Chapter 1

Preconditioning

When I first realized that practical imaging methods in widespread industrial use amounted merely to the adjoint of forward modeling, I (and others) thought an easy way to achieve fame and fortune would be to introduce the first steps towards inversion along the lines of Chapter **??**. Although inversion generally requires a prohibitive number of steps, I felt that moving in the gradient direction, the direction of steepest descent, would move us rapidly in the direction of practical improvements. This turned out to be optimistic. It was too slow. But then I learned about the conjugate gradient method that spectacularly overcomes a well-known speed problem with the method of steepest descents. I came to realize that it was still too slow. I learned this by watching the convergence in Figure 1.5. This led me to the helix method in Chapter **??**. Here we'll see how it speeds many applications.

We'll also come to understand why the gradient is such a poor direction both for steepest descent and for conjugate gradients. An indication of our path is found in the contrast between and exact solution $\mathbf{m} = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'\mathbf{d}$ and the gradient $\Delta\mathbf{m} = \mathbf{A}'\mathbf{d}$ (which is the first step starting from $\mathbf{m} = \mathbf{0}$). Notice that $\Delta\mathbf{m}$ differs from $\mathbf{m}$ by the factor $(\mathbf{A}'\mathbf{A})^{-1}$. This factor is sometimes called a spectrum and in some situations it literally is a frequency spectrum. In these cases, $\Delta\mathbf{m}$ simply gets a different spectrum from $\mathbf{m}$ and many iterations are required to fix it. Here we'll find that for many problems, "preconditioning" with the helix is a better way.

1.1 PRECONDITIONED DATA FITTING

Iterative methods (like conjugate-directions) can sometimes be accelerated by a change of variables. The simplest change of variable is called a "trial solution". Formally, we write the solution as

$$\mathbf{m} \quad = \quad \mathbf{S}\mathbf{p} \tag{1.1}$$

where $\mathbf{m}$ is the map we seek, columns of the matrix $\mathbf{S}$ are "shapes" that we like, and coefficients in $\mathbf{p}$ are unknown coefficients to select amounts of the favored shapes. The variables $\mathbf{p}$ are often called the "preconditioned variables". It is not necessary that $\mathbf{S}$ be an invertible matrix, but we'll see later that invertibility is helpful. Take this trial solution and insert it into a typical fitting goal

$$\mathbf{0} \quad \approx \quad \mathbf{Fm} \ - \ \mathbf{d} \tag{1.2}$$

and get

$$\mathbf{0} \quad \approx \quad \mathbf{FSp} \ - \ \mathbf{d} \tag{1.3}$$

We pass the operator **FS** to our iterative solver. After finding the best fitting **p**, we merely evaluate $\mathbf{m} = \mathbf{Sp}$ to get the solution to the original problem.

We hope this change of variables has saved effort. For each iteration, there is a little more work: Instead of the iterative application of **F** and $\mathbf{F}'$ we have iterative application of **FS** and $\mathbf{S}'\mathbf{F}'$. Our hope is that the number of iterations decreases because we are clever, or because we have been lucky in our choice of **S**. Hopefully, the extra work of the preconditioner operator **S** is not large compared to **F**. If we should be so lucky that $\mathbf{S} = \mathbf{F}^{-1}$, then we get the solution immediately. Obviously we would try any guess with $\mathbf{S} \approx \mathbf{F}^{-1}$. Where I have known such **S** matrices, I have often found that convergence is accelerated, but not by much. Sometimes it is worth using **FS** for a while in the beginning, but later it is cheaper and faster to use only **F**. A practitioner might regard the guess of **S** as prior information, like the guess of the initial model $\mathbf{m}_0$.

For a square matrix **S**, the use of a preconditioner should not change the ultimate solution. Taking **S** to be a wide rectangular matrix, reduces the number of adjustable parameters, changes the solution, gets it quicker, but lower resolution.

1.1.1 Preconditioner with a starting guess

In many applications, for many reasons, we have a starting guess $\mathbf{m}_0$ of the solution. You might worry that you could not find the starting preconditioned variable $\mathbf{p}_0 = \mathbf{S}^{-1}\mathbf{m}_0$ because you did not know the inverse of **S**. The way to avoid this problem is to reformulate the problem in terms of a new variable $\tilde{\mathbf{m}}$ where $\mathbf{m} = \tilde{\mathbf{m}} + \mathbf{m}_0$. Then $\mathbf{0} \approx \mathbf{Fm} - \mathbf{d}$ becomes $\mathbf{0} \approx \mathbf{F}\tilde{\mathbf{m}} - (\mathbf{d} - \mathbf{Fm}_0)$ or $\mathbf{0} \approx \mathbf{F}\tilde{\mathbf{m}} - \tilde{\mathbf{d}}$. Thus we have accomplished the goal of taking a problem with a nonzero starting model and converting it a problem of the same type with a zero starting model. Thus we do not need the inverse of **S** because the iteration starts from $\tilde{\mathbf{m}} = \mathbf{0}$ so $\mathbf{p}_0 = \mathbf{0}$.

1.2 PRECONDITIONING THE REGULARIZATION

The basic formulation of a geophysical estimation problem consists of setting up *two* goals, one for data fitting, and the other for model shaping. With two goals, preconditioning is somewhat different. The two goals may be written as:

$$\mathbf{0} \quad \approx \quad \mathbf{Fm} - \mathbf{d} \tag{1.4}$$
$$\mathbf{0} \quad \approx \quad \mathbf{Am} \tag{1.5}$$

which defines two residuals, a so-called “data residual” and a “model residual” that are usually minimized by conjugate-gradient, least-squares methods.

To fix ideas, let us examine a toy example. The data and the first three rows of the matrix below are random numbers truncated to integers. The model roughening operator **A** is a first differencing operator times 100.

```
 d(m)     F(m,n)                                                  iter     Norm
 ---     ------------------------------------------------        ---- -----------
  41.    -55. -90. -24. -13. -73.  61. -27. -19.  23. -55.          1 20.00396538
  33.      8. -86.  72.  87. -41.  -3. -29.  29. -66.  50.          2 12.14780140
 -58.     84. -49.  80.  44. -52. -51.   8.  86.  77.  50.          3  8.94393635
   0.    100.   0.   0.   0.   0.   0.   0.   0.   0.   0.          4  6.04517126
   0.   -100. 100.   0.   0.   0.   0.   0.   0.   0.   0.          5  2.64737511
   0.      0.-100. 100.   0.   0.   0.   0.   0.   0.   0.          6  0.79238468
   0.      0.   0.-100. 100.   0.   0.   0.   0.   0.   0.          7  0.46083349
   0.      0.   0.   0.-100. 100.   0.   0.   0.   0.   0.          8  0.08301232
   0.      0.   0.   0.   0.-100. 100.   0.   0.   0.   0.          9  0.00542009
   0.      0.   0.   0.   0.   0.-100. 100.   0.   0.   0.         10  0.00000565
   0.      0.   0.   0.   0.   0.   0.-100. 100.   0.   0.         11  0.00000026
   0.      0.   0.   0.   0.   0.   0.   0.-100. 100.   0.         12  0.00000012
   0.      0.   0.   0.   0.   0.   0.   0.   0.-100. 100.         13  0.00000000
```

Notice at the tenth iteration, the residual suddenly plunges 4 significant digits. Since there are ten unknowns and the matrix is obviously full-rank, conjugate-gradient theory tells us to expect the exact solution at the tenth iteration. This is the first miracle of conjugate gradients. (The residual actually does not drop to zero. What is printed in the `Norm` column is the square root of the sum of the squares of the residual components at the `iter`-th iteration minus that at the last interation.)

1.2.1 The second miracle of conjugate gradients

The second miracle of conjugate gradients is exhibited below. The data and data fitting matrix are the same, but the model damping is simplified.

```
 d(m)     F(m,n)                                                  iter     Norm
 ---     ------------------------------------------------        ----  ----------
  41.    -55. -90. -24. -13. -73.  61. -27. -19.  23. -55.          1  3.64410686
  33.      8. -86.  72.  87. -41.  -3. -29.  29. -66.  50.          2  0.31269890
 -58.     84. -49.  80.  44. -52. -51.   8.  86.  77.  50.          3 -0.00000021
   0.    100.   0.   0.   0.   0.   0.   0.   0.   0.   0.          4 -0.00000066
   0.      0. 100.   0.   0.   0.   0.   0.   0.   0.   0.          5 -0.00000080
   0.      0.   0. 100.   0.   0.   0.   0.   0.   0.   0.          6 -0.00000065
   0.      0.   0.   0. 100.   0.   0.   0.   0.   0.   0.          7 -0.00000088
   0.      0.   0.   0.   0. 100.   0.   0.   0.   0.   0.          8 -0.00000074
   0.      0.   0.   0.   0.   0. 100.   0.   0.   0.   0.          9 -0.00000035
   0.      0.   0.   0.   0.   0.   0. 100.   0.   0.   0.         10 -0.00000037
   0.      0.   0.   0.   0.   0.   0.   0. 100.   0.   0.         11 -0.00000018
   0.      0.   0.   0.   0.   0.   0.   0.   0. 100.   0.         12  0.00000000
   0.      0.   0.   0.   0.   0.   0.   0.   0.   0. 100.         13  0.00000000
```

Even though the matrix is full-rank, we see the residual drop about 6 decimal places after the third iteration! This convergence behavior is well known in the computational mathematics literature. Despite its practical importance, it doesn't seem to have a name or identified discoverer. So I call it the "second miracle."

Practitioners usually don't like the identity operator for model-shaping. Generally they prefer to penalize wiggliness. For practitioners, the lesson of the second miracle of conjugate gradients is that we have a choice of many iterations, or learning to transform independent variables so that the regularization operator becomes an identity matrix. Basically, such a transformation reduces the iteration count from something about the size of the model space to something about the size of the data space. Such a transformation is called preconditioning. In practice, data is often accumulated in bins. Then the iteration count is reduced (in principle) to the count of full bins and should be independent of the count of the empty bins. This allows refining the bins, enhancing the resolution.

More generally, the model goal $\mathbf{0} \approx \mathbf{Am}$ introduces a roughening operator like a gradient, Laplacian (and in chapter 6 a Prediction-Error Filter (PEF)). Thus the model goal is usually a filter, unlike the data-fitting goal which involves all manner of geometry and physics. When the model goal is a filter its inverse is also a filter. Of course this includes multidimensional filters with a helix.

The preconditioning transformation $\mathbf{m} = \mathbf{Sp}$ gives us

$$\begin{array}{lll} \mathbf{0} & \approx & \mathbf{FSp} - \mathbf{d} \\ \mathbf{0} & \approx & \mathbf{ASp} \end{array} \tag{1.6}$$

The operator $\mathbf{A}$ is a roughener while $\mathbf{S}$ is a smoother. The choices of both $\mathbf{A}$ and $\mathbf{S}$ are somewhat subjective. This suggests that we eliminate $\mathbf{A}$ altogether by *defining* it to be proportional to the inverse of $\mathbf{S}$, thus $\mathbf{AS} = \mathbf{I}$. The fitting goals become

$$\begin{array}{lll} \mathbf{0} & \approx & \mathbf{FSp} - \mathbf{d} \\ \mathbf{0} & \approx & \epsilon\, \mathbf{p} \end{array} \tag{1.7}$$

which enables us to benefit from the "second miracle". After finding $\mathbf{p}$, we obtain the final model with $\mathbf{m} = \mathbf{Sp}$.

Deconvolution on a helix is an all-purpose preconditioning strategy for multidimensional model regularization.

1.2.2 Importance of scaling

Another simple toy example shows us the importance of scaling. We use the same example as above except that the i-th column is multiplied by $i/10$ which means the i-th model variable has been divided by $i/10$.

```
d(m)     F(m,n)                                              iter     Norm
```

```
 ---     ----------------------------------------------     ---- -----------
  41.     -6. -18.  -7.  -5. -36.  37. -19. -15.  21. -55.        1 11.59544849
  33.      1. -17.  22.  35. -20.  -2. -20.  23. -59.  50.        2  6.97337770
 -58.      8. -10.  24.  18. -26. -31.   6.  69.  69.  50.        3  5.64414406
   0.     10.   0.   0.   0.   0.   0.   0.   0.   0.   0.        4  4.32118177
   0.      0.  20.   0.   0.   0.   0.   0.   0.   0.   0.        5  2.64755201
   0.      0.   0.  30.   0.   0.   0.   0.   0.   0.   0.        6  2.01631355
   0.      0.   0.   0.  40.   0.   0.   0.   0.   0.   0.        7  1.23219979
   0.      0.   0.   0.   0.  50.   0.   0.   0.   0.   0.        8  0.36649203
   0.      0.   0.   0.   0.   0.  60.   0.   0.   0.   0.        9  0.28528941
   0.      0.   0.   0.   0.   0.   0.  70.   0.   0.   0.       10  0.06712411
   0.      0.   0.   0.   0.   0.   0.   0.  80.   0.   0.       11  0.00374284
   0.      0.   0.   0.   0.   0.   0.   0.   0.  90.   0.       12 -0.00000040
   0.      0.   0.   0.   0.   0.   0.   0.   0.   0. 100.       13  0.00000000
```

We observe that solving the same problem for the scaled variables has required a severe increase in the number of iterations required to get the solution. We lost the benefit of the second CG miracle. Even the rapid convergence predicted for the 10-th iteration is delayed until the 12-th.

1.2.3 Statistical interpretation

This book is not a statistics book. Never-the-less, many of you have some statistical knowledge that allows you a statistical interpretation of these views of preconditioning.

A statistical concept is that we can combine many streams of random numbers into a composite model. Each stream of random numbers is generally taken to be uncorrelated with the others, to have zero mean, and to have the same variance as all the others. This is often abbreviated as IID, denoting Independent, Identically Distributed. Linear combinations like filtering and weighting operations of these IID random streams can build correlated random functions much like those observed in geophysics. A geophysical practitioner seeks to do the inverse, to operate on the correlated unequal random variables and create the statistical ideal random streams. The identity matrix required for the "second miracle", and our search for a good preconditioning transformation are related ideas. The relationship will become more clear in chapter 6 when we learn how to estimate the best roughening operator $\mathbf{A}$ as a prediction-error filter.

Two philosophies to find a preconditioner:

1. Dream up a smoothing operator $\mathbf{S}$.
2. Estimate a prediction-error filter $\mathbf{A}$, and then use its inverse $\mathbf{S} = \mathbf{A}^{-1}$.

The outstanding acceleration of convergence by preconditioning suggests that the philosophy of image creation by optimization has a dual orthonormality: First, Gauss (and common sense) tells us that the data residuals should be roughly equal in size. Likewise in Fourier space they should be roughly equal in size, which means they should be roughly white, i.e.

orthonormal. (I use the word “orthonormal” because white means the autocorrelation is an impulse, which means the signal is statistically orthogonal to shifted versions of itself.) Second, to speed convergence of iterative methods, we need a whiteness, another orthonormality, in the solution. The map image, the physical function that we seek, might not be itself white, so we should solve first for another variable, the whitened map image, and as a final step, transform it to the “natural colored” map.

1.2.4 The preconditioned solver

Summing up the ideas above, we start from fitting goals

$$\begin{array}{lcl} \mathbf{0} & \approx & \mathbf{Fm}-\mathbf{d} \\ \mathbf{0} & \approx & \mathbf{Am} \end{array} \tag{1.8}$$

and we change variables from $\mathbf{m}$ to $\mathbf{p}$ using $\mathbf{m}=\mathbf{A}^{-1}\mathbf{p}$

$$\begin{array}{lclccc} \mathbf{0} & \approx & \mathbf{Fm}-\mathbf{d} & = & \mathbf{FA}^{-1} & \mathbf{p}-\mathbf{d} \\ \mathbf{0} & \approx & \mathbf{Am} & = & \mathbf{I} & \mathbf{p} \end{array} \tag{1.9}$$

Preconditioning means iteratively fitting by adjusting the $\mathbf{p}$ variables and then finding the model by using $\mathbf{m}=\mathbf{A}^{-1}\mathbf{p}$. A new reusable preconditioned solver is the module `prec_solver` on this page. The variable `x` in `prec_solver` refers to $\mathbf{m}$. Likewise the modeling operator $\mathbf{F}$ is called `oper` and the smoothing operator $\mathbf{A}^{-1}$ is called `prec`. Details of the code are only slightly different from the regularized solver `reg_solver` on page ??.

```
module prec_solver {
  logical, parameter, private  :: T = .true., F = .false.
contains
  subroutine solver_prec( oper, solv, prec, nprec, x, dat, niter, eps, p0) {
    optional                                                        :: p0
   interface {
       integer function oper( adj, add, x, dat) {
         logical, intent (in) :: adj, add
         real, dimension (:)  :: x, dat
       }
       integer function solv( forget, x, g, rr, gg) {
         logical             :: forget
         real, dimension (:) :: x, g, rr, gg
       }
       integer function prec( adj, add, x, dat) {
         logical, intent (in) :: adj, add
         real, dimension (:)  :: x, dat
       }
    }
    real, dimension (:), intent (in)  :: dat, p0           # data, initial
    real, dimension (:), intent (out) :: x                 # solution
    real,                intent (in)  :: eps               # scaling
    integer,             intent (in)  :: niter, nprec      # iterations, size
    real, dimension (nprec)           :: g, p              # gradient, precond
    real, dimension (size (dat) + nprec), target :: rr, gg # residual, conj grad
    real, dimension (:), pointer      :: rd, rm, gd, gm
    integer                           :: i, stat1, stat2, stat
    rm => rr (1 : nprec) ; rd => rr (1 + nprec :)          # model and data resids
    gm => gg (1 : nprec) ; gd => gg (1 + nprec :)          # model and data grads
                           rd = -dat                       # initialize r_d
    if( present( p0)) {
       stat1 = prec( F, F, p0, x)       # x = S p
       stat2 = oper( F, T, x, rd)       # r_d = L x - dat
                 p = p0 ; rm = p0*eps}  # start with p0
    else {       p = 0. ; rm = 0.}      # start with zero
    do i = 1, niter {
       stat2 = oper( T, F, x, rd)
       stat1 = prec( T, F, g, x)        # g = S' L' r_d
       g = g + eps*rm                   # g = S' L' r_d + eps I r_m
       stat1 = prec( F, F, g, x)
       stat2 = oper( F, F, x, gd)       # g_d = L S g
       gm = eps*g                       # g_m = eps I g
       stat = solv (F, p, g, rr, gg)    # step in p and rr
    }
    stat1 = prec( F, F, p, x)           # x = S p
  }
}
```

1.3 OPPORTUNITIES FOR SMART DIRECTIONS

Recall the fitting goals (1.10)

$$\begin{array}{lllllll} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{F}\mathbf{m}-\mathbf{d} & = & \mathbf{F}\mathbf{A}^{-1} \quad \mathbf{p}-\mathbf{d} \\ \mathbf{0} & \approx & \mathbf{r}_m & = & \mathbf{A}\mathbf{m} & = & \quad \mathbf{I} \quad\quad \mathbf{p} \end{array} \tag{1.10}$$

Without preconditioning we have the search direction

$$\Delta\mathbf{m}_{\text{bad}} \quad = \quad \left[\begin{array}{cc} \mathbf{F}' & \mathbf{A}' \end{array} \right] \left[\begin{array}{c} \mathbf{r}_d \\ \mathbf{r}_m \end{array} \right] \tag{1.11}$$

and with preconditioning we have the search direction

$$\Delta\mathbf{p}_{\text{good}} \quad = \quad \left[\begin{array}{cc} (\mathbf{F}\mathbf{A}^{-1})' & \mathbf{I} \end{array} \right] \left[\begin{array}{c} \mathbf{r}_d \\ \mathbf{r}_m \end{array} \right] \tag{1.12}$$

The essential feature of preconditioning is not that we perform the iterative optimization in terms of the variable $\mathbf{p}$. The essential feature is that we use a search direction that is a gradient with respect to $\mathbf{p}'$ not $\mathbf{m}'$. Using $\mathbf{A}\mathbf{m} = \mathbf{p}$ we have $\mathbf{A}\Delta\mathbf{m} = \Delta\mathbf{p}$. This enables us to define a good search direction in model space.

$$\Delta\mathbf{m}_{\text{good}} \quad = \quad \mathbf{A}^{-1}\Delta\mathbf{p}_{\text{good}} \quad = \quad \mathbf{A}^{-1}(\mathbf{A}^{-1})'\mathbf{F}'\mathbf{r}_d + \mathbf{A}^{-1}\mathbf{r}_m \tag{1.13}$$

Define the gradient by $\mathbf{g} = \mathbf{F}'\mathbf{r}_d$ and notice that $\mathbf{r}_m = \mathbf{p}$.

$$\Delta\mathbf{m}_{\text{good}} \quad = \quad \mathbf{A}^{-1}(\mathbf{A}^{-1})'\,\mathbf{g} + \mathbf{m} \tag{1.14}$$

The search direction (1.14) shows a positive-definite operator scaling the gradient. Each component of any gradient vector is independent of each other. All independently point a direction for descent. Obviously, each can be scaled by any positive number. Now we have found that we can also scale a gradient vector by a positive definite matrix and we can still expect the conjugate-gradient algorithm to descend, as always, to the "exact" answer in a finite number of steps. This is because modifying the search direction with $\mathbf{A}^{-1}(\mathbf{A}^{-1})'$ is equivalent to solving a conjugate-gradient problem in $\mathbf{p}$.

1.4 NULL SPACE AND INTERVAL VELOCITY

A bread-and-butter problem in seismology is building the velocity as a function of depth (or vertical travel time) starting from certain measurements. The measurements are described elsewhere (BEI for example). They amount to measuring the integral of the velocity squared from the surface down to the reflector. It is known as the RMS (root-mean-square) velocity. Although good quality echos may arrive often, they rarely arrive continuously for all depths. Good information is interspersed unpredictably with poor information. Luckily we can also estimate the data quality by the "coherency" or the "stack energy". In summary, what we get from observations and preprocessing are two functions of travel-time depth, (1) the integrated (from the surface) squared velocity, and (2) a measure of the quality of the integrated velocity measurement. Some definitions:

d is a data vector whose components range over the vertical traveltime depth τ, and whose component values contain the scaled RMS velocity squared $\tau v_{\mathrm{RMS}}^2/\Delta\tau$ where $\tau/\Delta\tau$ is the index on the time axis.

W is a diagonal matrix along which we lay the given measure of data quality. We will use it as a weighting function.

C is the matrix of causal integration, a lower triangular matrix of ones.

D is the matrix of causal differentiation, namely, $\mathbf{D} = \mathbf{C}^{-1}$.

u is a vector whose components range over the vertical traveltime depth τ, and whose component values contain the interval velocity squared v_{interval}^2.

From these definitions, under the assumption of a stratified earth with horizontal reflectors (and no multiple reflections) the theoretical (squared) interval velocities enable us to define the theoretical (squared) RMS velocities by

$$\mathbf{Cu} \quad = \quad \mathbf{d} \tag{1.15}$$

With imperfect data, our data fitting goal is to minimize the residual

$$\mathbf{0} \quad \approx \quad \mathbf{W}[\mathbf{Cu} - \mathbf{d}] \tag{1.16}$$

To find the interval velocity where there is no data (where the stack power theoretically vanishes) we have the "model damping" goal to minimize the wiggliness **p** of the squared interval velocity **u**.

$$\mathbf{0} \quad \approx \quad \mathbf{D}\mathbf{u} \quad = \quad \mathbf{p} \tag{1.17}$$

We precondition these two goals by changing the optimization variable from interval velocity squared **u** to its wiggliness **p**. Substituting $\mathbf{u} = \mathbf{C}\mathbf{p}$ gives the two goals expressed as a function of wiggliness **p**.

$$\mathbf{0} \quad \approx \quad \mathbf{W}\left[\mathbf{C}^2\mathbf{p} - \mathbf{d}\right] \tag{1.18}$$

$$\mathbf{0} \quad \approx \quad \epsilon\, \mathbf{p} \tag{1.19}$$

1.4.1 Balancing good data with bad

Choosing the size of ϵ chooses the stiffness of the curve that connects regions of good data. Our first test cases gave solutions that we interpreted to be too stiff at early times and too flexible at later times. This leads to two possible ways to deal with the problem. One way modifies the model shaping and the other modifies the data fitting. The program below weakens the data fitting weight with time. This has the same effect as stiffening the model shaping with time.

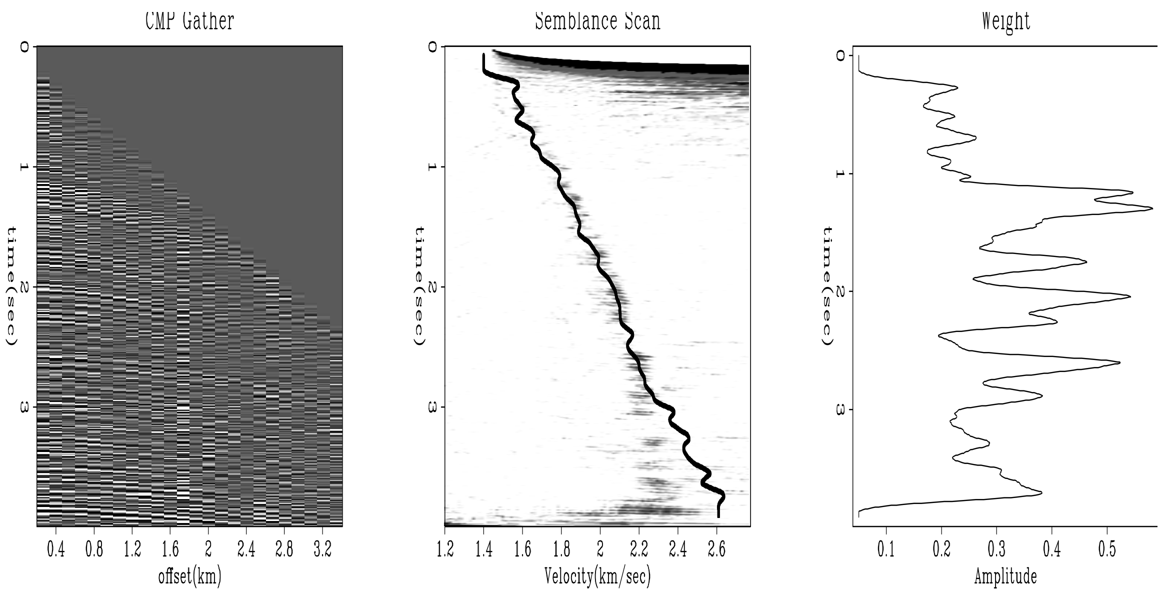

Figure 1.1: Raw CMP gather (left), Semblance scan (middle), and semblance value used for weighting function (right). (Clapp) $\boxed{\text{prc-clapp}}$ [ER]

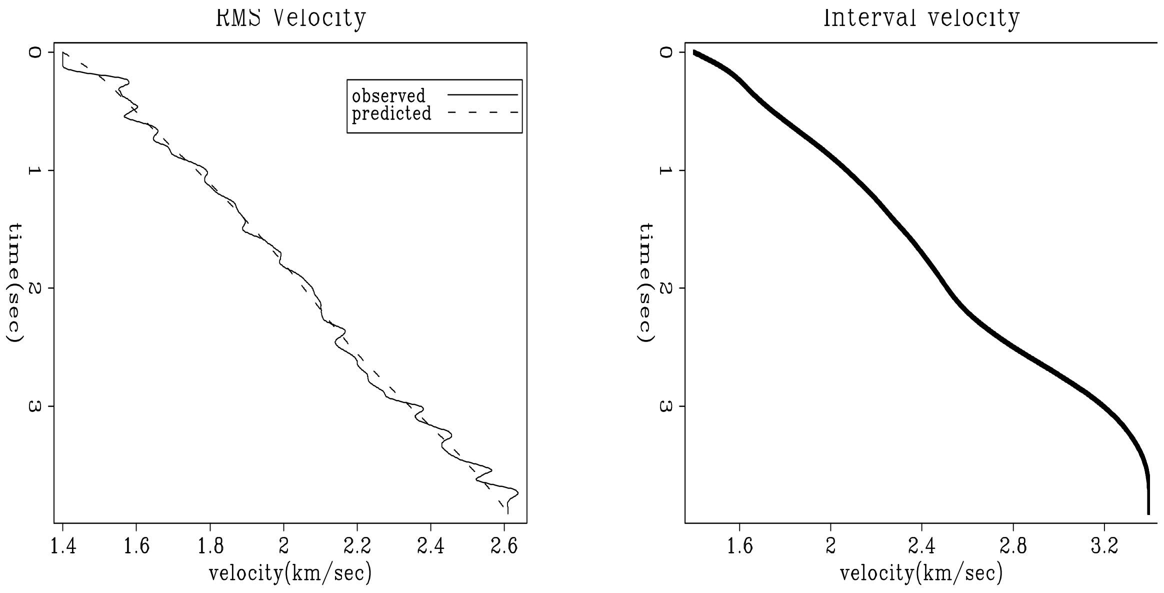

Figure 1.2: Observed RMS velocity and that predicted by a stiff model with $\epsilon = 4$. (Clapp) $\boxed{\text{prc-stiff}}$ [ER]

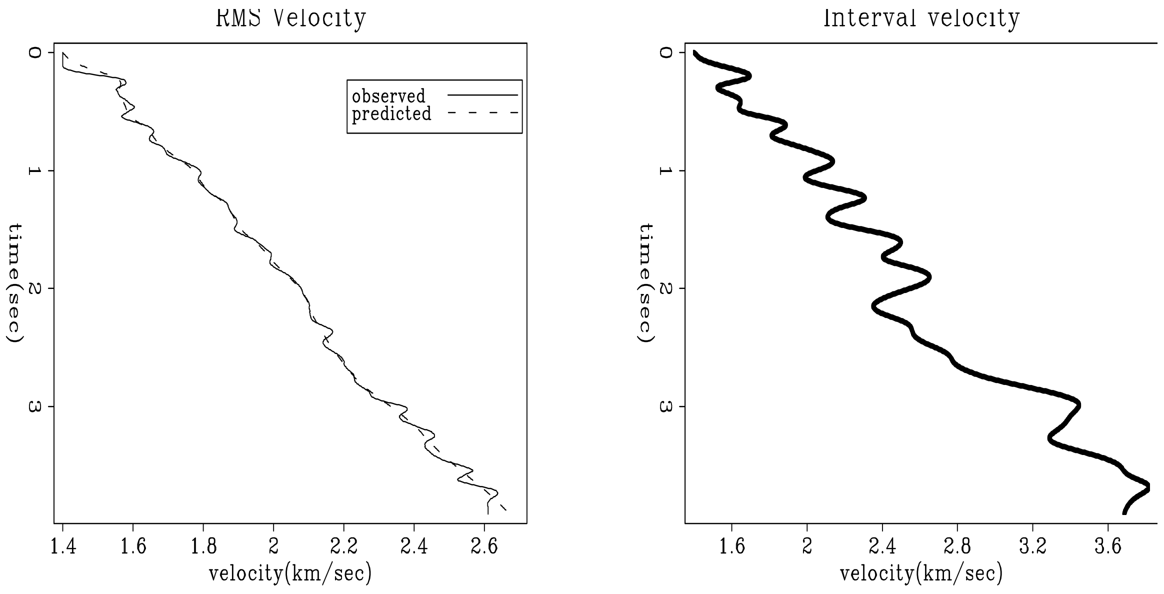

Figure 1.3: Observed RMS velocity and that predicted by a flexible model with $\epsilon = .25$ (Clapp) $\boxed{\text{prc-flex}}$ [ER]

1.4.2 Bigsolver

The regression (1.18) includes a weighting function, so we need yet another solver module very much like the regularized and preconditioned solvers that we used earlier. The idea is essentially the same. Instead of preparing such a solver here, I refer you to the end of the book for `solver_mod` on page ??, a big solver that incorporates everything that we need in the book. Hopefully we will not need to look at solvers again for a while. Module `vrms2int` on the current page was written by Bob Clapp to get the results in Figures 1.1 to 1.3. Notice that he is using `solver_mod` on page ??.

```
module vrms2int_mod {                # Transform from RMS to interval velocity
  use causint
  use cgstep_mod
  use solver_mod
contains
  subroutine vrms2int( niter, eps, weight, vrms, vint) {
    integer,            intent( in)     :: niter     # iterations
    real,               intent( in)     :: eps       # scaling parameter
    real, dimension(:), intent( in out) :: vrms      # RMS velocity
    real, dimension(:), intent( out)    :: vint      # interval velocity
    real, dimension(:), intent( in)     :: weight    # data weighting
    integer                             :: st,it,nt
    logical, dimension( size( vint))    :: mask
    real,    dimension( size( vrms))    :: dat ,wt
    nt = size( vrms)
    do it= 1, nt {
       dat( it) = vrms( it) * vrms( it) * it
       wt( it) = weight( it)*(1./it)           # decrease weight with time
       }
    mask = .false.;   mask( 1) = .true.            # constrain first point
    vint = 0.     ;   vint( 1) = dat( 1)
    call solver_prec( causint_lop, cgstep, x= vint, dat= dat, niter= niter,
                      known= mask, x0= vint, wt= wt,
                      prec=causint_lop, eps= eps, nprec= nt)
    call cgstep_close()
    st = causint_lop( .false., .false., vint, dat)
    do it= 1, nt
       vrms( it) = sqrt( dat( it)/it)
    vint = sqrt( vint)
    }
}
```

1.4.3 Lateral variations

The analysis above appears one dimensional in depth. Conventional interval velocity estimation builds a velocity-depth model independently at each lateral location. Here we have a logical path for combining measurements from various lateral locations. We can change the regularization to something like $\mathbf{0} \approx \nabla \mathbf{u}$. Instead of merely minimizing the vertical gradient of velocity we minimize its spatial gradient. Luckily we have preconditioning and the helix to speed the solution.

Likewise the construction of the data quality screen **G** would naturally involve the full three-dimensional setting.

1.4.4 Blocky models

Sometimes we seek a velocity model that increases smoothly with depth through our scattered measurements of good-quality RMS velocities. Other times, we seek a blocky model. (Where seismic data is poor, a well log could tell us whether to choose smooth or blocky.) Here we see an estimation method that can choose the blocky alternative, or some combination of smooth and blocky.

Consider a five layer model, each layer having unit traveltime thickness (so integration is simply summation). Let the squared interval velocities be (a,b,c,d,e) with strong reliable reflections at the base of layer c and layer e, and weak, incoherent, "bad" reflections at bases of (a,b,d). Thus we measure V_c^2 the RMS velocity squared of the top three layers and V_e^2 that for all five layers. Since we have no reflection from at the base of the fourth layer, the velocity in the fourth layer is not measured but a matter for choice. In a smooth linear fit we would want $d=(c+e)/2$. In a blocky fit we would want $d=e$.

Our screen for good reflections looks like $(0,0,1,0,1)$ and our screen for bad ones looks like the complement $(1,1,0,1,0)$. We put these screens on the diagonals of diagonal matrices **G** and **B**. Our fitting goals are:

$$3V_c^2 \approx a+b+c \tag{1.20}$$
$$5V_e^2 \approx a+b+c+d+e \tag{1.21}$$
$$u_0 \approx a \tag{1.22}$$
$$0 \approx -a+b \tag{1.23}$$
$$0 \approx -b+c \tag{1.24}$$
$$0 \approx -c+d \tag{1.25}$$
$$0 \approx -d+e \tag{1.26}$$

For the blocky solution, we do not want the fitting goal (1.25). Further explanations await completion of examples.

1.5 INVERSE LINEAR INTERPOLATION

The first example is a simple synthetic test for 1-D inverse interpolation. The input data were randomly subsampled (with decreasing density) from a sinusoid (Figure 1.4). The forward operator **L** in this case is linear interpolation. We seek a regularly sampled model that could predict the data with a forward linear interpolation. Sparse irregular distribution of the input data makes the regularization enforcement a necessity. I applied convolution with the simple $(1,-1)$ difference filter as the operator **D** that forces model continuity (the first-order spline). An appropriate preconditioner **S** in this case is recursive causal integration. As expected, pre-

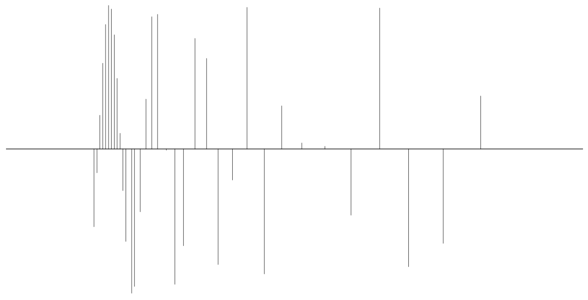
Figure 1.4: The input data are irregularly sampled. prc-data [ER]

conditioning provides a much faster rate of convergence. Since iteration to the exact solution is never achieved in large-scale problems, the results of iterative optimization may turn out quite differently. Bill Harlan points out that the two goals in (1.8) conflict with each other: the first one enforces “details” in the model, while the second one tries to smooth them out. Typically, regularized optimization creates a complicated model at early iterations. At first, the data fitting goal (1.8) plays a more important role. Later, the regularization goal (1.8) comes into play and simplifies (smooths) the model as much as needed. Preconditioning acts differently. The very first iterations create a simplified (smooth) model. Later, the data fitting goal adds more details into the model. If we stop the iterative process early, we end up with an insufficiently complex model, not in an insufficiently simplified one. Figure 1.5 provides a clear illustration of Harlan’s observation.

Figure 1.6 measures the rate of convergence by the model residual, which is a distance from the current model to the final solution. It shows that preconditioning saves many iterations. Since the cost of each iteration for each method is roughly equal, the efficiency of preconditioning is evident.

The module `invint2` on the current page invokes the solvers to make Figures 1.5 and 1.6. We use convolution with `helicon` on page ?? for the regularization and we use deconvolution with `polydiv` on page ?? for the preconditioning. The code looks fairly straightforward except for the oxymoron `known=aa%mis`.

```
module invint2 {   # Inverse linear interpolation
  use lint1
  use helicon                           # regularized    by helix   filtering
  use polydiv                           # preconditioned by inverse filtering
  use cgstep_mod + solver_mod
contains
  subroutine invint1( niter, coord,ord, o1,d1, mm,mmov, eps, aa, doprec) {
    logical,                 intent( in)  :: doprec
    integer,                 intent( in)  :: niter
    real,                    intent( in)  :: o1, d1, eps
    real,    dimension( :),   intent( in)  :: ord
    type( filter),           intent( in)  :: aa
    real,    dimension( :),   intent( out) :: mm
    real,    dimension( :,:), intent( out) :: mmov         # model movie
    real,    dimension( :),   pointer      :: coord        # coordinate
    call lint1_init( o1, d1, coord)
    if( doprec) {                                          # preconditioning
       call polydiv_init( size(mm), aa)
```

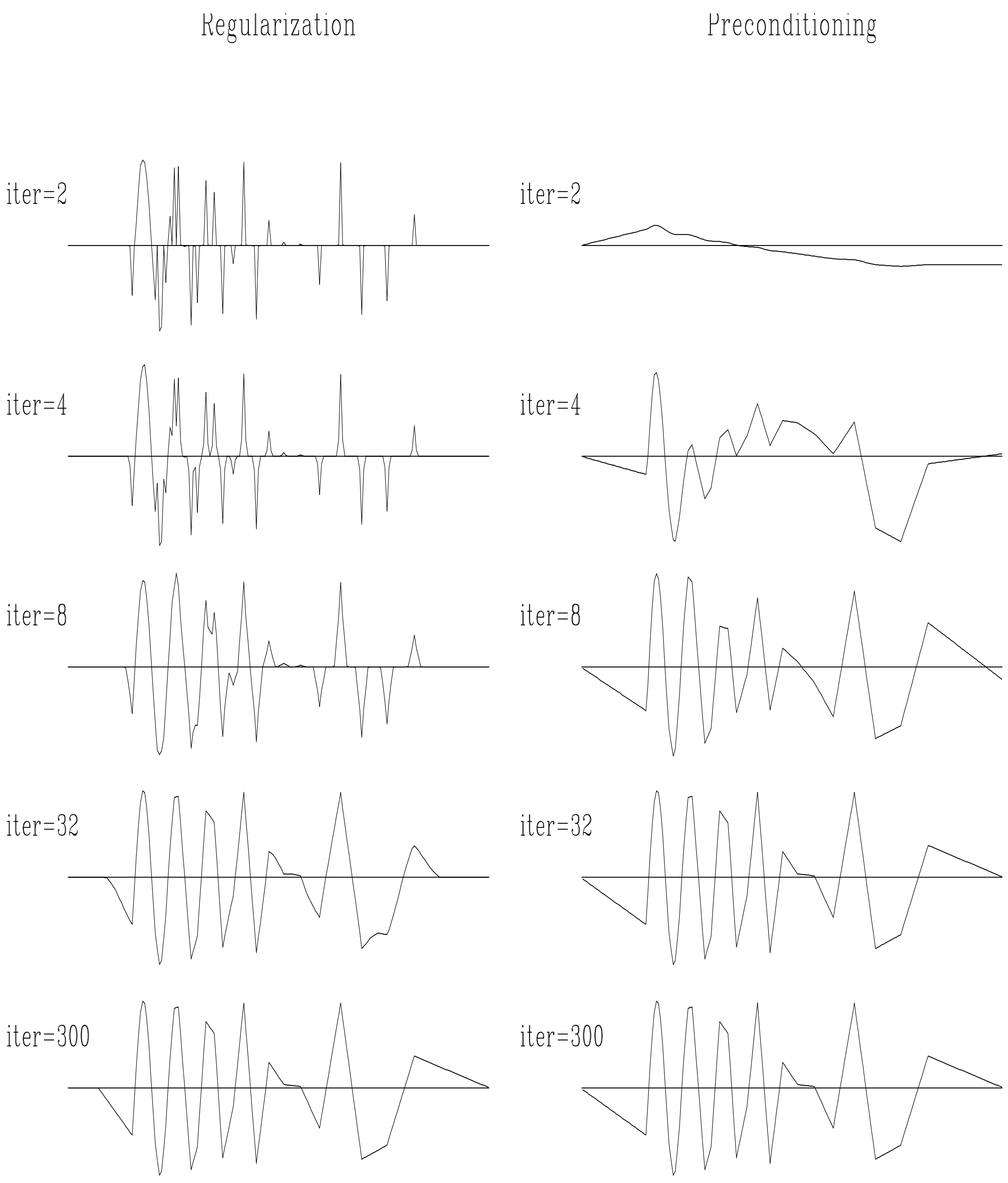

Figure 1.5: Convergence history of inverse linear interpolation. Left: regularization, right: preconditioning. The regularization operator **A** is the derivative operator (convolution with $(1, -1)$. The preconditioning operator **S** is causal integration. prc-conv1 [ER]

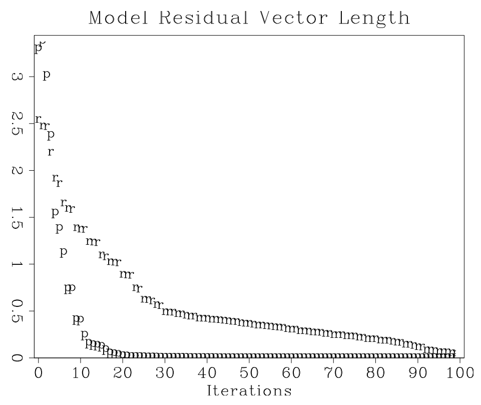

Figure 1.6: Convergence of the iterative optimization, measured in terms of the model residual. The “p” points stand for preconditioning; the “r” points, regularization. $\boxed{\text{prc-schwab1}}$ [ER]

```
        call solver_prec( lint1_lop, cgstep, niter = niter, x = mm, dat = ord,
                     prec = polydiv_lop, nprec = size(mm), eps = eps,
                     xmov = mmov)
        call polydiv_close()
     } else {                                             # regularization
        call helicon_init( aa)
        call solver_reg(  lint1_lop, cgstep, niter = niter, x = mm, dat = ord,
                     reg = helicon_lop, nreg = size(mm), eps = eps,
                     xmov = mmov)
     }
     call cgstep_close()
  }
}
```

1.6 EMPTY BINS AND PRECONDITIONING

There are at least three ways to fill empty bins. They seem to be all equivalent, though that is not as obvious as I would like it to be.

The original way in Chapter **??** is to restore missing data by ensuring that the restored data, after specified filtering, has minimum energy, say $\mathbf{Am} \approx \mathbf{0}$. Introduce the selection mask operator $\mathbf{K}$, a diagonal matrix with ones on the known data and zeros elsewhere (on the missing data). Thus $\mathbf{A(I-K+K)m} \approx \mathbf{0}$ or

$$\mathbf{A(I-K)m} \quad \approx \quad -\mathbf{AKm} \quad = \quad -\mathbf{Am}_k \,, \tag{1.27}$$

where we have defined $\mathbf{m}_k$ to be the data with missing values set to zero by $\mathbf{m}_k = \mathbf{Km}$.

A second way to find missing data is with the set of goals

$$\begin{array}{rcl} \mathbf{Km} & \approx & \mathbf{m}_k \\ \epsilon\mathbf{Am} & \approx & 0 \end{array} \tag{1.28}$$

and take the limit as the scalar $\epsilon \to 0$. At that limit, we should have the same result as equation (1.27).

A third way to find missing data is to precondition equation (1.28), namely, try the substitution $\mathbf{m} = \mathbf{A}^{-1}\mathbf{p}$.

$$\begin{array}{rcl} \mathbf{K}\mathbf{A}^{-1}\mathbf{p} & \approx & \mathbf{m}_k \\ \epsilon\mathbf{p} & \approx & 0 \end{array} \tag{1.29}$$

I think (hope) it is proven later that if we start from $\mathbf{p} = 0$ and if we are interested in the limit $\epsilon \to 0$ we can simply forget about the fitting goal $\epsilon\mathbf{p} \approx 0$.

1.6.1 Inverse masking code

The selection (or masking) operator **K** is implemented in `mask1()` on this page.

```
module mask1 { # masking operator
  logical, dimension( :), pointer :: m
#% _init( m)
#% _lop( x, y)
  if( adj)
           where( m) x += y
  else  #
           where( m) y += x
}
```

The inverting of the mask operator proceeds much as we inverted the linear interpolation operator with module `invint2` on page 13. The main difference is we swap the selection operator for the linear interpolation operator. (Philosophically, selection is like binning which is like nearest-neighbor interpolation.) The module `mis2` on this page, does the job.

```
module mis2 {
  use mask1 + helicon + polydiv + cgstep_mod + solver_mod
contains
# fill in missing data by minimizing power out of a given filter.
# by helix magic works in any number of dimensions
  subroutine mis1( niter, xx, aa, known, doprec) {
    logical,                intent( in)     :: doprec
    integer,                intent( in)     :: niter
    type( filter),          intent( in)     :: aa
    logical, dimension( :), intent( in)     :: known
    real,    dimension( :), intent( in out) :: xx       # fitting variables
    real,    dimension( :), allocatable     :: dd
    logical, dimension( :), pointer         :: kk
    integer                                 :: nx
    nx = size( xx)
    if( doprec) {                          #  preconditioned
       allocate( kk( nx)); kk = known
       call mask1_init( kk)
       call polydiv_init( nx, aa)
       call solver_prec( mask1_lop, cgstep, niter= niter, x= xx, dat= xx,
                    prec= polydiv_lop, nprec= nx, eps= 0.)
       call polydiv_close()
       deallocate( kk)
    } else {                              #  regularized
       allocate( dd( nx)); dd = 0.
       call helicon_init( aa)
       call solver( helicon_lop, cgstep, niter= niter, x= xx, dat= dd,
                    known = known, x0= xx)
       deallocate( dd)
    }
    call cgstep_close( )
  }
}
```

It is instructive to compare `mis2` on the preceding page with `invint2` on page 13. Both are essentially filling empty regions consistant with prior knowledge at particular locations and minimizing energy of the filtered field. Both use the helix and can be used in N-dimensional space.

1.7 SEABEAM: FILLING THE EMPTY BINS WITH A LAPLACIAN

Figure 1.7 shows a day's worth of data[1] collected at sea by SeaBeam, an apparatus for measuring water depth both directly under a ship, and somewhat off to the sides of the ship's track. The data is measurements of depth $h(x, y)$ at miscellaneous locations in the (x, y)-plane. The

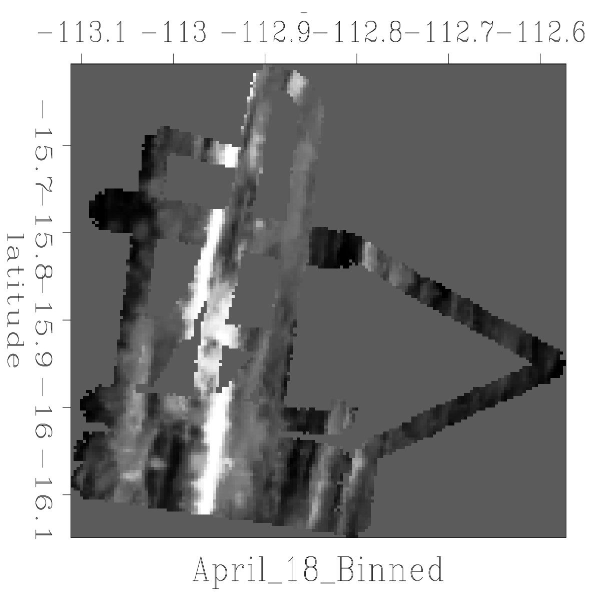

Figure 1.7: Depth of the ocean under ship tracks. prc-seabin90 [ER]

locations are scattered about, according to various aspects of the ship's navigation and the geometry of the SeaBeam sonic antenna. Figure 1.7 was made by binning with `bin2()` on page ?? and equation (**??**). The spatial spectra of the noise in the data could be estimated where tracks cross over themselves. More interesting are the empty mesh locations where no data is recorded. Here I left empty locations with a background value equal to the mean

[1] I'd like to thank Alistair Harding for this interesting data set named April 18.

depth $\bar{h}$. Supposing the roughening operator to be the Laplacian operator ∇^2 and using module `mis2` on page 16 led to the result in Figure 1.8. After many iterations, both regularization and preconditioning lead us to the same result. After a small number of iterations, we see that regularization has filled the small holes but it has not reached out far away from the known data. With preconditioning, it is the opposite.

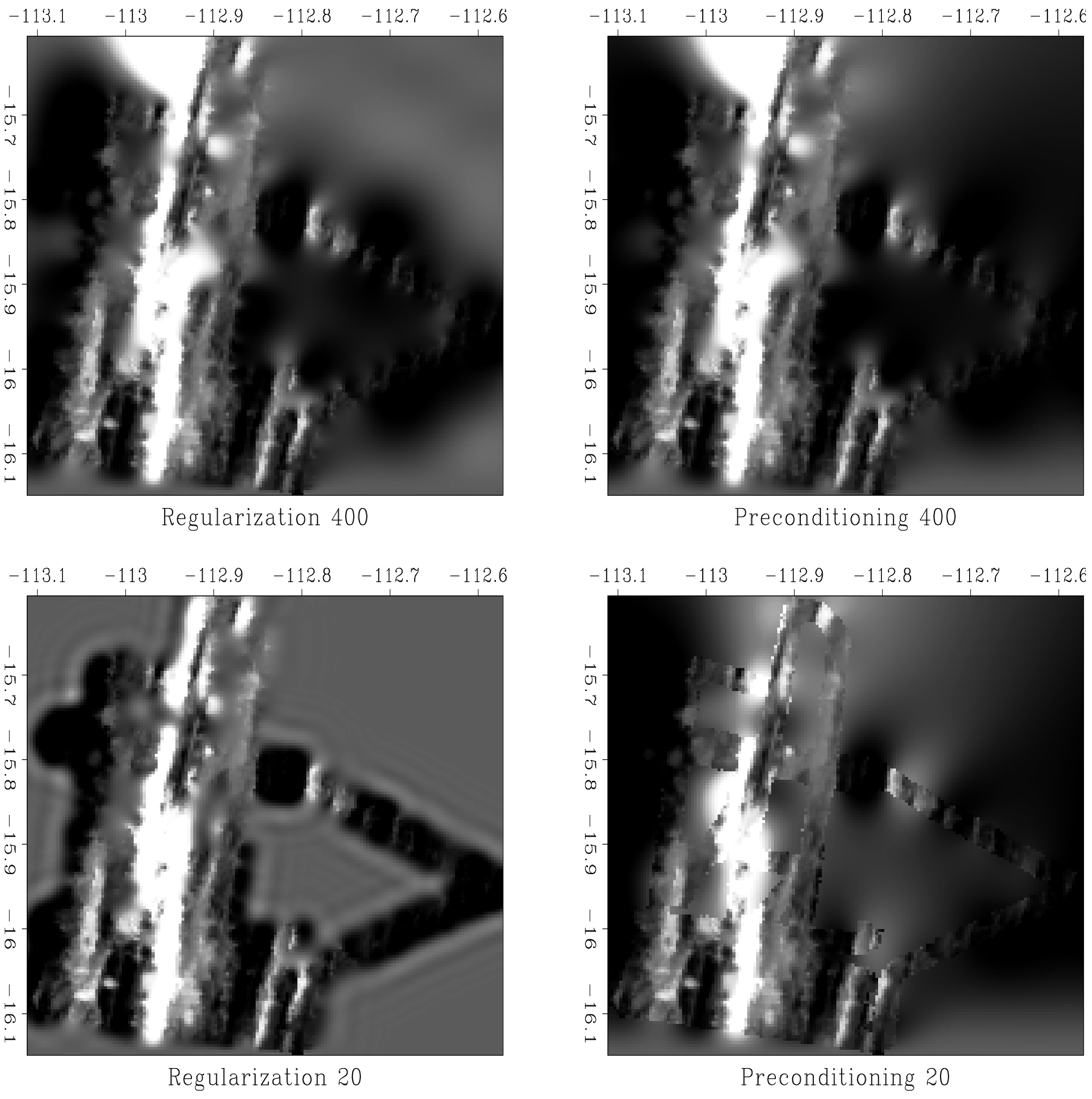

Figure 1.8: Views of the ocean bottom after filling. (We'll return to this data in the next chapter to do a better job.) (Fomel) prc-prcfill [ER,M]

1.8 UNDERDETERMINED LEAST-SQUARES

Construct theoretical data with

$$\mathbf{d} \quad = \quad \mathbf{F}\mathbf{m} \tag{1.30}$$

Assume there are fewer data points than model points and that the matrix $\mathbf{F}\mathbf{F}'$ is invertible. From the theoretical data we estimate a model $\mathbf{m}_0$ with

$$\mathbf{m}_0 \quad = \quad \mathbf{F}'(\mathbf{F}\mathbf{F}')^{-1}\mathbf{d} \tag{1.31}$$

To verify the validity of the estimate, insert the estimate (1.31) into the data modeling equation (1.30) and notice that the estimate $\mathbf{m}_0$ predicts the correct data.

Now we will show that of all possible models $\mathbf{m}$ that predict the correct data, $\mathbf{m}_0$ has the least energy. (I'd like to thank Sergey Fomel for this clear and simple proof that does *not* use Lagrange multipliers.) First split (1.31) into an intermediate result $\mathbf{d}_0$ and final result:

$$\mathbf{d}_0 \quad = \quad (\mathbf{F}\mathbf{F}')^{-1}\mathbf{d} \tag{1.32}$$
$$\mathbf{m}_0 \quad = \quad \mathbf{F}'\mathbf{d}_0 \tag{1.33}$$

Consider another model ($\mathbf{x}$ not equal to zero)

$$\mathbf{m} \quad = \quad \mathbf{m}_0 + \mathbf{x} \tag{1.34}$$

which fits the theoretical data. Since $\mathbf{d} = \mathbf{F}\mathbf{m}_0$, we see that $\mathbf{x}$ is a null space vector.

$$\mathbf{F}\mathbf{x} \quad = \quad \mathbf{0} \tag{1.35}$$

First we see that $\mathbf{m}_0$ is orthogonal to $\mathbf{x}$ because

$$\mathbf{m}_0'\mathbf{x} \quad = \quad (\mathbf{F}'\mathbf{d}_0)'\mathbf{x} \quad = \quad \mathbf{d}_0'\mathbf{F}\mathbf{x} \quad = \quad \mathbf{d}_0'\mathbf{0} \quad = \quad 0 \tag{1.36}$$

Therefore,

$$\mathbf{m}'\mathbf{m} \quad = \quad \mathbf{m}_0'\mathbf{m}_0 + \mathbf{x}'\mathbf{x} + 2\mathbf{x}'\mathbf{m}_0 \quad = \quad \mathbf{m}_0'\mathbf{m}_0 + \mathbf{x}'\mathbf{x} \quad \geq \quad \mathbf{m}_0'\mathbf{m}_0 \tag{1.37}$$

so adding null space to $\mathbf{m}_0$ can only increase its energy. In summary, the solution $\mathbf{m}_0 = \mathbf{F}'(\mathbf{F}\mathbf{F}')^{-1}\mathbf{d}$ has less energy than any other model that satisfies the data.

Not only does the theoretical solution $\mathbf{m}_0 = \mathbf{F}'(\mathbf{F}\mathbf{F}')^{-1}\mathbf{d}$ have minimum energy, but the result of iterative descent will too, provided that we begin iterations from $\mathbf{m}_0 = 0$ or any $\mathbf{m}_0$ with no null-space component. In (1.36) we see that the orthogonality $\mathbf{m}_0'\mathbf{x} = 0$ does not arise because $\mathbf{d}_0$ has any particular value. It arises because $\mathbf{m}_0$ is of the form $\mathbf{F}'\mathbf{d}_0$. Gradient methods contribute $\Delta\mathbf{m} = \mathbf{F}'\mathbf{r}$ which is of the required form.

1.9 SCALING THE ADJOINT

Given the usual linearized fitting goal 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 that 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

$$\mathbf{W}^2 \quad = \quad \frac{\mathbf{diag\ F'd}}{\mathbf{diag\ F'FF'd}} \tag{1.38}$$

which obviously has the correct physical units. (The mathematical function **diag** takes a vector and lies it along the diagonal of a square matrix.) The weight $\mathbf{W}^2$ can be thought of as a diagonal matrix containing the ratio of two images. A problem with the choice (1.38) is that the denominator might vanish or might even be negative. The way to stabilize any ratio is suggested at the beginning of Chapter **??**; that is, we revise the ratio a/b to

$$\mathbf{W}^2 \quad = \quad \frac{\mathbf{diag} < ab >}{\mathbf{diag} < b^2 + \epsilon^2 >} \tag{1.39}$$

where ϵ is a parameter to be chosen, and the angle braces indicate the possible need for local smoothing.

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 $\mathbf{p}$ by the substitution $\mathbf{m} = \mathbf{W}\mathbf{p}$. Then $\mathbf{d} \approx \mathbf{F}\mathbf{m} = \mathbf{F}\mathbf{W}\mathbf{p}$. To find $\mathbf{p}$ instead of $\mathbf{m}$, we do CD 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 $\mathbf{d} \approx \mathbf{F}\mathbf{W}\mathbf{p}$ to form the image $\hat{\mathbf{p}} = (\mathbf{F}\mathbf{W})'\mathbf{d}$. At the end of the iterations, we convert from $\mathbf{p}$ back to $\mathbf{m}$ with $\mathbf{m} = \mathbf{W}\mathbf{p}$. The result after the first iteration $\hat{\mathbf{m}} = \mathbf{W}\hat{\mathbf{p}} = \mathbf{W}(\mathbf{F}\mathbf{W})'\mathbf{d} = \mathbf{W}^2\mathbf{F}'\mathbf{d}$ turns out to be the same as Symes scaling.

By (1.38), $\mathbf{W}$ has physical units inverse to $\mathbf{F}$. Thus the transformation $\mathbf{F}\mathbf{W}$ has no units so the $\mathbf{p}$ variables have physical units of data space. Experimentalists might enjoy seeing the solution $\mathbf{p}$ with its data units more than viewing the solution $\mathbf{m}$ with its more theoretical model units.

The theoretical solution for underdetermined systems $\mathbf{m} = \mathbf{F}'(\mathbf{F}\mathbf{F}')^{-1}\mathbf{d}$ suggests an alternate approach using instead $\hat{\mathbf{m}} = \mathbf{F}'\mathbf{W}_d^2\mathbf{d}$. A possibility for $\mathbf{W}_d^2$ is

$$\mathbf{W}_d^2 \quad = \quad \frac{\mathbf{diag\ d}}{\mathbf{diag\ FF'd}} \tag{1.40}$$

Experience tells me that a broader methodology is needed. Appropriate scaling is required in both data space and model space. We need something that includes a weight for each space, $\mathbf{W}_m$ and $\mathbf{W}_d$ where $\hat{\mathbf{m}} = \mathbf{W}_m\mathbf{F}'\mathbf{W}_d\mathbf{d}$.

I have a useful practical example (stacking in $v(z)$ media) in another of my electronic books (BEI), where I found both $\mathbf{W}_m$ and $\mathbf{W}_d$ by iterative guessing. But I don't know how to give you a general strategy. I feel this is a major unsolved(?) opportunity.

1.10 ACKNOWLEDGEMENTS

Nearly everything I know about null spaces I learned from Dave Nichols, Bill Symes, Bill Harlan, and Sergey Fomel.

Index