GEO391
Multidimensional Data Analysis in Geosciences
Homework Assignments

Sergey Fomel

Contents

Chapter 1

Homework 1

ABSTRACT

This homework has three parts.

1. Theoretical questions and computations related to digital representation of numbers.
2. Analyzing digital elevation data from the Austin area. You will apply histogram equalization to enhance the image.
3. Analyzing seismic reflection data. You will apply an amplitude gain correction to enhance the image.

1.1 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 https://rsf.svn.sourceforge.net/svnroot/rsf/trunk/book/geo391/hw1
```

1.2 Digital representation of numbers

You can either write your answers to theoretical questions on paper or edit them in the file `hw1/paper.tex`. Please show all the mathematical derivations that you perform.

1. UT's "burnt orange" color is expressed by code `#CC5500`, where each pair of symbols (`CC`, `55`, and `00`) refers to a hexadecimal (base 16) representation of the red, green, and blue components. Convert these numbers to an octal (base 8) and a decimal (base 10) representations.

2. The C program listed below, when compiled and run from the command line, takes a string from the user and prints out the string characters. Modify the program to output ASCII integer codes for each character in the string. What is the ASCII code for the special new line character "`\n`"?

string.c

```c
#include <stdio.h> /* for printf and scanf */

int main(void)
{
    char *s, string[101];

    printf("Input a string:\n");
    scanf("%100s",string);

    /* loop over characters */
    for (s=string; *s != '\0'; s++)
        printf("%c\n",*s);
}
```

3. In the IEEE double-precision floating-point standard, 64 bits (binary digits) are used to represent a real number: 1 bit for the sign, 11 bits for the exponent, and 52 bits for the mantissa. A double-precision normalized non-zero number x can be written in this standard as

$$x = \pm(1.d_1 d_2 \cdots d_{52})_2 \times 2^{n-1023}$$

with $1 \leq n \leq 2046$, and $0 \leq d_k \leq 1$ for $k = 1, 2, \ldots, 52$. What is the largest number that can be expressed in this system?

4. The C program listed below tries to compute the *machine epsilon*: the smallest positive number ϵ such that $1 + \epsilon > 1$ in double-precision floating-point arithmetic.

   (a) Add the missing part of the program so that, when compiled, it runs without an assertion error.

   (b) Modify the program to find the machine epsilon for single-precision floating-point arithmetic.

epsilon.c

```c
#include <assert.h> /* for assert */
#include <float.h>  /* for DBL_EPSILON */

int main(void)
{
    int i;
    double eps, one;

    eps = 1.0;
    for (i=0; i < 100; i++) {
        eps /= 2;
        one = 1.0+eps;

        /* !!! ADD SOMETHING HERE !!! */
    }

    assert(DBL_EPSILON==eps);
}
```

1.3 Histogram equalization

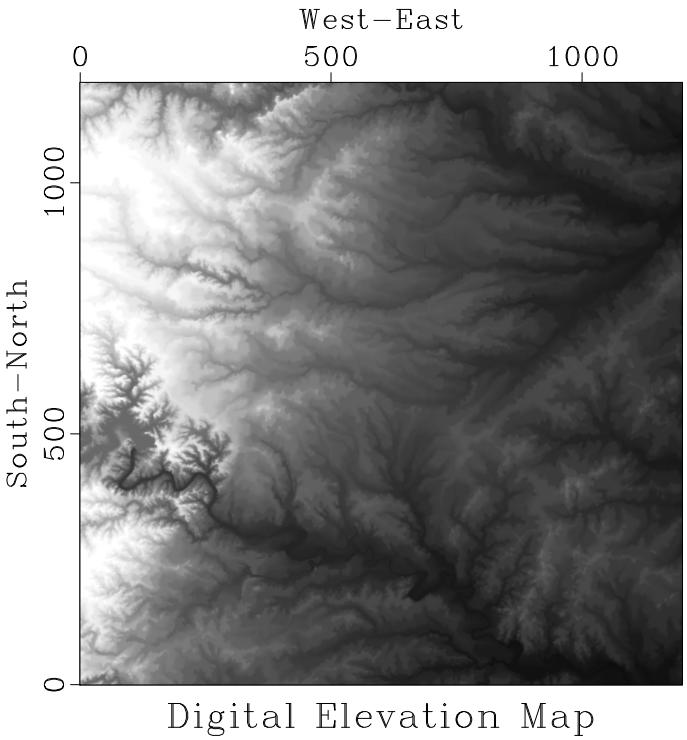


Figure 1.1: Digital elevation map of the West Austin area. hw1/dem byte

Figure 1.1 shows a digital elevation map of the West Austin area. Start by reproducing this figure on your screen.

1. Change directory to `hw1/dem`
2. Run

   ```
   scons byte.view
   ```

3. Examine the file `byte.rsf` which refers to the byte (unsigned character) numbers which get displayed on the screen.
   (a) Open `byte.rsf` with a text editor to check its contents.
   (b) Run

   ```
   sfin byte.rsf
   ```

   to check the data size and format.
   (c) Run

   ```
   sfattr < byte.rsf
   ```

   to check data attributes. What is the maximum and minimum value? What is the mean value? For an explanation of different attributes, run `sfattr` without input.

Each image has a certain distribution of values (a histogram). The histogram for the west Austin elevation map is shown in Figure 3.5. Notice the digitization artifacts. When different values in a histogram are not uniformly distributed, the image can have a low contrast. One way of improving the contrast is *histogram equalization*.

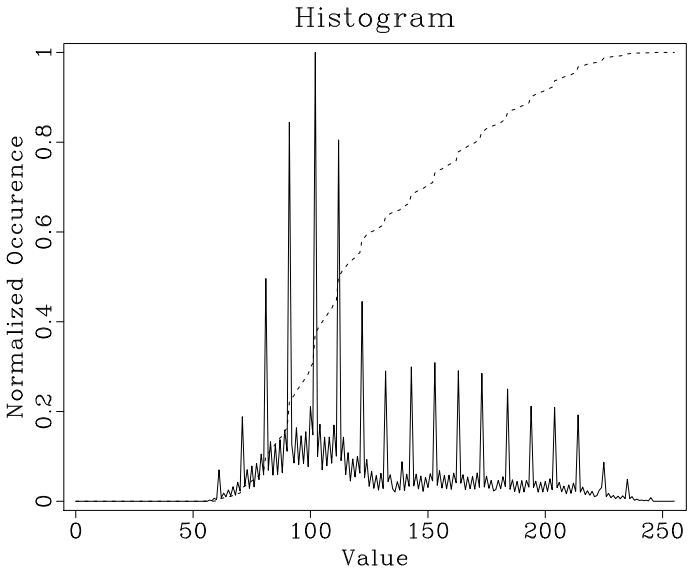


Figure 1.2: Normalized histogram (solid line) and cumulative histogram (dashed line) of the digital elevation data. hw1/dem hist

Let $f(x, y)$ be the original image. The equalized image will be $F(x, y)$. Let $h(f)$ be the histogram (probability distribution) of the original image values. Let $H(F)$ be the histogram of the modified image. The mapping of probabilities suggests

$$H(F)\,dF = h(f)\,df \tag{1.1}$$

or, if we want the modified histogram to be uniform,

$$\frac{dF}{df} = C\, h(f) \tag{1.2}$$

where C is a constant. Solving equation 1.2, we obtain the mapping

$$F(f) = f_0 + C \int_{f_0}^{f} h(\phi)\, d\phi \;, \tag{1.3}$$

where f_0 is the minimum value of f.

The algorithm for histogram equalization consists of the following three steps:

1. Taking an input image $f(x, y)$, compute its histogram $h(f)$.
2. Compute the cumulative histogram $F(f)$ according to equation (1.3). Choose an appropriate C so that the range of F is the same as the range of f.
3. Map every pixel $f(x, y)$ to the corresponding $F(x, y)$.

Your task:

1. Among the `Madagascar` programs, find a program that implements histogram equalization. **Hint:** you may find the `sfdoc` utility useful.
2. Edit the `SConstruct` file to add histogram equalization. Create a new figure and compare it with Figure 1.1.
3. Check the effect of equalization by recomputing the histogram in Figure 3.5 with equalized data. Run

   ```
   scons hist.view
   ```

   to display the figure on your screen.‘
4. **EXTRA CREDIT** for implementing the histogram equalization algorithm in a different computer language.

dem/SConstruct

```
from rsf.proj import *

# Download data
Fetch('austin-w.HH','bay')

# Convert to byte form
Flow('byte','austin-w.HH',
     '''
     dd form=native |
     byte pclip=100 allpos=y
     ''')

# Display
Result('byte',
       '''
       grey yreverse=n screenratio=1
       title="Digital Elevation Map"
       ''')

# Histogram
Flow('hist','byte',
     '''
     dd type=float |
     histogram n1=256 o1=0 d1=1 |
     dd type=float
     ''')
Plot('hist',
     'graph label1=Value label2=Occurence title=Histogram')

# Cumulative histogram
Flow('cumu','hist','causint')

Result('hist','hist cumu',
       '''
       cat axis=2 ${SOURCES[1]} | scale axis=1 |
       graph label1=Value label2="Normalized Occurence"
       title=Histogram dash=0,1
       ''')

# ADD HISTOGRAM EQUALIZATION

End()
```

1.4 Time-power amplitude-gain correction

Raw seismic reflection data come in the form of shot gathers $S(x,t)$, where x is the offset (horizontal distance from the receiver to the source) and t is recording time. Raw data are inconvenient for analysis because of rapid amplitude decay of seismic waves. The decay can be compensated by multiplying the data by a gain function. A commonly used function is a power of time. The gain-compensated gather is

$$S_\alpha(x,t) = t^\alpha \, S(x,t) \; . \tag{1.4}$$

The advantage of the time-power gain is its simplicity and the ability to reverse it by multiplying the data by $t^{-\alpha}$. What value of α should one use? Claerbout (1985) argues in favor of $\alpha = 2$: one factor of t comes from geometrical spreading and the other from scattering attenuation. Your task is to develop an algorithm for finding a better value of α for a given dataset.

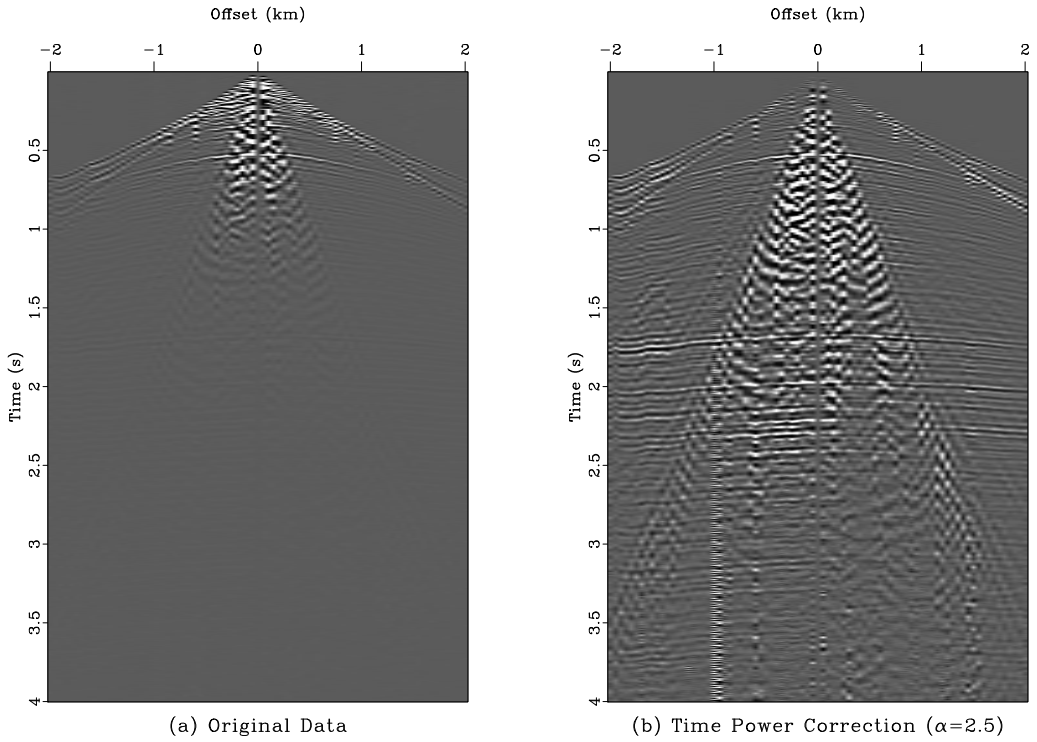


Figure 1.3: Seismic shot record before and after time-power gain correction. hw1/tpow tpow

Figure 1.3 shows a seismic shot record before and after applying the time-power gain (1.4) with $\alpha = 2$. Start by reproducing this figure on your screen.

1. Change directory to `hw1/tpow`
2. Run

```
scons tpow.view
```

3. Edit the `SConstruct` file. Find where the value of α is specified in this file and try changing it to a different value. Run `scons tpow.view` again to check the result.

4. How can we detect if the distribution of amplitudes after the gain correction is uniform? Suggest a measure (an objective function) that would take $S_\alpha(x,t)$ and produce one number that measures uniformity.

5. By modifying the program `objective.c`, compute your objective function for different values of α and display it in a figure. Does the function appear to have a unique minimum or maximum?

tpow/objective.c

```
#include <rsf.h>

int main(int argc, char* argv[])
{
    int it, nt, ix, nx, ia, na;
    float *trace, *ofunc;
    float a, a0, da, t, t0, dt, s;
    sf_file in, out;

    /* initialization */
    sf_init(argc,argv);
    in = sf_input("in");
    out = sf_output("out");

    /* get trace parameters */
    if (!sf_histint(in,"n1",&nt)) sf_error("Need n1=");
    if (!sf_histfloat(in,"d1",&dt)) dt=1.;
    if (!sf_histfloat(in,"o1",&t0)) t0=0.;

    /* get number of traces */
    nx = sf_leftsize(in,1);

    if (!sf_getint("na",&na)) na=1;
    /* number of alpha values */
    if (!sf_getfloat("da",&da)) da=0.;
    /* increment in alpha */
    if (!sf_getfloat("a0",&a0)) a0=0.;
    /* first value of alpha */

    /* change output data dimensions */
    sf_putint(out,"n1",na);
    sf_putint(out,"n2",1);
    sf_putfloat(out,"d1",da);
    sf_putfloat(out,"o1",a0);

    trace = sf_floatalloc(nt);
    ofunc = sf_floatalloc(na);
```

```
38
39      /* initialize */
40      for (ia=0; ia < na; ia++) {
41          ofunc[ia] = 0.;
42      }
43
44      /* loop over traces */
45      for (ix=0; ix < nx; ix++) {
46
47          /* read data */
48          sf_floatread(trace,nt,in);
49
50          /* loop over alpha */
51          for (ia=0; ia < na; ia++) {
52              a = a0+ia*da;
53
54              /* loop over time samples */
55              for (it=0; it < nt; it++) {
56                  t = t0+it*dt;
57
58                  /* apply gain t^alpha */
59                  s = trace[it]*powf(t,a);
60
61                  /* !!! MODIFY THE NEXT LINE !!! */
62                  ofunc[ia] += s*s;
63              }
64          }
65      }
66
67      /* write output */
68      sf_floatwrite(ofunc,na,out);
69
70      exit(0);
71  }
```

6. Suggest an algorithm for finding an optimal value of α by minimizing or maximizing the objective function. Your algorithm should be able to find the optimal value without scanning all possible values. **Hint:** if the objective function is $f(\alpha) = F[S_\alpha(x,t)]$ and

$$f(\alpha) \approx f(\alpha_0) + f'(\alpha_0)\,(\alpha - \alpha_0) + \frac{f''(\alpha_0)}{2}\,(\alpha - \alpha_0)^2 \tag{1.5}$$

then what is the optimal α?

7. **EXTRA CREDIT** for implementing your algorithm for an automatic estimation of α and testing it on the shot gather from Figure 1.3.

tpow/SConstruct

```
from rsf.proj import *

# Download data
Fetch('wz.25.H','wz')

# Convert and window
Flow('data','wz.25.H',
     '''
     dd form=native | window min2=-2 max2=2 |
     put label1=Time label2=Offset unit1=s unit2=km
     ''')

# Display
Plot('data','grey title="(a) Original Data"')
Plot('tpow','data',
     'pow pow1=2 | grey title="(b) Time Power Correction" ')

Result('tpow','data tpow','SideBySideAniso')

# Compute objective function
prog = Program('objective.c')
Flow('ofunc','data %s' % prog[0],
     './${SOURCES[1]} na=21 da=0.1 a0=1')

Result('ofunc',
       '''
       scale axis=1 |
       graph title="Objective Function"
       label1=alpha label2= unit1= unit2=
       ''')

End()
```

1.5 Completing the assignment

1. Change directory to `hw1`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Jensen'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

Claerbout, J. F., 1985, Imaging the Earth's interior: Blackwell Scientific Publications.

Chapter 2

Homework 2

ABSTRACT

This homework has four parts.

1. Theoretical questions related to data attributes.
2. Measuring the performance of sorting algorithms.
3. Analyzing a digital elevation map by applying a running average filter.
4. Analyzing your own data by applying a running average filter.

2.1 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/hw2
```

2.2 Data attributes

You can either write your answers to theoretical questions on paper or edit them in the file `hw2/paper.tex`. Please show all the mathematical derivations that you perform.

1. The varimax attribute is defined as

$$\phi[\mathbf{a}] = \frac{N \sum_{n=1}^{N} a_n^4}{\left(\sum_{n=1}^{N} a_n^2\right)^2} \tag{2.1}$$

   Suppose that the data vector $\mathbf{a}$ consists of random noise: the data values a_n are independent and identically distributed with a zero-mean Gaussian distribution: $E[a_n] = 0$, $E[a_n^2] = \sigma^2$, $E[a_n^4] = 3\,\sigma^4$. Find the mathematical expectation of $\phi[\mathbf{a}]$.

2. The *skewness* attribute evaluates the asymmetry of the data distribution. For zero-mean data, it is defined as

$$\kappa[\mathbf{a}] = \frac{\sum_{n=1}^{N} a_n^3}{\left(\frac{1}{N}\sum_{n=1}^{N} a_n^2\right)^{3/2}} \,. \tag{2.2}$$

   Define skewness squared $\kappa^2[\mathbf{a}]$ in terms of the similarity measure (squared correlation coefficient)

$$\gamma^2[\mathbf{a},\mathbf{b}] = \frac{\left(\sum_{n=1}^{N} a_n\, b_n\right)^2}{\sum_{n=1}^{N} a_n^2 \sum_{n=1}^{N} b_n^2} \,. \tag{2.3}$$

2.3 Sorting algorithms

1. Change directory to `hw2/sorting`.
2. Run

   ```
   scons movie.vpl
   ```

   and observe a movie illustrating the slow data sorting algorithm. The algorithm is implemented in the `slow_sort` function in the file `sorting.c`.
3. Run

   ```
   scons view
   ```

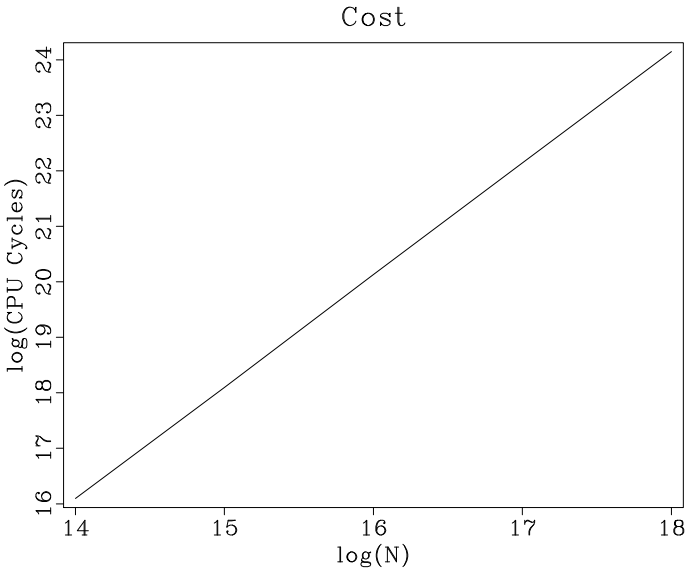


Figure 2.1: Experimental cost of slow sorting. The logarithm of the cost is shown against the logarithm of the data size. hw2/sorting cost

to compute the cost of slow sorting experimentally. The output is shown in Figure 2.1.

If we approximate the cost as $P(N) = C\,N^{\epsilon}$, what is the value of ϵ observed in the picture?

4. Open the file `sorting.c` in a text editor and edit it to fix the specified line in the `quick_sort` function.

5. Open the file `SConstruct` in a text editor and uncomment the specified line.

6. Rerun

   ```
   scons movie.vpl
   ```

   to observe a change in the sorting movie. Debug your changes to the program if necessary.

7. Run

   ```
   scons view
   ```

   to observe the change in the algorithm cost. What is the new experimental value of ϵ?

8. **EXTRA CREDIT** for improving the speed of the `quick_sort` algorithm even further.

sorting/sorting.c

```
1 #include <time.h>
2 #include <rsf.h>
3
4 static int na, nmovie=0;
```

```
5  static float *arr;
6  static sf_file movie;
7
8  static void write_movie(void)
9  {
10     if (NULL != movie)
11         sf_floatwrite(arr,na,movie);
12     nmovie++;
13 }
14
15 static void slow_sort(int n, float* list)
16 {
17     int k, k2;
18     float item1, item2;
19
20     for (k=0; k < n; k++) {
21         write_movie();
22
23         item1 = list[k];
24         /* assume everything up to k is sorted */
25         for (k2=k; k2 > 0; k2--) {
26             item2 = list[k2-1];
27             if (item1 >= item2) break;
28             list[k2]   = item2;
29         }
30         list[k2] = item1;
31     }
32 }
33
34 static void quick_sort(int n, float* list)
35 {
36     int l, r;
37     float ll, pivot;
38
39     if (n <= 1) return;
40
41     write_movie();
42
43     l = 1; /* left side */
44     r = n; /* right side */
45     pivot = list[0];
46
47     /* separate into two lists:
48        the left list for values <= pivot
49        and the right list for > pivot */
50     while (l < r) {
51         ll = list[l];
52         if (ll <= pivot) {
```

```
53              l++;
54          } else {
55              r--;
56              list[l] = list[r];
57              list[r] = ll;
58          }
59      }
60      list[0] = list[l-1];
61      list[l-1] = pivot;
62
63      quick_sort(l-1,list);
64
65      /* !!! UNCOMMENT AND EDIT THE NEXT LINE !!! */
66      /* quick_sort(?,?); */
67  }
68
69  int main(int argc, char* argv[])
70  {
71      char* type;
72      clock_t start, end;
73      float cycles;
74      sf_file in, cost;
75
76      /* initialize */
77      sf_init(argc,argv);
78
79      /* input file */
80      in = sf_input("in");
81      if (SF_FLOAT != sf_gettype(in))
82          sf_error("Need float input");
83      na = sf_filesize(in); /* data size */
84
85      /* cost file */
86      cost = sf_output("out");
87      sf_putint(cost,"n1",1);
88
89      /* movie file */
90      if (NULL != sf_getstring("movie")) {
91          movie = sf_output("movie");
92          sf_putint(movie,"n2",1);
93          sf_putint(movie,"n3",na+1);
94      } else {
95          movie = NULL;
96      }
97
98      if (NULL == (type = sf_getstring("type")))
99          type = "quick"; /* sort type */
100
```

```
    /* get data */
    arr = sf_floatalloc(na);
    sf_floatread(arr,na,in);

    /* sort */
    start = clock();
    if ('q'==type[0]) {
        quick_sort(na, arr);
    } else {
        slow_sort(na, arr);
    }
    end = clock();

    /* CPU cycles */
    cycles = end - start;
    sf_floatwrite(&cycles,1,cost);

    while (nmovie < na+1) write_movie();

    exit(0);
}
```

sorting/SConstruct

```
from rsf.proj import *

# Generate random data
######################
Flow('rand',None,
     'spike n1=524288 | noise rep=y type=n seed=2012')

prog = Program('sorting.c')

sort = 'slow'

# !!! UNCOMMENT THE NEXT LINE !!!
# sort = 'quick'

# Sorting movie
###############
Flow('movie','rand %s' % prog[0],
     '''
     window n1=200 |
     ./${SOURCES[1]} movie=$TARGET type=%s
     ''' % sort,stdout=0)
Plot('movie',
     '''
     graph symbol=o title="%s Sort" wantaxis=n symbolsz=5
```

```
      ''' % sort.capitalize(),view=1)

# Sorting cost
##############
na = 8192
costs = []
for n in range(5):
    na *= 2
    cost = 'cost%d' % n
    Flow(cost,'rand  %s' % prog[0],
         '''
         window n1=%d |
         ./${SOURCES[1]} type=%s
         ''' % (na,sort))
    costs.append(cost)
Flow('cost',costs,
     'cat axis=1 ${SOURCES[1:5]} | put o1=14 d1=1 unit1=')

Result('cost',
       '''
       math output="log(input)/log(2)" |
       graph title=Cost
       label1="log(N)" label2="log(CPU Cycles)"
       ''')

End()
```

2.4 Running median and running mean filters

We return the digital elevation map of the West Austin Area, shown in Figure 2.2.

In this exercise, we will separate the data into "signal" and "noise" by applying running mean and median filters. The result of applying a running median filter is shown in Figure 2.3. Running median effectively smooths the data by removing local outliers.

The algorithm is implemented in program `running.c`.

running/running.c

```
/* Apply running mean or median filter */

#include <rsf.h>

static float slow_median(int n, float* list)
/* find median by slow sorting, changes list */
{
    int k, k2;
    float item1, item2;
```

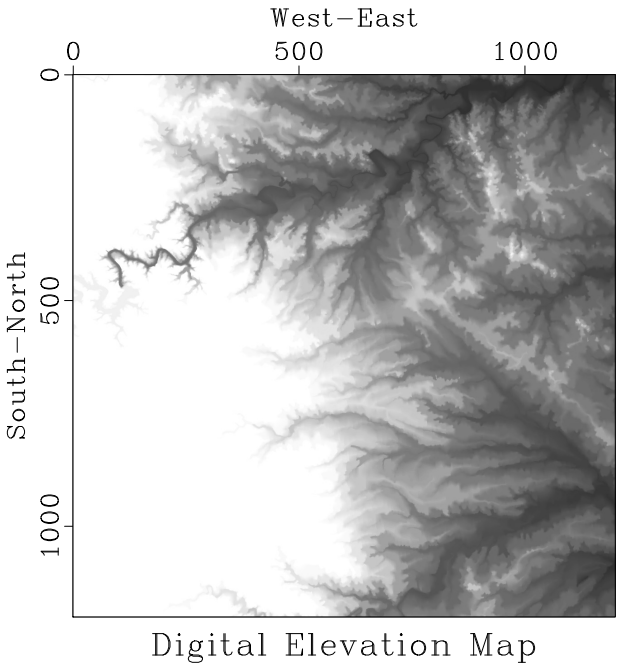


Figure 2.2: Digital elevation map of the west Austin area. hw2/running dem

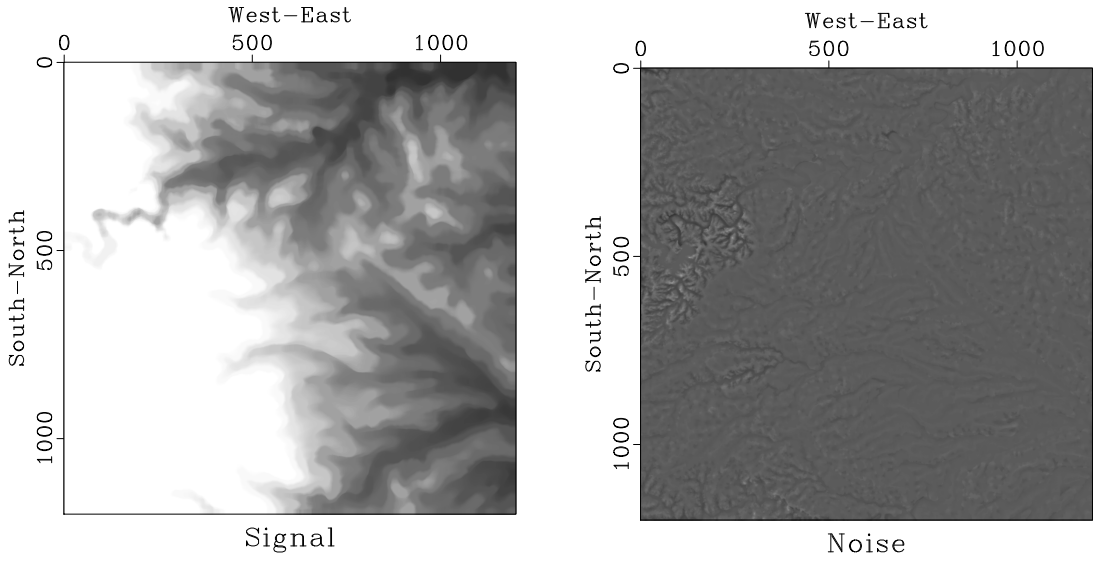


Figure 2.3: Data separated into signal (a) and noise (b) by applying a running median filter. hw2/running ave,res

```
10
11      for (k=0; k < n; k++) {
12          item1 = list[k];
13
14          /* assume everything up to k is sorted */
15          for (k2=k; k2 > 0; k2--) {
16              item2 = list[k2-1];
17              if (item1 >= item2) break;
18              list[k2]   = item2;
19          }
20          list[k2] = item1;
21      }
22
23      return list[n/2];
24  }
25
26  int main(int argc, char* argv[])
27  {
28      int w1, w2, nw, s1,s2, j1,j2, i1,i2,i3, n1,n2,n3;
29      char *what;
30      float **data, **signal, **win;
31      sf_file in, out;
32
33      sf_init (argc,argv);
34      in = sf_input("in");
35      out = sf_output("out");
36
37      /* get data dimensions */
38      if (!sf_histint(in,"n1",&n1)) sf_error("No n1=");
39      if (!sf_histint(in,"n2",&n2)) sf_error("No n2=");
40      n3 = sf_leftsize(in,2);
41
42      /* input and output */
43      data = sf_floatalloc2(n1,n2);
44      signal = sf_floatalloc2(n1,n2);
45
46      if (!sf_getint("w1",&w1)) w1=5;
47      if (!sf_getint("w2",&w2)) w2=5;
48      /* sliding window width */
49
50      nw = w1*w2;
51      win = sf_floatalloc2(w1,w2);
52
53      what = sf_getstring("what");
54      /* what to compute
55         (fast median, slow median, mean) */
56      if (NULL == what) what="fast";
57
```

```
    for (i3=0; i3 < n3; i3++) {

        /* read data plane */
        sf_floatread(data[0],n1*n2,in);

        for (i2=0; i2 < n2; i2++) {
            s2 = SF_MAX(0,SF_MIN(n2-w2,i2-w2/2-1));
            for (i1=0; i1 < n1; i1++) {
                s1 = SF_MAX(0,SF_MIN(n1-w1,i1-w1/2-1));

                /* copy window */
                for (j2=0; j2 < w2; j2++) {
                    for (j1=0; j1 < w1; j1++) {
                        win[j2][j1] = data[s2+j2][s1+j1];
                    }}

                switch (what[0]) {
                    case 'f': /* fast median */
                        signal[i2][i1] =
                            sf_quantile(nw/2,nw,win[0]);
                        break;
                    case 's': /* slow median */
                        signal[i2][i1] =
                            slow_median(nw,win[0]);
                        break;
                    case 'm': /* mean */
                    default:
                        /* !!! ADD CODE !!! */
                        break;
                }
            }
        }

        /* write out */
        sf_floatwrite(signal[0],n1*n2,out);
    }

    exit(0);
}
```

1. Change directory to `hw2/running`.
2. Run

   ```
   scons view
   ```

   to reproduce the figures on your screen.

3. Modify the `running.c` program and the `SConstruct` file to compute running mean instead of running median. Compare the results.

4. **EXTRA CREDIT** for improving the efficiency of the running median algorithm. Run

   ```
   scons time.vpl
   ```

   to display a figure that compares the efficiency of running median computations using the slow sorting from function `median` in program `running.c` and the fast quantile algorithm (library function `sf_quantile`). Your goal is to make the algorithm even faster. Consider parallelization, reusing previous windows, other fast sorting strategies, etc.

running/SConstruct

```
from rsf.proj import *

# Download data
Fetch('austin-w.HH','bay')

# Convert format
Flow('dem','austin-w.HH','dd form=native')

# Display
def plot(title):
    return '''
    grey clip=250 allpos=y title="%s"
    screenratio=1
    ''' % title

Result('dem',plot('Digital Elevation Map'))

# Running median program
run = Program('running.c')

w = 30

# !!! CHANGE BELOW !!!
Flow('ave','dem %s' % run[0],
     './${SOURCES[1]} w1=%d w2=%d what=fast' % (w,w))
Result('ave',plot('Signal'))

# Difference
Flow('res','dem ave','add scale=1,-1 ${SOURCES[1]}')
Result('res',plot('Noise') + ' allpos=n')

##########################################################################

```

```
34 import sys
35
36 if sys.platform=='darwin':
37       gtime = WhereIs('gtime')
38       if not gtime:
39             print "For computing CPU time, please install gtime."
40 else:
41       gtime = WhereIs('gtime') or WhereIs('time')
42
43 # slow or fast
44 for case in ('fast','slow'):
45
46     ts = []
47     ws = []
48
49     time = 'time-' + case
50     wind = 'wind-' + case
51
52
53
54     # loop over window size
55     for w in range(3,16,2):
56         itime = '%s-%d' % (time,w)
57         ts.append(itime)
58
59         iwind = '%s-%d' % (wind,w)
60         ws.append(iwind)
61
62         # measure CPU time
63
64
65
66         Flow(iwind,None,'spike n1=1 mag=%d' % (w*w))
67         Flow(itime,'dem %s' % run[0],
68               '''
69               ( (%s -f "%%%S %%U"
70               ./${SOURCES[1]} < ${SOURCES[0]}
71                w1=%d w2=%d what=%s > /dev/null ) 2>&1 )
72                > time.out &&
73               (tail -1 time.out;
74                echo in=time0.asc n1=2 data_format=ascii_float)
75                > time0.asc &&
76               dd form=native < time0.asc | stack axis=1 norm=n
77               > $TARGET &&
78               /bin/rm time0.asc time.out
79               ''' % (gtime,w,w,case),stdin=0,stdout=-1)
80
81     Flow(time,ts,'cat axis=1 ${SOURCES[1:%d]}' % len(ts))
```

```
82      Flow(wind,ws,'cat axis=1 ${SOURCES[1:%d]}' % len(ws))
83
84      # complex numbers for plotting
85      Flow('c'+time,[wind,time],
86           '''
87           cat axis=2 ${SOURCES[1]} |
88           transp |
89           dd type=complex
90           ''')
91
92  # Display CPU time
93  Plot   ('time','ctime-fast ctime-slow',
94          '''
95          cat axis=1 ${SOURCES[1]} | transp |
96          graph dash=0,1 wanttitle=n
97          label2="CPU Time" unit2=s
98          label1="Window Size" unit1=
99          ''',view=1)
100
101 End()
```

2.5 Your own data

Your final task is to apply the technique of the previous section to your own data:

1. Select a dataset suitable for running mean/median filters.
2. Apply the algorithm of the previous section and choose appropriate parameters.
3. Include the results in your homework.

2.6 Completing the assignment

1. Change directory to `hw2`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Hoare'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.

Chapter 3

Homework 3

ABSTRACT

This homework has four parts, one theoretical and three computational.

1. Theoretical questions related to digital data analysis.
2. Data compression using 2-D Fourier transform.
3. Missing data interpolation using compressive properties of the 2-D Fourier transform.
4. Analyzing your own data.

3.1 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/hw3
```

3.2 Digital signal analysis

You can either write your answers to theoretical questions on paper or edit them in the file `hw3/paper.tex`. Please show all the mathematical derivations that you perform.

1. The matrix in equation (3.1) represents a convolution operator with zero boundary conditions.

$$\mathbf{F} = \begin{bmatrix} f_1 & f_0 & 0 & 0 & 0 & 0 \\ f_2 & f_1 & f_0 & 0 & 0 & 0 \\ f_3 & f_2 & f_1 & f_0 & 0 & 0 \\ 0 & f_3 & f_2 & f_1 & f_0 & 0 \\ 0 & 0 & f_3 & f_2 & f_1 & f_0 \\ 0 & 0 & 0 & f_3 & f_2 & f_1 \end{bmatrix} . \tag{3.1}$$

   The operator is implemented in the C function `hw3/conv.c`.

hw3/conv.c

```
void conv_lop (bool adj, bool add,
               int nx, int ny, float* xx, float* yy)
/*< linear operator >*/
{
    int f, x, y, x0, x1;

    assert (ny == nx);
    sf_adjnull (adj, add, nx, ny, xx, yy);

    for (f=0; f < nf; f++) {
        x0 = SF_MAX(0,1-f);
        x1 = SF_MIN(nx,nx+1-f);
        for (x = x0; x < x1; x++) {
            if( adj) {
                /* add code */
            } else {
                yy[x+f-1] += xx[x] * ff[f];
            }
        }
    }
}
```

   (a) Modify the matrix and the program to implement periodic boundary conditions.

   (b) Add the code for the adjoint (matrix transpose) operator.

2. The C code in `hw3/filter.c` implements a recursive filtering operator.

hw3/filter.c

```
void filter_lop (bool adj, bool add,
                 int nx, int ny, float* xx, float* yy)
```

```
3  /*< linear operator >*/
4  {
5      int i;
6      float t;
7
8      assert (ny == nx);
9      sf_adjnull (adj, add, nx, ny, xx, yy);
10
11     if (adj) {
12         /* add code */
13     } else {
14         t = a*xx[0];
15         yy[0] += t;
16         for (i = 1; i < nx; i++) {
17             t = a*xx[i] + b*xx[i-1] + c*t;
18             yy[i] += t;
19         }
20     }
21 }
```

(a) Express this filter in the Z-transform notation as a ratio of two polynomials.

(b) Add code for the adjoint operator.

3. Show that, using the helix transform and imposing helical boundary conditions, it is possible to compute a 2-D digital Fourier transform using 1-D FFT program. Assuming the input data is of size $N \times N$, would this approach have any computational advantages?

3.3 Fourier compression

In this exercise, we will use a depth slice selected from a 3-D seismic volume and shown in Figure 3.1[1]. Notice a channel structure.

The goal of your assignment is to find a compressed representation of the data in the Fourier transform domain. Figure 3.2 shows the Fourier transform of the data from Figure 3.1. We can see that most of the energy gets concentrated near the center (zero frequency).

There are two alternative ways to compress data in the Fourier domain:

- One approach is to select a range of frequencies that contain the most important information. An advantage of this approach is the ability to subsample the original data by transforming back from a windowed range of frequencies. The results from this method are shown in Figure 3.3.

[1]Courtesy of Matt Hall (ConocoPhillips Canada Ltd.)

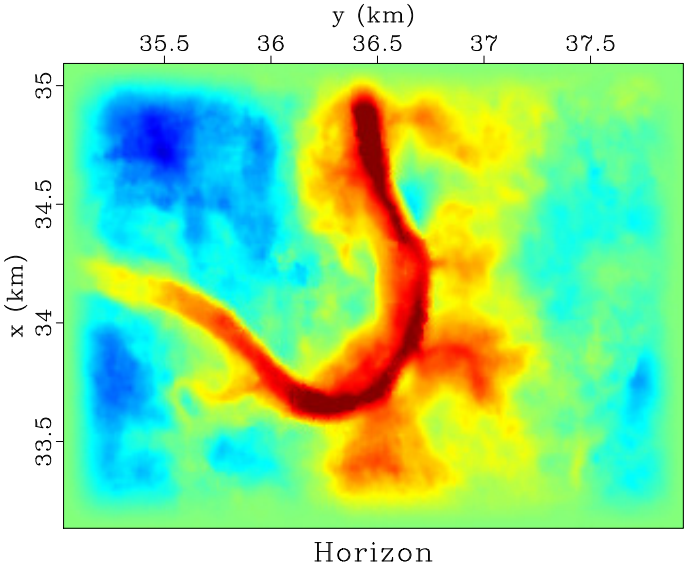


Figure 3.1: Seismic depth slice with a channel structure. hw3/fourier data

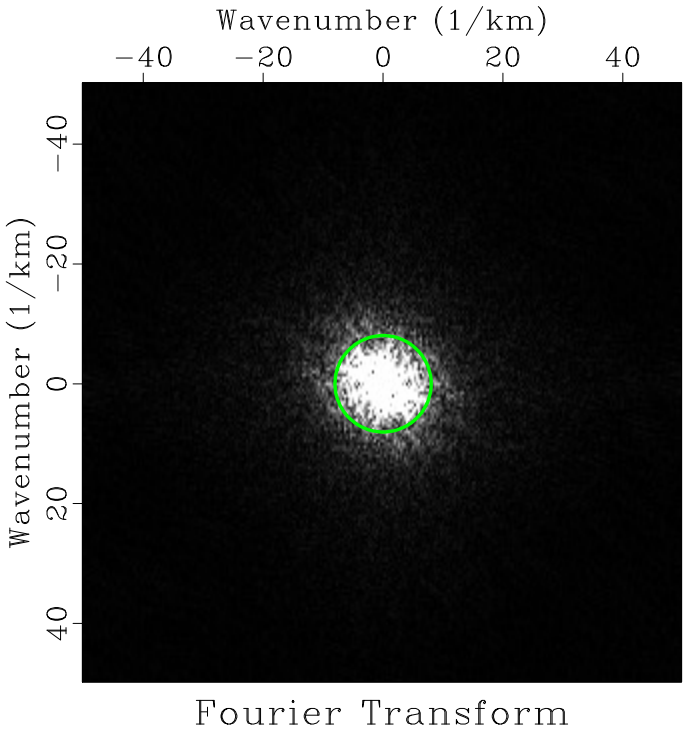


Figure 3.2: Absolute value of the Fourier transform of the seismic slice from Figure 3.1. The circle inside shows a window selected for compression. hw3/fourier fft

- Another approach is to zero all Fourier coefficients below a certain threshold value, regardless of which frequencies they represent. The results from this method are shown in Figure 3.4. Figure 3.5 shows a selected threshold plotted against the histogram of Fourier coefficients.

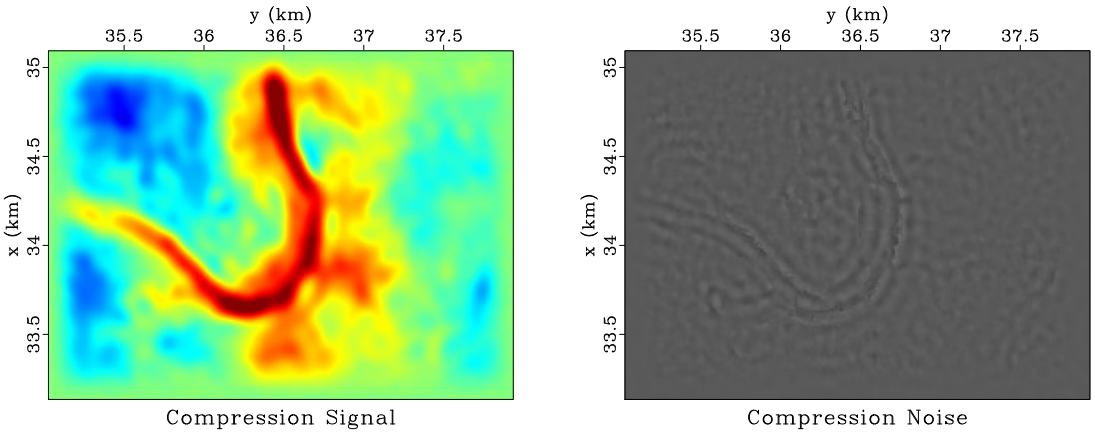


Figure 3.3: Data separated into signal (a) and noise (b) by applying Fourier compression with windowing. hw3/fourier sig,cut

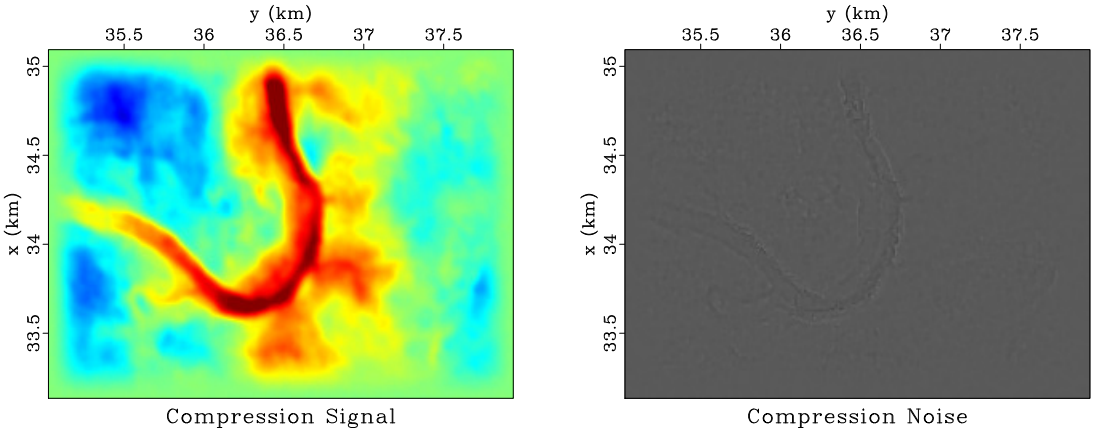


Figure 3.4: Data separated into signal (a) and noise (b) by applying Fourier compression with thresholding. hw3/fourier thr,noi

1. Change directory to `hw3/fourier`.
2. Run

   ```
   scons view
   ```

   to reproduce the figures on your screen.
3. Modify the `SConstruct` file to decrease the size of the window so that the noise level increases in Figure 3.3(b). How do you measure the noise level? Find a level that you find negligibly small.

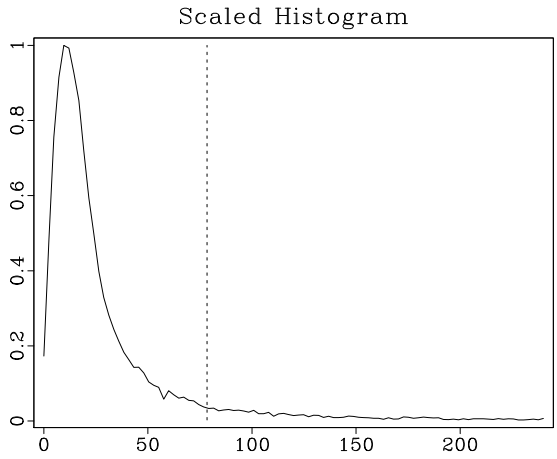


Figure 3.5: Normalized histogram of Fourier coefficients (by absolute value). The vertical line shows a selected threshold. hw3/fourier hist

4. Modify the `SConstruct` file to increase the threshold value so that the compression achieves the same quality as in the previous case. The noise level in Figure 3.4(b) should match that in Figure 3.3(b).

5. Compare the number of nonzero Fourier coefficients in both cases. Which method achieves a better compression?

6. **EXTRA CREDIT** for finding a way for a better compression of the data in the Fourier domain. Your data reconstruction should have the same noise level, yet the number of non-zero coefficients in the Fourier domain should be smaller.

```
from rsf.proj import *

# Get data
##########
Fetch('horizon.asc','hall')

# Convert format
Flow('data','horizon.asc',
     '''
     echo in=$SOURCE data_format=ascii_float n1=3 n2=57036 |
     dd form=native | window n1=1 f1=-1 | add add=-65 |
     put
     n2=291 o2=35.031 d2=0.01 label2=y unit2=km
     n1=196 o1=33.139 d1=0.01 label1=x unit1=km |
     costaper nw1=25 nw2=25
     ''')

# Display
def plot(title):
    return '''
    grey color=j title="%s"
    transp=y yreverse=n clip=14
    ''' % title
```

```
Result('data',plot('Horizon'))

# 2-D Fourier Transform
########################
Flow('fft','data',
     'rtoc | fft3 axis=1 pad=1 | fft3 axis=2 pad=1')
Plot('fft',
     '''
     math output="abs(input)" | real |
     grey title="Fourier Transform" allpos=y screenratio=1
     ''')

# A. Compression by Windowing
##############################

cut = 8 # !!! CHANGE ME !!!

# Create a frame
Flow('frame','fft','real | math output="sqrt(x1*x1+x2*x2)" ')
Plot('frame',
     '''
     contour nc=1 c0=%g plotfat=5 plotcol=3
     wantaxis=n wanttitle=n screenratio=1
     ''' % cut)
Result('fft','fft frame','Overlay')

# Cut a hole
Flow('fcut','frame fft',
     '''
     mask max=%g |
     dd type=float | rtoc |
     mul ${SOURCES[1]}
     ''' % cut)

# Inverse FFT
Flow('sig','fcut',
     'fft3 axis=2 inv=y | fft3 axis=1 inv=y | real')
Result('sig',plot('Compression Signal'))

Flow('cut','data sig','add scale=1,-1 ${SOURCES[1]}')
Result('cut',plot('Compression Noise') + 'color=I')

# B. Compression by Thresholding
#################################

thr = 80 # !!! CHANGE ME !!!

# Plot histogram
```

```
Plot('hist','fft',
     '''
     math output="abs(input)" | real |
     histogram o1=0 d1=%g n1=101 |
     dd type=float | scale axis=1 |
     graph title="Scaled Histogram" pad1=n
     label1= unit1= label2= unit2=
     ''' % (0.03*thr))
Flow('line.asc',None,
     'echo 0 0 0 1 n1=4 data_format=ascii_float in=$TARGET')
Plot('line','line.asc',
     '''
     dd type=complex form=native |
     graph min1=-1 max1=2 plotcol=5
     wantaxis=n wanttitle=n dash=1
     ''')
Result('hist','hist line','Overlay')

# Thresholding
Flow('fthr','fft','thr thr=%g' % thr)

# Inverse FFT
Flow('thr','fthr',
     'fft3 axis=2 inv=y | fft3 axis=1 inv=y | real')
Result('thr',plot('Compression Signal'))

# Subtract from Data
Flow('noi','data thr','add scale=1,-1 ${SOURCES[1]}')
Result('noi',plot('Compression Noise') + 'color=I')

End()
```

3.4 Projection onto convex sets

The goal of the next exercise is to figure out if one can use compactness of the Fourier transform to reconstruct missing data. The missing parts are created artificially by cutting holes in the original data (Figure 3.6).

Figures 3.7(a) and 3.7(b) show the digital Fourier transform of the original data and the data with holes. We observe again that the support of the data in the Fourier domain is compact thanks to the data smoothness. Cutting holes in the physical domain creates discontinuities that make the Fourier response spread beyond the original support. Figure 3.8 shows a Fourier-domain mask designed to contain the support of the original data.

To accomplish the task of missing data interpolation, we will use an iterative method known as POCS (*projection onto convex sets*). By definition, a convex set $\mathcal{C}$ is a set of functions such that, for any $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ from the set, $g(\mathbf{x}) = \lambda\, f_1(\mathbf{x}) + (1-\lambda)\, f_2(\mathbf{x})$ (for

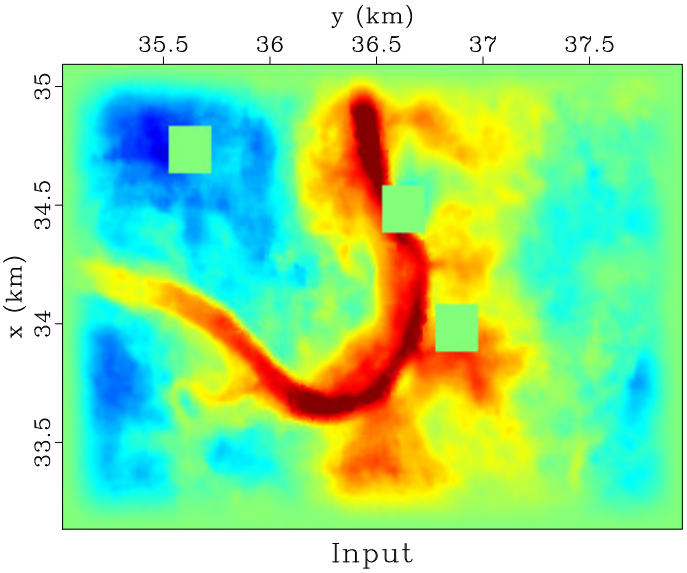


Figure 3.6: Seismic depth slice after removing selected parts of the data. hw3/pocs hole

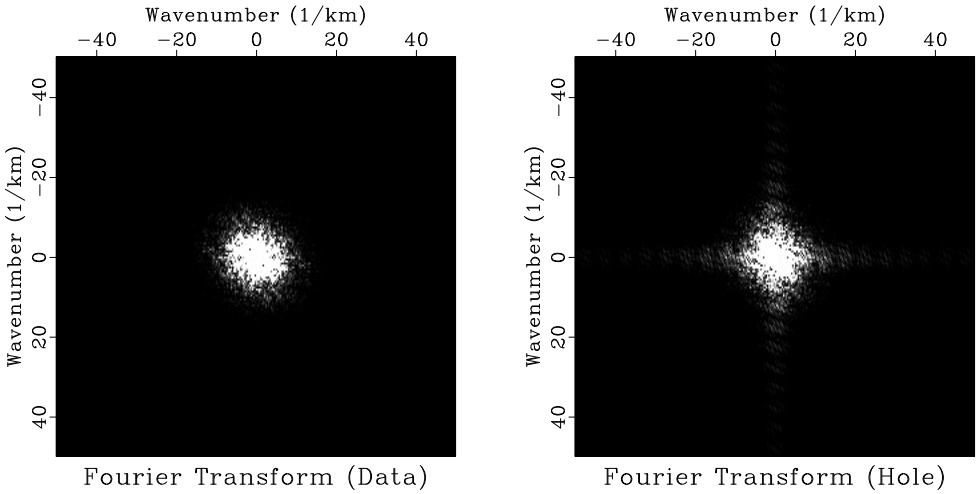


Figure 3.7: Fourier transform of the original data (a) and data with holes with holes (b). The absolute value is displayed hw3/pocs fft-data,fft-hole

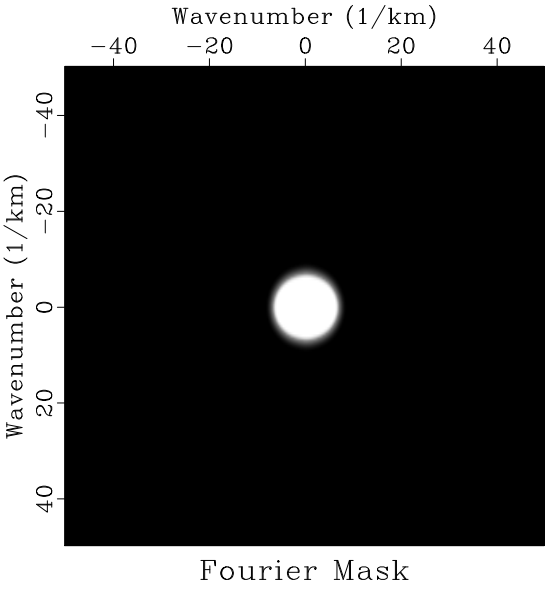


Figure 3.8: Fourier-domain mask for selecting a convex set. hw3/pocs fft-mask

$0 \leq \lambda \leq 1$) also belongs to the set. A projection onto a convex set means finding a function in the set that is of the shortest distance to the given function. The POCS theorem states that if one wants to find a function that belongs to the intersection of two convex sets C_1 and C_2, the task can be accomplished iteratively by alternating projections onto the two sets.

In our example, C_1 is the set of all functions that are equal to the known data outside of the holes. C_2 is the set of all functions that have a predifined compact support in the Fourier domain (and therefore are smooth in the physical domain). The algorithm consists of the following steps:

1. Apply 2-D Fourier transform.
2. Multiply by a Fourier-transform mask to enforce compact support.
3. Apply inverse 2-D Fourier transform.
4. Replace data outside of the holes with known data.
5. Repeat.

The output after 5 iterations is shown in Figure 3.9.

```
from rsf.proj import *

# Download data
Fetch('horizon.asc','hall')

# Convert format
Flow('data','horizon.asc',
     '''
     echo in=$SOURCE data_format=ascii_float n1=3 n2=57036 |
```

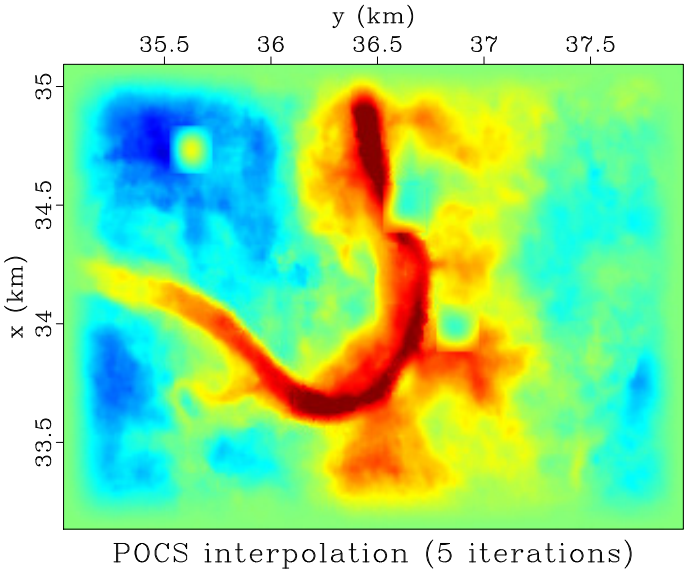


Figure 3.9: Missing data interpolated by iterative projection onto convex sets. hw3/pocs pocs

```
10      dd form=native | window n1=1 f1=-1 | add add=-65 |
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      costaper nw1=25 nw2=25
15      ''')
16
17  # Display
18  def plot(title):
19      return '''
20      grey color=j title="%s"
21      transp=y yreverse=n clip=14
22      ''' % title
23  Result('data',plot('Horizon'))
24
25  # Cut three square holes (!!! CHANGE ME !!!)
26  cut = '''
27  cut n1=20 n2=20 f1=125 f2=150 |
28  cut n1=20 n2=20 f1=150 f2=50  |
29  cut n1=20 n2=20 f1=75  f2=175
30  '''
31
32  Flow('hole','data',cut)
33  Flow('mask','data',
34       'math output=1 | %s | math output=1-input' % cut)
35  Plot('hole',plot('Input'))
36  Result('hole','Overlay')
37
38  # Fourier transform
```

```
forw = 'rtoc | fft3 axis=1 pad=1 | fft3 axis=2 pad=1'
back = 'fft3 axis=2 inv=y | fft3 axis=1 inv=y | real'

for data in ('data','hole'):
    fft = 'fft-'+data
    Flow(fft,data,forw)
    Result(fft,
           '''
           math output="abs(input)" | real |
           grey allpos=y title="Fourier Transform (%s)"
           screenratio=1
           ''' % data.capitalize())

# Create Fourier mask
Flow('fft-mask','fft-hole',
     '''
     real | math output="x1*x1+x2*x2" | mask min=50 |
     dd type=float | math output=1-input |
     smooth rect1=5 rect2=5 repeat=3 | rtoc
     ''')
Result('fft-mask',
       '''
       real |
       grey allpos=y title="Fourier Mask" screenratio=1
       ''')

# POCS iterations
niter=5 # !!! CHANGE ME !!!

data = 'hole'
plots = ['hole']
for iter in range(niter):
    old = data
    data = 'data%d' % iter

    # 1. Forward FFT
    # 2. Multiply by Fourier mask
    # 3. Inverse FFT
    # 4. Multiply by space mask
    # 5. Add data outside of hole
    Flow(data,[old,'fft-mask','mask','hole'],
         '''
         %s | mul ${SOURCES[1]} |
         %s | mul ${SOURCES[2]} |
         add ${SOURCES[3]}
         ''' % (forw,back))
    Plot(data,plot('Iteration %d' % (iter+1)))
    plots.append(data)
```

```
87 # Put frames in a movie
88 Plot('pocs',plots,'Movie',view=1)
89
90 # Last frame
91 Result('pocs',data,
92         plot('POCS interpolation (%d iterations)' % niter))
93
94 End()
```

Your task:

1. Change directory to `hw3/pocs`
2. Run

   ```
   scons view
   ```

   to reproduce the figures on your screen.
3. Additionally, you can run

   ```
   scons pocs.vpl
   ```

   to see a movie of different iterations.
4. By modifying appropriate parameters in the `SConstruct` file and repeating computations, find out
   (a) How many iterations are required for convergence?
   (b) How large can you make the holes and still be able to achieve a reasonably good reconstruction?
5. **EXTRA CREDIT** for finding a different convex set for either faster or more accurate missing data reconstruction.

3.5 Your own data

Your final task is to apply one of the data analysis techniques of the previous sections to your own data:

1. Select a dataset suitable for compression or interpolation.
2. Apply one of the algorithms of the previous two sections and choose appropriate parameters.
3. Include the results in your homework.

3.6 Completing the assignment

1. Change directory to `hw3`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Fourier'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.

Chapter 4

Homework 4

ABSTRACT

This homework has four parts, two theoretical and two computational.

1. Theoretical questions related to B-spline interpolation.
2. Theoretical questions related to the conjugate-gradient algorithm.
3. Data interpolation after coordinate transformation.
4. Irregular data interpolation contest using rainfall data from Switzerland.

4.1 Prerequisites

Completing the computational part of this homework assignment requires

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

4.2 Theory

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

1. The parabolic B-spline $\beta_2(x)$ is a function defined as

$$\beta_2(x) = \int_{-\infty}^{\infty} \beta_1(t)\, \beta_0(x-t)\, dt \;, \tag{4.1}$$

where

$$\beta_0(x) = \begin{cases} 1 & \text{for} \quad |x| \le 1/2 \\ 0 & \text{for} \quad |x| > 1/2 \end{cases} \tag{4.2}$$

and

$$\beta_1(x) = \int_{-\infty}^{\infty} \beta_0(t)\, \beta_0(x-t)\, dt \;= \begin{cases} 1-|x| & \text{for} \quad |x| \le 1 \\ 0 & \text{for} \quad |x| > 1 \end{cases} \tag{4.3}$$

   (a) Find an explicit expression for $\beta_2(x)$.

   (b) Show that decomposing a continuous data function $d(x)$ into the convolution basis with parabolic B-spines

$$d(x) = \sum_k c_k\, \beta_2(x-k) \tag{4.4}$$

   leads to an interpolation filter of the form

$$Z^\sigma \approx B_2(Z) = \frac{a_0(\sigma)\, Z^{-1} + a_1(\sigma) + a_2(\sigma)\, Z}{b_0\, Z^{-1} + b_1 + b_2\, Z} \;. \tag{4.5}$$

   Define $a_0(\sigma)$, $a_1(\sigma)$, $a_2(\sigma)$, b_0, b_1, and b_2.

2. The following algorithm finds a solution to the least-squares optimization problem $\min |\mathbf{F}\,\mathbf{m} - \mathbf{d}|^2$, where $\mathbf{d}$ is data, $\mathbf{m}$ is the model we want to estimate, and $\mathbf{F}$ is a linear operator that connects them.

CONJUGATE GRADIENTS($\mathbf{F}, \mathbf{d}, N$)

$\mathbf{m} \leftarrow \mathbf{0}$
$\mathbf{r} \leftarrow -\mathbf{d}$
for $n \leftarrow 1, 2, \ldots, N$
do
  $\mathbf{g}_m \leftarrow \mathbf{F}^T\, \mathbf{r}$
  $\mathbf{g}_r \leftarrow \mathbf{F}\, \mathbf{g}_m$
  $\rho \leftarrow \mathbf{g}_m^T\, \mathbf{g}_m$
  if $n = 1$
    then $\beta \leftarrow 0$
    else $\beta \leftarrow \rho/\hat{\rho}$
  $\begin{bmatrix} \mathbf{s}_m \\ \mathbf{s}_r \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{g}_m \\ \mathbf{g}_r \end{bmatrix} + \beta \begin{bmatrix} \mathbf{s}_m \\ \mathbf{s}_r \end{bmatrix}$

12 $\quad \alpha \leftarrow -\rho/(\mathbf{s}_r^T\,\mathbf{s}_r)$
13 $\quad \begin{bmatrix} \mathbf{m} \\ \mathbf{r} \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{m} \\ \mathbf{r} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{s}_m \\ \mathbf{s}_r \end{bmatrix}$
14 $\quad \hat{\rho} \leftarrow \rho$
15 **return m**

(a) In applications, it is often advantageous to apply model re-parametrization or *preconditioning*. Suppose that, instead of solving for $\mathbf{m}$ directly, you first solve for $\mathbf{x}$ such that $\mathbf{m} = \mathbf{P}\,\mathbf{x}$. Show how to incorporate the linear preconditioning operator $\mathbf{P}$ in the algorithm above.

(b) Prove that the output of the algorithm after N iterations is

$$\mathbf{m}_N = \sum_{n=1}^{N} \frac{\mathbf{s}_n\,\mathbf{s}_n^T}{\mathbf{s}_n^T\,\mathbf{F}^T\,\mathbf{F}\,\mathbf{s}_n}\,\mathbf{F}^T\,\mathbf{d}\;, \tag{4.6}$$

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

(c) Assuming that $\mathbf{m}$ is the true model, show that

$$\mathbf{m}_N = \sum_{n=1}^{N} \frac{\mathbf{g}_n\,\mathbf{g}_n^T}{\mathbf{g}_n^T\,\mathbf{g}_n}\,\mathbf{m}\;, \tag{4.7}$$

where $\mathbf{g}_n$ is the gradient at n-th iteration.

4.3 Interpolation after coordinate transformation

In this exercise, we will use a slice out of a 3-D CT-scan of a carbonate rock sample, shown in Figure 4.1(a)[1]. Notice microfracture channels.

The goal of the exercise is to apply a coordinate transformation to the original data. A particular transformation that we will study is coordinate rotation. Figure 4.1(b) shows the original slice rotated by 90 degrees. A 90-degree rotation in this case amounts to simple transpose. However, rotation by a different angle requires interpolation from the original grid to the modified grid.

The task of coordinate rotation is accomplished by the C program `rotate.c`. Two different methods are implemented: nearest-neighbor interpolation and bilinear interpolation.

To test the accuracy of different methods, we can rotate the original data in small increments and then compare the result of rotating to 360° with the original data. Figure 4.2 compares the error of the nearest-neighbor and bilinear interpolations after rotating the original slice in increments of 20°. The accuracy is comparatively low for small discontinuous features like microfracture channels.

To improve the accuracy further, we need to employ a longer filter. One popular choice is *cubic convolution* interpolation, invented by Robert Keys (a geophysicist, currently at

[1]Courtesy of Jim Jennings (currently at Shell.)

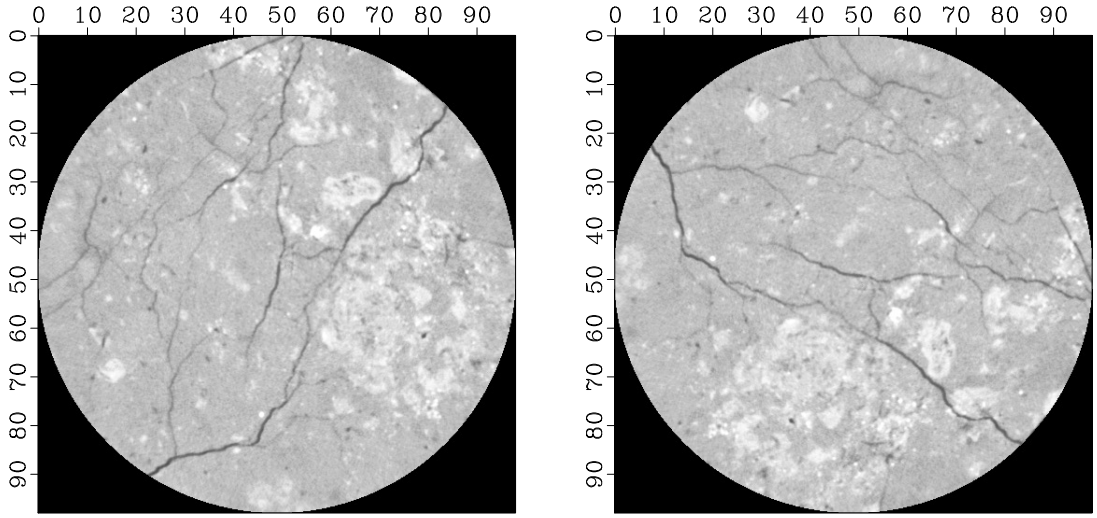


Figure 4.1: Slice of a CT-scan of a carbonate rock sample. (a) Original. (b) After clockwise rotation by 90°. hw4/rotate circle,rotate

ConocoPhillips). The cubic convolution filter can be expressed as the filter (Keys, 1981)

$$Z^{\sigma} \approx C(Z) = -\frac{\sigma(1-\sigma)^2}{2} Z^{-1} + \frac{(1-\sigma)(2+2\sigma-3\sigma^2)}{2} + \frac{\sigma(1+4\sigma-3\sigma^2)}{2} Z - \frac{(1-\sigma)\sigma^2}{2} Z^2 . \quad (4.8)$$

and is designed to approximate the ideal sinc-function interpolator.

rotate/rotate.c

```
/* Rotate around. */
#include <rsf.h>

int main(int argc, char* argv[])
{
    int n1, n2, i1, i2, k1, k2;
    float x1, x2, c1, c2, cosa, sina, angle;
    float **orig, **rotd;
    char *interp;
    sf_file inp, out;

    /* initialize */
    sf_init(argc,argv);
    inp = sf_input("in");
    out = sf_output("out");

    /* get dimensions from input */
    if (!sf_histint(inp,"n1",&n1)) sf_error("No n1= in inp");
    if (!sf_histint(inp,"n2",&n2)) sf_error("No n2= in inp");
```

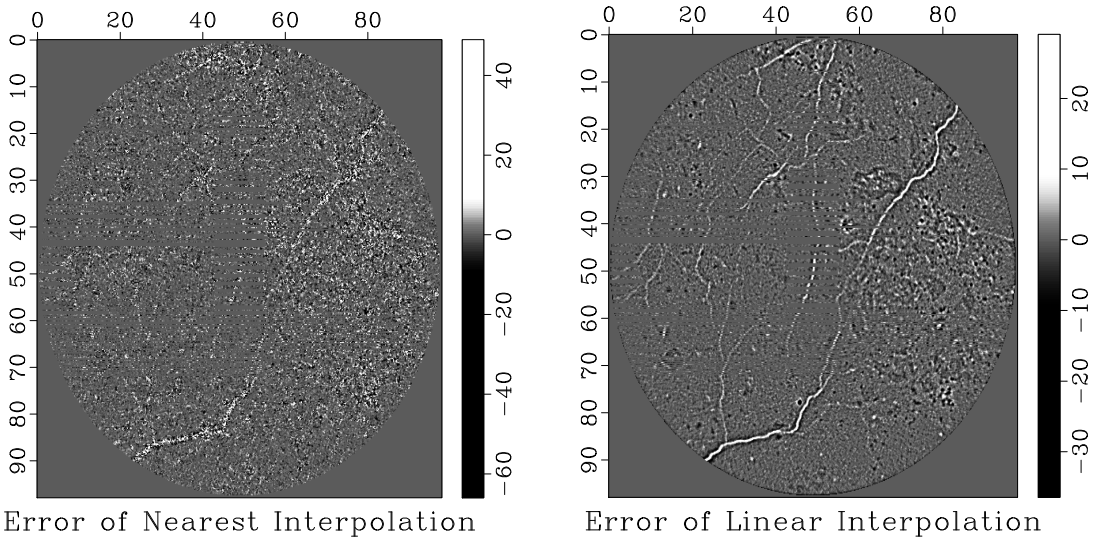


Figure 4.2: Error of different interpolation methods computed after full circle rotation in increments of 20 degrees. (a) Nearest-neighbor interpolation. (b) Bi-linear interpolation. hw4/rotate nearest,linear

```
20
21      /* get parameters from command line */
22      if (!sf_getfloat("angle",&angle)) angle=90.;
23      /* rotation angle */
24
25      if (NULL == (interp = sf_getstring("interp")))
26          interp="nearest";
27      /* [n,l,c] interpolation type */
28
29      /* convert degrees to radians */
30      angle *= SF_PI/180.;
31      cosa = cosf(angle);
32      sina = sinf(angle);
33
34      orig = sf_floatalloc2(n1,n2);
35      rotd = sf_floatalloc2(n1,n2);
36
37      /* read data */
38      sf_floatread(orig[0],n1*n2,inp);
39
40      /* central point */
41      c1 = (n1-1)*0.5;
42      c2 = (n2-1)*0.5;
43
44      for (i2=0; i2 < n2; i2++) {
45          for (i1=0; i1 < n1; i1++) {
```

```
46
47              /* rotated coordinates */
48              x1 = c1+(i1-c1)*cosa-(i2-c2)*sina;
49              x2 = c2+(i1-c1)*sina+(i2-c2)*cosa;
50
51              /* nearest neighbor */
52              k1 = floorf(x1); x1 -= k1;
53              k2 = floorf(x2); x2 -= k2;
54
55              switch(interp[0]) {
56                  case 'n': /* nearest neighbor */
57                      if (x1 > 0.5) k1++;
58                      if (x2 > 0.5) k2++;
59                      if (k1 >=0 && k1 < n1 &&
60                          k2 >=0 && k2 < n2) {
61                          rotd[i2][i1] = orig[k2][k1];
62                      } else {
63                          rotd[i2][i1] = 0.;
64                      }
65                      break;
66                  case 'l': /* bilinear */
67                      if (k1 >=0 && k1 < n1-1 &&
68                          k2 >=0 && k2 < n2-1) {
69                          rotd[i2][i1] =
70                              (1.-x1)*(1.-x2)*orig[k2][k1]   +
71                              x1     *(1.-x2)*orig[k2][k1+1] +
72                              (1.-x1)*x2     *orig[k2+1][k1] +
73                              x1     *x2     *orig[k2+1][k1+1];
74                      } else {
75                          rotd[i2][i1] = 0.;
76                      }
77                      break;
78                  case 'c': /* cubic convolution */
79                      /* !!! ADD CODE !!! */
80                      break;
81                  default:
82                      sf_error("Unknown interpolation %s",
83                               interp);
84                      break;
85              }
86          }
87      }
88
89      /* write result */
90      sf_floatwrite(rotd[0],n1*n2,out);
91
92      exit(0);
93  }
```

rotate/SConstruct

```
from rsf.proj import *

# Download data
Fetch('slice.rsf','ctscan')
Flow('circle','slice','dd type=float')

grey = 'grey wanttitle=n screenratio=1 bias=128 clip=105'

Result('circle',grey)

# Rotate program
program = Program('rotate.c')
rotate = str(program[0])

# Rotate by 90 degrees
Flow('rotate',['circle',rotate],
     './${SOURCES[1]} angle=90 interp=nearest')

Result('rotate',grey)

# Mask for the circle
Flow('mask','circle',
     '''
     put d1=1 o1=-255.5 d2=1 o2=-255.5 |
     math output="sqrt(x1*x1+x2*x2)" |
     mask min=255.5 | dd type=float |
     smooth rect1=3 rect2=3 |
     mask max=0 | dd type=float |
     put d1=0.1914 o1=0 d2=0.1914 o2=0
     ''')

for case in ('nearest','linear'): # !!! MODIFY ME !!!
    new = 'circle'
    rotates = []
    for r in range(18):
        old = new
        new = '%s-circle%d' % (case,r)
        Flow(new,[old,rotate],
             './${SOURCES[1]} angle=20 interp=%s' % case)
        Plot(new,grey)
        rotates.append(new)

    # Movie of rotating circle
    Plot(case,rotates,'Movie',view=1)

```

```
    # Plot error
    Result(case,[new,'circle','mask'],
           '''
           add scale=1,-1 ${SOURCES[1]} |
           add mode=p ${SOURCES[2]} |
           %s bias=0 scalebar=y clip=12
           wanttitle=y title="Error of %s Interpolation"
           ''' % (grey,case.capitalize()))

End()
```

Your task:

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

   ```
   scons view
   ```

   to reproduce the figures on your screen.
3. Additionally, you can run

   ```
   scons nearest.vpl
   ```

   and

   ```
   scons linear.vpl
   ```

   to see movies of incremental slice rotation with different methods.
4. Modify the `rotate.c` program and the `SConstruct` file to implement the cubic convolution interpolation and to compare its results with the two other methods.
5. **EXTRA CREDIT** for implementing an interpolation algorithm, which is more accurate than cubic convolution.

4.4 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 4.3 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

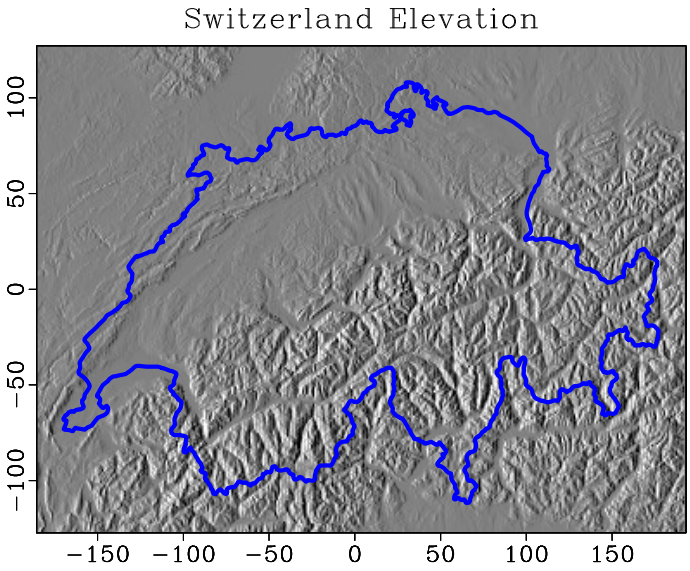


Figure 4.3: Digital elevation map of Switzerland. hw4/rain elev

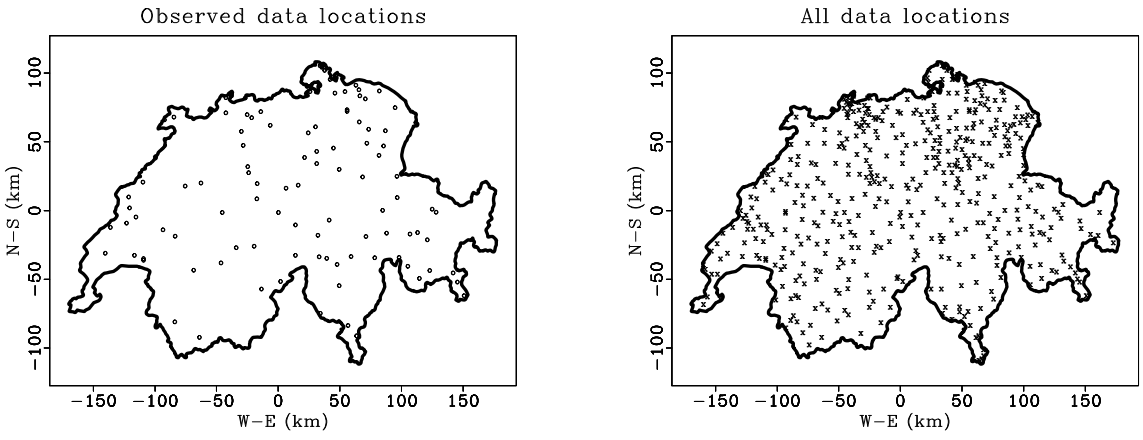


Figure 4.4: Left: locations of weather stations used as input data in the spatial interpolation contest. Right: all weather stations locations. hw4/rain raindata

other measurements using different techniques and to compare the results with the known data. Figure 4.4 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.

4.4.1 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 4.5(a). Does it succeed in hiding the acquisition footprint? Figure 4.5(b) provides a comparison between interpolated and known data values. It also indicates the value of the correlation coefficient.

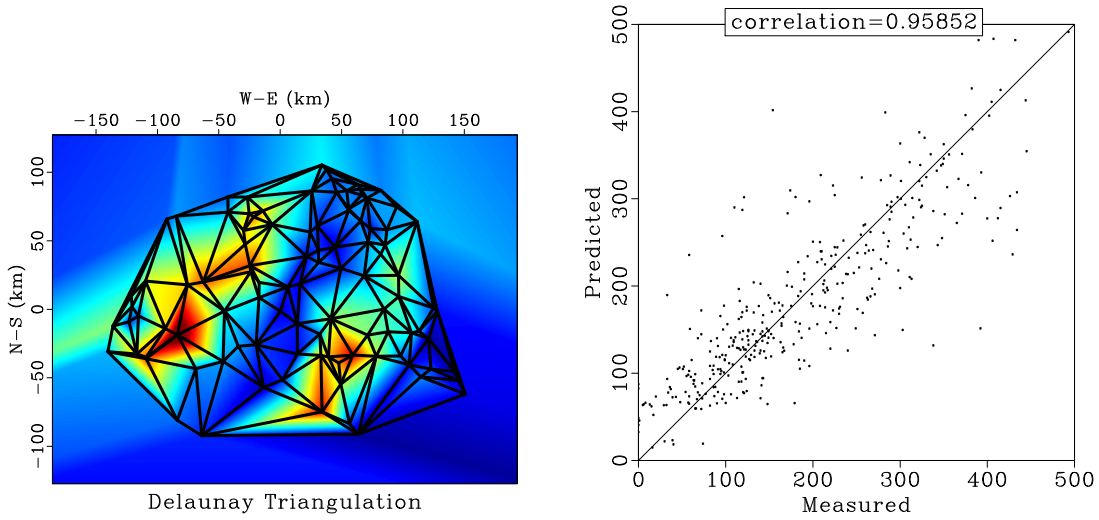


Figure 4.5: (a) Rainfall data interpolated using Delaunay triangulation. (b) Correlation between interpolated and true data values. hw4/rain trian,trian-pred

4.4.2 Gradient regularization

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

$$\min \left(|\mathbf{F}\,\mathbf{m} - \mathbf{d}|^2 + \epsilon^2 |\mathbf{R}\,\mathbf{m}|^2 \right) \ , \tag{4.9}$$

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 4.6 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 4.7.

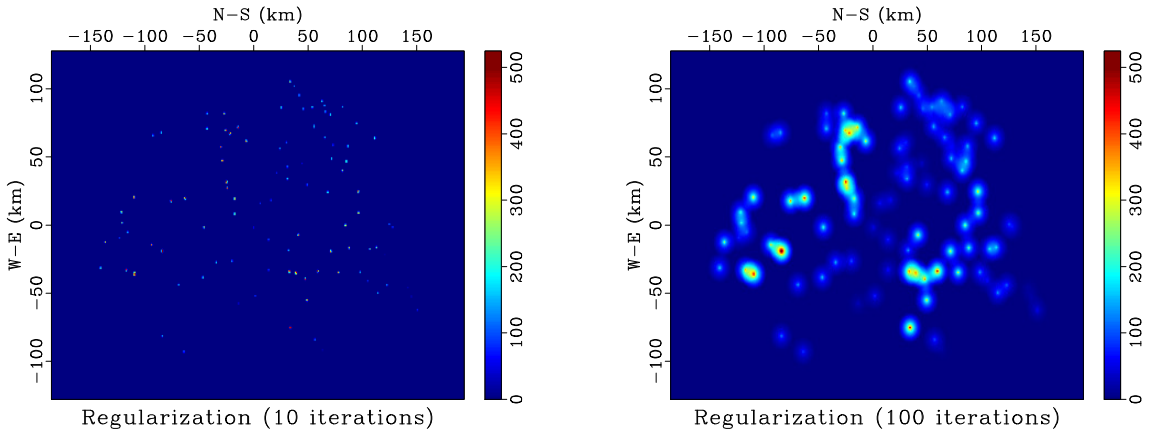


Figure 4.6: Rainfall data interpolated using regularization with the gradient filter. hw4/rain inter0

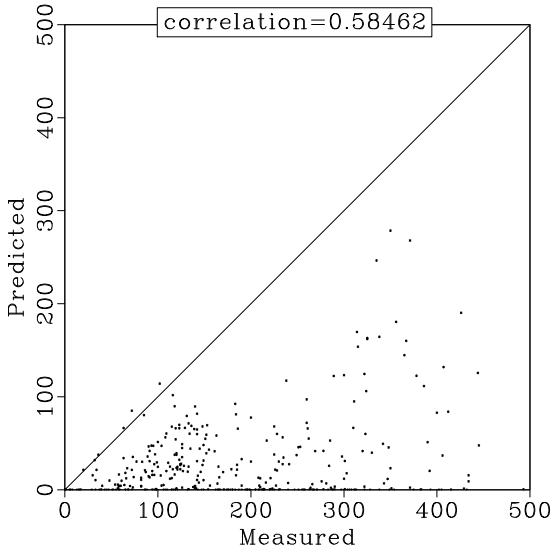


Figure 4.7: Correlation between interpolated and true data values for regularization with 100 iterations. hw4/rain inter0-100-pred

4.4.3 Helical derivative preconditioning

An alternative to the optimization problem (4.9) is the problem of minimizing $|\mathbf{x}|^2 + |\mathbf{r}|^2$ under the constraint

$$\mathbf{F}\,\mathbf{P}\,\mathbf{x} + \epsilon\,\mathbf{r} = \mathbf{d}\ . \tag{4.10}$$

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

$$\mathbf{P}\,\mathbf{P}^T = \left(\mathbf{R}^T\,\mathbf{R}\right)^{-1}\ . \tag{4.11}$$

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

$$L_2(Z_1, Z_2) = 4 - Z_1 - Z_1^{-1} - Z_2 - Z_2^{-1}\ . \tag{4.12}$$

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

$$L_H(Z) = 4 - Z - Z^{-1} - Z^{N_1} - Z^{-N_1}\ , \tag{4.13}$$

and factor it into two minimum-phase parts $L_H(Z) = D(Z)\,D(1/Z)$ using the Wilson-Burg algorithm (Fomel et al., 2003). The factorization is tested in Figure 4.8, where the impulse response of the Laplacian filter gets inverted by recursive filtering (polynomial division) on a helix.

Figure 4.9 shows the interpolation result using conjugate-gradient optimization with equation (4.10) after 10 and 100 iterations. The corresponding correlation analysis is shown in Figure 4.10.

Your task:

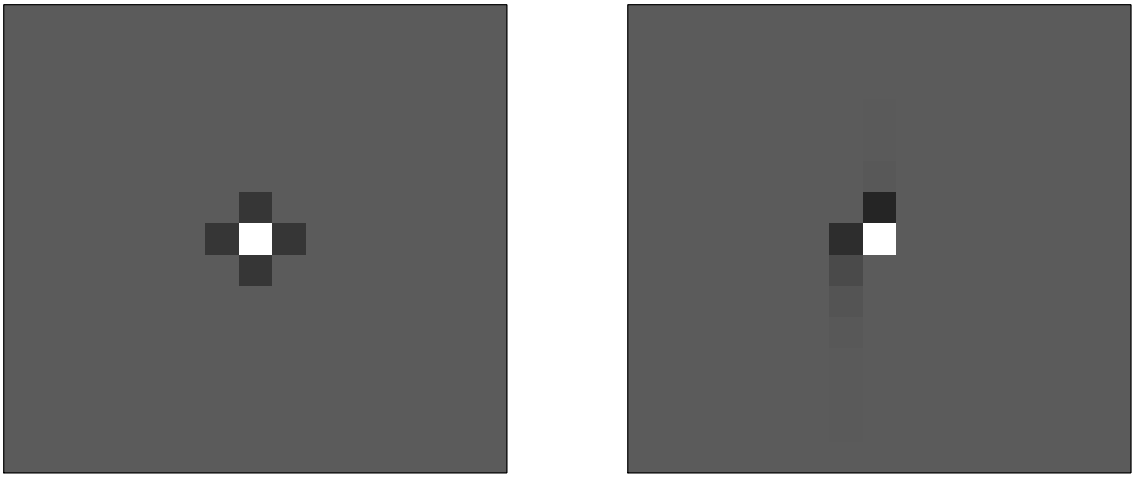
1. Change directory to `hw4/rain`
2. Run

   ```
   scons view
   ```

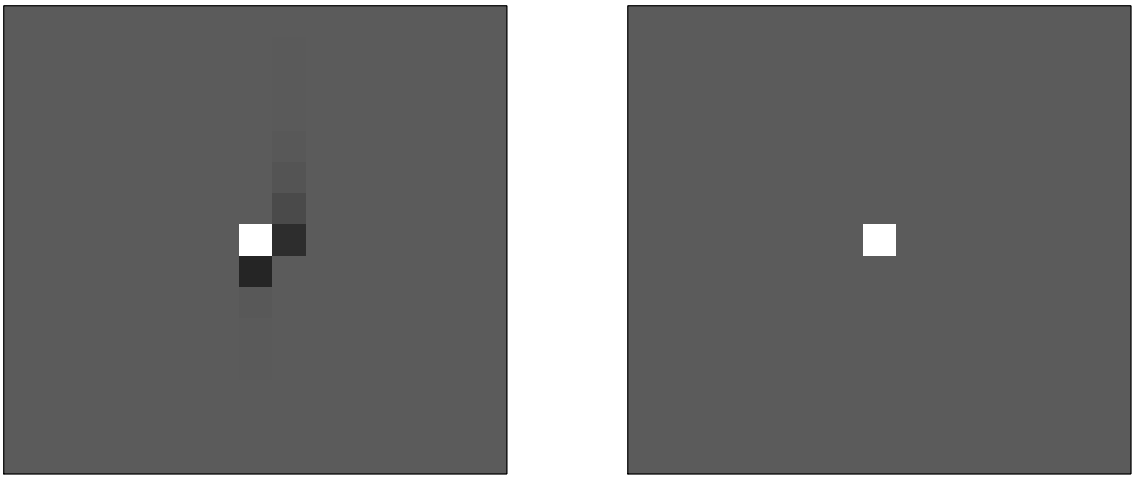
   to reproduce the figures on your screen.
3. Modify the `SConstruct` file to accomplish the following tasks
   (a) Find out the number of conjugate-gradient iterations needed for the regularization method to achieve a result comparable with the preconditioning method.
   (b) Replace the five-point Laplacian filter with the more isotropic nine-point filter

   $$\begin{aligned}\hat{L}_2(Z_1, Z_2) = 20 \quad - \quad & 4\,Z_1 - 4\,Z_1^{-1} - 4\,Z_2 - 4\,Z_2^{-1} \\ & -Z_1\,Z_2 - Z_1\,Z_2^{-1} - Z_2\,Z_1^{-1} - Z_1^{-1}\,Z_2^{-1}\end{aligned} \tag{4.14}$$

   and repeat the experiment.
4. What can you conclude about the three methods used in this comparison?



(a) Laplacian (b) Laplacian/Factor



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

Figure 4.8: 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 $D(Z)\,D(1/Z)$. hw4/rain laplace

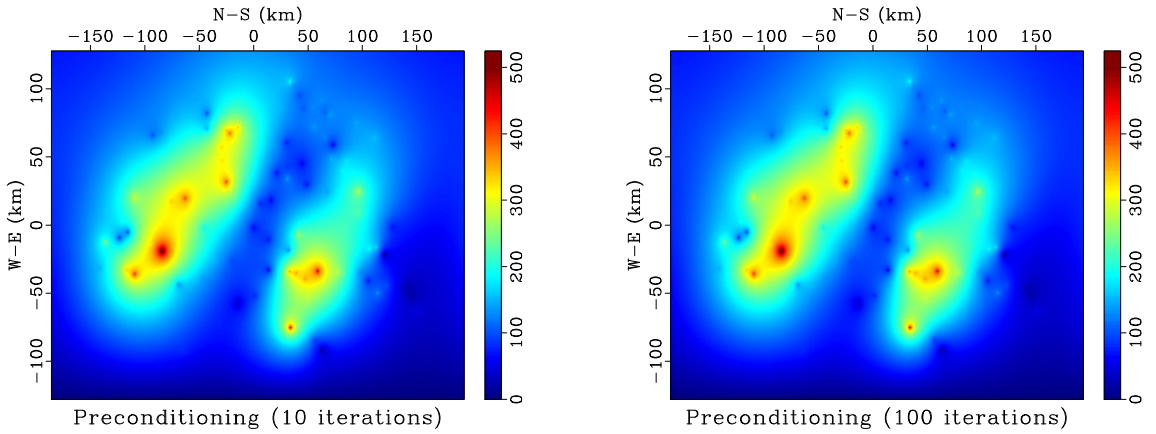


Figure 4.9: Rainfall data interpolated using preconditioning with the inverse helical filter. hw4/rain inter1

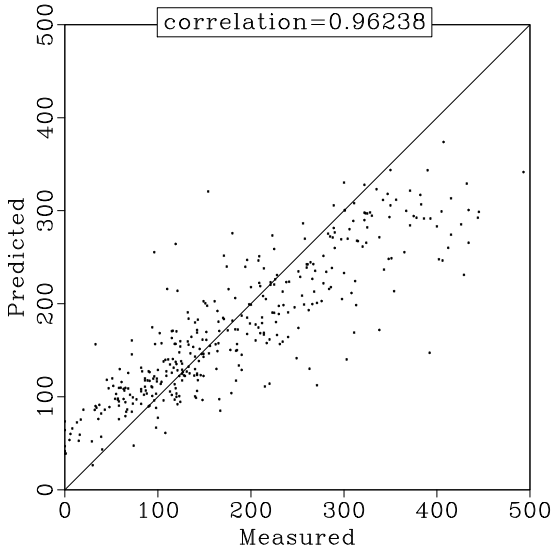


Figure 4.10: Correlation between interpolated and true data values for preconditioning with 100 iterations. hw4/rain inter1-100-pred

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

```
/* Data regulatization by inverse interpolation. */
#include <rsf.h>

static void lint (float x, int n, float* w)
/*< linear interpolation>*/
{
    w[0] = 1.0f - x;
    w[1] = x;
}

static void regrid( int dim          /* dimensions */,
                    const int* nold /* old size [dim] */,
                    const int* nnew /* new size [dim] */,
                    sf_filter aa     /* filter */)
/*< change data size >*/
{
    int i, h0, h1, h, ii[SF_MAX_DIM];

    for (i=0; i < dim; i++) {
        ii[i] = nold[i]/2-1;
    }

    h0 = sf_cart2line( dim, nold, ii);
    h1 = sf_cart2line( dim, nnew, ii);
    for (i=0; i < aa->nh; i++) {
        h = aa->lag[i] + h0;
        sf_line2cart( dim, nold, h, ii);
        aa->lag[i] = sf_cart2line( dim, nnew, ii) - h1;
    }
}

int main (int argc, char* argv[])
{
    int id, nd, nm, nx, ny, na, ia, niter, three, n[2], m[2];
    float *mm, *dd, **xy;
    float x0, y0, dx, dy, a0, eps;
    char *lagfile;
    bool prec;
    sf_filter aa;
    sf_file in, out, flt, lag;

```

```
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);
```

```
    exit(0);
}
```

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')

Flow('alldata','alldata.hh',
     'window n1=2 | dd type=complex form=native | window')
Plot('alldata',
     '''
     graph symbol=x symbolsz=4
     title="All data locations" plotcol=7
     ''' + box)
Plot('data','alldata border','Overlay')

Flow('obs','obsdata.hh',
     'window n1=2 | dd type=complex form=native | window')
Plot('obs',
     '''
     graph symbol=o symbolsz=4
     title="Observed data locations" plotcol=5
     ''' + box)
Plot('obsdata','obs border','Overlay')

Result('raindata','obsdata data','SideBySideIso')

Flow('coord','coord.hh','dd form=native')
Flow('obsdata','obsdata.hh','dd form=native')

# Triangulation
################
Flow('trian edges','obsdata elev',
     'tri2reg pattern=${SOURCES[1]} edgeout=${TARGETS[1]}')
Plot('edges',
     '''
     graph plotcol=7 plotfat=8
     wanttitle=n wantaxis=n
     ''' + box)
Plot('trian',
     '''
     grey yreverse=n transp=n allpos=y
     color=j clip=500 title="Delaunay Triangulation"
     label1="W-E (km)" label2="N-S (km)"
     ''' + box)
Result('trian','trian edges','Overlay')

# Laplacian filter
##################

# !!! CHANGE BELOW !!!
Flow('lag.asc',None,
     '''
```

```
92        echo 1 100 n1=2 n=100,100
93        data_format=ascii_int in=$TARGET
94        ''')
95  Flow('lag','lag.asc','dd form=native')
96
97  # !!! CHANGE BELOW !!!
98  Flow('flt.asc','lag',
99        '''
100       echo -1 -1 a0=2 n1=2 lag=$SOURCE
101       data_format=ascii_float in=$TARGET
102       ''',stdin=0)
103 Flow('flt','flt.asc','dd form=native')
104
105 # Spectral factorization on a helix
106 Flow('lapflt laplag','flt',
107       'wilson eps=1e-4 lagout=${TARGETS[1]}')
108
109 def plotfilt(title):
110     return '''
111     grey wantaxis=n title="%s" pclip=100
112     crowd=0.85 screenratio=1
113     ''' % title
114
115 # Filter impulse response
116 Flow('spike',None,'spike n1=15 n2=15 k1=8 k2=8')
117 Flow('imp0','spike flt','helicon filt=${SOURCES[1]} adj=0')
118 Flow('imp1','spike flt','helicon filt=${SOURCES[1]} adj=1')
119 Flow('imp','imp0 imp1','add ${SOURCES[1]}')
120 Plot('imp',plotfilt('(a) Laplacian'))
121
122 # Test factorization
123 Flow('fac0','imp lapflt',
124       'helicon filt=${SOURCES[1]} adj=0 div=1')
125 Flow('fac1','imp lapflt',
126       'helicon filt=${SOURCES[1]} adj=1 div=1')
127 Plot('fac0',plotfilt('(b) Laplacian/Factor'))
128 Plot('fac1',plotfilt('(c) Laplacian/Factor\''))
129 Flow('fac','fac0 lapflt',
130       'helicon filt=${SOURCES[1]} adj=1 div=1')
131 Plot('fac',plotfilt('(d) Laplacian/Factor/Factor\''))
132
133 Result('laplace','imp fac0 fac1 fac','TwoRows',
134        vppen='gridsize=5,5 xsize=11 ysize=11')
135
136 # Maximum number of iterations
137 ##############
138 nmax = 100 # CHANGE ME!!!
139
```

```
# Inverse interpolation program
program = Program('invint.c')
invint = str(program[0])

for prec in range(2):
    iters = []
    inter = 'inter%d' % prec
    for niter in [10,nmax]:
        it = 'inter%d-%d' % (prec,niter)
        Flow(it,['obsdata',invint,'lapflt'],
             '''
             ./${SOURCES[1]} prec=%d niter=%d
             filt=${SOURCES[2]}
             nx=376 ny=253 eps=0.01
             dx=1.00997 dy=1.00997
             x0=-185.556 y0=-127.262
             ''' % (prec,niter))
        Plot(it,
             '''
             grey scalebar=y yreverse=n transp=n allpos=y
             minval=0 maxval=525 color=j clip=500
             title="%s (%d iterations)"
             ''' % (('Regularization',
                     'Preconditioning')[prec],niter))
        iters.append(it)
    Result(inter,iters,'SideBySideIso')

# 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','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]} |
```

```
188             graph symbol="*" wanttitle=n
189             screenratio=1 min1=0 max1=500 min2=0 max2=500
190             label1=Measured label2=Predicted
191             ''')
192
193     num = case+'-num'
194     den = case+'-den'
195     cor = case+'-cor'
196
197     Flow(num,['predict',pred],
198             'add mode=p ${SOURCES[1]} | stack axis=1 norm=n')
199     Flow(den,pred,'add mode=p $SOURCE | stack axis=1 norm=n')
200     Flow(cor+'.asc',[num,den,'norm'],
201             '''
202             math a1=${SOURCES[1]} a2=${SOURCES[2]}
203             output="input/sqrt(a1*a2)" |
204             dd form=ascii --out=$TARGET
205             format="label=correlation=%7.5g"
206             ''',stdout=0)
207     Plot(cor,cor+'.asc',
208             'box x0=5.5 y0=9 xt=0 par=$SOURCE',stdin=0)
209
210     Result(pred,[pred,'line',cor],'Overlay')
211
212 End()
```

4.5 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 Keys'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.

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.

Keys, R. G., 1981, Cubic convolution interpolation for digital image processing: IEEE Transactions on Acoustics, Speech, and Signal Processing, **ASSP-29**, 1153–1160.