Stanford Exploration Project, Report 94, November 11, 1997, pages 141–165

On model-space and data-space regularization: A tutorial

Sergey Fomel[1]

keywords: *least squares, optimization, regularization, preconditioning*

ABSTRACT

Constraining ill-posed inverse problems often requires regularized optimization. I describe two alternative approaches to regularization. The first approach involves a column operator and an extension of the data space. The second approach constructs a row operator and expands the model space. In large-scale problems, when the optimization is incomplete, the two methods of regularization behave differently. I illustrate this fact with simple examples and discuss its implications for geophysical problems.

INTRODUCTION

Regularization is a method of imposing additional conditions for solving inverse problems with optimization methods. When model parameters are not fully constrained by the problem (the inverse problem is mathematically ill-posed), regularization limits the variability of the model and guides the iterative optimization to the desired solution by adding assumptions about the model power, smoothness, predictability, etc. In other words, it constrains the model null space to an *a priori* chosen pattern. A thorough mathematical theory of regularization has been introduced by works of Tikhonov's school (Tikhonov and Arsenin, 1977).

In this paper, I discuss two alternative formulations of regularized least-square inversion problems. The first formulation, which I call *model-space*, extends the data space and constructs a composite column operator. The second, *data-space*, formulation extends the model space and constructs a composite row operator. This second formulation is intrinsically related to the concept of model preconditioning. I illustrate the general theory with examples from *Three-Dimensional Filtering* (Claerbout, 1994).

Two excellent references cover almost all of the theoretical material in this note. One is the paper by Ryzhikov and Troyan (1991). The other one is a short note by

[1]**email:** sergey@sep.stanford.edu

Harlan, available by courtesy of the author on the World Wide Web (Harlan, 1995). I have attempted to translate some of the ideas in these two references to the linear operator language, familiar to the readers of Claerbout (1992, 1994).

MODEL-SPACE REGULARIZATION

Let us denote the linear forward modeling operator by $\boldsymbol{L}$. Then the basic matrix equation to be inverted is

$$\boldsymbol{L m} = \boldsymbol{d} \;, \tag{1}$$

where $\boldsymbol{m}$ stands for the model vector, and $\boldsymbol{d}$ represents the data vector.

Quite often the size of the data space is smaller than the desired size of the model space. This is typical for some interpolation problems (Claerbout, 1992, 1994), but may also be the case in tomographic problems. Even if the data size is larger than the model size, certain components of the model $\boldsymbol{m}$ may not be fully constrained by equation (1). In interpolation applications, this situation corresponds to empty bins in the model space. In tomography applications, it corresponds to shadow zones in the model, not illuminated by the tomographic rays.

Model-space regularization suggests adding equations to system (1) to obtain a fully constrained (well-posed) inverse problem. These additional equations are based on prior assumptions about the model and typically take the form

$$\boldsymbol{D m} \approx \boldsymbol{0} \;, \tag{2}$$

where $\boldsymbol{D}$ represents the imposed condition in the form of a linear operator. In many applications, $\boldsymbol{D}$ can be thought of as a filter, enhancing "bad" components in the model, or as a differential equation that we assume the model should satisfy.

The full system of equations (1)-(2) can be written in a short notation as

$$\boldsymbol{G_m m} = \left[\begin{array}{c} \boldsymbol{L} \\ \lambda \boldsymbol{D} \end{array} \right] \boldsymbol{m} = \left[\begin{array}{c} \boldsymbol{d} \\ \boldsymbol{0} \end{array} \right] = \hat{\boldsymbol{d}} \;, \tag{3}$$

where $\hat{\boldsymbol{d}}$ is the effective data vector:

$$\hat{\boldsymbol{d}} = \left[\begin{array}{c} \boldsymbol{d} \\ \boldsymbol{0} \end{array} \right] \;, \tag{4}$$

$\boldsymbol{G_m}$ is a *column* operator:

$$\boldsymbol{G_m} = \left[\begin{array}{c} \boldsymbol{L} \\ \lambda \boldsymbol{D} \end{array} \right] \;, \tag{5}$$

and λ is a scaling parameter. The subscript $\boldsymbol{m}$ stands for *model space* to help us distinguish $\boldsymbol{G_m}$ from the analogous data-space operator, introduced in the next section.

Now that the inverse problem (3) is fully constrained, we can solve it by means of unconstrained least-square optimization, minimizing the squared power $\hat{\boldsymbol{r}}^T\hat{\boldsymbol{r}}$ of the compound residual vector

$$\hat{\boldsymbol{r}} = \hat{\boldsymbol{d}} - \boldsymbol{G_m m} = \begin{bmatrix} \boldsymbol{d} - \boldsymbol{Lm} \\ -\lambda \boldsymbol{Dm} \end{bmatrix} . \tag{6}$$

The formal solution of the regularized optimization problem has the known form

$$<\boldsymbol{m}>= \left(\boldsymbol{G_m^T G_m}\right)^{-1} \boldsymbol{G_m^T} \hat{\boldsymbol{d}} = \left(\boldsymbol{L^T L} + \lambda^2 \boldsymbol{D^T D}\right)^{-1} \boldsymbol{L^T d} \; . \tag{7}$$

To recall the derivation of formula (7), consider the objective function

$$\hat{\boldsymbol{r}}^T\hat{\boldsymbol{r}} = \left(\hat{\boldsymbol{d}} - \boldsymbol{G_m m}\right)^T \left(\hat{\boldsymbol{d}} - \boldsymbol{G_m m}\right)$$

and take its partial derivative with respect to the model vector $\boldsymbol{m}$. Setting the derivative equal to zero leads to the *normal* equations

$$\boldsymbol{G_m^T G_m m} = \boldsymbol{G_m^T} \hat{\boldsymbol{d}} \; , \tag{8}$$

whose solution has the form of formula (7).

For the sake of simplicity, we will consider separately a "trivial" regularization, which seeks the smallest possible model from all the models, defined by equation (1). For this form of regularization, $\boldsymbol{D^T D}$ is an identity operator. If we denote the model-space identity operator by $\boldsymbol{I_m}$, the least-square estimate in this case takes the form

$$<\boldsymbol{m}>= \left(\boldsymbol{L^T L} + \lambda^2 \boldsymbol{I_m}\right)^{-1} \boldsymbol{L^T d} \; . \tag{9}$$

DATA-SPACE REGULARIZATION

In this section, I consider an alternative formulation of the regularized least-square optimization.

We start again with the basic equation (1) and introduce a residual vector $\boldsymbol{r}$, defining it by the relationship

$$\lambda \boldsymbol{r} = \boldsymbol{d} - \boldsymbol{Lm} \; , \tag{10}$$

where λ is a scaling parameter. Let us consider a compound model $\hat{\boldsymbol{m}}$, composed of the model vector $\boldsymbol{m}$ itself and the residual $\boldsymbol{r}$. With respect to the compound model, equation (10) can be rewritten as

$$\begin{bmatrix} \boldsymbol{L} & \lambda \boldsymbol{I_d} \end{bmatrix} \begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{r} \end{bmatrix} = \boldsymbol{G_d} \hat{\boldsymbol{m}} = \boldsymbol{d} \; , \tag{11}$$

where $\boldsymbol{G_d}$ is a *row operator*:

$$\boldsymbol{G_d} = \begin{bmatrix} \boldsymbol{L} & \lambda \boldsymbol{I_d} \end{bmatrix} \; , \tag{12}$$

and $\boldsymbol{I_d}$ represents the data-space identity operator.

System (11) is clearly under-determined with respect to the compound model $\hat{\boldsymbol{m}}$. If from all possible solutions of this system we seek the one with the minimal power $\hat{\boldsymbol{m}}^T\hat{\boldsymbol{m}}$, the formal (ideal) result takes the well-known form

$$<\hat{\boldsymbol{m}}>=\left[\begin{array}{c} <\boldsymbol{m}> \\ <\boldsymbol{r}> \end{array}\right] = \boldsymbol{G}_d^T\left(\boldsymbol{G}_d\boldsymbol{G}_d^T\right)^{-1}\boldsymbol{d} = \left[\begin{array}{c} \boldsymbol{L}^T\left(\boldsymbol{L}\boldsymbol{L}^T+\lambda^2\boldsymbol{I}_d\right)^{-1}\boldsymbol{d} \\ \lambda\left(\boldsymbol{L}\boldsymbol{L}^T+\lambda^2\boldsymbol{I}_d\right)^{-1}\boldsymbol{d} \end{array}\right] . \quad (13)$$

To recall the derivation of formula (13), decompose the effective model vector $\hat{\boldsymbol{m}}$ into two terms

$$\hat{\boldsymbol{m}} = \boldsymbol{G}_d^T\boldsymbol{d}_0 + \boldsymbol{m}_0 \ , \quad (14)$$

where $\boldsymbol{d}_0$ and $\boldsymbol{m}_0$ are to be determined. First, we choose $\boldsymbol{m}_0$ to be an orthogonal supplement to $\boldsymbol{G}_d^T\boldsymbol{d}_0$. The orthogonality implies that the objective function $\hat{\boldsymbol{m}}^T\hat{\boldsymbol{m}} = \boldsymbol{d}_0^T\boldsymbol{G}_d\boldsymbol{G}_d^T\boldsymbol{d}_0+\boldsymbol{m}_0^T\boldsymbol{m}_0$ is minimized only when $\boldsymbol{m}_0=\boldsymbol{0}$. To determine $\boldsymbol{d}_0$, substitute (14) into equation (11) and solve the corresponding linear system. The result takes the form of equation (13).

Let us show that estimate (13) is exactly equivalent to estimate (9) from the "trivial" model-space regularization. Consider the operator

$$\boldsymbol{G} = \boldsymbol{L}^T\boldsymbol{L}\boldsymbol{L}^T + \lambda^2\boldsymbol{L}^T \ , \quad (15)$$

which is a mapping from the data space to the model space. We can group the multiplicative factors in formula (25) in two different ways, as follows:

$$\boldsymbol{G} = \boldsymbol{L}^T(\boldsymbol{L}\boldsymbol{L}^T+\lambda^2\boldsymbol{I}_d) = (\boldsymbol{L}^T\boldsymbol{L}+\lambda^2\boldsymbol{I}_m)\boldsymbol{L}^T \ . \quad (16)$$

Regrouping the terms in (16), we arrive at the exact equality between the model estimates $<\boldsymbol{m}>$ from equations (13) and (9):

$$\boldsymbol{L}^T(\boldsymbol{L}\boldsymbol{L}^T+\lambda^2\boldsymbol{I}_d)^{-1} \equiv (\boldsymbol{L}^T\boldsymbol{L}+\lambda^2\boldsymbol{I}_m)^{-1}\boldsymbol{L}^T \ . \quad (17)$$

To obtain equation (17), multiply both sides of (16) by $(\boldsymbol{L}^T\boldsymbol{L}+\lambda^2\boldsymbol{I}_m)^{-1}$ from the left and by $(\boldsymbol{L}\boldsymbol{L}^T+\lambda^2\boldsymbol{I}_d)^{-1}$ from the right. For $\lambda \neq \boldsymbol{0}$, both these matrices are indeed invertible.

Not only the optimization estimate, but also the form of the objective function, is exactly equivalent for both data-space and mode-space cases. The objective function of model-space least squares $\hat{\boldsymbol{r}}^T\hat{\boldsymbol{r}}$ is connected with the data-space objective function $\hat{\boldsymbol{m}}^T\hat{\boldsymbol{m}}$ by the simple proportionality

$$\hat{\boldsymbol{r}}^T\hat{\boldsymbol{r}} = \lambda^2\hat{\boldsymbol{m}}^T\hat{\boldsymbol{m}} \ . \quad (18)$$

This fact implies that the iterative methods of optimization – most notably, the conjugate-gradient method (Hestenes and Steifel, 1952) – should yield the same results for both formulations. Of course, this conclusion doesn't take into account the numerical effects of finite-precision computations.

To move to a more general (and interesting) case of "non-trivial" data-space regularization, we need to refer to the concept of model *preconditioning* (Nichols, 1994). A preconditioning operator $\boldsymbol{P}$ is used to introduce a new model $\boldsymbol{x}$ with the equality

$$\boldsymbol{m} = \boldsymbol{P x} \; . \tag{19}$$

Substituting definition (19) into formula (11), we arrive at the following "preconditioned" form of the operator $\boldsymbol{G_d}$:

$$\tilde{\boldsymbol{G}}_{\boldsymbol{d}} = \left[\begin{array}{cc} \boldsymbol{LP} & \lambda \boldsymbol{I_d} \end{array} \right] \; . \tag{20}$$

The operator $\tilde{\boldsymbol{G}}_{\boldsymbol{d}}$ applies to the compound model vector

$$\hat{\boldsymbol{x}} = \left[\begin{array}{c} \boldsymbol{x} \\ \boldsymbol{r} \end{array} \right] \; . \tag{21}$$

Substituting formula (20) into (13) leads to the following estimate for $\hat{\boldsymbol{x}}$:

$$<\hat{\boldsymbol{x}}> = \left[\begin{array}{c} <\boldsymbol{x}> \\ <\boldsymbol{r}> \end{array} \right] = \tilde{\boldsymbol{G}}_{\boldsymbol{d}}^{\boldsymbol{T}} \left(\tilde{\boldsymbol{G}}_{\boldsymbol{d}} \tilde{\boldsymbol{G}}_{\boldsymbol{d}}^{\boldsymbol{T}} \right)^{-1} \boldsymbol{d} = \left[\begin{array}{c} \boldsymbol{P^T L^T} \left(\boldsymbol{LPP^T L^T} + \lambda^2 \boldsymbol{I_d} \right)^{-1} \boldsymbol{d} \\ \lambda \left(\boldsymbol{LPP^T L^T} + \lambda^2 \boldsymbol{I_d} \right)^{-1} \boldsymbol{d} \end{array} \right] \; . \tag{22}$$

Applying formula (19), we obtain the corresponding estimate for the initial model $\boldsymbol{m}$, as follows:

$$<\boldsymbol{m}> = \boldsymbol{P} <\boldsymbol{x}> = \boldsymbol{PP^T L^T} \left(\boldsymbol{LPP^T L^T} + \lambda^2 \boldsymbol{I_d} \right)^{-1} \boldsymbol{d} \; . \tag{23}$$

Now we can show that estimate (23) is exactly equivalent to formula (7) from the model-space regularization under the condition

$$\left(\boldsymbol{PP^T} \right)^{-1} = \boldsymbol{D^T D} \; . \tag{24}$$

Condition (24) assumes that the operator $\boldsymbol{C} \equiv \boldsymbol{PP^T}$ is invertible[2]. Consider the operator

$$\boldsymbol{G} = \boldsymbol{L^T L C L^T} + \lambda^2 \boldsymbol{L^T} \; , \tag{25}$$

which is another mapping from the data space to the model space. Grouping the multiplicative factors in two different ways, we can obtain the equality

$$\boldsymbol{G} = \boldsymbol{L^T} (\boldsymbol{LCL^T} + \lambda^2 \boldsymbol{I_d}) = (\boldsymbol{L^T L} + \lambda^2 \boldsymbol{C^{-1}}) \boldsymbol{CL^T} \; , \tag{26}$$

or, in another form,

$$\boldsymbol{CL^T} (\boldsymbol{LCL^T} + \lambda^2 \boldsymbol{I_d})^{-1} \equiv (\boldsymbol{L^T L} + \lambda^2 \boldsymbol{C^{-1}})^{-1} \boldsymbol{L^T} \; . \tag{27}$$

[2]The reader familiar with the statistical literature will recognize in $\boldsymbol{C}$ the model covariance matrix. The statistical roots of the least-square optimization are in the method of maximum likelihood. Connecting this statistical estimation method with least squares requires an assumption of uncorrelated additive noise with zero-mean Gaussian distribution.

The left-hand side of equality (27) is exactly the projection operator from formula (23), and the right-hand side is the operator from formula (7).

Comparing formulas (23) and (7), it is interesting to note that we can turn a trivial regularization into a non-trivial one by simply replacing the exact adjoint operator $\boldsymbol{L^T}$ by the operator $\boldsymbol{CL^T}$, which is a transformation from the data space to the model space, followed by enforcing model correlations with the operator $\boldsymbol{C}$. This fact can be additionally confirmed by the equality

$$\boldsymbol{CL^T}(\boldsymbol{LCL^T} + \lambda^2 \boldsymbol{I_d})^{-1} \equiv (\boldsymbol{CL^T L} + \lambda^2 \boldsymbol{I_m})^{-1} \boldsymbol{CL^T} \ , \tag{28}$$

which is derived analogously to formula (27). Iterative optimization methods, which don't require exact adjoint operators [e.g. the method of conjugate directions (Fomel, 1996)] could be employed for the task.

Though the final results of the model-space and data-space regularization are identical, the effect of preconditioning may alter the behavior of iterative gradient-based methods, such as the method of conjugate gradients. Though the objective functions are equal, their gradients with respect to the model parameters are different. Note, for example, that the first iteration of the model-space regularization yields $\boldsymbol{L^T d}$ as the model estimate regardless of the regularization operator, while the first iteration of the model-space regularization yields $\boldsymbol{CL^T d}$, which is a "simplified" version of the model. Since iteration to the exact solution is never achieved in the large-scale problems, the results of iterative optimization may turn out quite differently. Harlan (1995) points out that the two components of the model-space regularization [equations (1) and (2)] conflict with each other: the first one enforces "details" in the model, while the second one tries to smooth them out. He describes the advantage of preconditioning:

> The two objective functions produce different results when optimization is incomplete. A descent optimization of the original (model-space – *S.F.*) objective function will begin with complex perturbations of the model and slowly converge toward an increasingly simple model at the global minimum. A descent optimization of the revised (data-space – *S.F.*) objective function will begin with simple perturbations of the model and slowly converge toward an increasingly complex model at the global minimum. ... A more economical implementation can use fewer iterations. Insufficient iterations result in an insufficiently complex model, not in an insufficiently simplified model.

Examples in the next section illustrate these conclusions.

EXAMPLES

All the test examples in this section are borrowed directly from (Claerbout, 1994)[3], taking advantage of SEP's standards of reproducible research.

Inverse Linear Interpolation

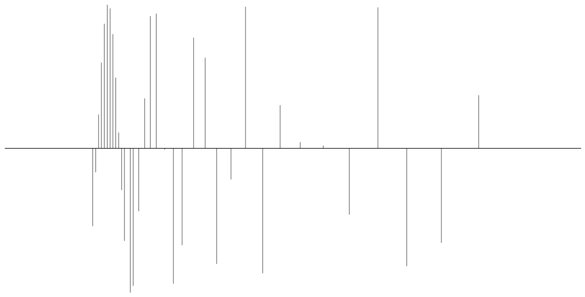

Figure 1: The input data are irregularly sampled. sergey1-data [ER]

The first example is a simple synthetic test for 1-D inverse interpolation. The input data were randomly subsampled (with decreasing density) from a sinusoid (Figure 1). The forward operator $\boldsymbol{L}$ 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. Following Claerbout, I applied convolution with the simple $\mathbf{(1, -1)}$ difference filter as the operator $\boldsymbol{D}$ that forces model continuity (the first-order spline). An appropriate preconditioner $\boldsymbol{P}$ in this case is recursive causal integration. Figures 2 and 3 show the results of inverse interpolation after exhaustive 300 iterations of the conjugate-direction method. The results from the model-space and data-space regularization look similar except for the boundary conditions outside the data range. As a result of using the causal integration for preconditioning, the rightmost part of the model in the data-space case stays at a constant level instead of decreasing to zero. If we specifically wanted a zero-value boundary condition, it wouldn't be difficult to implement it by adding a zero-value data point at the boundary.

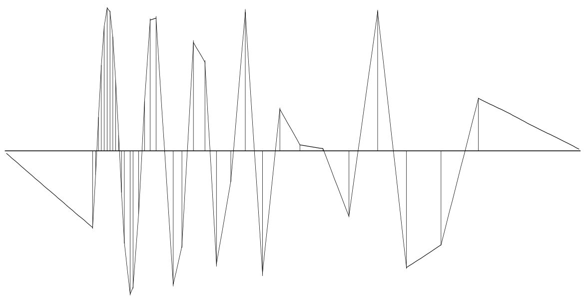

Figure 2: Estimation of a continuous function by the model-space regularization. The difference operator $\boldsymbol{D}$ is the derivative operator (convolution with $\mathbf{(1, -1)}$). sergey1-im1 [ER]

As expected from the general theory, the data-space regularization provides a much faster rate of convergence. Following Matthias Schwab's suggestion, I measured

[3]The Fortran-90 version of the electronic book

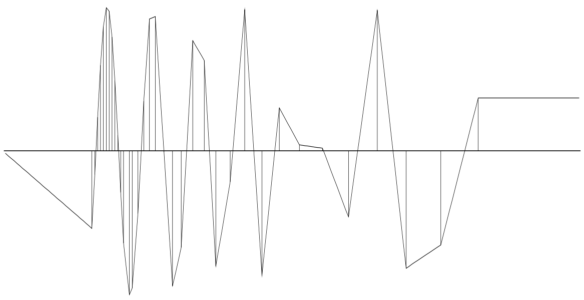

Figure 3: Estimation of a continuous function by the data-space regularization. The preconditioning operator $\boldsymbol{P}$ is causal integration. sergey1-fm1 [ER]

the rate of convergence by the model residual, which is a distance from the current model to the final solution. Figure 5 shows that the data regularization method converged to the final solution in about 6 times fewer iterations than the model regularization. Since the cost of each iteration for each method is roughly equal, the computational economy should be evident. Figure 4 shows the final solution, and the estimates from model- and data-space regularization after only 5 iterations of conjugate directions. The data-space estimate looks much closer to the final solution than its competitor.

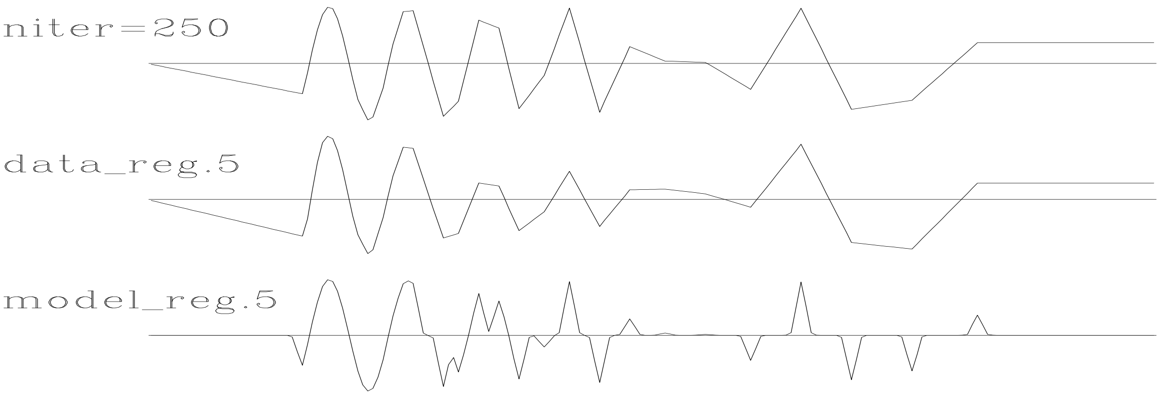

Figure 4: The top figure is the exact solution found in 250 iterations. The middle is with data-space regularization after 5 iterations. The bottom is without model-space regularization after 5 iterations. sergey1-early1 [ER]

Changing the preconditioning operator changes the regularization result. Figure 6 shows the result of data-space regularization after applying a triangle smoother as the model preconditioner.

If, instead of looking for a smooth interpolation, we want to limit the number of frequency components, then the best choice for the model-space regularization operator $\boldsymbol{D}$ is a prediction-error filter (PEF). To obtain a mono-frequency output, we need to use a three-point PEF, which has the $\boldsymbol{Z}$-transform representation $D(Z) = 1 + a_1 Z + a_2 Z^2$. In this case, the corresponding preconditioner $\boldsymbol{P}$ could be the three-point *recursive* filter $P(Z) = 1/(1 + a_1 Z + a_2 Z^2)$. To test this idea, I estimated the PEF $D(Z)$ from the output of inverse linear interpolation (Figure 3), and ran the data-space regularized estimation again, substituting the recursive filter $P(Z) = 1/D(Z)$

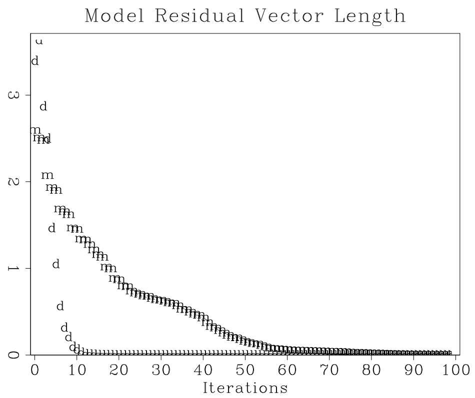

Figure 5: Convergence of the iterative optimization, measured in terms of the model residual. The "d" points stand for data-space regularization; the "m" points, model-space regularization. sergey1-schwab1 [ER]

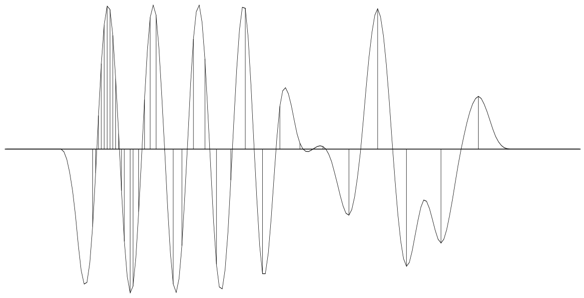

Figure 6: Estimation of a smooth function by the data-space regularization. The preconditioning operator $\boldsymbol{P}$ is a triangle smoother. sergey1-fm6 [ER]

in place of the causal integration. I repeated this two-step procedure three times to get a better estimate for the PEF. The result, shown in Figure 7, exhibits the desired mono-frequency output.

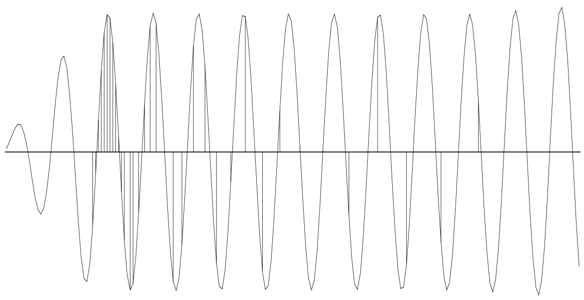

Figure 7: Estimation of a mono-frequency function by the data-space regularization. The preconditioning operator $\boldsymbol{P}$ is a recursive filter (the inverse of PEF). sergey1-pm1 [ER]

Deburst

The next example of regularization is the problem of removing large spiky noise from experimental data. The input synthetic data, shown in the top plot of Figure 8, contains numerous noise spikes and bursts. Some of the noise bursts are a hundred times larger than shown. Simple median smoothing (second top plot in Figure 8) can remove some individual spikes, but fails to provide an adequate output overall. Claerbout suggests *iteratively reweighted least squares* as a robust efficient method of despiking. The operator $\boldsymbol{L}$ in this case is as simple as identity, but we weight equation (1) by some weighting operator $\boldsymbol{W}$, which is chosen to suppress non-Gaussian statistical distribution of the noise. Good results in the considered example were achieved with the "Cauchy" weighting function

$$\boldsymbol{W}(r_i) = \mathbf{diag}\left(\frac{\mathbf{1}}{\sqrt{\mathbf{1}+\frac{r_i^2}{\bar{r}^2}}}\right) , \tag{29}$$

where r_i denote components of the residual $\boldsymbol{r} = \boldsymbol{d} - \boldsymbol{m}$, and $\bar{r}$ is the residual median value. The dependence on $\boldsymbol{r}$ of the weighted operator makes the problem nonlinear, but iterative reweighting allows us to approach it with piecewise linearization.

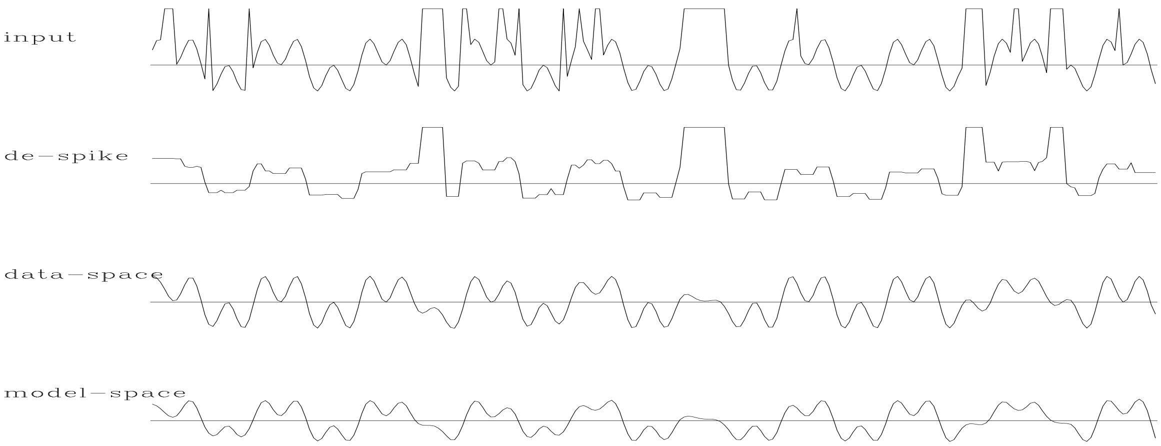

Figure 8: The top is synthetic data with noise spikes and bursts. Next is after despiking with running medians. The two bottom plots are outputs of the deburst process with regularized iteratively reweighted least squares. sergey1-burst4 [ER]

Claerbout's model-space regularization used convolution with the Laplacian filter $(1, -2, 1)$ as the roughening operator $\boldsymbol{D}$. For a comparison with the data-space regularization, I applied triangle smoothing as the preconditioning operator $\boldsymbol{P}$. The results, shown in the two bottom plot of Figure 8, look similar. Both methods succeeded in removing the noise bursts from the data and producing a smooth output. The data-space regularization did a better job of preserving the amplitudes of the original data. This effect partially results from a low dependency on the scaling parameter λ, which I reduced to 0.01 (compared with 1 in the case of model-space regularization.) The model residual plot in Figure 9 shows again a considerably faster convergence for the data-space method, in complete agreement with the theory.

STACK EQUALIZATION

In his notes *3-D seismic imaging*[4], Biondo Biondi discusses partial stacking as the simplest prototype example of an imaging operator. A simple implementation of partial stacking can employ normal moveout (NMO) to correct for traveltime differences

[4]available at http://sepwww.stanford.edu/sep/biondo/GP291/Notes/Ps/notes-latest.ps

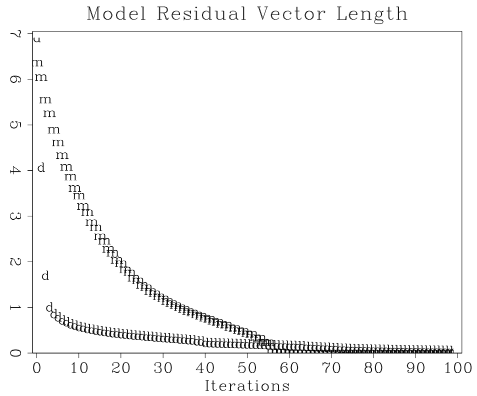

Figure 9: Convergence of the iterative optimization for deburst, measured in terms of the model residual. The "d" points stand for data-space regularization; the "m" points, model-space regularization. sergey1-conv4 [ER]

in the data. Stacking irregular data after residual NMO can be regarded then as an inverse interpolation problem, which one can solve by optimization methods. In this section, I include a simple example of an efficient data-space regularization for optimizing partial stack.

Following Biondi's reproducible example, I use a potion of the Conoco North Sea dataset with offsets, windowed in the range from 400 to 600 m. The geometry of the CMP locations of the input traces is illustrated in Figure 10. The fold distribution, shown as a map view in Figure 11 and as a histogram in Figure 12 is fairly dense overall, but has noticeable holes (empty bins). The regions of zero fold make the inverse interpolation problem underconstrained and suggest an application of a regularized optimization scheme.

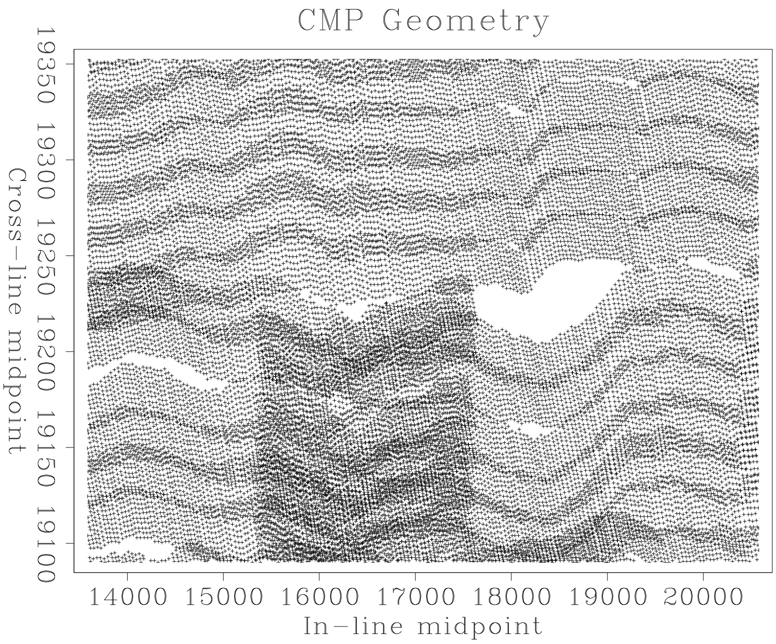

Figure 10: Midpoint geometry of the input data sergey1-cmp [ER]

Figure 13 shows the result of simple binning (the adjoint of nearest neighbor interpolation), normalized by the inverse of the fold density. Obviously, the regions of zero fold don't receive any signal, which can lead to undesirable artifacts in future

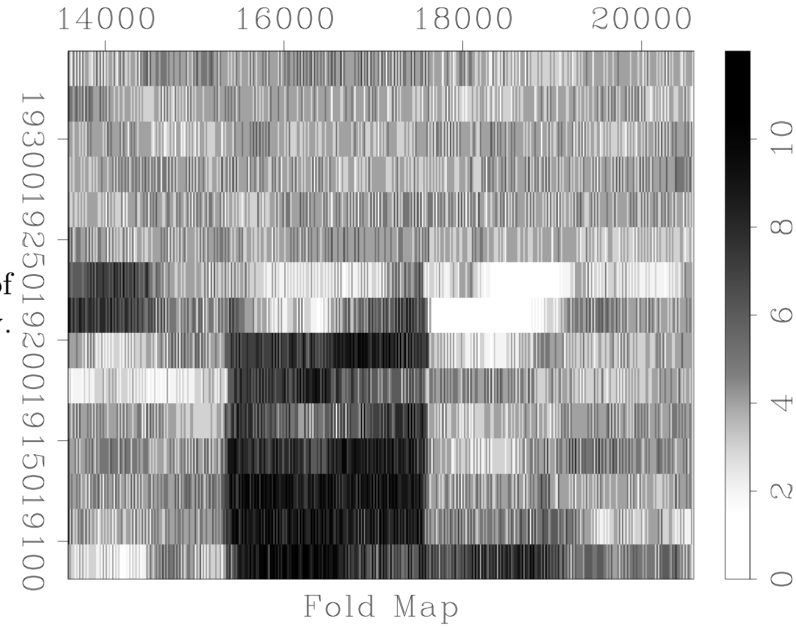

Figure 11: Fold distribution of the input data. A map view. sergey1-fold [ER]

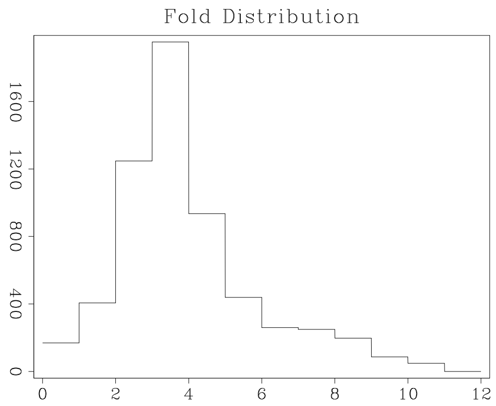

Figure 12: Fold distribution of the input data. A histogram. sergey1-hist [ER]

processing. Figure 14 is the result of non-regularized optimization, with 5 conjugate-gradient iterations at each time slice level. This approach not only fails to fill the empty holes, but also creates unbalanced output because of the poor conditioning of the inverse problem. Figure 15 shows the result of a data-space regularized inversion with a small smoothing filter as a preconditioner. The convergence is fast, and the result looks much improved. Because of the fast convergence of the data-space regularization, the inverse interpolation scheme is inexpensive to apply. One easy way of improving the result further is to change the simple nearest neighbor interpolation operator to a more accurate one. Figure 16 is the result of the regularized inverse interpolation with the Lagrange 4-point interpolator.

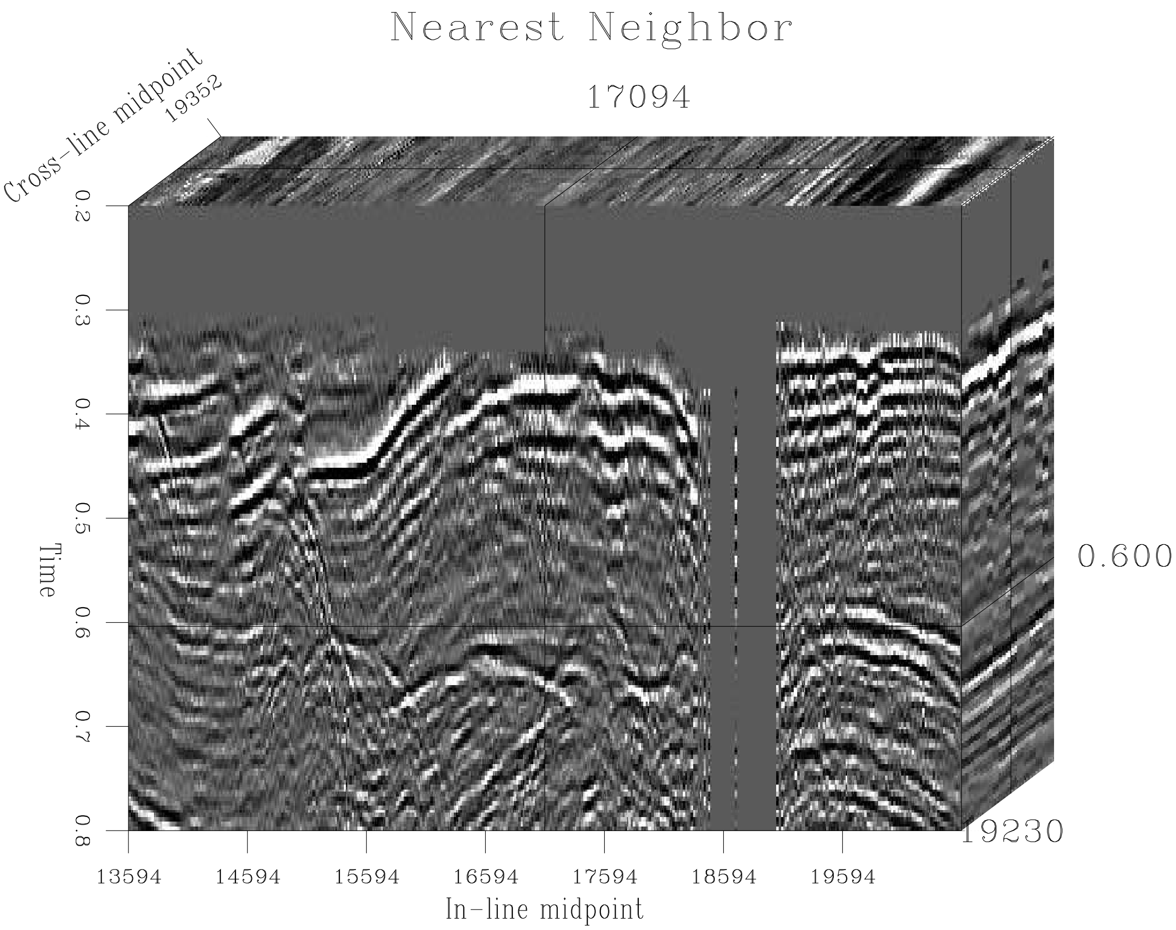

Figure 13: Data transformed to a regular grid by normalized adjoint interpolation (simple binning.) sergey1-bin [ER]

As demonstrated by Biondi et al. (1996), an accurate interpolation for dipping reflector events and diffractions requires the azimuth moveout operator. Another interesting, though untested, approach is to use the inverse of a prediction-error filter for preconditioning the inverse interpolation.

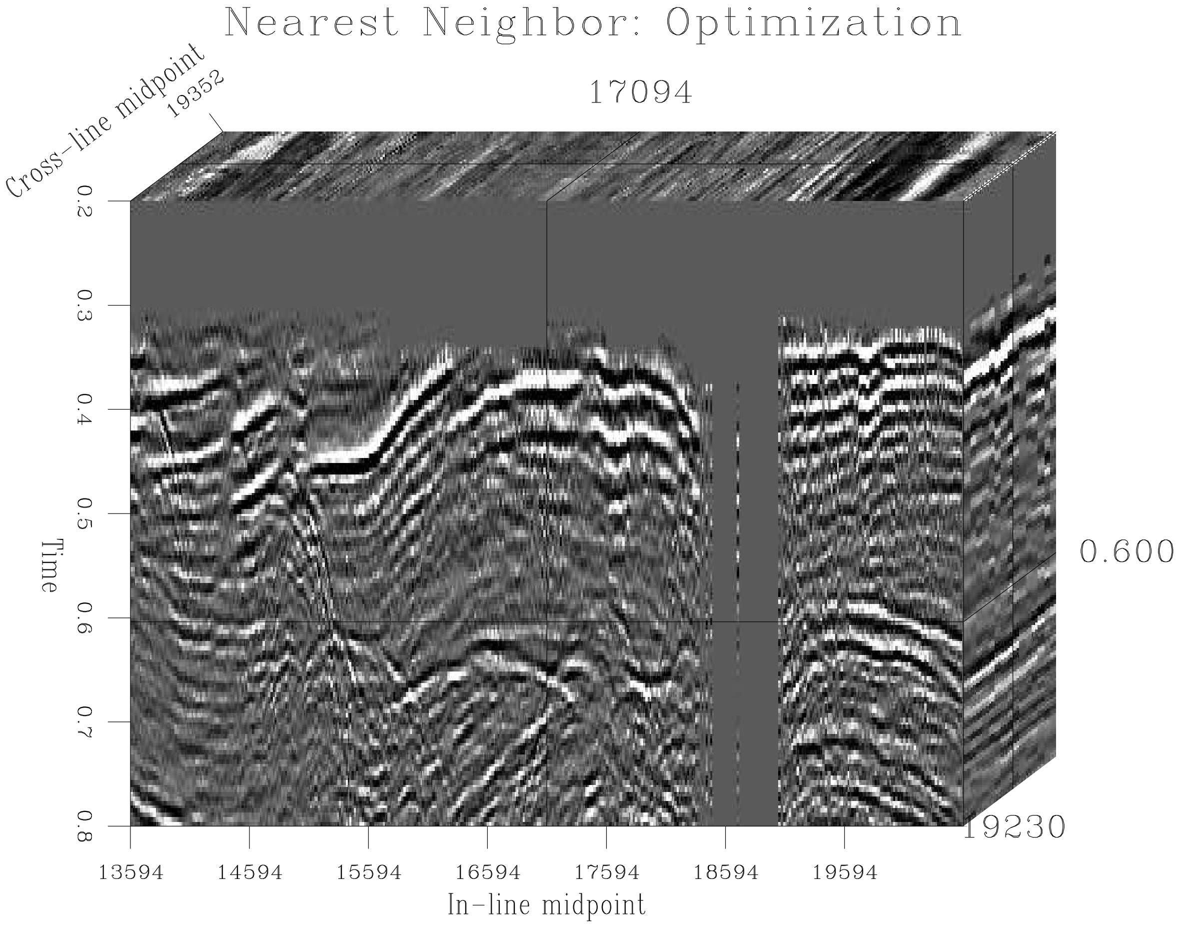

Figure 14: Data transformed to a regular grid by non-regularized inverse interpolation. $\boxed{\text{sergey1-invbin}}$ [ER]

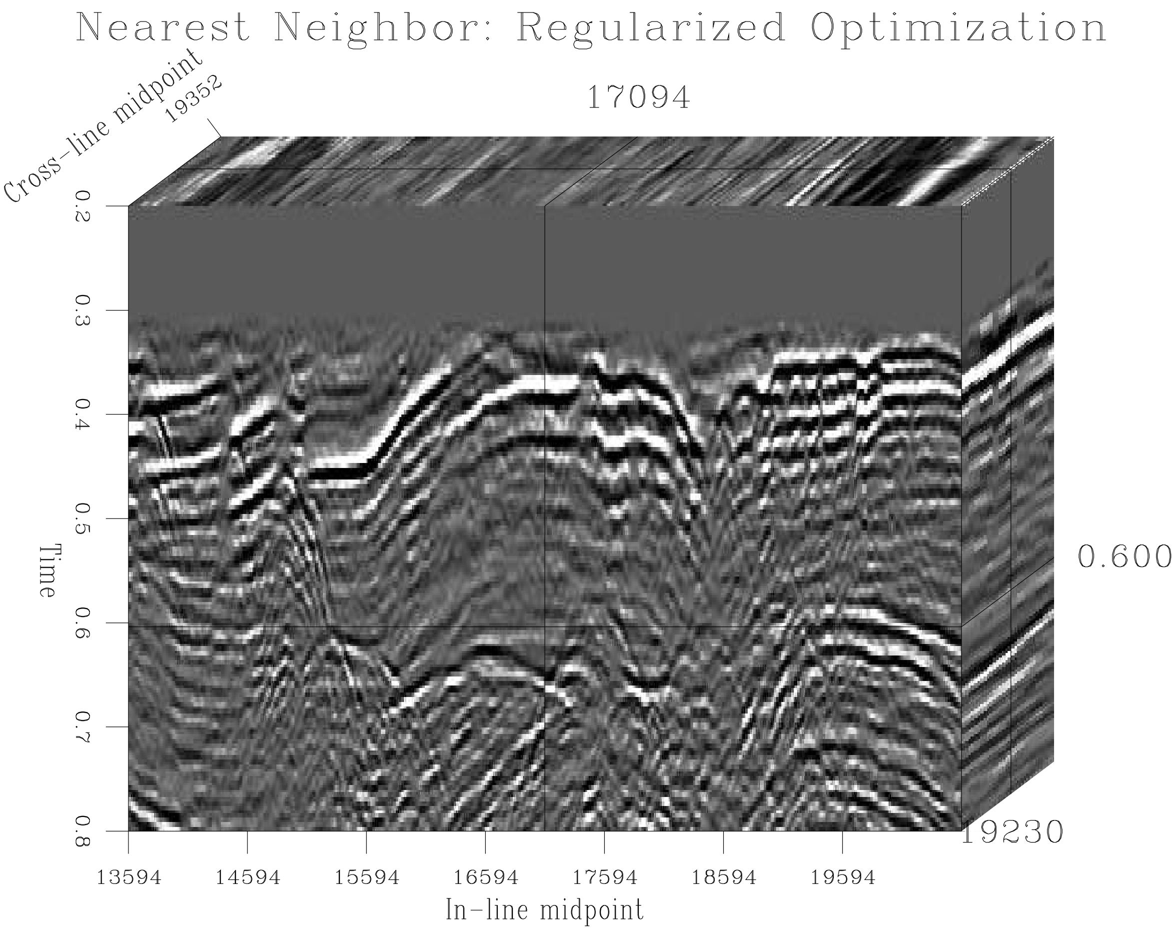

Figure 15: Data transformed to a regular grid by data-space regularized inverse interpolation. sergey1-regbin [ER]

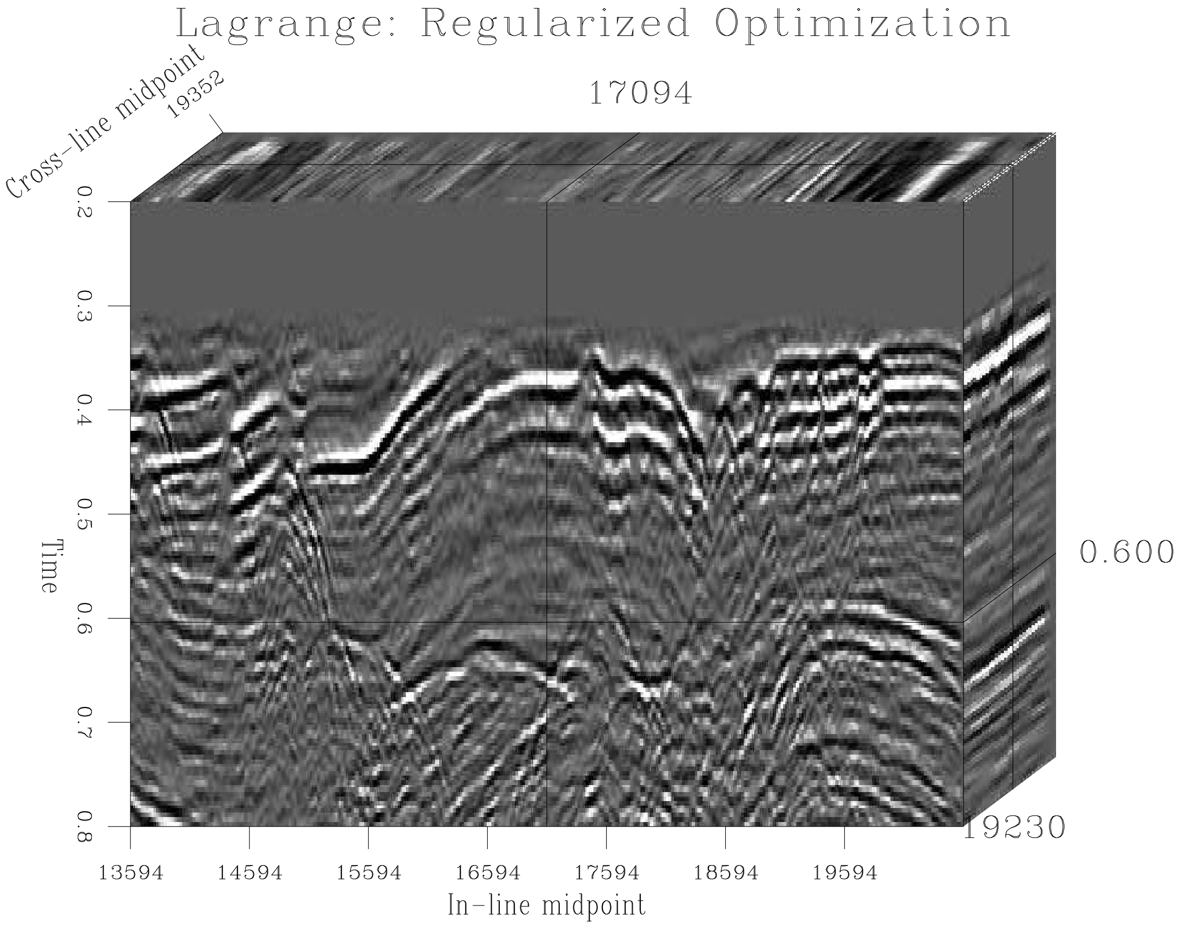

Figure 16: Data transformed to a regular grid by data-space regularized inverse interpolation, using Lagrange's 4-point interpolator as the forward interpolation operator. sergey1-regtin [ER]

DISCUSSION

I summarize the differences between model-space and data-space regularization in Table 0.1. Which of the two approaches is preferable in practical applications? In the case of "trivial" regularization (i.e., constraining the problem by the model power minimization), the answer to this question depends on the relative size of the model and data vectors: data-space regularization may be preferable when the data size is noticeably smaller than the model size. In the case of non-trivial regularization, the answer may additionally depend on the following questions:

- Which of the operators $\boldsymbol{D}$ or $\boldsymbol{P}$ ($\boldsymbol{C^{-1}}$ or $\boldsymbol{C}$) is easier to construct and implement?
- Is an initial estimate for $\boldsymbol{x}$ available? In data-space regularization, it is difficult to start from a non-zero value of the model $\boldsymbol{m}$.
- Is it possible to approximate or to compute analytically one of the inverted matrices in formula (27)?

The derivation of formula (27) suggests experimenting with the operator $\boldsymbol{W_m G W_d} = \boldsymbol{W_m} \left(\boldsymbol{L^T L C L^T} + \lambda^2 \boldsymbol{L^T}\right) \boldsymbol{W_d}$, where $\boldsymbol{W_m}$ approximates $(\boldsymbol{L^T L} + \lambda^2 \boldsymbol{C^{-1}})^{-1}$, and $\boldsymbol{W_d}$ approximates $(\boldsymbol{L C L^T} + \lambda^2 \boldsymbol{I_d})^{-1}$.

Ryzhikov and Troyan (1991) present a curious interpretation of the operator $\boldsymbol{LCL^T}$ in ray tomography applications. In these applications, each data point corresponds to a ray, connecting the source and receiver pair. If the model-space operator $\boldsymbol{C^{-1}} = \boldsymbol{D^T D}$ has the meaning of a differential equation (Laplace's, Helmholtz's, etc.), then, according to Ryzhikov and Troyan, each value in the matrix $\boldsymbol{LCL^T}$ has the physical meaning of a potential energy between two rays, considered as charged strings in a potential field. Such an interpretation leads to a fast direct computation of the matrix operator $\boldsymbol{G}$.

The scaling parameter λ controls the relative amount of *a priori* information added to the problem. In a sense, it allows us to reduce the search for an adequate model null space to a one-dimensional problem of choosing the value of λ. Solving the optimization problem with different values of λ leads to the continuation approach, proposed by Bube and Langan (1994). As Nichols (1994) points out, preconditioning can reduce the sensitivity of the problem to the parameter λ if initial system (1) is essentially under-determined.

ACKNOWLEDGMENTS

Jon Claerbout's encourangement and challenging suggestions motivated my interest in the subject. I am also grateful to Biondo Biondi, Gennady Ryzhikov, Bill Harlan,

Table 0.1: Comparison between model-space and data-space regularization

Trivial regularization	*Model-space*	*Data-space*
effective model	$\boldsymbol{m}$	$\hat{\boldsymbol{m}} = \begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{r} \end{bmatrix}$
effective data	$\hat{\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{0} \end{bmatrix}$	$\boldsymbol{d}$
effective operator	$\boldsymbol{G_m} = \begin{bmatrix} \boldsymbol{L} \\ \lambda \boldsymbol{I_m} \end{bmatrix}$	$\boldsymbol{G_d} = \begin{bmatrix} \boldsymbol{L} & \lambda \boldsymbol{I_d} \end{bmatrix}$
optimization problem	minimize $\hat{\boldsymbol{r}}^{\boldsymbol{T}} \hat{\boldsymbol{r}}$, where $\hat{\boldsymbol{r}} = \hat{\boldsymbol{d}} - \boldsymbol{G_m m}$	minimize $\hat{\boldsymbol{m}}^{\boldsymbol{T}} \hat{\boldsymbol{m}}$ under the constraint $\boldsymbol{G_d} \hat{\boldsymbol{m}} = \boldsymbol{d}$
formal estimate for $\boldsymbol{m}$	$\left(\boldsymbol{L^T L} + \lambda^2 \boldsymbol{I_m}\right) \boldsymbol{L^T d}$	$\boldsymbol{L^T}(\boldsymbol{L L^T} + \lambda^2 \boldsymbol{I_d})^{-1} \boldsymbol{d}$

Non-trivial regularization	*Model-space*	*Data-space*
effective model	$\boldsymbol{m}$	$\hat{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{r} \end{bmatrix}$
effective data	$\hat{\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{0} \end{bmatrix}$	$\boldsymbol{d}$
effective operator	$\boldsymbol{G_m} = \begin{bmatrix} \boldsymbol{L} \\ \lambda \boldsymbol{D} \end{bmatrix}$	$\tilde{\boldsymbol{G}}_{\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{LP} & \lambda \boldsymbol{I_d} \end{bmatrix}$
optimization problem	minimize $\hat{\boldsymbol{r}}^{\boldsymbol{T}} \hat{\boldsymbol{r}}$, where $\hat{\boldsymbol{r}} = \hat{\boldsymbol{d}} - \boldsymbol{G_m m}$	minimize $\hat{\boldsymbol{x}}^{\boldsymbol{T}} \hat{\boldsymbol{x}}$ under the constraint $\tilde{\boldsymbol{G}}_{\boldsymbol{d}} \hat{\boldsymbol{x}} = \boldsymbol{d}$
formal estimate for $\boldsymbol{m}$	$\left(\boldsymbol{L^T L} + \lambda^2 \boldsymbol{C}^{-1}\right) \boldsymbol{L^T d}$, where $\boldsymbol{C}^{-1} = \boldsymbol{D^T D}$	$\boldsymbol{C L^T}(\boldsymbol{L C L^T} + \lambda^2 \boldsymbol{I_d})^{-1} \boldsymbol{d}$, where $\boldsymbol{C} = \boldsymbol{P P^T}$.

Jim Berryman, Bill Symes, Dave Nichols, and Yalei Sun for insightful discussions. The topic of regularization deserves a more detailed study than this short paper, and I hope to continue discussing it with the experts.

The 3-D North Sea dataset was released to SEP by Conoco and its partners, BP and Mobil.

REFERENCES

Biondi, B., Fomel, S., and Chemingui, N., 1996, Application of azimuth moveout to the coherent partial stacking of a 3-D marine data set: SEP–**92**, 1–12.

Bube, K. P., and Langan, R. T., 1994, A continuation approach to regularization for traveltime tomography: 64th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 980–983.

Claerbout, J. F., 1992, Earth Soundings Analysis: Processing Versus Inversion: Blackwell Scientific Publications.

Claerbout, J. F., 1994, Applications of Three-Dimensional Filtering: Stanford Exploration Project.

Fomel, S., and Claerbout, J., 1996, Simple linear operators in Fortran 90: SEP–**93**, 317–328.

Fomel, S., 1996, Least-square inversion with inexact adjoints. Method of conjugate directions: A tutorial: SEP–**92**, 253–265.

Harlan, W. S., 1995, Regularization by model redefinition: http://sepwww.stanford.edu/oldsep/harlan/papers/regularization.ps.gz.

Hestenes, M. R., and Steifel, E., 1952, Methods of conjugate gradients for solving linear systems: J. Res. Nat. Bur. Stand., **49**, 409–436.

Nichols, D., 1994, A simple example of a null space and how to modify it: SEP–**82**, 177–182.

Ryzhikov, G., and Troyan, V., 1991, On regularization methods in 3-D ray tomography: Geophysical Data Interpretation by Inverse Modeling, Proc. of the 9-th International Seminar on Model Optimization in Exploration Geophysics, 53–61.

Tikhonov, A. N., and Arsenin, V. Y., 1977, Solution of ill-posed problems: John Wiley and Sons.

APPENDIX A

REGULARIZED REUSABLE LINEAR SOLVER

Experimenting with different forms of regularization, different kinds of interpolation and regularization operators, and different optimization algorithms was made easy with an improved design of the generic solver subroutine (module.)

The Fortran-90[5] subprograms in this appendix are simplified from the actual version, which is more heavily loaded with optional parameters. For example, the actual solver routine can implement an iterative reweighting that I used in the deburst example, work with non-linear operators, etc. Two solver subroutines together with some others (e.g. LSQR solver) are packed in a MODULE, with overloading of the subroutine names, so that the user can easily switch from one solver to another without changing the name of the program (and without even knowing the details of its implementation.)

The data-space regularization routine on the facing page takes three functions as its arguments. Functions `oper` and `reg` correspond to the linear operators $\boldsymbol{L}$ and $\boldsymbol{P}$. They comply with the generic linear operator interface, defined by Fomel and Claerbout (1996). Function `solv` implements one step of an optimization descent. Its arguments are a logical parameter `forget`, which controls a conditional restarting of the optimization, the current effective model `mod`, the gradient vector `g`, the data residual vector `rr`, and the conjugate gradient vector `gg`. An example of a `solv` function is `conjgrad` on page 164, which implements a classic version of the conjugate-gradient method[6]. Subroutine `lin_solver_dat` constructs the effective model vector `x`, which consists of the model-space part `xm` and the data-space part `xd`. Similarly, the effective gradient vector `g` is split into the the model-space part `gm` and the data-space part `gd`. Subroutine `chain` from module `chainmod` on page 163 is called to compute a chain of the operators $\boldsymbol{P}$ and $\boldsymbol{L}$ (`reg` and `oper`).

The model-space regularization routine on page 162 has an analogous design. In this case, function `reg` corresponds to the model regularization operator $\boldsymbol{D}$. We construct the effective residual vector `rr`, which consists of the model-space part `rm` and the data-space part `rd`. Similarly, the effective conjugate gradient vector `gg` is split into the the model-space part `gm` and the data-space part `gd`. Subroutine `array` from module `chainmod` on page 163 is called to compute an array of the operators $\boldsymbol{L}$ and $\boldsymbol{D}$ (`oper` and `reg`).

[5]Sidestepping the religious C++ versus Fortran war, I would like to point out that an object-oriented design is more important than the actual implementation. The design should be easily transferrable from Fortran-90 to other advanced languages.

[6]Our current library has about 6 other functions of the same interface, implementing different linear and nonlinear optimization methods.

```
subroutine lin_solver_dat (oper, solv, reg, nreg, mod, dat, niter, eps, mod0)
  interface
     integer function oper (adj, add, mod, dat)
       logical, intent (in) :: adj, add
       real, dimension (:)  :: mod, dat
     end function oper
     integer function solv (forget, mod, g, rr, gg)
       logical              :: forget
       real, dimension (:)  :: mod, g, rr, gg
     end function solv
     integer function reg (adj, add, mod, dat)
       logical, intent (in) :: adj, add
       real, dimension (:)  :: mod, dat
     end function reg
  end interface
  real, dimension (:),   intent (in)           :: dat         ! data
  real, dimension (:),   intent (in), optional :: mod0        ! initial model
  real, dimension (:),   intent (out)          :: mod         ! model
  integer,               intent (in)           :: niter, nreg ! size of x
  real,                  intent (in)           :: eps         ! scaling

  real, dimension (size (dat) + nreg), target  :: x, g
  real, dimension (size (dat))                 :: rr, gg
  real, dimension (:), pointer                 :: xm, xd, gm, gd
  integer                                      :: iter, stat
  logical                                      :: forget

  xm => x (1 : nreg) ; xd => x (1 + nreg:) ; xd = 0.
  gm => g (1 : nreg) ; gd => g (1 + nreg:)
  if (present (mod0)) then
     xm = mod0 ; call chain (oper, reg, .false., .false., xm, rr, mod)
     rr = dat - rr
  else
     xm = 0. ; rr = dat
  end if

  forget = .false.
  do iter = 1, niter
     call chain (oper, reg, .true. , .false., gm, rr, mod)  ; gd =   eps*rr
     call chain (oper, reg, .false., .false., gm, gg, mod)  ; gg =gg+eps*gd
     stat = solv (forget, x, g, rr, gg)
  end do
  stat = reg (.false., .false., xm, mod)
end subroutine lin_solver_dat
```

```
subroutine lin_solver_mod (oper, solv, reg, nreg, mod, dat, niter, mod0)
  interface
     integer function oper (adj, add, mod, dat)
       logical, intent (in) :: adj, add
       real, dimension (:)  :: mod, dat
     end function oper
     integer function solv (forget, mod, g, rr, gg)
       logical             ::forget
       real, dimension (:) :: mod, g, rr, gg
     end function solv
     integer function reg (adj, add, mod, dat)
       logical, intent (in) :: adj, add
       real, dimension (:)  :: mod, dat
     end function reg
  end interface
  real, dimension (:), intent (in)            :: dat         ! data
  real, dimension (:), intent (in), optional  :: mod0        ! initial model
  real, dimension (:), intent (out)           :: mod         ! model
  integer,             intent (in)            :: niter, nreg ! size of D mod

  real, dimension (size (mod))                :: g
  real, dimension (size (dat) + nreg), target :: rr, gg
  real, dimension (:), pointer                :: rd, rm, gd, gm
  integer                                     :: iter, stat
  logical                                     :: forget

  rm => rr (1 : nreg) ; rd => rr (1 + nreg :)
  gm => gg (1 : nreg) ; gd => gg (1 + nreg :)

  if (present (mod0)) then
     mod = mod0
     stat = oper (.false., .false., mod, rr) ; rr = dat - rr
     stat = reg  (.false., .false., mod, rm) ; rm =     - rm
  else
     mod = 0. ; rd = dat ; rm = 0.
  end if

  forget = .false.
  do iter = 1, niter
     call array (oper, reg, .true. , .false., g, rd, rm)
     call array (oper, reg, .false., .false., g, gd, gm)
     stat = solv (forget, mod, g, rr, gg)
  end do
end subroutine lin_solver_mod
```

```
module chainmod
contains

  subroutine chain (oper1, oper2, adj, add, mod, dat, tmp)
    interface
       integer function oper1 (adj, add, mod, dat)
         logical, intent (in)  :: adj, add
         real, dimension (:)   :: mod, dat
       end function oper1
       integer function oper2 (adj, add, mod, dat)
         logical, intent (in)  :: adj, add
         real, dimension (:)   :: mod, dat
       end function oper2
    end interface
    logical, intent (in)  :: adj, add
    real, dimension (:)   :: mod, dat, tmp

    integer               :: stat1, stat2
    if (adj) then
       stat1 = oper1 ( .true., .false., tmp, dat)
       stat2 = oper2 ( .true.,     add, mod, tmp)
    else
       stat2 = oper2 (.false., .false., mod, tmp)
       stat1 = oper1 (.false.,     add, tmp, dat)
    end if
  end subroutine chain

  subroutine array (oper1, oper2, adj, add, mod, dat1, dat2)
    interface
       integer function oper1 (adj, add, mod, dat)
         logical, intent (in) :: adj, add
         real, dimension (:)  :: mod, dat
       end function oper1
       integer function oper2 (adj, add, mod, dat)
         logical, intent (in) :: adj, add
         real, dimension (:)  :: mod, dat
       end function oper2
    end interface
    logical, intent (in)  :: adj, add
    real, dimension (:)   :: mod, dat1, dat2

    integer               :: stat1, stat2
    if (adj) then
       stat1 = oper1 (.true.,    add, mod, dat1)
       stat2 = oper2 (.true., .true., mod, dat2)
    else
       stat1 = oper1 (.false.,   add, mod, dat1)
       stat2 = oper2 (.false.,   add, mod, dat2)
    end if
  end subroutine array

end module chainmod
```

```
module conjgrad_mod
  real, dimension (:), allocatable, private  :: s, ss
  real, parameter,                 private  :: eps = 1.e-12

contains

  subroutine conjgrad_close ()
    deallocate (s, ss)
  end subroutine conjgrad_close

  function conjgrad (forget, x, g, rr, gg) result (stat)
    integer             :: stat
    real, dimension (:) :: x, g, rr, gg
    logical             :: forget

    real, save          :: rnp
    real                :: rn, alpha, beta

    rn = dot_product (g, g)
    if (.not. allocated (s)) then
       allocate (s  (size (x )))
       allocate (ss (size (rr)))
       forget = .true.
    end if

    if (forget .or. rnp < eps) then
       alpha = 0.d0
    else
       alpha = rn / rnp
    end if

    s =   g + alpha *  s    ! model step
    ss = gg + alpha * ss    ! data  step

    beta = dot_product (ss, ss)
    if (beta > eps) then
       alpha = rn / beta
       x  =  x + alpha *  s ! update model
       rr = rr - alpha * ss ! update residual
    end if

    rnp = rn ; forget = .false.
    stat = 0
  end function conjgrad

end module conjgrad_mod
```