# Homework 2

*Sir Charles Antony Richard Hoare*

## ABSTRACT

This homework has three parts.

1. Theoretical and programming questions related to data attributes and digital convolution.
2. Analyzing a digital elevation map by applying a running average filter.
3. 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`

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

```
svn co https://github.com/ahay/src/trunk/book/geo384h/hw2
```

## DATA ATTRIBUTES AND 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

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

Suppose that the data vector **a** contains 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. Consider the parabolic filter $F(Z)$ defines as

$$F(Z) = 1 + 4Z + 9Z^2 + \ldots + N^2 Z^{N-1}\,. \tag{2}$$

   (a) Show that this filter can be implemented using recursive filtering (polynomial division).

   (b) What is the advantage of recursive filtering? Does it depend on $N$?

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?

## RUNNING MEDIAN AND RUNNING MEAN FILTERS

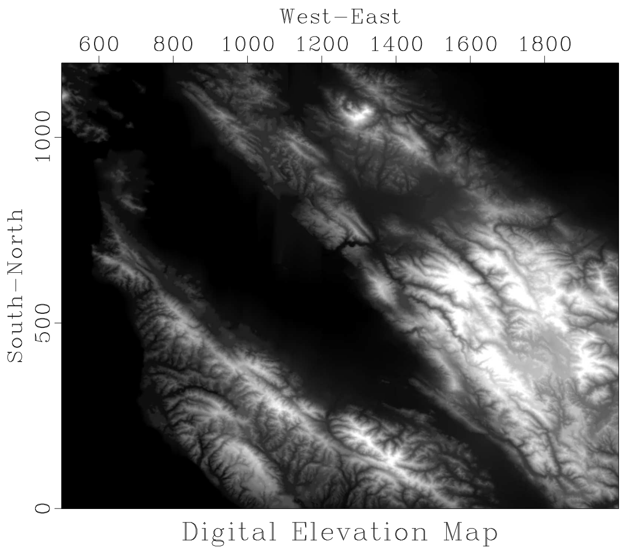

Figure 1: Digital elevation map of the San Francisco Bay Area. running/ bay

We return the digital elevation map of the San Francisco Bay Area, shown in Figure **??**.

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. Running median effectively smooths the data by removing local outliers.

The algorithm is implemented in programs below.

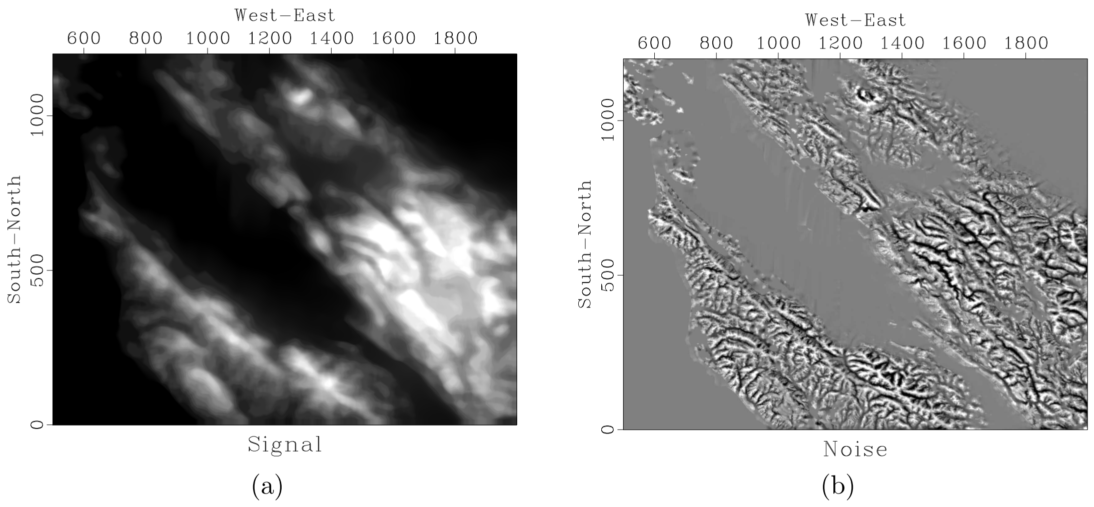

Figure 2: Data separated into signal (a) and noise (b) by applying a running median filter. running/ ave,res

running/run.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--) {
            item2 = list[k2-1];
            if (item1 >= item2) break;
            list[k2]   = item2;
        }
        list[k2] = item1;
    }

    return list[n/2];
}

```

```
int main(int argc, char* argv[])
{
    int w1, w2, nw, s1,s2, j1,j2, i1,i2,i3, n1,n2,n3;
    char *how;
    float **data, **signal, **win;
    sf_file in, out;

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

    /* get data dimensions */
    if (!sf_histint(in,"n1",&n1)) sf_error("No n1=");
    if (!sf_histint(in,"n2",&n2)) sf_error("No n2=");
    n3 = sf_leftsize(in,2);

    /* input and output */
    data = sf_floatalloc2(n1,n2);
    signal = sf_floatalloc2(n1,n2);

    if (!sf_getint("w1",&w1)) w1=5;
    if (!sf_getint("w2",&w2)) w2=5;
    /* sliding window width */

    nw = w1*w2;
    win = sf_floatalloc2(w1,w2);

    how = sf_getstring("how");
    /* what to compute
       (fast median, slow median, mean) */
    if (NULL == how) how="fast";

    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 (how[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 */
                        /* !!! ADD CODE !!! */
                        break;
                    default:
                        sf_error("Unknown method \"%s\"",how);
                        break;
                }
            }
        }

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

    exit(0);
}
```

running/run.py

```
#!/usr/bin/env python

import sys
import numpy as np
import m8r

def slow_median(data):
    'find median by slow sorting, changes data'

    n = len(data)
    for k in range(n):
        item1 = data[k]

        # assume everything up to k is sorted
        for k2 in range(k,-1,-1):
            item2 = data[k2-1]
            if item1 >= item2:
                break
            data[k2] = item2

        data[k2] = item1

    return data[n/2]

# initialization
par = m8r.Par()
inp = m8r.Input()
out = m8r.Output()

# get data dimensions
n1 = inp.int('n1')
n2 = inp.int('n2')
n3 = inp.leftsize(2)

# input and output
data = np.zeros([n2,n1],'f')
signal = np.zeros([n2,n1],'f')

# sliding window
w1 = par.int('w1',5)
w2 = par.int('w2',5)

nw = w1*w2
win = np.zeros([w2,w1],'f')

how = par.string('how','fast')
# what to compute
# (fast median, slow median, mean)

for i3 in range(n3):
    # read data plane
    inp.read(data)

    for i2 in range(n2):
        sys.stderr.write("\r%d of %d" % (i2,n2))
        s2 = max(0,min(n2-w2,i2-w2/2-1))
        for i1 in range(n1):
            s1 = max(0,min(n1-w1,i1-w1/2-1))

            # copy window
            win = data[s2:s2+w2,s1:s1+w1]

            if how[0] == 'f': # fast median
                signal[i2,i1] = np.median(win)
            elif how[0] == 's': # slow median
                signal[i2,i1] = slow_median(win.flatten())
            elif how[0] == 'm': # mean
                # !!! ADD CODE !!!
                pass
            else:
                sys.stderr.write(
                    "Unknown method \"%s\"\n" % how)
                sys.exit(1)
    sys.stderr.write("\n")
    # write out
    out.write(signal)

sys.exit(0)
```

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

   ```
   scons view
   ```

   to reproduce the figures on your screen.
3. Modify the `run.c` program (alternatively, `run.py` script) 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 `run.c` (or `run.py`) and the fast median algorithm. 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('bay.h','bay')

# Convert format
Flow('bay','bay.h',
     '''
     dd form=native |
     window f2=500 n2=1500
     ''')

# Display
def plot(title):
    return '''
    grey allpos=y title="%s" yreverse=n
    label1=South-North label2=West-East
    screenratio=0.8
    ''' % title

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

# Program for running average
run = Program('run.c')

# COMMENT ABOVE AND UNCOMMENT BELOW IF YOU WANT TO USE PYTHON
# run = Command('run.exe','run.py','cp $SOURCE $TARGET')
# AddPostAction(run,Chmod(run,0o755))

w = 30

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

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

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

import sys
```

```

if sys.platform=='darwin':
      gtime = WhereIs('gtime')
      if not gtime:
            print("For computing CPU time, install gtime.")
else:
      gtime = WhereIs('gtime') or WhereIs('time')

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

     ts = []
     ws = []

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

     # loop over window size
     for w in range(3,16,2):
          itime = '%s-%d' % (time,w)
          ts.append(itime)

          iwind = '%s-%d' % (wind,w)
          ws.append(iwind)

          # measure CPU time
          Flow(iwind,None,'spike n1=1 mag=%d' % (w*w))
          Flow(itime,'bay %s' % run[0],
                '''
                ( (%s -f "%%S %%U"
                ./${SOURCES[1]} < ${SOURCES[0]}
                 w1=%d w2=%d how=%s > /dev/null ) 2>&1 )
                 > time.out &&
                (tail -1 time.out;
                 echo in=time0.asc n1=2 data_format=ascii_float)
                 > time0.asc &&
                dd form=native < time0.asc | stack axis=1 norm=n
                > $TARGET &&
                /bin/rm time0.asc time.out
                ''' % (gtime,w,w,case),stdin=0,stdout=-1)

     Flow(time,ts,'cat axis=1 ${SOURCES[1:%d]}' % len(ts))
     Flow(wind,ws,'cat axis=1 ${SOURCES[1:%d]}' % len(ws))

     # complex numbers for plotting
```

```
    Flow('c'+time,[wind,time],
         '''
         cat axis=2 ${SOURCES[1]} |
         transp |
         dd type=complex
         ''')

# Display CPU time
Plot('time','ctime-fast ctime-slow',
     '''
     cat axis=1 ${SOURCES[1]} | transp |
     graph dash=0,1 wanttitle=n
     label2="CPU Time" unit2=s
     label1="Window Size" unit1=
     ''',view=1)

End()
```

## DERIVATIVE FILTERS

In this part of the assignment, we will use a digital elevation map of the Mount St. Helens area, shown in Figure 3.

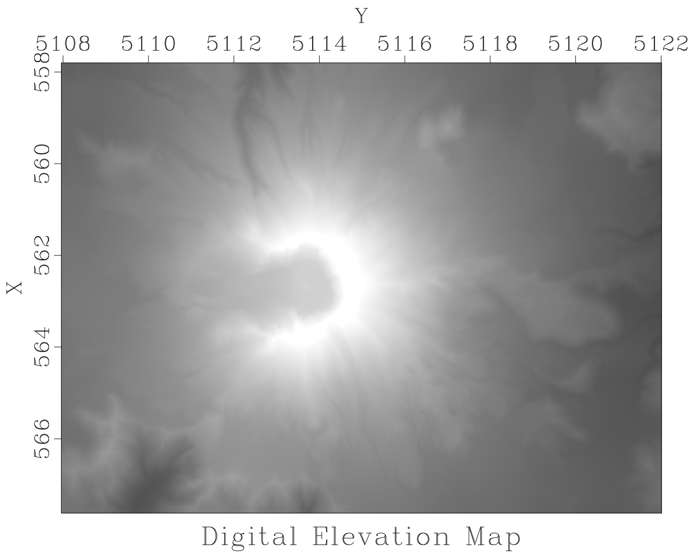

Figure 3: Digital elevation map of Mount St. Helens area. helix/ data

Figure 4 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.

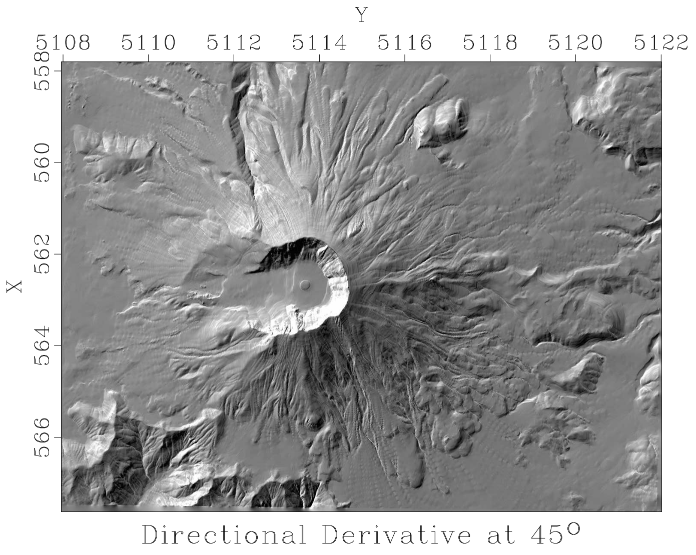

Figure 4: Directional derivative of elevation. helix/ der

Figure 5 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 is not confined to a preferential direction.

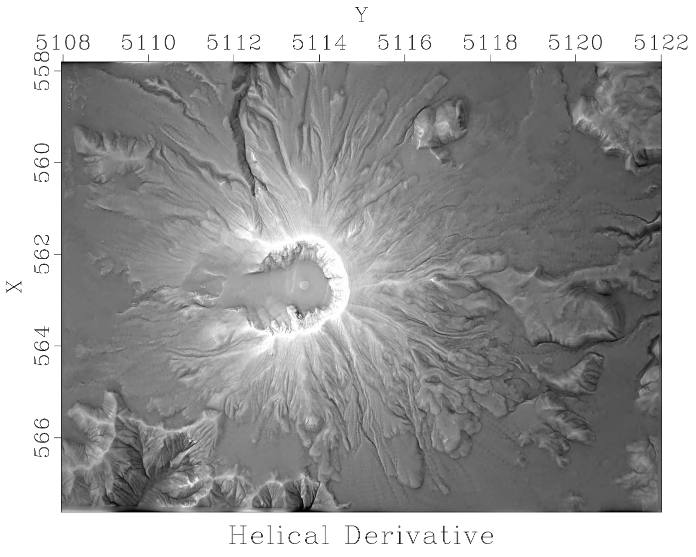

Figure 5: Helix derivative of elevation. helix/ helder

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

$$\hat{L}_2(Z_1, Z_2) = 20 \quad - \quad 4\,Z_1 - 4\,Z_1^{-1} - 4\,Z_2 - 4\,Z_2^{-1} \\ -Z_1\,Z_2 - Z_1\,Z_2^{-1} - Z_2\,Z_1^{-1} - Z_1^{-1}\,Z_2^{-1}\,. \qquad (6)$$

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

helix/SConstruct

```python
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
     n2=1400 o2=5107.965 d2=0.010035740 label2=Y |
     dd form=native |
     clip2 lower=0 | lapfill grad=y niter=200
     ''')

Result('data','grey title="Digital Elevation Map" allpos=y')
```

```

# Vertical and horizontal derivatives
Flow('der1','data','igrad')
Flow('der2','data','transp | igrad | transp')

ders = []
for alpha in range(0,360,15):
    der = 'der%d' % alpha

    # Directional derivative
    Flow(der,'der1 der2',
         '''
         add ${SOURCES[1]} scale=%g,%g
         ''' % (math.cos(alpha*math.pi/180),
                math.sin(alpha*math.pi/180)))
    ders.append(der)

Flow('ders',ders,
     '''
     cat axis=3 ${SOURCES[1:%d]} |
     put o3=0 d3=15
     ''' % len(ders))
Plot('ders','grey gainpanel=all wanttitle=n',view=1)

# !!! MODIFY BELOW !!!
alpha=45

Result('der','der%d' % alpha,
       '''
       grey title="Directional Derivative at %g\^o\_"
       ''' % alpha)

# Laplacian filter on a helix (!!! MODIFY !!!)

Flow('slag0.asc',None,
     '''echo 1 1000 n1=2 n=1000,1000
     data_format=ascii_int in=$TARGET
     ''')
Flow('slag','slag0.asc','dd form=native')

Flow('ss0.asc','slag',
     '''echo -1 -1 a0=2 n1=2
     lag=$SOURCE in=$TARGET data_format=ascii_float''')
Flow('ss','ss0.asc','dd form=native')

```

```
# Wilson-Burg factorization

na=50 # filter length

lags = list(range(1,na+1)) + list(range(1002-na,1002))

Flow('alag0.asc',None,
     '''echo %s n=1000,1000 n1=%d in=$TARGET
     data_format=ascii_int
     ''' % (' '.join([str(x) for x in 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 mean=y title="Helical Derivative" ')

def plotfilt(title):
    return '''
    window n1=11 n2=11 f1=50 f2=50 |
    grey wantaxis=n title="%s" pclip=100
    crowd=0.85 screenratio=1
    ''' % title

# Laplacian impulse response
Flow('spike',None,'spike n1=111 n2=111 k1=56 k2=56')
Flow('imp0','spike ss','helicon filt=${SOURCES[1]} adj=0')
Flow('imp1','spike ss','helicon filt=${SOURCES[1]} adj=1')
Flow('imp','imp0 imp1','add ${SOURCES[1]}')
Plot('imp',plotfilt('(a) Laplacian'))

# Test factorization
Flow('fac0','imp hflt',
     'helicon filt=${SOURCES[1]} adj=0 div=1')
Flow('fac1','imp hflt',
     'helicon filt=${SOURCES[1]} adj=1 div=1')
Plot('fac0',plotfilt('(b) Laplacian/Factor'))
Plot('fac1',plotfilt('(c) Laplacian/Factor\''))
Flow('fac','fac0 hflt',
     'helicon filt=${SOURCES[1]} adj=1 div=1')
Plot('fac',plotfilt('(d) Laplacian/Factor/Factor\''))

```

```
Result('laplace','imp fac0 fac1 fac','TwoRows',
       vppen='gridsize=5,5 xsize=11 ysize=11')

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`.)