Chapter 7

Noisy data

Noise comes in two distinct flavors. First is erratic bursty noise which is difficult to fit into a statistical model. It bursts out of our simple models. To handle this noise we need "robust" estimation procedures which we consider first.

Next is noise that has a characteristic spectrum, temporal spectrum, spatial spectrum, or dip spectrum. Such noise is called "stationary" noise. A special case of stationary noise is low frequency drift of the mean value of a signal.

In real life, we need to handle both bursty noise and stationary noise at the same time. We'll try that now.

7.1 MEANS, MEDIANS, PERCENTILES AND MODES

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

$$m_2 = \frac{1}{N} \sum_{i=1}^{N} d_i \tag{7.1}$$

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

$$0 = \frac{d}{dm_2} \sum_{i=1}^{N} (m_2 - d_i)^2 \tag{7.2}$$

The median of the d_i values is found when the values are sorted from smallest to largest and then the value in the middle is selected. The median is delightfully well behaved even if some of your data values happen to be near infinity. Analytically, the median arises from the

optimization

$$\min_{m_1} \sum_{i=1}^{N} |m_1 - d_i| \tag{7.3}$$

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

$$\text{sgn}(x) \quad = \quad \lim_{\epsilon \longrightarrow 0} \frac{x}{|x| + \epsilon} \tag{7.4}$$

The gradient vanishes at the minimum.

$$0 \quad = \quad \frac{d}{dm_1} \sum_{i=1}^{N} |m_1 - d_i| \tag{7.5}$$

The derivative is easy and the result is a sum of sgn() functions,

$$0 \quad = \quad \sum_{i=1}^{N} \text{sgn}(m_1 - d_i) \tag{7.6}$$

In other words it is a sum of plus and minus ones. If the sum is to vanish, the number of plus ones must equal the number of minus ones. Thus 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.

Before this chapter, our model building was all based on the ℓ^2 norm. The median is clearly a good idea for data containing large bursts of noise, but the median is a single value while geophysical models are made from many unknown elements. The ℓ^1 norm offers us the new opportunity to build multiparameter models where the data includes huge bursts of noise.

Yet another average is the "**mode**," which is the most commonly occurring value. For example, in the number sequence $(1,1,2,3,5)$ the mode is 1 because it occurs the most times. Mathematically, the mode minimizes the zero norm of the residual, namely $\ell^0 = |m_0 - d_i|^0$. 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 $\ell^0(m)$ 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.

$\ell^2(m)$ and $\ell^1(m)$ are convex functions of m (positive second derivative for all m), and this fact leads to the triangle inequalities $\ell^p(a) + \ell^p(b) \geq \ell^p(a+b)$ for $p \geq 1$ and assures slopes lead to a unique (if $p > 1$) bottom. Because there is no triangle inequality for ℓ^0, it should not be called a "norm" but a "measure."

Because most values are at the mode, the mode is where a probability function is maximum. The mode occurs with the maximum likelihood. It is awkward to contemplate the mode for floating-point values where the probability is minuscule (and irrelevant) that any two values are identical. A more natural concept is to think of the mode as the bin containing the most values.

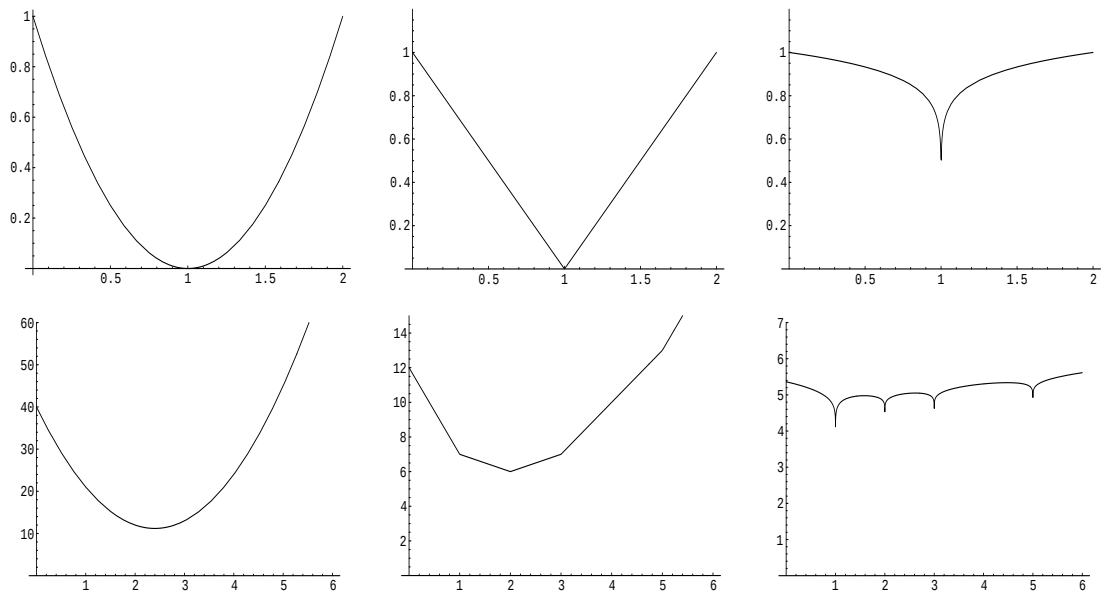

Figure 7.1: Mean, median, and mode. The coordinate is m. Top is the ℓ^2, ℓ^1, and $\ell^{1/10} \approx \ell^0$ measures of $m-1$. Bottom is the same measures of the data set $(1,1,2,3,5)$. (Made with Mathematica.) noiz-norms [CR]

7.1.1 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 $a_i \le a_{i+1}$ 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** which does not fully order the whole list, only enough of it to find the desired quantile. Hoare's algorithm is an outstanding example of the power of a recursive function, a function that calls itself. The main idea is this: We start by selecting a random value taken from the list of numbers. Then we split the list into two piles, one pile all values greater than the selected, the other pile all less. The quantile is in one of these piles, and by looking at their sizes, we know which one. So we repeat the process on that pile and ignore the other other one. Eventually the pile size reduces to one, and we have the answer.

In Fortran 77 or C it would be natural to split the list into two piles as follows:

> We divide the list of numbers 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.

Fortran 90 has some higher level intrinsic vector functions that simplify matters. When `a` is a vector and `ak` is a value, `a>ak` is a vector of logical values that are true for each component that is larger than `ak`. The integer count of how many components of `a` are larger than `ak` is given by the Fortran intrinsic function `count(a>ak)`. A vector containing only values less than `ak` is given by the Fortran intrinsic function `pack(a,a<ak)`.

Theoretically about $2n$ comparisons are expected to find the median of a list of n values. The code below (from Sergey Fomel) for this task is `quantile`.

```
module quantile_mod {    # quantile finder.    median = quantile( size(a)/2, a)
contains
  recursive function quantile( k, a) result( value) {
    integer,             intent (in)  :: k          # position in array
    real, dimension (:), intent (in)  :: a
    real                               :: value      # output value of quantile
    integer                            :: j
    real                               :: ak
    ak = a( k)
    j = count( a < ak)                                # how many a(:) < ak
    if( j >= k)
                value = quantile( k, pack( a, a < ak))
    else {
        j = count( a > ak) + k - size( a)
        if( j > 0)
                value = quantile( j, pack( a, a > ak))
        else
                value = ak
        }
    }
}
```

7.1.2 The weighted mean

The **weighted mean** m is

$$m = \frac{\sum_{i=1}^{N} w_i^2 d_i}{\sum_{i=1}^{N} w_i^2} \tag{7.7}$$

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

$$0 = \frac{d}{dm} \sum_{i=1}^{N} [w_i(m - d_i)]^2 \tag{7.8}$$

7.1.3 Weighted L.S. conjugate-direction template

The pseudocode for minimizing the *weighted* residual $\mathbf{0} \approx \mathbf{r} = \mathbf{W}(\mathbf{F}\mathbf{m} - \mathbf{d})$ by conjugate-direction method, is effectively like that for 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, like the derivative along a data recording trajectory (so then $\mathbf{W} \neq \mathbf{W}'$).

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

7.1.4 Multivariate ℓ^1 estimation by iterated reweighting

The easiest method of model fitting is linear least squares. This means minimizing the sums of squares of residuals (ℓ^2). On the other hand, we often encounter huge noises and it is much safer to minimize the sums of absolute values of residuals (ℓ^1). (The problem with ℓ^0 is that there are multiple minima, so the gradient is not a sensible way to seek the deepest).

There exist specialized techniques for handling ℓ^1 multivariate fitting problems. They should work better than the simple iterative reweighting outlined here.

A penalty function that ranges from ℓ^2 to ℓ^1, depending on the constant $\bar{r}$ is

$$E(\mathbf{r}) \quad = \quad \sum_i \left(\sqrt{1 + r_i^2/\bar{r}^2} - 1 \right) \tag{7.9}$$

Where $r_i/\bar{r}$ is small, the terms in the sum amount to $r_i^2/2\bar{r}^2$ and where $r_i^2/\bar{r}^2$ is large, the terms in the sum amount to $|r_i/\bar{r}|$. We define the residual as

$$r_i \quad = \quad \sum_j F_{ij} m_j - d_i \tag{7.10}$$

We will need

$$\frac{\partial r_i}{\partial m_k} \quad = \quad \sum_j F_{ij} \delta_{jk} \quad = \quad F_{ik} \tag{7.11}$$

where we briefly used the notation that δ_{jk} is 1 when $j = k$ and zero otherwise. Now, to let us find the descent direction $\Delta\mathbf{m}$, we will compute the k-th component g_k of the gradient $\mathbf{g}$. We

have

$$g_k \quad = \quad \frac{\partial E}{\partial m_k} \quad = \quad \sum_i \frac{1}{\sqrt{1+r_i^2/\bar{r}^2}} \frac{r_i}{\bar{r}^2} \frac{\partial r_i}{\partial m_k} \tag{7.12}$$

$$\mathbf{g} \quad = \quad \Delta\mathbf{m} \quad = \quad \mathbf{F}' \, \mathbf{diag} \left(\frac{1}{\sqrt{1+r_i^2/\bar{r}^2}} \right) \mathbf{r} \tag{7.13}$$

where we have use the notation **diag**() to designate a diagonal matrix with its argument distributed along the diagonal.

Continuing, we notice that the new weighting of residuals has nothing to do with the linear relation between model perturbation and residual perturbation; that is, we retain the familiar relations, $\mathbf{r} = \mathbf{F}\mathbf{m} - \mathbf{d}$ and $\Delta\mathbf{r} = \mathbf{F}\Delta\mathbf{m}$.

In practice we have the question of how to choose $\bar{r}$. I suggest that $\bar{r}$ be proportional to median($|r_i|$) or some other percentile.

7.1.5 Nonlinear L.S. conjugate-direction template

Nonlinear optimization arises from two causes:

1. Nonlinear physics. The operator depends upon the solution being attained.
2. Nonlinear statistics. We need robust estimators like the ℓ^1 norm.

The computing template below is useful in both cases. It is almost the same as the template for weighted linear least-squares except that the residual is recomputed at each iteration. Starting from the usual weighted least-squares template we simply move the iteration statement a bit earlier.

iterate {
$\mathbf{r} \quad \longleftarrow \quad \mathbf{F}\mathbf{m} - \mathbf{d}$
$\mathbf{W} \quad \longleftarrow \quad \mathbf{diag}[w(\mathbf{r})]$
$\mathbf{r} \quad \longleftarrow \quad \mathbf{W}\mathbf{r}$
$\Delta\mathbf{m} \quad \longleftarrow \quad \mathbf{F}'\mathbf{W}'\,\mathbf{r}$
$\Delta\mathbf{r} \quad \longleftarrow \quad \mathbf{W}\mathbf{F}\,\Delta\mathbf{m}$
$(\mathbf{m}, \mathbf{r}) \quad \longleftarrow \quad \text{cgstep}(\mathbf{m}, \mathbf{r}, \Delta\mathbf{m}, \Delta\mathbf{r})$
}

where **diag**$[w(\mathbf{r})]$ is whatever weighting function we choose along the diagonal of a diagonal matrix.

Now let us see how the weighting functions relate to robust estimation: Notice in the code template that $\mathbf{W}$ is applied twice in the definition of $\Delta\mathbf{m}$. Thus $\mathbf{W}$ is the square root of the diagonal operator in equation (1.13).

$$\mathbf{W} \quad = \quad \mathbf{diag}\left(\frac{1}{\sqrt{\sqrt{1+r_i^2/\bar{r}^2}}}\right) \tag{7.14}$$

Module `solver_irls` on page 7 implements the computational template above. In addition to the usual set of arguments from the `solver()` subroutine on this page, it accepts a user-defined function (parameter `wght`) for computing residual weights. Parameters `nmem` and `nfreq` control the restarting schedule of the iterative scheme.

```
module solver_irls {                        # 0 = Wdiag Wop (Fop Jop m - d) and periodic restart
  use chain0_mod + solver_report_mod
  logical, parameter, private  :: T = .true., F = .false.
contains
  subroutine solver_irls( m,d, Fop, stepper, niter &
  ,             Wop,Jop,Wdiag,m0,nmem,nfreq,err,resd,mmov,rmov,verb) {
    optional :: Wop,Jop,Wdiag,m0,nmem,nfreq,err,resd,mmov,rmov,verb
    interface { #------------------------- begin definitions -----------
       integer function Fop  (adj,add,m,d){real::m(:),d(:);logical,intent(in)::adj,add}
       integer function Wop  (adj,add,m,d){real::m(:),d(:);logical,intent(in)::adj,add}
       integer function Jop  (adj,add,m,d){real::m(:),d(:);logical,intent(in)::adj,add}
       integer function Wdiag(res, w)     {real::res(:),w(:)}
       integer function stepper(forget,m,g,rr,gg) {
         real, dimension(:)  ::        m,g,rr,gg
         logical             :: forget            }
    }
    real, dimension(:),    intent(in)          :: d, m0
    integer,               intent(in)          :: niter, nmem, nfreq
    logical,               intent(in)          :: verb
    real, dimension(:),    intent(out)         :: m,err, resd
    real, dimension(:,:),  intent(out)         ::        rmov, mmov
    real, dimension(size(m))                   :: g
    real, dimension(size(d)),         target   :: rr, gg
    real, dimension(size(d)+size(m)), target   :: tt
    real, dimension(:), pointer                :: rd, gd, td
    real, dimension(:), pointer                :: rm, gm, tm, wht
    integer                                    :: iter, stat
    logical                                    :: forget
    rd => rr(1:size(d));
    gd => gg(1:size(d));
    td => tt(1:size(d)); tm => tt(1+size(d):)
    if(present( Wop)) stat=Wop(F,F,-d,rd) # begin initialization --------
    else rd = -d                                        #Rd = -W  d
    if(present(Wdiag))allocate(wht(size(d)))
    if(present( m0)){ m=m0                              #m  =     m0
        if(present( Wop)) call chain0(Wop,Fop,F,T,m,rd,td)
        else              stat =          Fop(F,T,m,rd   ) #Rd+=  WF m0
    } else m=0
    forget = F; #------------------------- begin iterations ------------
```

```
    do iter = 1,niter {
       if(present(nmem)) forget = (iter>nmem)                    # initialize forget
       if(present(Wdiag).and.forget) stat = Wdiag(rd,wht)        # estimate weights
       if(present(Wdiag)) {
          rd = wht*rd
          if(present(Wop)) call chain0(Wop,Fop,T,F,g,wht*rd,td)
          else             stat =          Fop(T,F,g,wht*rd   )   # g  = (WF)'Rd
       } else {
          if(present(Wop)) call chain0(Wop,Fop,T,F,g,    rd,td)
          else             stat =          Fop(T,F,g,    rd   )   # g  = (WF)'Rd
       }
       if(forget.and.present(nfreq)) forget=(mod(iter,nfreq)==0)  # periodic restart
       if(present(Jop)){ tm=g;  stat  = Jop(F,F,tm, g  )}         # g  =   J  g
       if(present(Wop)) call chain0(Wop,Fop,F,F,g,gd,td)
       else             stat =          Fop(F,F,g,gd   )   #Gd = (WF) g
       if(present(Wdiag)) gd=wht*gd
       stat = stepper(forget, m,g, rr,gg)                  #m+=dm; R+=dR
       if(stat ==1) exit # got stuck descending
       if(present(Wdiag)) {
            rd = -d
            stat = Fop(F,T,m,rd)
            if(present( Wop)){
            td = rd
            stat = Wop(F,F,td,rd)}
       }
       if(present( mmov)) mmov(:,iter) =  m(:size(mmov,1)) # report -----
       if(present( rmov)) rmov(:,iter) = rd(:size(rmov,1))
       if(present( err ))  err(  iter) = dot_product(rd,rd)
       if(present( verb)){ if(verb) call solver_report(iter,m,g,rd)}
    }
    if(present( resd)) resd = rd
  }
}
```

We can ask whether `cgstep()`, which was not designed with nonlinear least-squares in mind, is doing the right thing with the weighting function. First, we know we are doing weighted linear least-squares correctly. Then we recall that on the first iteration, the conjugate-directions technique reduces to steepest descent, which amounts to a calculation of the scale factor α with

$$\alpha \quad = \quad - \frac{\Delta \mathbf{r} \cdot \mathbf{r}}{\Delta \mathbf{r} \cdot \Delta \mathbf{r}} \tag{7.15}$$

Of course, `cgstep()` knows nothing about the weighting function, but notice that the iteration loop above nicely inserts the weighting function both in $\mathbf{r}$ and in $\Delta\mathbf{r}$, as required by (1.15).

Experience shows that difficulties arise when the weighting function varies rapidly from one iteration to the next. Naturally, the conjugate-direction 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 slowly variable.

7.1.6 Minimizing the Cauchy function

A good trick (I discovered accidentally) is to use the weight

$$\mathbf{W} \quad = \quad \mathbf{diag}\left(\frac{1}{\sqrt{1+r_i^2/\bar{r}^2}}\right) \tag{7.16}$$

Sergey Fomel points out that this weight arises from minimizing the **Cauchy function**:

$$E(\mathbf{r}) \quad = \quad \sum_i \log(1+r_i^2/\bar{r}^2) \tag{7.17}$$

A plot of this function is found in Figure 1.2.

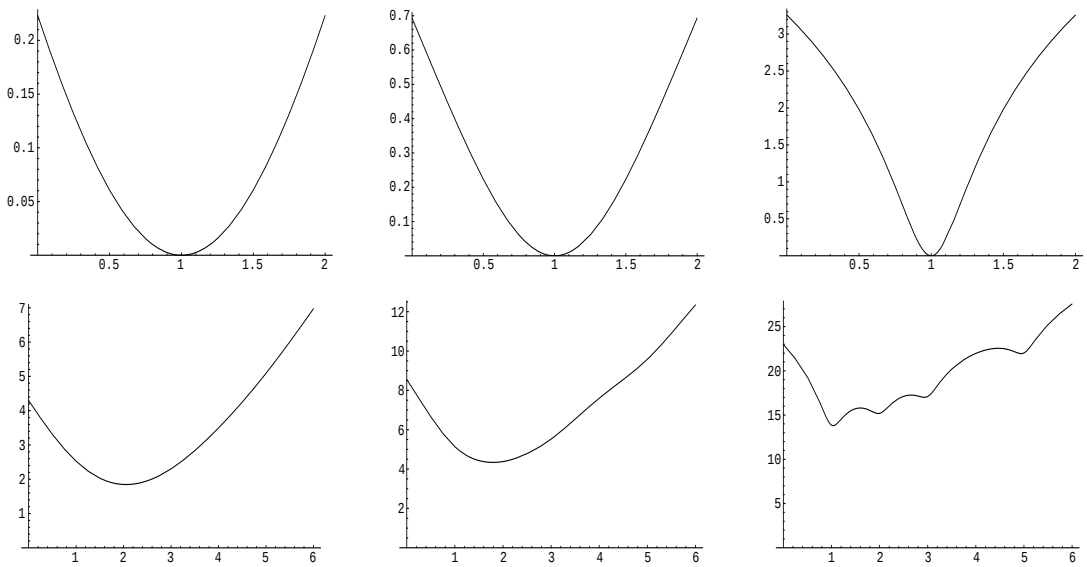

Figure 7.2: The coordinate is m. Top is Cauchy measures of $m-1$. Bottom is the same measures of the data set $(1,1,2,3,5)$. Left, center, and right are for $\bar{r}=(2,1,.2)$. noiz-cauchy [CR]

Because the second derivative is not positive everywhere, the Cauchy function introduces the possibility of multiple solutions, but because of the good results we see in Figure 1.3, you might like to try it anyway. Perhaps the reason it seems to work so well is that it uses mostly residuals of "average size," not the big ones or the small ones. This happens because $\Delta\mathbf{m}$ is made from $\mathbf{F}'$ and the components of $\mathbf{W}^2\mathbf{r}$ which are a function $r_i/(1+r_i^2/\bar{r}^2)$ that is maximum for those residuals near $\bar{r}$.

Module `irls` on page 9 supplies two useful weighting functions that can be interchanged as arguments to the reweighted scheme on page 7.

```
module irls {
  use quantile_mod
contains
  integer function l1 (res, weight)  {
    real, dimension (:)  :: res, weight
    real                 :: rbar
    rbar = quantile( int( 0.5*size(res)), abs (res))            # median
    weight = 1. / sqrt( sqrt (1. + (res/rbar)**2));   l1 = 0
    }
  integer function cauchy (res, weight)  {
    real, dimension (:)  :: res, weight
    real                 :: rbar
    rbar = quantile( int( 0.5*size(res)), abs (res))             # median
    weight = 1. / sqrt (1. + (res/rbar)**2);          cauchy = 0
    }
}
```

7.2 NOISE BURSTS

Sometimes noise comes in isolated **spikes**. Sometimes it comes in **bursts** or bunches (like grapes). Figure 1.3 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 Figure 1.5.)

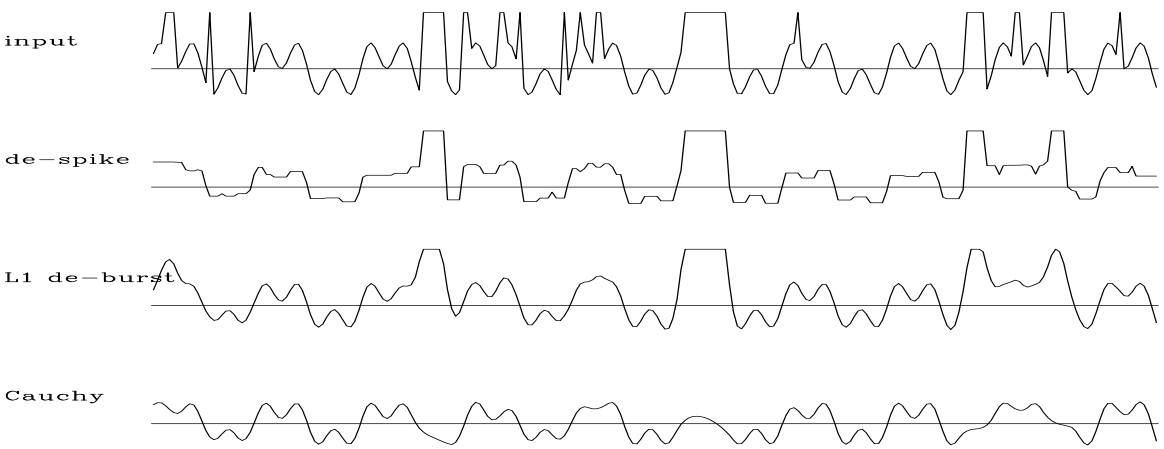

Figure 7.3: Top is synthetic data with noise spikes and bursts. (Most bursts are a hundred times larger than shown.) Next is after running medians. Bottom is after the two processes described here. noiz-burst90 [ER]

7.2.1 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 1.3, which

shows a problem of the method: The window is not long enough to clean the long bursts, but it is already so long that it distorts the signal by flattening its peaks. The window size really should not be chosen in advance but should depend upon by what is encountered on the data. This I have not done because the long-burst problem is solved by another method described next.

7.2.2 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 fitting problem

$$\begin{array}{rcl} 0 & \approx & \mathbf{W}(\mathbf{h} - \mathbf{d}) \\ 0 & \approx & \epsilon \, \nabla^2 \mathbf{h} \end{array} \tag{7.18}$$

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 shown in Figure 1.3 with $\epsilon = 1$. Experience showed similar performances for $0 \approx \nabla \mathbf{h}$ and $0 \approx \nabla^2 \mathbf{h}$. Better results, however, will be found later in Figure 1.5, where the ∇^2 operator is replaced by an operator designed to predict this very predictable signal.

7.3 MEDIAN BINNING

We usually add data into bins. When the data has erratic noise, we might prefer to take the median of the values in each bin. Subroutine `medianbin2()` (in the library, but not listed here) performs the chore. It is a little tricky because we first need to find out how many data values go into each bin, then we must allocate that space and copy each data value from its track location to its bin location. Finally we take the median in the bin. A small annoyance with medians is that when bins have an even number of points, like two, there no middle. To handle this problem, subroutine `medianbin2()` uses the average of the middle two points.

A useful byproduct of the calculation is the residual: For each data point its bin median is subtracted. The residual can be used to remove suspicious points before any traditional least-squares analysis is made. An overall strategy could be this: First a coarse binning with many points per bin, to identify suspicious data values, which are set aside. Then a sophisticated least squares analysis leading to a high-resolution depth model. If our search target is small, recalculate the residual with the high-resolution model and reexamine the suspicious data values.

Figure 1.4 compares the water depth in the Sea of Galilee with and without median binning. The difference does not seem great here but it is more significant than it looks. Later processing will distinguish between empty bins (containing an exact zero) and bins with small values in them. Because of the way the depth sounder works, it often records an erroneously

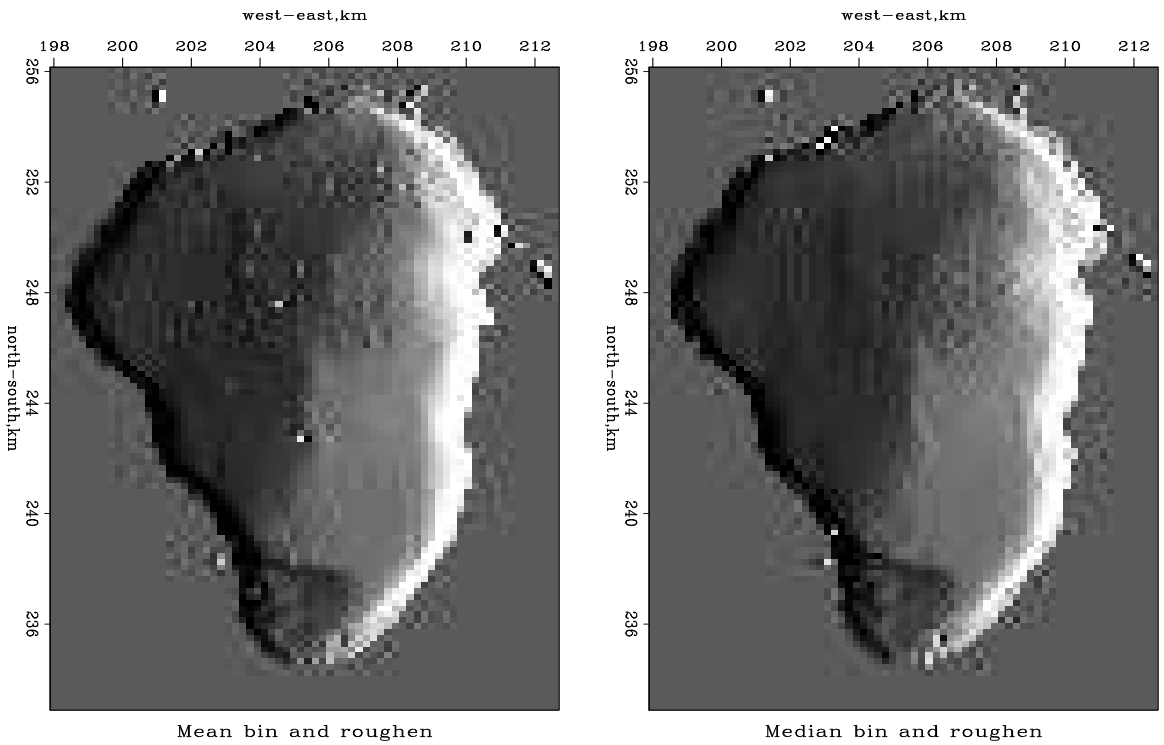

Figure 7.4: Galilee water depth binned and roughened. Left is binning with the mean, right with the median. noiz-medbin90 [ER,M]

near-zero depth. This will make a mess of our later processing (missing data fill) unless we cast out those data values. This was done by median binning in Figure 1.4 but the change is disguised by the many empty bins.

Median binning is a useful tool, but where bins are so small that they hold only one or two points, there the median for the bin is the same as the usual arithmetic average.

7.4 ROW NORMALIZED PEF

We often run into **bursty noise**. This can overwhelm the estimate of a prediction-error filter. To overcome this problem we can use a weighting function. The weight for each row in fitting matrix (6.27) is adjusted so that each row has about the same contribution as each other row. A first idea is that the weight for the n-th row would be the inverse of the sum of the absolute values of the row. This is easy to compute: First make a vector the size of the PEF **a** but with each element unity. Second, take a copy of the signal vector **y** but with the absolute value of each component. Third, convolve the two. The convolution of the ones with the absolute values could be the inverse of the weighting function we seek. However, any time we are forming an inverse we need to think about the possibility of dividing by zero, how it could arise, and how divisions by "near zero" could be even worse (because a poor result is not immediately recognized). Perhaps we should use something between ℓ^1 and ℓ^2 or Cauchy. In

any case, we must choose a scaling parameter that separates "average" rows from unusually large ones. For this choice in subroutine `rnpef1()`, I chose the median.

7.5 DEBURST

We can use the same technique to throw out fitting equations from defective data that we use for missing data. Recall the theory and discussion leading up to Figure 1.3. There we identified defective data by its lack of continuity. We used the fitting equations $0 \approx w_i(y_{i+1} - 2y_i + y_{i-1})$ where the weights w_i were chosen to be approximately the inverse to the residual $(y_{i+1} - 2y_i + y_{i-1})$ itself.

Here we will first use the second derivative (Laplacian in 1-D) to throw out bad points, while we determine the PEF. Having the PEF, we use it to fill in the missing data.

```
module pefest {      # Estimate a PEF avoiding zeros and bursty noise on input.
  use quantile_mod
  use helicon
  use misinput
  use pef
contains
  subroutine pefest1( niter, yy, aa) {
    integer, intent( in)         :: niter
    real, dimension( :), pointer :: yy
    type( filter)                :: aa
    real, dimension(:),allocatable  :: rr
    real                         :: rbar
    integer                      :: stat
                allocate(rr(size(yy)))
    call helicon_init( aa)                          # starting guess
    stat = helicon_lop( .false., .false., yy, rr)
    rbar = quantile( size( yy)/3, abs( rr))         # rbar=(r safe below rbar)
    where( aa%mis)  yy = 0.
    call find_mask(( yy /= 0. .and. abs( rr) < 5 * rbar), aa)
    call find_pef( yy, aa, niter)
                deallocate(rr)
  }
}
```

The result of this "PEF-deburst" processing is shown in Figure 1.5.

Given the PEF that comes out of `pefest1()`[1], subroutine `fixbad1()` below convolves it with the data and looks for anomalous large outputs. For each that is found, the input data is declared defective and set to zero. Then subroutine `mis1()` on page 145 is invoked to replace the zeroed values by reasonable ones.

```
module fixbad { # Given a PEF, find bad data and restore it.
  use mis2_mod
```

[1] If you are losing track of subroutines defined earlier, look at the top of the module to see what other modules it uses. Then look in the index to find page numbers of those modules.

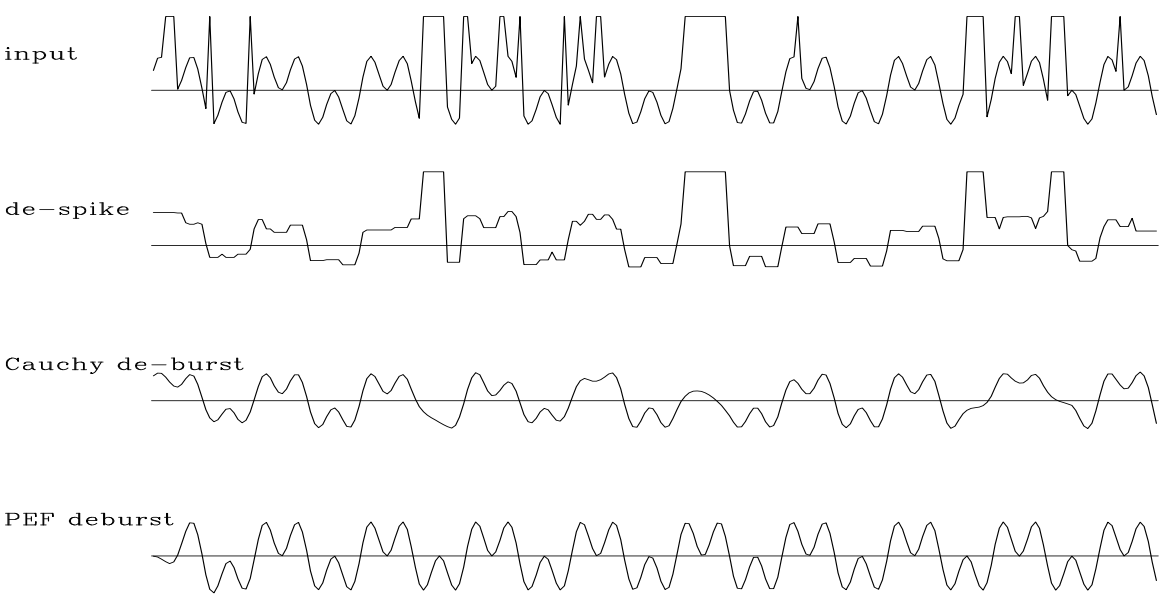

Figure 7.5: Top is synthetic data with noise spikes and bursts. (Some bursts are fifty times larger than shown.) Next is after running medians. Next is Laplacian filter Cauchy deburst processing. Last is PEF-deburst processing. noiz-pefdeburst90 [ER]

```
  use helicon
  use quantile_mod
contains
  subroutine fixbad1 (niter, aa, yy) {
    integer,       intent (in)      :: niter
    type( filter), intent (in)      :: aa
    real,    dimension (:)          :: yy
    real,    dimension (size (yy))  :: rr
    logical, dimension (size (yy))  :: known
    integer                         :: stat
    call helicon_init( aa)
    stat = helicon_lop (.false., .false., yy, rr); rr = abs (rr)
    known = ( yy > 0.) .and. ( rr < 4. * quantile( size(rr)/2, rr))
    call mis2 (niter, yy, aa, known, 2)
 }
}
```

7.5.1 Potential seismic applications of two-stage infill

Two-stage data infill has many applications that I have hardly begun to investigate.

Shot continuation is an obvious task for a data-cube extrapolation program. There are two applications of shot-continuation. First is the obvious one of repairing holes in data in an unobtrusive way. Second is to cooperate with reflection tomographic studies such as that proposed by Matthias **Schwab**.

Offset continuation is a well-developed topic because of its close link with **dip moveout** (**DMO**). DMO is heavily used in the industry. I do not know how the data-cube extrapolation code I am designing here would fit into DMO and stacking, but because these are such important processes, the appearance of a fundamentally new tool like this should be of interest. It is curious that the DMO operator is traditionally derived from theory, and the theory requires the unknown velocity function of depth, whereas here I propose estimating the offset continuation operator directly from the data itself, without the need of a velocity model.

Obviously, one application is to extrapolate off the sides of a **constant-offset section**. This would reduce migration semicircles at the survey's ends.

Another application is to extrapolate off the **cable ends** of a common-midpoint gather or a common shot point gather. This could enhance the prediction of multiple reflections or reduce artifacts in velocity analysis.

Obviously, the methodology and code in this chapter is easily extendable to four dimensions (prestack 3-D data).

7.6 TWO 1-D PEFS VERSUS ONE 2-D PEF

Here we look at the difference between using two 1-D PEFs, and one 2-D PEF. Figure 1.6 shows an example of sparse tracks; it is not realistic in the upper-left corner (where it will be used for testing), in a quarter-circular disk where the data covers the model densely. Such a dense region is ideal for determining the 2-D PEF. Indeed, we cannot determine a 2-D PEF from the sparse data lines, because at any place you put the filter (unless there are enough adjacent data lines), unknown filter coefficients will multiply missing data. So every fitting goal is nonlinear and hence abandoned by the algorithm.

The set of test data shown in Figure 1.6 is a superposition of three functions like plane waves. One plane wave looks like low-frequency horizontal layers. Notice that the various layers vary in strength with depth. The second wave is dipping about 30° down to the right and its waveform is perfectly sinusoidal. The third wave dips down 45° to the left and its waveform is bandpassed random noise like the horizontal beds. These waves will be handled differently by different processing schemes, so I hope you can identify all three. If you have difficulty, view the figure at a grazing angle from various directions.

Later we will make use of the dense data region, but first let $\mathbf{U}$ be the east-west PE operator and $\mathbf{V}$ be the north-south operator and let the signal or image be $\mathbf{h} = h(x, y)$. The fitting residuals are

$$\begin{aligned} \mathbf{0} &\approx (\mathbf{I} - \mathbf{J})(\mathbf{h} - \mathbf{d}) \\ \mathbf{0} &\approx \mathbf{U}\,\mathbf{h} \\ \mathbf{0} &\approx \mathbf{V}\,\mathbf{h} \end{aligned} \tag{7.19}$$

where $\mathbf{d}$ is data (or binned data) and $(\mathbf{I} - \mathbf{J})$ masks the map onto the data.

Figure 1.7 shows the result of using a single one-dimensional PEF along either the vertical or the horizontal axis.

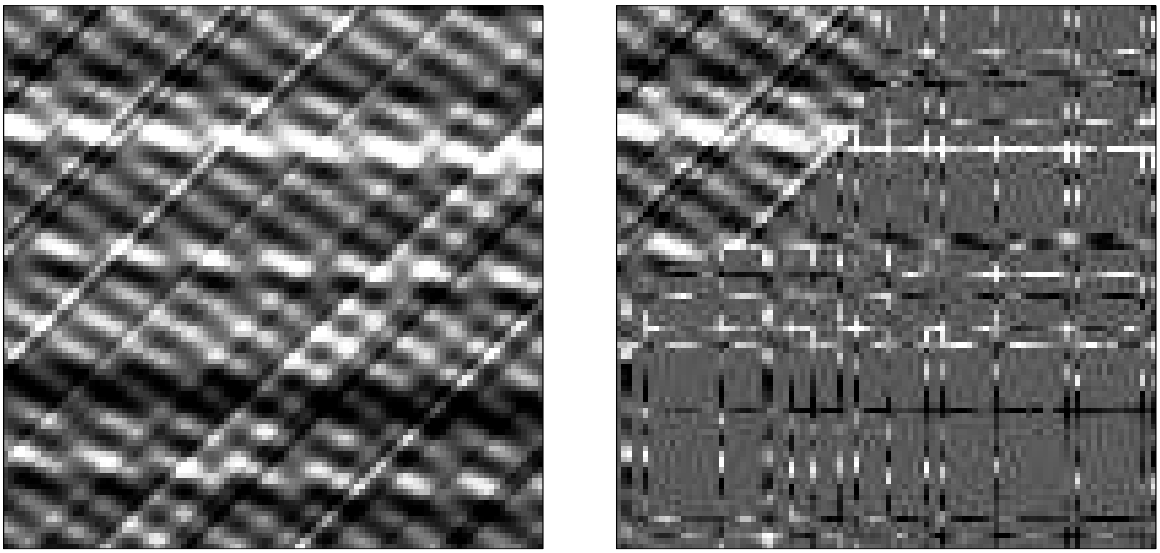

Figure 7.6: Synthetic wavefield (left) and as observed over survey lines (right). The wavefield is a superposition of waves from three directions. noiz-duelin90 [ER]

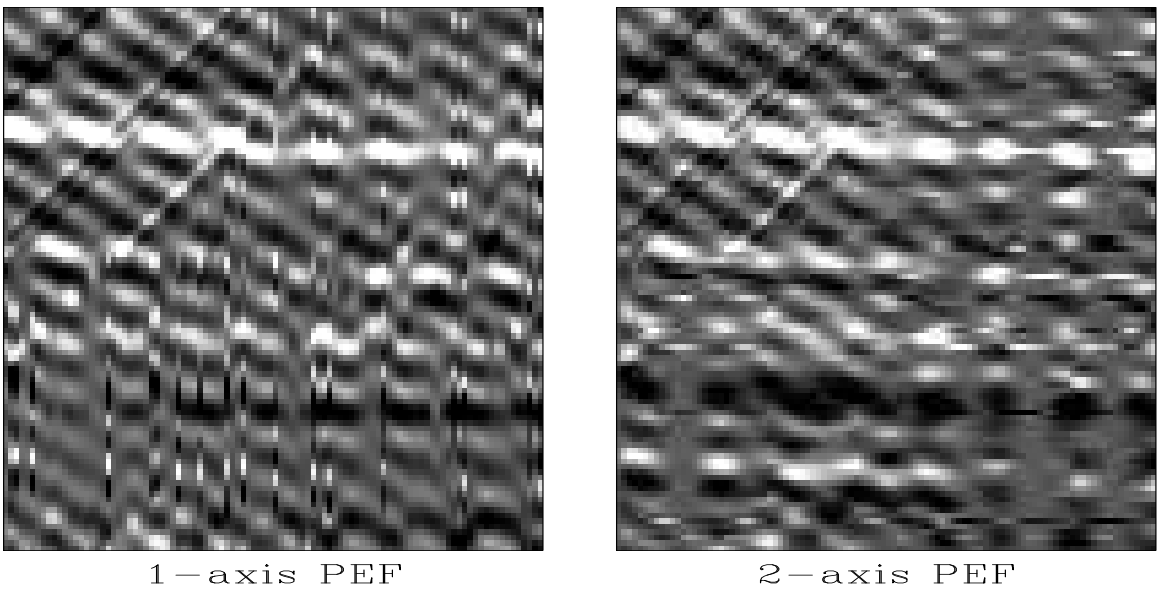

Figure 7.7: Interpolation by 1-D PEF along the vertical axis (left) and along the horizontal axis (right). noiz-dueleither90 [ER]

Figure 1.8 compares the use of a pair of 1-D PEFs versus a single 2-D PEF (which needs the "cheat" corner in Figure 1.6. Studying Figure 1.8 we conclude (what theory predicts) that

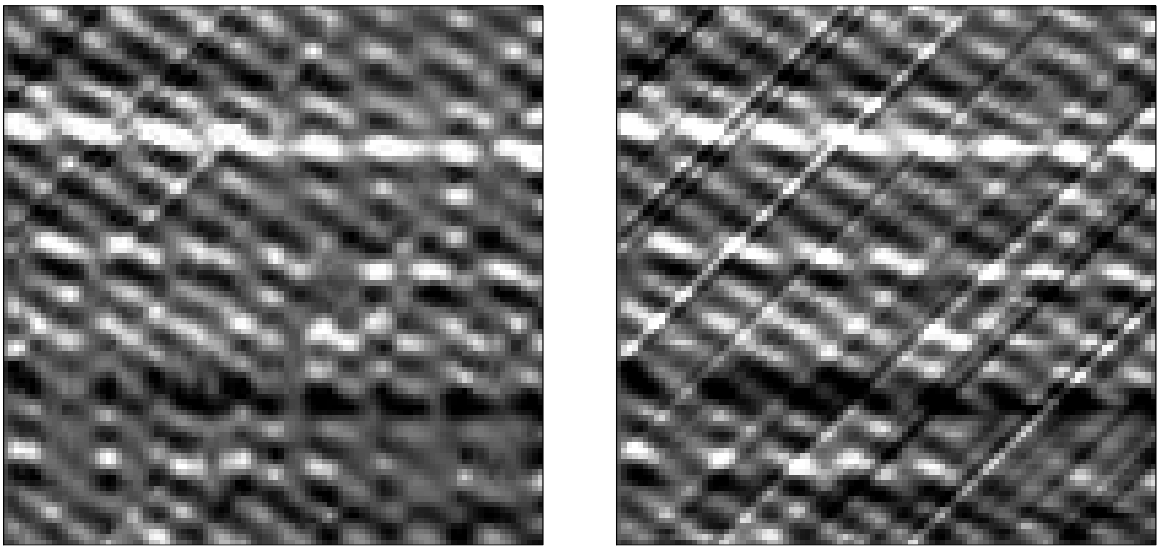

Figure 7.8: Data infilled by a pair of 1-D PEFs (left) and by a single 2-D PEF (right). noiz-duelversus90 [ER,M]

- These waves are predictable with a pair of 1-D filters:
  - Horizontal (or vertical) plane-wave with random waveform
  - Dipping plane-wave with a sinusoidal waveform
- These waves are predictable with a single 2-D filter:
  - both of the above
  - Dipping plane-wave with a random waveform

7.7 ALTITUDE OF SEA SURFACE NEAR MADAGASCAR

A satellite points a radar at the ground and receives echos we investigate here. These echos are recorded only over the ocean. The echo tells the distance from the orbit to the ocean surface. After various corrections are made for earth and orbit ellipticities the residual shows tides, wind stress on the surface, and surprisingly a signal proportional to the depth of the water. Gravity of mountains on the water bottom pulls water towards them raising sea level there.

The raw data investigated here[2] had a strong north-south tilt which I[3] removed at the outset.

[2]I wish to thank David T. Sandwell http://topex.ucsd.edu/ for providing me with this subset of satellite altimetry data, commonly known as Topex-Posidon data.

[3]The calculations here were all done for us by Jesse Lomask.

Figure 1.9 gives our first view of altimetry data (ocean height) from southeast of the island of Madagascar. About all we can see is satellite tracks. The satellite is in a circular polar orbit. To

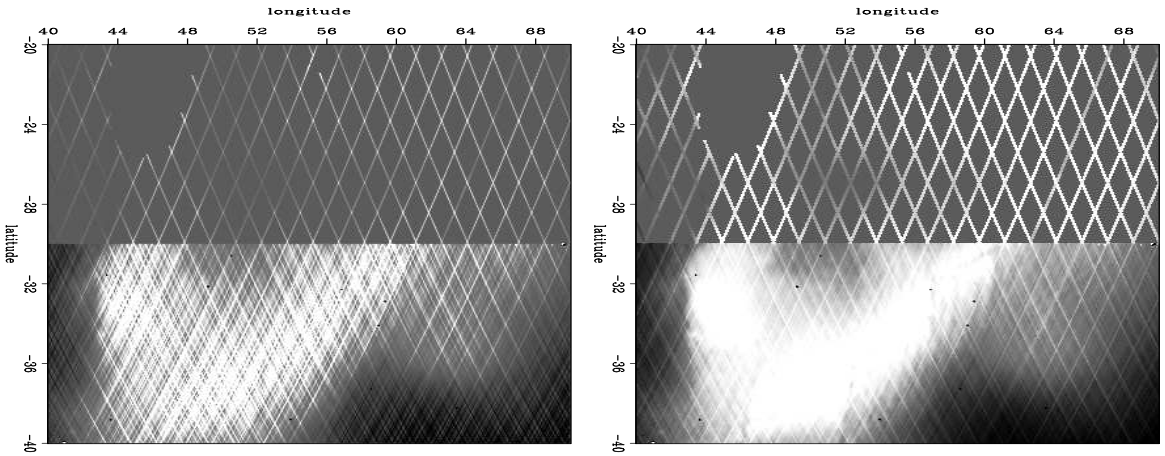

Figure 7.9: Sea height under satellite tracks. The island of Madagascar is in the empty area at $(46°, -22°)$. Left is the adjoint $\mathbf{L}'\mathbf{d}$. Right is the adjoint normalized by the bin count, $\mathbf{diag}(\mathbf{L}'\mathbf{1})^{-1}\mathbf{L}'\mathbf{d}$. You might notice a few huge, bad data values. Overall, the topographic function is too smooth, suggesting we need a roughener. noiz-jesse1 [ER,M]

us the sun seems to rotate east to west as does the circular satellite orbit. Consequently, when the satellite moves northward across the site we get altitude measurements along a SE-NW line. When it moves southward we get measurements along a NE-SW line. This data is from the cold war era. At that time dense data above the $-30°$ parallel was secret although sparse data was available. (The restriction had to do with precision guidance of missiles. Would the missile hit the silo? or miss it by enough to save the retaliation missile?)

Here are some definitions: Let components of $\mathbf{d}$ be the data, altitude measured along a satellite track. The model space is $\mathbf{h}$, altitude in the (x, y)-plane. Let $\mathbf{L}$ denote the 2-D linear interpolation operator from the track to the plane. Let $\mathbf{H}$ be the helix derivative, a filter with response $\sqrt{k_x^2 + k_y^2}$. Except where otherwise noted, the roughened image $\mathbf{p}$ is the preconditioned variable $\mathbf{p} = \mathbf{H}\mathbf{h}$. The derivative along a track in data space is $\frac{d}{dt}$. A weighting function that vanishes when any filter hits a track end or a bad data point is $\mathbf{W}$.

Figure 1.10 shows the entire data space, over a half million data points (actually 537974). Altitude is measured along many tracks across the image. In Figure 1.10 the tracks are placed end-to-end, so it is one long vector (displayed in about 50 signal rows). A vector of equal length is the missing data marker vector. This vector is filled with zeros everywhere except where data is missing or known bad or known to be at the ends of the tracks. The long tracks are the ones that are sparse in the north.

Figure 1.11 brings this information into model space. Applying the adjoint of the linear interpolation operator $\mathbf{L}'$ to the data $\mathbf{d}$ gave our first image $\mathbf{L}'\mathbf{d}$ in model space in Figure 1.9. The track noise was so large that roughening it made it worse. A more inviting image arose when I normalized the image before roughening it. Put a vector of all ones $\mathbf{1}$ into the adjoint of

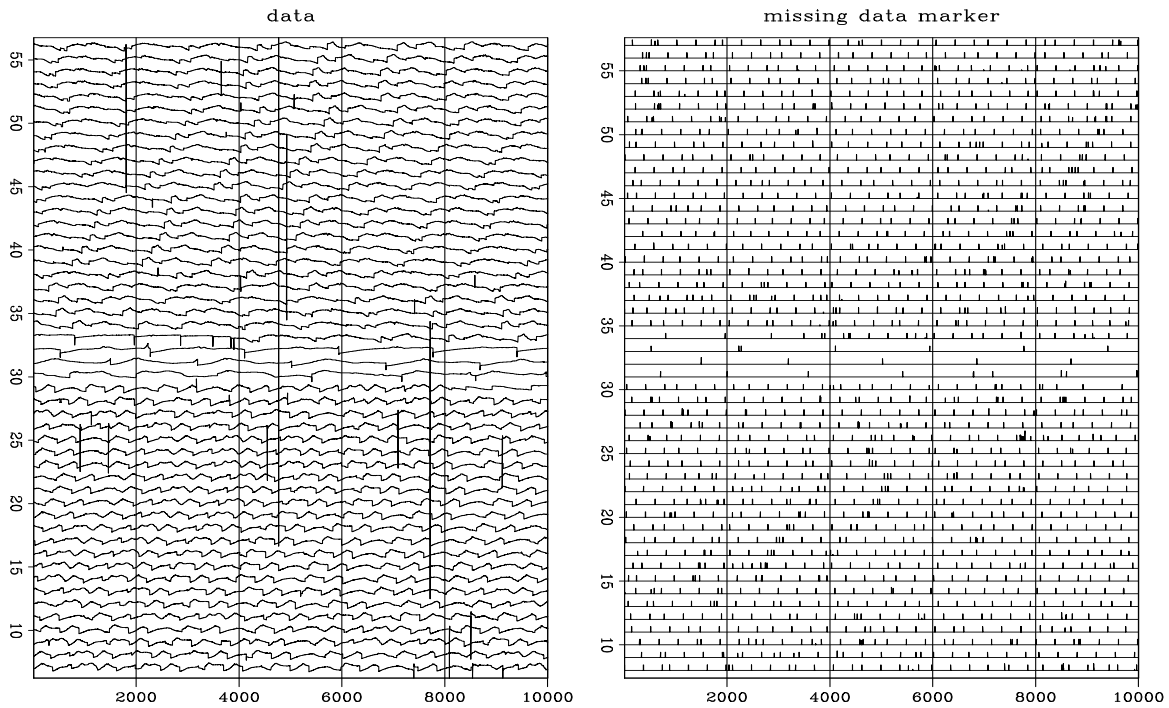

Figure 7.10: All the data **d** and the missing data markers. noiz-jesse5 [ER,M]

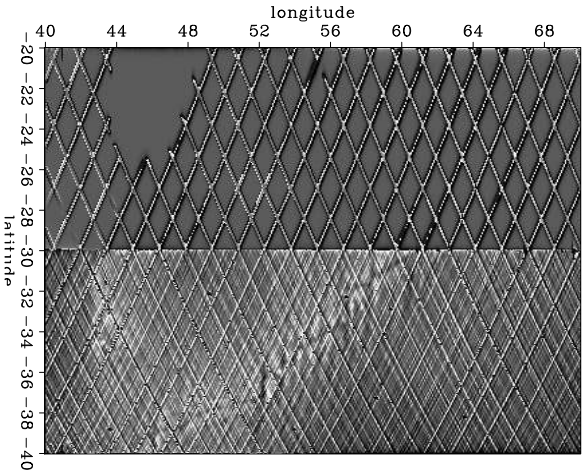

Figure 7.11: The roughened, normalized adjoint, $\mathbf{H}\ \mathbf{diag}(\mathbf{L}'\mathbf{1})^{-1}\mathbf{L}'\mathbf{d}$. Some topography is perceptible through a maze of tracks. noiz-jesse2 [ER,M]

the linear interpolation operator $\mathbf{L}'$. What comes out $\mathbf{L}'\mathbf{1}$ is roughly the number of data points landing in each pixel in model space. More precisely, it is the sum of the linear interpolation weights. This then, if it is not zero, is used as a divisor. The division accounts for several tracks contributing to one pixel. In matrix formalism this image is $\mathbf{diag}(\mathbf{L}'\mathbf{1})^{-1}\mathbf{L}'\mathbf{d}$. In Figure 1.11 this image is roughened with the helix derivative $\mathbf{H}$.

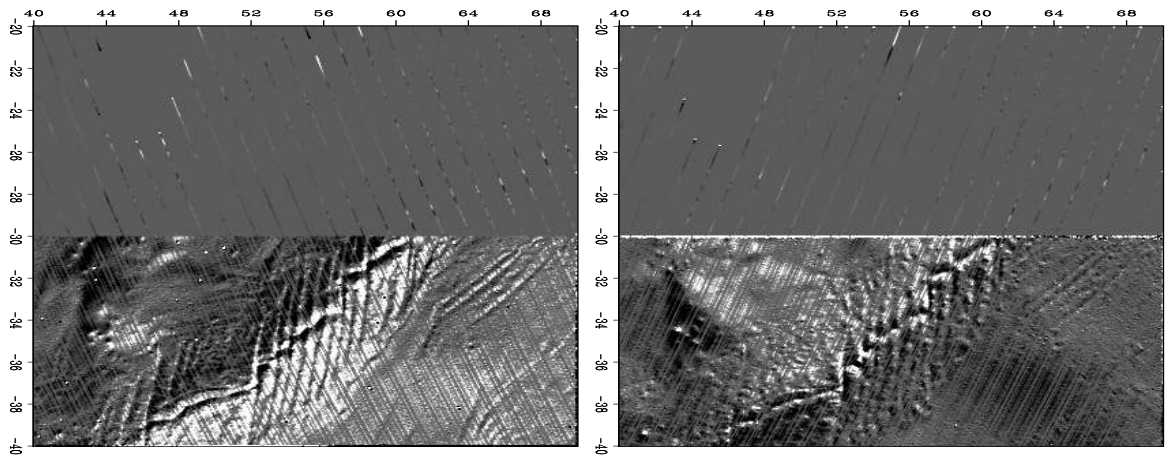

Figure 7.12: With a simple roughening derivative in data space, model space shows two nice topographic images. Let $\mathbf{n}$ denote ascending tracks. Let $\mathbf{s}$ denote descending tracks. Left is $\mathbf{L}'\frac{d}{dt}\mathbf{n}$. Right is $\mathbf{L}'\frac{d}{dt}\mathbf{s}$. noiz-jesse3 [ER,M]

There is a simple way here to make a nice image—roughen along data tracks. This is done in Figure 1.12. The result is two attractive images, one for each track direction. Unfortunately, there is no simple relationship between the two images. We cannot simply add them because the shadows go in different directions. Notice also that each image has noticeable tracks that we would like to suppress further.

A geological side note: The strongest line, the line that marches along the image from southwest to northeast is a sea-floor spreading axis. Magma emerges along this line as a source growing plates that are spreading apart. Here the spreading is in the north-south direction. The many vertical lines in the image are called "transform faults".

Fortunately, we know how to merge the data. The basic trick is to form the track derivative not on the data (which would falsify it) but on the residual which (in Fourier space) can be understood as choosing a different weighting function for the statistics. A track derivative on the residual is actually two track derivatives, one on the observed data, the other on the modeled data. Both data sets are changed in the same way. Figure 1.13 shows the result. The altitude function remains too smooth for nice viewing by variable brightness, but roughening it with $\mathbf{H}$ makes an attractive image showing, in the south, no visible tracks.

The north is another story. We would like the sparse northern tracks to contribute to our viewing pleasure. We would like them to contribute to a northern image of the earth, not to an image of the data acquisition footprint. This begins to happen in Figure 1.14. The process of fitting data by choosing an altitude function $\mathbf{h}$ would normally include some regularization

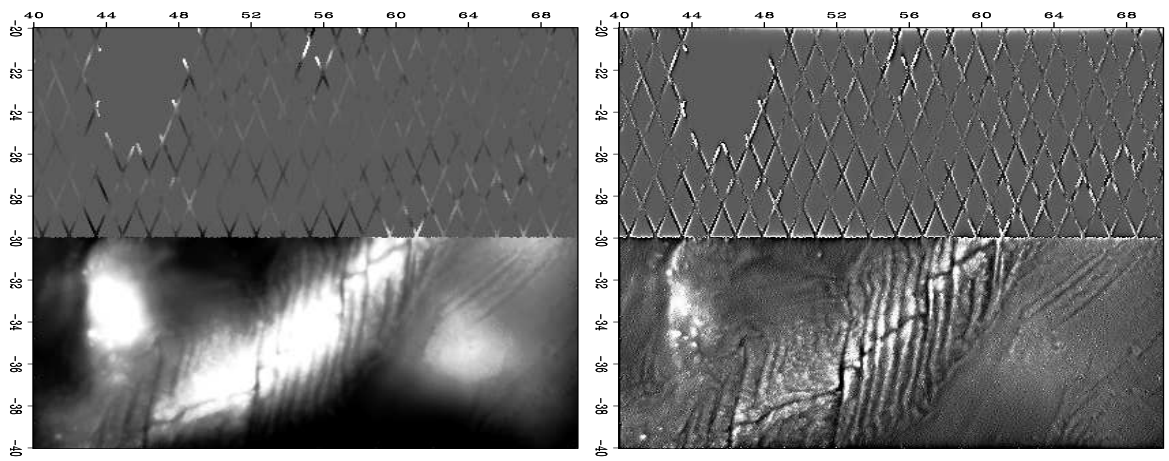

Figure 7.13: All data merged into a track-free image (hooray!) by applying the track derivative, not to the data, but to the residual. Left is $\mathbf{h}$ estimated by $\mathbf{0} \approx \mathbf{W}\frac{d}{dt}(\mathbf{Lh}-\mathbf{d})$. Right is the roughened altitude, $\mathbf{p}=\mathbf{Hh}$. noiz-jesse10 [ER,M]

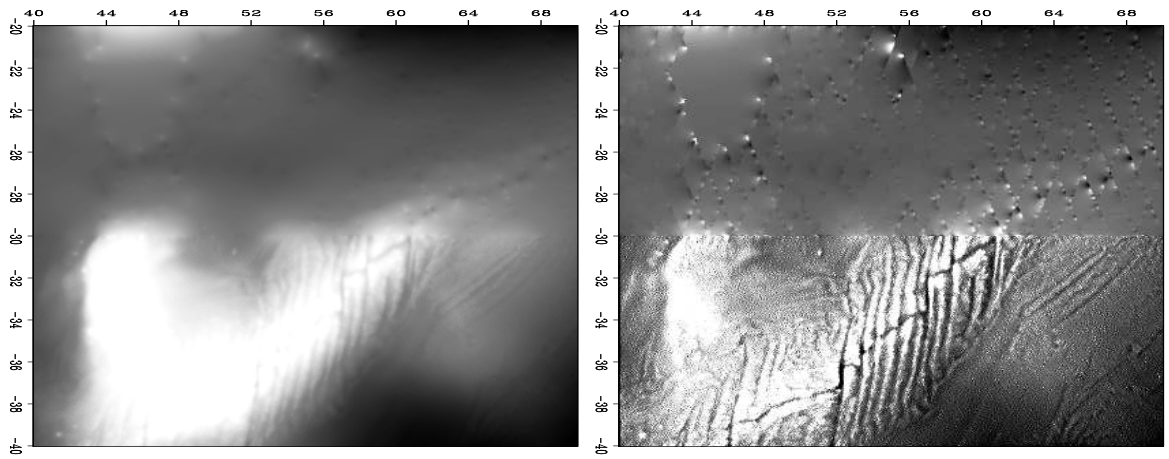

Figure 7.14: Using the track derivative in residual space and helix preconditioning in model space we start building topography in the north. Left is $\mathbf{h}=\mathbf{H}^{-1}\mathbf{p}$ where $\mathbf{p}$ is estimated by $\mathbf{0} \approx \mathbf{W}\frac{d}{dt}(\mathbf{LH}^{-1}\mathbf{p}-\mathbf{d})$ for only 10 iterations. Right is $\mathbf{p}=\mathbf{Hh}$. noiz-jesse8 [ER,M]

(model styling), such as $\mathbf{0} \approx \nabla \mathbf{h}$. Instead we adopt the usual trick of changing to preconditioning variables, in this case $\mathbf{h} = \mathbf{H}^{-1}\mathbf{p}$. As we iterate with the variable $\mathbf{p}$ we watch the images of $\mathbf{h}$ and $\mathbf{p}$ and quit either when we are tired, or more hopefully, when we are best satisified with the image. This subjective choice is rather like choosing the ϵ that is the balance between data fitting goals and model styling goals. The result in Figure 1.14 is pleasing. We have begun building topography in the north that continues in a consistant way with what is in the south. Unfortunately, this topography does fade out rather quickly as we get off the data acquisition tracks.

If we have reason to suspect that the geological style north of the 30th parallel matches that south of it (the stationarity assumption) we can compute a PEF on the south side and use it for interpolation on the north side. This is done in Figure 1.15. This is about as good as we

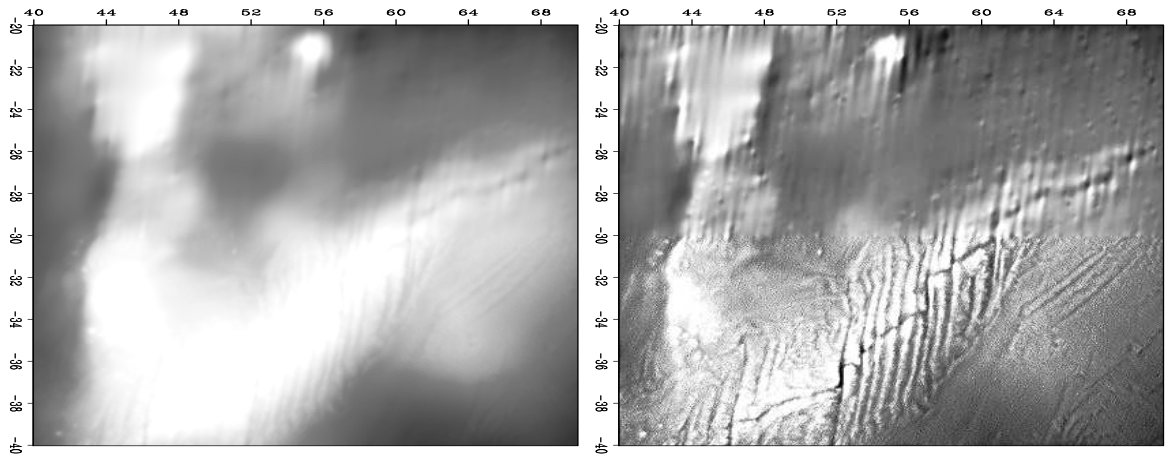

Figure 7.15: Given a PEF $\mathbf{A}$ estimated on the densely defined southern part of the model, $\mathbf{p}$ was estimated by $\mathbf{0} \approx \mathbf{W}\frac{d}{dt}(\mathbf{L}\mathbf{A}^{-1}\mathbf{p} - \mathbf{d})$ for 50 iterations. Left is $\mathbf{h} = \mathbf{A}^{-1}\mathbf{p}$. Right is $\mathbf{p} = \mathbf{H}\mathbf{h}$. noiz-jesse9 [ER,M]

are going to get. Our fractured ridge continues nicely into the north. Unfortunately, we have imprinted the fractured ridge texture all over the northern space, but that's the price we must pay for relying on the stationarity assumption.

The fitting residuals are shown in Figure 1.16. The physical altitude residuals tend to be rectangles, each the duration of a track. While the satellite is overflying other earth locations the ocean surface is changing its altitude. The fitting residuals (right side) are very fuzzy. They appear to be "white", though with ten thousand points crammed onto a line a couple inches long, we cannot be certain. We could inspect this further. If the residuals turn out to be significantly non-white, we might do better to change $\frac{d}{dt}$ to a PEF along the track.

7.8 ELIMINATING NOISE AND SHIP TRACKS IN GALILEE

The Sea of Galilee data set has enchanted my colleagues and me because the data has comprehensible defects that have frustrated many of our image estimation designs. The best results

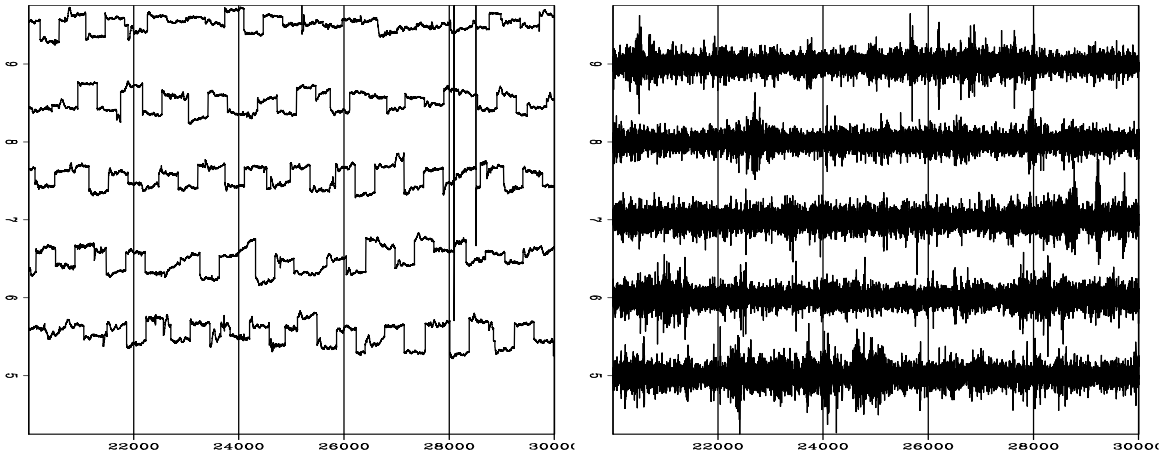

Figure 7.16: The residual at fifty thousand of the half million (537,974) data points in Figure 1.15. Left is physical residual $\mathbf{L}\mathbf{A}^{-1}\mathbf{p}-\mathbf{d}$. Right is fitting residual $\mathbf{W}\frac{d}{dt}(\mathbf{L}\mathbf{A}^{-1}\mathbf{p}-\mathbf{d})$. noiz-jesse9_res [ER,M]

found so far were prepared for us here by Antoine Guitton based on our 2004 Geophysics paper.

We are given depth-sounding data from the Sea of Galilee. 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. The ultimate goal is to produce a good map of the depth to bottom, and images useful for identifying archaeological, geological, and geophysical details of the water bottom. In particular, we hope to identify some ancient shorelines around the lake and meaningful geological features inside the lake. The ancient shorelines might reveal early settlements of archeological interest or old fishing ports. The pertinence of this data set to our daily geophysical problems is threefold: (1) We often need to interpolate irregular data. (2) The data has noise bursts of various types. (3) The data has systematic error (drift) which tends to leave data-acquisition tracks in the resulting image.

The Galilee data set was introduced in chapter 3 and recently plotted in Figure 1.4. Actually, that figure is a view of 2-D model space. One of the first things I learned (the hard way) is the importance of viewing both the model space and the residuals in data space.

Be sure to plot both model space and data space. You should try to understand the results in both spaces and might like to watch movies of each as the iteration progresses.

The raw data (Figure 1.17), is distributed irregularly across the lake surface. It is 132,044 triples (x_i, y_i, z_i), where x_i ranges over about 12 km, where y_i ranges over about 20 km, and z_i is depth in multiples of 10 cm. (It would have been helpful if a fourth value had been included, the clock-date time t_i, of the measurement.) The ship surveyed a different amount of distance every day of the survey. Figure 1.17 displays the whole survey as one long track. On one traverse across the lake, the depth record is U shaped. A few V shaped tracks result from deep-water vessel turn arounds. All depth values (data points) used for building the final

map are shown here. Each point corresponds to one depth measurement inside the lake. For display convenience, the long signal is broken into 23 strips of 5718 depth measurements. We have no way to know that sometimes the ship stops a little while with the data recorder running; sometimes it shuts down overnight or longer; but mostly it progresses at some unknown convenient speed. So the horizontal axis in data space is a measurement number that scales in some undocumented way to distance along the track.

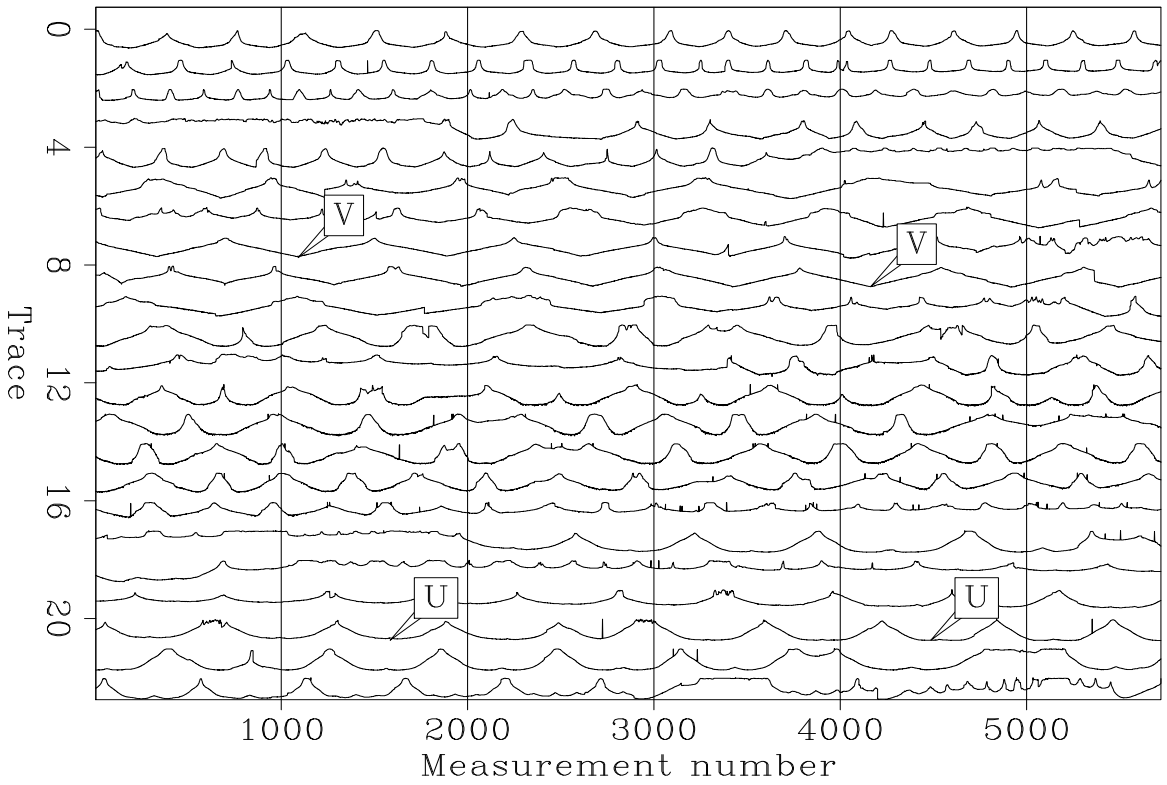

Figure 7.17: The complete Galilee data space. noiz-antoine1 [ER]

7.8.1 Attenuation of noise bursts and glitches

Let $\mathbf{h}$ be an abstract vector containing as components the water depth over a 2-D spatial mesh. Let $\mathbf{d}$ be an abstract vector whose successive components are depths along the vessel tracks. One way to grid irregular data is to minimize the length of the residual vector $\mathbf{r}_d(\mathbf{h})$:

$$\mathbf{0} \quad \approx \quad \mathbf{r}_d \quad = \quad \mathbf{B}\mathbf{h} - \mathbf{d} \tag{7.20}$$

where $\mathbf{B}$ is a 2-D linear interpolation (or binning) operator and $\mathbf{r}_d$ is the data residual. Where tracks cross or where multiple data values end up in the same bin, the fitting goal (1.20) takes an average. Figure 1.4 is a display of simple binning of the raw data. (Some data points are outside the lake. These must represent navigation errors.)

Some model-space bins will be empty. For them we need an additional "model styling" goal, i.e. regularization. For simplicity we might minimize the gradient.

$$\begin{array}{rcrcl} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{B}\mathbf{h} - \mathbf{d} \\ \mathbf{0} & \approx & \mathbf{r}_h & = & \epsilon \nabla \mathbf{h} \end{array} \tag{7.21}$$

where $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$ and $\mathbf{r}_h$ is the model space residual. Choosing a large scaling factor ϵ will tend to smooth our entire image, not just the areas of empty bins. We would like ϵ to be any number small enough that its main effect is to smooth areas of empty bins. When we get into this further, though, we'll see that because of noise some smoothing across the nonempty bins is desireable too.

7.8.2 Preconditioning for accelerated convergence

As usual we precondition by changing variables so that the regularization operator becomes an identity matrix. The gradient ∇ in equation (1.21) has no inverse, but its spectrum $-\nabla'\nabla$, can be factored ($-\nabla'\nabla = \mathbf{H}'\mathbf{H}$) into triangular parts $\mathbf{H}$ and $\mathbf{H}'$ where $\mathbf{H}$ is the helix derivative. This $\mathbf{H}$ is invertible by deconvolution. The quadratic form $\mathbf{h}'\nabla'\nabla\mathbf{h} = \mathbf{h}'\mathbf{H}'\mathbf{H}\mathbf{h}$ suggests the new preconditioning variable $\mathbf{p} = \mathbf{H}\mathbf{h}$. The fitting goals in equation (1.21) thus become

$$\begin{array}{rcrcl} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{B}\mathbf{H}^{-1}\mathbf{p} - \mathbf{d} \\ \mathbf{0} & \approx & \mathbf{r}_p & = & \epsilon \mathbf{p} \end{array} \tag{7.22}$$

with $\mathbf{r}_p$ the residual for the new variable $\mathbf{p}$. Experience shows that an iterative solution for $\mathbf{p}$ converges much more rapidly than an iterative solution for $\mathbf{h}$, thus showing that $\mathbf{H}$ is a good choice for preconditioning. We could view the estimated final map $\mathbf{h} = \mathbf{H}^{-1}\mathbf{p}$, however in practice, because the depth function is so smooth, we usually prefer to view the roughened depth $\mathbf{p}$.

There is no simple way of knowing beforehand the best value of ϵ. Practitioners like to see solutions for various values of ϵ. Practical exploratory data analysis is pragmatic. Without a simple, clear theoretical basis, analysts generally begin from $\mathbf{p} = \mathbf{0}$ and then abandon the fitting goal $\mathbf{0} \approx \mathbf{r}_p = \epsilon\mathbf{p}$. Effectively, they take $\epsilon = 0$. Then they examine the solution as a function of iteration, imagining that the solution at larger iterations corresponds to smaller ϵ and that the solution at smaller iterations corresponds to larger ϵ. In all our explorations, we follow this approach and omit the regularization in the estimation of the depth maps. Having achieved the general results we want, we should include the parameter ϵ and adjust it until we see a pleasing result at an "infinite" number of iterations. We should but usually we do not.

7.8.3 ℓ^1 norm

Spikes and erratic noise glitches can be suppressed with an approximate ℓ^1 norm. One main problem with the Galilee data is the presence of outliers in the middle of the lake and at the track ends. We could attenuate these spikes by editing or applying running median filters.

However, the former involves human labor while the latter might compromise small details by smoothing and flattening the signal. Here we formulate the estimation to eliminate the drastic effect of the noise spikes. We introduce a weighting operator that deemphasizes high residuals as follows:

$$\begin{array}{rcccl} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{W}(\mathbf{B}\mathbf{H}^{-1}\mathbf{p}-\mathbf{d}) \\ \mathbf{0} & \approx & \mathbf{r}_p & = & \epsilon\mathbf{p} \end{array} \tag{7.23}$$

with a diagonal matrix $\mathbf{W}$:

$$\mathbf{W} = \mathbf{diag}\left(\frac{1}{(1+r_i^2/\bar{r}^2)^{1/4}}\right) \tag{7.24}$$

where r_i is the residual for one component of $\mathbf{r}_d$ and $\bar{r}$ is a prechosen constant. This weighting operator ranges from ℓ^2 to ℓ^1, depending on the constant $\bar{r}$. We take $\bar{r} = 10$ cm because the data was given to us as integer multiples of 10 cm. (A somewhat larger value might be more appropriate).

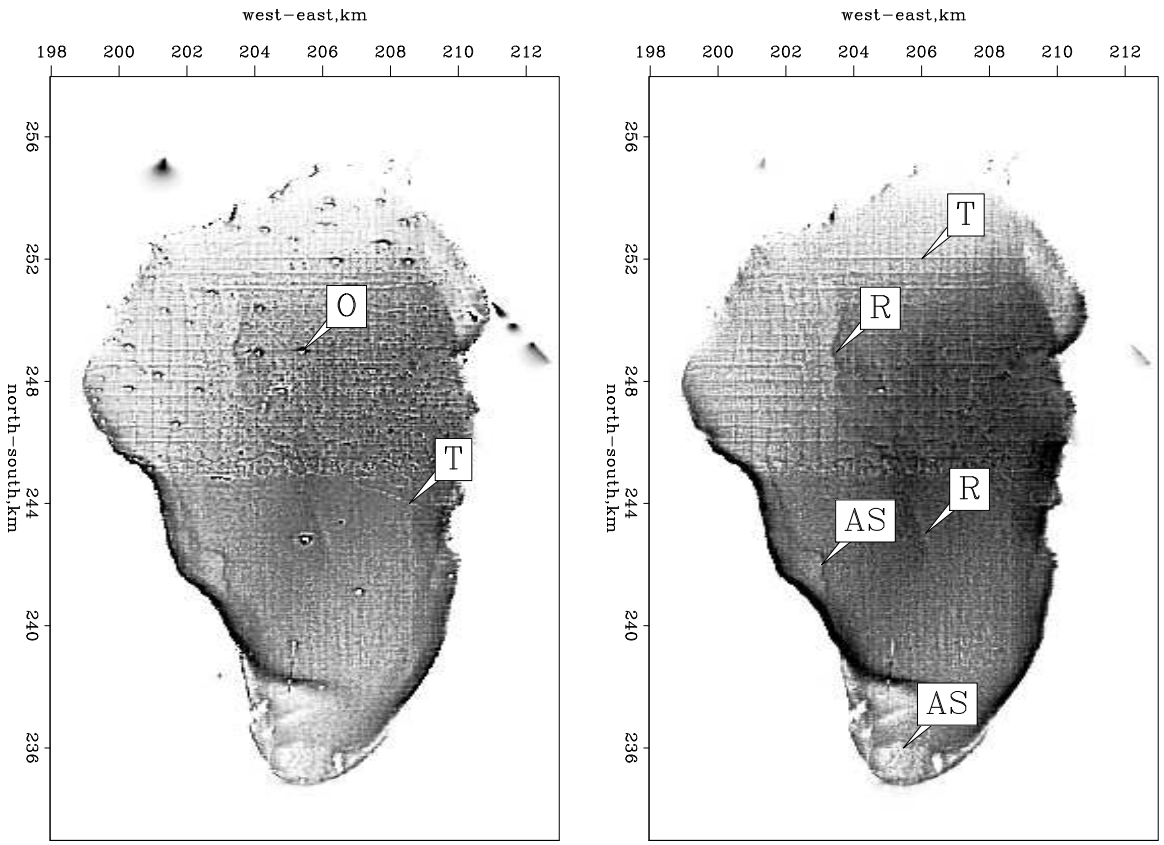

Figure 7.18: Estimated $\mathbf{p}$ in a least-squares sense (left) and in an ℓ^1 sense (right). Pleasingly, isolated spikes are attenuated. Some interesting features are shown by the arrows: AS points to few ancient shores, O points to some outliers, T points to few tracks, and R points to a curious feature. noiz-antoine2 [ER,M]

Figure 1.18 displays $\mathbf{p}$ estimated in a least-squares sense on the left and in a ℓ^1 sense on the right (equation (1.23) with a small $\bar{r}$). Most of the glitches are no longer visible. One obvious

glitch remains near $(x, y) = (205, 238)$. Evidently a north-south track has a long sequence of biased measurements that our ℓ^1 cannot overcome. Some ancient shorelines in the western and southern parts of the Sea of Galilee are now easier to identify (shown as AS). We also start to see a valley in the middle of the lake (shown as R). Data outside the lake (navigation errors) have been mostly removed. Data acquisition tracks (mostly north-south lines and east-west lines, one of which is marked with a T) are even more visible after the suppression of the outliers.

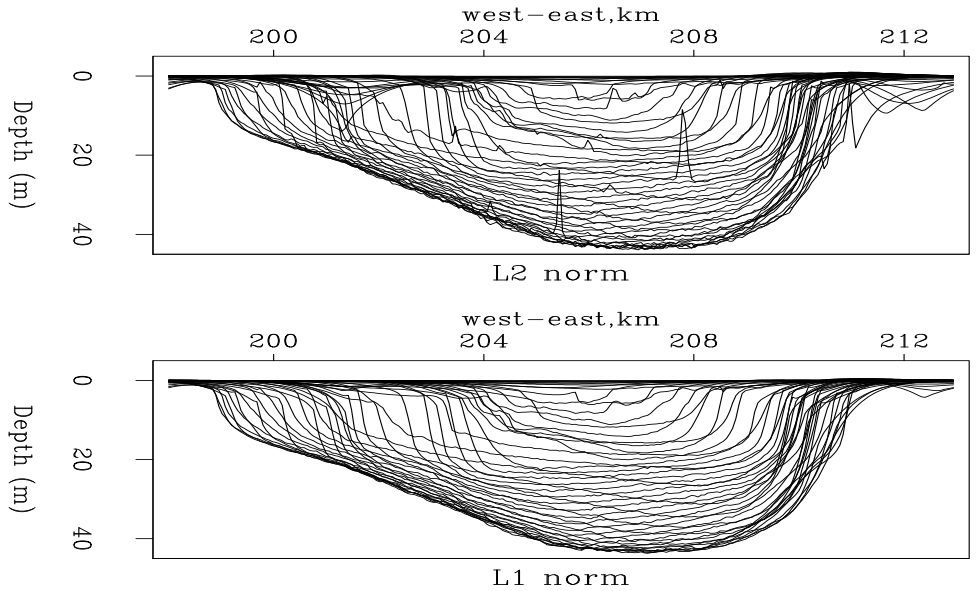

Figure 7.19: East-west cross sections of the lake bottom ($\mathbf{h} = \mathbf{H}^{-1}\mathbf{p}$). Top with the ℓ^2 solution. Bottom with the ℓ^1 approximating procedure. noiz-antoine3 [ER,M]

Figure 1.19 shows the bottom of the Sea of Galilee ($\mathbf{h} = \mathbf{H}^{-1}\mathbf{p}$) with ℓ^2 (top) fitting and ℓ^1 (bottom) fitting. Each line represents one east-west transect, transects at half-kilometer intervals on the north-south axis. The ℓ^1 result is a nice improvement over the ℓ^2 maps. The glitches inside and outside the lake have mostly disappeared. Also, the ℓ^1 norm gives positive depths everywhere. Although not visible everywhere in all the figures, topography is produced outside the lake. Indeed, the effect of regularization is to produce synthetic topography, a natural continuation of the lake floor surface.

We are now halfway to a noise-free image. Figure 1.18 shows that vessel tracks overwhelm possible fine scale details. Next we investigate a strategy based on the idea that the inconsistency between tracks comes mainly from different human and seasonal conditions during the data acquisition. Since we have no records of the weather and the time of the year the data were acquired we presume that the depth differences between different acquisition tracks must be small and relatively smooth along the super track.

7.8.4 Abandoned strategy for attenuating tracks

An earlier strategy to remove the ship tracks is to filter the residual as follows:

$$\begin{array}{rcrcl} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{W}\frac{d}{ds}(\mathbf{B}\mathbf{H}^{-1}\mathbf{p}-\mathbf{d}), \\ \mathbf{0} & \approx & \mathbf{r}_p & = & \epsilon\mathbf{p}, \end{array} \tag{7.25}$$

where $\frac{d}{ds}$ is the derivative along the track. The derivative removes the drift from the field data (and the modeled data). An unfortunate consequence of the track derivative is that it creates more glitches and spiky noise at the track ends and at the bad data points. Several students struggled with this idea without good results.

One explanation (of unknown validity) given for the poor results is that perhaps the numerical conditioning of the algebraic problem is worsened by the operators $\mathbf{W}$, $\frac{d}{ds}$, $\mathbf{B}$, and $\mathbf{H}^{-1}$, drastically slowing the convergence. Another explanation is that the operator $\frac{d}{ds}$ is too simple. Perhaps we should have a five or ten point low-cut filter—or maybe a PEF. A PEF could be estimated from the residual itself. Unfortunately, such a longer filter would smear the bad effect of noise glitches onto more residuals, effectively spoiling more measurements.

We concluded that the data is bad only in a very low frequency sense. Perhaps the lake is evaporating, or it is raining, or the load in the boat has been changed or shifted. It's a fact that any very low-frequency reject filter is necessarily a long filter, and that means that it must catch many noise spikes. Thus we should not attempt to filter out the drift from the residual. Instead we should model the drift.

> In the presence of both noise bursts and noise with a sensible spectrum (systematic noise), the systematic noise should be modeled while the noise bursts should be handled with ℓ^1.

7.8.5 Modeling data acquisition drift

To model data drift we imagine a vector $\mathbf{q}$ of random numbers that will be passed thru a low-pass filter (like a leaky integrator) $\mathbf{L}$. The modeled data drift is $\mathbf{L}\mathbf{q}$. We will solve for $\mathbf{q}$. A price we pay is an increase of the number of unknowns. Augmenting earlier fitting goals (1.23) we have:

$$\begin{array}{rcrcl} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{W}(\mathbf{B}\mathbf{H}^{-1}\mathbf{p}+\lambda\mathbf{L}\mathbf{q}-\mathbf{d}), \\ \mathbf{0} & \approx & \mathbf{r}_p & = & \epsilon_1\mathbf{p}, \\ \mathbf{0} & \approx & \mathbf{r}_q & = & \epsilon_2\mathbf{q}, \end{array} \tag{7.26}$$

where $\mathbf{h} = \mathbf{H}^{-1}\mathbf{p}$ estimates the interpolated map of the lake, and where $\mathbf{L}$ is a drift modeling operator (leaky integration), $\mathbf{q}$ is an additional variable to be estimated, and λ is a balancing constant to be discussed. We then minimize the misfit function,

$$g_2(\mathbf{p},\mathbf{q}) = \|\mathbf{r}_d\|^2 + \epsilon_1^2\|\mathbf{r}_p\|^2 + \epsilon_2^2\|\mathbf{r}_q\|^2, \tag{7.27}$$

Now the data $\mathbf{d}$ is being modeled in two ways by two parts which add, a geological map part $\mathbf{B}\mathbf{H}^{-1}\mathbf{p}$ and a recording system drift part $\lambda\mathbf{L}\mathbf{q}$. Clearly, by adjusting the balance of ϵ_1 to ϵ_2

we are forcing the data to go one way or the other. There is nothing in the data itself that says which part of the theory should claim it.

It is a customary matter of practice to forget the two ϵs and play with the λ. If we kept the two ϵs, the choice of λ would be irrelevant to the final result. Since we are going to truncate the iteration, choice of λ matters. It chooses how much data energy goes into the equipment drift function and how much into topography. Antoine ended out with with $\lambda = 0.08$.

There is another parameter to adjust. The parameter ρ controlling the decay of the leaky integration. Antoine found that value $\rho = 0.99$ was a suitable compromise. Taking ρ smaller allows the track drift to vary too rapidly thus falsifying data in a way that falsifies topography. Taking ρ closer to unity does not allow adequately rapid variation of the data acquistion system thereby pushing acquisition tracks into the topography.

Figure 1.20 (bottom-left corner) shows the estimated roughened image $\mathbf{p}$ with $\lambda\mathbf{L}$ data-drift modeling and (top-left corner) $\mathbf{p}$ without it. Data-drift modeling (bottom-left) yields an image that is essentially track-free without loss of detail. Top right shows the poor result of applying the derivative $\frac{d}{ds}$ along the tracks. Tracks are removed but the topography is unclear.

The bottom-right part of Figure 1.20 provides important diagnostic information. The estimated instrumentation drift $\mathbf{Lq}$ has been transformed to model space $\mathbf{B}'\mathbf{Lq}$. We do not like to see hints of geology in this space but we do. Adjusting λ or ρ we can get rid of the geology here, but then survey tracks will appear in the lake image. The issue of decomposing data into signal and noise parts is dealt with further in chapter 9.

Figures 1.21 and 1.22 show selected segments of data space. Examining here the discrepancy between observed data and modeled data offers us an opportunity to get new ideas. The top plot is the input data $\mathbf{d}$. Next is the estimated noise-free data $\mathbf{BH}^{-1}\mathbf{p}$. Then the estimated secular variations $\lambda\mathbf{Lq}$. Finally residual $\mathbf{BH}^{-1}\mathbf{p} + \lambda\mathbf{Lq} - \mathbf{d}$ after a suitable number of iterations. The modeled data in both Figures 1.21b and 1.22b show no remaining spikes.

The estimated instrument drift is reasonable, mostly under a meter for measurments with a nominal precision of 10 cm. There are some obvious problems though. It is not a serious problem that the drift signal is always positive. Applying the track derivative means that zero frequency is in the null space. An arbitrary constant may be moved from water depth to track calibration. More seriously, the track calibration fluctuates more rapidly than we might imagine. Worse still, Figure 1.22c shows the instrument drift correlates with water depth(!). This suggests we should have a slower drift function (bigger ρ or weaker λ), but Antoine assures me that this would push data acquisition tracks into the lake image. If the data set had included the date-time of each measurement we would have been better able to model drift. Instead of allowing a certain small change of drift with each measurement, we could have allowed a small change in proportion to the time since the previous measurement.

An interesting feature of the data residual in Figure 1.22d is that it has more variance in deep water than in shallow. Perhaps the depth sounder has insufficient power for deeper water or for the softer sediments found in deeper water. On the other hand, this enhanced deep water variance is not seen in Figure 1.21d which is puzzling. Perhaps the sea was rough for one day of recording but not another.

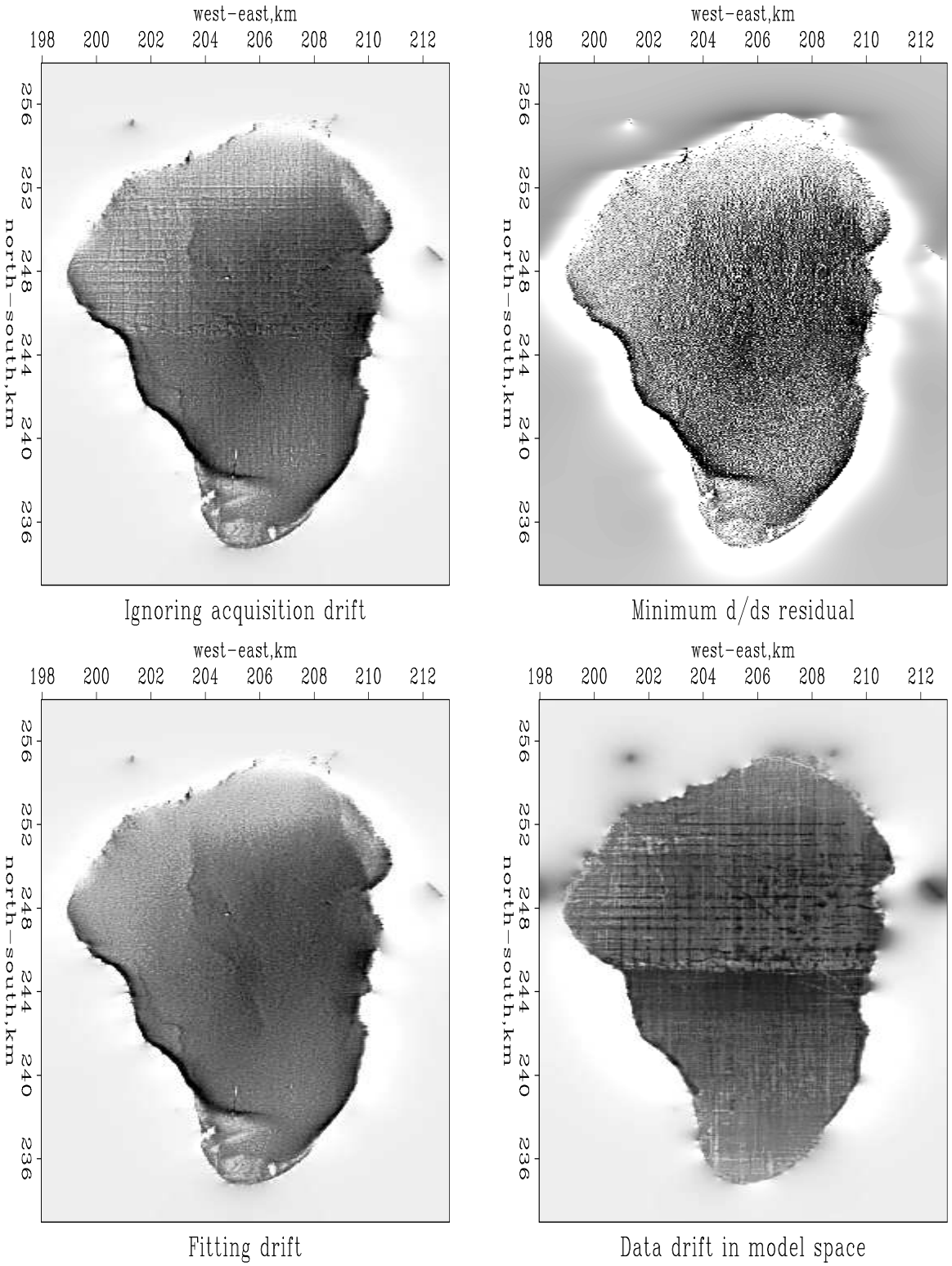

Figure 7.20: Top left: Estimated **p** without attenuation of the tracks, i.e., regression (1.23). Top right: Estimated **p** with the derivative along the tracks, i.e., regression (1.25). Bottom left: Estimated **p** without tracks, i.e., regression (1.26). Bottom right: recorder drift in model space $\mathbf{B}'\mathbf{L}\mathbf{q}$. noiz-antoine4 [ER,M]

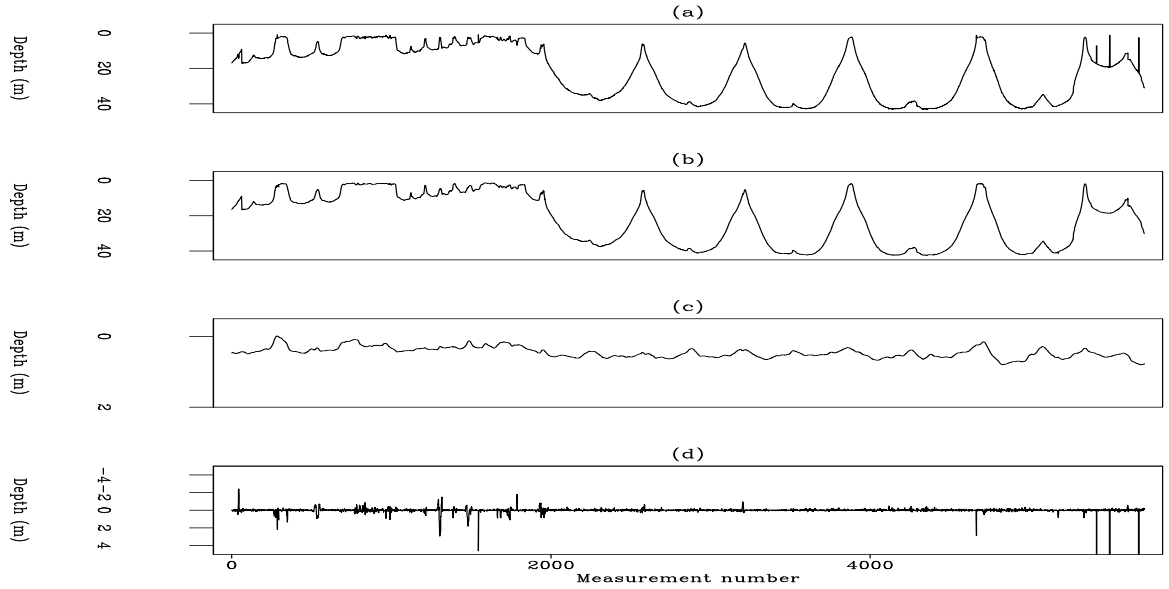

Figure 7.21: (a) Track 17 (input data) in Figure 1.17. (b) The estimated noise-free data $\mathbf{BH}^{-1}\mathbf{p}$. (c) Estimated drift $\mathbf{Lq}$. (d) Data residual. noiz-antoine5 [ER,M]

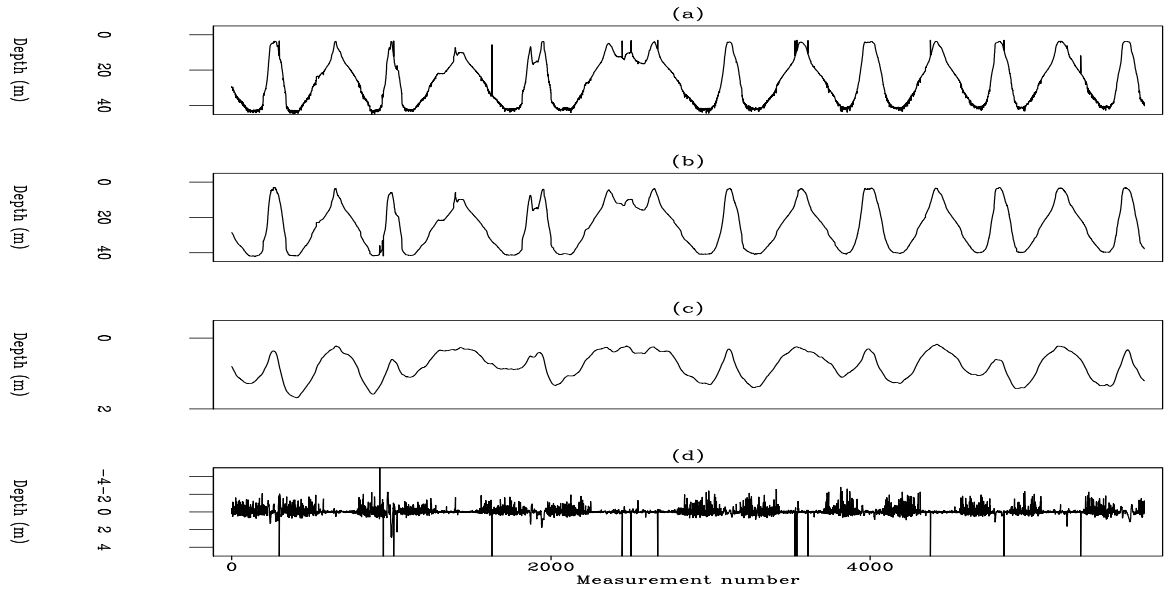

Figure 7.22: (a) Track 14 (input data) in Figure 1.17. (b) Modeled data, $\mathbf{BH}^{-1}\mathbf{p}$. (c) Estimated drift. (d) Data-space residual. noiz-antoine6 [ER]

7.8.6 Regridding

We often have an image $\mathbf{h}_0$ on a coarse mesh that we would like to use on a refined mesh. This regridding chore reoccurs on many occasions so I present reusable code. When a continuum is being mapped to a mesh, it is best to allocate to each mesh point an equal area on the continuum. Thus we take an equal interval between each point, and a half an interval beyond the end points. Given `n` points, there are `n-1` intervals between them, so we have

```
min = o - d/2
max = o + d/2 + (n-1)*d
```

which may be back solved to

```
d = (max-min)/n
o = (min*(n-.5) + max/2)/n
```

which is a memorable result for `d` and a less memorable one for `o`. With these not-quite-trivial results, we can invoke the linear interpolation operator `lint2`. It is designed for data points at irregular locations, but we can use it for regular locations too. Operator `refine2` defines pseudoirregular coordinates for the bin centers on the fine mesh and then invokes `lint2` to carry data values from the coarse regular mesh to the pseudoirregular finer mesh. Upon exiting from `refine2`, the data space (normally irregular) is a model space (always regular) on the finer mesh.

```
module refine2 { # Refine mesh.
# Input mm(m1,m2) is coarse.  Output dd(n1,n2) linear interpolated.
#
use lint2
real, dimension( :, :), pointer, private :: xy
#% _init( co1,cd1,co2,cd2, m1,m2, fo1,fd1,fo2,fd2, n1,n2)
  integer, intent( in)  :: m1,m2, n1,n2
  real,    intent( in)  :: co1,cd1,co2,cd2                # coarse grid
  real,    intent( out) :: fo1,fd1,fo2,fd2                # fine grid
  integer               :: i1,i2, id
  real                  :: xmin,xmax, ymin,ymax, x,y
  allocate (xy( n1*n2, 2))
  xmin = co1-cd1/2;   xmax = co1 +cd1*(m1-1) +cd1/2        # Great formula!
  ymin = co2-cd2/2;   ymax = co2 +cd2*(m2-1) +cd2/2
  fd1= (xmax-xmin)/n1;  fo1= (xmin*(n1-.5) + xmax/2)/n1    # Great formula!
  fd2= (ymax-ymin)/n2;  fo2= (ymin*(n2-.5) + ymax/2)/n2
  do i2=1,n2 {  y = fo2 + fd2*(i2-1)
  do i1=1,n1 {  x = fo1 + fd1*(i1-1)
        id = i1+n1*(i2-1)
        xy( id, :) = (/ x, y /)
        }}
  call lint2_init( m1,m2, co1,cd1, co2,cd2, xy)
#% _lop (mm, dd)
  integer stat1
  stat1 = lint2_lop( adj, .true., mm, dd)
```

```
#% _close
deallocate (xy)
}
```

Finally, here is the 2-D linear interpolation operator `lint2`, which is a trivial extension of the 1-D version `lint1` on page 16.

```
module lint2 {                                      # (Bi)Linear interpolation in 2-D
integer                         :: m1,m2
real                            :: o1,d1, o2,d2
real, dimension (:,:), pointer :: xy
#%  _init (     m1,m2, o1,d1, o2,d2, xy)
#%  _lop  ( mm (m1,m2), dd (:))
integer i, ix,iy, id
real    f, fx,gx, fy,gy
do id= 1, size(dd) {
   f = (xy(id,1)-o1)/d1; i=f; ix= 1+i; if( 1>ix .or. ix>=m1) cycle; fx=f-i; gx= 1.-fx
   f = (xy(id,2)-o2)/d2; i=f; iy= 1+i; if( 1>iy .or. iy>=m2) cycle; fy=f-i; gy= 1.-fy
               if( adj) {
                       mm(ix  ,iy  ) += gx * gy * dd(id)
                       mm(ix+1,iy  ) += fx * gy * dd(id)
                       mm(ix  ,iy+1) += gx * fy * dd(id)
                       mm(ix+1,iy+1) += fx * fy * dd(id)
                       }
               else
                       dd(id) = dd(id) + gx * gy * mm(ix  ,iy  ) +
                                         fx * gy * mm(ix+1,iy  ) +
                                         gx * fy * mm(ix  ,iy+1) +
                                         fx * fy * mm(ix+1,iy+1)
   }
}
```

Index

bursts, 10
bursty noise, 12

cable ends, 15
Cauchy function, 9
constant-offset section, 15

deburst, 10
despike, 10
dip moveout, 15
DMO, 15

fixbad module, 13

Hoare's algorithm, 3

irls module, 9

L-0 norm, 2
L-1 norm, 2
L-2 norm, 2
L1 or ℓ^1, 5
L2 or ℓ^2, 5
lint2 operator module, 33

mean, 1
median, 1
median smoothing, 10
mode, 1, 2
module
    `fixbad`, restore damaged data, 13
    `irls`, weighting functions for iterative reweighting, 9
    `pefest`, estimate PEF in 1-D avoiding bad data, 13
    `quantile`, percentile, 4
    `solver_irls`, iteratively reweighted optimization, 7

nonlinear optimization, 6

operator
    `lint2`, 2-D linear interpolation, 33
    `refine2`, refine 2-D mesh, 32

pefest module, 13
percentile, 3

quantile module, 4

refine2 operator module, 32
regridding, 32
row normalized PEF, 12

Schwab, 14
solver_irls module, 7
spikes, 10

weighted mean, 4