Stanford Exploration Project, Report 65, January 13, 1998, pages 211–221

Chapter 4

Univariate problems

This chapter looks at problems in which there is just one unknown. These are called *univariate* problems. Such problems illustrate pitfalls, alternatives, and opportunities before we move on to the next five chapters with *multiple* unknowns (that obscure the pitfalls, alternatives, and opportunities).

The chapter begins with a section on crosstalk, "Segregating P and S waves" that is omitted here but may be found in SEP-61 page 365.

HOW TO DIVIDE NOISY SIGNALS

(This section is a heavy rewrite of classical material with a surprising twist near the end). Multiplication in the Fourier domain is convolution in the time domain. Division is deconvolution. We have already encountered the polynomial-division feedback operation $X(Z) = Y(Z)/F(Z)$. The division is challenging when F has observational error. First by switching from the Z-domain to the ω-domain we avoid needing to know if F is minimum phase. But the ω-domain has pitfalls too. We may find for some real ω that $F(Z(\omega))$ vanishes so we cannot divide by that F. Failure erupts if zero-division occurs. More insidiously, when zero-division is avoided by a near miss, then results turn out poor.

Dividing by zero smoothly

Think of any real numbers x, y, and f and any program containing $x = y/f$. How can we change the program so that it never divides by zero? A popular answer is to change $x = y/f$ to $x = yf/(f^2 + \epsilon^2)$ where ϵ is any tiny value. When $|f| >> |\epsilon|$ then x is approximately y/f as expected. But when the divisor f vanishes, the result is safely zero instead of infinity. The transition between the two is smooth but some criterion is needed to choose the value of ϵ. This is may not be the only way or best way to cope with zero division, but it is a good way and it permeates the subject of signal analysis.

In the Fourier domain, X, Y, and F, are complex numbers. Then what do we do with $X = Y/F$? Multiply top and bottom by the complex conjugate $\overline{F}$, and again add ϵ^2 to the denominator. Thus

$$X(\omega) = \frac{\overline{F(\omega)}\ Y(\omega)}{\overline{F(\omega)}F(\omega)\ +\ \epsilon^2} \tag{1}$$

Now the denominator must always be a positive number greater than zero, so division is always safe.

Damped solution

Equation (1) is the solution to an optimization problem that arises in many applications. Now that we know the solution, let us formally define the problem. First, we solve a simpler problem with real values. We choose to minimize the quadratic function of x

$$Q(x) = (fx - y)^2 + \epsilon^2 x^2 \tag{2}$$

The second term is called a *damping factor* because it prevents x from going to $\pm\infty$ when $f = 0$. Set $dQ/dx = 0$ getting

$$0 = f(fx - y) + \epsilon^2 x \tag{3}$$

which yields the earlier answer $x = fy/(f^2 + \epsilon^2)$.

With Fourier transforms, the signal X is a complex number at each frequency ω. So we generalize equation (2) to

$$Q(\bar{X}, X) = (\overline{FX - Y})(FX - Y) + \epsilon^2 \bar{X}X = (\bar{X}\bar{F} - \bar{Y})(FX - Y) + \epsilon^2 \bar{X}X \tag{4}$$

To minimize Q we could use a real-values approach where we express $X = u + iv$ in terms of two real values u and v and then set $\partial Q/\partial u = 0$ and $\partial Q/\partial v = 0$. Alternately, we can use a complex-values approach where we set $\partial Q/\partial X = 0$ and $\partial Q/\partial \bar{X} = 0$. Let us examine $\partial Q/\partial \bar{X}$.

$$\frac{\partial Q(\bar{X}, X)}{\partial \bar{X}} = \bar{F}(FX - Y) + \epsilon^2 X = 0 \tag{5}$$

The derivative $\partial Q/\partial X$ is the complex conjugate of $\partial Q/\partial \bar{X}$. So if one is zero, the other is too. Thus we don't need to specify both $\partial Q/\partial X = 0$ and $\partial Q/\partial \bar{X} = 0$ because either one is enough. I usually take $\partial Q/\partial \bar{X} = 0$. Solving (5) for X gives the answer (1).

Unknown filter

Equation (1) solves $Y = XF$ for X. This is called the deconvolution problem with a known wavelet F. An opposite interpretation of $Y = XF$ is that the filter F is unknown, but the input X and output Y are given. Here stabilization is less likely to be needed, but it would be if the input and output do not fill the frequency band. Performing derivatives as above, but with respect to F instead of X gives again equation (1) with X and F interchanged.

$$F(\omega)=\frac{\overline{X(\omega)}\ Y(\omega)}{\overline{X(\omega)}X(\omega)\ +\ \epsilon^2} \tag{6}$$

Example of deconvolution with a known wavelet

The top trace of Figure 1 shows a marine reflection seismic trace from northern Scandinavia. The most pronounced feature is a series of multiple reflections from the ocean bottom seen at .6s intervals. These reflections share a similar waveshape that alternates in polarity. (The alternation of polarity will be more apparent after deconvolution.) The alternation of polarity results from a negative reflection coefficient at the ocean surface (where the acoustic pressure vanishes). The spectrum of the top trace has a comb pattern that results from the periodicity of the multiples.

../jon1/./Fig/harlan2.ps

Figure 1: The top half is time functions and the bottom half is the corresponding frequency functions. The top signal is a marine seismogram 4s long. A wavelet windowed between 0.5s and 1s was used to deconvolve the signal with a tiny ϵ (bottom) and a larger ϵ (middle). (Calculated by Bill Harlan.)

Letting the raw trace be Y, a filter F was chosen by extracting a portion of Y, that portion from 0.5 to 1 second that is the water bottom reflection. The spectrum of this windowed portion of the trace is like that of the whole trace except the comb pattern is absent (for reasons given in chapter 1). The second trace of Figure 1 shows the result of a damped deconvolution of the data with this wavelet. The reflections now look like bandlimited impulses, and the spectrum appears only slightly broader than before. The third trace of Figure 1 shows the result of deconvolution with a much smaller damping factor. It is noisy at high frequencies.

Experiments with ϵ^2, showed that satisfactory results were found within a large numerical range. The damped deconvolution shown in Figure 1 took ϵ to be 0.01 times the average over ω of $|F|$ and for the underdamped example it was 0.00001. Bill Harlan and I each experimented with ϵ varying with frequency but did not get results interesting enough to show.

Self fulfilling prophecy?

Pitfalls arise from the popular theoretical analysis explaining ϵ. Since this theory is easily misused, I'd like to show you the trap. First, the model $Y = FX$ is supplemented by an explicit additive noise $N(\omega)$ becoming

$$Y(\omega)=N(\omega) + F(\omega)X(\omega) \tag{7}$$

Because we have many frequencies we can hope to learn more by averaging over frequency. Of particular interest is the *variance* (described more fully in chapter 11)

$$\sigma_X^2=\frac{1}{n} \sum_{j=1}^{n} \bar{X}(\omega_j)X(\omega_j) \tag{8}$$

and likewise σ_N^2.

The general linear estimation method is to minimize something that looks like a sum of *relative* errors.

$$Q(X,N)=\frac{\bar{X}\,X}{\sigma_X^2} \;+\; \frac{\bar{N}\,N}{\sigma_N^2} \tag{9}$$

Notice the variances put both terms of the sum to the same physical units. Requesting the noise N to be as small as possible seems like a good idea and asking X to be small too seems to work against any bad effect of zero division in $X = Y/F$. By introducing (7) into (9) we can eliminate either N or X. Eliminating N we have

$$Q(X)=\frac{\bar{X}\,X}{\sigma_X^2} \;+\; \frac{(\overline{FX-Y})(FX-Y)}{\sigma_N^2} \tag{10}$$

Minimizing $Q(X)$ by setting its derivative by $\bar{X}$ to zero gives

$$0 \;=\; \frac{X}{\sigma_X^2} \;+\; \frac{\bar{F}(FX-Y)}{\sigma_N^2} \tag{11}$$

$$X \;=\; \frac{\bar{F}Y}{\bar{F}F \;+\; \frac{\sigma_N^2}{\sigma_X^2}} \tag{12}$$

This completes the "logical explanation" of equation (1). Equation (12) expresses the answer, but it does so in terms of the unknowns σ_N^2 and σ_X^2. Given an initial estimate of σ_N^2/σ_X^2 equation (12) gives us X and (7) gives N, so we can then compute σ_N^2 and σ_X^2. Presumably these computed values are better than our initial guesses. In statistics, the variances in (12) are called *priors*, and it makes sense to check them, and even more sense to correct them. And from the corrected values we should be able to iterate, further improving the corrections. Equation (12) applies for each of the many frequencies, and there is only a single unknown, the ratio σ_N^2/σ_X^2 so it seems like there is plenty of information and the bootstrapping procedure might work. A pessimist might call this bootstrapping "self fulfilling prophecy." What do you think?

Truth is stranger than fiction. I tried it. With my first starting value for the ratio σ_N^2/σ_X^2 the iteration led to the ratio being infinite. Another starting value led to the ratio being zero. All starting values led to one or the other of these nonsensical values. Eventually I deduced there must be a metastable value dividing the two possibilities, but that is hardly reassuring. I conclude we cannot bootstrap these prior assumptions. The solutions produced do not tend to the correct variance, nor is the variance ratio correct, so they cannot be used to bootstrap the variance. Philosophically we can be thankful the result failed to converge so we did not get a false confidence in the solution. I conclude that linear estimation theory, while beguiling and appearing to be a universal guide to practice, is actually incomplete. The incompleteness will grow in later chapters with multivariate least squares, because there the scalar σ_x^2 becomes a matrix. We continue our search for "universal truth" by studying more examples.

NONSTATIONARITY

The edge effects when moving a window around on a wall of data can be confusing. To illustrate Fortran (Ratfor) technique I wrote a routine `nonstat()` and indented it so you can see that the 1-axis is handled identically to the 2-axis. First look at the looping instructions which for each axis are one line of initialization and test at the top of the loop, and a line for incrementing at the bottom. The `nonstat()` routine moves around boxes of any specified size, jumping along the t-axis and x-axis in steps of any specified size.

```
# slide a window around on a wall of data.  Count times each data point used.
#
subroutine nonstat( n1,n2, w1,w2, k1,k2, count)
integer n1,n2           # size of data wall
integer w1,w2           # size of window
integer k1,k2           # increments for jumping along the wall
integer s1,s2, e1,e2    # starting and ending points of window on wall
integer j1,j2           # output math size of array of windows
integer i1,i2
real    count(n1,n2)    # mathematically it is typically smaller
call zero( n1*n2, count)
j2=1; s2=1; e2=s2+w2-1; while( e2<=n2) {
j1=1; s1=1; e1=s1+w1-1; while( e1<=n1) {
```

```
        do i1=1,w1 {
        do i2=1,w2 {
                count(s1+i1-1, s2+i2-1) =  count(s1+i1-1, s2+i2-1) + 1.
                }}
          j1=j1+1; s1=s1+k1; e1=s1+w1-1
        } j1=j1-1
          j2=j2+1; s2=s2+k2; e2=s2+w2-1
        } j2=j2-1
return; end
```

Typically we'll be filtering in each window. The filter output is generally a residual that will be minimized. We won't be running the filter off the edges of the window because that would cause truncation effects preventing us from having small window sizes. So in a given data window the number of residual values is less than the number of data values. The residual has some natural alignment with the data, and we have the problem of splicing together the residuals from each window into a global residual. To do this, we have a space of global residuals on a mesh the size of the data mesh on which we keep a count of times a window residual has added in to the global residual. When everything is finished, the effect of overlapping windows can be compensated by dividing by the count of residuals at each mesh point.

CONTINUUM PICKING

Picking is the process of identifying dipping seismic events. Here we will do something like that, but we will do it in a continuum, i.e. dips will be picked continuously and set on a uniform mesh. The method is based on the plane-wave destructor operator.

The plane-wave destructor (PWD)

A plane wave in a wavefield $u(t, x) = u(t - px)$ with stepout p can be extinguished with a partial differential operator that we write as a matrix $\mathbf{A}$ where

$$0 \quad \approx \quad v(t,x) \quad = \quad \left(\frac{\partial}{\partial x} - p_i \frac{\partial}{\partial t} \right) \; u(t,x) \tag{13}$$

$$\mathbf{0} \quad \approx \quad \mathbf{v} \quad = \quad \mathbf{A}\mathbf{u} \tag{14}$$

The parameter p is called the wave number or Snell parameter and $|p|$ can take on any value less than $1/v$ where v is the medium velocity. The wave's angle of propagation is $\sin\theta = pv$.

```
#   vv = (aa Dx - pp Dt) uu
#
subroutine wavekill1( aa, pp, m1,n1,n2 , uu, vv)
integer conj, i1,i2, m1,n1,n2
real aa, pp, s11, s12, s21, s22, uu(m1,n2), vv(m1,n2-1)
```

```
s11 = -aa+pp;   s12 = aa+pp
s21 = -aa-pp;   s22 = aa-pp
call zero( m1*(n2-1), vv)
do i1=1, n1-1 {         # vv is one point shorter than uu on both axes.
do i2=1, n2-1 {
        vv(i1,i2) = vv(i1,i2) +
                uu(i1  ,i2) * s11 + uu(i1  ,i2+1) * s12 +
                uu(i1+1,i2) * s21 + uu(i1+1,i2+1) * s22
        }}
return; end
```

In a finite difference representation, the difference operator $\delta_x - p_i\delta_t$ is a two-dimensional filter that fits on a 2×2 differencing star. As a matrix operation, this two dimensional convolution is denoted $\mathbf{A}$. (More details can be found in my previous book "Imaging the Earth's Interior".) The program `wavekill1()` applies the operator $a\delta_x - p\delta_t$ which can be specialized the operators δ_x, δ_t, $\delta_x - p_i\delta_t$. I carefully arranged the side boundaries so the filter never runs off the sides of the data. Thus the output is shorter than the input by one point both the t-axis and the x-axis. The reason for using these side boundaries is so that large data sets can be chopped into independent sections without the boundaries themselves affecting the result. By chopping a large data set into sections, we can handle curved events as piecewise linear.

When only one wave is present and the data is adequately sampled, then finding the best value of p is a single-parameter, linear least-squares problem. Let $\mathbf{x}$ be a vector whose components are values of $\partial u/\partial x$ taken on a mesh in (t, x). Likewise let $\mathbf{t}$ contain $\partial u/\partial t$. Since we wish $\mathbf{x} - p\,\mathbf{t} \approx \mathbf{0}$, we minimize the quadratic function of p

$$Q(p) = (\mathbf{x} - p\,\mathbf{t}) \cdot (\mathbf{x} - p\,\mathbf{t}) \tag{15}$$

by setting to zero the derivative getting

$$p = \frac{\mathbf{x} \cdot \mathbf{t}}{\mathbf{t} \cdot \mathbf{t}} \tag{16}$$

As data will not always fit the model very well, it may be helpful to have some measure of goodness of fit. I suggest

$$Q^2 = 1 \;-\; \frac{(\mathbf{x} - p\,\mathbf{t}) \cdot (\mathbf{x} - p\,\mathbf{t})}{\mathbf{x} \cdot \mathbf{x}} \tag{17}$$

which on inserting $p = (\mathbf{x} \cdot \mathbf{t})/(\mathbf{t} \cdot \mathbf{t})$ leads to

$$Q = \frac{\mathbf{x} \cdot \mathbf{t}}{\sqrt{(\mathbf{x} \cdot \mathbf{x})(\mathbf{t} \cdot \mathbf{t})}} \tag{18}$$

which is known as the *normalized correlation*. The program for this calculation is straightforward. I named the program `puck()` to denote *picking* on a contin*uu*m.

```
# measure coherency and dip, and compute residual  res = (Dx - p Dt) uu
#
subroutine puck ( m1, n1, n2, uu, coh, pp, res )
integer i1, i2, m1, n1,n2
real    uu(m1,n2), res(m1,n2), xx, xt, tt, coh, pp
temporary real dx(m1,n2-1), dt(m1,n2-1)
call wavekill1( 1., 0.,  m1,n1,n2 , uu, dx)
call wavekill1( 0.,-1.,  m1,n1,n2 , uu, dt)
xx = 1.e-30; tt = 1.e-30; xt = 0.
do i1=1,n1-1 {
do i2=1,n2-1 {
        xt = xt + dt(i1,i2) * dx(i1,i2)
        tt = tt + dt(i1,i2) * dt(i1,i2)
        xx = xx + dx(i1,i2) * dx(i1,i2)
        }}
coh = sqrt((xt/tt)*(xt/xx))
pp  =  xt/tt
call wavekill1( 1., pp,  m1,n1,n2 , uu, res)
return; end
```

Finally, an undesirable feature of equation (13) is that the derivatives roughen the data so the spectrum of $\mathbf{x}$ and $\mathbf{t}$ is not that of $\mathbf{u}$. What I am doing now is putting the data through leaky integration on the time axis before I start. I am planning to switch this from t to x so the operator can be thought of as $1 - p\delta_t^x$. Then the residual should be in phase with the input signal so I can subtract it from the data leaving "cleaned up" data.

Nonstationarity, moving windows

Seismic reflections are not generally planar though they are locally so. The `puck()` method was designed to be ignorant of side boundaries so that it can be applied in a small window and the window moved freely about the data. A strength of the method is that the window can be smaller than a wavelength—it can be merely two traces wide. A sample is shown in Figures 2 through 4.

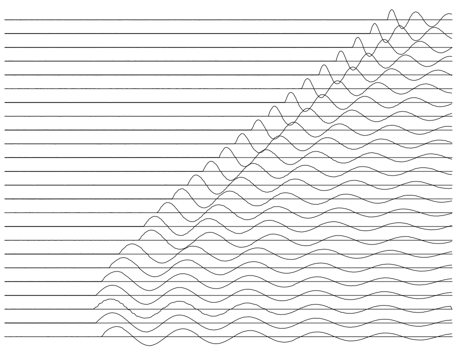

Figure 2: Input synthetic data. jon1-puckin [NR]

The residual shows several features: It is small in the central region of the data; it is large where there is spatial aliasing; and it is rough because it is made from derivatives and the synthetic data was made with nearest neighbor interpolation.

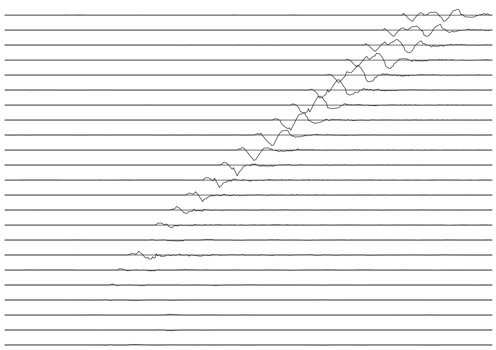

Figure 3: Residuals, i.e. an evaluation of $U_x - pU_t$. jon1-residual [NR]

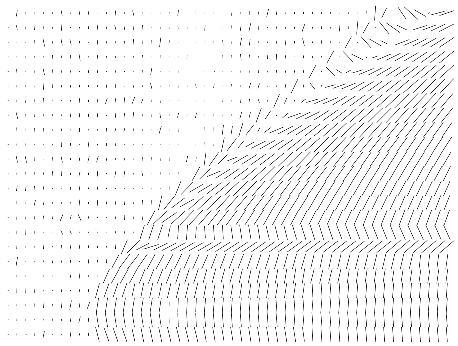

Figure 4: Output values of p are shown by the slope of short line segments. jon1-puckout [NR]

The residuals are small. If a weaker dip were present it would be revealed in the residuals. In Figure 3 there is no weaker dip, only artifacts of inadequate sampling of the signals and the transient onset of the signal. Later in chapter 8 several strong dips will be estimated simultaneously.

```
# slide a window around on a wall of data measuring coherency, dip, residual
#
subroutine slider( n1,n2, w1,w2, k1,k2, data, coh, pp, res)
integer n1,n2, w1,w2, k1,k2, s1,s2, e1,e2, i1,i2
integer p1,p2           # math size of pp(p1,p2), coh(p1,p2)
real    data(n1,n2)                     # input
real    res(n1,n2)                      # outputs. math size (n1-1,n2-1)
real    pp(n1,n2), coh(n1,n2)           # outputs.
temporary real  window(n1,n2), tres(n1,n2), count(n1,n2)
call zero( n1*n2, res)
call zero( n1*n2, count)
p2=1; s2=1; e2=s2+w2-1; while( e2<=n2) {
p1=1; s1=1; e1=s1+w1-1; while( e1<=n1) {
        do i1=1,w1
                do i2=1,w2
                        window(i1,i2) = data(s1+i1-1, s2+i2-1)
        call puck ( n1, w1, w2, window, coh(p1,p2), pp(p1,p2), tres)
        do i1=1,w1-1 {
        do i2=1,w2-1 {
                res(  s1+i1-1, s2+i2-1) =    res(s1+i1-1, s2+i2-1) + tres(i1,i2)
                count(s1+i1-1, s2+i2-1) =  count(s1+i1-1, s2+i2-1) + 1.
                }}
          p1=p1+1; s1=s1+k1; e1=s1+w1-1
        } p1=p1-1
```

```
          p2=p2+1; s2=s2+k2; e2=s2+w2-1
        } p2=p2-1
do i2=1,n2-1
        do i1=1,n1-1
                if( count(i1,i2) > 0. )
                        res(i1,i2) = res(i1,i2) / count(i1,i2)
return; end
```

A disadvantage of the method is that the finite difference operator is susceptible to spatial aliasing as well as distortions at spatial frequencies that are high but not yet aliased. This suggests the next step—estimating missing interlaced traces—that is taken up in chapter 8.