Chapter 5

Preconditioning

When problems have less than about 10^3 unknowns, "exact" methods are generally satisfactory. When there are more unknowns, we use iterative methods. Iterative methods are about as fast as exact methods, but they allow us to hope that when we give up exhausted that we are near the best that can be done. With more experience, I learned that we often remain far from the solution we seek. Thus we seek generally applicable problem reformulations that will converge more rapidly. Luckily we find that the helix gives a huge speedup in many problems, in those problems that are densely sampled in model space.

5.1 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

$$\mathbf{m} \quad = \quad \mathbf{Sp} \tag{5.1}$$

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

$$\mathbf{0} \quad \approx \quad \mathbf{Fm} \;-\; \mathbf{d} \tag{5.2}$$

and get

$$\mathbf{0} \quad \approx \quad \mathbf{FSp} \;-\; \mathbf{d} \tag{5.3}$$

We pass the operator $\mathbf{FS}$ to our iterative solver. After finding the best fitting $\mathbf{p}$, 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 $\mathbf{F}$ and $\mathbf{F}'$ we have iterative

application of $\mathbf{FS}$ 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 $\mathbf{S}$. Hopefully, the extra work of the preconditioner operator $\mathbf{S}$ is not large compared to $\mathbf{F}$. 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 $\mathbf{S} \approx \mathbf{F}^{-1}$. Even where I have known such $\mathbf{S}$ matrices, I have often found that convergence is accelerated, but not very much. Sometimes it is worth using $\mathbf{FS}$ for a while in the beginning, but later it is cheaper and faster to use only $\mathbf{F}$. A practitioner might regard the guess of $\mathbf{S}$ as prior information, like the guess of the initial model $\mathbf{m}_0$.

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

5.1.1 Preconditioner conflicts with starting guess

In many applications, for many reasons, we have a starting guess $\mathbf{m}_0$ of the solution. It is usually a wise idea to use the starting guess rather than begin anew from $\mathbf{m} = \mathbf{0}$. This presents a problem. To do preconditioning, we begin by defining (implicitly) the new underlying model variables $\mathbf{p}$ by the substitution $\mathbf{m} = \mathbf{Sp}$. When we are given an initial $\mathbf{m}_0$, then the corresponding $\mathbf{p}_0$ should be $\mathbf{p}_0 = \mathbf{S}^{-1}\mathbf{m}_0$ but we might not know $\mathbf{S}^{-1}$. So a noninvertible preconditioner is in conflict with a nonzero starting model. You cannot have both. (If you want to explore a null space by trying various starting solutions, it might suffice to try various starting $\mathbf{p}_0$; you need not try various starting $\mathbf{m}_0$.)

5.2 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:

$$\mathbf{0} \approx \mathbf{Fm} - \mathbf{d} \tag{5.4}$$

$$\mathbf{0} \approx \mathbf{Am} \tag{5.5}$$

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 $\mathbf{A}$ 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.)

5.2.1 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!

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 later chapters, 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.

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

$$\begin{aligned} \mathbf{0} &\approx \mathbf{FSp} - \mathbf{d} \\ \mathbf{0} &\approx \mathbf{ASp} \end{aligned} \tag{5.6}$$

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

$$\begin{aligned} \mathbf{0} &\approx \mathbf{FSp} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\, \mathbf{p} \end{aligned} \tag{5.7}$$

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

Because model shaping is generally a filtering operation, and because preconditioning operators are best when they are invertible, deconvolution on a helix is an all-purpose preconditioning strategy for multidimensional model regularization.

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

5.2.2 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.

5.2.3 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 $\mathbf{S} = \mathbf{A}^{-1}$.

From examining these toy problems, we suspect that on large problems we will speed the convergence if we change variables to convert the regularization matrix into an identity. These small examples are confirmed by many peoples' experience with larger examples. Luckily we have multidimensional filters on the helix so we can readily transform one regularization type to the other.

The outstanding accelleration 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.

5.3 PRECONDITIONING AND INTERVAL VELOCITY

The basic idea of preconditioning as described above is that we start from fitting goals

$$\begin{aligned} \mathbf{0} &\approx \mathbf{F}\mathbf{m} - \mathbf{d} \\ \mathbf{0} &\approx \mathbf{A}\mathbf{m} \end{aligned} \tag{5.8}$$

and change variables with $\mathbf{m} = \mathbf{S}\mathbf{p}$ getting

$$\begin{aligned} \mathbf{0} &\approx \mathbf{F}\mathbf{S}\mathbf{p} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\, \mathbf{p} \end{aligned} \tag{5.9}$$

where we have established $\mathbf{A}\mathbf{S} = \mathbf{I}$ (to take advantage of the second CG miracle), either by finding $\mathbf{S} = \mathbf{A}^{-1}$ or by revising the problem formulation $\mathbf{A}$ to be $\mathbf{S}^{-1}$.

It would seem that the next step is to invoke a regularized solver such as module `reg_solver` on page 82 with the new operator $\mathbf{F}\mathbf{S}$ and damping operator $\mathbf{I}$. Alternately, a new reusable preconditioned solver is the module `prec_solver` on page 6.

```
module prec_solver {
  logical, parameter, private  :: T = .true., F = .false.
contains
  subroutine solver_prec (oper, solv, prec, nprec, x, dat, niter, eps, x0) {
    optional                                                        :: x0
   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, x0          # 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 (x0)) {
     stat1 = prec( F, F, x0, x)       # x = S p
     stat2 = oper( F, T, x, rd)       # r_d = L x - dat
                 p = x0 ; rm = x0*eps} # start with x0
  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, T, 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
 }
}
```

5.4 NULL SPACE AND INTERVAL VELOCITY

A basic 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 travel time depth τ, and whose component values contain the scaled RMS velocity squared $\tau v_{\text{RMS}}^2/\Delta\tau$ 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^2_{interval}.

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

$$\mathbf{Cu} \quad = \quad \mathbf{d} \tag{5.10}$$

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

$$\mathbf{0} \quad \approx \quad \mathbf{W}\,[\mathbf{Cu} - \mathbf{d}] \tag{5.11}$$

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**.

$$\mathbf{0} \quad \approx \quad \mathbf{Du} \quad = \quad \mathbf{p} \tag{5.12}$$

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

$$\mathbf{0} \quad \approx \quad \mathbf{W}\left[\mathbf{C}^2\mathbf{p} - \mathbf{d}\right] \tag{5.13}$$

$$\mathbf{0} \quad \approx \quad \epsilon\,\mathbf{p} \tag{5.14}$$

5.4.1 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.

```
module vrms2int_mod {               # Transform from RMS to interval velocity
  use causint
  use cgstep_mod
  use solver_mod
```

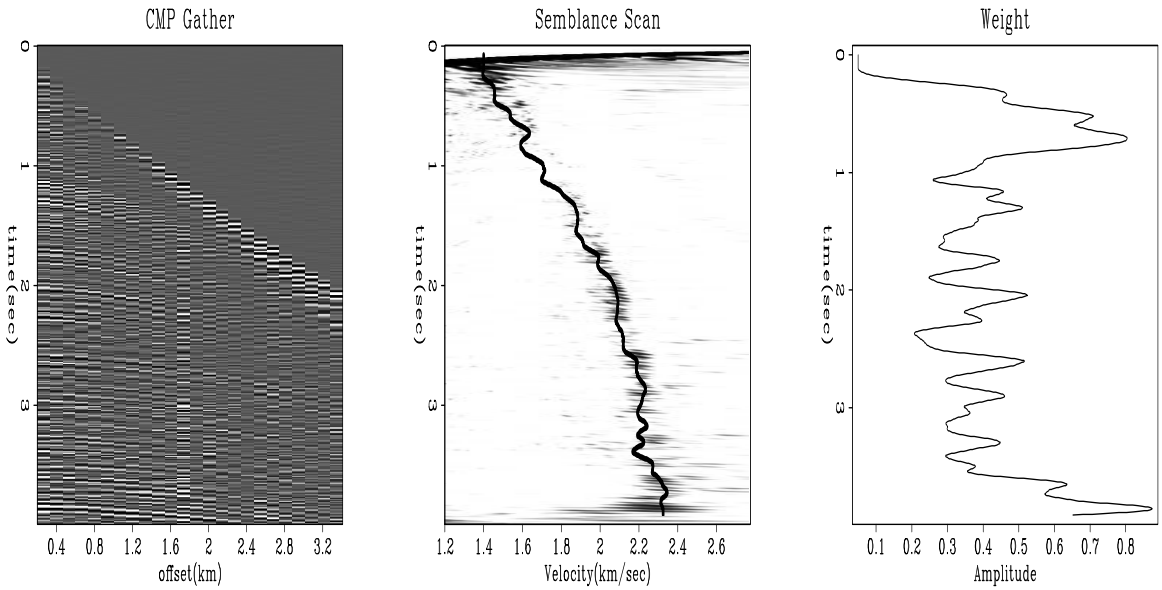

Figure 5.1: Raw CMP gather (left), Semblance scan (middle), and semblance value used for weighting function (right). (Clapp) prc-clapp [ER]

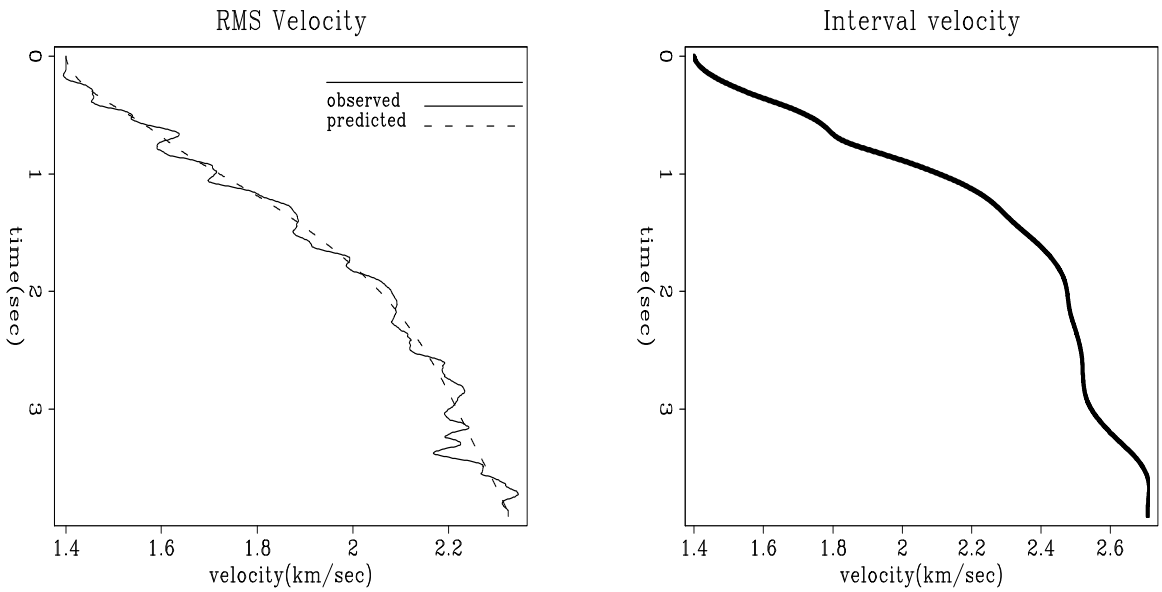

Figure 5.2: Observed RMS velocity and that predicted by a stiff model with $\epsilon = 4$. (Clapp) prc-stiff [ER]

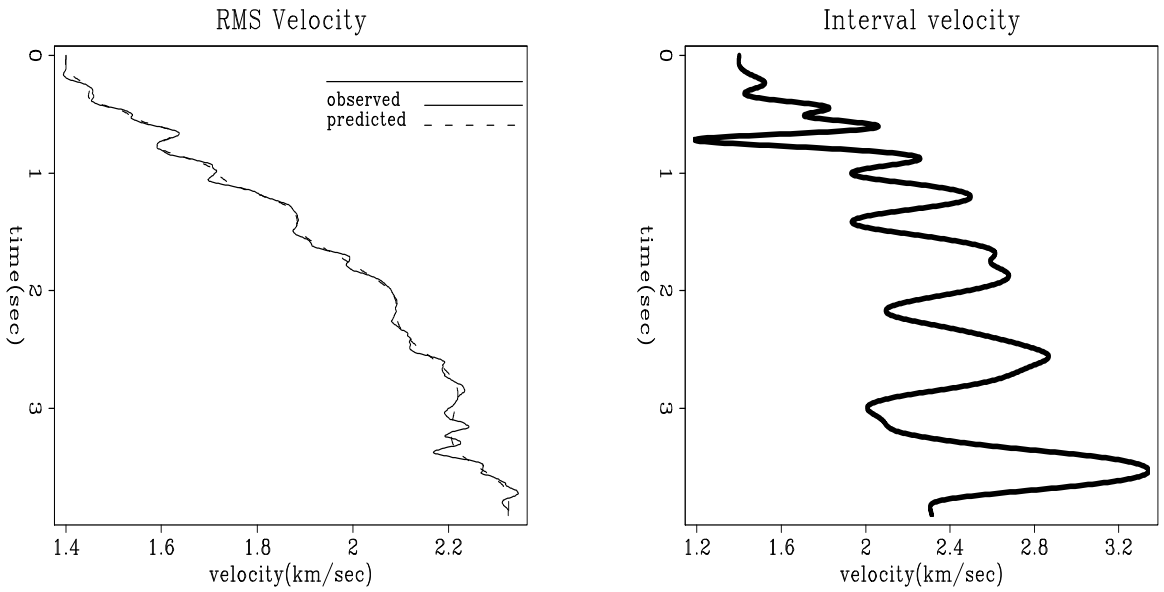

Figure 5.3: Observed RMS velocity and that predicted by a flexible model with $\epsilon = .25$ (Clapp) prc-flex [ER]

```
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(:), pointer         :: weight    # data weighting
    integer                             :: st,it,nt
    logical, dimension( size( vint))    :: mask
    real,    dimension( size( vrms))    :: dat
    nt = size( vrms)
    do it= 1, nt {
       dat( it) = vrms( it) * vrms( it) * it
       weight( 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= weight,
                      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)
    }
}
```

5.4.2 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_d^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 $\mathbf{G}$ and $\mathbf{B}$. Our fitting goals are:

$$
\begin{aligned}
3V_c^2 &\approx a + b + c && (5.15)\\
5V_e^2 &\approx a + b + c + d + e && (5.16)\\
u_0 &\approx a && (5.17)\\
0 &\approx -a + b && (5.18)\\
0 &\approx -b + c && (5.19)\\
0 &\approx -c + d && (5.20)\\
0 &\approx -d + e && (5.21)
\end{aligned}
$$

For the blocky solution, we do not want the fitting goal (1.20). We can remove it by multiplying the model goals by a diagonal-matrix badpass screen $\mathbf{B}$ eliminating the goal $0 \approx -c + d$. In abstract, our fitting goals become

$$
\begin{aligned}
\mathbf{0} &\approx \mathbf{r} = \mathbf{Cu} - \mathbf{d} && (5.22)\\
\mathbf{0} &\approx \mathbf{p} = \mathbf{BDu} - \mathbf{u}_0 && (5.23)
\end{aligned}
$$

where $\mathbf{u}_0$ is a zero vector with a top component of u_0. Since $\mathbf{B}$ is not invertable, we cannot backsolve the preconditioned variable $\mathbf{p}$ for the squared interval velocity $\mathbf{u} = \mathbf{u}_0 + \mathbf{D}^{-1}\mathbf{B}^{-1}\mathbf{p}$. Instead, we use $\mathbf{G}$ for $\mathbf{B}^{-1}$ thus redefining the implicit relationship for $\mathbf{u}$.

$$
\mathbf{u} = \mathbf{u}_0 + \mathbf{D}^{-1}\mathbf{Gp} \qquad (5.24)
$$

where $\mathbf{G}$ is the goodpass screen. Since $\mathbf{D}^{-1} = \mathbf{C}$ the fitting goals become

$$
\begin{aligned}
\mathbf{0} &\approx \mathbf{r} = \mathbf{C}^2\mathbf{Gp} + \mathbf{Cu}_0 - \mathbf{d}\\
\mathbf{0} &\approx \mathbf{p}
\end{aligned}
\qquad (5.25)
$$

After fitting with $\mathbf{p}$, we define the squared interval-velocity $\mathbf{u}$ using (1.24).

The formulation (1.25) is so logical that we might have guessed it: The goal $\mathbf{0} \approx \mathbf{p}$ says that $\mathbf{p}$ is mostly zero. What emerges from $\mathbf{G}$ is a sprinkling of impulses. Then $\mathbf{C}^2$ converts the pulses to ramp functions (zero until a certain place, then growing linearly) which are used to fit the data (integrated velocity). Differentiating the data-fitting ramps converts them to the desired blocks of constant velocity. One iteration is required for each impulse.

Choosing $\mathbf{G}$ to be an identity $\mathbf{I}$ gives smooth velocity models, such as caused by the increasing consolidation of the rocks with depth. Choosing the screen $\mathbf{G}$ to have a sprinkling of pass locations picks the boundaries of blocks of constant velocity. The choice can be made by people with subjective criteria (like geologists) or we can assist by using the data itself in various ways to select our degree of preference between the blocky and smooth models. For example, we could put seismic coherency or amplitude on the goodpass diagonal matrix $\mathbf{G}$. Clearly, much remains to be gained from experience.

5.4.3 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 $\mathbf{G}$ would naturally involve the full three-dimensional setting.

5.5 INVERSE LINEAR INTERPOLATION

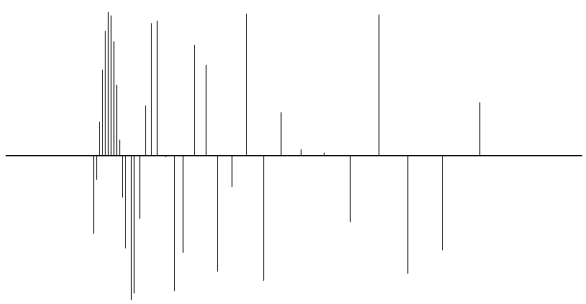

Figure 5.4: The input data are irregularly sampled. prc-data [ER]

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 $\mathbf{L}$ 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 $\mathbf{D}$ that forces model continuity (the first-order spline). An appropriate preconditioner $\mathbf{S}$ in this case is recursive causal integration. As expected, preconditioning 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 components of the model-space regularization [equations (1.8) and (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.

Following Matthias Schwab's suggestion, I measured the rate of convergence by the model residual, which is a distance from the current model to the final solution. Figure 1.6 shows that the data regularization method converged to the final solution in many fewer iterations than the model regularization. Since the cost of each iteration for each method is roughly equal, the efficiency of preconditioning is evident. Figure 1.5 shows the final solution, and the estimates from preconditioning and regularization after different numbers of conjugate-direction iterations. At early iterations, the preconditioning estimate is much closer to the final solution.

```
module invint2 {   # Inverse linear interpolation
  use lint1
  use helicon      # regularized    by helix   filtering
  use polydiv      # preconditioned by inverse filtering
  use cgstep_mod
  use solver_mod
contains
  subroutine invint1( niter, coord, ord, o1,d1, mm,mmov, eps, aa,lag, precon) {
    logical,                  intent( in)  :: precon
    integer,                  intent( in)  :: niter
    real,                     intent( in)  :: o1, d1, eps
    real,    dimension( :),   intent( in)  :: ord
    real,    dimension( :),   intent( out) :: mm
    real,    dimension( :,:), intent( out) :: mmov         # model movie
    real,    dimension( :),   pointer      :: coord, aa    # coordinate, filter
    integer, dimension( :),   pointer      :: lag          # filter lag
    call lint1_init( o1, d1, coord)
    if( precon) {                                          # preconditioning
       call polydiv_init( aa, lag)
       call solver_prec( lint1_lop, cgstep, niter = niter, x = mm, dat = ord,
                    prec = polydiv_lop, nprec = size(mm), eps = eps,
```

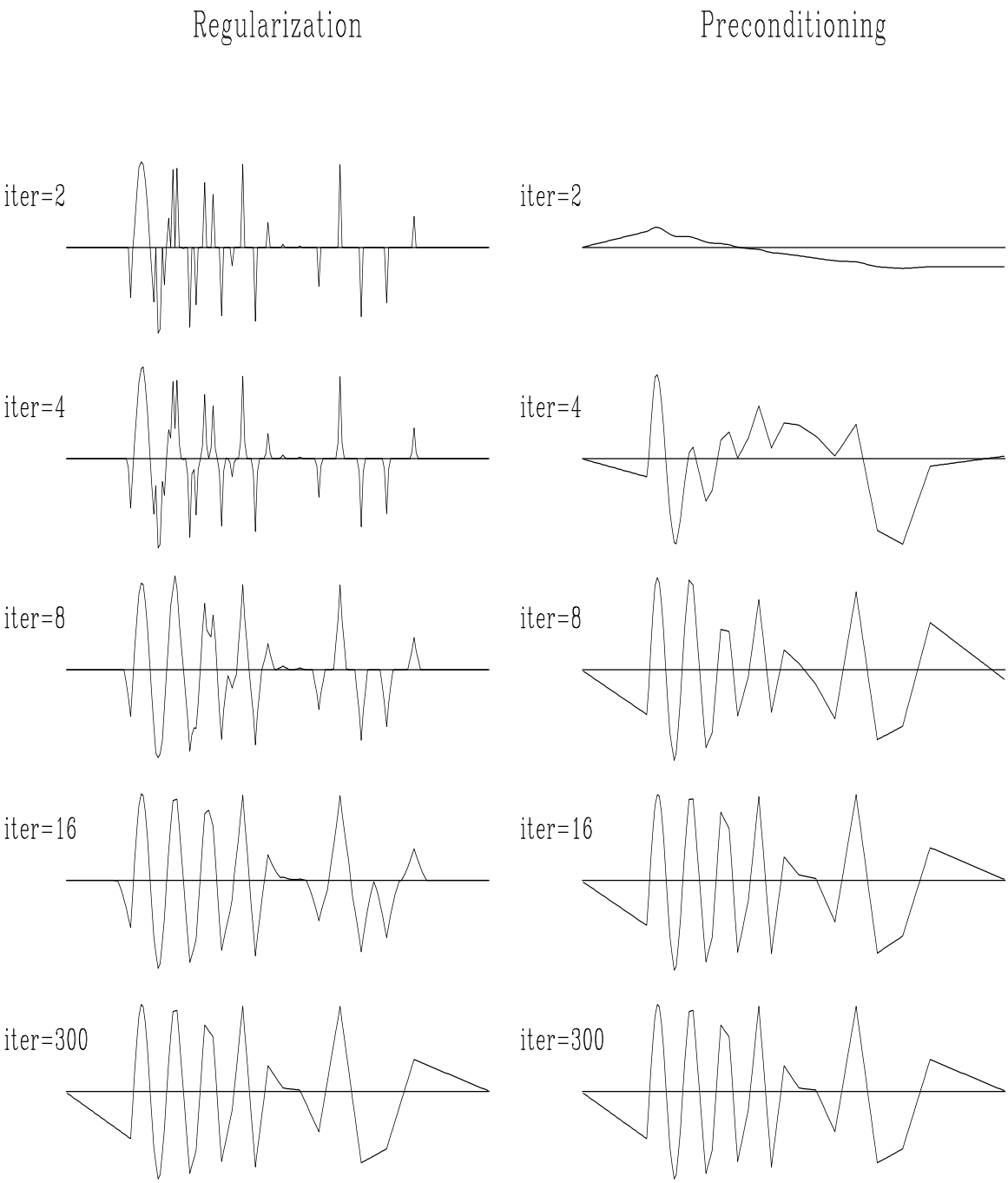

Figure 5.5: Convergence history of inverse linear interpolation. Left: regularization, right: preconditioning. The regularization operator $\mathbf{A}$ is the derivative operator (convolution with $(1, -1)$. The preconditioning operator $\mathbf{S}$ is causal integration. prc-conv1 [ER]

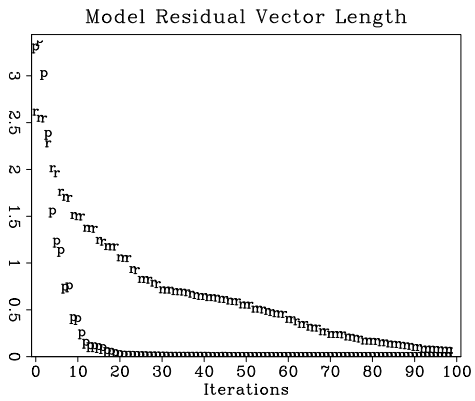

Figure 5.6: Convergence of the iterative optimization, measured in terms of the model residual. The "p" points stand for preconditioning; the "r" points, regularization. prc-schwab1 [ER]

```
                    xmov = mmov)
    } else {                                                # regularization
       call helicon_init( aa, lag)
       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()
  }
}
```

5.6 EMPTY BINS AND PRECONDITIONING

Chapter 3 formulates the basic principle of missing data interpolation as follows:

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

Mathematically, this principle can be expressed by the simple equation

$$\boldsymbol{D}\boldsymbol{m} \approx \boldsymbol{0} \ , \tag{5.26}$$

where $\boldsymbol{m}$ is the data vector, and $\boldsymbol{D}$ is the specified filter. The approximate equality sign means that equation (1.26) is solved by minimizing the squared norm (the power) of its left side. Additionally, the known data values must be preserved in the optimization scheme. Introducing the mask operator $\boldsymbol{K}$, which can be considered as a diagonal matrix with zeros on the missing data locations and ones elsewhere, we can rewrite equation (1.26) in the more rigorous form

$$\boldsymbol{D}(\boldsymbol{I}-\boldsymbol{K})\boldsymbol{m} \approx -\boldsymbol{D}\boldsymbol{K}\boldsymbol{m} = -\boldsymbol{D}\boldsymbol{m}_{\boldsymbol{k}} \ , \tag{5.27}$$

in which $\boldsymbol{I}$ is the identity operator, and $\boldsymbol{m}_{\boldsymbol{k}}$ is the known portion of the data. It is important to note that equation (1.27) corresponds to the limiting case of the

regularized linear system

$$\begin{cases} \boldsymbol{K}\boldsymbol{m} = \boldsymbol{m_k} \ , \\ \boldsymbol{\lambda}\boldsymbol{D}\boldsymbol{m} \approx \boldsymbol{0} \end{cases} \tag{5.28}$$

for the scaling coefficient $\boldsymbol{\lambda}$ approaching zero. This means that we put far more weight on the first equation in (1.28) and use the second equation only to constrain the null space of the solution. Applying the general theory of preconditioning, one can immediately transform system (1.28) to the equation

$$\boldsymbol{K}\boldsymbol{P}\boldsymbol{x} \approx \boldsymbol{m_k} \ , \tag{5.29}$$

where $\boldsymbol{P}$ is a preconditioning operator, and $\boldsymbol{x}$ is a new variable, connected with $\boldsymbol{m}$ by the simple relationship

$$\boldsymbol{m} = \boldsymbol{P}\boldsymbol{x} \ .$$

If $\boldsymbol{D}$ is represented by a discrete convolution, the natural choice for $\boldsymbol{P}$ is the corresponding deconvolution operator:

$$\boldsymbol{P} = \boldsymbol{D}^{-1} \ . \tag{5.30}$$

The helix transform provides a constructive way of implementing multidimensional deconvolution by one-dimensional recursive filtering. The following module, `mis2` on page 16, does the actual interpolation.

```
module mis2 {
  use helicon
  use polydiv
  use mask1
  use cgstep_mod
  use 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, lag, known,  precon, trans) {
    logical,                intent( in)     :: precon, trans
    integer,                intent( in)     :: niter
    logical, dimension( :), intent( in)     :: known
    real,    dimension( :), intent( in out) :: xx
    real,    dimension( :), pointer         :: aa
    integer, dimension( :), pointer         :: lag
    logical, dimension( :), pointer         :: mm
    real,    dimension( :), allocatable     :: yy, dd
    integer                                 :: nx, ny, pad, n1, n2
    nx = size( xx); pad = maxval( lag); ny = nx + 2*pad
    allocate( yy( ny), mm( ny)); yy = 0.; mm = .false.
    if (trans) { n1 = 1;       n2 = nx     # emulate transient boundaries
       mm( ny-pad+1:ny)  = .true.          # zero values beyond end.
    } else {     n1 = 2*pad+1; n2 = ny}
    yy (n1:n2) = xx; mm (n1:n2) = known
    if( precon) {                           #  KP p = K y,   m = P p
       call mask1_init( mm)
       call polydiv_init( aa, lag)
```

```
      call solver_prec( mask1_lop, cgstep, niter= niter, x= yy, dat= yy,
                  prec= polydiv_lop, nprec= ny, eps= 0.)
   } else {                                #  KA m = 0
      allocate( dd( ny));  dd = 0.
      call helicon_init( aa, lag)
      call solver( helicon_lop, cgstep, niter= niter, x = yy, dat = dd,
                  known = mm, x0 = yy)
      deallocate( dd)
   }
   call cgstep_close( )
   where( xx == 0.) xx = yy(n1:n2)
   deallocate( yy, mm)
  }
}
```

The operator `mask1`, apearing in `mis2` on page 16. is a simple masked assignment (copy under mask.)

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

5.7 VIRTUAL-RESIDUAL PRECONDITIONING

Sergey Fomel has developed a wholly different approach to preconditioning to that described above. He calls his method the "model-extension" method. I call it the "virtual-residual" method. Both names are suggestive of the method. Fomel's preconditioning method depends on some fundamental facts from the theory of least squares. I establish only one of them here before presenting his algorithm.

5.7.1 The classic underdetermined least-squares problem

We begin with the construction of theoretical data

$$\mathbf{d} \quad = \quad \mathbf{Fm} \tag{5.31}$$

We assume that there are fewer data points than model points and that the matrix $\mathbf{FF'}$ is invertible. From the theoretical data we estimate a model $\mathbf{m_0}$ with

$$\mathbf{m_0} \quad = \quad \mathbf{F'(FF')^{-1}d} \tag{5.32}$$

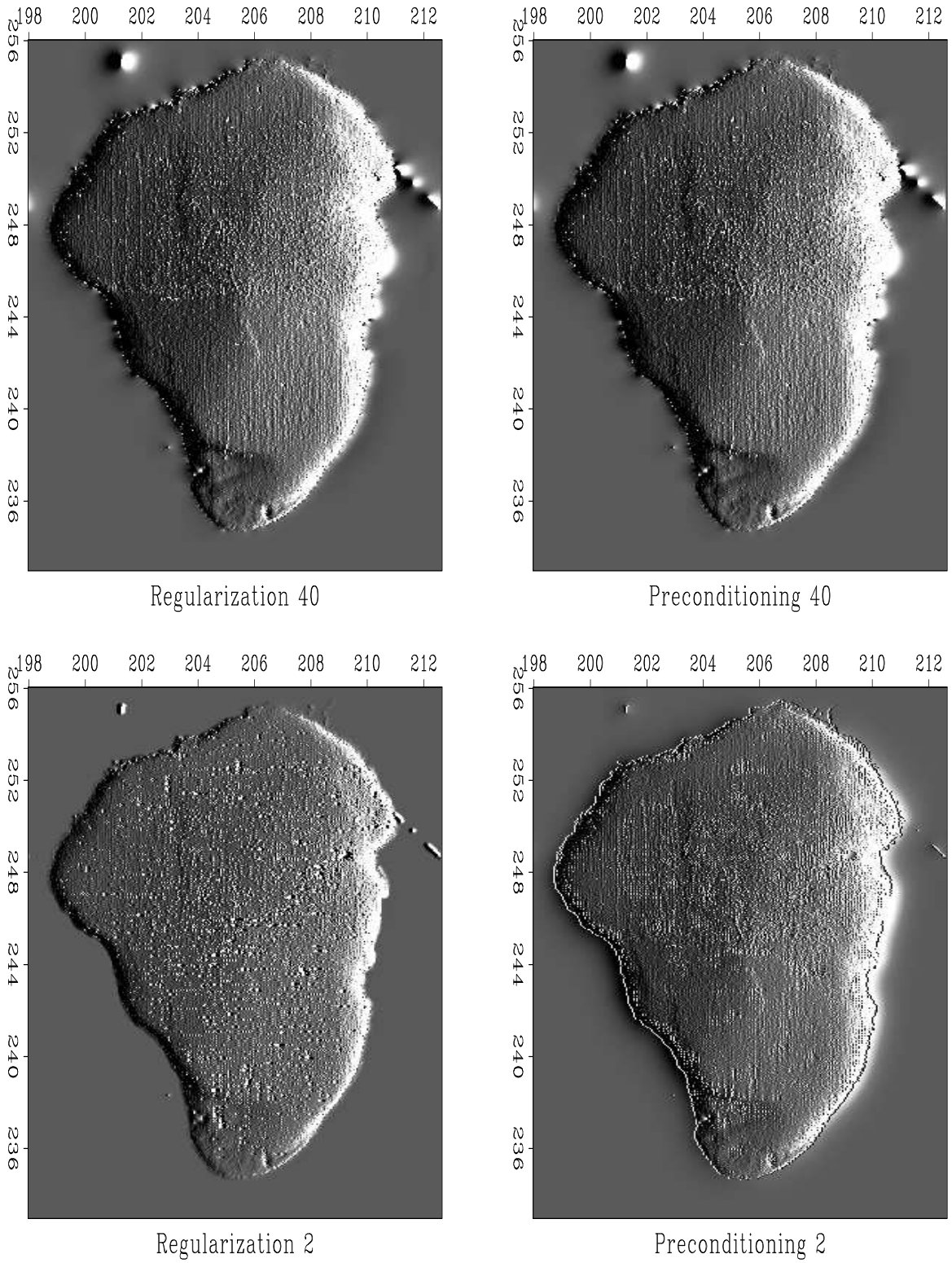

Figure 5.7: Views of the bottom of the Sea of Galilee. Missing data interpolation with preconditioning (right) and without preconditioning (left) after 40 and 2 iterations (top and bottom plots). (Fomel) prc-fill [ER]

To verify the validity of the estimate, insert the estimate (1.32) into the data modeling equation (1.31) 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 (1.32) into an intermediate result $\mathbf{d_0}$ and final result:

$$\mathbf{d_0} = (\mathbf{F}\mathbf{F}')^{-1}\mathbf{d} \tag{5.33}$$

$$\mathbf{m_0} = \mathbf{F}'\mathbf{d_0} \tag{5.34}$$

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

$$\mathbf{m} = \mathbf{m_0} + \mathbf{x} \tag{5.35}$$

which fits the theoretical data. Since $\mathbf{d} = \mathbf{F}\mathbf{m_0}$, we see that $\mathbf{x}$ is a null space vector.

$$\mathbf{F}\mathbf{x} = \mathbf{0} \tag{5.36}$$

First we see that $\mathbf{m_0}$ is orthogonal to $\mathbf{x}$ because

$$\mathbf{m_0'}\mathbf{x} = (\mathbf{F}'\mathbf{d_0})'\mathbf{x} = \mathbf{d_0'}\mathbf{F}\mathbf{x} = \mathbf{d_0'}\mathbf{0} = \mathbf{0} \tag{5.37}$$

Therefore,

$$\mathbf{m}'\mathbf{m} = \mathbf{m_0'}\mathbf{m_0} + \mathbf{x}'\mathbf{x} + \mathbf{2x}'\mathbf{m_0} = \mathbf{m_0'}\mathbf{m_0} + \mathbf{x}'\mathbf{x} \geq \mathbf{m_0'}\mathbf{m_0} \tag{5.38}$$

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 $\mathbf{m_0} = \mathbf{0}$ or any $\mathbf{m_0}$ with no null-space component. In (1.37) we see that the orthogonality $\mathbf{m_0'}\mathbf{x} = \mathbf{0}$ 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 $\mathbf{\Delta m} = \mathbf{F}'\mathbf{r}$ which is of the required form.

5.7.2 The virtual-residual preconditioning algorithm

Now we are prepared for Sergey Fomel's model-extended preconditioning. We begin with the usual definition of preconditioned variables $\mathbf{m} = \mathbf{S}\mathbf{p}$ and the opposite of our usual sign convention for the residual.

$$-\boldsymbol{\epsilon}\,\mathbf{r} = \mathbf{F}\mathbf{m} - \mathbf{d} \tag{5.39}$$

$$-\boldsymbol{\epsilon}\,\mathbf{r} = \mathbf{F}\mathbf{S}\mathbf{p} - \mathbf{d} \tag{5.40}$$

which we arrange with a column vector of unknowns that includes not only $\mathbf{p}$ but also $\mathbf{r}$.

$$\mathbf{0} \quad \approx \quad \hat{\mathbf{r}} \quad = \quad [\mathbf{FS} \quad \epsilon\, \mathbf{I}] \begin{bmatrix} \mathbf{p} \\ \mathbf{r} \end{bmatrix} \; - \; \mathbf{d} \tag{5.41}$$

The interesting thing is that we can use our familiar conjugate-gradient programs with this system of equations. We only need to be careful to distinguish the residual in the bottom half of our solution vector, from the virtual residual $\hat{\mathbf{r}}$ (which should vanish identically) in the iterative fitting problem. The fitting begins with the initializations:

$$\hat{\mathbf{p}}_0 \quad = \quad \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{0} \end{bmatrix} \tag{5.42}$$

and

$$\hat{\mathbf{r}}_0 \quad = \quad \mathbf{FSp}_0 - \mathbf{d} \quad = \quad \mathbf{Fm}_0 - \mathbf{d} \tag{5.43}$$

As the iteration begins we have gradients of the two parts of the model

$$\mathbf{g}_m \;=\; \mathbf{S}'\mathbf{F}'\hat{\mathbf{r}} \tag{5.44}$$

$$\mathbf{g}_d \;=\; \epsilon\, \hat{\mathbf{r}} \tag{5.45}$$

which imply a perturbation in the theoretically zero residual

$$\Delta\hat{\mathbf{r}} \quad = \quad \mathbf{FSg}_m + \epsilon\, \mathbf{g}_d \tag{5.46}$$

Then step sizes and steps are determined as usual for conjugate-direction fitting.

The preconditioning module `vr_solver` on this page takes three functions as its arguments. Functions `oper` and `prec` correspond to the linear operators $\mathbf{F}$ and $\mathbf{S}$. Function `solv` implements one step of an optimization descent. Its arguments are a logical parameter `forget`, which controls a conditional restarting of the optimization, the current effective model `x0`, the gradient vector `g`, the data residual vector `rr`, and the conjugate gradient vector `gg`. Subroutine `solver_vr` constructs the effective model vector `x`, which consists of the model-space part `xm` and the data-space part `xd`. Similarly, the effective gradient vector `g` is split into the the model-space part `gm` and the data-space part `gd`.

For brevity I am omitting the code for the virtual-residual solver `vrsolver` which is in the library. It follows the same pattern as `prec_solver` on page 6.

5.8 SCALING THE ADJOINT

Given the usual linearized fitting goal between data space and model space, $\mathbf{d} \approx \mathbf{Fm}$, 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 $\mathbf{m}_{\mathbf{theor}} = (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{d}$ 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 $\hat{\mathbf{m}} = \mathbf{W^2 F' d}$ where we choose the weighting function to be

$$\mathbf{W^2} \quad = \quad \mathbf{diag}\left(\frac{\mathbf{F'd}}{\mathbf{F'FF'd}}\right) \tag{5.47}$$

which obviously has the correct physical units. The weight $\mathbf{W^2}$ can be thought of as a diagonal matrix containing the ratio of two images. A problem with the choice (1.47) is that the denominator might vanish or might even be negative. The way to stabilize any ratio is suggested at the beginning of Chapter 2; that is, we revise the ratio (1.47) by changing $\mathbf{W^2} = \mathbf{diag}(\boldsymbol{a}/\boldsymbol{b})$ to

$$\mathbf{W^2} \quad = \quad \mathbf{diag}\left(\frac{< \boldsymbol{ab} >}{< \boldsymbol{b^2} + \boldsymbol{\epsilon^2} >}\right) \tag{5.48}$$

where $\boldsymbol{\epsilon}$ is a parameter to be chosen, and the angle braces indicate the possible need for local smoothing.

Because a scaled adjoint is a guess at the solution to the fitting problem, it is logical to choose values for $\boldsymbol{\epsilon}$ and the smoothing parameters that give fastest convergence of the conjugate-direction solver.

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 $\mathbf{m} = \mathbf{Wp}$. Then $\mathbf{d} \approx \mathbf{Fm} = \mathbf{FWp}$. To find $\mathbf{p}$ instead of $\mathbf{m}$, we do CD iteration with the operator $\mathbf{FW}$ instead of with $\mathbf{F}$. As usual, the first step of the iteration is to use the adjoint of $\mathbf{d} \approx \mathbf{FWp}$ to form the image $\hat{\mathbf{p}} = \mathbf{(FW)'d}$. At the end of the iterations, we convert from $\mathbf{p}$ back to $\mathbf{m}$ with $\mathbf{m} = \mathbf{Wp}$. The result after the first iteration $\hat{\mathbf{m}} = \mathbf{W}\hat{\mathbf{p}} = \mathbf{W(FW)'d} = \mathbf{W^2 F' d}$ turns out to be the same as Symes scaling.

By (1.47), $\mathbf{W}$ has physical units inverse to $\mathbf{F}$. Thus the transformation $\mathbf{FW}$ has no units so the $\mathbf{p}$ variables have physical units of data space. It might be more practical to view the solution $\mathbf{p}$ with data units than to view the solution $\mathbf{m}$ with the more theoretical model units.

Some experience tells me that the ideas of this section are defective. Appropriate scaling is required in both data space and model space. We need both $\mathbf{W_1}$ and $\mathbf{W_2}$ where $\hat{\mathbf{m}} = \mathbf{W_1 F' W_2 d}$.

I have a useful practical example (stacking in $\boldsymbol{v}(\boldsymbol{z})$ media) in another of my electronic books (BEI), where I found both $\mathbf{W_1}$ and $\mathbf{W_2}$ by iterative guessing. But I don't know how to give you a general strategy. I feel this is a major unsolved(?) opportunity for someone.

5.9 ACKNOWLEDGEMENTS

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

Index

invint2 module, 13

mask1 operator module, 17
mis2 module, 16
module
    `invint2`, Inverse linear interpolation, 13
    `mis2`, Missing data interpolation with and without preconditioning, 16
    `precsolver`, Preconditioned solver, 6
    `vrms2int`, Converting RMS to interval velocity, 8

operator
    `mask1`, copy under mask, 17

precsolver module, 6

vrms2int module, 8