Regularization is model styling

Jon Claerbout

Regularization is a method used in mathematics and statistics to deal with insufficient information. The reader must supply additional information in the form of an operator. From where is this operator to come, and what does it mean? It amounts to us, as practitioners, specifying a "style" of model. Where the model is a signal or an image, it amounts to specifying one weighting function in physical space and another in Fourier space.

EMPTY BINS AND INVERSE INTERPOLATION

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

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

Let u denote an unknown (missing) value. The dataset on which the examples are based is $(\cdots, u, u,\ 1, u, 2, 1, 2, u, u, \cdots)$. Theoretically, we could adjust the missing u values (each different) to minimize the energy in the unfiltered data. Those adjusted values would obviously turn out to be all zeros. The unfiltered data is data that has been filtered by an impulse function. To find the missing values that minimize energy out of other filters, we can use subroutine `mis1()`. Figure 1 shows interpolation of the dataset with $(1, -1)$ as a roughening filter. The interpolated data matches the given data where they overlap.

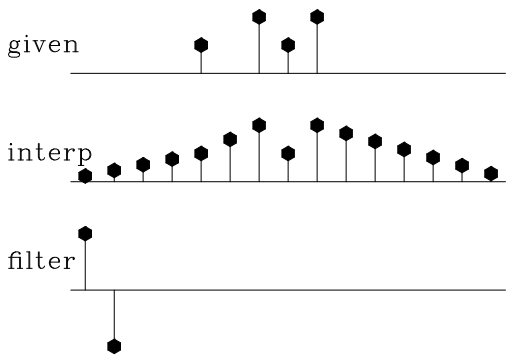

Figure 1: Top is given data. Middle is given data with interpolated values. Missing values seem to be interpolated by straight lines. Bottom shows the filter $(1, -1)$. Its output (not shown) has minimum energy.

Figures 1–4 illustrate the rougher the filter, the smoother the interpolated data, and vice versa. Switch attention from the residual spectrum to the residual. The residual for Figure 1 is the *slope* of the signal (because the filter $[1, -1]$ is a *first derivative*), and the slope is constant (uniformly distributed) along the straight lines where the least-squares

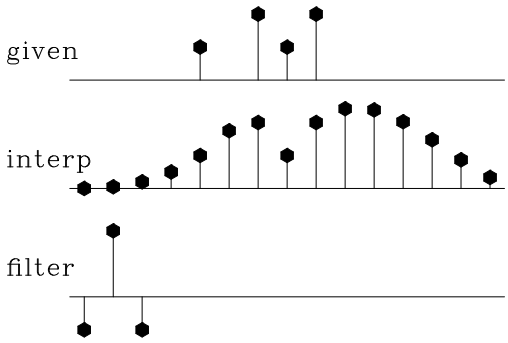

Figure 2: Top is the same input data as in Figure 1. Middle is interpolated. Bottom shows the filter $(-1, 2, -1)$. The missing data seems to be interpolated by parabolas.

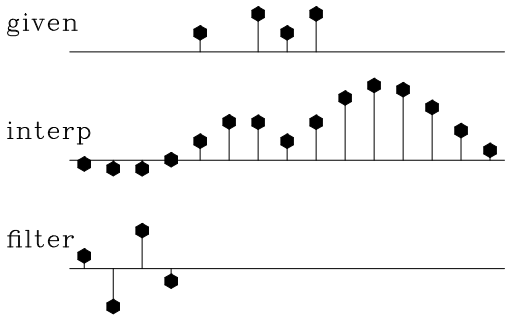

Figure 3: Top is the same input. Middle is interpolated. Bottom shows the filter $(1, -3, 3, -1)$. The missing data is very smooth. It shoots upward high off the right end of the observations, apparently to match the data slope there.

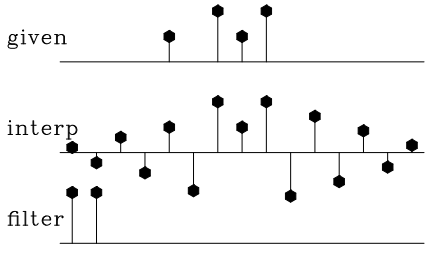

Figure 4: Bottom shows the filter $(1, 1)$. The interpolation is rough. Like the given data itself, the interpolation has much energy at the Nyquist frequency. But unlike the given data, it has little zero-frequency energy.

procedure is choosing signal values. So, these examples confirm the idea that the **least-squares method** abhors large values (because they are squared). Thus, least squares tends to distribute residuals uniformly in both time and frequency to the extent allowed by the **constraint**s.

This idea helps us answer the question, what is the best filter to use? It suggests choosing the filter to have an amplitude spectrum that is inverse to the spectrum we want for the interpolated data. A systematic approach is given in Chapter **??**, but I offer a simple subjective analysis here: Looking at the data, we see that all points are positive. It seems, therefore, that the data is rich in low frequencies; thus, the filter should contain something like $(1, -1)$, which vanishes at zero frequency. Likewise, the data seems to contain Nyquist frequency; so, the filter should contain $(1, 1)$. The result of using the filter $(1, -1) * (1, 1) = (1, 0, -1)$ is shown in Figure 5. Foregoing is my best subjective interpolation based on the idea that the missing data should look like the given data. The resulting **interpolation** and **extrapolation**s are so good that you can hardly guess which data values are given and which are interpolated. We care about this because the goal in geophysical image making is to create an image that hides locations of our measurements (and missing measurements!).

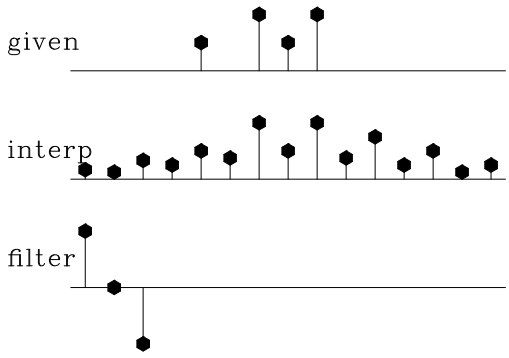

Figure 5: Top is the same as in Figures 1 to 4. Middle is interpolated. Bottom shows the filter $(1, 0, -1)$, which comes from the coefficients of $(1, -1) * (1, 1)$. Both the given data and the interpolated data have significant energy at both zero and Nyquist frequencies.

Missing-data program

Now, let us see how Figures 1-5 were calculated. Matrices could have been pulled apart according to subscripts of known and missing data. Instead I computed them with operators, and applied only operators and their adjoints. First, we inspect the matrix approach because it is more conventional.

Matrix approach to missing data

Customarily, we have referred to data by the symbol $\mathbf{d}$. Now that we are dividing the data space into two parts, known and unknown (or missing), we refer to this complete space as the model (or map) space $\mathbf{m}$.

There are 15 data points in Figures 1-5. Of the 15, 4 are known and 11 are missing. Denote the known by k and the missing by u. Then, the sequence of missing and known is $(u, u, u, u, k, u, k, k, k, u, u, u, u, u, u)$. Because I cannot print 15×15 matrices, please allow

me to describe instead a data space of 6 values $(m_1, m_2, m_3, m_4, m_5, m_6)$ with known values only m_2 and m_3, $\mathbf{m}$ arranged as (u, k, k, u, u, u).

Our approach is to minimize the energy in the residual, which is the filtered map (model) space. We state the fitting goals $\mathbf{0} \approx \mathbf{Fm}$ as:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \approx \mathbf{r} = \begin{bmatrix} a_1 & 0 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & 0 & 0 & 0 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 & 0 \\ 0 & a_3 & a_2 & a_1 & 0 & 0 \\ 0 & 0 & a_3 & a_2 & a_1 & 0 \\ 0 & 0 & 0 & a_3 & a_2 & a_1 \\ 0 & 0 & 0 & 0 & a_3 & a_2 \\ 0 & 0 & 0 & 0 & 0 & a_3 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \end{bmatrix} \tag{1}$$

Rearranging these fitting goals, and bringing the columns multiplying known data values (m_2 and m_3) to the left, we get $\mathbf{y} = -\mathbf{F}_k \mathbf{m}_k \approx \mathbf{F}_u \mathbf{m}_u$.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} = - \begin{bmatrix} 0 & 0 \\ a_1 & 0 \\ a_2 & a_1 \\ a_3 & a_2 \\ 0 & a_3 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} m_2 \\ m_3 \end{bmatrix} \approx \begin{bmatrix} a_1 & 0 & 0 & 0 \\ a_2 & 0 & 0 & 0 \\ a_3 & 0 & 0 & 0 \\ 0 & a_1 & 0 & 0 \\ 0 & a_2 & a_1 & 0 \\ 0 & a_3 & a_2 & a_1 \\ 0 & 0 & a_3 & a_2 \\ 0 & 0 & 0 & a_3 \end{bmatrix} \begin{bmatrix} m_1 \\ m_4 \\ m_5 \\ m_6 \end{bmatrix} \tag{2}$$

Equation (2) is the familiar form of an overdetermined system of equations $\mathbf{y} \approx \mathbf{F}_u \mathbf{m}_u$ that we could solve for $\mathbf{m}_u$ as illustrated earlier by conjugate directions or by a wide variety of well-known methods.

The trouble with this matrix approach is that it is awkward to program the partitioning of the operator into the known and missing parts, particularly if the application of the operator uses arcane techniques, such as those used by the fast–Fourier-transform operator or various numerical approximations to differential, or partial differential, operators that depend on regular data sampling. Even for the modest convolution operator, we already have a library of convolution programs that handle a variety of end effects, and it would be much nicer to use the library as it is rather than recode it for all possible geometrical arrangements of missing data values.

Operator approach to missing data

For the operator approach to the fitting goal $-\mathbf{F}_k \mathbf{m}_k \approx \mathbf{F}_u \mathbf{m}_u$, we rewrite it as $-\mathbf{F}_k \mathbf{m}_k \approx \mathbf{FJm}$ where

$$-\mathbf{F}_k\mathbf{m}_k \;\approx\; \begin{bmatrix} a_1 & 0 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & 0 & 0 & 0 & 0 \\ a_3 & a_2 & a_1 & 0 & 0 & 0 \\ 0 & a_3 & a_2 & a_1 & 0 & 0 \\ 0 & 0 & a_3 & a_2 & a_1 & 0 \\ 0 & 0 & 0 & a_3 & a_2 & a_1 \\ 0 & 0 & 0 & 0 & a_3 & a_2 \\ 0 & 0 & 0 & 0 & 0 & a_3 \end{bmatrix} \begin{bmatrix} 1 & . & . & . & . & . \\ . & 0 & . & . & . & . \\ . & . & 0 & . & . & . \\ . & . & . & 1 & . & . \\ . & . & . & . & 1 & . \\ . & . & . & . & . & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \end{bmatrix} \;=\; \mathbf{FJm} \quad (3)$$

Notice the introduction of the new diagonal matrix $\mathbf{J}$, called a **mask**ing matrix or a **constraint-mask** matrix because it multiplies constrained variables by zero leaving freely adjustable variables untouched. Experience shows that a better name than "mask matrix" is "**selector** matrix" because what comes out of it, that which is selected, is a less-confusing name for it than which is rejected. With a selector matrix, the whole data space seems freely adjustable, both the missing data values and known values. We see that the CD method does not change the known (constrained) values. In general, we derive the fitting goal (3) by:

$$\mathbf{0} \;\approx\; \mathbf{Fm} \quad (4)$$
$$\mathbf{0} \;\approx\; \mathbf{F}(\mathbf{J}+(\mathbf{I}-\mathbf{J}))\mathbf{m} \quad (5)$$
$$\mathbf{0} \;\approx\; \mathbf{FJm}+\mathbf{F}(\mathbf{I}-\mathbf{J})\mathbf{m} \quad (6)$$
$$\mathbf{0} \;\approx\; \mathbf{FJm}+\mathbf{Fm}_{\text{known}} \quad (7)$$
$$\mathbf{0} \;\approx\; \mathbf{r} \;=\; \mathbf{FJm}+\mathbf{r}_0 \quad (8)$$

As usual, we find a direction to go $\Delta\mathbf{m}$ by the gradient of the residual energy.

$$\Delta\mathbf{m} = \frac{\partial}{\partial \mathbf{m}^{\mathrm{T}}}\, \mathbf{r}^{\mathrm{T}}\mathbf{r} = \left(\frac{\partial}{\partial \mathbf{m}^{\mathrm{T}}}\, \mathbf{r}^{\mathrm{T}}\right)\mathbf{r} = \left(\frac{\partial}{\partial \mathbf{m}^{\mathrm{T}}}\, (\mathbf{m}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}+\mathbf{r}_0^{\mathrm{T}})\right)\mathbf{r} = \mathbf{J}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}\mathbf{r} \quad (9)$$

We begin the calculation with the known data values where missing data values are replaced by zeros, namely $(\mathbf{I}-\mathbf{J})\mathbf{m}$. Filter this data, getting $\mathbf{F}(\mathbf{I}-\mathbf{J})\mathbf{m}$, and load it into the residual $\mathbf{r}_0$. With this initialization completed, we begin an iteration loop. First we compute $\Delta\mathbf{m}$ from equation (9).

$$\Delta\mathbf{m} \quad \longleftarrow \quad \mathbf{J}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}\mathbf{r} \quad (10)$$

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

$$\Delta\mathbf{r} \quad \longleftarrow \quad \mathbf{FJ}\Delta\mathbf{m} \quad (11)$$

Equation (11) applies the filtering again. Then, use the method of steepest descent (or conjugate direction) to choose the appropriate scaling (or inclusion of previous step) of $\Delta\mathbf{m}$ and $\Delta\mathbf{r}$, and update $\mathbf{m}$ and $\mathbf{r}$, accordingly, and iterate.

The subroutine to find missing data is `mis1()`. It assumes that zero values in the input data correspond to missing data locations. It uses our convolution operator `tcai1()`. You can also check the book Index for other operators and modules.

user/gee/mis1.c

```
54 void mis1(int niter          /* number of iterations */,
55           float *xx         /* data/model */,
56           const bool *known /* mask for known data */,
57           const char *step  /* solver */)
58 /*< interpolate >*/
59 {
60     switch (step[1]) {
61         case 'g': /* conjugate gradients */
62             sf_solver (tcai1_lop, sf_cgstep, nx, ny, xx, zero,
63                        niter, "x0", xx, "known", known, "end");
64             sf_cgstep_close();
65             break;
66         case 'd': /* conjugate directions */
67             sf_cdstep_init();
68             sf_solver (tcai1_lop, sf_cdstep, nx, ny, xx, zero,
69                        niter, "x0", xx, "known", known, "end");
70             sf_cdstep_close();
71             break;
72         default:
73             sf_error("%s: unknown step %s",__FILE__,step);
74             break;
75     }
76 }
```

I sought reference material on conjugate gradients with constraints and did not find anything, leaving me to fear that this chapter was in error, and I had lost the magic property of convergence in a finite number of iterations. I tested the code, and it did converge in a finite number of iterations. The explanation is that these constraints are almost trivial. We pretended we had extra variables, and computed a $\Delta\mathbf{m} = \mathbf{g}$ for each. Using $\mathbf{J}$ we then set the gradient $\Delta\mathbf{m} = \mathbf{g}$ to zero, therefore making no changes to anything, like as if we had never calculated the extra $\Delta\mathbf{m}$s.

WELLS NOT MATCHING THE SEISMIC MAP

Accurate knowledge comes from **wells**, but wells are expensive and far apart. Less accurate knowledge comes from surface seismology, but this knowledge is available densely in space and can indicate significant **trend**s between the wells. For example, a prospective area may contain 15 wells but 600 or more seismic stations. To choose future well locations, it is helpful to match the known well data with the seismic data. Although the seismic data is delightfully dense in space, it often mismatches the wells because there are systematic differences in the nature of the measurements. These discrepancies are sometimes attributed to velocity **anisotropy**. To work with such measurements, we do not need to track down the physical model, we need only to merge the information somehow to appropriately **map** the trends between wells and make a proposal for the next drill site. Here, we consider only a scalar value at each location. Take $\mathbf{w}$ to be a vector of 15 components, each component being the seismic travel time to some fixed depth in a well. Likewise, let $\mathbf{s}$ be a 600-component vector each with the seismic travel time to that fixed depth as estimated wholly from surface seismology. Such empirical corrections are often called "**fudge factor**s". An example is the Chevron oil field in Figure 6. The binning of the seismic data in Figure 6 is

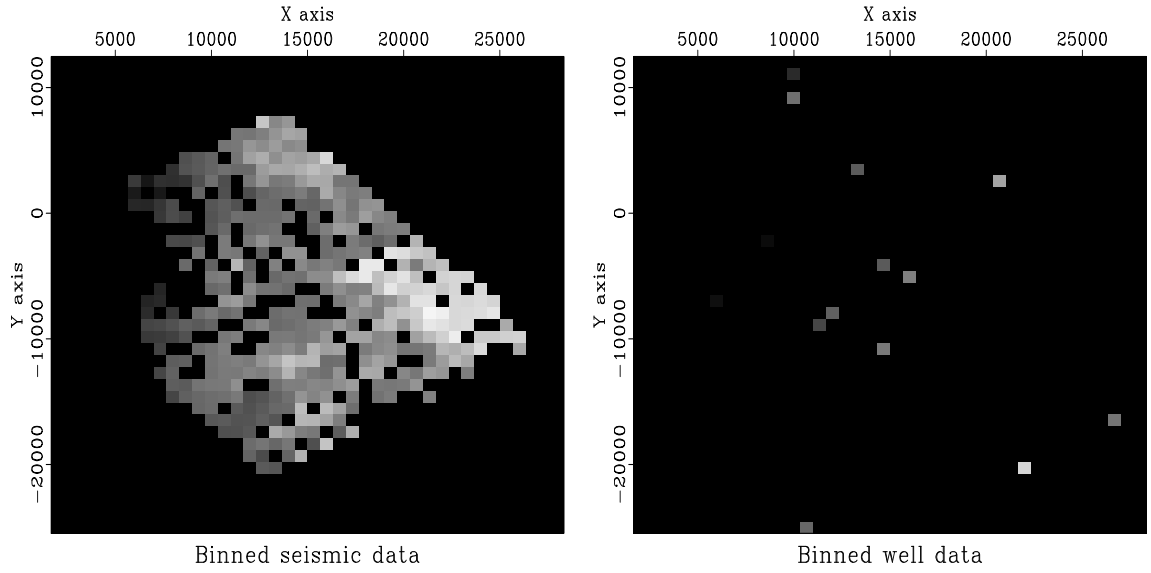

Figure 6: Binning by data push. Left is seismic data. Right is well locations. Values in bins are divided by numbers in bins. (Toldi)

not really satisfactory when we have available the techniques of missing data estimation to fill the empty bins. Using the ideas of subroutine `mis1()` , we can extend the seismic data

into the empty part of the plane. We use the same principle that we minimize the energy in the filtered map where the map must match the data where it is known. I chose the filter $\mathbf{A} = \nabla^{\mathrm{T}} \nabla = -\nabla^2$ to be the Laplacian operator (actually, its negative) to obtain the result in Figure 7.

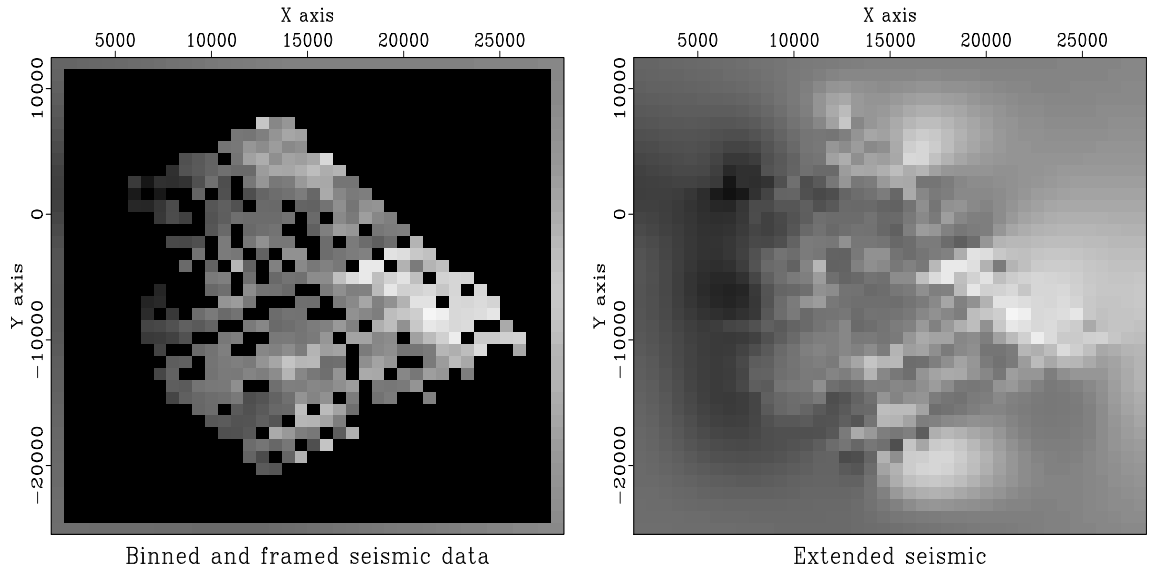

Figure 7: Seismic binned (left) and extended (right) by minimizing energy in $\nabla^2 \mathbf{s}$.

There are basically two ways to handle boundary conditions. First, as we did in Figure 1, by using a transient filter operator that assumes zero outside to the region of interest. Second is to use an internal filter operator. Internal filters introduce the hazard that solutions could be growing at the boundaries. Growing solutions are rarely desirable. In that case, it is better to assign boundary values, which is what I did here in Figure 7. I did not do it because it is better, but did it to minimize the area surrounding the data of interest.

The first job is to fill the gaps in the seismic data. We did such a job in one dimension in Figures 1–5. More computational details later come later. Let us call the extended seismic data $\mathbf{s}$.

Think of a map of a model space $\mathbf{m}$ of infinitely many hypothetical wells that must match the real wells, where we have real wells. We must find a map that matches the wells exactly and somehow matches the seismic information elsewhere. Let us define the vector $\mathbf{w}$, as shown in Figure 6 so $\mathbf{w}$ is observed values at wells and zeros elsewhere.

Where the seismic data contains sharp bumps or streaks, we want our final Earth model to have those features. The wells cannot provide the rough features, because the wells are too far apart to provide high-spatial frequencies. The well information generally conflicts with the seismic data at low-spatial frequencies because of systematic discrepancies between the two types of measurements. Thus we must accept that $\mathbf{m}$ and $\mathbf{s}$ may differ at low-spatial frequencies (where gradient and Laplacian are small).

Our final map $\mathbf{m}$ would be very unconvincing if it simply jumped from a well value at one point to a seismic value at a neighboring point. The map would contain discontinuities around each well. Our philosophy of finding an Earth model $\mathbf{m}$ is that our Earth map

should contain no obvious "footprint" of the data acquisition (well locations). We adopt the philosophy that the difference between the final map (extended wells), and the seismic information $\mathbf{x} = \mathbf{m} - \mathbf{s}$ should be smooth. Thus, we seek the minimum residual $\mathbf{r}$, which is the roughened difference between the seismic data $\mathbf{s}$ and the map $\mathbf{m}$ of hypothetical omnipresent wells. With roughening operator $\mathbf{A}$ we fit:

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \mathbf{A}(\mathbf{m}-\mathbf{s}) \quad = \quad \mathbf{A}\mathbf{x} \tag{12}$$

along with the constraint that the map should match the wells at the wells. We honor this constraint by initializing the map $\mathbf{m} = \mathbf{w}$ to the wells (where we have wells, and zero elsewhere). After we find the gradient direction to suggest some changes to $\mathbf{m}$, we simply do not allow those changes at well locations by using a mask. We apply a "missing data selector" to the gradient which zeros out possible changes at well locations. Like with the goal (7), we have:

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \mathbf{A}\mathbf{J}\mathbf{x} + \mathbf{A}\mathbf{x}_{\text{known}} \tag{13}$$

After minimizing $\mathbf{r}$ by adjusting $\mathbf{x}$, we have our solution $\mathbf{m} = \mathbf{x} + \mathbf{s}$.

Now, we prepare some roughening operators $\mathbf{A}$. We have already coded a 2-D gradient operator `igrad2`. Let us combine it with its adjoint to get the 2-D Laplacian operator. (You might notice that the Laplacian operator is "self-adjoint," meaning that the operator does the same calculation that its adjoint does. Any operator of the form $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is self-adjoint because $(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$.)

Subroutine `lapfill()` is the same idea as `mis1()` except that the filter $\mathbf{A}$ has been specialized to the Laplacian implemented by module `laplac2`.

Subroutine `lapfill()` can be used for each of our two applications: (1) extending the seismic data to fill space, and (2) fitting the map exactly to the wells and approximately to the seismic data. When extending the seismic data, the initially non-zero components $\mathbf{s} \neq \mathbf{0}$ are fixed and cannot be changed.

The final map is shown in Figure 8.

Results can be computed with various filters. I tried both ∇^2 and ∇. There are disadvantages of each, ∇ being too cautious and ∇^2 perhaps being too aggressive. Figure 9 shows the difference $\mathbf{x}$ between the extended seismic data and the extended wells. Notice that for ∇ the difference shows a localized "tent pole" disturbance about each well. For ∇^2, there could be a large overshoot between wells, especially if two nearby wells have significantly different values. I do not see that problem here.

My overall opinion is that the Laplacian does the better job in this case. I have that opinion because in viewing the extended gradient, I can clearly see where the wells are. The wells are where we have acquired data. We would like our map of the world to not show where we acquired data. Perhaps our estimated map of the world cannot help but show where we have and have not acquired data, but we would like to minimize that aspect.

A good image of the Earth hides our data **acquisition footprint**.

To understand the behavior theoretically, recall that in one dimension the filter ∇ interpolates with straight lines and ∇^2 interpolates with cubics. The reason is that the fitting goal $\mathbf{0} \approx \nabla\mathbf{m}$, leads to $\frac{\partial}{\partial \mathbf{m}^{\mathrm{T}}}\mathbf{m}^{\mathrm{T}}\nabla^{\mathrm{T}}\nabla\mathbf{m} = \mathbf{0}$ or $\nabla^{\mathrm{T}}\nabla\mathbf{m} = \mathbf{0}$; whereas, the fitting goal

system/generic/laplac2.c

```
46      for     (i2=0; i2 < n2; i2++) {
47          for (i1=0; i1 < n1; i1++) {
48              j = i1+i2*n1;
49
50              if (i1 > 0) {
51                  if (adj) {
52                      p[j-1] -= r[j];
53                      p[j]   += r[j];
54                  } else {
55                      r[j] += p[j] - p[j-1];
56                  }
57              }
58              if (i1 < n1-1) {
59                  if (adj) {
60                      p[j+1] -= r[j];
61                      p[j]   += r[j];
62                  } else {
63                      r[j] += p[j] - p[j+1];
64                  }
65              }
66
67              if (i2 > 0) {
68                  if (adj) {
69                      p[j-n1] -= r[j];
70                      p[j]    += r[j];
71                  } else {
72                      r[j] += p[j] - p[j-n1];
73                  }
74              }
75              if (i2 < n2-1) {
76                  if (adj) {
77                      p[j+n1] -= r[j];
78                      p[j]    += r[j];
79                  } else {
80                      r[j] += p[j] - p[j+n1];
81                  }
82              }
83          }
84      }
```

system/generic/lapfill.c

```
void lapfill(int niter    /* number of iterations */,
             float* mm    /* model [m1*m2] */,
             bool *known /* mask for known data [m1*m2] */)
/*< interpolate >*/
{
    if (grad) {
        sf_solver (sf_igrad2_lop, sf_cgstep, n12, 2*n12, mm, zero,
                   niter, "x0", mm, "known", known,
                   "verb", verb, "end");
    } else {
        sf_solver (laplac2_lop, sf_cgstep, n12, n12, mm, zero,
                   niter, "x0", mm, "known", known,
                   "verb", verb, "end");
    }
    sf_cgstep_close ();
}
```

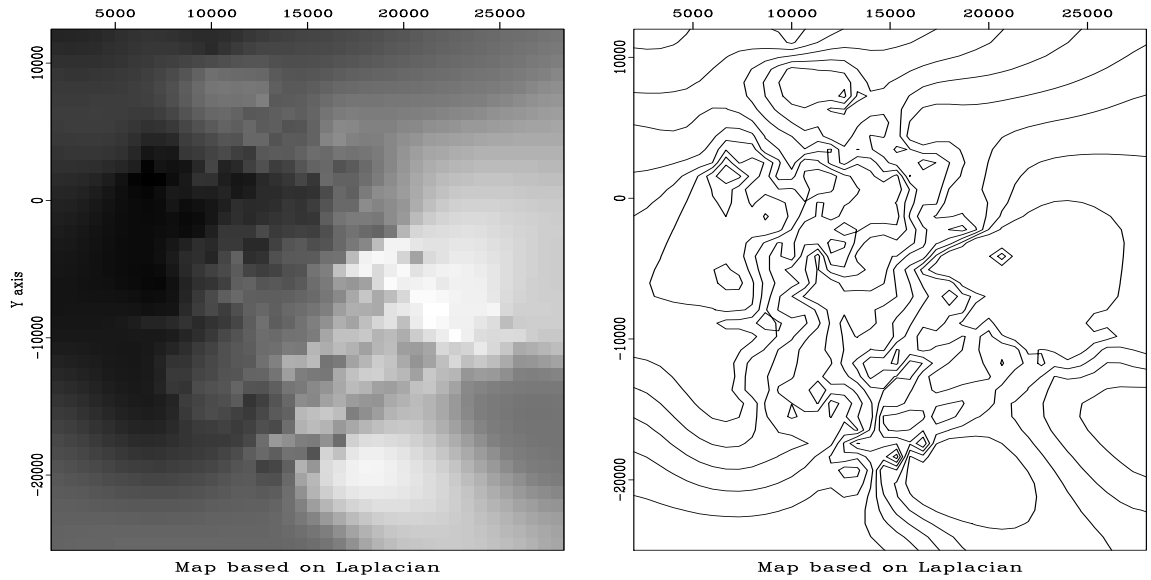

Figure 8: Final map based on Laplacian roughening.

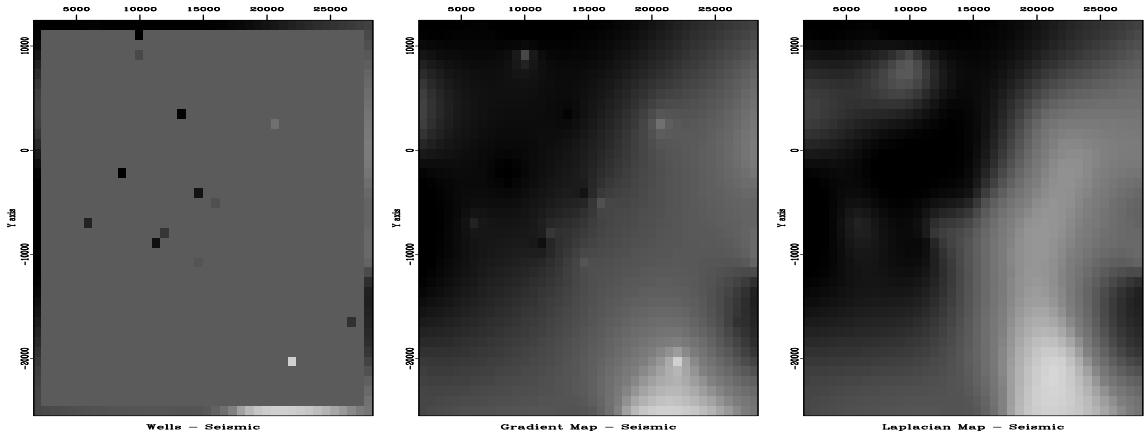

Figure 9: Difference between wells (the final map) and the extended seismic data. Left is plotted at the wells (with gray background for zero). Center is based on gradient roughening and shows tent-pole-like residuals at wells. Right is based on Laplacian roughening.

$\mathbf{0} \approx \nabla^2\mathbf{m}$ leads to $\nabla^4\mathbf{m} = \mathbf{0}$, which is satisfied by cubics. In two dimensions, minimizing the output of ∇ gives us solutions of Laplace's equation with sources at the known data. It is as if ∇ stretches a rubber sheet over poles at each well; whereas, ∇^2 bends a stiff plate.

Just because ∇^2 gives smoother maps than ∇ does not mean those maps are closer to reality. An objectively better choice for the model styling goal is addressed in Chapter **??**. It is the same issue we noticed when comparing Figures 1-5.

SEARCHING THE SEA OF GALILEE

Figure 10 shows a bottom-sounding survey of the Sea of Galilee[1] at various stages of processing. The ultimate goal is not only a good map of the depth to bottom, but images useful for the purpose of identifying **archaeological**, geological, or geophysical details of the sea bottom. The Sea of Galilee is unique, because it is a *fresh*-water lake *below* sea-level. It seems to be connected to the great rift (pull-apart) valley crossing east Africa. We might delineate the Jordan River delta. We might find springs on the water bottom. We might find archaeological objects.

The raw data is 132,044 triples, (x_i, y_i, z_i), where x_i ranges over 12 km and where y_i ranges over 20 km. The lines you see in Figure 10 are sequences of data points, i.e., the track of the survey vessel. The depths z_i are recorded at a discretization interval of 10 cm.

The first frame in Figure 10 shows simple binning. A coarser mesh would avoid the empty bins but lose resolution. As we refine the mesh for more detail, the number of empty bins grows, as does the care needed in devising a technique for filling them. This first frame uses the simple idea from Chapter **??** of spraying all the data values to the nearest bin with `bin2()` and dividing by the number in the bin. Bins with no data obviously need to be filled in some other way. I used a missing data program like that in the recent section on

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

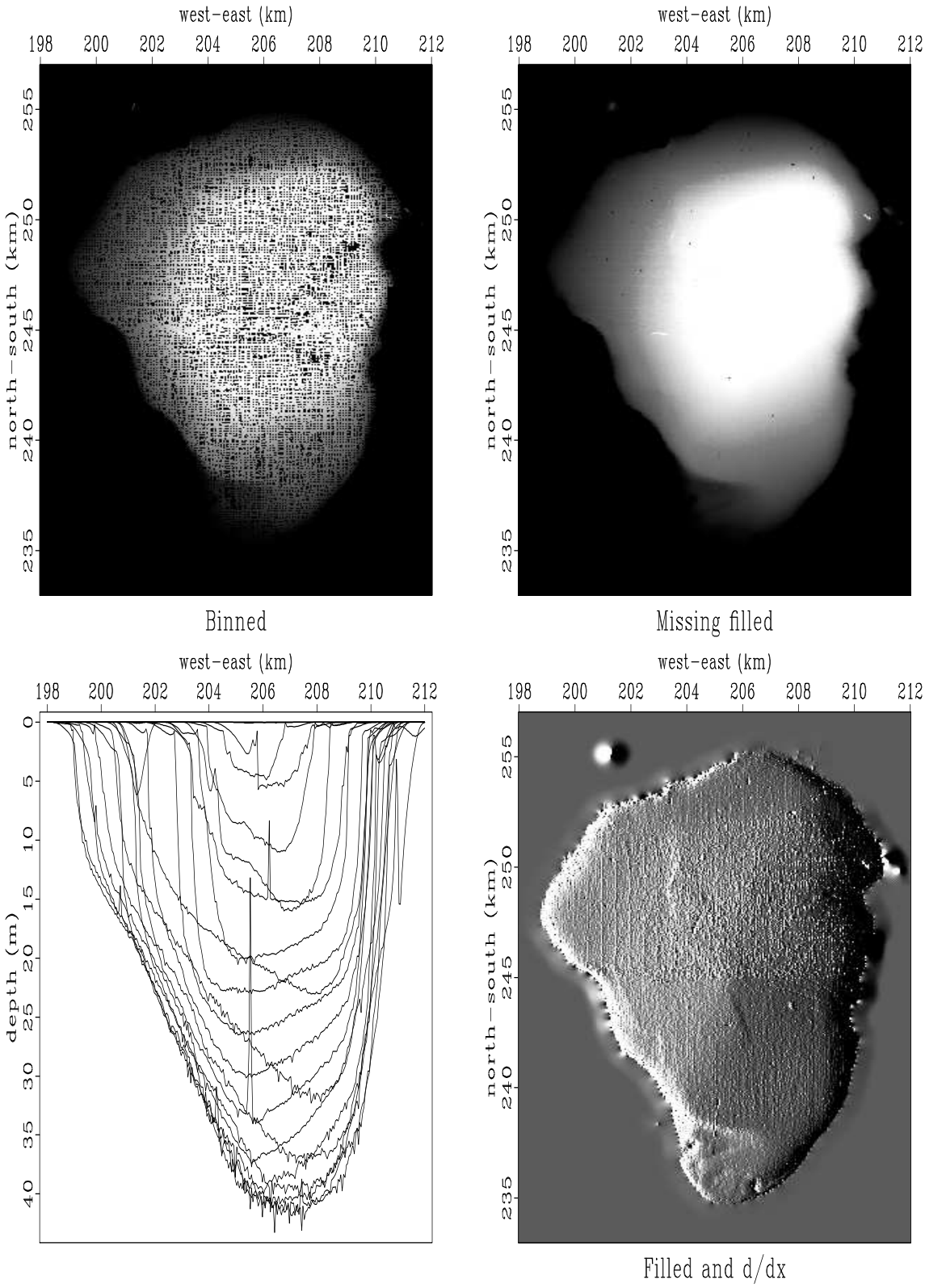

Figure 10: Views of the bottom of the Sea of Galilee.

"wells not matching the seismic map." Instead of roughening with a Laplacian, however, I used the gradient operator `igrad2`. The solver is `grad2fill()`.

api/c/igrad2.c

```
48      for (i2=0; i2 < n2-1; i2++) {
49          for (i1=0; i1 < n1-1; i1++) {
50              i = i1+i2*n1;
51              if (adj) {
52                  p[i+1]  += r[i];
53                  p[i+n1] += r[i+n12];
54                  p[i]    -= (r[i] + r[i+n12]);
55              } else {
56                  r[i]    += (p[i+1]  - p[i]);
57                  r[i+n12] += (p[i+n1] - p[i]);
58              }
59          }
60      }
```

api/c/grad2fill.c

```
55  void sf_grad2fill(int niter    /* number of iterations */,
56                    float* mm    /* estimated model */,
57                    bool *known /* mask */)
58  /*< Run optimization >*/
59  {
60      sf_solver (sf_igrad2_lop, sf_cgstep, n12, 2*n12, mm, zero,
61                 niter, "x0", mm, "known", known, "end");
62      sf_cgstep_close ();
63  }
```

The output of the roughening operator is an image, a filtered version of the depth, a filtered version of something real. Such filtering can enhance the appearance of interesting features. For example, in scanning the shoreline of the roughened image (after missing data was filled), we see several ancient shorelines, now submerged. The roughened map is often more informative than the map.

The views expose several defects of the data acquisition and of our data processing. The impulsive glitches (St. Peter's fish?) need to be removed; but we must be careful not to throw out the sunken ships along with the bad data points. Even our best image shows clear evidence of the recording vessel's tracks. Strangely, some tracks are deeper than others. Perhaps the survey is assembled from work done in different seasons, and the water level varied by season. Perhaps, some days the vessel was more heavily loaded and the depth sounder was on a deeper keel. As for the navigation equipment, we can see that some data values are reported outside the lake!

We want the sharpest possible view of this classical site. A treasure hunt is never easy

and no one guarantees we can find anything of great value; but at least the exercise is a good warm-up for submarine petroleum exploration.

CODE FOR THE REGULARIZED SOLVER

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

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

$$\begin{aligned} \mathbf{0} &\approx \mathbf{Fm} - \mathbf{d} \\ \mathbf{0} &\approx \mathbf{Am} \end{aligned} \tag{14}$$

Suppose we take the roughening filter to be the second difference operator $(1, -2, 1)$ scaled by a constant ϵ, and suppose we have a data point near each end of the model and a third data point exactly in the middle. Then, for a model space 6 points long, the fitting goal could look like:

$$\left[\begin{array}{cccccc} .8 & .2 & . & . & . & . \\ . & . & 1 & . & . & . \\ . & . & . & . & .5 & .5 \\ \hline \epsilon & . & . & . & . & . \\ -2\epsilon & \epsilon & . & . & . & . \\ \epsilon & -2\epsilon & \epsilon & . & . & . \\ . & \epsilon & -2\epsilon & \epsilon & . & . \\ . & . & \epsilon & -2\epsilon & \epsilon & . \\ . & . & . & \epsilon & -2\epsilon & \epsilon \\ . & . & . & . & \epsilon & -2\epsilon \\ . & . & . & . & . & \epsilon \end{array}\right] \left[\begin{array}{c} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{array}\right] - \left[\begin{array}{c} d_0 \\ d_1 \\ d_2 \\ \hline 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right] = \left[\begin{array}{c} \mathbf{r}_d \\ \mathbf{r}_m \end{array}\right] \approx \mathbf{0} \tag{15}$$

The residual vector has two parts, a data part $\mathbf{r}_d$ on top and a model part $\mathbf{r}_m$ on the bottom. The data residual should vanish except where contradictory data values happen to lie in the same place. The model residual is the roughened model.

Finding something unexpected is good science and engineering. We should look both in data space and model space. In data space, we look at the residual $\mathbf{r}$. In model space, we look at the residual projected there $\Delta\mathbf{m} = \mathbf{F}^{\mathrm{T}}\mathbf{r}$. After iterating to completion, we have $\Delta\mathbf{m} = \mathbf{0} = \mathbf{F}^{\mathrm{T}}\mathbf{r}_d + \mathbf{A}^{\mathrm{T}}\mathbf{r}_m$, a sum of two images identical but for polarity. This image tells

us what we have learned from the data; and how the model differs from what we thought it would be.

After all the definitions, we load the negative of the data into the residual. If a starting model $\mathbf{m}_0$ is present, then we update the data part of the residual $\mathbf{r}_d = \mathbf{F}\mathbf{m}_0 - \mathbf{d}$, and we load the model part of the residual $\mathbf{r}_m = \mathbf{A}\mathbf{m}_0$. Otherwise, we begin from a zero model $\mathbf{m}_0 = \mathbf{0}$; and thus, the model part of the residual $\mathbf{r}_m$ is also zero. After this initialization, subroutine `solver_reg()` begins an iteration loop by first computing the proposed model perturbation $\Delta\mathbf{m}$ (called `g` in the program) with the adjoint operator:

$$\Delta\mathbf{m} \quad \longleftarrow \quad \begin{bmatrix} \mathbf{F}^{\mathrm{T}} & \mathbf{A}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_m \end{bmatrix} \tag{16}$$

Using this value of $\Delta\mathbf{m}$, we can find the implied change in residual $\Delta\mathbf{r}$ as:

$$\Delta \begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_m \end{bmatrix} \quad \longleftarrow \quad \begin{bmatrix} \mathbf{F} \\ \mathbf{A} \end{bmatrix} \Delta\mathbf{m} \tag{17}$$

and the last thing in the loop is to use the optimization step function `stepper()` to decide the length of the step size and to decide how much of the previous step to include.

An example of using the new solver is subroutine `invint1`. I chose to implement the model roughening operator $\mathbf{A}$ with the convolution subroutine `tcai1()` which has transient end effects (and an output length equal to the input length plus the filter length). The adjoint of subroutine `tcai1()` suggests perturbations in the convolution input (not the filter).

user/gee/invint1.c

```
24 void invint1(int niter                    /* number of iterations */,
25              int nd                      /* data size */,
26              float *coord                /* data coordinates */,
27              const float *dd             /* data values */,
28              int n1, float o1, float d1  /* model grid */,
29              int na, const float *aa     /* filter */,
30              float *mm                   /* estimated model */,
31              float eps                   /* regularization */)
32 /*< inverse interpolation >*/
33 {
34     lint1_init( o1, d1, coord); /* interpolation */
35     tcai1_init( na, aa);        /* filtering */
36
37     sf_solver_reg(lint1_lop, sf_cgstep, tcai1_lop,
38                   n1+na, n1, nd, mm, dd, niter, eps);
39     sf_cgstep_close( );
40 }
```

Figure 11 shows an example for a $(1, -2, 1)$ filter with $\epsilon = 1$. The continuous curve representing the model $\mathbf{m}$ passes through the data points. Because the models are computed

with transient convolution end-effects, the models tend to damp linearly to zero outside the region where signal samples are given.

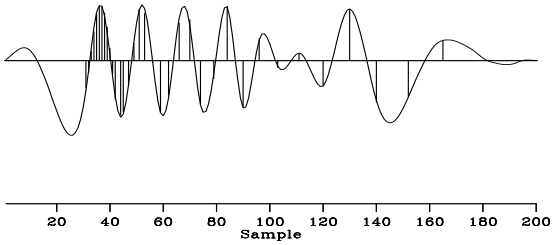

Figure 11: Sample points and estimation of a continuous function through them.

To show an example where the result is clearly a theoretical answer, I prepared another figure with the simpler filter $\mathbf{A} = (1, -1)$. When we minimize energy in the first derivative of the waveform, the residual becomes distributed uniformly between data points so the solution there is a straight line. Theoretically, it should be a straight line, because a straight line has a vanishing second derivative; and that condition arises by differentiating by $\mathbf{x}^{\mathrm{T}}$, the minimized quadratic form $\mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x}$, and getting $\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{0}$. (By this logic, the curves between data points in Figure 11 must be cubics.) The $(1, -1)$ result is shown in Figure 12.

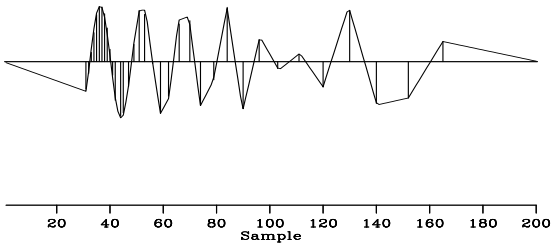

Figure 12: The same data samples and a function through them that minimizes the energy in the first derivative.

The example of Figure 12 has been a useful test case for me. You will see it again in later chapters. What I would like to show you here is a movie showing the convergence to Figure 12. Convergence occurs rapidly where data points are close together. The large gaps, however, fill at a rate of one point per iteration.

Abandoned theory for matching wells and seismograms

Let us consider theory to construct a map $\mathbf{m}$ that fits dense seismic data $\mathbf{s}$ and the well data $\mathbf{w}$. The first goal $\mathbf{0} \approx \mathbf{L}\mathbf{m} - \mathbf{w}$ says that when we linearly interpolate from the map, we should get the well data. The second goal $\mathbf{0} \approx \mathbf{A}(\mathbf{m} - \mathbf{s})$ (where $\mathbf{A}$ is a roughening operator like ∇ or ∇^2) says that the map $\mathbf{m}$ should match the seismic data $\mathbf{s}$ at high frequencies but need not do so at low frequencies.

$$\begin{aligned} \mathbf{0} &\approx \mathbf{L}\mathbf{m} - \mathbf{w} \\ \mathbf{0} &\approx \mathbf{A}(\mathbf{m} - \mathbf{s}) \end{aligned} \tag{18}$$

Although (18) is the way I originally formulated the well-fitting application, I abandoned it for several reasons: First, the map had ample pixel resolution compared to other sources of error, so I switched from linear interpolation to binning. Once I was using binning, I had

available the simpler empty-bin approaches. These approaches have the advantage that it is not necessary to experiment with the relative weighting between the two goals in (18). A formulation like (18) is more likely to be helpful where we need to handle rapidly changing functions where binning is inferior to linear interpolation, perhaps in reflection seismology where high resolution is meaningful.

PRECONCEPTION AND CROSS VALIDATION

First, we first look at data $\mathbf{d}$. Then we think about a model $\mathbf{m}$, and an operator $\mathbf{L}$ to link the model and the data. Sometimes, the operator is merely the first term in a series expansion about $(\mathbf{m}_0, \mathbf{d}_0)$. Then, we fit $\mathbf{d} - \mathbf{d}_0 \approx \mathbf{L}(\mathbf{m} - \mathbf{m}_0)$. To fit the model, we must reduce the fitting residuals. Realizing the importance of a data residual is not always simply the size of the residual, but is a function of it, we conjure up (topic for later chapters) a weighting function (which could be a filter) operator $\mathbf{W}$. With $\mathbf{W}$ we define our data residual:

$$\mathbf{r}_d \quad = \quad \mathbf{W}[\mathbf{L}(\mathbf{m} - \mathbf{m}_0) \;-\; (\mathbf{d} - \mathbf{d}_0)] \tag{19}$$

Next, we realize that the data might not be adequate to determine the model, perhaps because our comfortable dense sampling of the model ill fits our economical sparse sampling of data. Thus we adopt a fitting goal that mathematicians call "regularization," and we might call a "model styling" goal or more simply, a quantification of our preconception of the best model. We quantify our goals by choosing an operator $\mathbf{A}$, often simply a roughener like a gradient (the choice again a topic in this and later chapters). It defines our model residual by $\mathbf{Am}$ or $\mathbf{A}(\mathbf{m} - \mathbf{m}_0)$, say we choose:

$$\mathbf{r}_m \quad = \quad \mathbf{Am} \tag{20}$$

In an ideal world, our model preconception (**prejudice**?) would not conflict with measured data, but real life is much more interesting than that. The reason we pay for data acquisition is that conflicts between data and preconceived notions invariably arise. We need an adjustable parameter that measures our "**bullheadedness**," how much we intend to stick to our preconceived notions in spite of contradicting data. This parameter is generally called epsilon ϵ, because we like to imagine that our bullheadedness (prejudice?) is small. (In mathematics, ϵ is often taken to be an infinitesimally small quantity.) Although any bullheadedness seems like a bad thing, it must be admitted that measurements are imperfect too. Thus, as a practical matter, we often find ourselves minimizing:

$$\min \quad := \quad \mathbf{r}_d \cdot \mathbf{r}_d \;+\; \epsilon^2 \; \mathbf{r}_m \cdot \mathbf{r}_m \tag{21}$$

and wondering what to choose for ϵ. I have two suggestions: My simplest suggestion is to choose ϵ so that the residual of data fitting matches that of model styling. Thus:

$$\epsilon \quad = \quad \sqrt{\frac{\mathbf{r}_d \cdot \mathbf{r}_d}{\mathbf{r}_m \cdot \mathbf{r}_m}} \tag{22}$$

My second suggestion is to think of the force on our final solution. In physics, force is associated with a gradient. We have a gradient for the data fitting and another for the model styling:

$$\mathbf{g}_d \quad = \quad \mathbf{L}^{\mathrm{T}} \mathbf{W}^{\mathrm{T}} \mathbf{r}_d \tag{23}$$

$$\mathbf{g}_m \quad = \quad \mathbf{A}^{\mathrm{T}} \mathbf{r}_m \tag{24}$$

We could balance these forces by the choice:

$$\epsilon \quad = \quad \sqrt{\frac{\mathbf{g}_d \cdot \mathbf{g}_d}{\mathbf{g}_m \cdot \mathbf{g}_m}} \tag{25}$$

Although we often ignore ϵ in discussing the formulation of an application, when time comes to solve the problem, reality intercedes. Generally, $\mathbf{r}_d$ has different physical units than $\mathbf{r}_m$ (likewise $\mathbf{g}_d$ and $\mathbf{g}_m$), and we cannot allow our solution to depend on the accidental choice of units in which we express the problem. I have had much experience choosing ϵ, but it is only recently that I boiled it down to the suggestions of equations (22) and (25). Normally I also try other values, like double or half previous choices, and I examine the solutions for subjective appearance. The epsilon story continues in Chapter **??**.

Computationally, we could choose a new ϵ with each iteration, but it is more expeditious to freeze ϵ, solve the problem, recompute ϵ, and solve the problem again. I have never seen a case in which more than one repetition was necessary.

People who work with small applications (less than about 10^3 vector components) have access to an attractive theoretical approach called "cross-validation." Simply speaking, we could solve the problem many times, each time omitting a different data value. Each solution would provide a model that could be used to predict the omitted data value. The quality of these predictions is a function of ϵ which provides a guide to finding it. My objections to cross validation are two-fold: First, I do not know how to apply it in the large applications we solve in this book (I should think more about it); and second, people who worry much about ϵ, perhaps first should think more carefully about their choice of the filters $\mathbf{W}$ and $\mathbf{A}$, which is the focus of this book. Notice that both $\mathbf{W}$ and $\mathbf{A}$ can be defined with a scaling factor like ϵ. Often more important in practice, with $\mathbf{W}$ and $\mathbf{A}$ we have a scaling factor that need not be constant but can be a function of space or spatial frequency within the data space and/or model space.

EXERCISES:

1 Figures 1–4 seem to extrapolate to vanishing signals at the side boundaries. Why is that so, and what could be done to leave the sides unconstrained in that way?

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

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

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

5 It is desired to find a compromise between the Laplacian roughener and the gradient roughener. What is the size of the residual space?

6 Like the seismic prospecting industry, you have solved a huge problem using binning. You have computer power left over to do a few iterations with linear interpolation. How much does the cost per iteration increase? Should you refine your model mesh, or can you use the same model mesh that you used when binning?

7 Nuclear energy, having finally reached its potential, has dried up the prospecting industries so you find yourself doing **medical imaging** (or **earthquake seismology**). You probe the human body from all sides on a dense regular mesh in cylindrical coordinates. Unfortunately, you need to represent your data in Fourier space. There is no such thing as a fast Fourier transform in cylindrical coordinates, while slow Fourier transforms are pitifully slow. Your only hope to keep up with your competitors is to somehow do your FTs in cartesian coordinates. Write down the sequence of steps to achieve your goals using the methods of this chapter.