Multidimensional recursive filters via a helix with application to velocity estimation and 3-D migration

Jon Claerbout, Stanford University

SUMMARY

For many years it has been true that our most powerful signal-analysis techniques are in *one*-dimensional space, while our most important applications are in *multi*-dimensional space. The helical coordinate system makes a giant step towards overcoming this difficulty.

FILTERING ON A HELIX

Figure 1 shows some two-dimensional shapes that are convolved together. The left panel shows an impulse response function, the center shows some impulses, and the right shows the superposition of responses.

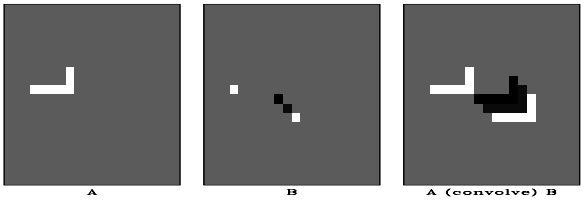

Figure 1: Two-dimensional convolution (as done in one dimension).

A surprising, indeed amazing, fact is that Figure 1 was not computed with a two-dimensional convolution program. It was computed with a one-dimensional computer program. It could have been done with anybody's one-dimensional convolution program, either in the time domain or in the fourier domain. This magical trick is done with the helical coordinate system.

A basic idea of filtering, be it in one dimension, two dimensions, or more, is that you have some filter coefficients and some sampled data; you pass the filter over the data; at each location you find an output by crossmultiplying the filter coefficients times the underlying data and summing the terms.

The helical coordinate system is much simpler than you might imagine. Ordinarily, a plane of data is thought of as a collection of columns, side by side. Instead, imagine the columns stored end-to-end, and then coiled around a cylinder. This is the helix. Fortran programmers will recognize that Fortran's way of storing 2-D arrays in one-dimensional memory is exactly what we need for this helical mapping. Seismologists sometimes use the word "supertrace" to describe a collection of seismograms stored "end-to-end".

Figure 2 shows a helical mesh for 2-D data on a cylinder. Darkened squares depict a 2-D filter shaped like the Laplacian operator. The input data, the filter, and the output data are all on helical meshes all of which could be unrolled into linear strips. A compact 2-D filter like a Laplacian, on a helix is a sparse 1-D filter with long empty gaps.

Since the values output from filtering can be computed in any order, we can slide the filter coil over the data coil in any direction. The order that you produce the outputs is irrelevant. You could compute the results in parallel. We could, however, slide the filter over the data in the screwing order that a nut passes over a bolt. The screw order is the same order that would be used if we were to unwind the coils into one-dimensional strips and convolve them across one another. The same filter coefficients overlay the same data values if the 2-D coils are unwound into 1-D strips. The helix idea allows us to obtain the same convolution output in either of two ways, a one-dimensional way, or a two-dimensional way. I used the one-dimensional way to compute the obviously two-dimensional result in Figure 1.

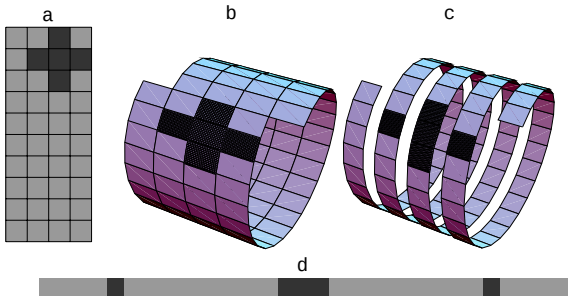

Figure 2: Filtering on a helix. The same filter coefficients overlay the same data values if the 2-D coils are unwound into 1-D strips. (drawing by Sergey Fomel)

Convolution creates an output Q_t from an input P_t and a filter F_τ with $Q_t = P_t + \sum_{\tau>0} F_\tau P_{t-\tau}$. Deconvolution (polynomial division) can undo convolution (polynomial multiplication) by backsolving, by simply rearranging the terms $P_t = Q_t - \sum_{\tau>0} F_\tau P_{t-\tau}$. Deconvolution is recursive filtering. Recursive filter outputs cannot be computed in parallel, but must be computed sequentially as in one dimension, namely, in the order that the nut screws on the bolt.

Deconvolution (polynomial division) can undo convolution (polynomial multiplication). A magical property of the helix is that we can consider 1-D convolution to be the same as 2-D convolution. Hence is a second magical property: We can use 1-D *de*convolution to undo convolution, whether that convolution was 1-D or 2-D. Thus, we have discovered how to undo 2-D convolution. We have discovered that 2-D deconvolution on a helix is equivalent to 1-D deconvolution. The helix enables us to do multidimensional deconvolution.

Recursive filtering sometimes solves big problems with astonishing speed. It can propagate energy rapidly for long distances. Unfortunately, recursive filtering can also be unstable. The most interesting case, near resonance, is also near instability. There is a large literature and extensive technology about recursive filtering in one dimension. The helix allows us to apply that technology to two (and more) dimensions. It is a wonderful insight. We could not previously do 2-D deconvolution because we had no stability theory for it. We cannot simply use polynomial division to undo the 2-D Laplacian operator for example, because the output will diverge. Poles and zeros tell us about 1-D stability but we don't have them for 2-D polynomials.

In 3-D we simply append one plane after another (like a 3-D Fortran array). It is easier to code than to explain or visualize a spool or torus wrapped with string, etc.

Figures 3 and 4 contain this 2-D filter

$$\begin{bmatrix} 0 & -1/4 \\ 1 & -1/4 \\ -1/4 & -1/4 \end{bmatrix} \tag{1}$$

Let us experiment using this 2-D filter as a recursive filter. In Figure 3 the input is shown on the left. This input contains two copies of the filter (1) near the top of the frame and some impulses near the bottom boundary. The second frame in Figure 3 is the result of deconvolution by the filter (1). Notice that deconvolution turns the filter itself into an

impulse, while it turns the impulses into comet-like images. The use of a helix is seen by the comet images wrapping on the vertical axis.

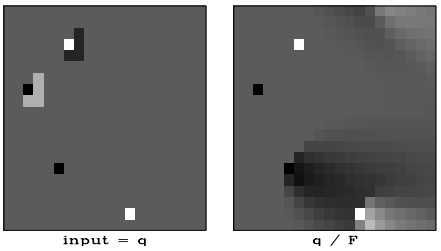

Figure 3: Illustration of 2-D deconvolution. Left is the input. Right is after deconvolution with the filter (1)

In Figure 4, many inputs are tested. Starting from the left are a low-pass blob, a Ricker wavelet, the filter (1) itself, and a couple impulses, one near the bottom boundary. The second frame shows deconvolution by the filter (1). The third frame compounds the second frame with an adjoint (reverse time) deconvolution. (Instead of blowing plumes to the right, it blows them to the left.) The fourth frame convolves the result with the original filter and its adjoint; and we see we are back where we started. No errors, no evidence remains of any of the boundaries where we have wrapped and truncated!

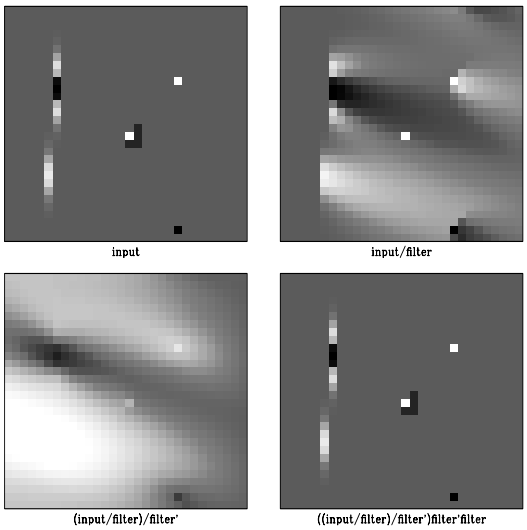

Figure 4: Recursive filtering backwards (leftward on the space axis) is done by the *adjoint* of 2-D deconvolution. Here we see that 2-D deconvolution compounded with its adjoint is exactly inverted by 2-D convolution and its adjoint.

In seismology we often have occasion to steer summation along beams. Such an impulse response is shown in Figure 5.

I have long had an interest in filters that would destroy plane waves. The inverse of such a filter creates plane waves. A filter that creates two plane waves is illustrated in figure 6.

Time-series analysis is rich with concepts that the helix now allows us to apply to many dimensions. First is the notion of an impulse function. Observe that an impulse function on the 2-D surface of the helical cylinder maps to an impulse function on the 1-D line of the unwound coil. An autocorrelation function that is an impulse corresponds both to a white (constant) spectrum in 1-D and to a white (constant) spectrum in 2-D. An autocorrelation of a typical two-dimensional data field will drop off with two-dimensional distance from the zero lag. On the

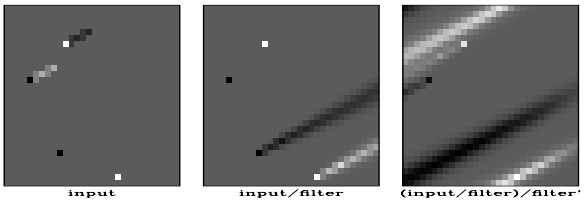

Figure 5: This filter is my guess at a simple low-order 2-D filter whose inverse times its inverse adjoint, is approximately a dipping seismic arrival.

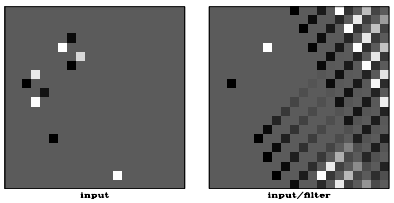

Figure 6: This filter is my guess at a simple low-order 2-D filter whose inverse contains plane waves of two different dips. One of them is spatially aliased.

one-dimensional helix, the autocorrelation gets re-energized when the lag is an integer multiple of the circumference of the helix. A causal filter in one dimension has a curious shape on the two-dimensional helix. I use the convention that the zero-lag response of the 1-D filter has the value "1". In one dimension, the causal filter has zeros before the "1" and various values after it. Supposing that nonzero filter coefficients lie within a short distance (two lags) from the "1", we can extract from the helix the 1-D causal filter and view it as a two-dimensional array

$$\begin{array}{ccc} h & c & 0 \\ p & d & 0 \\ q & e & \mathbf{1} \\ s & f & a \\ u & g & b \end{array} = \begin{array}{ccc} h & c & \cdot \\ p & d & \cdot \\ q & e & \cdot \\ s & f & a \\ u & g & b \end{array} + \begin{array}{ccc} \cdot & \cdot & 0 \\ \cdot & \cdot & 0 \\ \cdot & \cdot & \mathbf{1} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{array} \qquad (2)$$

where $a, b, c, ..., u$ are adjustable coefficients. Thus we conclude that the 2-D analog of a 1-D causal filter has its abrupt beginning along the side of the 2-D filter.

A special causal filter that unites many well established concepts in time-series analysis is the prediction-error-filter (PEF). A 2-D PEF, like a 1-D PEF, is a causal filter with adjustable coefficients as in the array (2), that are adjusted to minimize the filter's output energy (for a particular input signal). That the 2-D PEF should have its beginning along a side (instead of a corner) has always been an abstract and difficult concept, until I fell upon the helix explanation. The PEF has magical mathematical properties and stable recursions. Space does not permit me to list the many properties of one-dimensional signal analysis that are now, because of the helix, awaiting application to multidimensional data.

THE HELIX AND FINITE DIFFERENCES

Discretize the (x, y)-plane to an $N \times M$ array and pack the array into a vector of $N \times M$ components. Likewise pack the Laplacian operator $\partial_{xx} + \partial_{yy}$ into a matrix. For a 4×2 plane, that matrix is shown in equation (3).

$$-\nabla^2 \;=\; \left[\begin{array}{cccc|cccc} -4 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ 1 & -4 & 1 & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & 1 & -4 & 1 & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & 1 & -4 & h & \cdot & \cdot & 1 \\ \hline 1 & \cdot & \cdot & h & -4 & 1 & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & 1 & -4 & 1 & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & -4 & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & -4 \end{array}\right] \tag{3}$$

The two-dimensional matrix of coefficients for the Laplacian operator is shown in (3), where, on a cartesian space, $h = 0$, and in the helix geometry, $h = 1$. Notice that the partitioning becomes transparent for the helix, $h = 1$. With the partitioning thus invisible, the matrix simply represents one-dimensional convolution and we have an alternative analytical approach, Fourier Transform. We often need to solve sets of simultaneous equations with a matrix similar to (3). A costly method is to factor the matrix into upper and lower triangular form that can be "backsolved" which in this case amounts to recursive filtering.

The Fourier approach is roughly equivalent to factoring $-\nabla^2$ into upper and lower triangular matrices, but it is much faster owing to the convolutional nature of the matrix. The (negative of the) Laplacian operator is regarded as an autocorrelation

$$\mathbf{r} \;=\; (-1, 0, \cdots, 0, -1, 4, -1, 0, \cdots, 0, -1) \tag{4}$$

Using the Kolmogoroff spectral-factorization method, a wavelet is found which has this autocorrelation. This wavelet is

$$\mathbf{a} \;=\; (1.8, -.65, -.04, -.02, \cdots, -.04, -.09, -.2, -.56) \tag{5}$$

and its autocorrelation is (4). Displaying (4) and (5) on a helix, the 2-D autocorrelation is:

$$\mathbf{r} \quad = \quad \begin{bmatrix} & & -1 & \\ -1 & & 4 & -1 \\ & & -1 & \end{bmatrix} \tag{6}$$

and the 2-D wavelet with this autocorrelation is

$$\mathbf{a} \;=\; \begin{bmatrix} & & & & 1.8 & -.65 & -.04 & \cdots \\ \cdots & -.09 & -.2 & -.56 & & & & \end{bmatrix} \tag{7}$$

where now I am wrapping the top row around to a second row. In the representation (7) we see the coefficients diminishing rapidly away from maximum value 1.791. In a more abstract notation, we might write

$$-\nabla^2 \quad = \quad \mathbf{A}'\mathbf{A} \tag{8}$$

Where the triangular matrix $\mathbf{A}$ is constructed from the row $\mathbf{a}$ as follows: Each row of the matrix $\mathbf{A}'$ contains the row $\mathbf{a}$ shifted along the row so that the value 1.791 lies along the main diagonal.

Unfortunately, we see that the factored operator $\mathbf{A}$ has a great number of nonzero terms, but fortunately they seem to be converging rapidly so truncation seems reasonable. We can use this feedback operator to solve Poisson's equation very rapidly. Polynomial division and its adjoint gives us $\mathbf{p} = (\mathbf{q}/\mathbf{A})/\mathbf{A}'$ which means that we have solved the

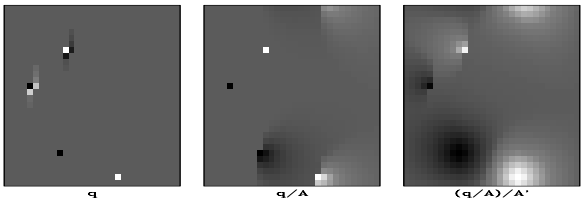

Figure 7: Deconvolution by a filter whose autocorrelation is the two-dimensional Laplacian operator. Solves the Poisson equation.

PDE $\nabla^2\mathbf{p} = -\mathbf{q}$ by using polynomial division on a helix. Using the seven coefficients shown, the cost is fourteen multiplications (because we need to run both ways) per mesh point. An example is shown in Figure 7. The center panel of Figure 7 shows the inverse of the helical derivative.

We are all aware of the factorization of ∇^2 into a divergence dotted into a gradient, where the divergence is the adjoint of the gradient. In two-dimensional physical space, the gradient maps one field to *two* fields. The factorization of $-\nabla^2$ with the helix gives us an operator that maps one field to *one* field. Any fact this basic should be well known in some arcane field of mathematics or theoretical physics. Meanwhile, being ignorant of any pre-existant name, I have chosen the name "helix derivative" or "helical derivative" for the operator $\mathbf{A}$. Since $\mathbf{A}'\mathbf{A} = -\nabla^2$, the operator $\mathbf{A}$ must be something like a derivative. To compare the operator $\frac{\partial}{\partial y}$ to the helix derivative by applying them to a local topographic map. The result shown in Figure 8 is that $\mathbf{A}$ enhances drainage patterns whereas $\frac{\partial}{\partial y}$ enhances mountain ridges.

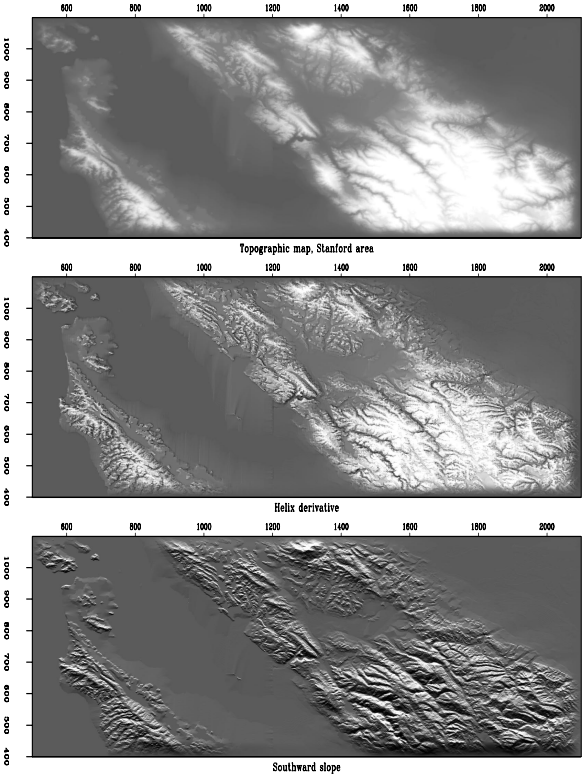

Figure 8: Topography, helical derivative, slope south.

PRELIMINARY APPLICATIONS

Multidimensional recursive filters should have many applications. In our lab we have done some preliminary work on four.

Geoestimation: empty bins, hiding acquistition footprint
The basic formulation of a geophysical estimation problem consists of setting up two goals, one for data fitting, and the other for model smoothing. We have data $\mathbf{d}$, a map or model $\mathbf{m}$, a transformation $\mathbf{L}$, and a roughening filter like the gradient or the helix derivative $\mathbf{A}$. The two goals may be written as:

$$\mathbf{0} \approx \mathbf{r} = \mathbf{Lm} - \mathbf{d} \tag{9}$$
$$\mathbf{0} \approx \mathbf{p} = \mathbf{Am} \tag{10}$$

which defines two residuals, a data residual $\mathbf{r}$, and a model residual $\mathbf{p}$, which are usually minimized by iterative conjugate-gradient, least-squares methods. We can invert the matrix $\mathbf{A}$ when the Laplacian $-\nabla^2 = \mathbf{A}'\mathbf{A}$ is factored with the invertible helix derivative but not when the Laplacian is factored into $\mathbf{div} \cdot \mathbf{grad}$. We change the free variable in the fitting goals from $\mathbf{m}$ to $\mathbf{p}$ (by inverting (10)) with $\mathbf{m} = \mathbf{A}^{-1}\mathbf{p}$ and substituting into both goals getting new goals

$$\mathbf{0} \approx \mathbf{LA}^{-1}\mathbf{p} - \mathbf{d} \tag{11}$$
$$\mathbf{0} \approx \mathbf{p} \tag{12}$$

I find that iterative solvers converge much more quickly when the free variable is the roughened map (or preconditioned variable) $\mathbf{p}$ rather than the map $\mathbf{m}$ itself.

Figure 9 (left) shows ocean depth measured by a Seabeam apparatus. Locations not surveyed are evident as the homogeneous gray area. Using a process akin to "blind deconvolution" a 2-D prediction error filter $\mathbf{A}$ is found. Then missing data values are estimated and shown on the right. Preconditioning with the helix speeded this estimation by a factor of about 30. The figure required a few seconds of calculation for about 10^5 unknowns.

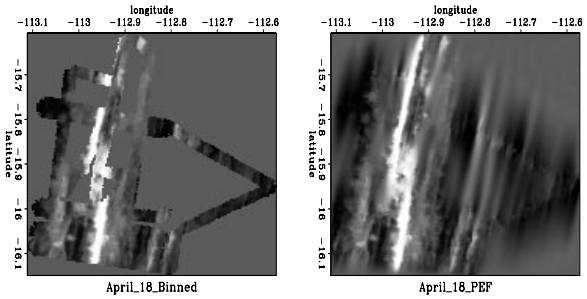

Figure 9: Filling empty bins with a prediction-error filter.

Wavefield extrapolation and 3-D poststack migration
Elsewhere in this abstract volume, find my paper with James Rickett entitled, "Implicit 3-D depth migration by wavefield extrapolation with helical boundary conditions".

Velocity spreading and smoothing
Another application spreads velocity (or any measurements) from randomly located wells to fill all space. It also solves an empty bin problem. This problem differs from the ocean-depth problem because the dip is not a constant function of time and space. It also differs because nowhere is information given densely enough that we can estimate the PEF. We assume the dip is estimatable by some other means, such as by looking at the dip of the reflectivity. From such dip information, we specify a two-dimensional recursive filter such as that in Figure 5. The difference from Figure 5 is that the dip is a variable function of time and space. Early results (not shown) of Bob Clapp are encouraging.

Interval velocity estimation
Geophysical measurements serve to specify RMS velocity reliably in some locations, but not all, thereby leaving a null space to be specified according to geological preconceptions. We define: $\mathbf{d}$ is a data vector whose components range over the vertical travel time depth τ, and whose component values contain the scaled RMS velocity squared $\tau v_{\text{RMS}}^2/\Delta\tau$. $\mathbf{C}$ is the matrix of causal integration, a lower triangular matrix of ones. $\mathbf{u}$ is a vector whose components range over vertical traveltime depth τ, and whose component values contain the interval velocity squared v_{interval}^2. $\mathbf{G}$ is a good data selector. It could be stack power or coherency, or it could be unity where you seek boundaries of a blocky model and zero inside blocks. $\mathbf{A}$ is the multidimensional PEF charactorizing the covariance of our desired velocity function. $\mathbf{u} = \mathbf{u}_0 + \mathbf{A}^{-1}\mathbf{Gp}$ implicitly defines the preconditioning variable $\mathbf{p}$. The fitting goals for blocky velocity models that are consistant over the midpoint axis is:

$$\begin{aligned} \mathbf{0} &\approx \mathbf{r} = \mathbf{CA}^{-1}\mathbf{Gp} + \mathbf{Cu}_0 - \mathbf{d} \\ \mathbf{0} &\approx \mathbf{p} \end{aligned} \tag{13}$$

After fitting with $\mathbf{p}$, we derive the squared interval-velocity with $\mathbf{u} = \mathbf{u}_0 + \mathbf{A}^{-1}\mathbf{Gp}$.

(space reserved for illustration, if I get one)

ACKNOWLEDGEMENT AND REFERENCE

I am delighted to acknowledge many inspirational conversations with Sergey Fomel, who also made the helix illustration, and who coded the first verification that the helix preconditioning drastically speeds the Seabeam estimation.

Background material for this article may be found in my free textbooks found at my web site http://sepwww.stanford.edu/sep/prof/.