Preconditioning

Jon Claerbout

In Chapter **??**, we developed adjoints and in Chapter **??**, we developed inverse operators. Logically, correct solutions come only through inversion. Real life, however, seems nearly the opposite. This situation is puzzling but intriguing. It seems an easy path to fame and profit would be to go beyond adjoints by introducing some steps of inversion. It is not that easy. Images contain so many unknowns. Mostly, we cannot iterate to completion and need concern ourselves with the rate of convergence. Often, necessity limits us to a handful of iterations whereby in principle, millions or billions are required.

When you fill your car with gasoline, it derives more from an adjoint than an inverse. Industrial seismic data processing relates more to adjoints than to inverses though there is a place for both, of course. It cannot be much different with medical imaging.

First consider cost. For simplicity, consider a data space with N values and a model (or image) space of the same size. The computational cost of applying a dense adjoint operator increases in direct proportion to the number of elements in the matrix, in this case N^2. To achieve the minimum discrepancy between modeled data and observed data (inversion) theoretically requires N iterations raising the cost to N^3.

Consider an image of size $m \times m = N$. Continuing, for simplicity, to assume a dense matrix of relations between model and data, the cost for the adjoint is m^4, whereas, the cost for inversion is m^6. We consider computational costs for the year 2000, but noticing that costs go as the sixth power of the mesh size, the overall situation will not change much in the foreseeable future. Suppose you give a stiff workout to a powerful machine; you take an hour to invert a 4,096$\times$ 4,096 matrix. The solution, a vector of 4096 components could be laid into an image of size 64 $\times$ 64= $2^6 \times 2^6$ = 4,096. Here is what we are looking at for costs:

adjoint cost	$(m \times m)^2$	$(512 \times 512)^2$	$(2^9 2^9)^2$	2^{36}
inverse cost	$(m \times m)^3$	$(64 \times 64)^3$	$(2^6 2^6)^3$	2^{36}

These numbers tell us that for applications with dense operators, the biggest images that we are likely to see coming from inversion methods are 64×64, whereas, those from adjoint methods are 512×512. For comparison, your vision is comparable to your computer screen at 1,000 $\times$ 1,000.

`http://sepwww.stanford.edu/sep/jon/family/jos/gifmovie.html` holds a movie blinking between Figures 1 and 2.

This cost analysis is oversimplified in that most applications do not require dense operators. With sparse operators, the cost advantage of adjoints is even more pronounced because for adjoints, the cost savings of operator sparseness translate directly to real cost savings. The situation is less favorable and more muddy for inversion. The reason that Chapter 2 covers iterative methods and neglects exact methods is that in practice iterative

Figure 1: Jos greets Andrew, "Welcome back Andrew" from the Peace Corps. At a resolution of 512 × 512, this picture is approximately the same as the resolution as the paper on which it is printed, or the same as your viewing screen, if you have scaled it up to 50% of screen size.

methods are not run to theoretical completion, but until we run out of patience. But that leaves hanging the question of what percent of theoretically dictated work is actually necessary. If we struggle to accomplish merely one percent of the theoretically required work, can we hope to achieve anything of value?

Cost is a big part of the story, but the story has many other parts. Inversion, while being the only logical path to the best answer, is a path littered with pitfalls. The first pitfall is that the data is rarely able to determine a complete solution reliably. Generally, there are aspects of the image that are not learnable from the data.

Figure 2: Jos greets Andrew, "Welcome back Andrew" again. At a resolution of 64 × 64, the pixels are clearly visible. From far the pictures are the same. From near, examine their glasses.

When I first realized that practical imaging methods with wide 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 toward 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 optimism was soon exposed. Improvements came too slowly. But then, I learned about the conjugate gradient method that spectacularly overcomes a well-known speed problem with the method of steepest descent. I came to realize it was still too slow. I learned by watching the convergence in Figure 8, which led me to the helix method in Chapter 4. Here we see how it speeds many applications.

We also come to understand why the gradient is such a poor direction both for steepest descent and conjugate gradients. An indication of our path is found in the contrast between an exact solution and the gradient.

$$\mathbf{m} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{d} \tag{1}$$

$$\Delta\mathbf{m} = \mathbf{A}^{\mathrm{T}}\mathbf{d} \tag{2}$$

Equations (1) and (2) differ by the factor $(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}$. This factor is sometimes called a spectrum, and in some situations, it literally is a frequency spectrum. Our updates do not have the spectrum of the thing we are trying to build. No wonder it's slow! Here we find for many applications that "preconditioning" with the helix is a better way.

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:

$$\mathbf{m} = \mathbf{S}\mathbf{p} \tag{3}$$

where $\mathbf{m}$ is the map we seek, columns of the matrix $\mathbf{S}$ are "shapes" 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 see later that invert-ability is helpful. Inserting the trial solution $\mathbf{m} = \mathbf{S}\mathbf{p}$ into $\mathbf{0} \approx \mathbf{F}\mathbf{m} - \mathbf{d}$ gives:

$$\mathbf{0} \approx \mathbf{F}\mathbf{m} - \mathbf{d} \tag{4}$$

$$\mathbf{0} \approx \mathbf{F}\mathbf{S}\mathbf{p} - \mathbf{d} \tag{5}$$

We pass the operator $\mathbf{FS}$ to our iterative solver. After finding the best fitting $\mathbf{p}$, we merely evaluate $\mathbf{m} = \mathbf{S}\mathbf{p}$ 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 $\mathbf{F}^{\mathrm{T}}$, we have iterative application of $\mathbf{FS}$ and $\mathbf{S}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}$.

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 $\mathbf{S} \approx \mathbf{F}^{-1}$. Where I have known such $\mathbf{S}$ matrices, I have often found that convergence is accelerated, but not by 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 tall rectangular matrix reduces the number of adjustable parameters, changes the solution, gets it quicker, but lowers resolution.

Preconditioner with a starting guess

We often have a starting solution $\mathbf{m}_0$. 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 $\mathbf{S}$.

We solve this problem using a shifted unknown $\tilde{\mathbf{m}}$.

$$
\begin{array}{rcll}
\mathbf{0} &\approx& \mathbf{F}\mathbf{m} - \mathbf{d} & \text{typical regression} \\
\mathbf{0} &\approx& \mathbf{F}(\tilde{\mathbf{m}} + \mathbf{m}_0) - \mathbf{d} & \text{Define } \mathbf{m} = \tilde{\mathbf{m}} + \mathbf{m}_0 \\
\mathbf{0} &\approx& \mathbf{F}\tilde{\mathbf{m}} + \mathbf{F}\mathbf{m}_0 - \mathbf{d} & \\
\mathbf{0} &\approx& \mathbf{F}\tilde{\mathbf{m}} - \tilde{\mathbf{d}} & \text{Defines } \tilde{\mathbf{d}} \\
& & & \text{Implicitly define } \mathbf{p} \text{ by } \tilde{\mathbf{m}} = \mathbf{S}\mathbf{p}. \\
\mathbf{0} &\approx& \mathbf{F}\mathbf{S}\mathbf{p} - \tilde{\mathbf{d}} & \text{You iterate for } \mathbf{p}. \\
\tilde{\mathbf{m}} &=& \mathbf{S}\mathbf{p} & \text{from your definition} \\
\mathbf{m} &=& \tilde{\mathbf{m}} + \mathbf{m}_0 & \text{Got the answer.}
\end{array}
$$

which solves the problem never needing $\mathbf{S}^{-1}$. Unfortunately, as we see later, this conclusion is only valid while there is no regularization.

Guessing the preconditioner

We are tasked with coming up with "trial solution"—a pretty vague assignment. Some kind of a scaling, smoothing, or shaping transformation $\mathbf{S}$ of some mysterious "preconditioned space" $\mathbf{p}$ should represent the model $\mathbf{m}$ we seek. We begin by investigating how the shaper $\mathbf{S}$ alters the gradient.

$$
\begin{array}{rcll}
\mathbf{m} &=& \mathbf{S}\mathbf{p} & \text{introduces } \mathbf{S}\text{, implicitly defines } \mathbf{p} \\
\Delta\mathbf{m} &=& \mathbf{S}\Delta\mathbf{p} & \text{consequence of the above} \\
\Delta\mathbf{m} &=& \mathbf{F}^{\mathrm{T}}\mathbf{r} & \text{gradient is adjoint upon residual} \\
\mathbf{0} \approx \mathbf{r} &=& \mathbf{F}\mathbf{m} - \mathbf{d} & \text{residual in terms of } \mathbf{m} \\
\mathbf{r} &=& \mathbf{F}(\mathbf{S}\mathbf{p}) - \mathbf{d} & \text{residual in terms of } \mathbf{p} \\
\mathbf{0} \approx \mathbf{r} &=& (\mathbf{F}\mathbf{S})\mathbf{p} - \mathbf{d} & \text{reordering calculation} \\
\Delta\mathbf{p} &=& (\mathbf{F}\mathbf{S})^{\mathrm{T}}\mathbf{r} & \text{gradient is adjoint upon residual} \\
\Delta\mathbf{p} &=& \mathbf{S}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}\mathbf{r} & \text{reordering} \\
\Delta\mathbf{m} &=& (\mathbf{S}\mathbf{S}^{\mathrm{T}})\mathbf{F}^{\mathrm{T}}\mathbf{r} & \text{recalling } \Delta\mathbf{m} = \mathbf{S}\Delta\mathbf{p}
\end{array}
$$

We may compare the gradient $\Delta\mathbf{m}$ with and without preconditioning.

$$
\begin{array}{rcll}
\Delta\mathbf{m} &=& \mathbf{F}^{\mathrm{T}}\mathbf{r} & \text{original} \\
\Delta\mathbf{m} &=& (\mathbf{S}\mathbf{S}^{\mathrm{T}})\mathbf{F}^{\mathrm{T}}\mathbf{r} & \text{with preconditioning transformation}
\end{array}
$$

When the first vanishes, the second also vanishes. When the second vanishes, the first vanishes provided $(\mathbf{S}\mathbf{S}^{\mathrm{T}})$ is a nonsingular matrix. As our choice of $\mathbf{S}$ is quite arbitrary, it is marvelous the freedom we have to monkey with the gradient.

Remember that $\mathbf{r}$ starts off being $-\mathbf{d}$. Compare the $(\mathbf{S}\mathbf{S}^{\mathrm{T}})$ scaled gradient to the analytic solution.

$$\begin{aligned} \Delta\mathbf{m} &= (\mathbf{S}\mathbf{S}^{\mathrm{T}})\ \mathbf{F}^{\mathrm{T}}\mathbf{r} && \text{modified gradient} \\ \mathbf{m} &= (\mathbf{F}^{\mathrm{T}}\mathbf{F})^{-1}\mathbf{F}^{\mathrm{T}}\mathbf{d} && \text{analytic solution} \end{aligned}$$

Mathematically, we see it would be delightful if $(\mathbf{S}\mathbf{S}^{\mathrm{T}})$ were something like $(\mathbf{F}^{\mathrm{T}}\mathbf{F})^{-1}$, but we rarely have ideas how to arrange it. We do, however, have some understanding of the world of images, and understand where on the image we would like iterations to concentrate first, and what spatial frequencies are more relevant than others. If we cannot go all the way, as we cannot in giant imaging problems, it is important to make the important steps early.

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:

$$\mathbf{0} \approx \mathbf{F}\mathbf{m} - \mathbf{d} \tag{6}$$

$$\mathbf{0} \approx \mathbf{A}\mathbf{m} \tag{7}$$

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

To fix ideas, let us examine a toy example. The data and the first three rows of the following matrix 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   Sum(|grad|)

-100.     62.  18.   2.  75.  99.  45.  93. -41. -15.  80.      1    69262.0000
 -83.     31.  80.  92. -67.  72.  81. -41.  87. -17. -38.      2    19012.8203
  20.      3. -21.  58.  38.   9.  18. -81.  22. -14.  20.      3    10639.0791
   0.    100.-100.   0.   0.   0.   0.   0.   0.   0.   0.      4     4578.7988
   0.      0. 100.-100.   0.   0.   0.   0.   0.   0.   0.      5     2332.3352
   0.      0.   0. 100.-100.   0.   0.   0.   0.   0.   0.      6     1676.6978
   0.      0.   0.   0. 100.-100.   0.   0.   0.   0.   0.      7      622.7415
   0.      0.   0.   0.   0. 100.-100.   0.   0.   0.   0.      8      454.1242
   0.      0.   0.   0.   0.   0. 100.-100.   0.   0.   0.      9      290.6053
   0.      0.   0.   0.   0.   0.   0. 100.-100.   0.   0.     10      216.0749
   0.      0.   0.   0.   0.   0.   0.   0. 100.-100.   0.     11        1.0488
   0.      0.   0.   0.   0.   0.   0.   0.   0. 100.-100.    12        0.0061
   0.      0.   0.   0.   0.   0.   0.   0.   0.   0. 100.    13        0.0000
```

The right-most column shows the sum of the absolute values of the gradient. Notice at the 11th iteration, the gradient suddenly plunges. Because there are ten unknowns and the

matrix is obviously full-rank, conjugate-gradient theory tells us to expect the exact solution at the 11th iteration. This sudden convergence is the first miracle of conjugate gradients. Failure to achieve a precisely zero gradient at the 11th step is a precision issue that could be addressed with double precision arithmetic. The residual magnitude (not shown) does not approach zero, because 13 linear equations defeat the ten adjustable coefficients.

The second miracle of conjugate gradients

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

```
 d(m)     F(m,n)                                                 iter   Sum(|grad|)

-100.     62.  18.   2.  75.  99.  45.  93. -41. -15.  80.      1  69262.0000
 -83.     31.  80.  92. -67.  72.  81. -41.  87. -17. -38.      2   5486.2095
  20.      3. -21.  58.  38.   9.  18. -81.  22. -14.  20.      3   2755.6702
   0.    100.   0.   0.   0.   0.   0.   0.   0.   0.   0.      4      0.0012
   0.      0. 100.   0.   0.   0.   0.   0.   0.   0.   0.      5      0.0011
   0.      0.   0. 100.   0.   0.   0.   0.   0.   0.   0.      6      0.0006
   0.      0.   0.   0. 100.   0.   0.   0.   0.   0.   0.      7      0.0006
   0.      0.   0.   0.   0. 100.   0.   0.   0.   0.   0.      8      0.0005
   0.      0.   0.   0.   0.   0. 100.   0.   0.   0.   0.      9      0.0005
   0.      0.   0.   0.   0.   0.   0. 100.   0.   0.   0.     10      0.0012
   0.      0.   0.   0.   0.   0.   0.   0. 100.   0.   0.     11      0.0033
   0.      0.   0.   0.   0.   0.   0.   0.   0. 100.   0.     12      0.0033
   0.      0.   0.   0.   0.   0.   0.   0.   0.   0. 100.     13      0.0000
```

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

Practitioners usually do not like the identity operator for model-space 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 the regularization operator becomes an identity matrix. Basically, such a transformation reduces the iteration count from something the size of the model space to something the size of the data space. Such a transformation is called "preconditioning."

More generally, the model goal $\mathbf{0} \approx \mathbf{Am}$ introduces a roughening operator like a gradient, a Laplacian, or in Chapter **??**, a Prediction-Error Filter (PEF). Thus, the model goal is usually a filter, unlike the data-fitting goal that 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:

$$\begin{aligned} \mathbf{0} &\approx \mathbf{FSp} - \mathbf{d} \\ \mathbf{0} &\approx \mathbf{ASp} \end{aligned} \tag{8}$$

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 freedom of choice suggests 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:

$$\begin{aligned} \mathbf{0} &\approx \mathbf{FSp} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\, \mathbf{p} \end{aligned} \tag{9}$$

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

The solution $\mathbf{m}$ is likely to come out smooth, because we typically over-sample axes of physical quantities. We typically penalize roughness in it by our choice of a regularization operator which means the preconditioning variable $\mathbf{p}$ typically has a wider frequency bandwidth than $\mathbf{m}$. In Chapter **??**, we see how to make the spectrum of $\mathbf{p}$ come out white (tending to flat spectrum).

Importance of scaling

Another simple toy example shows us the importance of scaling. We use the same example as the previous one, except we make the diagonal penalty function vary slowly with location.

```
 d(m)     F(m,n)                                                 iter   Sum(|grad|)

-100.     62.  16.   2.  53.  59.  22.  37. -12.  -3.   8.      1  42484.1016
 -83.     31.  72.  74. -47.  43.  40. -16.  26.  -3.  -4.      2   8388.0635
  20.      3. -19.  46.  27.   5.   9. -32.   7.  -3.   2.      3   4379.3032
   0.    100.   0.   0.   0.   0.   0.   0.   0.   0.   0.      4   1764.9844
   0.      0.  90.   0.   0.   0.   0.   0.   0.   0.   0.      5    868.9418
   0.      0.   0.  80.   0.   0.   0.   0.   0.   0.   0.      6    502.5179
   0.      0.   0.   0.  70.   0.   0.   0.   0.   0.   0.      7    450.0512
   0.      0.   0.   0.   0.  60.   0.   0.   0.   0.   0.      8    185.2923
   0.      0.   0.   0.   0.   0.  50.   0.   0.   0.   0.      9    247.1021
   0.      0.   0.   0.   0.   0.   0.  40.   0.   0.   0.     10    338.7060
   0.      0.   0.   0.   0.   0.   0.   0.  30.   0.   0.     11    119.5686
   0.      0.   0.   0.   0.   0.   0.   0.   0.  20.   0.     12     34.3372
   0.      0.   0.   0.   0.   0.   0.   0.   0.   0.  10.     13      0.0000
```

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 11th iteration is delayed until the 13th.

Another curious fact may be noted here. The gradient does not decrease monotonically. It is known theoretically that the residual does decrease monotonically, but the gradient need not. I did not show the norm of the residual, because I wanted to display a function that vanishes at convergence, and the residual does not.

You better make your residuals IID!

In the statistical literature is a concept that repeatedly arises, the idea that some statistical variables are IID, namely Independent, Identically Distributed. In practice, we see many random-looking variables, some much closer than others to IID. Theoretically, the ID part of IID means the random variables come from Identical probability Density functions. In

practice, the ID part mostly means the variables have the same variance. The "I" before the ID means the variables are statistically Independent of one another. Neighboring values should not be positively correlated, meaning low frequencies are present. In the subject area of this book, signals, images, and Earth volumes, the "I" before the ID means our residual spaces are white—have all frequencies present in roughly equal amounts. In other words the "I" means the statistical variables have no significant correlation in time or space. Chapter **??** gives a method of finding a filter as a model styler (regularizer) that accomplishes this goal. IID random variables have fairly uniform variance in both physical space and in Fourier space.

> IID random variables have uniform variance in both physical space and Fourier space.

In a geophysical project, it is important the residual between observed data and modeled data is not far from IID. To raw residuals, we should apply weights and filters to get IID residuals. We minimize sums of squares of residuals. If any residuals are small, the squares are tiny, so such regression equations are effectively ignored. We would hardly ever want residuals ignored. Echo seismograms get weak at late time. So, even with a bad fit, the difference between real and theoretical seismograms is necessarily weak at late times. We do not want the data at late times to be ignored. So, we boost up the residual there. We choose $\mathbf{W}$ to be a diagonal matrix that boosts late times in the regression $\mathbf{0} \approx \mathbf{r} = \mathbf{W}(\mathbf{Fm} - \mathbf{d})$.

An example with too much low (spatial) frequency in a residual might arise in a topographic study. It is not unusual for the topographic wavelength to exceed the survey size. Here, we should choose $\mathbf{W}$ to be a filter to boost up the higher frequencies. Perhaps, $\mathbf{W}$ should contain a derivative or a Laplacian. If you set up and solve a data-modeling problem and then find $\mathbf{r}$ is not IID, you should consider changing your $\mathbf{W}$. Chapter **??** provides a systematic approach to whitening residuals.

Now, let us include regularization $\mathbf{0} \approx \mathbf{Am}$ and a preconditioning variable $\mathbf{p}$. We have our data-fitting goal and our model-styling goal; the first with a residual $\mathbf{r}_d$ in data space, the second with a residual $\mathbf{r}_m$ in model space. We have had to choose a regularization operator $\mathbf{A} = \mathbf{S}^{-1}$ and a scaling factor ϵ.

$$0 \;\approx\; \mathbf{r}_d \;=\; \mathbf{W}(\mathbf{FSp} - \mathbf{d}) \;=\; \tilde{\mathbf{F}}\mathbf{Sp} - \tilde{\mathbf{d}} \tag{10}$$

$$0 \;\approx\; \mathbf{r}_m \;=\; \epsilon\, \mathbf{p} \tag{11}$$

This system of two regressions could be packed into one; the two residual vectors stacked on top of each other, likewise the operators $\mathbf{F}$ and $\epsilon\mathbf{I}$. The IID notion seems to apply to this unified system which gives us a clue as to how we should have chosen the regularization operator $\mathbf{A}$. Not only should $\mathbf{r}_d$ be IID, but also should $\mathbf{r}_m$—within a scale ϵ, $\mathbf{r}_m = \mathbf{p}$. Thus, the preconditioning variable is not simply something to speed computational convergence. It is a variable that should be IID. If it is not coming out that way, we should consider changing $\mathbf{A}$. Chapter **??** addresses the task of choosing an $\mathbf{A}$, so $\mathbf{r}_m$ comes out IID.

> We should choose a weighting function (and/or operator) $\mathbf{W}$, so data residuals are IID. We should also choose our regularization operator $\mathbf{A} = \mathbf{S}^{-1}$ so the preconditioning variable $\mathbf{p}$ comes out IID.

Choice of a unitless epsilon

The parameter epsilon ϵ strikes the balance between our data-fitting goal and our model-styling goal. These two regression systems typically have differing physical units; therefore, the numerical value of ϵ is accidental, for example comparing milliseconds to meters.

$$\begin{array}{lcl} \mathbf{0} & \approx & \mathbf{r}_d = \mathbf{W}(\mathbf{FSp} - \mathbf{d}) \\ \mathbf{0} & \approx & \mathbf{r}_m = \epsilon\ \mathbf{p} \end{array} \tag{12}$$

The numerical value of ϵ is meaningless before we learn to express the idea in a unitless (dimensionless) manner. Without pretending we are doing physics, let us use some of the language of thermodynamics, a physical field that does deal with equilibria and random fluctuations. Define an energy ratio u and a volume ratio v that can be used to bring ϵ to unitless form. Naturally, the square roots arise, because we are minimizing quadratic functions of residuals.

$$u = \text{energy ratio} \quad = \quad \sqrt{\frac{\mathbf{r}_d \cdot \mathbf{r}_d}{\mathbf{p} \cdot \mathbf{p}}}$$

$$v = \text{volume ratio} \quad = \quad \sqrt{\frac{n_{r_d}}{n_p}}$$

Can we really think of "volume" as related to the number n_p of components in the model space? Perhaps. Likewise the data space? Less likely. And, is the energy measure really an appropriate one? Maybe. What is the goal of these speculative thoughts? The goal is to give you a starting numerical value for ϵ, say $\epsilon = 1$. Your final guide is your own experimental experience. Try either one of these next two regressions:

$$\mathbf{0} \approx \mathbf{r}_m = \epsilon_{\text{extrinsic}}\ u\ \mathbf{p} \tag{13}$$

$$\mathbf{0} \approx \mathbf{r}_m = \epsilon_{\text{intrinsic}}\ (u/v)\ \mathbf{p} \tag{14}$$

THE PRECONDITIONED SOLVER

Summing up the previous ideas, we start from fitting goals:

$$\begin{array}{lcl} \mathbf{0} & \approx & \mathbf{Fm} - \mathbf{d} \\ \mathbf{0} & \approx & \mathbf{Am} \end{array} \tag{15}$$

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

$$\begin{array}{lcl} \mathbf{0} & \approx & \mathbf{Fm} - \mathbf{d} = \mathbf{FA}^{-1}\ \mathbf{p} - \mathbf{d} \\ \mathbf{0} & \approx & \mathbf{Am} = \mathbf{I}\ \mathbf{p} \end{array} \tag{16}$$

Preconditioning means iteratively fitting by adjusting the $\mathbf{p}$ variables and then finding the model by using $\mathbf{m} = \mathbf{A}^{-1}\mathbf{p}$.

OPPORTUNITIES FOR SMART DIRECTIONS

Recall the fitting goals (10) with weights $\mathbf{W}$ being absorbed into the operator $\mathbf{F}$ and the data $\mathbf{d}$.

$$\begin{array}{rcccccccc} \mathbf{0} & \approx & \mathbf{r}_d & = & \mathbf{F}\mathbf{m} & - & \mathbf{d} & = & \mathbf{F}\mathbf{A}^{-1} \quad \mathbf{p} \quad - \quad \mathbf{d} \\ \mathbf{0} & \approx & \mathbf{r}_m & = & \mathbf{A}\mathbf{m} & & & = & \mathbf{I} \quad \mathbf{p} \end{array} \tag{17}$$

Without preconditioning, we have the search direction:

$$\Delta \mathbf{m}_{\text{bad}} \quad = \quad \begin{bmatrix} \mathbf{F}^{\mathrm{T}} & \mathbf{A}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_m \end{bmatrix} \tag{18}$$

and with preconditioning, we have the search direction:

$$\Delta \mathbf{p}_{\text{good}} \quad = \quad \begin{bmatrix} (\mathbf{F}\mathbf{A}^{-1})^{\mathrm{T}} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_m \end{bmatrix} \tag{19}$$

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 $\mathbf{p}^{\mathrm{T}}$ not $\mathbf{m}^{\mathrm{T}}$. Using $\mathbf{A}\mathbf{m} = \mathbf{p}$, we have $\mathbf{A}\Delta\mathbf{m} = \Delta\mathbf{p}$, which enables us to define a good search direction in $\mathbf{m}$ space.

$$\Delta \mathbf{m}_{\text{good}} \quad = \quad \mathbf{A}^{-1}\Delta \mathbf{p}_{\text{good}} \quad = \quad \mathbf{A}^{-1}(\mathbf{A}^{-1})^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}\mathbf{r}_d + \mathbf{A}^{-1}\mathbf{r}_m \tag{20}$$

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

$$\Delta \mathbf{m}_{\text{good}} \quad = \quad \mathbf{A}^{-1}(\mathbf{A}^{-1})^{\mathrm{T}}\ \mathbf{g} + \mathbf{m} \tag{21}$$

The search direction (21) shows a positive-definite operator scaling the gradient. Each component of any gradient vector is independent of each other. All independently point (negatively) to 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-direction algorithm to descend, as always, to the "exact" answer in a finite number of steps. The reason is that modifying the search direction with $\mathbf{A}^{-1}(\mathbf{A}^{-1})^{\mathrm{T}}$ is equivalent to solving a conjugate-gradient problem in $\mathbf{p}$. We'll see in Chapter **??**, that our specifying $\mathbf{A}^{-1}(\mathbf{A}^{-1})^{\mathrm{T}}$ amounts to us specifying a prior expectation of the spectrum of the model $\mathbf{m}$.

The meaning of the preconditioning variable p

To accelerate convergence of iterative methods, we often change variables. The model-styling regression $\mathbf{0} \approx \epsilon\mathbf{A}\mathbf{m}$ is changed to $\mathbf{0} \approx \epsilon\mathbf{p}$. Experience shows, however, that the variable $\mathbf{p}$ is often more interesting to look at than the model $\mathbf{m}$. Why should a new variable introduced for computational convenience turn out to have more interpretive value? There is a little theory explaining why. Begin from

$$\mathbf{0} \quad \approx \quad \mathbf{W}(\mathbf{F}\mathbf{m} - \mathbf{d}) \tag{22}$$

$$\mathbf{0} \quad \approx \quad \epsilon\, \mathbf{A}\mathbf{m} \tag{23}$$

Introduce the preconditioning variable $\mathbf{p}$.

$$\begin{aligned} \mathbf{0} &\approx \mathbf{W}(\mathbf{F}\mathbf{A}^{-1}\mathbf{p} - \mathbf{d}) && (24) \\ \mathbf{0} &\approx \epsilon\,\mathbf{p} && (25) \end{aligned}$$

Rewriting as a single regression:

$$\mathbf{0} \approx \begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_m \end{bmatrix} = \begin{bmatrix} \mathbf{W}\mathbf{F}\mathbf{A}^{-1} \\ \epsilon\,\mathbf{I} \end{bmatrix} \mathbf{p} - \begin{bmatrix} \mathbf{W}\mathbf{d} \\ \mathbf{0} \end{bmatrix} \tag{26}$$

The gradient vanishes at the best solution. To get the gradient, we put the residual into the adjoint operator. Thus, we put the residuals (column vector) in (26) into the transpose of the operator in (26), the row $((\mathbf{W}\mathbf{F}\mathbf{A}^{-1})^{\mathrm{T}}, \epsilon\mathbf{I})$. Finally, replace the $\approx$ by $=$. Thus,

$$\begin{aligned} \mathbf{0} &= (\mathbf{W}\mathbf{F}\mathbf{A}^{-1})^{\mathrm{T}}\,\mathbf{r}_d + \epsilon\,\mathbf{r}_m \\ \mathbf{0} &= (\mathbf{W}\mathbf{F}\mathbf{A}^{-1})^{\mathrm{T}}\,\mathbf{r}_d + \epsilon^2\,\mathbf{p} && (27) \end{aligned}$$

The two terms in Equation (27) are identical but oppositely signed. These terms represent images in model space. This image represents the fight between the data space residual and the model space residual. You really do want to plot this image. It shows the battle of the model wanted by the data against our preconceived statistical model expressed by our model styling goal. That is why the preconditioned variable $\mathbf{p}$ is interesting to inspect and interpret. It is not simply a computational convenience. It is telling you what you have learned from data (that someone has recorded at great expense!).

> The preconditioning variable $\mathbf{p}$ is not simply a computational convenience. This model-space image $\mathbf{p}$ tells us where our data contradicts our prior model. Admire it! Make a movie of it evolving with iteration.

If I were young and energetic like you, I would write a new basic tool for optimization. Instead of scanning only the space of the gradient and previous step, it would scan also over the "smart" direction. Using both directions should offer the benefit of preconditioning the regularization at early iterations while offering more assured fitting data at late iterations. The improved module for `cgstep` would need to solve a 3×3 matrix. I would also be looking for ways to assure all $\Delta\mathbf{m}$ directions were scaled to have the prior model spectrum and prior energy function of space.

Need for an invertible preconditioner

It is important to use regularization to solve many examples. It is important to precondition, because in practice, computer power is often a limiting factor. It is important to be able to begin from a nonzero starting solution, because in nonlinear problems we must restart from an earlier solution. Putting all three requirements together leads to a little problem. It turns out the three together lead us to needing a preconditioning transformation that is invertible. Let us see why this is so.

$$\begin{aligned} \mathbf{0} &\approx \mathbf{F}\mathbf{m} - \mathbf{d} \\ \mathbf{0} &\approx \mathbf{A}\mathbf{m} \end{aligned} \tag{28}$$

First, we change variables from $\mathbf{m}$ to $\mathbf{u} = \mathbf{m} - \mathbf{m}_0$. Clearly, $\mathbf{u}$ starts from $\mathbf{u}_0 = 0$, and $\mathbf{m} = \mathbf{u} + \mathbf{m}_0$. Then, our regression pair becomes:

$$\begin{aligned} \mathbf{0} &\approx \mathbf{F}\mathbf{u} + (\mathbf{F}\mathbf{m}_0 - \mathbf{d}) \\ \mathbf{0} &\approx \mathbf{A}\mathbf{u} + \mathbf{A}\mathbf{m}_0 \end{aligned} \tag{29}$$

This result differs from the original regression in only two minor ways, (1) revised data, and (2) a little more general form of the regularization, the extra term $\mathbf{A}\mathbf{m}_0$.

Now, let us introduce preconditioning. From the regularization, we see preconditioning introduces the preconditioning variable $\mathbf{p} = \mathbf{A}\mathbf{u}$. Our regression pair becomes:

$$\begin{aligned} \mathbf{0} &\approx \mathbf{F}\mathbf{A}^{-1}\mathbf{p} + (\mathbf{F}\mathbf{m}_0 - \mathbf{d}) \\ \mathbf{0} &\approx \mathbf{p} + \mathbf{A}\mathbf{m}_0 \end{aligned} \tag{30}$$

Here is the problem: We now require both $\mathbf{A}$ and $\mathbf{A}^{-1}$ operators. In 2- and 3-dimensional spaces, we do not know very many operators with an easy inverse. That reason is why I found myself pushed to come up with the helix methodology of Chapter **??**—because it provides invertible operators for smoothing and roughening.

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 in many books, for example my book *BEI* (Basic Earth Imaging). They amount to measuring the integral of the velocity squared from the surface down to the reflector, known as the root-mean-square (RMS) velocity. Although good quality echoes 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. Needed definitions are as follows:

$\mathbf{d}$ is a data vector in which components range over the vertical traveltime depth τ. Its 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.

$\mathbf{W}$ is a diagonal matrix along which we lay the given measure of data quality. We use it as a weighting function.

$\mathbf{C}$ is the matrix of causal integration, a lower triangular matrix of ones.

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

$\mathbf{u}$ is a vector containing the interval velocity squared v_{interval}^2 ranging over the vertical traveltime depth τ.

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:

$$\mathbf{Cu} \quad = \quad \mathbf{d} \tag{31}$$

In other words, any component of $\mathbf{d}_i$ measures the integral of a material property from the Earth surface to the depth of i. We wish to find the material property everywhere, which is $\mathbf{u}$. If we integrate it from the surface downward with causal integration $\mathbf{C}$, we should get the measurements $\mathbf{d}$.

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

$$\mathbf{0} \quad \approx \quad \mathbf{W}\,[\mathbf{Cu} - \mathbf{d}] \tag{32}$$

where $\mathbf{W}$ is some weighting function, we need to choose. 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 $\mathbf{p}$ of the squared interval velocity $\mathbf{u}$.

$$\mathbf{0} \quad \approx \quad \mathbf{Du} \quad = \quad \mathbf{p} \tag{33}$$

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

$$\mathbf{0} \quad \approx \quad \mathbf{W}\left[\mathbf{C}^2\mathbf{p} - \mathbf{d}\right] \tag{34}$$

$$\mathbf{0} \quad \approx \quad \epsilon\,\mathbf{p} \tag{35}$$

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 we interpreted to be too stiff at early times and too flexible at later times, which suggests we weaken ϵ at early times and strengthen it later. Because we wanted to keep ϵ constant with time, we strengthened $\mathbf{W}$ at early times and weakened it at later times as you see in the following program:

Lateral variations

The previous analysis appears 1-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.

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.

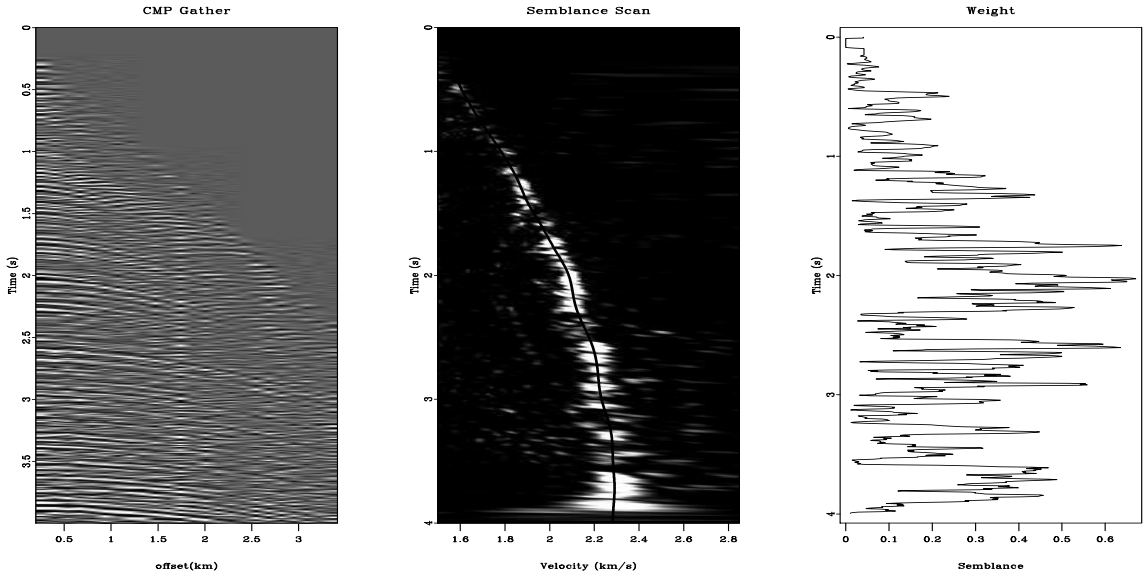

Figure 3: Raw CMP gather (left), semblance scan (middle), and semblance value used for weighting function (right).

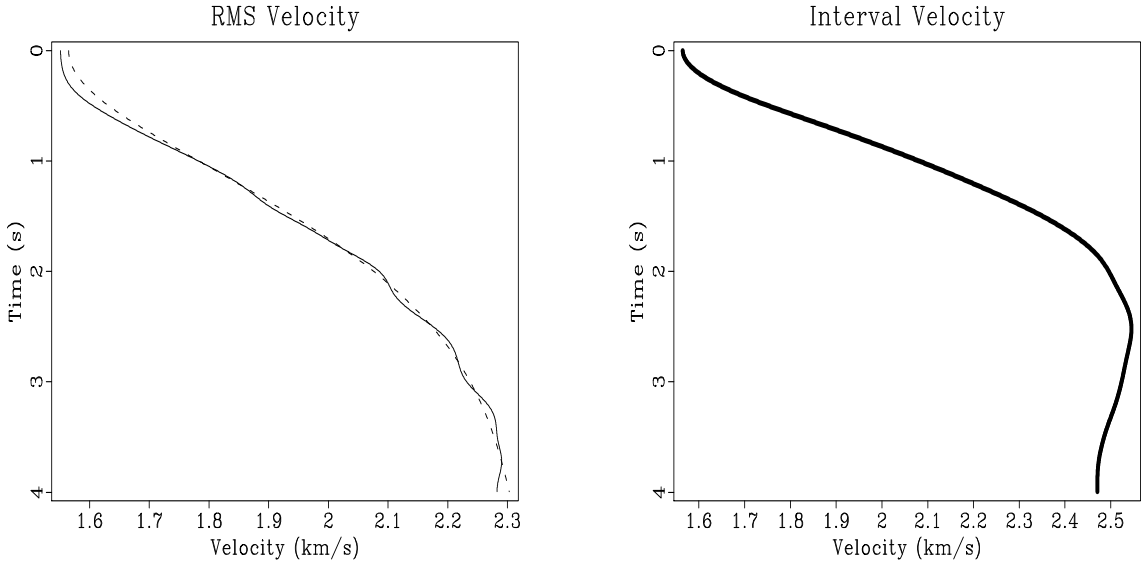

Figure 4: Observed RMS velocity and that predicted by a stiff model with $\epsilon = 4$. (Clapp)

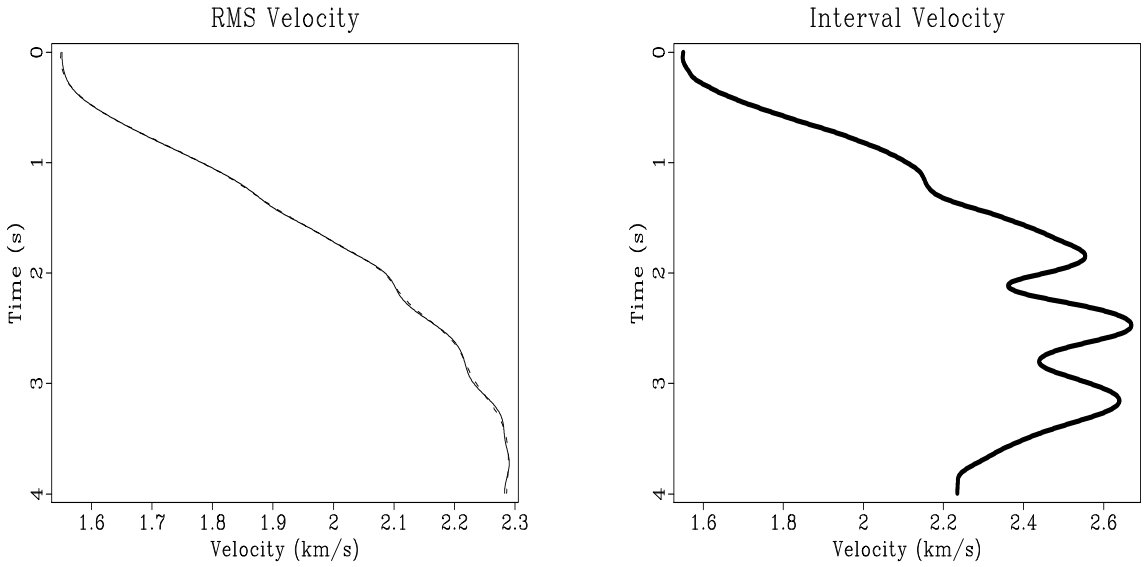

Figure 5: Observed RMS velocity and that predicted by a flexible model with $\epsilon = .25$ (Clapp)

(Where seismic data is poor, a well log could tell us whether or not 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 the five-layer model in Figure 6. Each layer has 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 for all five layers. Because 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 $d = e$.

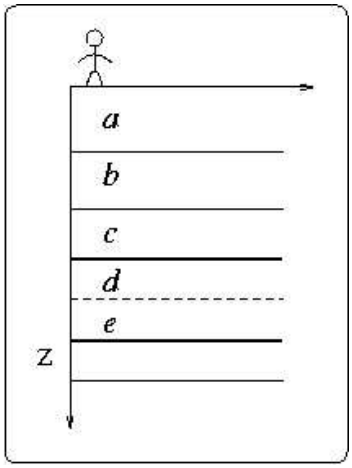

Figure 6: A layered Earth model. The layer interfaces cause reflections. Each layer has a constant velocity in its interior.

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:

$$
\begin{aligned}
3V_c^2 &\approx a+b+c && (36)\\
5V_e^2 &\approx a+b+c+d+e && (37)\\
u_0 &\approx a && (38)\\
0 &\approx -a+b && (39)\\
0 &\approx -b+c && (40)\\
0 &\approx -c+d && (41)\\
0 &\approx -d+e && (42)
\end{aligned}
$$

For the blocky solution, we do not want the fitting goal (41). Further explanations await completion of examples.

INVERSE LINEAR INTERPOLATION

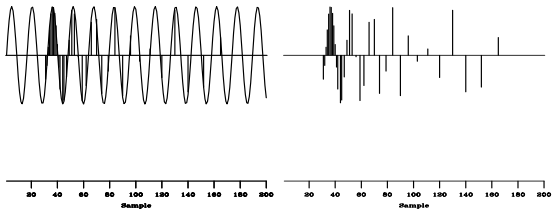

Figure 7: The input data are irregularly sampled.

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 7). 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, preconditioning provides a much faster rate of convergence. Because 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 (15) conflict with each other: the first one enforces "details" in the model, while the second one tries to smooth away the details. Typically, regularized optimization creates a complicated model at early iterations. At first, the data-fitting goal (15) plays a more important role. Later, the styling goal (15) 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 an insufficiently simplified one. Figure 8 provides a clear illustration of Harlan's observation.

Figure 9 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. Because the cost of each iteration for each method is roughly equal, the efficiency of preconditioning is evident.

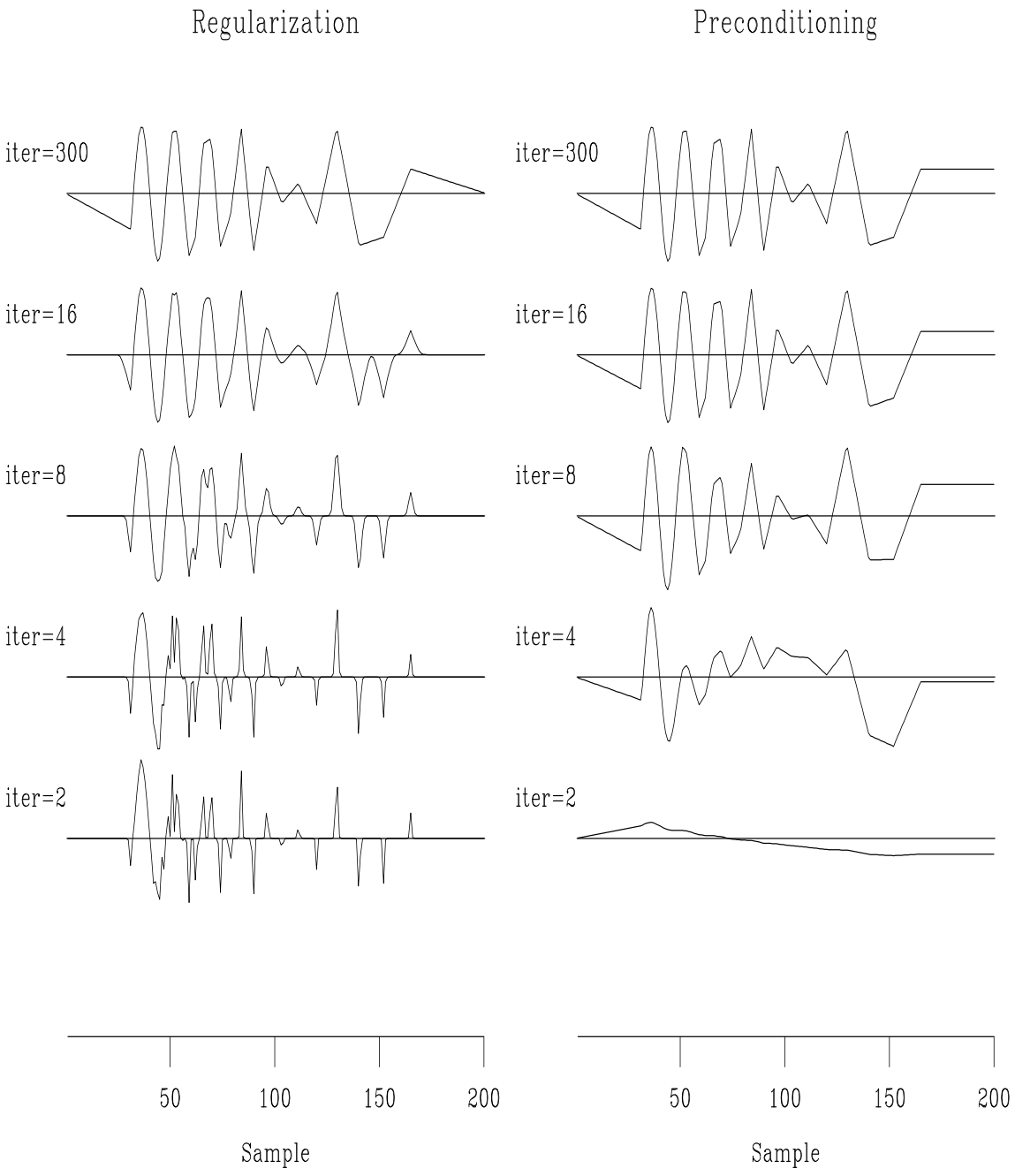

Figure 8: Convergence history of inverse linear interpolation. Left: regularization, right: preconditioning. The regularization operator $\mathbf{A}$ is the derivative operator (convolution with $(1, -1)$. The preconditioning operator $\mathbf{S}$ is causal integration.

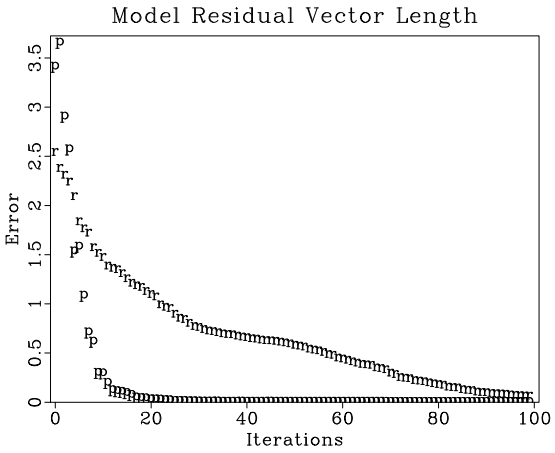

Figure 9: Convergence of the iterative optimization, measured in terms of the model residual. The "p" points stand for preconditioning; the "r" points, regularization.

EMPTY BINS AND PRECONDITIONING

There are at least three ways to fill empty bins. Two require a roughening operator $\mathbf{A}$ while the third requires a smoothing operator, which (for comparison purposes) we denote $\mathbf{A}^{-1}$. The three methods are generally equivalent though they differ in significant details.

The original way in Chapter **??** is to restore missing data by ensuring 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, $\mathbf{0} \approx \mathbf{A}(\mathbf{I}-\mathbf{K}+\mathbf{K})\mathbf{m}$ or:

$$\mathbf{0} \quad \approx \quad \mathbf{A}(\mathbf{I}-\mathbf{K})\mathbf{m} \;+\; \mathbf{A}\mathbf{m}_k \;, \tag{43}$$

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

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

$$\begin{array}{lcl} \mathbf{0} & \approx & \mathbf{K}\mathbf{m} \;-\; \mathbf{m}_k \\ \mathbf{0} & \approx & \epsilon\mathbf{A}\mathbf{m} \end{array} \tag{44}$$

and take the limit as the scalar $\epsilon \to 0$. At that limit, we should have the same result as equation (43).

There is an important philosophical difference between the first method and the second. The first method strictly honors the known data. The second method acknowledges that when data misfits the regularization theory, it might be the fault of the data, so the data need not be strictly honored. Just what balance is proper falls to the numerical choice of ϵ, a nontrivial topic.

A third way to find missing data is to precondition equation (44), namely, try the substitution $\mathbf{m} = \mathbf{A}^{-1}\mathbf{p}$.

$$\begin{array}{lcl} \mathbf{0} & \approx & \mathbf{K}\mathbf{A}^{-1}\mathbf{p} \;-\; \mathbf{m}_k \\ \mathbf{0} & \approx & \epsilon\mathbf{p} \end{array} \tag{45}$$

There is no simple way of knowing beforehand what is the best value of ϵ. Practitioners like to see solutions for various values of ϵ. Of course, that can cost a lot of computational effort. Practical exploratory data analysis is more pragmatic. Without a simple clear theoretical

basis, analysts generally begin from $\mathbf{p} = 0$ and abandon the fitting goal $\epsilon\mathbf{I}\mathbf{p} \approx 0$. Implicitly, they take $\epsilon = 0$. Then, they examine the solution as a function of iteration, imagining that the solution at larger iterations corresponds to smaller ϵ. There is an eigenvector analysis indicating some kind of basis for this approach, but I believe there is no firm guidance.

SeaBeam

Figure 10 shows an image of deep seawater bottom in the Pacific of a sea-floor spreading center produced acoustically by a device called SeaBeam. Students here tried all three methods of filling empty bins on the this data using the Laplacian as a regularizer. From an interpretive point of view, differences among the three methods were minor and as expected, therefore, only one is shown in Figure 10.

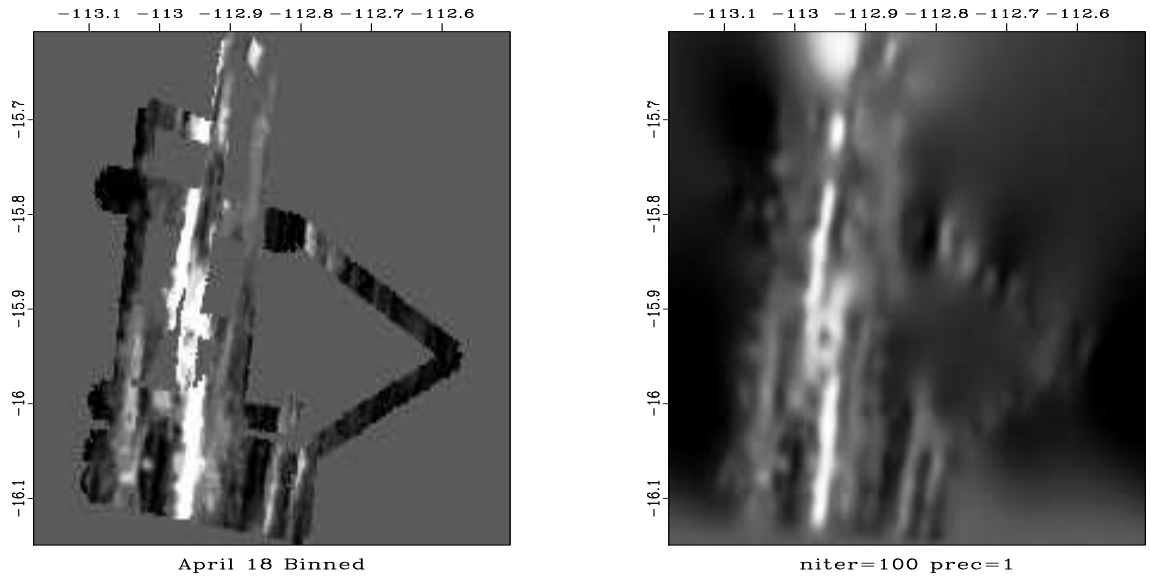

Figure 10: Seabeam data before and after empty bin filling with a laplacian.

GIANT PROBLEMS

This book does not solve giant problems, but it does solve personal-computer-sized problems in the manner of giant problems. There is big money in solving giant problems. Big money brings specialist solutions beyond the scope of this book. But let us take a look. Closest to me is the seismic survey industry. Model space is 3-D, a cube, roughly a thousand 2-D screen fulls, a screen full being roughly 1,000×1,000, a gigabyte in total. Data space is 5-dimensional. A seismogram is a thousand time points. Our energy source lies in two dimensions on the Earth surface plane, as do our receivers. 1+2+2=5. All this compounds roughly to 1,000 to the 5^{th} power, a thousand terabytes, a petabyte. Fully convergent solutions needing 10^{15} iterations of operators is ridiculous, while more than a handful are nearly so. We think mainly of using only the adjoint. Theory and experimentation offer some guidance. Remember that adjoints are great when unitary (already an inverse). Adjoints are improved if they can be made more unitary. The basic strategy for improving an

adjoint is finding one good diagonal-weighting function before the adjoint and another after it. Recalling "IID," adjoints are also made more unitary by filter matrices that have the effect of whitening output. Simple filter matrices are the gradient and the Laplacian. More generally, a compact way to whiten spectra is multidimensional autoregression, expounded in Chapter **??**.

A hundred iterations

Lurking in every giant problem are many problems of smaller size. In the large-scale seismic imaging problem lie problems of velocity estimation, multiple-reflection elimination, and many more.

Envision a large problem feasible in a hundred iterations. Many of my colleagues work on such problems. Maybe half would also use exotic parallel computer architectures. Those with ample energy and intellectual capacity to tackle such machines are rewarded by speed-up factors of 10 to a 100, rewarded also by a diverse population of industries hiring. This skill stays in demand because new architectures rapidly obsolete earlier generations. The other half, people like me, have the luxury of software (like in this book) decaying at a slower pace, leaving us needed time to tune our imaginations to extracting the structure of more complex problems.

It is a giant leap of faith that we can accomplish something of value with a mere hundred iterations in a task that theoretically demands quadrillions. Experience shows that we often do, and we do so by experimenting with "intuitive" methods. The first, I shall call "faking the epsilon."

Faking the epsilon

Burdened by a problem of oppressive size, any trick, honest or sly, is nice to know. I will tell you a trick that is widely used. Many studies are done ignoring (abandoning) the model styling regression (second fitting regression as follows):

$$\begin{aligned} \mathbf{0} &\approx \mathbf{F}\mathbf{A}^{-1}\mathbf{p} - \mathbf{d} \\ \mathbf{0} &\approx \epsilon\, \mathbf{p} \end{aligned} \tag{46}$$

Because we have a numerically poor idea of what epsilon should be, it is nice to be rid of it. The pragmatic person iterates the data-fitting regression only, watches the solution as a function of iteration, and stops when tired or (more hopefully) stops at the iteration that is subjectively most pleasing. The epsilon-faking trick does not really speed things. But, it eliminates the need for scan over epsilon. It also simplifies the coding (insert smiley emoticon).

Why does this crude approximation seem to work? The preconditioner is based on an analytic solution ($\mathbf{A}^{-1}$ is an inverse) to the regularization, so naturally, early iterations tend to already fit the regularization, so early iterations are struggling instead to fit the data. The longer you run though, the better the data fit, and the more the actual regularization should be coming into play. But ongoing research often fails to run that far.

Figure 8 shows the idea that early iterations fit the straight lines. Straight lines honor the preconditioner. At later iterations the data fits better. Why do straight-line solutions honor the regularization? Refer to the discussion near Figure **??**.

When preconditioning becomes a liability

Theoretically, preconditioning does not reduce the number of iterations required for an exact solution, but it gets us closer quicker; so, we may hope to omit all the work of the later iterations. Surprisingly and unfortunately, several of my colleagues have observed later iterations in which preconditioning actually slows convergence. Then, we are better off reverting to the nonpreconditioned initial form. Sorry, but I am unable to offer guidance or any method to cope with this issue other than your own application-dependent experimentation.

Earthquake depth illustrates a null space

In the dawn of the era of computerized earthquake seismology, someone decided to add earthquake depth to their catalog. Traditionally, they had solved for only three unknowns, latitude, longitude, and time of source at the source, i.e., origin time. Now, they would add a fourth, the depth. They wrote down the 4×4 system of equations and solved it. Erratic results. So then, they froze the depth at zero, solved for the old three variables; only then introducing the depth. Problem solved. (Compared to seismograph station separation, zero depth is an excellent approximation.)

I first understood the earthquake experience as an issue with nonlinear problems. True that earthquake travel time is not a linear function of distance, so the nonlinearity could lead to difficulty. But, something more is going on. When all seismometers are far from the earthquake, the waves arrive propagating nearly vertically (Earth curvature and $v(z)$ ray bending). Source depth affects such data in much the same way as time origin shift, so they are near a null space. Whenever near a null space, especially with a nonlinear problem, a good starting solution is needed.

The starting solution matters!

In principle, regularization solves the null-space problem, but that is only for those people lucky enough to have applications so small they can afford to iterate to completion. Think of this trivial 2-D null-space situation: A parabolic penalty on one spatial axis with no penalty on the other axis. Imagine a house facing northeast with a parabolic rain gutter mounted perfectly horizontally on one edge of the house roof. The null space is anywhere on the center line along the bottom of the gutter. Anywhere you begin, steepest descent brings you immediately to the gutter bottom in a location that depends on where you began. Now tilt the gutter a little bit so the water drips off one end of the rain drain. Steepest descent now overshoots a little so, as we saw in Chapter **??**, a tortuous path of right-angle turning ensues. (Recall Figure **??**.) The conjugate direction method quickly solves this trivial 2-D problem, but in a 150,000 dimensional lake bottom problem, conjugate directions taken only a few dozen iterations do not do as well. When the data-modeling operator contains a null

space, only the regularization can pull us away from it, and a small number of iterations may be unable to do the job. So, we need a good starting location.

> Textbook theory may tell us final solutions are independent of the starting location, but we learn otherwise from nonlinear problems, and we learn otherwise from linear but large problems.

Null space versus starting solution

The simplest null-space problem has one data point d emerging from two model points.

$$d \quad \approx \quad \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{47}$$

The null space is any solution that produces no data. You can add an arbitrary amount β of the null space getting another solution as good as the first. Here is the full solution.

$$\begin{bmatrix} x \\ y \end{bmatrix} \quad \approx \quad \frac{d}{2ab} \begin{bmatrix} b \\ a \end{bmatrix} \; + \; \beta \begin{bmatrix} -b \\ a \end{bmatrix} \tag{48}$$

Iterative methods can neither subtract nor add any null space to your initial solution. It is obvious in this simple case, because the gradient (here the matrix adjoint) dotted into the null-space vector vanishes. Suppose a and b are matrices, while d, x, and y are vectors. Although more complicated, something similar happens. You can test if an application involves a null space by comparing the results of various starting solutions.

Other traps arise in the world of images. Rarely are we able to iterate to full completion, so we might say, "practically speaking, this application has null spaces." For example, if we know that zero frequency is theoretically a null space, we would say, "The null space contains low frequencies." We cannot avoid such issues.

The textbook way of dealing with null spaces is to require the researcher to set up model styling goals (regularizations). Finding such goals demands assumptions from the researcher, assumptions that are often hard to specify. Luckily, there is another path to consider. Thinking more like a physicist, we could choose the initial solution more carefully.

In regression (47) extended to images, we might hope not to have a null-space problem when we begin iterating from $(\mathbf{x}, \mathbf{y}) = (\mathbf{0}, \mathbf{0})$, but this is not true. It is a pitfall, which in an application context, took me some years to recognize. Notice what happens the first step you move away from $(\mathbf{x}, \mathbf{y}) = (\mathbf{0}, \mathbf{0})$. Your solution becomes a constant β times the gradient. The image extension of (47) being:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \quad = \quad \beta \begin{bmatrix} \mathbf{A}^{\mathrm{T}}\mathbf{d} \\ \mathbf{B}^{\mathrm{T}}\mathbf{d} \end{bmatrix} \tag{49}$$

If the operators $\mathbf{A}$ and $\mathbf{B}$ resemble filters, it is pretty clear that $\mathbf{x}$ and $\mathbf{y}$ are correlated, which physically could be nonsense. We might be trying to discover if and how $\mathbf{x}$ and $\mathbf{y}$ are correlated. Or we might wish to demand they be uncorrelated.

I have no general method for you, but offer a suggestion that works for one family of applications and may be suggestive for others. Traditionally, it might happen that $\mathbf{y}$ is ignored, effectively taking $\mathbf{y} = \mathbf{0}$ which happens when the data is better explained by $\mathbf{A}$ alone than by $\mathbf{B}$ alone. Solve first for $\mathbf{x}$ without $\mathbf{y}$. Call it $\mathbf{x}_0$. Now define a new variable $\tilde{\mathbf{x}}$ such that $\mathbf{x} = \mathbf{x}_0 + \tilde{\mathbf{x}}$. Introducing your innovative concept (estimating $\mathbf{y}$) your regression becomes:

$$\mathbf{0} \approx \mathbf{r} = \mathbf{A}(\mathbf{x}_0 + \tilde{\mathbf{x}}) + \mathbf{B}\mathbf{y} - \mathbf{d} \tag{50}$$

$$\mathbf{0} \approx \mathbf{r} = \mathbf{A}\tilde{\mathbf{x}} + \mathbf{B}\mathbf{y} - (\mathbf{d} - \mathbf{A}\mathbf{x}_0) \tag{51}$$

Start off from $(\tilde{\mathbf{x}}, \mathbf{y}) = (\mathbf{0}, \mathbf{0})$. Like Equation (49), the first step leads to:

$$\begin{bmatrix} \tilde{\mathbf{x}} \\ \mathbf{y} \end{bmatrix} = \beta \begin{bmatrix} \mathbf{A}^{\mathrm{T}}\mathbf{r} \\ \mathbf{B}^{\mathrm{T}}\mathbf{r} \end{bmatrix} \tag{52}$$

which is very different from Equation (49), because $\mathbf{r}$ is very different from $\mathbf{d}$. Although we may still have an annoying or inappropriate correlation between $\tilde{\mathbf{x}}$ and $\mathbf{y}$, it is a lot less annoying than a correlation between $\mathbf{x}$ and $\mathbf{y}$.