

# Homework 5

*Thomas Bayes*

## ABSTRACT

This homework has three parts.

1. Theoretical questions related to Bayesian inversion.
2. Missing data interpolation for ocean floor topography.
3. Extracting a channel structure from a seismic horizon.

## BAYESIAN INVERSION

Assuming the model  $\mathbf{m}$  has the probability distribution function

$$P(\mathbf{m}) = \frac{1}{\sqrt{\pi \det[\mathbf{C}_m]}} e^{-(\mathbf{m}-\mathbf{m}_0)^T \mathbf{C}_m^{-1} (\mathbf{m}-\mathbf{m}_0)}, \quad (1)$$

where  $\mathbf{C}_m$  is the model covariance matrix, and the conditional probability of data  $\mathbf{d}$  (given model  $\mathbf{m}$ ) is

$$P(\mathbf{d}|\mathbf{m}) = \frac{1}{\sqrt{\pi \det[\mathbf{C}_n]}} e^{-(\mathbf{d}-\mathbf{F}\mathbf{m})^T \mathbf{C}_n^{-1} (\mathbf{d}-\mathbf{F}\mathbf{m})}, \quad (2)$$

where  $\mathbf{C}_n$  is the noise covariance matrix, Bayesian inversion suggests the model estimate (given data  $\mathbf{d}$ ) of the form

$$\mathbf{m}_* = \mathbf{m}_0 + \mathbf{C}_m \mathbf{F}^T (\mathbf{F} \mathbf{C}_m \mathbf{F}^T + \mathbf{C}_n)^{-1} (\mathbf{d} - \mathbf{F} \mathbf{m}_0) \quad (3)$$

1. Find the *bias* in estimate (3), i.e. for the true model  $\mathbf{m} = \mathbf{m}_1$ , find the mathematical expectation of  $\mathbf{m}_* - \mathbf{m}_1$ .
2. Find the covariance of estimate (3) (the mathematical expectation of  $\mathbf{m}_* \mathbf{m}_*^T$ ).

Shaping regularization applied in the data space suggests an estimate

$$\mathbf{m}_* = \mathbf{m}_0 + \mathbf{B} [\mathbf{I} + \mathbf{S}_d (\mathbf{F} \mathbf{B} - \mathbf{I})]^{-1} \mathbf{S}_d (\mathbf{d} - \mathbf{F} \mathbf{m}_0) \quad (4)$$

where  $\mathbf{B}$  is the “backward” operator,  $\mathbf{S}_d$  is the “data shaping” operator, and  $\mathbf{I}$  is the identity operator.

1. Find the connection between  $\mathbf{B}$ ,  $\mathbf{S}_d$ ,  $\mathbf{F}$ ,  $\mathbf{C}_m$ , and  $\mathbf{C}_n$  that makes equations (3) and (4) equivalent.
2. Suppose that the data vector  $\mathbf{d}$  consists of two signal components  $\mathbf{d}_1$  and  $\mathbf{d}_2$ , which are uncorrelated and have zero mean and covariances  $\mathbf{C}_1$  and  $\mathbf{C}_2$  respectively. Use Bayesian inversion to formulate a separation of  $\mathbf{d}$  into  $\mathbf{d}_1$  and  $\mathbf{d}_2$ .

## MISSING OCEAN FLOOR DATA INTERPOLATION

SeaBeam is an apparatus for measuring water depth both directly under a boat and somewhat off to the sides of the boat's track. In this part of the assignment, we will use a benchmark dataset from Claerbout (2008): SeaBeam data from a single day of acquisition. The original data are shown in Figure 1a.

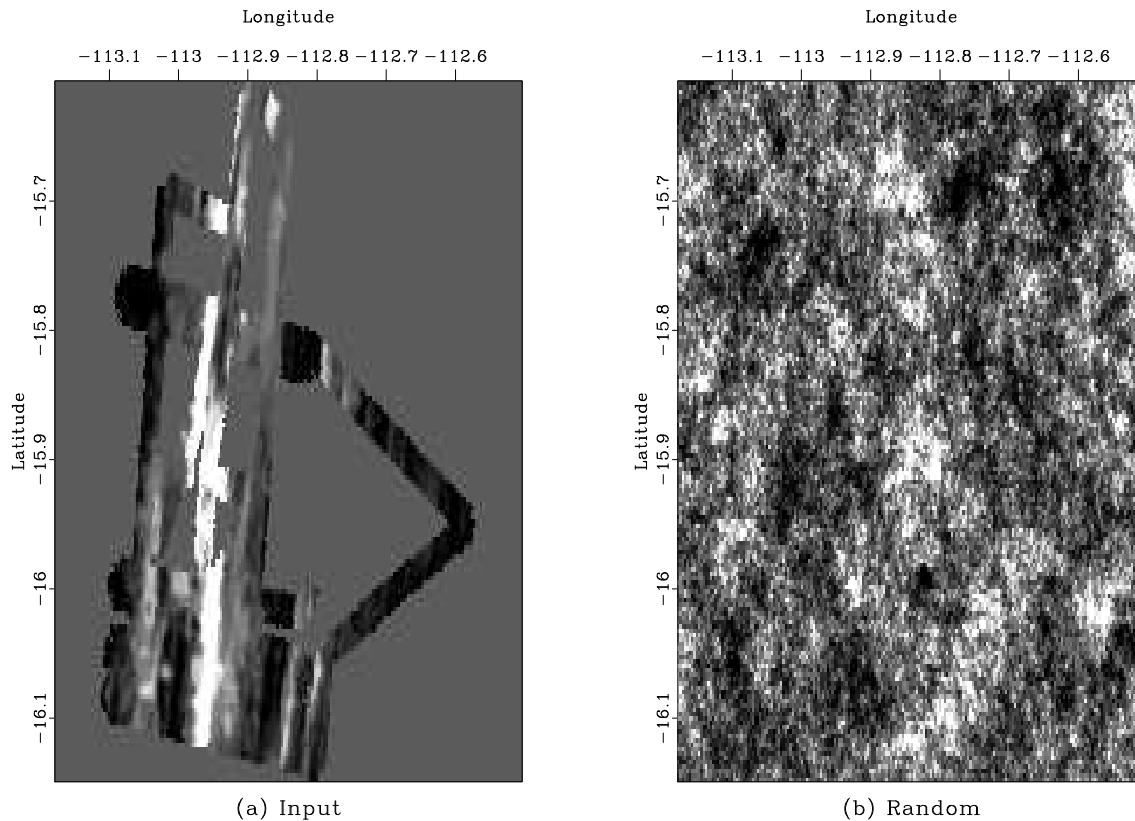


Figure 1: (a) Water depth measurements from one day of SeaBeam acquisition. (b) Random initial model with covariance specified by the inverse Laplacian filter.

To find missing data, we will use estimate (3), where  $\mathbf{F}$  is the mask operator (a diagonal matrix with ones and zeros on the diagonal using ones to mask the known data locations),  $\mathbf{C}_m$  is a stationary filter, and  $\mathbf{C}_n$  is close to zero,

Our first choice for  $\mathbf{C}_m$  is the inverse of a five-point Laplacian filter

$$L_2(Z_1, Z_2) = 4 - Z_1 - 1/Z_1 - Z_2 - 1/Z_2 \quad (5)$$

To build the inverse, we put the Laplacian filter on a helix, where it takes the form

$$L_1(Z) = 4 - Z - 1/Z - Z^{N_1} - 1/Z^{N_1} \quad (6)$$

and factor it into two parts  $L_1(Z) = D(Z) D(1/Z)$  using the Wilson-Burg algorithm (Fomel et al., 2003). The factorization is tested in Figure 2, where the impulse response of the Laplacian filter gets inverted by recursive filtering (polynomial division) on a helix.

The Laplacian filter is a common a priori choice for enforcing smoothness in the estimated model. Figure 1b shows a random model created by dividing random normally distributed noise by  $D(Z)$ . We will use it as an initial model for the missing data reconstruction problem.

There are two alternative formulations of Bayesian least-squares inversion:

**Overdetermined least-squares** Minimize the power (least-squares length) of the data misfit  $\hat{\mathbf{d}} - \hat{\mathbf{F}}_m \mathbf{m}$ , where

$$\hat{\mathbf{d}} = \begin{bmatrix} \mathbf{D}_n (\mathbf{d} - \mathbf{F} \mathbf{m}_0) \\ \mathbf{0} \end{bmatrix}, \quad (7)$$

$$\hat{\mathbf{F}}_m = \begin{bmatrix} \mathbf{D}_n \mathbf{F} \\ \mathbf{D}_m \end{bmatrix}, \quad (8)$$

with  $\mathbf{C}_m^{-1} = \mathbf{D}_m^T \mathbf{D}_m$  and  $\mathbf{C}_n^{-1} = \mathbf{D}_n^T \mathbf{D}_n$ . In our case,  $\mathbf{D}_n$  is multiplication by a large number, and  $\mathbf{D}_m$  is convolution with  $D(Z)$  (polynomial *multiplication* on a helix).

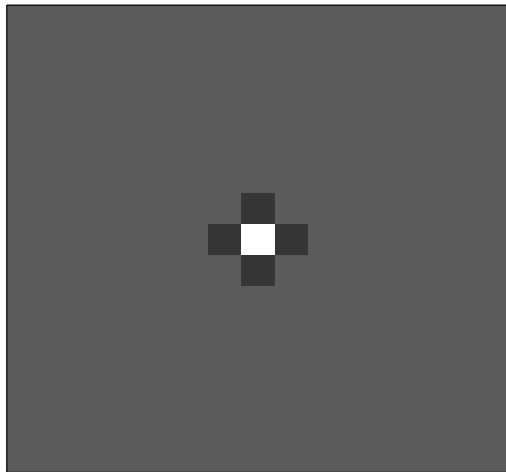
**Underdetermined least-squares** Minimize the power (least-squares length) of the extended model  $\hat{\mathbf{m}}$  provided that  $\mathbf{d} = \mathbf{F} \mathbf{m}_0 + \hat{\mathbf{F}}_d \hat{\mathbf{m}}$ , where

$$\hat{\mathbf{m}} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix}, \quad (9)$$

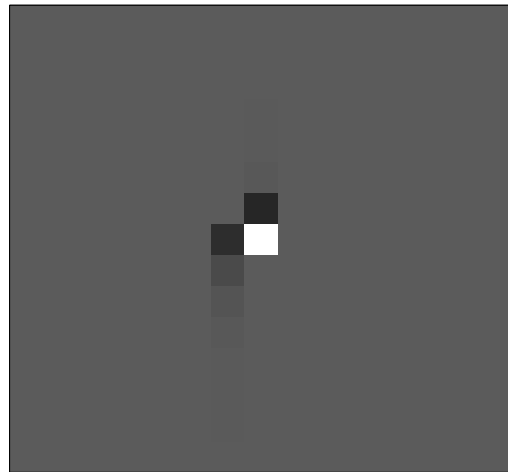
$$\hat{\mathbf{F}}_d = \begin{bmatrix} \mathbf{F} \mathbf{P}_m & \mathbf{P}_n \end{bmatrix} \quad (10)$$

with  $\mathbf{m} = \mathbf{P}_m \mathbf{p}$ ,  $\mathbf{C}_m = \mathbf{P}_m^T \mathbf{P}_m$  and  $\mathbf{C}_n = \mathbf{P}_n^T \mathbf{P}_n$ . In our case,  $\mathbf{P}_n$  is multiplication by a small number, and  $\mathbf{P}_m$  is convolution with  $1/D(Z)$  (polynomial *division* on a helix).

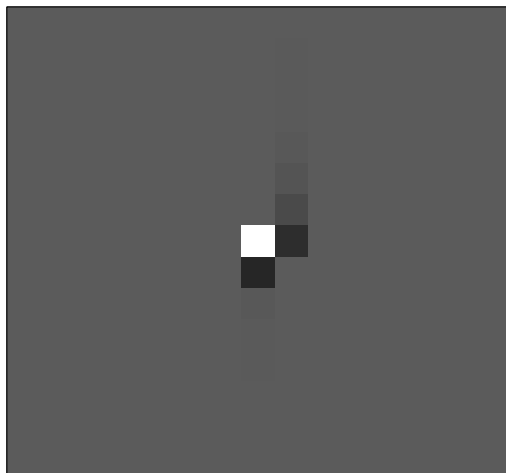
The results of missing data reconstruction from both methods after 10 conjugate-gradient iterations are shown in Figure 3.



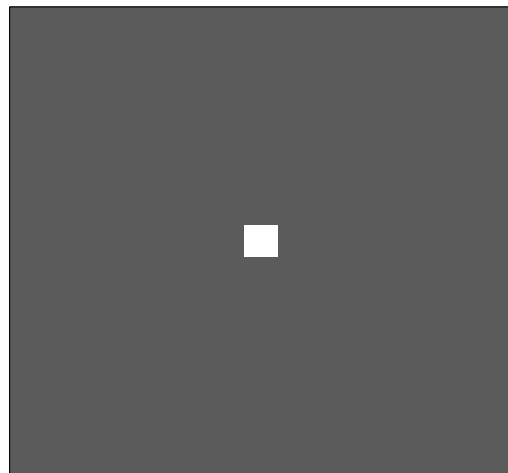
(a) Laplacian



(b) Laplacian/Factor



(c) Laplacian/Factor'



(d) Laplacian/Factor/Factor'

Figure 2: Impulse response of the five-point Laplacian filter (a) gets inverted by recursive filtering (polynomial division) on a helix. (b) Division by  $D(Z)$ . (c) Division by  $D(1/Z)$ . (d) Division by  $D(Z)D(1/Z)$ .

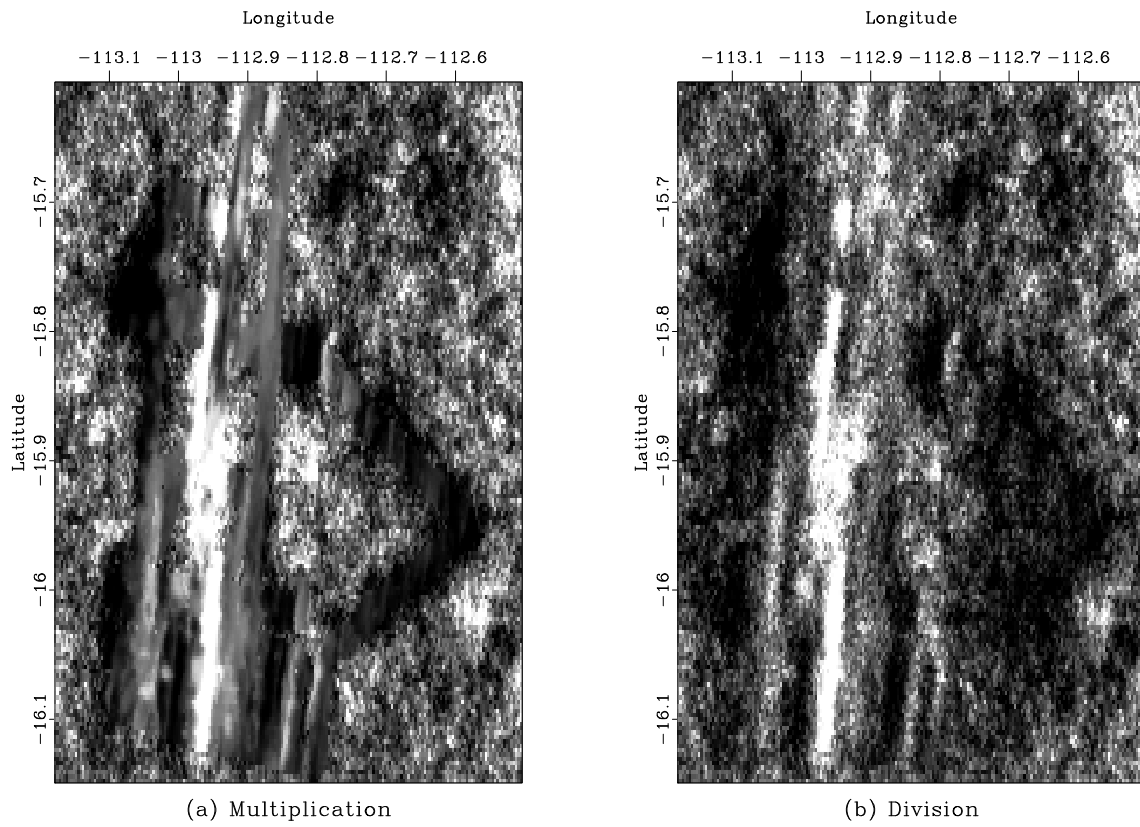


Figure 3: Result of missing data interpolation after 10 iterations using (a) polynomial multiplication and (b) polynomial division on a helix.

Your task:

1. Change directory to `geo391/hw5/seabeam`
2. Run

```
scons view
```

to reproduce the figures on your screen.

3. Modifying the `SConstruct` file to accomplish the following tasks
  - (a) Find out the number of conjugate-gradient iterations, when the “multiplication” and “division” methods produce visually similar results.
  - (b) Find out which of the two methods converges to this solution faster.
  - (c) Replace the five-point Laplacian filter with the more isotropic nine-point filter

$$\hat{L}_2(Z_1, Z_2) = 20 - 4Z_1 - 4/Z_1 - 4Z_2 - 4/Z_2 - Z_1Z_2 - Z_1/Z_2 - Z_2/Z_1 - 1/(Z_1Z_2) \quad (11)$$

and repeat the experiment.

4. **EXTRA CREDIT** for finding and implementing a more appropriate filter for this problem than the Laplacian.

```

1 from rsfproj import *
2
3 # Download data
4 Fetch('seabin.hh', 'seab')
5
6 # Create mesh
7 Flow('mesh', 'seabin.hh', 'dd form=native | pad n1=200 beg1=20')
8
9 # Mask for known values
10 Flow('mask', 'mesh',
11      'add mode=p $SOURCE | mask min=1e-6 | dd type=float')
12
13 def plotdata(title):
14     return '''
15     grey title="%s" transp=n yreverse=n clip=0.65
16     labell=Longitude label2=Latitude
17     ''' % title
18
19 # Plot input data

```

```

20 Plot('mesh',plotdata('(a) Input'))
21
22 # Laplacian filter
23 Flow('lag.asc',None,
24     ' ',
25     echo 1 100 n1=2 n=100,100
26     data_format=ascii_int in=$TARGET
27     ' ')
28 Flow('lag','lag.asc','dd form=native')
29
30 Flow('flt.asc','lag',
31     ' ',
32     echo -1 -1 a0=2 n1=2 lag=$SOURCE
33     data_format=ascii_float in=$TARGET
34     ' ',stdin=0)
35 Flow('flt','flt.asc','dd form=native')
36
37 # Spectral factorization on a helix
38 Flow('lapflt laplag','flt',
39     'wilson eps=1e-3 lagout=${TARGETS[1]}')
40
41 def plotfilt(title):
42     return ' ',
43     grey wantaxis=n title="%s" pclip=100
44     crowd=0.85 screenratio=1
45     ' ' % title
46
47 # Filter impulse response
48 Flow('spike',None,'spike n1=15 n2=15 k1=8 k2=8')
49 Flow('imp0','spike flt','helicon filt=${SOURCES[1]} adj=0')
50 Flow('imp1','spike flt','helicon filt=${SOURCES[1]} adj=1')
51 Flow('imp','imp0 imp1','add ${SOURCES[1]}')
52 Plot('imp',plotfilt('(a) Laplacian'))
53
54 # Test factorization
55 Flow('fac0','imp lapflt',
56     'helicon filt=${SOURCES[1]} adj=0 div=1')
57 Flow('fac1','imp lapflt',
58     'helicon filt=${SOURCES[1]} adj=1 div=1')
59 Plot('fac0',plotfilt('(b) Laplacian/Factor'))
60 Plot('fac1',plotfilt('(c) Laplacian/Factor\'))
61 Flow('fac','fac0 lapflt',
62     'helicon filt=${SOURCES[1]} adj=1 div=1')
63 Plot('fac',plotfilt('(d) Laplacian/Factor/Factor\'))
64

```

```

65 Result('laplace', 'imp fac0 fac1 fac', 'TwoRows',
66         vppen='gridsize=5,5 xsize=11 ysize=11')
67
68 seed = 1030 # MODIFY ME !!!
69
70 # Random initial model
71 Flow('rand', 'mesh lapflt',
72      ', , ,
73      noise rep=y seed=%d var=0.02 |
74      helicon filt=${SOURCES[1]} div=y
75      ', , , % seed)
76 Plot('rand', plotdata('(b) Random'))
77 Result('mesh', 'mesh rand', 'SideBySideAniso')
78
79 niter = 10 # MODIFY ME !!!
80
81 # Missing data interpolation
82 for prec in (0,1):
83     interp = 'interp%d' % prec
84     #  $K[x] \sim K[d-K[m0]]$ ;  $D[x] \sim 0$ ;  $m = m+m0$ 
85     Flow(interp, 'rand mask mesh lapflt',
86          ', , ,
87          add mode=p ${SOURCES[1]} |
88          add scale=-1,1 ${SOURCES[2]} |
89          miss filt=${SOURCES[3]} mask=${SOURCES[1]}
90          niter=%d prec=%d exact=n |
91          add ${SOURCES[0]}
92          ', , , % (niter, prec))
93     title = ('(a) Multiplication', '(b) Division')[prec]
94     Plot(interp, plotdata(title))
95 Result('interp', 'interp0 interp1', 'SideBySideAniso')
96
97 End()

```

## CHANNEL EXTRACTION

In this part of the assignment, we return to the seismic horizon slice from a previous homework. Our task is to extract the geometry of a sand channel, which is the most prominent geological feature in this slice. To perform the extraction, we use an edge detection algorithm from the image processing literature (Canny, 1986). In a nutshell, Canny's edge detector picks areas of high gradient that seem to be aligned along an edge. The original horizon and extracted channel edges are shown in Figure 4.

The initial result is not very clear, because it is affected by random noise (fluctua-

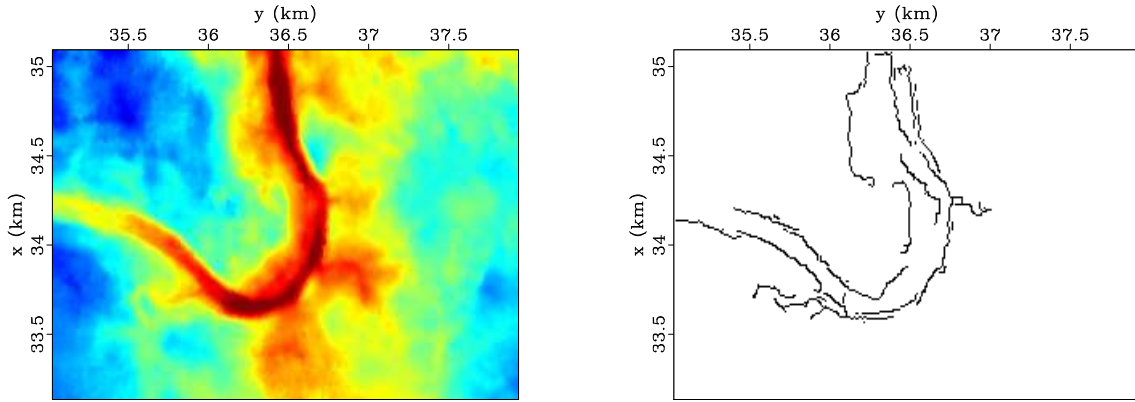


Figure 4: Horizon slice from 3-D seismic (left) and the result of edge detection (right).

tions in seismic amplitude). The general method of removing the noise is smoothing. Figure 5 shows the original horizon after smoothing with a triangle filter. The channel can be extracted more reliably. However, its width is enlarged by smoothing.

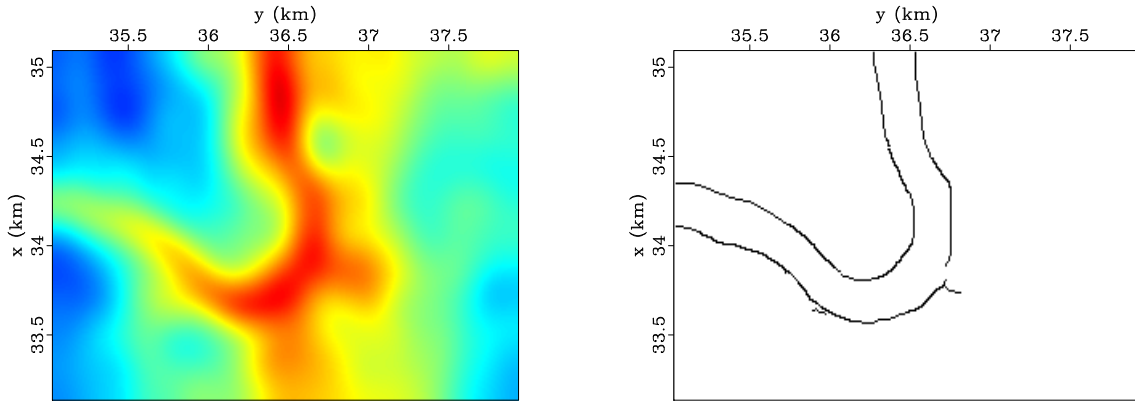


Figure 5: Horizon smoothed by a triangle smoothing filter (left) and the result of edge detection (right).

To preserve the original channel shape while removing random noise, we need a more sophisticated method. Figure 6 shows the result of *anisotropic diffusion* (Weickert, 1998): non-stationary smoothing by solving the nonlinear partial differential equation

$$\frac{\partial U}{\partial t} = \frac{a}{|\nabla U|} \nabla \cdot \left( \frac{\nabla U}{|\nabla U|} \right), \quad (12)$$

where  $U(\mathbf{x}, t)$  is the image evolving in artificial time  $t$ , and  $a$  is a scaling coefficient. One finite-difference step with equation (13) can be understood as a regularized least-squares inversion problem. First, approximate the equation with

$$\frac{\mathbf{U}_k - \mathbf{U}_{k-1}}{\Delta t} = -a \mathbf{D}^T \mathbf{W}^T \mathbf{W} \mathbf{D} \mathbf{U}_k, \quad (13)$$

where  $\mathbf{U}_t$  is the discretized image at time step  $k$ ,  $\mathbf{D}$  approximates the gradient operator, and  $\mathbf{W}$  is a diagonal weight approximating  $1/|\nabla U|$ . Solving for  $\mathbf{U}_k$  produces

$$\mathbf{U}_k = (\mathbf{I} + \epsilon^2 \mathbf{D}^T \mathbf{W}^T \mathbf{W} \mathbf{D})^{-1} \mathbf{U}_{k-1} \quad (14)$$

where  $\epsilon = a \Delta t$ . Equation (14) has the form of equation (3) with  $\mathbf{F} = \mathbf{I}$  and  $\mathbf{C}_m^{-1} = \epsilon^2 \mathbf{D}^T \mathbf{W}^T \mathbf{W} \mathbf{D}$ . In this homework, we are going to implement both stationary and non-stationary diffusion by regularized least-squares inversion with equation (14).

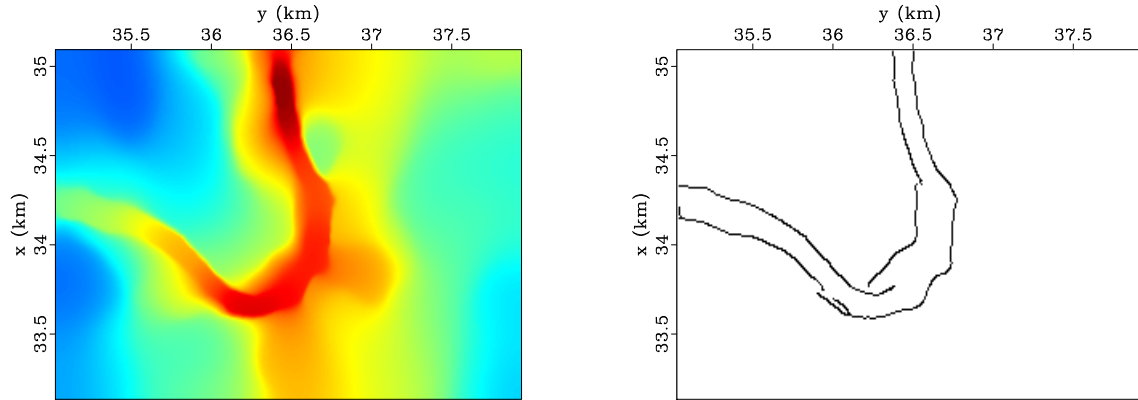


Figure 6: Horizon smoothed by anisotropic diffusion (left) and the result of edge detection (right).

```

19 #include <assert.h>
20
21 #include <rsf.h>
22
23 static int n1, n2, n12;
24
25 static void gradient (bool adj, bool add,
26                     int nx, int ng, float* x, float* g)
27 /*< gradient operator >*/
28 {
29     int i1, i2, i;
30
31     assert(n12 == n1*n2 && nx == n12 && ng == 2*n12);
32
33     sf_adjnull (adj, add, nx, ng, x, g);
34
35     for (i2=1; i2 < n2-1; i2++) {
36         for (i1=1; i1 < n1-1; i1++) {
37             i = i1+i2*n1; /* map 2-D to 1-D */
38
39             if (adj) {
40                 /* !!! ADD CODE !!! */

```

```

41         } else {
42             g[i] += 0.5*(x[i+1] - x[i-1]);
43             g[i+n12] += 0.5*(x[i+n1] - x[i-n1]);
44         }
45     }
46 }
47 }
48
49 int main(int argc, char* argv[])
50 {
51     int niter, i, repeat;
52     float *data, eps, dot1[2], dot2[2];
53     sf_file in, out;
54
55     sf_init(argc, argv);
56     in = sf_input("in");
57     out = sf_output("out");
58
59     /* Get dimensions */
60     if (!sf_histint(in, "n1", &n1)) sf_error("No n1=");
61     if (!sf_histint(in, "n2", &n2)) sf_error("No n2=");
62     n12 = n1*n2;
63
64     /* First let us do the dot-product test */
65     sf_dot_test(gradient, n12, 2*n12, dot1, dot2);
66     sf_warning("Dot product test");
67     sf_warning("*****");
68     sf_warning("Check if %g == %g", dot1[0], dot1[1]);
69     sf_warning("Check if %g == %g", dot2[0], dot2[1]);
70     sf_warning("*****");
71
72     if (!sf_getint("niter", &niter)) niter=10;
73     /* number of conjugate-gradient iterations */
74     if (!sf_getint("repeat", &repeat)) repeat=1;
75     /* number of smoothing iterations */
76     if (!sf_getfloat("eps", &eps)) eps=1.;
77     /* regularization parameter */
78
79     /* Allocate and read data */
80     data = sf_floatalloc(n12);
81     sf_floatread(data, n12, in);
82
83     for (i=0; i < repeat; i++) {
84         /* Call a generic function for solving
85          .    $F[m] = \tilde{d}$ 

```

```

86     eps*D[m] = ~ 0
87 */
88     sf_solver_reg(sf_copy_lop /* F */,
89                 sf_cgstep /* CG step */,
90                 gradient /* D */,
91                 2*n12 /* size of D[m] */,
92                 n12 /* size of m */,
93                 n12 /* size of d */,
94                 data /* m (output) */,
95                 data /* d (input) */,
96                 niter /* # iterations */,
97                 eps /* eps */,
98                 "verb",true,"end");
99     sf_cgstep_close(); /* free internal storage */
100 }
101
102 sf_floatwrite(data,n12,out);
103
104 exit(0);
105 }

```

Your task:

1. Change directory to `geo391/hw5/channel`
2. Run

```
scons view
```

to reproduce the figures on your screen.

3. Modify the `diffuse.c` file to fill in the missing part.
4. Compare the result of smoothing by regularized inversion with other results. Select the values of critical parameters `niter=`, `eps=`, and `repeat=` that produce the best result.
5. Modify the `diffuse.c` program to introduce a weight function in the gradient computation. The weight should scale down gradient values near the edge. Compare your results with other results and select the best values for critical parameters.
6. **EXTRA CREDIT** for finding and implementing a better method for channel extraction.

```

1 from rsfproj import *
2
3 # Download data
4 Fetch('horizon.asc', 'hall')
5
6 # Convert format
7 Flow('horizon', 'horizon.asc',
8     '',
9     echo in=$SOURCE data.format=ascii_float n1=3 n2=57036 |
10    dd form=native | window n1=1 f1=-1 |
11    put
12    n2=291 o2=35.031 d2=0.01 label2=y unit2=km
13    n1=196 o1=33.139 d1=0.01 label1=x unit1=km
14    '')
15
16 # Triangle smoothing
17 Flow('smooth', 'horizon', 'smooth rect1=20 rect2=20')
18
19 # Anisotropic diffusion
20 Flow('anisod', 'horizon', 'impl2 rect1=50 rect2=50 tau=1')
21
22 # Rotate program
23 program = Program('diffuse.c')
24 diffuse = str(program[0])
25
26 # Smoothing by regularized inversion
27 Flow('diffuse', ['horizon', diffuse],
28     './${SOURCES[1]} niter=10 repeat=1 eps=1')
29
30 # Display results
31 for horizon in ('horizon', 'smooth', 'anisod'):
32     Plot(horizon,
33         '',
34         grey color=j bias=65 clip=14 yreverse=n wanttitle=n
35         '')
36     edge = 'edge-' + horizon
37     Flow(edge, horizon, 'canny max=98 | dd type=float')
38     Plot(edge, 'grey allpos=y yreverse=n wanttitle=n')
39     Result(horizon, [horizon, edge], 'SideBySideIso')
40
41 End()

```

## COMPLETING THE ASSIGNMENT

1. Change directory to `geo391/hw5`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Bayes's.
3. Run

```
sftour scons lock
sftour scons -c
```

and

```
scons pdf
```

4. Submit your result (file `paper.pdf`) by printing it out or by e-mail.

## REFERENCES

- Canny, J., 1986, A computational approach to edge detection: *IEEE Trans. Pattern Analysis and Machine Intelligence*, **8**, 679–714.
- Claerbout, J., 2008, Image estimation by example: Environmental soundings image enhancement: Stanford Exploration Project.
- Fomel, S., P. Sava, J. Rickett, and J. F. Claerbout, 2003, The Wilson-Burg method of spectral factorization with application to helical filtering: *Geophysical Prospecting*, **51**, 409–420.
- Weickert, J., 1998, Anisotropic diffusion in image processing: B. G. Teubner Stuttgart.