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

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

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

   Suppose that the data vector $\mathbf{a}$ 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. You are given a table of numbers $x_1, x_2, \ldots, x_N$ and $y_1, y_2, \ldots, y_N$ and want to fit a parabolic model $y(x) = a + b\,x + c\,x^2$ to them.

   (a) Using the method of least squares, find a set of liner equations for coefficients a, b, and c for the case $N > 3$.

   (b) Using the method of least squares, find a, b, and c for the case of $N = 2$.

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

$$\mathbf{F} = \left[\begin{array}{cccccc} f_1 & f_0 & 0 & 0 & 0 & 0 \\ f_2 & f_1 & f_0 & 0 & 0 & 0 \\ f_3 & f_2 & f_1 & f_0 & 0 & 0 \\ 0 & f_3 & f_2 & f_1 & f_0 & 0 \\ 0 & 0 & f_3 & f_2 & f_1 & f_0 \\ 0 & 0 & 0 & f_3 & f_2 & f_1 \end{array}\right] . \tag{2}$$

   The operator is implemented in the C code below.

conv.c

```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++) {
        for (y = 0; y < ny; y++) {
            x = y-f+1;

            /* !!! CHANGE BELOW !!! */
```

```
            if (x < 0   ) continue;
            if (x >= nx) break;

            if( adj) {
                /* !!! INSERT CODE !!! */
            } else {
                yy[y] += 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.

4. The following C code (included in `filter.c` file in directory `geo391/hw3`) implements a recursive filtering operator.

filter.c

```
void filter_lop (bool adj, bool add,
                 int nx, int ny, float* xx, float* yy)
/*< linear operator >*/
{
    int i;
    float t;

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

    if (adj) {
        /* !!! INSERT CODE !!! */
    } else {
        t = a*xx[0];
        yy[0] += t;
        for (i = 1; i < nx; i++) {
            t = a*xx[i] + b*xx[i-1] + c*t;
            yy[i] += t;
        }
    }
}
```

(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 a slow 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.

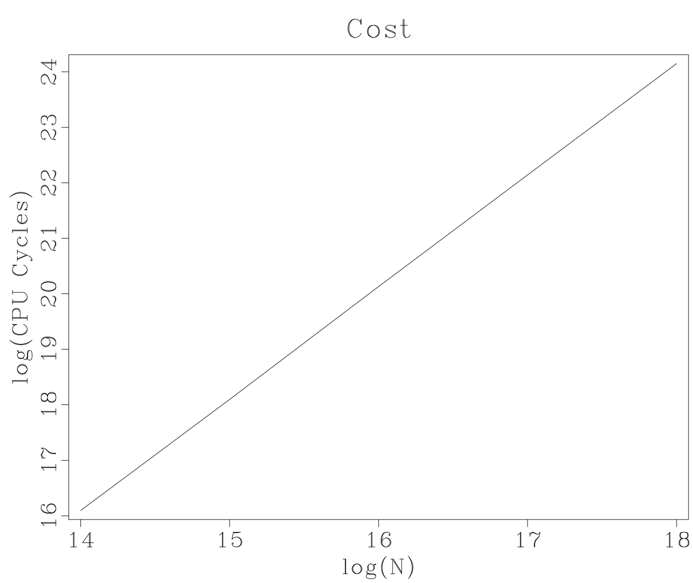


Figure 1: Experimental cost of slow sorting. The logarithm of the cost is shown against the logarithm of the data size.

If we approximate the cost as $P(N) = C\,N^{\epsilon}$, what is the experimental 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 further.

sorting/sorting.c

```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;
        }
```

```c
        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 */
    r = n; /* right side */
    pivot = list[0];

    /* separate into two lists:
       the left list for values <= pivot
       and the right list for > pivot */
    while (l < r) {
        ll = list[l];
        if (ll <= pivot) {
            l++;
        } else {
            r--;
            list[l] = list[r];
            list[r] = ll;
        }
    }
    list[0] = list[l-1];
    list[l-1] = pivot;

    quick_sort(l-1,list);

    /* !!! UNCOMMENT AND EDIT THE NEXT LINE !!! */
    /* quick_sort(?,?); */
}

int main(int argc, char* argv[])
{
    char* type;
    clock_t start, end;
    float cycles;
    sf_file in, cost;
```

```
    /* initialize */
    sf_init(argc,argv);

    /* input file */
    in = sf_input("in");
    if (SF_FLOAT != sf_gettype(in))
        sf_error("Need float input");
    na = sf_filesize(in); /* data size */

    /* cost file */
    cost = sf_output("out");
    sf_putint(cost,"n1",1);

    /* movie file */
    if (NULL != sf_getstring("movie")) {
        movie = sf_output("movie");
        sf_putint(movie,"n2",1);
        sf_putint(movie,"n3",na+1);
    } else {
        movie = NULL;
    }

    if (NULL == (type = sf_getstring("type")))
        type = "quick"; /* sort type */

    /* 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()
```

RUNNING MEDIAN AND RUNNING MEAN FILTERS

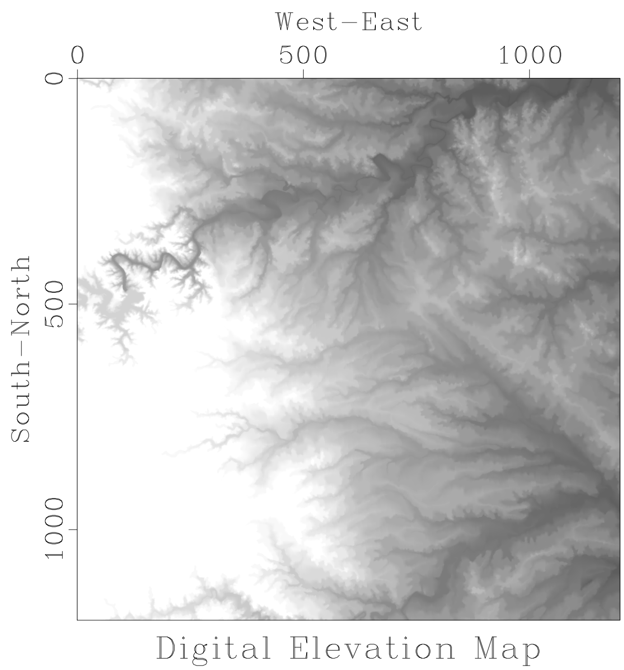


Figure 2: Digital elevation map of the west Austin area.

We return the digital elevation map of the West Austin Area, shown in Figure 2.

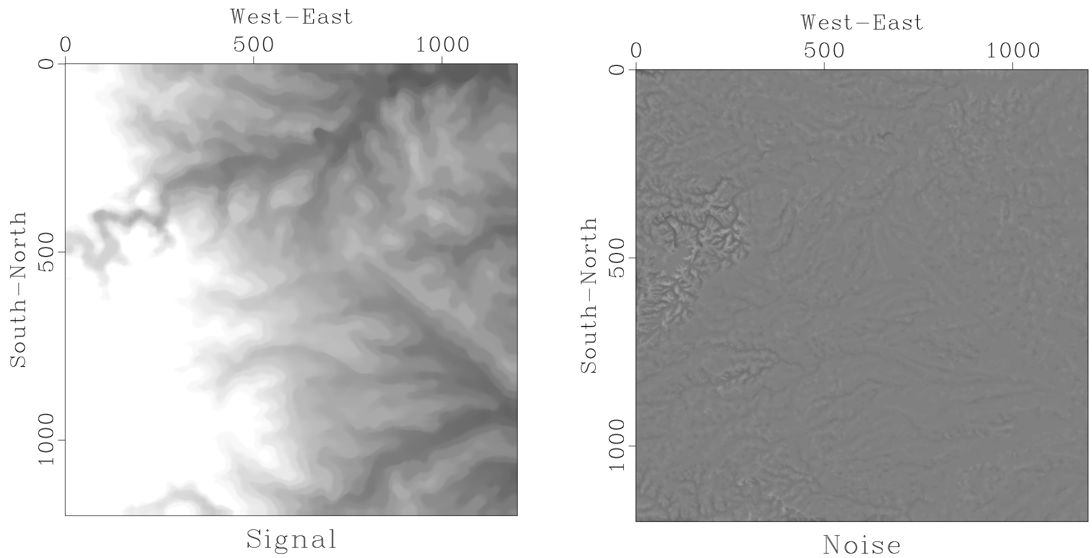
In this exercise, we will separate the data into “signal” and “noise” by applying running mean and median filters. The result of applying a running median filter is shown in Figure 3. Running median effectively smooths the data by removing local outliers.

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

running/running.c

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

#include <rsf.h>
```



(a) (b)

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

```
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[])
{
```

```c
    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 */
                    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'))

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

w = 30

# !!! CHANGE BELOW !!!
Flow('ave','dem %s' % run[0],
     './${SOURCES[1]} w1=%d w2=%d how=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')
     if not gtime:
          print "For computing CPU time, please 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,'dem %s' % run[0],
             '''
             ( (%s -f "%%S %%U"
             ./${SOURCES[1]} < ${SOURCES[0]}
              w1=%d w2=%d what=%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 4.

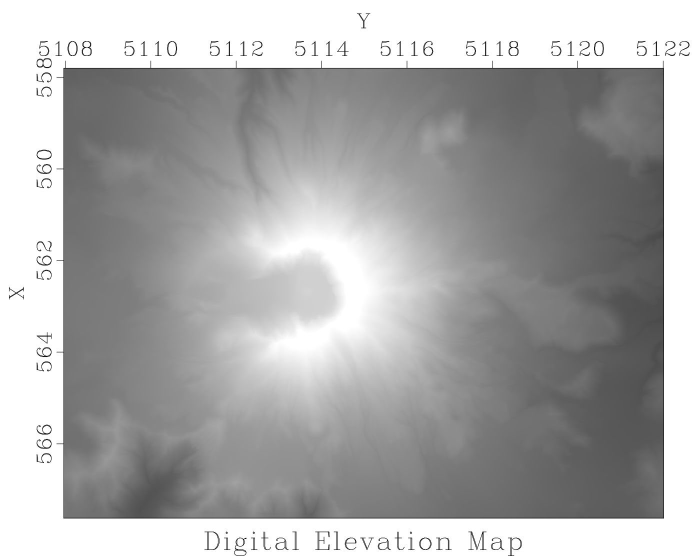


Figure 4: Digital elevation map of Mount St. Helens area.

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

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

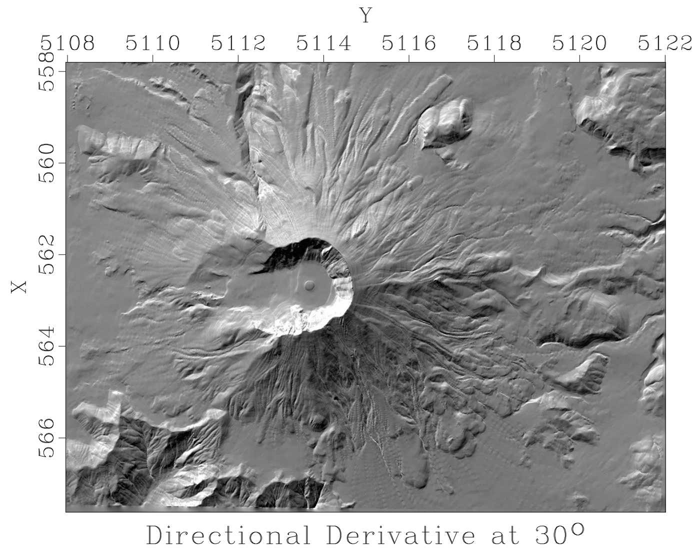


Figure 5: Directional derivative of elevation.

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.

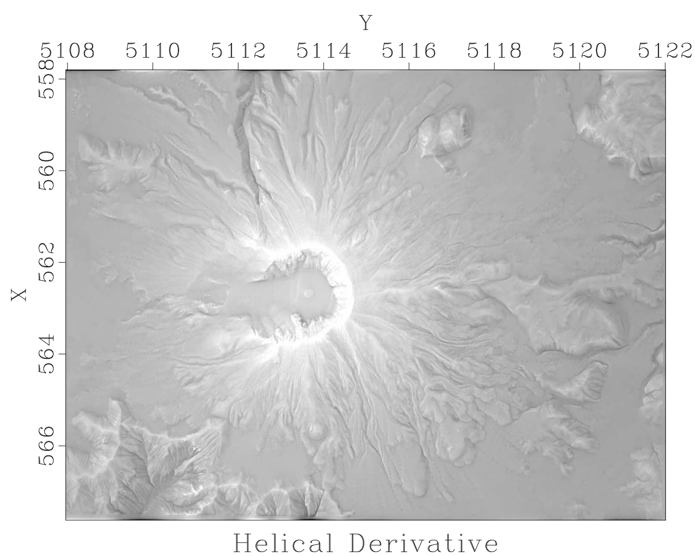


Figure 6: Helix derivative of elevation.

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 α 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 α automatically.

5. A more accurate version of the Laplacian filter is

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

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

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
     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(0,360,10):
    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=10
     ''' % len(ders))

# Semblance

Flow('ders2','ders','mul $SOURCE')
Flow('stack','ders2','stack axis=2 norm=n | stack axis=1 norm=n')
Flow('stack2','ders2',
     'mul $SOURCE | stack axis=2 norm=n | stack axis=1 norm=n')
Flow('sembl','stack stack2',
     'mul ${SOURCES[0]} | div ${SOURCES[1]} | scale axis=1')
Result('sembl',
       'graph title=Semblance label1=Degree unit1="\^o\_" ')

Plot('ders','grey gainpanel=all wanttitle=n',view=1)

# !!! MODIFY BELOW !!!
alpha=30

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

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" ')

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`) on paper or by e-mail.