Homework 4

Magnus Rudolph Hestenes

ABSTRACT

This homework has three parts.

1. Theoretical questions related to conjugate gradients and shaping optimization.
2. Attenuation of surface-wave noise in seismic data using match filtering.
3. Irregular data interpolation contest.

PREREQUISITES

Completing the computational part of this homework assignment requires

- `Madagascar` software environment available from `http://www.ahay.org/`
- LaTeX environment with `SEGTeX` available from `http://www.ahay.org/wiki/SEGTeX`

To do the assignment on your personal computer, you need to install the required environments. Please ask for help if you don't know where to start.

The homework code is available from the `Madagascar` repository by running

```
svn co http://svn.code.sf.net/p/rsf/code/trunk/book/geo391/hw4
```

THEORY

You can either write your answers to theoretical questions on paper or (preferably) edit them in the file `hw4/paper.tex`. Please show all the mathematical derivations that you perform.

1. The conjugate gradient algorithm is applied to iteratively optimizing $\|\mathbf{A}\,\mathbf{x} - \mathbf{b}\|^2$ for $\mathbf{x}$ starting with $\mathbf{x}_0 = \mathbf{0}$. Prove that, after N-th iteration, the solution is given

by $\mathbf{x}_N = \mathbf{F}_N \, \mathbf{b}$, where

$$\mathbf{F}_N = \sum_{n=1}^{N} \frac{\mathbf{s}_n \, \mathbf{s}_n^T}{\mathbf{s}_n^T \, \mathbf{A}^T \, \mathbf{A} \, \mathbf{s}_n} \, \mathbf{A}^T \tag{1}$$

and $\mathbf{s}_n$ is the model step at n-th iteration.

2. If the model shaping operator $\mathbf{S}_m$ admits a symmetric splitting $\mathbf{S}_m = \mathbf{H}_m \, \mathbf{H}_m^T$ with square and invertable $\mathbf{H}_m$, the model shaping equation can be rewritten in a symmetric form

$$\left[\mathbf{I} + \mathbf{S}_m \left(\mathbf{B} \, \mathbf{F} - \mathbf{I}\right)\right]^{-1} \mathbf{S}_m \, \mathbf{B} \, \mathbf{d} = \mathbf{H}_m \, \left[\mathbf{I} + \mathbf{H}_m^T \left(\mathbf{B} \, \mathbf{F} - \mathbf{I}\right) \mathbf{H}_m\right]^{-1} \mathbf{H}_m^T \, \mathbf{B} \, \mathbf{d} \; . \tag{2}$$

   (a) Prove equation (10).

   (b) Assuming a symmetric splitting for the data shaping operator $\mathbf{S}_d = \mathbf{H}_d^T \, \mathbf{H}_d$, find a symmetric form of the data shaping equation

$$\mathbf{B} \, \left[\mathbf{I} + \mathbf{S}_d \left(\mathbf{F} \, \mathbf{B} - \mathbf{I}\right)\right]^{-1} \mathbf{S}_d \, \mathbf{d} = \tag{3}$$

MATCH FILTERING FOR ATTENUATION OF SURFACE SEISMIC WAVES

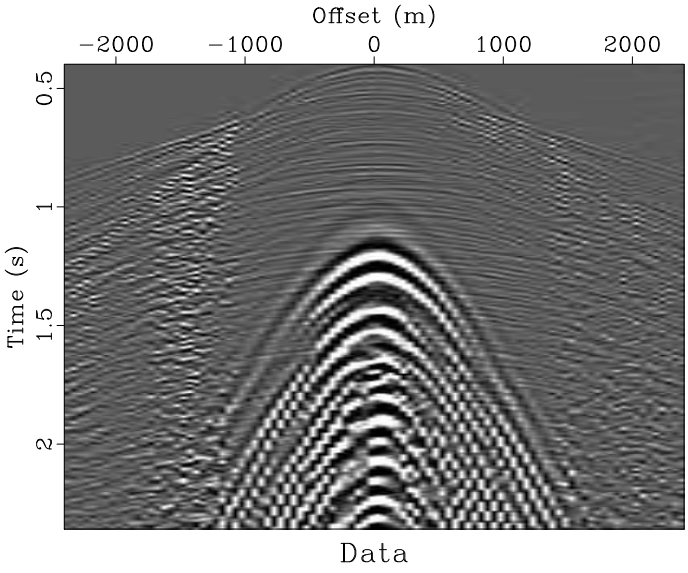


Figure 1: Seismic shot record from sand dunes in the Middle East. The data are contaminated by ground roll propagating in the sand.

Figure 1 shows a section out of a seismic shot record collected over sand dunes in the Middle East. The data are contaminated by ground roll propagating in the sand. A major data analysis task is to separate the signal (reflection waves) from the noise (surface waves).

A quick look at the data spectrum (Figure 2) shows that the noise is mostly concentrated at low frequencies. We can use this fact to create a noise model by low-pass filtering.

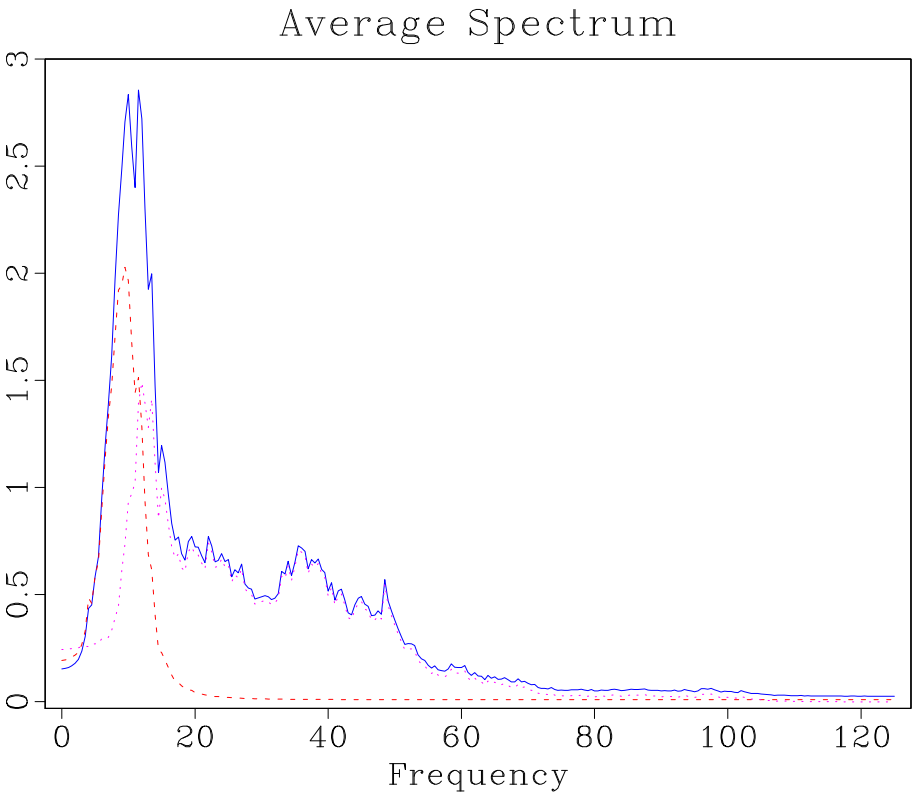


Figure 2: Data spectrum. Solid line – original data. Dashed line – initial noise model and signal model.

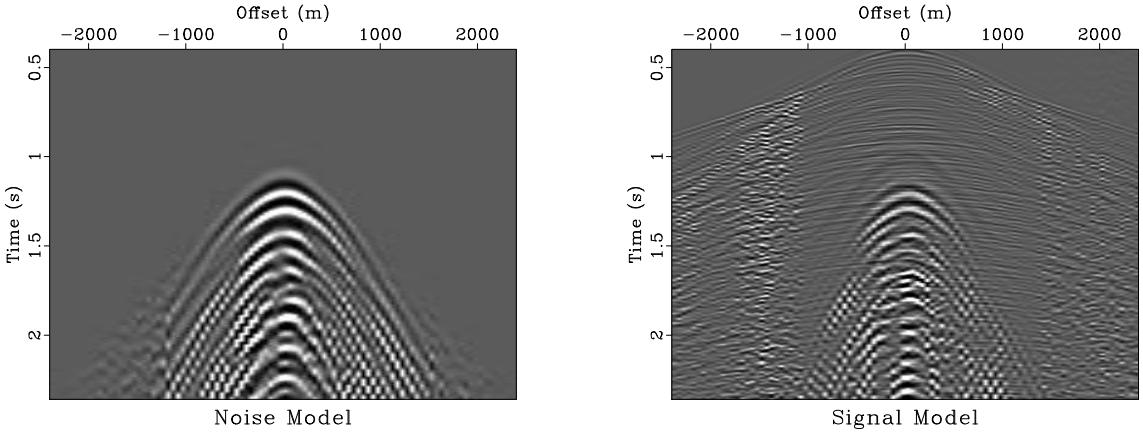


Figure 3: (a) Noise model created by low-pass filtering of the original data. (b) Result of subtraction of the noise model from the data.

Figure 3 shows the noise model from low-pass filtering and inner muting and the result of subtracting this model from the data. Our next task is to match the model to the true noise by solving the least-squares optimization problem

$$\min \|\mathbf{N}\,\mathbf{f} - \mathbf{d}\|^2 \,, \tag{4}$$

where $\mathbf{d}$ is the data, $\mathbf{f}$ is a *matching filter*, and $\mathbf{N}$ represents convolution of the noise model $\mathbf{n}_0$ with the filter. After minimization, $\mathbf{n} = \mathbf{N}\,\mathbf{f}$ becomes the new noise model, and $\mathbf{d} - \mathbf{n}$ becomes the estimated signal. Match filtering is implemented in program `match.c`. Some parts of this program are left out for you to fill.

match/match.c

```
/* Match filtering */
#include <rsf.h>

int main(int argc, char* argv[])
{
    bool adj;
    int n1, n2, i1, i2, i, j, nf;
    float *data, *noiz, *filt;
    sf_file inp, out, oth;

    sf_init(argc,argv);
    inp = sf_input("in");
    out = sf_output("out");
    oth = sf_input("other");

    if (!sf_getbool("adj",&adj)) adj=false;
    /* adjoint flag */

    if (adj) {
        /* input data, output filter */
        if (!sf_histint(inp,"n1",&n1)) sf_error("No n1=");
        if (!sf_histint(inp,"n2",&n2)) sf_error("No n2=");
        if (!sf_getint("nf",&nf)) sf_error("Need nf=");
        /* filter size */

        sf_putint(out,"n1",nf);
        sf_putint(out,"n2",1);
    } else {
        /* input filter, output data */
        if (!sf_histint(inp,"n1",&nf)) sf_error("No n1=");
        if (!sf_histint(oth,"n1",&n1)) sf_error("No n1=");
        if (!sf_histint(oth,"n2",&n2)) sf_error("No n2=");

        sf_fileflush(out,oth);  /* copy data dimensions */
```

```
35      }
36
37      filt = sf_floatalloc(nf);
38      data = sf_floatalloc(n1);
39      noiz = sf_floatalloc(n1);
40
41      if (adj) {
42          for (i=0; i < nf; i++) /* !!! COMPLETE LINE !!! */
43      } else {
44          sf_floatread(filt,nf,inp);
45      }
46
47      for (i2=0; i2 < n2; i2++) {
48          sf_floatread(noiz,n1,oth);
49
50          if (adj) {
51              sf_floatread /* !!! COMPLETE LINE !!! */
52          } else {
53              for (i1=0; i1 < n1; i1++) data[i1] = 0.;
54          }
55
56          for (i=0; i < nf; i++) {
57              for (i1=0; i1 < n1; i1++) {
58                  j=i1-i+nf/2; /* symmetric filter */
59
60                  /* zero value boundary conditions */
61                  if (j < 0 || j >= n1) continue;
62
63                  if (adj) {
64                      filt[i] += /* !!! COMPLETE LINE !!! */
65                  } else {
66                      data[i1] += noiz[j]*filt[i];
67                  }
68              }
69          }
70
71          if (!adj) sf_floatwrite(data,n1,out);
72      }
73
74      if (adj) sf_floatwrite  /* !!! COMPLETE LINE !!! */
75
76      exit(0);
77  }
```

Your task:

1. Change directory to `hw4/match`
2. Run

   ```
   scons view
   ```

   to reproduce the figures on your screen.
3. Modify the `match.c` file to fill in missing parts.
4. Test your modifications by running the dot product test.

   ```
   scons dot.test
   ```

   Repeating this several times, make sure that the numbers in the test match.
5. Modify the `SConstruct` file to display the results of match filtering and include them in your assignment. Try improving the results by finding better parameters.
6. **EXTRA CREDIT** The seismic record in the example above is a section from 3-D data record (shot gather). Try using match filtering to predict one 2-D section out of 3-D data from another section. You may need to modify the program to make the filter two-dimensional.

match/SConstruct

```
from rsf.proj import *

# Critical parameters
##############################
cut = 12 # cutoff frequency
nf  = 11 # filter length
##############################

# Download data
Fetch('dune3D.H','mideast')

# Plotting macro
def grey(title):
    return '''
    window n1=490 |
    grey clip=2.5 title="%s"
    label1=Time unit1=s label2=Offset unit2=m
    ''' % title
```

```
19
20 # Select one 2-D slice
21 Flow('data','dune3D.H',
22       '''
23       dd form=native |
24       window n3=1 f3=2 n1=500 f1=100 |
25       scale dscale=100
26       ''')
27 Result('data',grey('Data'))
28
29 # Create noise model by low-pass filtering
30 Flow('noise0','data',
31       '''
32       bandpass fhi=%g |
33       mutter half=n v0=1500 t0=0.8 hyper=y tp=0.12 |
34       cut n1=90
35       ''' % cut)
36 Plot('noise0',grey('Noise Model'))
37
38 # Signal = Data - Noise
39 Flow('signal0','data noise0','add scale=1,-1 ${SOURCES[1]}')
40 Plot('signal0',grey('Signal Model'))
41
42 Result('noise0','noise0 signal0','SideBySideIso')
43
44 # Plot spectrum
45 Plot('spec','data',
46       '''
47       spectra all=y |
48       graph title="Average Spectrum" max2=3 label2=
49       ''')
50 Plot('nspec0','noise0',
51       '''
52       spectra all=y |
53       graph wanttitle=n wantaxis=n max2=3 plotcol=5 dash=1
54       ''')
55 Plot('sspec0','signal0',
56       '''
57       spectra all=y |
58       graph wanttitle=n wantaxis=n max2=3 plotcol=4 dash=2
59       ''')
60 Result('spec0','spec nspec0 sspec0','Overlay')
61
62 # Matching filter program
63 match = Program('match.c')[0]
```

```
64
65 # Dot product test
66 Flow('filt0',None,'spike n1=%d' % nf)
67 Flow('dot.test','%s data noise0 filt0' % match,
68       '''
69       dottest ./${SOURCES[0]} nf=%d
70       dat=${SOURCES[1]} other=${SOURCES[2]}
71       mod=${SOURCES[3]}
72       ''' % nf,stdin=0,stdout=-1)
73
74 # Conjugate-gradient optimization
75 Flow('filt','data %s noise0 filt0' % match,
76       '''
77       conjgrad ./${SOURCES[1]} nf=%d niter=%d
78       other=${SOURCES[2]} mod=${SOURCES[3]}
79       ''' % (nf,2*nf))
80
81 # Extract new noise and signal
82 Flow('noise','filt %s noise0' % match,
83       './${SOURCES[1]} other=${SOURCES[2]}')
84 Flow('signal','data noise','add scale=1,-1 ${SOURCES[1]}')
85
86 End()
```

SPATIAL INTERPOLATION CONTEST

In 1997, the European Communities organized a Spatial Interpolation Comparison. Many different organizations participated with the results published in a special issue of the *Journal of Geographic Information and Decision Analysis* (Dubois, 1999) and a separate report (Dubois et al., 2003).

The comparison used a dataset from rainfall measurements in Switzerland on the 8th of May 1986, the day of the Chernobyl disaster. Figure 4 shows the data area: the Digital Elevation Model of Switzerland with superimposed country's borders. A total of 467 rainfall measurements were taken that day. A randomly selected subset of 100 measurements was used as the input data the 1997 Spatial Interpolation Comparison in order to interpolate other measurements using different techniques and to compare the results with the known data. Figure 5 shows the spatial locations of the selected data samples and the full dataset.

In this assignment, you will try different techniques of spatial data interpolation and will participate in the interpolation contest.

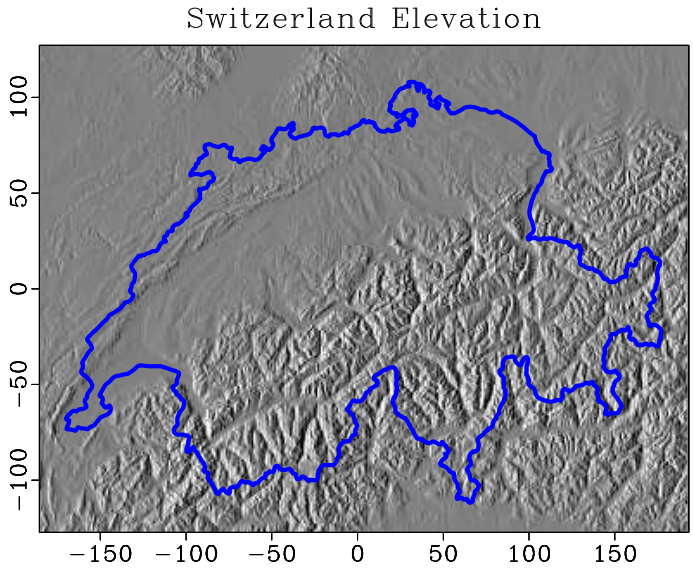


Figure 4: Digital elevation map of Switzerland.

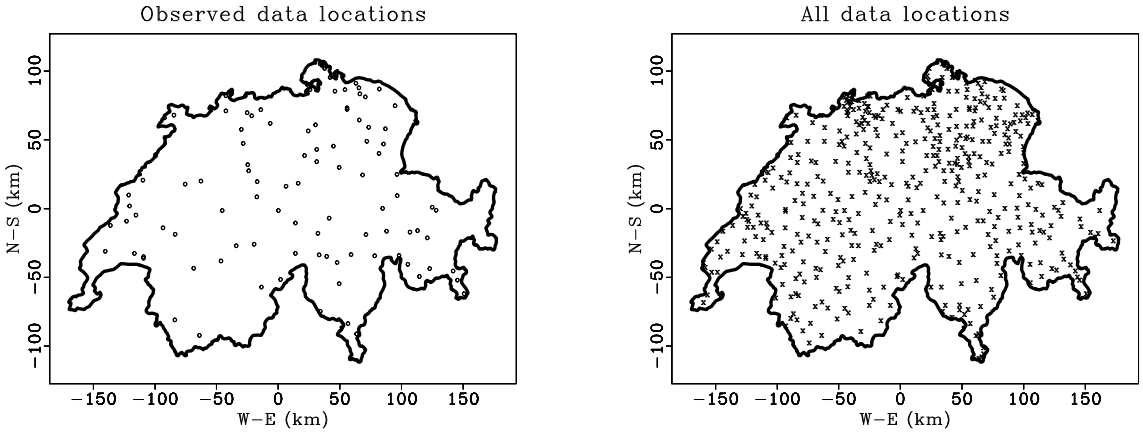


Figure 5: Left: locations of weather stations used as input data in the spatial interpolation contest. Right: all weather stations locations.

Delaunay triangulation

The first technique we are going to try is Delaunay triangulation with linear interpolation of rainfall values inside each triangle. The result is shown in Figure 6a. Does it succeed in hiding the acquisition footprint? Figure 6b provides a comparison between interpolated and known data values. It also indicates the value of the correlation coefficient.

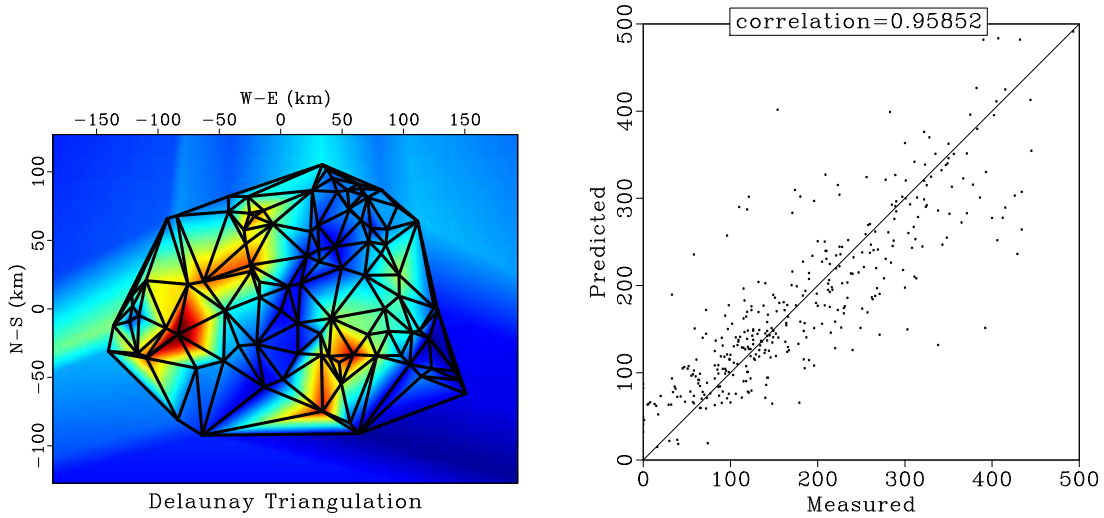


Figure 6: (a) Rainfall data interpolated using Delaunay triangulation. (b) Correlation between interpolated and true data values.

Gradient regularization

An alternative technique is a solution of the regularized least-squares optimization problem

$$\min \left(|\mathbf{F}\,\mathbf{m} - \mathbf{d}|^2 + \epsilon^2 |\mathbf{R}\,\mathbf{m}|^2 \right) \ , \tag{5}$$

where $\mathbf{d}$ is irregular data, $\mathbf{m}$ is model estimated on a regular grid, $\mathbf{F}$ is forward interpolation from the regular grid to irregular locations, ϵ is a scaling parameter, and $\mathbf{R}$ is the regularization operator related to the inverse of the assumed model covariance. In our experiment, $\mathbf{R}$ is the finite-difference gradient filter.

Figure 7 shows the interpolation result after 10 and 100 iterations. 100 iterations are not enough to converge to an acceptable solution, which is evident from the correlation analysis in Figure 8.

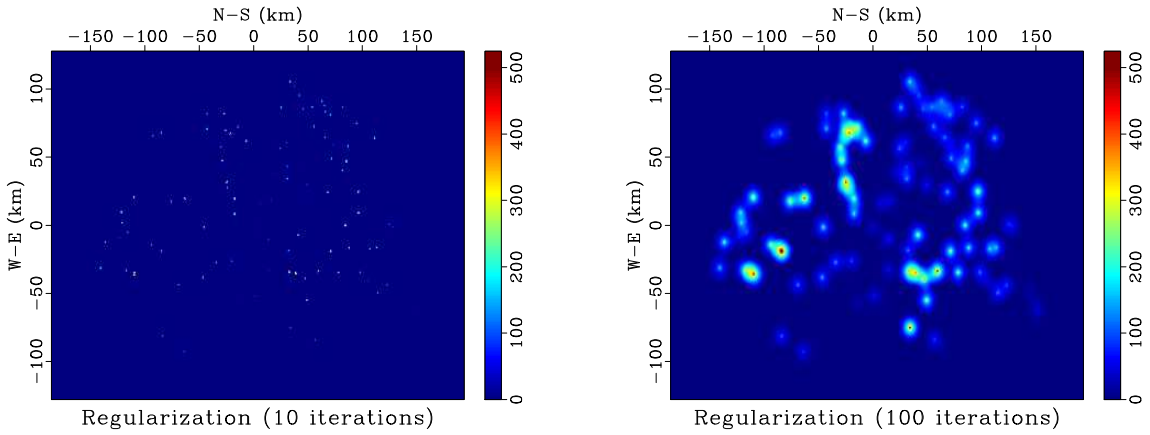


Figure 7: Rainfall data interpolated using regularization with the gradient filter.

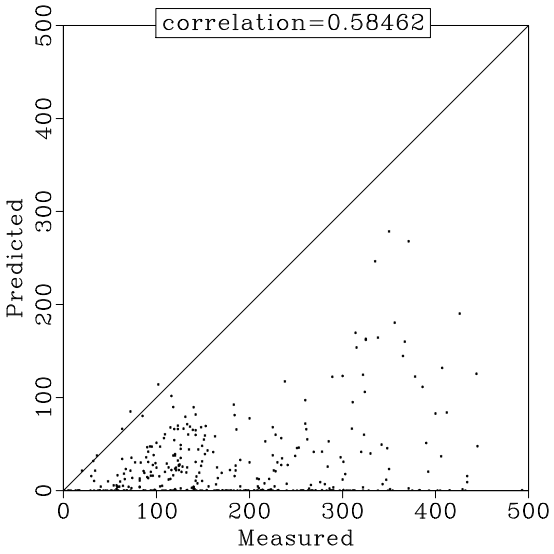


Figure 8: Correlation between interpolated and true data values for regularization with 100 iterations.

Helical derivative preconditioning

An alternative to the optimization problem (5) is the problem of minimizing $|\mathbf{x}|^2+|\mathbf{r}|^2$ under the constraint

$$\mathbf{F}\,\mathbf{P}\,\mathbf{x} + \epsilon\,\mathbf{r} = \mathbf{d}\ . \tag{6}$$

The model $\mathbf{m}$ is defined by $\mathbf{m} = \mathbf{P}\,\mathbf{x}$, and the *preconditioning* operator $\mathbf{P}$ is related to the regularization operator $\mathbf{R}$ according to

$$\mathbf{P}\,\mathbf{P}^T = \left(\mathbf{R}^T\,\mathbf{R}\right)^{-1}\ . \tag{7}$$

The autocorrelation of the gradient filter $\mathbf{R}^T\,\mathbf{R}$ is the Laplacian filter, which can be represented as a five-point polynomial

$$L_2(Z_1, Z_2) = 4 - Z_1 - Z_1^{-1} - Z_2 - Z_2^{-1}\ . \tag{8}$$

To invert the Laplacian filter, we can put on a helix, where it takes the form

$$L_H(Z) = 4 - Z - Z^{-1} - Z^{N_1} - Z^{-N_1}\ , \tag{9}$$

and factor it into two minimum-phase parts $L_H(Z) = D(Z)\,D(1/Z)$ using the Wilson-Burg algorithm (**?**). The factorization is tested in Figure 9, where the impulse response of the Laplacian filter gets inverted by recursive filtering (polynomial division) on a helix.

Figure 10 shows the interpolation result using conjugate-gradient optimization with equation (6) after 10 and 100 iterations. The corresponding correlation analysis is shown in Figure 11.

Shaping regularization

For the shaping regularization approach, we are going to try the simple iteration

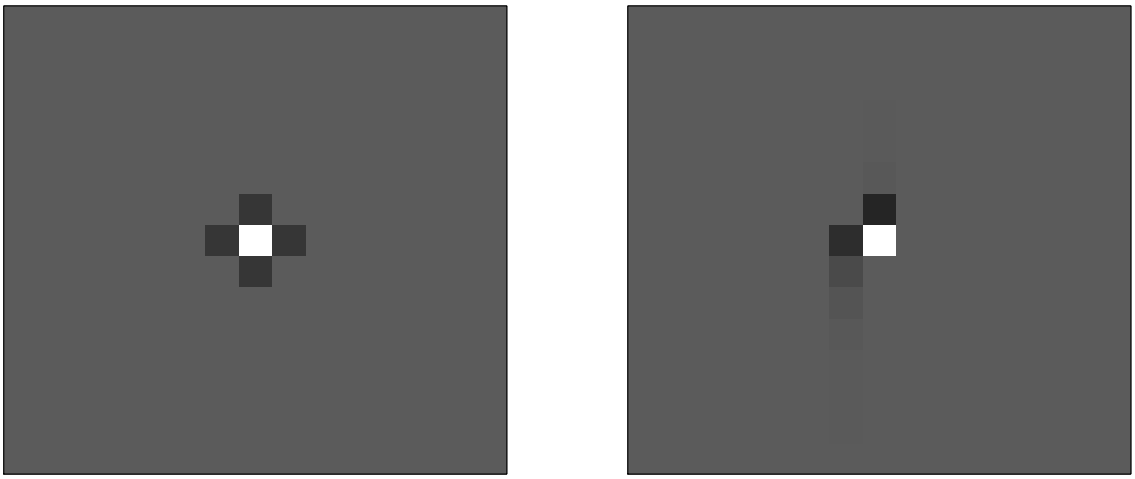
$$\mathbf{m}_{n+1} = \mathbf{S}_m\left[\mathbf{m}_n + \tilde{\mathbf{m}} - \mathbf{B}\,\mathbf{F}\,\mathbf{m}_n\right]\ , \tag{10}$$

where the forward operator $\mathbf{F}$ is bilinear interpolation, the backward operator $\mathbf{B}$ is interpolation by Delaunay triangulation, and the model shaping operator $\mathbf{S}_m$ is triangle smoothing.

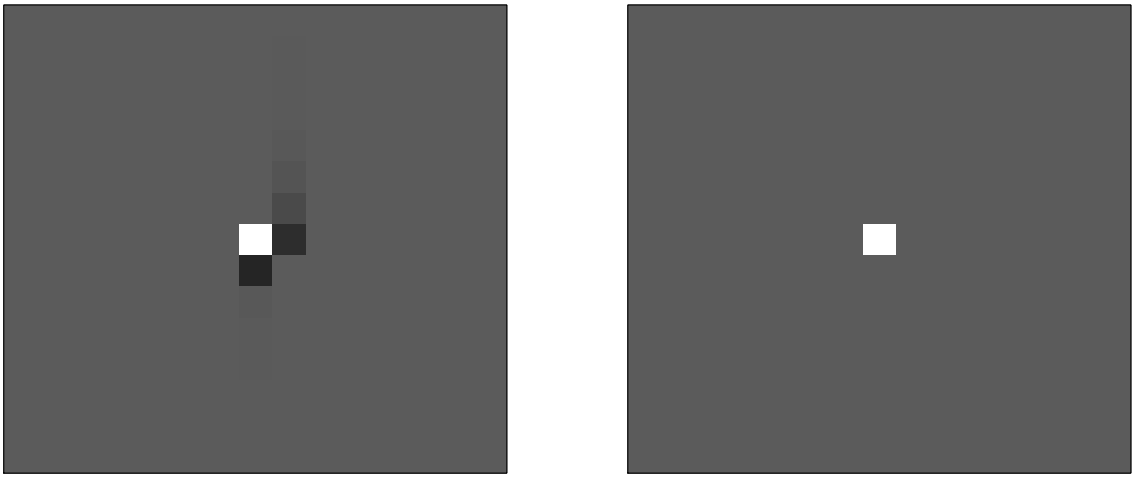
Figure 12 shows the interpolation result using 10 shaping iterations. The corresponding correlation analysis is shown in Figure 13.

Your task:

1. Change directory to `hw4/rain`
2. Run



(a) Laplacian (b) Laplacian/Factor



(c) Laplacian/Factor' (d) Laplacian/Factor/Factor'

Figure 9: 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 $D(Z)\,D(1/Z)$.

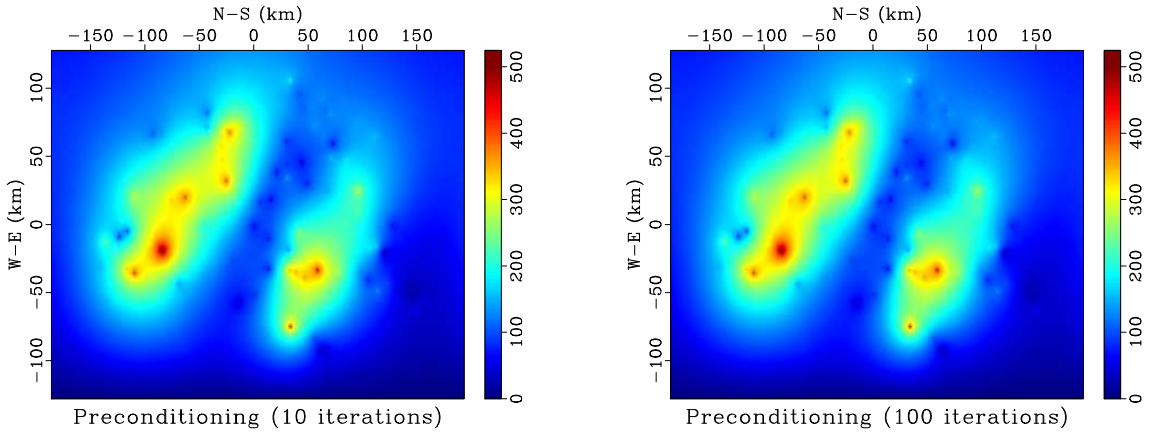


Figure 10: Rainfall data interpolated using preconditioning with the inverse helical filter.

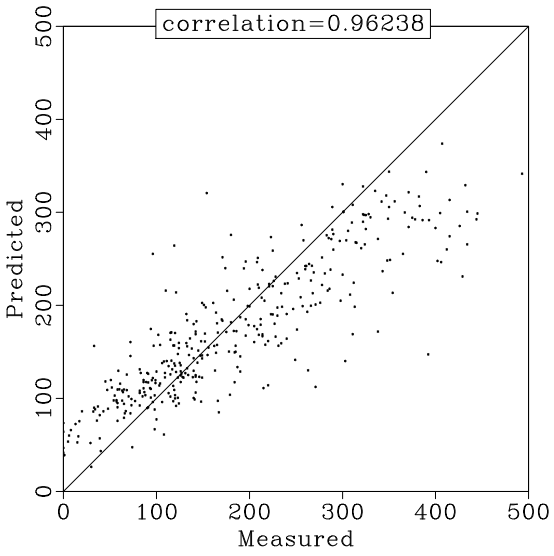


Figure 11: Correlation between interpolated and true data values for preconditioning with 100 iterations.

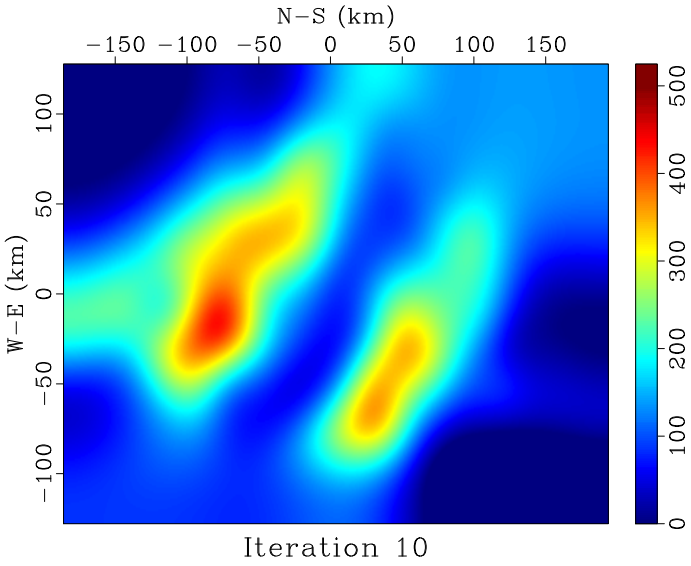


Figure 12: Rainfall data interpolated using shaping regularization.

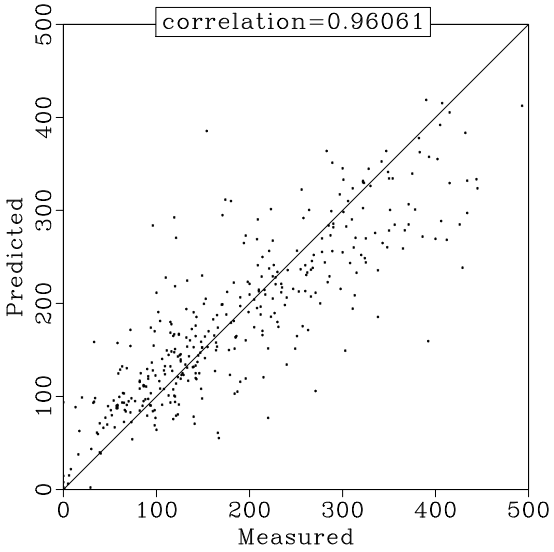


Figure 13: Correlation between interpolated and true data values for shaping regularization with 10 iterations.

```
scons view
```

to reproduce the figures on your screen.

3. Modify the `SConstruct` file to accomplish the following tasks:

   (a) Find out the number of conjugate-gradient iterations needed for the gradient regularization method to achieve a result comparable with the preconditioning method.

   (b) Find out the number of iterations (10) needed for the shaping regularization method to achieve a result comparable with the preconditioning method.

4. What can you conclude about the four methods used in this comparison?

5. **EXTRA CREDIT** Participate in the Spatial Interpolation Contest. Find and implement a method that would provide a better interpolation of the missing values than either of the methods we tried so far. You can change any of the parameters in the existing methods or write your own program but you can use only the 100 original data points as input.

rain/invint.c

```
1  /* Data regulatization by inverse interpolation. */
2  #include <rsf.h>
3
4  static void lint (float x, int n, float* w)
5  /*< linear interpolation>*/
6  {
7      w[0] = 1.0f - x;
```

```
8       w[1] = x;
9   }
10
11  static void regrid( int dim          /* dimensions */,
12                      const int* nold  /* old size [dim] */,
13                      const int* nnew  /* new size [dim] */,
14                      sf_filter aa     /* filter */)
15  /*< change data size >*/
16  {
17      int i, h0, h1, h, ii[SF_MAX_DIM];
18
19      for (i=0; i < dim; i++) {
20          ii[i] = nold[i]/2-1;
21      }
22
23      h0 = sf_cart2line( dim, nold, ii);
24      h1 = sf_cart2line( dim, nnew, ii);
25      for (i=0; i < aa->nh; i++) {
26          h = aa->lag[i] + h0;
27          sf_line2cart( dim, nold, h, ii);
28          aa->lag[i] = sf_cart2line( dim, nnew, ii) - h1;
29      }
30  }
31
32  int main (int argc, char* argv[])
33  {
34      int id, nd, nm, nx, ny, na, ia, niter, three, n[2], m[2];
35      float *mm, *dd, **xy;
36      float x0, y0, dx, dy, a0, eps;
37      char *lagfile;
38      bool prec;
39      sf_filter aa;
40      sf_file in, out, flt, lag;
41
42      sf_init (argc,argv);
43      in = sf_input("in");
44      out = sf_output("out");
45
46      /* read data */
47
48      if (SF_FLOAT != sf_gettype(in)) sf_error("Need float");
49      if (!sf_histint(in,"n1",&three) || 3 != three)
50          sf_error("Need n1=3 in in");
51      if (!sf_histint(in,"n2",&nd)) sf_error("Need n2=");
52
```

```
53      xy = sf_floatalloc2(3,nd);
54      sf_floatread(xy[0],3*nd,in);
55
56      dd = sf_floatalloc(nd);
57      for (id=0; id < nd; id++) dd[id] = xy[id][2];
58
59      /* create model */
60
61      if (!sf_getint ("nx",&nx)) sf_error("Need nx=");
62      if (!sf_getint ("ny",&ny)) sf_error("Need ny=");
63      /* Number of bins */
64
65      sf_putint(out,"n1",nx);
66      sf_putint(out,"n2",ny);
67
68      if (!sf_getfloat("x0",&x0)) sf_error("Need x0=");
69      if (!sf_getfloat("y0",&y0)) sf_error("Need y0=");
70      /* grid origin */
71
72      sf_putfloat (out,"o1",x0);
73      sf_putfloat (out,"o2",y0);
74
75      if (!sf_getfloat("dx",&dx)) sf_error("Need dx=");
76      if (!sf_getfloat("dy",&dy)) sf_error("Need dy=");
77      /* grid sampling */
78
79      sf_putfloat (out,"d1",dx);
80      sf_putfloat (out,"d2",dy);
81
82      nm = nx*ny;
83      mm = sf_floatalloc(nm);
84
85      sf_int2_init (xy, x0,y0, dx,dy, nx,ny, lint, 2, nd);
86
87      /* read filter */
88      flt = sf_input("filt");
89
90      if (NULL == (lagfile = sf_histstring(flt,"lag")))
91          sf_error("Need lag= in filt");
92      lag = sf_input(lagfile);
93
94      n[0] = nx;
95      n[1] = ny;
96      if (!sf_histints (lag,"n",m,2)) {
97          m[0] = nx;
```

```
98          m[1] = ny;
99      }
100
101     if (!sf_histint(flt,"n1",&na)) sf_error("No n1= in filt");
102     aa = sf_allocatehelix (na);
103
104     if (!sf_histfloat(flt,"a0",&a0)) a0=1.;
105     sf_floatread (aa->flt,na,flt);
106
107     for( ia=0; ia < na; ia++) {
108         aa->flt[ia] /= a0;
109     }
110
111     sf_intread(aa->lag,na,lag);
112     regrid (2, m, n, aa);
113
114     if (!sf_getbool("prec",&prec)) prec=false;
115     /* if use preconditioning */
116
117     if (!sf_getint("niter",&niter)) niter=20;
118     /* number of iterations */
119
120     if (!sf_getfloat("eps",&eps)) eps=0.01;
121     /* regularization parameter */
122
123     if (prec) {
124         sf_polydiv_init (nm, aa);
125         sf_solver_prec (sf_int2_lop, sf_cgstep,
126                         sf_polydiv_lop,
127                         nm, nm, nd,
128                         mm, dd, niter, eps, "end");
129     } else {
130         sf_igrad2_init (nx, ny);
131         sf_solver_reg (sf_int2_lop, sf_cgstep,
132                        sf_igrad2_lop,
133                        2*nm, nm, nd,
134                        mm, dd, niter, eps, "end");
135     }
136
137     sf_floatwrite (mm,nm,out);
138     exit(0);
139 }
```

rain/SConstruct

```
from rsf.proj import *

# Download data
Fetch(['border.hh','elevation.HH',
       'alldata.hh','obsdata.hh',
       'coord.hh','predict.hh'],'rain')

# Plot limits
box = '''
min1=-185.556 max1=193.18275
min2=-127.262 max2=127.25044
'''

# Switzerland map
##################

# Border
Flow('border','border.hh','dd form=native')

f2 = 0
def border(name,n2):
    global f2
    Flow(name,'border',
         '''
         window n2=%d f2=%d |
         dd type=complex | window
         ''' % (n2,f2))
    Plot(name,'graph wanttitle=n plotcol=6 plotfat=8 ' + box)
    f2 = f2 + n2

border('border1',338)
border('border2',234)
border('border3',717)
Plot('border','border1 border2 border3','Overlay')

# Elevation
Flow('elev','elevation.HH','dd form=native')
Plot('elev',
     '''
     igrad |
     grey title="Switzerland Elevation" transp=n yreverse=n
     wantaxis=n wantlabel=n wheretitle=t wherexlabel=b
     ''')
Result('elev','elev border','Overlay')

```

```
46 Flow('alldata','alldata.hh',
47       'window n1=2 | dd type=complex form=native | window')
48 Plot('alldata',
49       '''
50       graph symbol=x symbolsz=4
51       title="All data locations" plotcol=7
52       ''' + box)
53 Plot('data','alldata border','Overlay')
54
55 Flow('obs','obsdata.hh',
56       'window n1=2 | dd type=complex form=native | window')
57 Plot('obs',
58       '''
59       graph symbol=o symbolsz=4
60       title="Observed data locations" plotcol=5
61       ''' + box)
62 Plot('obsdata','obs border','Overlay')
63
64 Result('raindata','obsdata data','SideBySideIso')
65
66 Flow('coord','coord.hh','dd form=native')
67 Flow('obsdata','obsdata.hh','dd form=native')
68
69 # Triangulation
70 ################
71 Flow('trian edges','obsdata elev',
72       'tri2reg pattern=${SOURCES[1]} edgeout=${TARGETS[1]}')
73 Plot('edges',
74       '''
75       graph plotcol=7 plotfat=8
76       wanttitle=n wantaxis=n
77       ''' + box)
78 Plot('trian',
79       '''
80       grey yreverse=n transp=n allpos=y
81       color=j clip=500 title="Delaunay Triangulation"
82       label1="W-E (km)" label2="N-S (km)"
83       ''' + box)
84 Result('trian','trian edges','Overlay')
85
86 # Laplacian filter
87 ###################
88
89 Flow('lag.asc',None,
90       '''
```

```
91      echo 1 100 n1=2 n=100,100
92      data_format=ascii_int in=$TARGET
93      ''')
94  Flow('lag','lag.asc','dd form=native')
95
96  Flow('flt.asc','lag',
97      '''
98      echo -1 -1 a0=2 n1=2 lag=$SOURCE
99      data_format=ascii_float in=$TARGET
100     ''',stdin=0)
101 Flow('flt','flt.asc','dd form=native')
102
103 # Spectral factorization on a helix
104 Flow('lapflt laplag','flt',
105     'wilson eps=1e-4 lagout=${TARGETS[1]}')
106
107 def plotfilt(title):
108     return '''
109     grey wantaxis=n title="%s" pclip=100
110     crowd=0.85 screenratio=1
111     ''' % title
112
113 # Filter impulse response
114 Flow('spike',None,'spike n1=15 n2=15 k1=8 k2=8')
115 Flow('imp0','spike flt','helicon filt=${SOURCES[1]} adj=0')
116 Flow('imp1','spike flt','helicon filt=${SOURCES[1]} adj=1')
117 Flow('imp','imp0 imp1','add ${SOURCES[1]}')
118 Plot('imp',plotfilt('(a) Laplacian'))
119
120 # Test factorization
121 Flow('fac0','imp lapflt',
122     'helicon filt=${SOURCES[1]} adj=0 div=1')
123 Flow('fac1','imp lapflt',
124     'helicon filt=${SOURCES[1]} adj=1 div=1')
125 Plot('fac0',plotfilt('(b) Laplacian/Factor'))
126 Plot('fac1',plotfilt('(c) Laplacian/Factor\''))
127 Flow('fac','fac0 lapflt',
128     'helicon filt=${SOURCES[1]} adj=1 div=1')
129 Plot('fac',plotfilt('(d) Laplacian/Factor/Factor\''))
130
131 Result('laplace','imp fac0 fac1 fac','TwoRows',
132        vppen='gridsize=5,5 xsize=11 ysize=11')
133
134 # Maximum number of iterations
135 ##############
```

```
136 nmax = 100 # CHANGE ME!!!
137
138 # Inverse interpolation program
139 program = Program('invint.c')
140 invint = str(program[0])
141
142 for prec in range(2):
143     iters = []
144     inter = 'inter%d' % prec
145     for niter in [10,nmax]:
146         it = 'inter%d-%d' % (prec,niter)
147         Flow(it,['obsdata',invint,'lapflt'],
148              '''
149              ./${SOURCES[1]} prec=%d niter=%d
150              filt=${SOURCES[2]}
151              nx=376 ny=253 eps=0.01
152              dx=1.00997 dy=1.00997
153              x0=-185.556 y0=-127.262
154              ''' % (prec,niter))
155         Plot(it,
156              '''
157              grey scalebar=y yreverse=n transp=n allpos=y
158              minval=0 maxval=525 color=j clip=500
159              title="%s (%d iterations)"
160              ''' % (('Regularization',
161                      'Preconditioning')[prec],niter))
162         iters.append(it)
163     Result(inter,iters,'SideBySideIso')
164
165 # Shaping regularization
166 ########################
167 # Forward - bi-linear interpolation
168 # Backward - triangulation
169 # Shaping - triangle smoothing
170
171 # Maximum number of iterations
172 ##############
173 nshape = 10 # CHANGE ME!!!
174
175 m0 = 'trian'
176 m = m0
177
178 # Coordinates of observed data
179 Flow('obscoord','obsdata','window n1=2')
180
```

```
ms = []
for i in range(1,nshape+1):
    mi = 'shaping%d' % i
    Flow(mi,[m,'obscoord',m0],
         '''
         extract head=${SOURCES[1]} xkey=0 ykey=1      |
         transp | cat ${SOURCES[1]} axis=1 order=2,1   |
         tri2reg pattern=${SOURCES[2]}                 |
         add ${SOURCES[0]} ${SOURCES[2]} scale=-1,1,1  |
         smooth rect1=20 rect2=20 repeat=2
         ''')
    Plot(mi,
         '''
         grey scalebar=y yreverse=n transp=n allpos=y
         minval=0 maxval=525 color=j clip=500
         title="Iteration %d"
         ''' % i)
    m = mi
    ms.append(mi)
Plot('ms',ms,'Movie',view=1)
Result('shaping',mi,'Overlay')

# Prediction comparisons
########################

Flow('predict','predict.hh','dd form=native')
Flow('norm','predict',
     'add mode=p $SOURCE | stack axis=1 norm=n')

Plot('line',None,
     '''
     math n1=2 o1=0 d1=500 output=x1 |
     graph plotcol=7 wanttitle=n wantaxis=n
     screenratio=1 min1=0 max1=500 min2=0 max2=500
     ''')

for case in ('trian','shaping%d' % nshape,
             'inter0-%d' % nmax,'inter1-%d' % nmax):
    pred = case+'-pred'
    Flow(pred,[case,'coord'],
         'extract head=${SOURCES[1]} xkey=0 ykey=1')
    Plot(pred,['predict',pred],
         '''
         cmplx ${SOURCES[1]} |
         graph symbol="*" wanttitle=n
```

```
226             screenratio=1 min1=0 max1=500 min2=0 max2=500
227             label1=Measured label2=Predicted
228             ''')
229
230     num = case+'-num'
231     den = case+'-den'
232     cor = case+'-cor'
233
234     Flow(num,['predict',pred],
235             'add mode=p ${SOURCES[1]} | stack axis=1 norm=n')
236     Flow(den,pred,'add mode=p $SOURCE | stack axis=1 norm=n')
237     Flow(cor+'.asc',[num,den,'norm'],
238             '''
239             math a1=${SOURCES[1]} a2=${SOURCES[2]}
240             output="input/sqrt(a1*a2)" |
241             dd form=ascii --out=$TARGET
242             format="label=correlation=%7.5g"
243             ''',stdout=0)
244     Plot(cor,cor+'.asc',
245             'box x0=5.5 y0=9 xt=0 par=$SOURCE',stdin=0)
246
247     Result(pred,[pred,'line',cor],'Overlay')
248
249 End()
```

COMPLETING THE ASSIGNMENT

1. Change directory to `hw4`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Hestenes's.
3. Run

   ```
   sftour scons lock
   ```

   to update all figures.
4. Run

   ```
   sftour scons -c
   ```

   to remove intermediate files.
5. Run

```
scons pdf
```

to create the final document.

6. Submit your result (file `paper.pdf`) on paper or by e-mail.

REFERENCES

Dubois, G., 1999, Spatial interpolation comparison 97: Foreword and introduction: Journal of Geographic Information and Decision Analysis, **2**, 1–10.

Dubois, G., J. Malczewski, and M. D. Cort, eds., 2003, Mapping radioactivity in the environment. Spatial Interpolation Comparison 1997.: Office for Official Publications of the European Communities.