Chapter 8

Spatial aliasing and scale invariance

Landforms are not especially predictable. Therefore, crude PEF approximations are often satisfactory. Wavefields are another matter, however. Consider the "shape" of the acoustic wavefronts at this moment in the room you are in. The acoustic wavefield has statistical order in many senses. If the 3-D volume is filled with waves emitted from a few point sources, then (with some simplifications) what could be a volume of information is actually a few 1-D signals. When we work with wavefronts we can hope for more dramatic, even astounding, results from estimating properly.

> The plane-wave model links an axis that is not aliased (time) with axes (space) that often are.

We often characterize data from any region of (t, x)-space as "good" or "noisy" when we really mean it contains "few" or "many" plane-wave events in that region. Where regions are noisy, there is no escaping the simple form of the Nyquist limitation. Where regions are good we may escape it. Real data typically contains both kinds of regions. Undersampled data with a broad distribution of plane waves is nearly hopeless. Undersampled data with a sparse distribution of plane waves offer us the opportunity to resample without aliasing. Consider data containing a spherical wave. The angular bandwidth in a plane-wave decomposition appears huge *until we restrict attention to a small region* of the data. (Actually a spherical wave contains very little information compared to an arbitrary wave field.) It can be very helpful in reducing the local angular bandwidth if we can deal effectively with tiny pieces of data. If we can deal with tiny pieces of data, then we can adapt to rapid spatial and temporal variations. This chapter will show such tiny windows of data.

8.1 INTERPOLATION BEYOND ALIASING

A traditional method of data interpolation on a regular mesh is a four-step procedure: (1) Set zero values at the points to be interpolated; (2) **Fourier transform**; (3) Set to zero the high frequencies; and (4) Inverse transform. This is a fine method and is suitable for many applications in both one dimension and higher dimensions. However, this method fails to take advantage of our prior knowledge that seismic data has abundant fragments of plane waves that link an axis that is not aliased (time) to axes that might be (space).

8.1.1 Interlacing a filter

The filter below, however, can be designed despite alternate missing traces, because it can destroy plane waves. If the plane wave should happen to pass halfway between the "d" and the "e", those two points could interpolate the halfway point, at least for well-sampled temporal frequencies, and the time axis should always be well sampled.

$$
\begin{array}{ccccccccc}
a & \cdot & b & \cdot & c & \cdot & d & \cdot & e \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot
\end{array}
\tag{8.1}
$$

We could use module `pef` on page 147 to find the filter (1.1), if we set up the lag table `lag` appropriately. Then we could throw away alternate zeroed rows and columns (rescale the lag) to get the filter

$$
\begin{array}{ccccc}
a & b & c & d & e \\
\cdot & \cdot & 1 & \cdot & \cdot
\end{array}
\tag{8.2}
$$

which could be given to subroutine `mis1()` on page 122, because both the filters (1.1) and (1.2) have the same dip characteristics.

```
module lace {                                  # interlace missing traces
  use boundary
  use pef
  use format
  use mis2
contains
  subroutine interlace( dd, xx, known, niter, j, n1,n2, lag1,lag2, a1,a2,a3) {
    integer, intent( in)            :: niter, j, n1,n2, lag1,lag2, a1,a2,a3
    real,    dimension (:)          :: xx                    # output
    logical, dimension (:), pointer :: known, bad            # selectors
    real,    dimension (:), pointer :: dd, aa                # input, PEF
    integer, dimension (:), pointer :: small, large          # helix lags
    integer                         :: i1, i2, i3, na, i
    na = (a1-lag1) + (a2-lag2)*a1 + (a3-1)*a1*a2             # filter size
    allocate (small (na), large (na), aa (na), bad(size(dd)))
    do i3 = 1, a3 {                                          # find helix lags
    do i2 = 1, a2 { if( i3 == 1 .and. i2 < lag2                 ) cycle
```

```
    do i1 = 1, a1 { if( i3 == 1 .and. i2 == lag2 .and. i1 <= lag1) cycle
              i  =  i1-lag1    + (i2-lag2)*a1 + (i3-1)*a1*a2
       small( i) =  i1-lag1    + (i2-lag2)*n1 + (i3-1)*n1*n2  # a(i1,i2,i3)
       large( i) = (i1-lag1)*j + (i2-lag2)*n1 + (i3-1)*n1*n2  # j = jump
    }}}
    call bound3 (n1,n2,lag1*j,lag2,a1*j,a2,a3,bad)            # bad regressors
    call find_pef  (dd, aa, large, bad, na*2)                 # estimate aa
    call print3 (n1,n2,lag1,  lag2,a1,  a2,a3,aa, small)      # print aa
    call mis1 (niter, xx, aa, small, known, .false., .false.) # interlace
    deallocate (small, large, aa, bad)
  }
}
```

Figure 1.1 shows three plane waves recorded on five channels and the interpolated data. Both the original data and the interpolated data can be described as "beyond

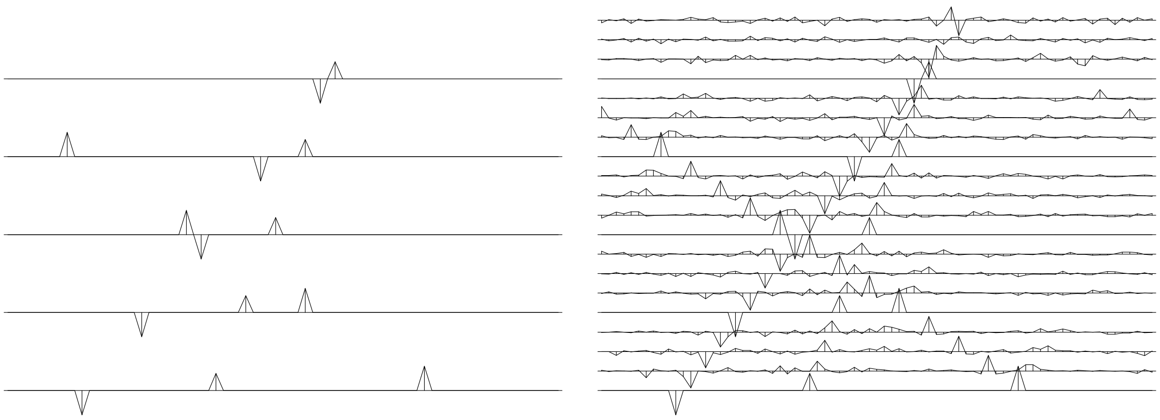

Figure 8.1: Left is five signals, each showing three arrivals. With the data shown on the left (and no more), the signals have been interpolated. Three new traces appear between each given trace, as shown on the right. lal-lace390 [ER]

aliasing," because on the input data the signal shifts exceed the signal duration. The calculation requires only a few seconds of a two-stage least-squares method, in which the first stage estimates a PEF (inverse spectrum) of the known data, and the second uses the PEF to estimate the missing traces.

Before the method of this section can be applied to field data for migration, remember that the data must be broken into many overlapping tiles of size about like those shown here and the results from each tile pieced together.

8.2 MULTISCALE, SELF-SIMILAR FITTING

Large objects often resemble small objects. To express this idea we use ***axis scaling*** and we apply it to the basic theory of prediction-error filter (PEF) fitting and missing-data estimation.

Equations (1.3) and (1.4) compute the same thing by two different methods, $\mathbf{r} = \mathbf{Y}\mathbf{a}$ and $\mathbf{r} = \mathbf{A}\mathbf{y}$. When it is viewed as fitting goals minimizing $||\mathbf{r}||$ and used along with suitable constraints, (1.3) leads to finding filters and **spectra**, while (1.4) leads to finding **missing data**.

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ \hline r_6 \\ r_7 \\ r_8 \\ r_9 \end{bmatrix} = \begin{bmatrix} y_2 & y_1 \\ y_3 & y_2 \\ y_4 & y_3 \\ y_5 & y_4 \\ y_6 & y_5 \\ \hline y_3 & y_1 \\ y_4 & y_2 \\ y_5 & y_3 \\ y_6 & y_4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_1 \\ \mathbf{Y}_2 \end{bmatrix} \mathbf{a} \tag{8.3}$$

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ \hline r_6 \\ r_7 \\ r_8 \\ r_9 \end{bmatrix} = \begin{bmatrix} a_2 & a_1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & a_2 & a_1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & a_2 & a_1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & a_2 & a_1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & a_2 & a_1 \\ \hline a_2 & \cdot & a_1 & \cdot & \cdot & \cdot \\ \cdot & a_2 & \cdot & a_1 & \cdot & \cdot \\ \cdot & \cdot & a_2 & \cdot & a_1 & \cdot \\ \cdot & \cdot & \cdot & a_2 & \cdot & a_1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \mathbf{y} \tag{8.4}$$

A new concept embedded in (1.3) and (1.4) is that one filter can be applicable for different **stretching**s of the filter's time axis. One wonders, "Of all classes of filters, what subset remains appropriate for stretchings of the axes?"

8.2.1 Examples of scale-invariant filtering

When we consider all functions with vanishing gradient, we notice that the gradient vanishes whether it is represented as $(1, -1)/\Delta x$ or as $(1, 0, -1)/2\Delta x$. Likewise for the Laplacian, in one dimension or more. Likewise for the wave equation, as long as there is no viscosity and as long as the time axis and space axes are stretched by the same amount. The notion of "dip filter" seems to have no formal definition, but the idea that the spectrum should depend mainly on slope in Fourier space implies a filter that is scale-invariant. I expect the most fruitful applications to be with **dip filter**s.

Resonance or **viscosity** or damping easily spoils scale-invariance, however. The resonant frequency of a filter shifts if we stretch the time axis. The difference equations

$$y_t - \alpha y_{t-1} = 0 \tag{8.5}$$

$$y_t - \alpha^2 y_{t-2} = 0 \tag{8.6}$$

both have the same solution $y_t = y_0 \alpha^{-t}$. One difference equation has the filter $(1, -\alpha)$, while the other has the filter $(1, 0, -\alpha^2)$, and α is not equal to α^2. Although these operators differ, when $\alpha \approx 1$ they might provide the same general utility, say as a roughening operator in a fitting goal.

Another aspect to scale-invariance work is the presence of "parasitic" solutions, which exist but are not desired. For example, another solution to $y_t - y_{t-2} = 0$ is the one that oscillates at the Nyquist frequency.

(Viscosity does not necessarily introduce an inherent length and thereby spoil scale-invariance. The approximate frequency independence of sound absorption per wavelength typical in real rocks is a consequence of physical inhomogeneity at all scales. See for example **Kjartansson**'s **constant Q** viscosity, described in **IEI**. Kjartansson teaches that the decaying solutions $t^{-\gamma}$ are scale-invariant. There is no "decay time" for the function $t^{-\gamma}$. Differential equations of finite order and difference equations of finite order cannot produce $t^{-\gamma}$ damping, yet we know that such damping is important in observations. It is easy to manufacture $t^{-\gamma}$ damping in Fourier space; $\exp[(-i\omega)^{\gamma+1}]$ is used. Presumably, difference equations can make reasonable approximations over a reasonable frequency range.)

8.2.2 Scale-invariance introduces more fitting equations

The fitting goals (1.3) and (1.4) have about double the usual number of fitting equations. Scale-invariance *introduces extra equations.* If the range of scale-invariance is wide, there will be more equations. Now we begin to see the view to the end of the tunnel:

1. Refining a computational mesh improves accuracy.
2. Refining a data mesh makes empty bins.
3. Empty bins spoil analysis.
4. If there are not too many empty bins we can find a PEF.
5. With a PEF we can fill the empty bins.
6. To get the PEF and to fill bins we need enough equations.
7. Scale-invariance introduces more equations.

An example of these concepts is shown in Figure 1.2. Additionally, when we have a PEF, often we still cannot find missing data because conjugate-direction iterations do not converge fast enough (to fill large holes). Multiscale convolutions should converge quicker because they are like mesh-refinement, which is quick. An example of these concepts is shown in Figure 1.3.

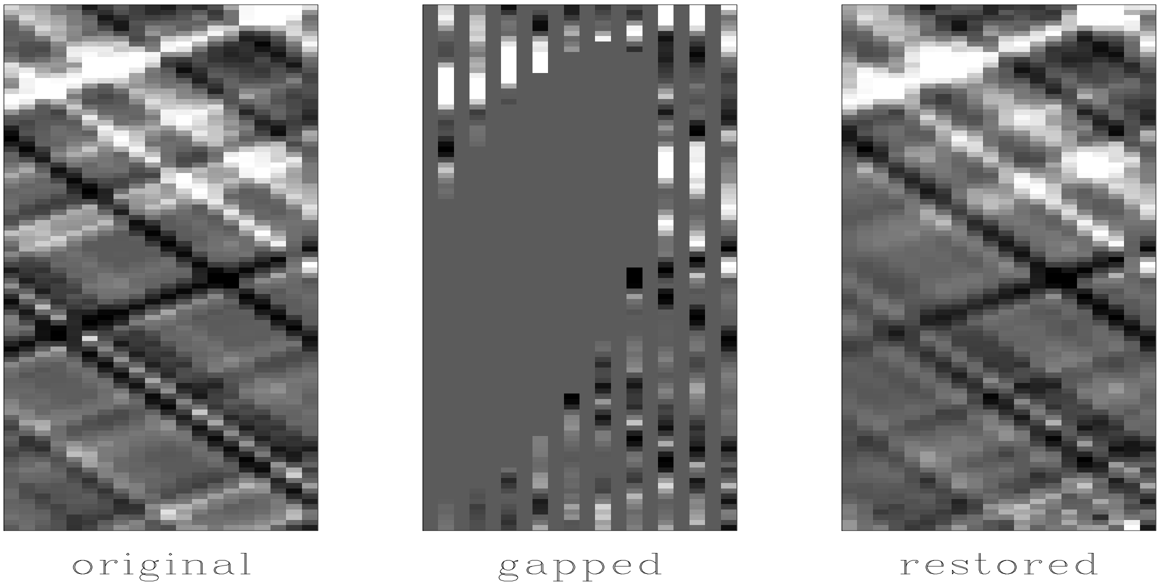

Figure 8.2: Overcoming aliasing with multiscale fitting. $\boxed{\text{lal-mshole90}}$ [ER]

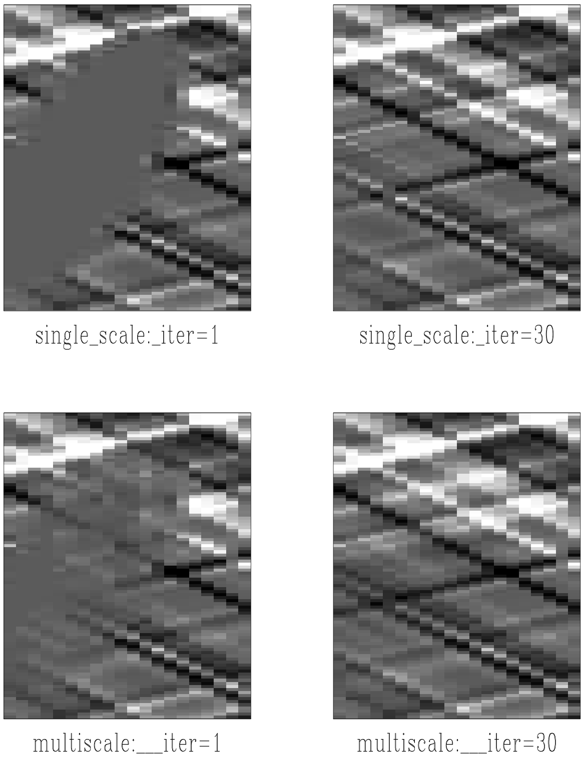

Figure 8.3: Large holes are filled faster with multiscale operators. $\boxed{\text{lal-msiter90}}$ [ER]

8.2.3 Coding the multiscale filter operator

Equation (1.3) shows an example where the first output signal is the ordinary one and the second output signal used a filter interlaced with zeros. We prepare subroutines that allow for more output signals, each with its own filter interlace parameter given in the table `jump(ns)`. Each entry in the jump table corresponds to a particular scaling of a filter axis. The number of output signals is `ns` and the number of zeros interlaced between filter points for the j-th signal is `jump(j)-1`.

First we examine code for estimating a prediction-error filter that is applicable at many scales. The coding is much like that in `hconest` on page 144, the new ingredient being the loop over scales and the jumping along axes.

```
module mshconest {      # multi-scale helix convolution, adjoint is the filter.
  integer, private                   :: nx, ns
  real,    dimension( :),    pointer :: x
  logical, dimension( :, :), pointer :: bad     # output depends on bad inputs
  integer, dimension( :),    pointer :: lag, jump
#% _init( bad, x, lag, jump)
   nx = size( x); ns = size( jump)
#% _lop( a (:), y (nx,ns))
    integer  ia, ix, iy, is
    do is = 1, ns {
    do ia = 1, size( a) {
         do iy = 1  + lag( ia)*jump( is), nx {    if( bad( iy, is)) cycle
            ix = iy - lag( ia)*jump( is)
             if( adj)
                    a( ia) += y( iy, is) * x( ix)
             else
                    y( iy, is) += a( ia) * x( ix)
         }
    }}
}
```

The **multiscale prediction-error filter** finding subroutine is nearly identical to the usual subroutine `find_pef()` on page 147. (That routine cleverly ignores missing data while estimating a PEF.) To easily extend `pef` to multiscale filters we replace its call to the ordinary 2-D filter subroutine by a call to `mshconest`. Of course, it also needs to know that multiscale filtering has a larger output space (with more panels) than single-scale filtering does.

```
module mspef {  # Find multi-scale prediction-error filter (helix magic)
  use mshconest
  use cgstep_mod
  use solver_mod
contains
  subroutine find_pef( yy, aa, lag, jump, mx, niter) {
    integer,                   intent( in)  :: niter
    real,    dimension( :),    pointer      :: yy
    logical, dimension( :, :), pointer      :: mx
```

```
    integer, dimension( :),    pointer      :: lag, jump
    real,    dimension( :),    intent( out) :: aa
    integer                                 :: is, nx
    real,    dimension( size( yy), size( jump)) :: dd
    nx = size( yy)
    do is = 1, size( jump)
       dd (:,is) = -yy
    call mshconest_init( mx, yy, lag, jump)
    call solver( mshconest_lop, cgstep, aa, pack(dd, .true.), niter)
    call cgstep_close()
  }
}
```

Likewise, I coded up the operator in (1.4).

```
module mshelicon {                                  # Multi-scale Convolution
#                           Designed for gapped filters (helical 2-D, 3-D, etc).
#             Requires the filter be causal with an implicit "1." at the onset.
integer                         :: nx, ns
real,    dimension (:), pointer :: aa
integer, dimension (:), pointer :: lag, jump
#%  _init (nx, ns, aa, lag, jump)
if( any( lag <= 0)) call erexit ("Filter is not causal")
#%  _lop ( xx (nx),  yy (nx,ns))
integer iy, ix, ia, is
do is= 1, ns {
if( adj)              # zero lag
        xx += yy (:,is)
else
        yy (:,is) += xx
do ia= 1, size(aa) {
do ix= 1, nx-lag(ia)*jump(is) {  iy = ix + lag(ia)*jump(is)
        if( adj)
                        xx(ix) += yy(iy,is) * aa(ia)
        else
                        yy(iy,is) += xx(ix) * aa(ia)
        }}}
}
```

The multiscale missing-data module `msmis2` is just like the usual missing-data module `mis2` on page 122 except that the filtering is done with a multiscale subroutine and the size of the residual must be precomputed externally.

```
module msmis2 { # multi-scale missing data interpolation
  use mshelicon
  use cgstep_mod
  use solver_mod
contains
# fill in missing data by minimizing power out of a given filter.
# by helix magic works in any number of dimensions
  subroutine mis1( niter, xx, aa, lag, jump, known) {
    integer,                intent( in)     :: niter
```

```
    logical, dimension( :), intent( in)     :: known
    real,    dimension( :), intent( in out) :: xx
    real,    dimension( :), pointer         :: aa
    integer, dimension( :), pointer         :: lag, jump
    logical, dimension( :), allocatable     :: mm
    real,    dimension( :), allocatable     :: yy, dd
    integer                                 :: nx, ny, pad, ns
    nx = size( xx); ns = size( jump)
    pad = maxval( lag)*maxval( jump); ny = nx + 2*pad
    allocate( yy( ny), mm( ny), dd( ny*ns)); dd = 0.
    yy( :2*pad) = 0.;      yy( 2*pad+1:) = xx
    mm( :2*pad) = .false.; mm( 2*pad+1:) = known
    call mshelicon_init( ny, ns, aa, lag, jump)
    call solver( mshelicon_lop, cgstep, niter= niter, x = yy, dat = dd,
                 known = mm, x0 = yy)
    call cgstep_close( )
    where( xx == 0.)  xx = yy( 2*pad+1:)
    deallocate( yy, mm, dd)
  }
}
```

8.3 References

Canales, L.L., 1984, Random noise reduction: 54th Ann. Internat. Mtg., Soc. Explor. Geophys., Expanded Abstracts, 525-527.

Rothman, D., 1985, Nonlinear inversion, statistical mechanics, and residual statics estimation: Geophysics, **50**, 2784-2798

Spitz, S., 1991, Seismic trace interpolation in the F-X domain: Geophysics, **56**, 785-794.

Index