        ak = a(k);      i = low;        j = hi
        repeat{ if( a(i) < ak)
                        i = i+1
                else {  while( a(j) > ak)       j = j-1
                        if( i > j)  break
                        aa = a(i);      a(i) = a(j);    a(j) = aa
                        i = i+1;        j = j-1
                        if( i > j)  break
                        }
                }
        if( j < k)  low = i
        if( k < i)  hi = j
        }
return; end
```

1.3.3 The weighted mean

The **weighted mean** m is

$$m \quad = \quad \frac{\sum_{i=1}^{N} \ w_i^2 d_i}{\sum_{i=1}^{N} \ w_i^2} \tag{1.23}$$

where $w_i^2 > 0$ is the squared weighting function. This is the solution to the L_2 regression $0 \approx w_i(m - d_i)$, in other words,

$$0 \quad = \quad \frac{d}{dm} \sum_{i=1}^{N} [w_i(m - d_i)]^2 \tag{1.24}$$

It is curious to notice that if the final residuals $r_i = m - d_i$ could be known in advance we might define the weighting function $w_i^2 = 1/|r_i|$ thereby converting an L_2 minimization to an L_1 minimization. In practice this is risky because we might have a zero division in the $w_i^2 = 1/|r_i|$. Convergence might be unpredictable.

1.3.4 Iterated reweighting

Iterated reweighting is a way to estimate **median**s and **mode**s by familiar arithmetic techniques. We estimate either average by a sequence of weighted means, each found with a different weighting function, each weighting function being based on the residuals of the previous iteration. Roughly, we want a weight $w_i = |r_i|^{-1/2}$ for medians and $w_i = |r_i|^{-1}$ for modes. To stay away from zero division and to handle small values in the usual L_2 way we use a weight like

$$w_i^{(\text{median})} \quad = \quad \frac{1}{\sqrt{1 + |r_i/\overline{r}|}} \tag{1.25}$$

$$w_i^{(\text{mode})} \quad = \quad \frac{1}{1 + |r_i/\overline{r}|} \tag{1.26}$$

where $\overline{r}$ is some kind of average residual magnitude such as

$$\overline{r} \quad = \quad \text{median}(|r_i|) \tag{1.27}$$

The choice of $\overline{r}$ is quite arbitrary. Another percentile (instead of the median) could be used. Taking $\overline{r}$ to be larger, m will tend to the mean m_2. Taking $\overline{r}$ to be smaller, the iteratively reweighted mean will tend to the median m_1 or the mode m_0 (depending on the choice of equation (1.25) or (1.26)). An oversimplification might add insight: Effectively, regression equations are being abandoned where $r > \overline{r}$. If you have plenty of regression equations, then go ahead and abandon many by using a lower percentile cutoff.

1.3.5 Weighted L.S. conjugate-gradient template

The pseudocode for minimizing the *weighted* residual $\mathbf{0} \approx \mathbf{r} = \mathbf{W}(\mathbf{Fm} - \mathbf{d})$ by conjugate-gradient method, is effectively like the unweighted method except that the initial residual is weighted and the operator $\mathbf{F}$ has the premultiplier $\mathbf{W}$. Naturally, the adjoint operator $\mathbf{F}'$ has the postmultiplier $\mathbf{W}'$. In some applications the weighting operator $\mathbf{W}$ is simply a weighting function or diagonal matrix (so then $\mathbf{W} = \mathbf{W}'$) and in other applications, the weighting operator $\mathbf{W}$ may be an operator, such as a derivative along a data recording trajectory (so then $\mathbf{W} \neq \mathbf{W}'$).

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

When the weighting function depends upon the residual we have a nonlinear least-squares problem. We solve this by the conjugate-gradient method by recomputing the weighting function inside the iteration loop as follows:

$$
\begin{array}{lll}
\mathbf{W} & \longleftarrow & \mathbf{I} \\
\text{iterate } \{ & & \\
 & \begin{array}{lll}
\mathbf{r} & \longleftarrow & \mathbf{W}(\mathbf{F}\mathbf{m}-\mathbf{d}) \\
\mathbf{W} & \longleftarrow & \mathbf{diag}[w(r)] \\
\Delta\mathbf{m} & \longleftarrow & \mathbf{F}'\mathbf{W}'\,\mathbf{r} \\
\Delta\mathbf{r} & \longleftarrow & \mathbf{W}\mathbf{F}\,\Delta\mathbf{m} \\
(\mathbf{m},\mathbf{r}) & \longleftarrow & \text{cgplus}(\mathbf{m},\mathbf{r},\Delta\mathbf{m},\Delta\mathbf{r}) \\
\} & &
\end{array} &
\end{array}
$$

where $\mathbf{diag}[w(r)]$ is whatever weighting function we choose along the diagonal of a diagonal matrix. Experience shows that difficulties arise when the weighting function varies rapidly from one iteration to the next. Naturally the conjugate-gradient method, which remembers the previous iteration, will have an inappropriate memory if the weighting function changes too rapidly. A practical approach is to be sure the changes in the weighting function are minor or slowly variable. Unfortunately we have no theory for guidance.

1.3.6 De-spiking with median smoothing

The easiest method to remove spikes is to pass a moving window across the data and output the median value in the window. This method of despiking was done in Figure 1.8 which shows a problem of the method: The window is not long enough to clean the long bursts, but it is so long that it distorts the signal by flattening its peaks. The window size really should not be chosen in advance but should be chosen by what is encountered on the data. This I have not done because the long-burst problem is solved by another method described next.

1.3.7 De-bursting

Most signals are smooth, but running medians assume they have no curvature. An alternate expression of this assumption is that the signal has minimal curvature $0 \approx h_{i+1} - 2h_i + h_{i-1}$, in other words, $\mathbf{0} \approx \nabla^2 \mathbf{h}$. Thus we propose to create the cleaned up data $\mathbf{h}$ from the observed data $\mathbf{d}$ with the regression

$$0 \approx \mathbf{W}(\mathbf{h}-\mathbf{d}) \tag{1.28}$$

$$0 \approx \epsilon\nabla^2\mathbf{h} \tag{1.29}$$

where $\mathbf{W}$ is a diagonal matrix with weights sprinkled along the diagonal, and where ∇^2 is a matrix with a roughener like $(1, -2, 1)$ distributed along the diagonal. This is what I did in Figure 1.8 with $\epsilon = 1$. Experience showed similar performance for $0 \approx \nabla\mathbf{h}$ as for $0 \approx \nabla^2\mathbf{h}$. Better results, however, will be found later in Figure ?? where the ∇^2 operator is replaced by an operator designed to predict this specific signal.

This is our first example of solving a **nonlinear** optimization problem. The code changes only a bit from linear code. The residual computed inside the conjugate gradient step routine, `cgplus()` (page ??), needs to be recomputed when we change the weights. To do this we carry the residual initialization code from outside the iteration loop to inside as shown in subroutine `deburst1()` (page 16). (presto! nonlinear least squares!)

```
subroutine deburst1( niter, p, eps, dd,n,  hh)
integer i,     iter, niter, p,          n,           na, nr, h
real                           eps, dd(n), hh(n), aa(3), rbar
temporary real  rr(n*2),dr(n*2),sr(n*2), ww(n),dh(n),sh(n)
h=n+1;          nr=n*2;
call null( hh, n);       na=3;   aa(1) = -eps; aa(2) = 2*eps; aa(3) = -eps
do iter= 0, niter {
        do i=1,n
                rr(i) = hh(i) - dd(i)
        call quantabs( n/2, n, rr, rbar)                # rbar = median(|r_d|)
        do i = 1, n {
                     if( p == 0) { ww(i) = 1. /     (1. + abs(rr(i)/rbar) ) }
                else if( p == 1) { ww(i) = 1. / sqrt(1. + abs(rr(i)/rbar) ) }
                else             { ww(i) = 1.                                }
                }
        call diag(  0,0,   ww, n, rr,    rr   )         # r_d = W r_d
        call icai1( 0,0,1, aa,na, hh, n, rr(h))         # r_h = A h
# do linear CG {
        call diag(  1,0,   ww, n, dh,    rr   )         # dh = W * r_d
        call icai1( 1,1,1, aa,na, dh, n, rr(h))         # dh = W * r_d + L' r_h
        call icai1( 0,0,1, aa,na, dh, n, dr(h))         # d r_h = L dh
        call diag(  0,0,   ww, n, dh,    dr   )         # d r_d = W dh
        call cgplus( iter,   n, hh,dh,sh,
                            nr, rr,dr,sr )
        }
return; end
```

I invite you to experiment with different noises and different debursting methods and tell me what you find. Notice we have two parameters to adjust, ϵ and the number of iterations.

(The simple median approach can be generalized to L_1-norm multivariate regression by the method of linear programming as was done in FGDP and in my classic paper with Francis **Muir**. The reason I do not continue that approach is that so far as I know, those L_1 techniques require storing the operator. The trouble is we often have problems where the data and model spaces themselves strain the available memory while the operator size (data size *times* model size) is unreasonably large. As you see, iteratively reweighted least squares can amount to something like an L_1-norm approach anyway.)

1.3.8 The view from here

Multivariate regression is significantly more complicated than running medians. You have struggled to get this high. Let us look at the view in several directions.

Having done the simple bursty noise problem in Figure 1.8 by multivariate regression method, we are ready to fit noisy data to *general* linear models, say $\mathbf{d} \approx \mathbf{Fm}$ with

$$0 \approx \mathbf{W}(\mathbf{Fh} - \mathbf{d}) \tag{1.30}$$

$$0 \approx \epsilon \nabla^2 \mathbf{h} \tag{1.31}$$

Please notice that we have have learned to handle noise bursts in the *residual*, which is not the same as noise bursts in the *data*. We are now ready for cases where the model changes rapidly by being high frequency or having poles or discontinuities, for example the altitude function across a cliff where the regression is something like:

$$0 \approx \mathbf{W}_d(\mathbf{Fh} - \mathbf{d}) \tag{1.32}$$

$$0 \approx \mathbf{W}_h \nabla^2 \mathbf{h} \tag{1.33}$$

1.4 LEVELING THE RESIDUALS

The regression $\mathbf{d} \approx \mathbf{Fm}$ says we have data to fit to a model. The regression $\mathbf{0} \approx \mathbf{W}_n(\mathbf{Fm} - \mathbf{d})$ says we will do it with a weighting function. More generally, the weighting function could be a filter, or a filter followed by a weighting function. I call $\mathbf{W}_n$ a *leveler*.

Theoretical studies generally take a leveler to be something that must be given as part of the problem formulation, a so-called prior. That may be an important assumption for linear theory, but we will solve many real-life problems where to solve them, we find the leveler.

Where do levelers come from? Experimentalists generally apply filters to their data to eliminate known bad parts of the frequency spectrum. Likewise, and less mysteriously, experimentalists excise (or truncate or weight by zero) bad or unavailable locations in time and space. When observed data is cleaned up by various preprocessing, it makes sense to suggest that the theoretical data should pass through the same preprocessing. Rather than filter observed data with a running median, we found it makes more sense to minimize the L_1-norm of the difference of theoretical and observed data. Applying a running median to the data implies the data should be constant or of constant slope. Working with the residual, the theoretical data has no such limitation.

The leveler matrix times its adjoint is well known in academic statistical literature as the inverse **covariance matrix**. A concise summary of the inverse covariance matrix is found in **Strang**'s textbook *Introduction to Applied Mathematics*, pages 143-145 which explains its role in making a **Best Linear Unbiased Estimate** (**BLUE**). Unfortunately, the fundamental definition is based on probability functions, themselves almost never known in practice. The covariance matrix definition also follows from the idea of "ensemble average" and ensembles are not found in the experimental world either. In practice, an ensemble average might be approximated by some kind of an average over available residuals. The leveler is what we need, and in this book we do not estimate a covariance matrix, invert it, and then factor the inverse into the product $\mathbf{W}'\mathbf{W}$. Instead we estimate the leveler directly as an operator which will "level out" residuals in time or frequency domain. This is something we learn to do by trial and error, a bootstrapping process, mathematically nonlinear.

Another leveler applies to model space $\mathbf{0} \approx \mathbf{A}_m\mathbf{m}$. In each application, practitioner's art is required to choose the levelers $\mathbf{W}_n$ and $\mathbf{A}_m$. A major goal of this book (unlike the goals of books in the more mathematical literature) is to provide insights into how to find both kinds of levelers.

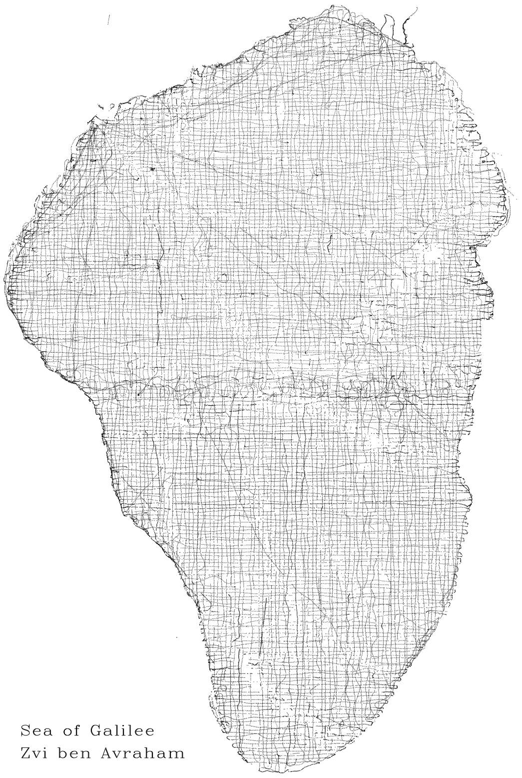
I call $\mathbf{W}_n$ and $\mathbf{A}_m$ "levelers" because when they are properly chosen, they will tend to reduce the numerical range of the final residuals. The method of minimum sums of *squares* of terms, tends to ignore smaller residuals because they are squared. Leveling thus tends to bring all regression equations to a scale where they can interact with the solution. Leveling brings things to an "appropriate" scale.

1.5 SEARCHING THE SEA OF GALILEE

Figure 1.10 shows a bottom sounding survey of the Sea of Galilee[1]. Let us discuss how we might process and display this data for the purpose of identifying **archeological**, geological, or geophysical details of the sea bottom. The Sea of Galilee is unique because it is a *fresh*-water lake *below* sea-level. It seems to be connected to the great rift (pull-apart) valley crossing east Africa. We might delineate the Jordan River delta. We might find springs on the water bottom. We might find archeological objects.

The data are 132,044 triples, (x_i, y_i, z_i), where x_i ranges over about 12 km and where y_i ranges over about 20 km. The tracks of surface locations are a sequence of dots so close together as to be barely discernable so you get an idea of the spatial resolution. The depths z_i are recorded to an accuracy of about 10 cm. The plot on the right is a 150×150 mesh into which data in each bin (rectangular pixel) has been

[1]Data collected by Zvi **ben Avraham**, TelAviv University. Please communicate with him `zvi@jupiter1.tau.ac.il` for more details or if you make something publishable with his data.



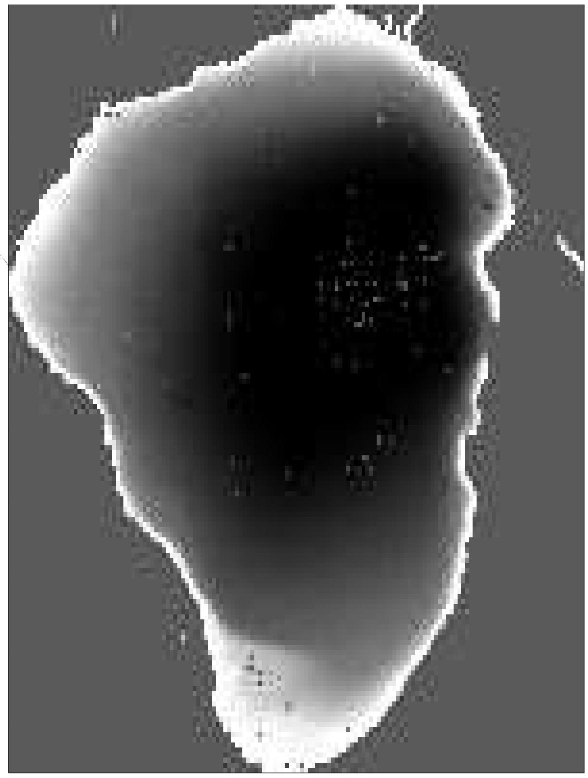


Figure 1.10: Left is surface locations. Right is binned depths. iin-galilee [NR]

simply averaged. We can do much better than this, but I chose low resolution for this first plot so as not to be distracted by too many empty bins (which are gray). In simplest terms, we want to remove the glitches, put the data on a regular mesh, and filter it to reveal the shorter wavelengths. The result should be like looking at an empty lake bed with various illumination and viewing angles.

Mathematically, the problem is simple on the coarse mesh shown in Figure 1.10. As we refine the mesh for more detail, the number of empty bins grows as does the care needed in devising a technique for filling them. We would use a regression like:

$$0 \approx \mathbf{W}(\mathbf{L}\mathbf{h} - \mathbf{d}) \tag{1.34}$$

$$0 \approx \epsilon \nabla \mathbf{h} \tag{1.35}$$

where data and operators are extended from 1-D to 2-D and where $\mathbf{L}$ is the nearest-neighbor binning operator or the linear interpolation operator.

Because Figure 1.10 shows little hint of microstructure on the sea floor, we will need to take great care in using the redundancy of crossing tracks to get a good estimate of the noise covariance so the final image is not badly streaked by recording track artifacts. The black-dot glitches (St. Peter's fish?) need removing but we must be careful not to throw out the sunken ships along with the bad data points. I had fewer glitches (but stronger ones) when I plotted the last value in each pixel instead of the average in the pixel. Medians would be better than means so for the regressions we need something like an L_1-norm or more simply, a weighting function inverse to the residual, say $w_i = 1/(|r_i| + \epsilon)$ where ϵ is the depth resolution.

We want the sharpest possible view of this classical site. A treasure hunt is never easy and no one guarantees we will find anything of great value but at least the exercise is a good warm-up for submarine petroleum exploration.

1.6 VESUVIUS PHASE UNWRAPPING

Figure 1.11 shows radar images of Mt. Vesuvius in Italy. These images are made from backscatter signals $s_1(t)$ and $s_2(t)$, recorded along two **satellite orbit**s 800 km high and 54 meters apart. The signals are very high frequency (the radar wavelength being 2.7 cm). They were Fourier transformed and one multiplied by the complex conjugate of the other getting the product $S_1(\omega)\bar{S}_2(\omega)$. The product's amplitude and phase are shown in Figure 1.11.

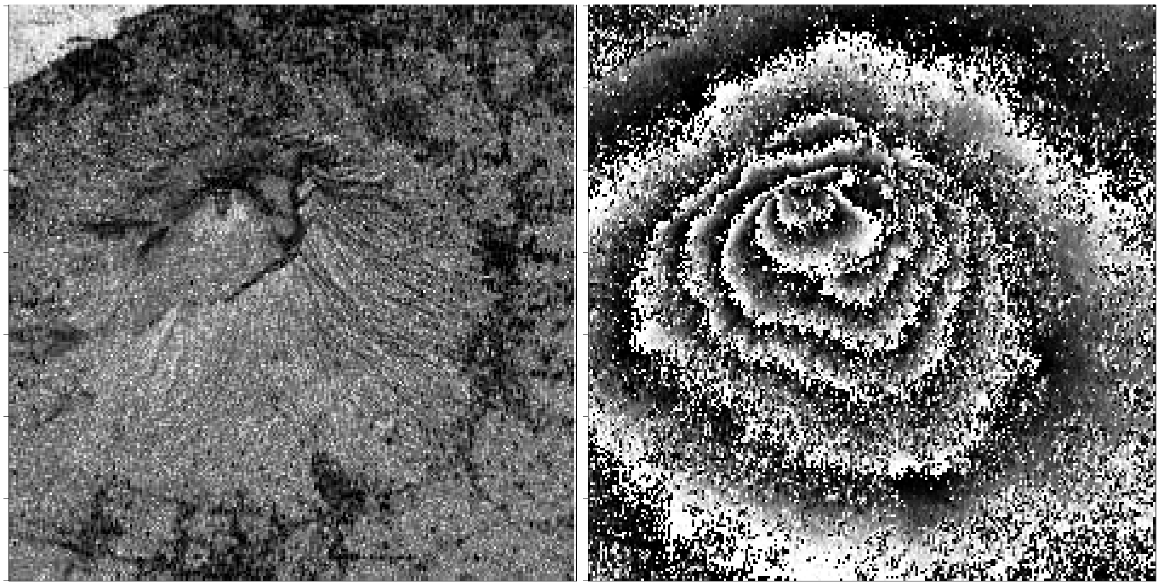


Figure 1.11: Radar image of Mt. Vesuvius. Left is the amplitude. Right is the phase. (Umberto Spagnolini) iin-vesuvio [NR]

From the figure we see that **contour**s of constant phase appear to be contours of constant altitude leading us to suppose that a study of radar theory would lead us to a linear relation between phase and altitude. Here I outline how the phase might be unwrapped, giving us an altitude map of Vesuvius, actually, the altitude would be measured along the viewing angle which happens to be 23 degrees from the vertical. Begin with a complex number Z representing the complex-valued image at any location in the (x, y)-plane.

$$re^{i\phi} = Z \tag{1.36}$$

$$\ln|r| + i\phi = \ln Z \tag{1.37}$$

$$\phi = \Im \ln Z \tag{1.38}$$

$$\frac{\partial \phi}{\partial x} = \Im \frac{1}{Z} \frac{\partial Z}{\partial x} \tag{1.39}$$

$$\frac{\partial \phi}{\partial x} = \Im \frac{\bar{Z} \frac{\partial Z}{\partial x}}{\bar{Z} Z} \tag{1.40}$$

For every point on the y-axis, equation (1.40) is a differential equation on the x-axis, and we could integrate them all to find $\phi(x, y)$. That sounds easy. On the other hand, the same equations are valid when x and y are interchanged, so we get twice as many equations as unknowns. For ideal data, either of these sets of equations should be equivalent to the other but for real data we expect to be fitting the regression

$$(\bar{Z} Z) \nabla \phi \approx \Im \bar{Z} \nabla Z \tag{1.41}$$

where $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$.

To relate equation (1.41) to the theme of this book, think of (1.41) as a regression, say $\mathbf{L}\phi \approx \mathbf{d}$. Since the number of equations exceeds the number of unknowns, we might not need to supplement the regression by $\mathbf{A}\phi \approx \mathbf{0}$ where $\mathbf{A}$ is the operator of two-dimensional prediction error.

I propose to represent the derivatives in $\mathbf{L} = (\bar{Z} Z)\nabla$ diagonally on a cartesian mesh. A differencing star I envision for application on the (x, y)-plane is

$$\mathbf{L} = \frac{\bar{Z} Z}{4} \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \times \left(\begin{array}{c} \begin{array}{|c|c|} \hline 0 & +1 \\ \hline -1 & 0 \\ \hline \end{array} \\ \\ \begin{array}{|c|c|} \hline -1 & 0 \\ \hline 0 & +1 \\ \hline \end{array} \end{array} \right) \tag{1.42}$$

The viewing angle in Figure 1.11 might be such that the mountain blocks some of the landscape behind it. This leads to the interesting possibility that the phase function must have a discontinuity where our viewing angle jumps over the hidden terrain. It will be interesting to discover whether solutions behave sensibly there.

An open question is whether the required number of iterations is reasonable or whether we would need to uncover a preconditioner or more rapid solution method.

1.7 SCALING THE ADJOINT

Given the usual linearized regression between data space and model space, $\mathbf{d} \approx \mathbf{F}\mathbf{m}$, the simplest image of the model space results from application of the adjoint operator $\hat{\mathbf{m}} = \mathbf{F}'\mathbf{d}$. Unless $\mathbf{F}$ has no physical units, however, the physical units of $\hat{\mathbf{m}}$ do not match those of $\mathbf{m}$ so we need a scaling factor. The theoretical solution $\mathbf{m}_{\rm theor} = (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{d}$ suggests the scaling units should be those of $(\mathbf{F}'\mathbf{F})^{-1}$. We could probe the operator $\mathbf{F}$ or its adjoint with white noise or a zero frequency input. Bill **Symes** suggests we probe with the data $\mathbf{d}$ because it has the spectrum of interest. He proposes we make our image with $\hat{\mathbf{m}} = \mathbf{W}^2\mathbf{F}'\mathbf{d}$ where we choose the weighting function to be:

$$\mathbf{W}^2 \quad = \quad \mathbf{diag}\left(\frac{\mathbf{F}'\mathbf{d}}{\mathbf{F}'\mathbf{F}\mathbf{F}'\mathbf{d}}\right) \tag{1.43}$$

which obviously has the correct physical units. The weight $\mathbf{W}^2$ can be thought of as a diagonal matrix containing the ratio of two images. A problem with the choice (1.43) is that the denominator might vanish or might even be negative. The way to stabilize any ratio is suggested at the beginning of chapter ??, namely, we revise the ratio (1.43) by changing $\mathbf{W}^2 = \mathbf{diag}(a/b)$ to

$$\mathbf{W}^2 \quad = \quad \mathbf{diag}\left(\frac{< ab >}{< b^2 + \epsilon^2 >}\right) \tag{1.44}$$

where ϵ is a parameter to be chosen, and the angle braces indicate the possible need for local smoothing with, for example, subroutine `sdoubint()` (page 24).

Since a scaled **adjoint** is a guess at the solution to the regression, it is logical to choose values for ϵ and the smoothing parameters that give fastest convergence of the conjugate-gradient regression.

To go beyond the scaled adjoint we can use $\mathbf{W}$ as a **preconditioner**. To use $\mathbf{W}$ as a preconditioner we define implicitly a new set of variables $\mathbf{x}$ by the substitution $\mathbf{m} = \mathbf{W}\mathbf{x}$. Then $\mathbf{d} \approx \mathbf{F}\mathbf{m} = \mathbf{F}\mathbf{W}\mathbf{x}$. To find $\mathbf{x}$ instead of $\mathbf{m}$ we do CG iteration with the operator $\mathbf{F}\mathbf{W}$ instead of with $\mathbf{F}$. As usual, the first step of the iteration is to use the adjoint of $\mathbf{d} \approx \mathbf{F}\mathbf{W}\mathbf{x}$ to form the image $\hat{\mathbf{x}} = (\mathbf{F}\mathbf{W})'\mathbf{d}$. At the end of the iterations we convert from $\mathbf{x}$ back to $\mathbf{m}$ with $\mathbf{m} = \mathbf{W}\mathbf{x}$. The result after the first iteration $\hat{\mathbf{m}} = \mathbf{W}\hat{\mathbf{x}} = \mathbf{W}(\mathbf{F}\mathbf{W})'\mathbf{d} = \mathbf{W}^2\mathbf{F}'\mathbf{d}$ turns out to be the same as **Symes** scaling.

By (1.43), $\mathbf{W}$ has physical units inverse to $\mathbf{F}$. Thus the transformation $\mathbf{F}\mathbf{W}$ has no units so the $\mathbf{x}$ variables have physical units of data space. It might be more practical to view the solution $\mathbf{x}$ with data units than to view the solution $\mathbf{m}$ with the more theoretical model units.

1.8 PRECONDITIONING

Iterative methods like CG can sometimes be accelerated by a **change of variables**. Say we are solving $\mathbf{F}\mathbf{m} \approx \mathbf{d}$. We implicitly define new variables $\mathbf{x}$ by a **trial solution** $\mathbf{m} = \mathbf{B}\mathbf{x}$ getting $(\mathbf{F}\mathbf{B})\mathbf{x} \approx \mathbf{d}$. After solving this system for $\mathbf{x}$, we merely evaluate $\mathbf{m} = \mathbf{B}\mathbf{x}$ to get the solution to the original problem. We hope this change of variables has saved effort. On the surface, it looks worse because instead of iterative application of $\mathbf{F}$ and $\mathbf{F}'$ we get iterative application of $\mathbf{F}\mathbf{B}$ and $\mathbf{B}'\mathbf{F}'$. This is no burden if the operator $\mathbf{B}$ is quicker than $\mathbf{F}$. Our hope is that the number of iterations decreases because we are clever or lucky in our choice of $\mathbf{B}$.

If we should be so lucky that the matrix $\mathbf{B}$ happens to be the inverse to the matrix $\mathbf{F}$, then we get the solution immediately. Physical operators generally include scaling in the physical domain and the spectral domain, so good candidates for a **precondition**ing operator **B** are those that do inverse scaling. Likewise in numerical analysis, it is well known that it is wise to choose **B** so that "columns of (**FB**) have equal scales" so **B** could be a diagonal matrix scaling columns of **F**.

The use of a preconditioner should not change the final solution unless, as Dave **Nichols** points out, the operator **F** has a **null space**. We have seen that iterating with **F** cannot change any null space component $\mathbf{Fm}_0$ in the initial guess. Likewise iterating with **FB** says the same about the the null space of **FB**. Where the number of unknowns exceeds the number of iterations we can afford, we expect a better answer to come from a preconditioned method with a better starting guess.

1.8.1 Preconditioning by integration

Often we seek solutions that are smooth in regions where there is no information. Integration is a way of making smooth functions from rough ones. The recursive nature of integration makes it a quick way to carry information a long distance. A good place to start is from the leaky integration operator **C** of equation (??). Often symmetry (or antisymmetry) seems appropriate. Two possible **precondition**ing substitutions are the trial solutions:

$$\mathbf{m} = (\mathbf{C} - \mathbf{C}')\mathbf{x} = \mathbf{B}_1\mathbf{x}_1 \tag{1.45}$$

$$\mathbf{m} = (\mathbf{C}'\mathbf{C} + \mathbf{C}\mathbf{C}')\mathbf{x} = \mathbf{B}_2\mathbf{x}_2 \tag{1.46}$$

The substitution (1.45) changes the problem-formulation variable to a new signal $\mathbf{x}_1$ which is like the derivative of the solution signal **m**. The substitution (1.46) changes variable to the second derivative of the solution. We can try both. Integration seems a natural way to quickly arrive at the solution we see in Figure 1.7 where we notice a corner in the solution wherever there is a data point. Starting from a small number of "corners" in **x**, double integration could give the solution **m**. Unfortunately, we need to experiment with the integration leak rate ρ in equation (??).

Subroutine `asymleak()` implements equation (1.45).

```
# Antisymmetric leaky integration  y=(C-C')x  where C = leaky integration.
#
subroutine asymleak( adj, add, rho, xx,nn,  yy  )
integer i,           adj, add,         nn
real                           rho, xx(nn), yy(nn)
temporary real                      cc(nn), cp(nn)
call adjnull(        adj, add,      xx,nn,  yy,nn)
if( adj == 0) { call leakint( 0, 0, rho, xx, nn, cc)    # cc = C x
                call leakint( 1, 0, rho, cp, nn, xx)    # cp = C'x
                do i = 1, nn
                        yy(i) =  (cc(i) - cp(i))        # y = (C-C')x
        }
else {          call leakint( 0, 0, rho, yy, nn, cc)    # cc = C y
                call leakint( 1, 0, rho, cp, nn, yy)    # cp = C'y
                do i = 1, nn
                        xx(i) =  (cp(i) - cc(i))        # x = (C'-C)y
```

```
        }
return;  end
```

Subroutine `sdoubint()` implements equation (1.46). This operator is its own adjoint.

```
# Two-way leaky integration  y=(C'C+CC')x  where C = leaky integration.
#
subroutine sdoubint( rho, xx,nn,  yy  )
integer i,                 nn
real                 rho, xx(nn), yy(nn)
temporary real     tt(nn), ca(nn), ac(nn)
call leakint( 0, 0, rho, xx, nn, tt)
call leakint( 1, 0, rho, ac, nn, tt)
call leakint( 1, 0, rho, tt, nn, xx)
call leakint( 0, 0, rho, tt, nn, ca)
do i = 1, nn
            yy(i) =  (ac(i) + ca(i)) * (1-rho) / 2.
return;  end
```

Convergence speed is no problem in the simple one-dimensional example of Figure 1.7, but it is in two- and three-dimensional problems where the number of unknowns vastly exceeds the number of iterations that we can afford. The simple example of Figure 1.7 provides a useful test case because in it we can iterate all the way to the exact solution. In this example there are 200 mesh points so theoretically convergence is achieved at the 200th iteration (and the movie shows it is achieved somewhat earlier). To test the change of variables of equation (1.46), Figure 1.12 shows the result after 5 iterations both with and without the **precondition**er. Notice that the preconditioned method is much closer to the ultimate answer. I got best results in this example with $\rho =$ `rho = .7`.

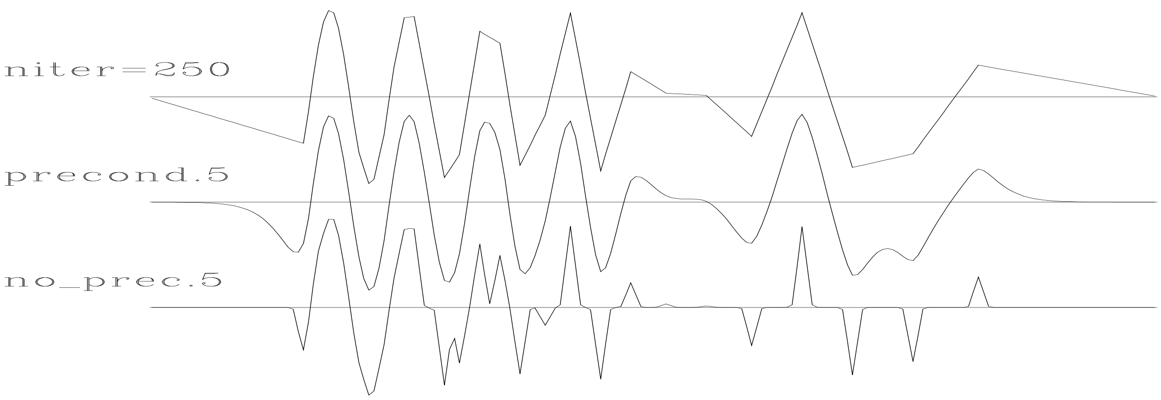


Figure 1.12: Top is the exact solution found in 200 iterations. Middle is with `sdoubint()` after 5 iterations. Bottom is without `sdoubint()` after 5 iterations. iin-early1-1a [NR]

The operator (1.13) for **linear interpolation** without the preconditioning (1.46) is $\left[\begin{array}{c}\mathbf{L}\\ \mathbf{A}\end{array}\right]$ and with the preconditioning it is $\left[\begin{array}{c}\mathbf{L}\\ \mathbf{A}\end{array}\right]\mathbf{B}$. Subroutine `invintp1()` (page 25) is derived from `invint1()` (page 8) by including preconditioning. Notice that it is almost the same as `invint1()`, the difference being three calls to `sdoubint()`, one in the adjoint operator, one in the forward operator, and one at the end to convert from "second-derivative" space to model space.

```
# invintp1 -- INVerse interpolation with preconditioning
#
subroutine invintp1( aa,a1,  ord,     dd,nd,  o1,d1, mm,m1, niter, rho)
integer iter,           a1,             nd,            m1, niter, rho, nr,rd,rm
real                  aa(a1), ord(nd), dd(nd), o1,d1, mm(m1)   # in=a,ord,d.  out=m.
temporary real xx(m1),dx(m1),sx(m1),dm(m1), rr( nd+m1+a1),dr( nd+m1+a1),sr( nd+m1+a1)
                                      nr= nd+m1+a1;     rd=1;  rm=1+nd
        call null (                   xx,m1        )       #  x   = 0
        call null (                   mm,m1        )       #  m   = 0
        call null (                          rr(rm),m1+a1) #  r_m = 0 = (A m)
        call negcopy(        nd,          dd,    rr(rd))    #  r_d = - d
do iter= 0, niter {
        call lint1( 1, 0, o1,d1, ord, dm,m1, rr(rd), nd)   #      L' r_d
        call tcai1( 1, 1, aa,a1,      dm,m1, rr(rm))       # dm= L' r_d + A' r_m
        call sdoubint( rho, dm, m1, dx )                   # dx= S dm
        # ----------
        call sdoubint( rho, dx, m1, dm )                   # dm= S dx
        call tcai1( 0, 0, aa,a1,      dm,m1, dr(rm))       # dr_m = A dm
        call lint1( 0, 0, o1,d1, ord, dm,m1, dr(rd), nd)   # dr_d = L dm
        call cgplus( iter, m1, xx, dx, sx,
                           nr, rr, dr, sr )
                call sdoubint( rho, xx, m1, mm )                 # movies
                call snap('mm',m1,1,mm); call snap('rr',nr,1,rr) # movies
        }
call         sdoubint( rho, xx, m1, mm )
return; end
```

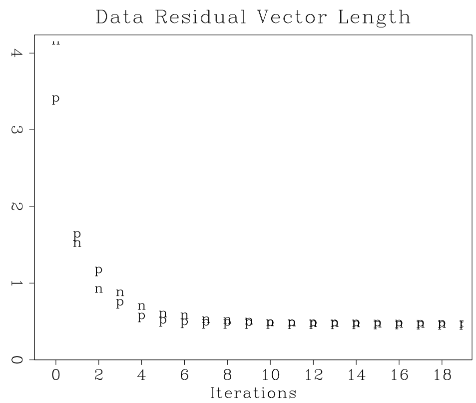


Figure 1.13: The "p" points are with preconditioning with `sdoubint()` and the "n" points are without. iin-usual1-1a [NR]

Figure 1.13 shows the residual vector length as a function of iteration. Surprisingly, by the objective measure of reducing error in data space, the preconditioning seems hardly an improvement. This has been characteristic of our experiences with preconditioning and we find it unsettling. How can the solution be better subjectively but not objectively? Matthias **Schwab** suggests that we evaluate preconditioners by $||\mathbf{m}_i - \mathbf{m}_\infty||$ where $\mathbf{m}_i$ is the model after the i-th iteration. Figure 1.14 shows it. If $||\mathbf{m}_i - \mathbf{m}_\infty||$ gives a better idea of the quality of the solution than $||\mathbf{Lm} - \mathbf{d}|| + \epsilon^2 ||\mathbf{Am}||$ then why not let it guide our descent? Unfortunately, that would be using the answer $\mathbf{m}_\infty$ to find the answer. Nevertheless, $||\mathbf{m}_i - \mathbf{m}_\infty||$ is a fine way to evaluate **preconditioning** strategies.

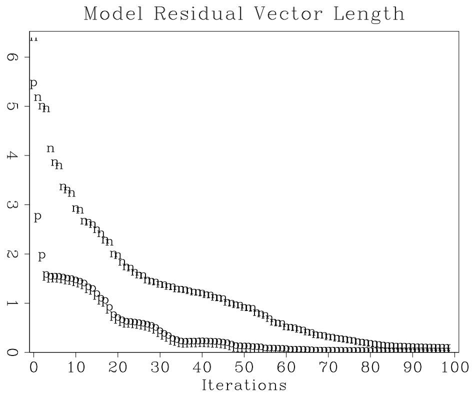


Figure 1.14: Distance of current solution from solution after 200 iterations. The bottom curve is with preconditioning. Top is without. iin-schwab1-1a [NR]

To understand this better think of two unknowns $\mathbf{m} = (m_1, m_2)$ where the contours of constant residual are ellipsoids around the best solution $\mathbf{m}_\infty$. Along any particular ellipsoid are a subset of models of equal quality measured by the residual in data space. Of this subset, minimizing $||\mathbf{m}_i - \mathbf{m}_\infty||$ selects the narrow waist of the ellipsoid as containing the best models.

We notice with interest that the preconditioner achieves in the first 5 iterations that which requires about 25 iterations without it. This benefit is diminished by the additional load of doing the preconditioning.

1.8.2 Recasting the problem in white variables

By the preconditioning transformation $\mathbf{m} = \mathbf{Bx}$ we have recast the original regression (1.12) into

$$\begin{aligned} \mathbf{0} &\approx \mathbf{LBx} - \mathbf{d} \\ \mathbf{0} &\approx \mathbf{ABx} \end{aligned} \tag{1.47}$$

where $\mathbf{A}$ is a roughener that arises in the problem formulation and where $\mathbf{B}$ is a smoother that arises in the problem solution. The choice of both $\mathbf{A}$ and $\mathbf{B}$ is somewhat subjective. $\mathbf{B}$ is often chosen to be somewhat inverse to $\mathbf{A}$. This suggests we eliminate $\mathbf{A}$ altogether by *defining* it to be proportional to the inverse of $\mathbf{B}$. Thus we would solve the regression

$$\begin{aligned} \mathbf{0} &\approx \mathbf{LBx} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\mathbf{Ix} \end{aligned} \tag{1.48}$$

and then obtain the final model by $\mathbf{m} = \mathbf{Bx}$. Theoretically, this is equivalent to solving

$$\begin{aligned} \mathbf{0} &\approx \mathbf{Lm} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\mathbf{B}^{-1}\mathbf{m} \end{aligned} \tag{1.49}$$

without needing to know the inverse operator $\mathbf{B}^{-1}$.

Although this changes the problem being solved, it does not seem to change it much. Earlier we specified a roughener and now we specify a smoother. Comparing

Figures 1.7 and 1.15 shows that the solutions are similar. Unfortunately the solution

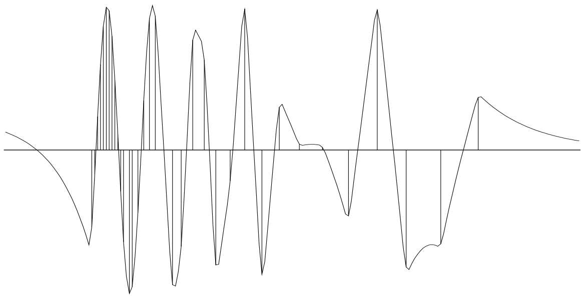


Figure 1.15: Alternative to Figure 1.7 using the regression (1.48) or (1.49). iin-whitvar [NR]

differs significantly at the ends. The smoothing method does not go to zero at the ends and I found it behaved increasingly strangely in the limit $\rho \to 1$. In that limit both methods interpolate with straight lines and I had hoped they would be the same solution. Thus I find the result of the the roughener method to be somewhat preferable. (Should this become a real problem, I would look back to well known methods (see IEI for example) for factoring a second derivative tridiagonal matrix into two bidiagonal matrices which can be solved recursively. Since these methods arise in connection with differential equations it seems likely that several boundary types will be available).

Figure 1.16 shows that over a large interval devoid of data, specifying the smoother gives a much quicker solution than specifying a roughener. Since speed will be important in problems in two and three dimensions, there should be a big role for the smoother method especially when there are large gaps in data.

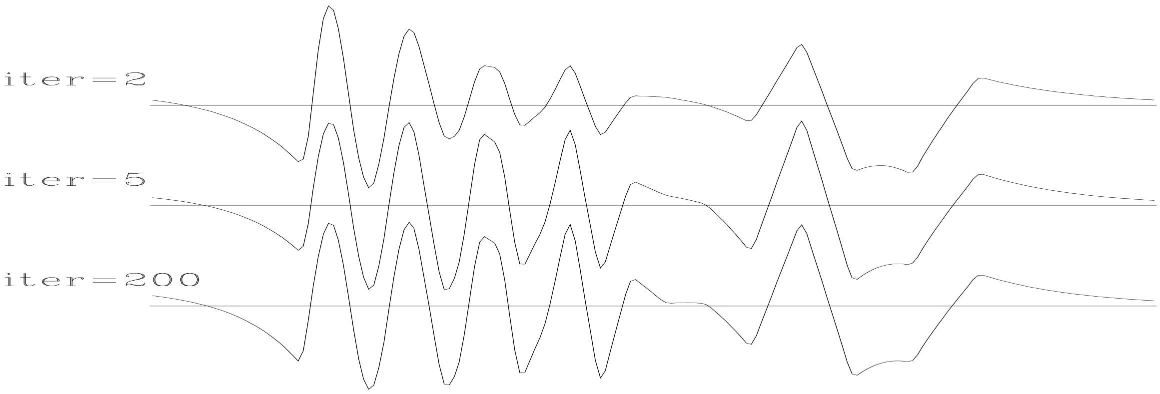


Figure 1.16: Solution to (1.48) or (1.49) after 2, 5, and 200 iterations. iin-whititer [NR]

1.8.3 Preconditioner conflicts with starting guess

In many applications, for many reasons, we have a starting guess $\mathbf{m}_0$ of the solution. It is usually a wise idea to use the starting guess rather than begin anew from $\mathbf{m} =$

0. This presents a small problem. With preconditioning we begin by defining new underlying model variables $\mathbf{x}$ implicitly by the substitution $\mathbf{m} = \mathbf{B}\mathbf{x}$. When we are given an initial $\mathbf{m}_0$, we have no direct way to find the associated $\mathbf{x}_0$ (unless both are zero) because we are not given $\mathbf{B}^{-1}$. This difficulty might not be serious, because in some applications we might have $\mathbf{x}_0$ while in others we might a reasonable guess at $\mathbf{B}^{-1}$. When $\mathbf{B}$ is essentially integration then $\mathbf{B}^{-1}$ is essentially differentiation.

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.