# Homework 2

*Sir Charles Antony Richard Hoare*


## ABSTRACT

This homework has four parts.

1. Theoretical and programming questions related to data attributes and digital convolution.
2. Measuring the performance of sorting algorithms.
3. Analyzing a digital elevation map by applying a running average filter.
4. Analyzing a digital elevation map by applying derivative filters.


## 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.

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
```

## DATA ATTRIBUTES AND DIGITAL CONVOLUTION

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

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

   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 matrix in equation (2) represents a convolution operator with zero boundary conditions.

$$\mathbf{F} = \begin{bmatrix} f_1 & f_0 & 0 & 0 & 0 & 0 \\ f_2 & f_1 & f_0 & 0 & 0 & 0 \\ f_3 & f_2 & f_1 & f_0 & 0 & 0 \\ 0 & f_3 & f_2 & f_1 & f_0 & 0 \\ 0 & 0 & f_3 & f_2 & f_1 & f_0 \\ 0 & 0 & 0 & f_3 & f_2 & f_1 \end{bmatrix} . \tag{2}$$

   The operator is implemented in the C function `hw2/conv.c`.

hw2/conv.c

```
15 void conv_lop (bool adj, bool add,
16                int nx, int ny, float* xx, float* yy)
17 /*< linear operator >*/
18 {
19     int f, x, y, x0, x1;
20
21     assert (ny == nx);
22     sf_adjnull (adj, add, nx, ny, xx, yy);
23
24     for (f=0; f < nf; f++) {
25         x0 = SF_MAX(0,1-f);
26         x1 = SF_MIN(nx,nx+1-f);
27         for (x = x0; x < x1; x++) {
28             if( adj) {
29                 /* add code */
30             } else {
31                 yy[x+f-1] += xx[x] * ff[f];
32             }
33         }
34     }
35 }
```

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

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

3. The C code in `hw2/filter.c` implements a recursive filtering operator.

hw2/filter.c

```
13 void filter_lop (bool adj, bool add,
14                  int nx, int ny, float* xx, float* yy)
15 /*< linear operator >*/
16 {
17     int i;
18     float t;
19
20     assert (ny == nx);
21     sf_adjnull (adj, add, nx, ny, xx, yy);
22
23     if (adj) {
24         /* add code */
25     } else {
26         t = a*xx[0];
27         yy[0] += t;
28         for (i = 1; i < nx; i++) {
29             t = a*xx[i] + b*xx[i-1] + c*t;
30             yy[i] += t;
31         }
32     }
33 }
```

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

(b) Add code for the adjoint operator.

## 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
   ```

   to compute the cost of slow sorting experimentally. The output is shown in Figure 1.

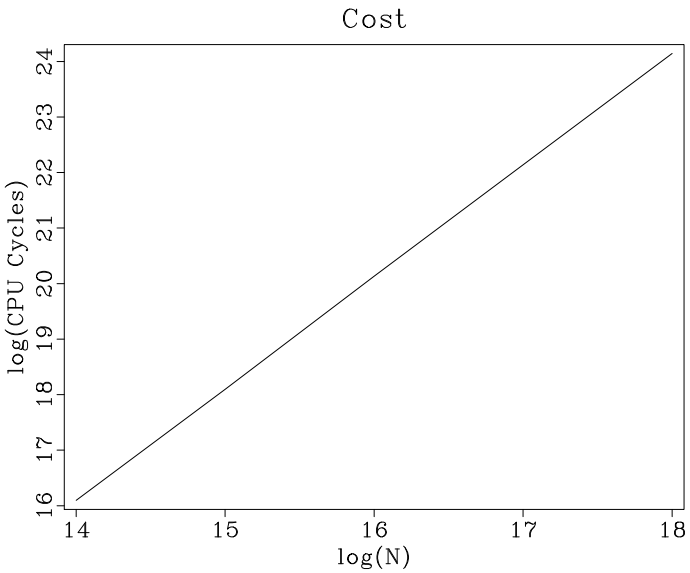


Figure 1: Experimental cost of slow sorting. The logarithm of the cost is shown against the logarithm of the data size.

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

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

sorting/sorting.c

```
#include <time.h>
#include <rsf.h>

static int na, nmovie=0;
static float *arr;
static sf_file movie;

static void write_movie(void)
{
    if (NULL != movie)
        sf_floatwrite(arr,na,movie);
    nmovie++;
}

static void slow_sort(int n, float* list)
{
    int k, k2;
    float item1, item2;

    for (k=0; k < n; k++) {
        write_movie();

        item1 = list[k];
        /* assume everything up to k is sorted */
        for (k2=k; k2 > 0; k2--) {
            item2 = list[k2-1];
            if (item1 >= item2) break;
            list[k2]   = item2;
        }
        list[k2] = item1;
    }
}

static void quick_sort(int n, float* list)
{
    int l, r;
    float ll, pivot;

    if (n <= 1) return;

    write_movie();

    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
101     /* get data */
102     arr = sf_floatalloc(na);
103     sf_floatread(arr,na,in);
104
105     /* sort */
106     start = clock();
107     if ('q'==type[0]) {
108         quick_sort(na, arr);
109     } else {
110         slow_sort(na, arr);
111     }
112     end = clock();
113
114     /* CPU cycles */
115     cycles = end - start;
116     sf_floatwrite(&cycles,1,cost);
117
118     while (nmovie < na+1) write_movie();
119
120     exit(0);
121 }
```

sorting/SConstruct

```
1  from rsf.proj import *
2
3  # Generate random data
4  ######################
5  Flow('rand',None,
6       'spike n1=524288 | noise rep=y type=n seed=2012')
7
8  prog = Program('sorting.c')
9
```

```
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()
```

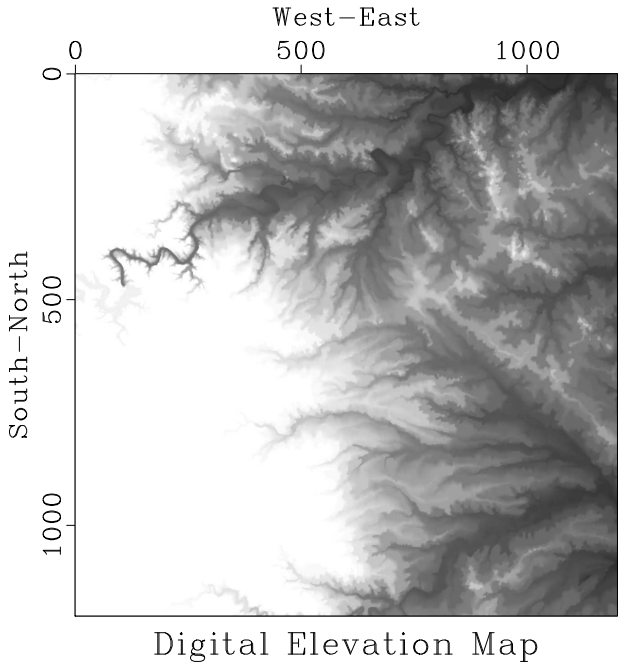


Figure 2: Digital elevation map of the west Austin area.

## RUNNING MEDIAN AND RUNNING MEAN FILTERS

In this part of the homework, we will analyze the digital elevation map of the West Austin Area, shown in Figure 2. Our task is to 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 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;

    for (k=0; k < n; k++) {
        item1 = list[k];

        /* assume everything up to k is sorted */
        for (k2=k; k2 > 0; k2--) {
```

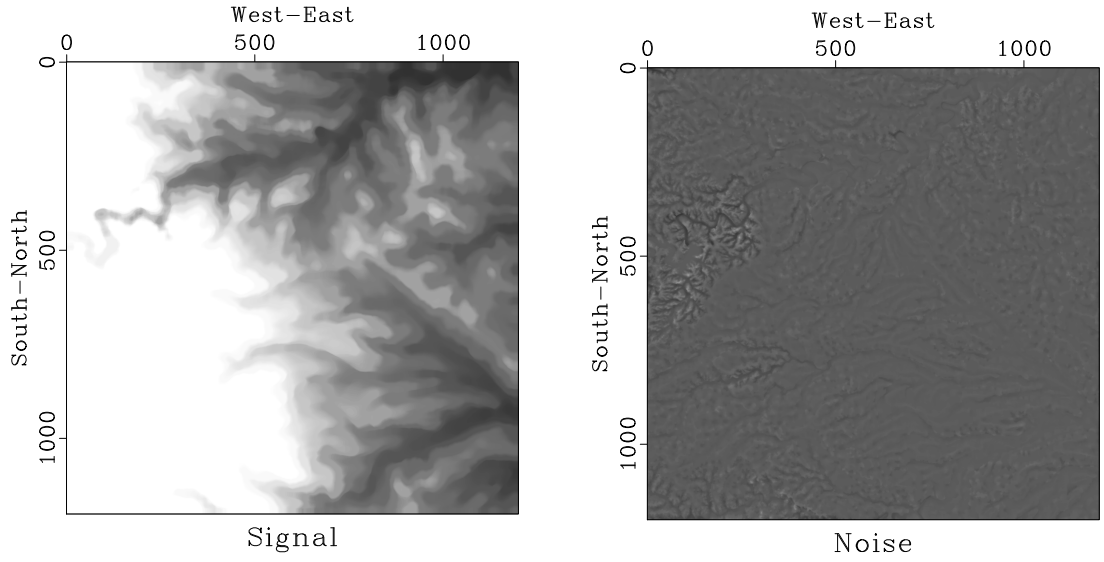


Figure 3: Data separated into signal (a) and noise (b) by applying a running median filter.

```
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
58      for (i3=0; i3 < n3; i3++) {
59
60          /* read data plane */
61          sf_floatread(data[0],n1*n2,in);
62
63          for (i2=0; i2 < n2; i2++) {
64              s2 = SF_MAX(0,SF_MIN(n2-w2,i2-w2/2-1));
65              for (i1=0; i1 < n1; i1++) {
66                  s1 = SF_MAX(0,SF_MIN(n1-w1,i1-w1/2-1));
67
68                  /* copy window */
69                  for (j2=0; j2 < w2; j2++) {
70                      for (j1=0; j1 < w1; j1++) {
71                          win[j2][j1] = data[s2+j2][s1+j1];
72                      }}
73
74                  switch (what[0]) {
75                      case 'f': /* fast median */
76                          signal[i2][i1] =
77                              sf_quantile(nw/2,nw,win[0]);
78                          break;
79                      case 's': /* slow median */
80                          signal[i2][i1] =
81                              slow_median(nw,win[0]);
82                          break;
83                      case 'm': /* mean */
84                      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. You may 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')

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

import sys

if sys.platform=='darwin':
     gtime = WhereIs('gtime')
else:
     gtime = WhereIs('gtime') or WhereIs('time')

if not gtime:
    print '''
For computing CPU time, please install GNU time!
    '''

# slow or fast
for case in ('fast','slow'):

    ts = []
    ws = []

    time = 'time-' + case
    wind = 'wind-' + case
```

```
54
55
56
57      # loop over window size
58      for w in range(3,16,2):
59          itime = '%s-%d' % (time,w)
60          ts.append(itime)
61
62          iwind = '%s-%d' % (wind,w)
63          ws.append(iwind)
64
65          # measure CPU time
66
67
68
69          Flow(iwind,None,'spike n1=1 mag=%d' % (w*w))
70          Flow(itime,'dem %s' % run[0],
71               '''
72               ( (%s -f "%%S %%U"
73               ./${SOURCES[1]} < ${SOURCES[0]}
74                w1=%d w2=%d what=%s > /dev/null ) 2>&1 )
75                > time.out &&
76               (tail -1 time.out;
77                echo in=time0.asc n1=2 data_format=ascii_float)
78                > time0.asc &&
79               dd form=native < time0.asc | stack axis=1 norm=n
80               > $TARGET &&
81               /bin/rm time0.asc time.out
82               ''' % (gtime,w,w,case),stdin=0,stdout=-1)
83
84      Flow(time,ts,'cat axis=1 ${SOURCES[1:%d]}' % len(ts))
85      Flow(wind,ws,'cat axis=1 ${SOURCES[1:%d]}' % len(ws))
86
87      # complex numbers for plotting
88      Flow('c'+time,[wind,time],
89           '''
90           cat axis=2 ${SOURCES[1]} |
91           transp |
92           dd type=complex
93           ''')
94
95  # Display CPU time
96  Plot   ('time','ctime-fast ctime-slow',
97          '''
98          cat axis=1 ${SOURCES[1]} | transp |
```

```
99          graph dash=0,1 wanttitle=n
100         label2="CPU Time" unit2=s
101         label1="Window Size" unit1=
102         '' ',view=1)
103
104 End()
```

## DERIVATIVE FILTERS

In this part of the assignment, we return to the digital elevation map of the of the of the Mount St. Helens area, shown in Figure 4.

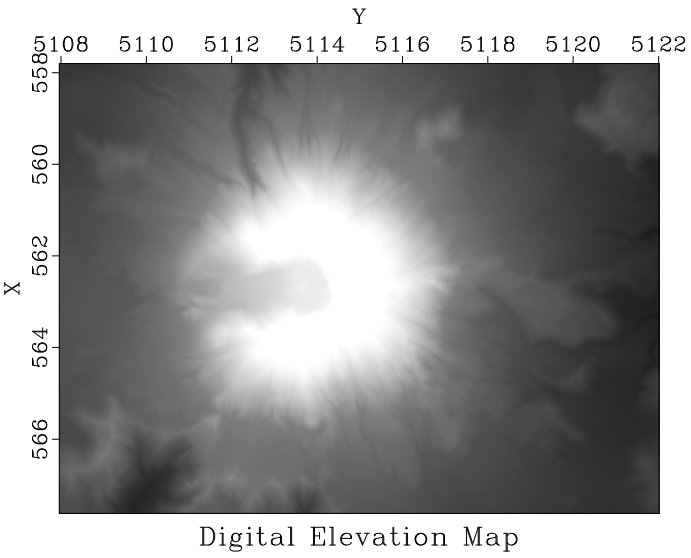


Figure 4: Digital elevation map of Mount St. Helens area.

Figure 5 shows a directional derivative, a digital approximation to

$$\cos\alpha \, \frac{\partial}{\partial x_1} + \sin\alpha \, \frac{\partial}{\partial x_2} \ , \tag{3}$$

applied to the data. A directional derivative highlights the structure of the mountain as if illuminating it with a light source.

Figure 6 shows an application of *helical derivative*, a filter designed by spectral factorization of the Laplacian filter

$$L(Z_1, Z_2) = 4 - Z_1 - 1/Z_1 - Z_2 - 1/Z_2 \ . \tag{4}$$

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

$$L_H(Z) = 4 - Z - Z^{-1} - Z^{N_1} - Z^{-N_1} \ , \tag{5}$$

and factor it into two minimum-phase parts $L_H(Z) = D(Z)\, D(1/Z)$ using the Wilson-Burg algorithm. The helical derivative $D(Z)$ enhances the image but does not have a preferential direction.

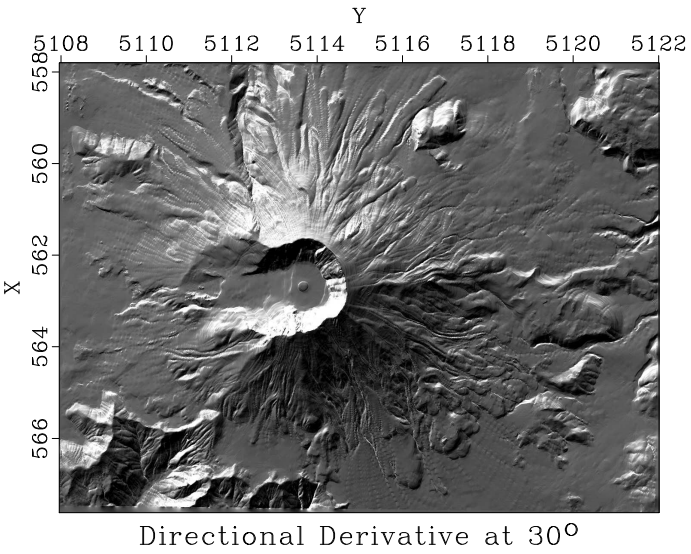


Figure 5: Directional derivative of elevation.

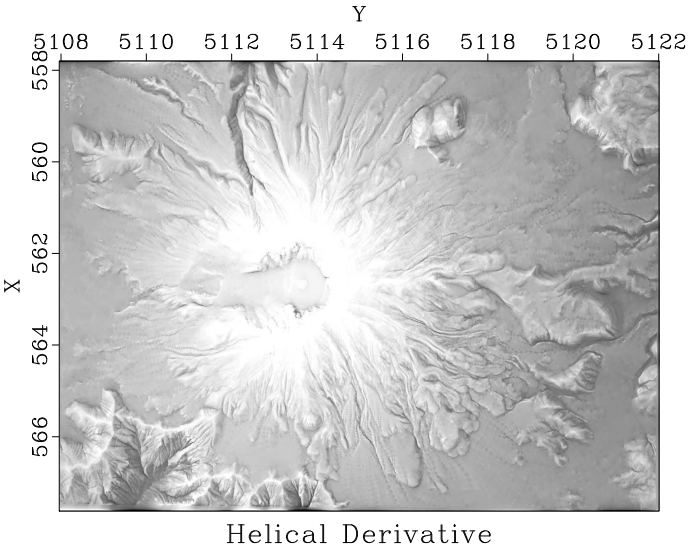


Figure 6: Helix derivative of elevation.

1. Change directory to `hw2/helix`.

2. Run

   ```
   scons view
   ```

   to reproduce the figures on your screen.

3. Edit the `SConstruct` file. Find the parameter that corresponds to $\alpha$ in equation (3) and try to modify it until you create the most interesting image. After changing the parameter, you can view the result by running

   ```
   scons der.view
   ```

4. **EXTRA CREDIT** for suggesting and implementing a method for finding optimal $\alpha$ automatically.

5. A more accurate version of the Laplacian filter is

$$\begin{aligned}\hat{L}_2(Z_1, Z_2) = 20 \;\; - \;\; & 4\,Z_1 - 4\,Z_1^{-1} - 4\,Z_2 - 4\,Z_2^{-1} \\ & -Z_1\,Z_2 - Z_1\,Z_2^{-1} - Z_2\,Z_1^{-1} - Z_1^{-1}\,Z_2^{-1}\;. \end{aligned} \tag{6}$$

   Modify the `SConstruct` file to use filter (6) instead of (4).

6. **EXTRA CREDIT** An even more accurate version of the Laplacian filter involves polynomial division in addition to polynomial multiplication. Find the coefficient for the polynomial division and modify the `SConstruct` file to implement the corresponding helical derivative.

helix/SConstruct

```
from rsf.proj import *
import math

# Download data
txt = 'st-helens_after.txt'
Fetch(txt,'data',
      server='https://raw.githubusercontent.com',
      top='agile-geoscience/notebooks/master')
Flow('data.asc',txt,'/usr/bin/tail -n +6')

# Convert to RSF format
Flow('data','data.asc',
     '''
     echo in=$SOURCE data_format=ascii_float
     label=Elevation unit=m
     n1=979 o1=557.805 d1=0.010030675 label1=X
```

```
17      n2=1400 o2=5107.965 d2=0.010035740 label2=Y |
18      dd form=native |
19      clip2 lower=0 | lapfill grad=y niter=200
20      ''')
21
22  Result('data','grey title="Digital Elevation Map" allpos=y')
23
24  # Vertical and horizontal derivatives
25  Flow('der1','data','igrad')
26  Flow('der2','data','transp | igrad | transp')
27
28  # !!! CHANGE BELOW !!!
29  alpha=30
30
31  # Directional derivative
32  Flow('der','der1 der2',
33       '''
34       add ${SOURCES[1]} scale=%g,%g
35       ''' % (math.cos(alpha*math.pi/180),
36              math.sin(alpha*math.pi/180)))
37
38  Result('der',
39         '''
40         grey title="Directional Derivative at %g\^o\_"
41         ''' % alpha)
42
43  # Laplacian filter on a helix
44
45  Flow('slag0.asc',None,
46       '''echo 1 1000 n=1000,1000 n1=2 in=$TARGET
47       data_format=ascii_int
48       ''')
49  Flow('slag','slag0.asc','dd form=native')
50
51  Flow('ss0.asc','slag',
52       '''echo -1 -1 a0=2 n1=2
53       lag=$SOURCE in=$TARGET data_format=ascii_float''')
54  Flow('ss','ss0.asc','dd form=native')
55
56  # Wilson-Burg factorization
57
58  na=50 # filter length
59
60  lags = range(1,na+1) + range(1001-na,1001)
61
```

```
Flow('alag0.asc',None,
     '''echo %s n=1000,1000 n1=%d in=$TARGET
     data_format=ascii_int
     ''' % (' '.join(map(str,lags)),2*na))
Flow('alag','alag0.asc','dd form=native')

Flow('hflt hlag','ss alag',
     'wilson lagin=${SOURCES[1]} lagout=${TARGETS[1]}')

# Helical derivative

Flow('helder','data hflt hlag','helicon filt=${SOURCES[1]}')
Result('helder','grey title="Helical Derivative" ')

End()
```

## 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.