Homework 3

Joseph Fourier

ABSTRACT

This homework has four parts.

1. Theoretical questions related to digital filters and forward interpolation.
2. Data interpolation after coordinate transformation.
3. Data compression using 2-D Fourier transform.
4. Missing data interpolation using compressive properties of the Fourier transform.

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/hw3
```

THEORETICAL PART

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 Taylor series expansion of the inverse sine function around zero is

$$\arcsin x = x + \frac{1}{2}\frac{x^3}{3} + \frac{1\cdot 3}{2\cdot 4}\frac{x^5}{5} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\frac{x^7}{7} + \cdots \tag{1}$$

   (a) Show how one can use expansion (1) to design a digital filter that approximates the derivative operator.
   Hint: Use the identity $1/Z - Z = 2\,i\,\sin(\omega\,\Delta t)$.

   (b) In particular, find a seven-point derivative filter of the form

$$D(Z) = d_{-3}/Z^3 + d_{-2}/Z^2 + d_{-1}/Z + d_0 + d_1\,Z + d_2\,Z^2 + d_3\,Z^3\ . \tag{2}$$

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

$$\beta_2(x) = \int_{-\infty}^{\infty} \beta_1(t)\,\beta_0(x-t)\,dt\ , \tag{3}$$

   where

$$\beta_0(x) = \begin{cases} 1 & \text{for} \quad |x| \le 1/2 \\ 0 & \text{for} \quad |x| > 1/2 \end{cases} \tag{4}$$

   and

$$\beta_1(x) = \int_{-\infty}^{\infty} \beta_0(t)\,\beta_0(x-t)\,dt\ = \begin{cases} 1-|x| & \text{for} \quad |x| \le 1 \\ 0 & \text{for} \quad |x| > 1 \end{cases} \tag{5}$$

   (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

$$d(x) = \sum_k c_k\,\beta_2(x-k) \tag{6}$$

   leads to an interpolation filter of the form

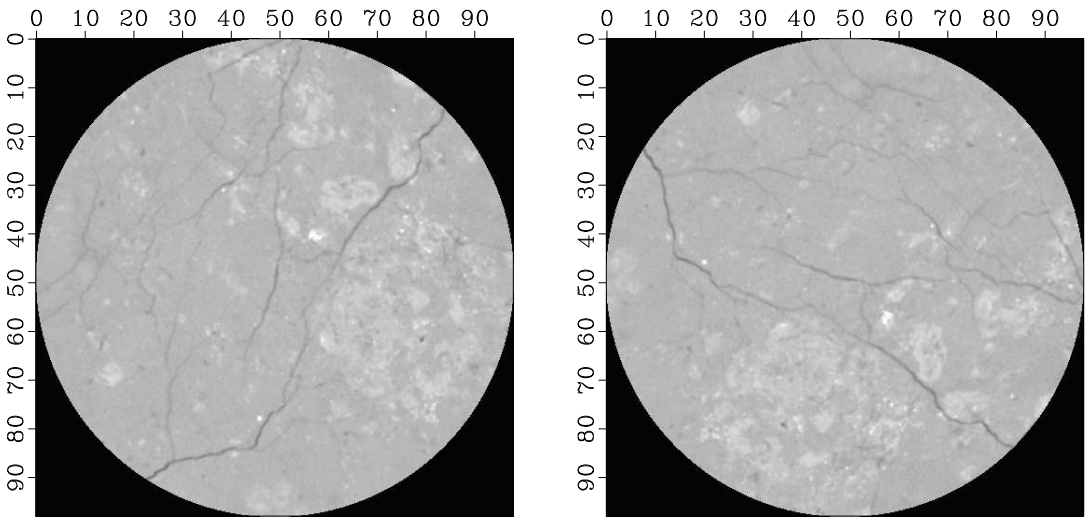
$$Z^\sigma \approx B_2(Z) = \frac{a_0(\sigma)\,Z^{-1} + a_1(\sigma) + a_2(\sigma)\,Z}{b_0\,Z^{-1} + b_1 + b_2\,Z}\ . \tag{7}$$

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

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 1a[1]. Notice microfracture channels.

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



(a) (b)

Figure 1: Slice of a CT-scan of a carbonate rock sample. (a) Original. (b) After clockwise rotation by 90°. rotate/ circle,rotate

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 1b 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.

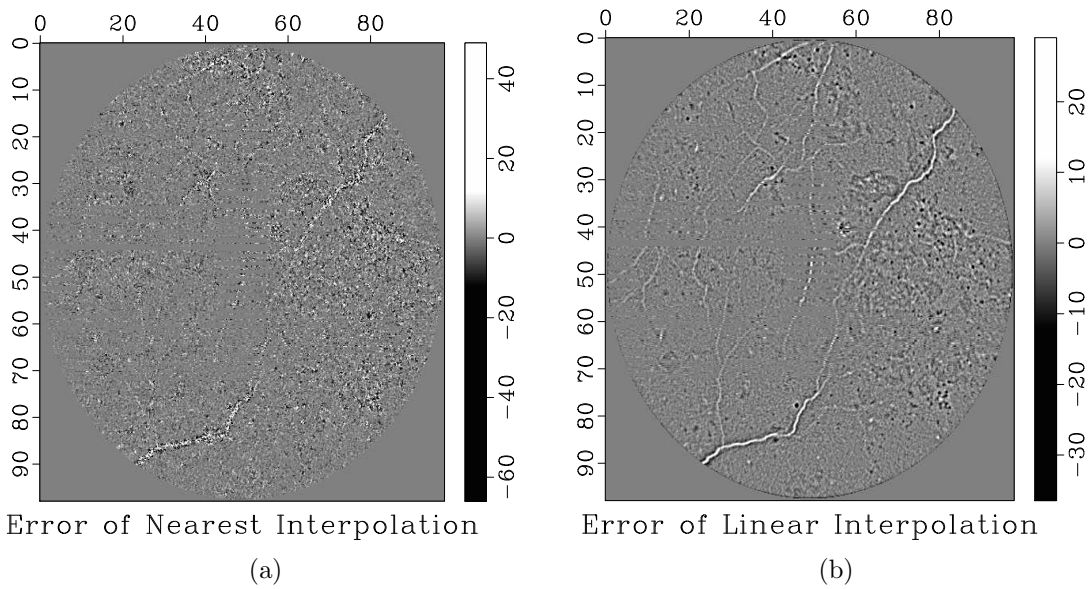
The task of coordinate rotation is accomplished by the C program `rotate.c` (alternatively, Python script `rotate.py`.) Two different methods are implemented: nearest-neighbor interpolation and bilinear interpolation.

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

$$\begin{aligned} Z^\sigma \approx C(Z) \quad = \quad & -\frac{\sigma\,(1-\sigma)^2}{2}\,Z^{-1} + \frac{(1-\sigma)\,(2+2\,\sigma-3\sigma^2)}{2} + \\ & \frac{\sigma\,(1+4\,\sigma-3\,\sigma^2)}{2}\,Z - \frac{(1-\sigma)\,\sigma^2}{2}\,Z^2\,. \end{aligned} \tag{8}$$

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



(a) (b)

Figure 2: Error of different interpolation methods computed after full circle rotation in increments of 20 degrees. (a) Nearest-neighbor interpolation. (b) Bi-linear interpolation. rotate/ nearest,linear

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");

    /* get parameters from command line */
    if (!sf_getfloat("angle",&angle)) angle=90.;
```

```
    /* rotation angle */

    if (NULL == (interp = sf_getstring("interp")))
        interp="nearest";
    /* [n,l,c] interpolation type */

    /* convert degrees to radians */
    angle *= SF_PI/180.;
    cosa = cosf(angle);
    sina = sinf(angle);

    orig = sf_floatalloc2(n1,n2);
    rotd = sf_floatalloc2(n1,n2);

    /* read data */
    sf_floatread(orig[0],n1*n2,inp);

    /* central point */
    c1 = (n1-1)*0.5;
    c2 = (n2-1)*0.5;

    for (i2=0; i2 < n2; i2++) {
        for (i1=0; i1 < n1; i1++) {

            /* rotated coordinates */
            x1 = c1+(i1-c1)*cosa-(i2-c2)*sina;
            x2 = c2+(i1-c1)*sina+(i2-c2)*cosa;

            /* nearest neighbor */
            k1 = floorf(x1); x1 -= k1;
            k2 = floorf(x2); x2 -= k2;

            switch(interp[0]) {
                case 'n': /* nearest neighbor */
                    if (x1 > 0.5) k1++;
                    if (x2 > 0.5) k2++;
                    if (k1 >=0 && k1 < n1 &&
                        k2 >=0 && k2 < n2) {
                        rotd[i2][i1] = orig[k2][k1];
                    } else {
                        rotd[i2][i1] = 0.;
                    }
                    break;
                case 'l': /* bilinear */
                    if (k1 >=0 && k1 < n1-1 &&
```

```
                            k2 >=0 && k2 < n2-1) {
                            rotd[i2][i1] =
                                (1.-x1)*(1.-x2)*orig[k2][k1]   +
                                x1     *(1.-x2)*orig[k2][k1+1] +
                                (1.-x1)*x2     *orig[k2+1][k1] +
                                x1     *x2     *orig[k2+1][k1+1];
                        } else {
                            rotd[i2][i1] = 0.;
                        }
                        break;
                    case 'c': /* cubic convolution */
                        /* !!! ADD CODE !!! */
                        break;
                    default:
                        sf_error("Unknown method %s",interp);
                        break;
                }
            }
        }

    /* write result */
    sf_floatwrite(rotd[0],n1*n2,out);

    exit(0);
}
```

rotate/rotate.py

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

import sys
import math
import numpy as np
import m8r

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

# get dimensions from input
n1 = inp.int('n1')
n2 = inp.int('n2')

# get parameters from command line
```

```
angle = par.float('angle',90.)
# rotation angle

interp = par.string('interp','nearest')
# [n,l,c] interpolation type

# convert degrees to radians
angle = angle*math.pi/180.
cosa = math.cos(angle)
sina = math.sin(angle)

orig = np.zeros([n1,n2],'f')
rotd = np.zeros([n1,n2],'f')

# read data
inp.read(orig)

# central point
c1 = (n1-1)*0.5
c2 = (n2-1)*0.5

for i2 in range(n2):
    for i1 in range(n1):
        # rotated coordinates
        x1 = c1+(i1-c1)*cosa-(i2-c2)*sina
        x2 = c2+(i1-c1)*sina+(i2-c2)*cosa

            # nearest neighbor
        k1 = int(math.floor(x1))
        k2 = int(math.floor(x2))
        x1 -= k1
        x2 -= k2

        if interp[0] == 'n':
            # nearest neighbor
            if x1 > 0.5:
                k1 += 1
            if x2 > 0.5:
                k2 += 1
            if k1 >=0 and k1 < n1 \
              and k2 >=0 and k2 < n2:
                rotd[i2,i1] = orig[k2,k1]
            else:
                rotd[i2,i1] = 0.
        elif interp[0] == 'l':
```

```
            # bilinear
            if k1 >=0 and k1 < n1-1 \
              and k2 >=0 and k2 < n2-1:
                rotd[i2,i1] = \
                            (1.-x1)*(1.-x2)*orig[k2,k1]   + \
                            x1     *(1.-x2)*orig[k2,k1+1] + \
                            (1.-x1)*x2     *orig[k2+1,k1] + \
                            x1     *x2     *orig[k2+1,k1+1]
            else:
                rotd[i2,i1] = 0.
        elif interp[0] == 'c':
            # cubic convolution
                    # !!! ADD CODE !!!
            break
        else:
            sys.stderr.write('Unknown method "%s"' % interp)
            sys.exit(1)

 # write result */
out.write(rotd)
sys.exit(0)
```

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
# exe = Program('rotate.c')

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

rotate = str(exe[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 `hw3/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.

FOURIER COMPRESSION

In this exercise, we will use a depth slice selected from a 3-D seismic volume and shown in Figure 3 (Hall, 2007). Notice a channel structure.

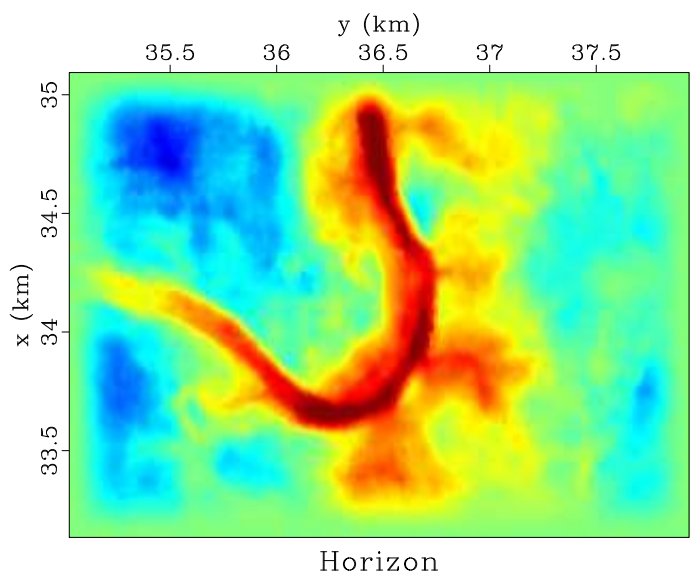


Figure 3: Seismic depth slice with a channel structure. compress/ data

The goal of your assignment is to find a compressed representation of the data in the Fourier transform domain. Figure 4 shows the Fourier transform of the data from Figure 3. 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:

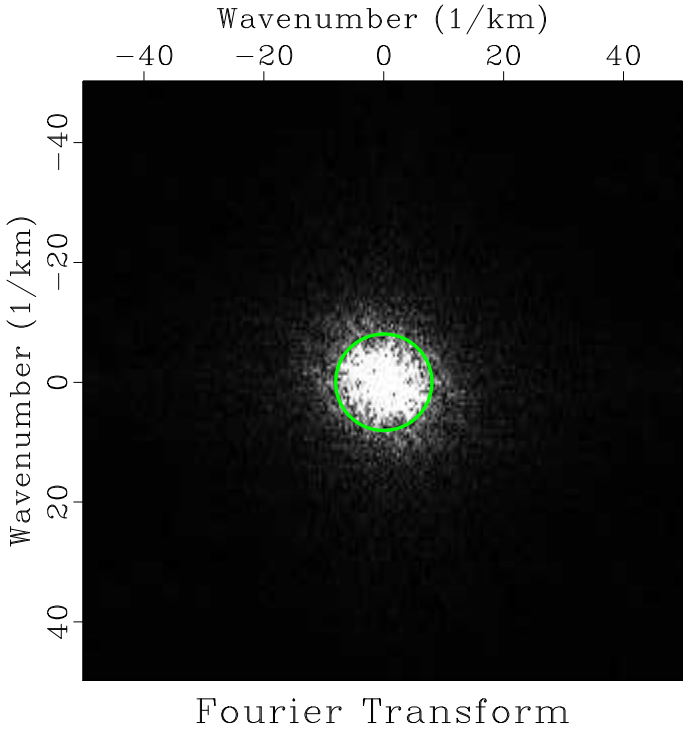


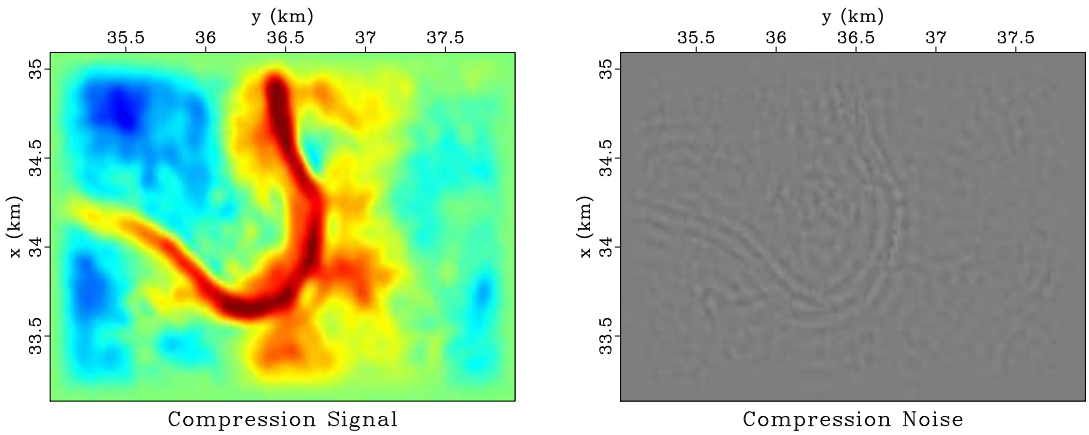
Figure 4: Absolute value of the Fourier transform of the seismic slice from Figure 3. The circle inside shows a window selected for compression. compress/ fft

- 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 5.
- 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 6. Figure 7 shows the selected threshold plotted against the histogram of Fourier coefficients.

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

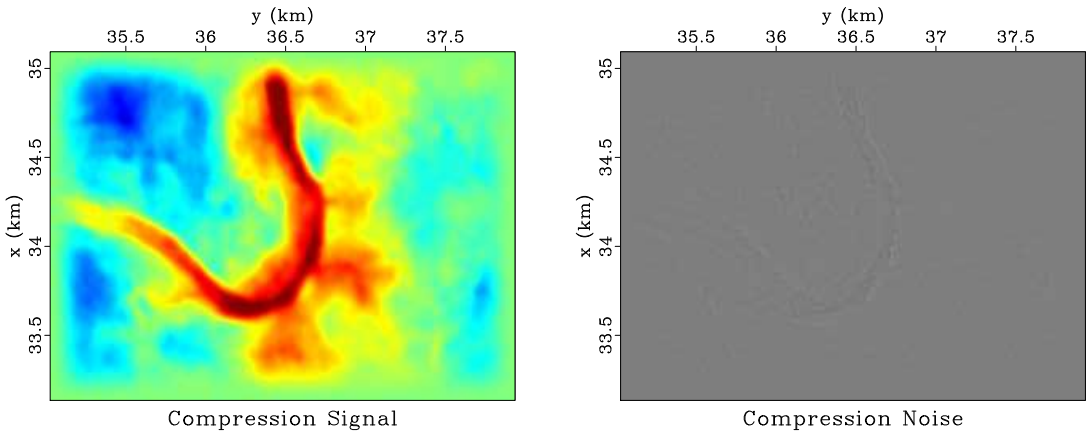
   ```
   scons view
   ```

   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 5b. How do you measure the noise level? Find a level that you find negligibly small.
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 6b should match that in Figure 5b.



(a) (b)

Figure 5: Data separated into signal (a) and noise (b) by applying Fourier compression with windowing. compress/ sig,cut



(a) (b)

Figure 6: Data separated into signal (a) and noise (b) by applying Fourier compression with thresholding. compress/ thr,noi

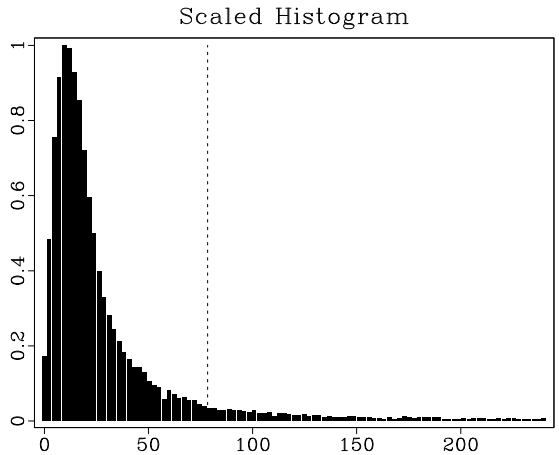


Figure 7: Normalized histogram of Fourier coefficients (by absolute value). The vertical line shows the selected threshold. compress/ hist

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 |
     bargraph 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()
```

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 8).

Figures 9a and 9b 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

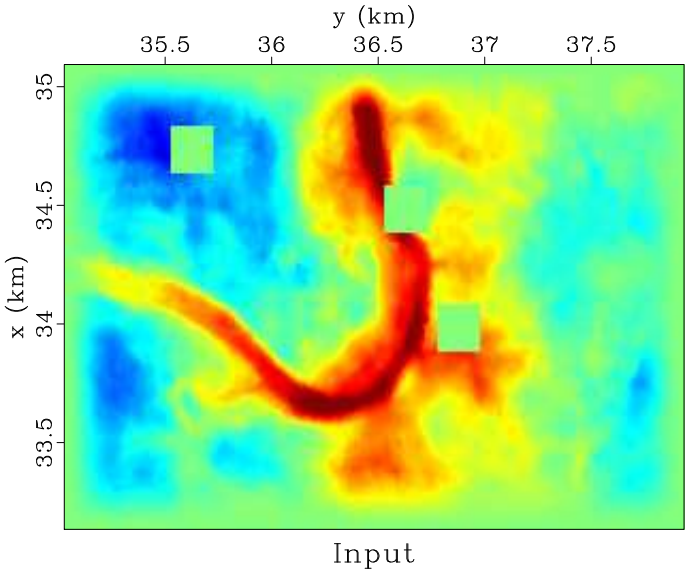
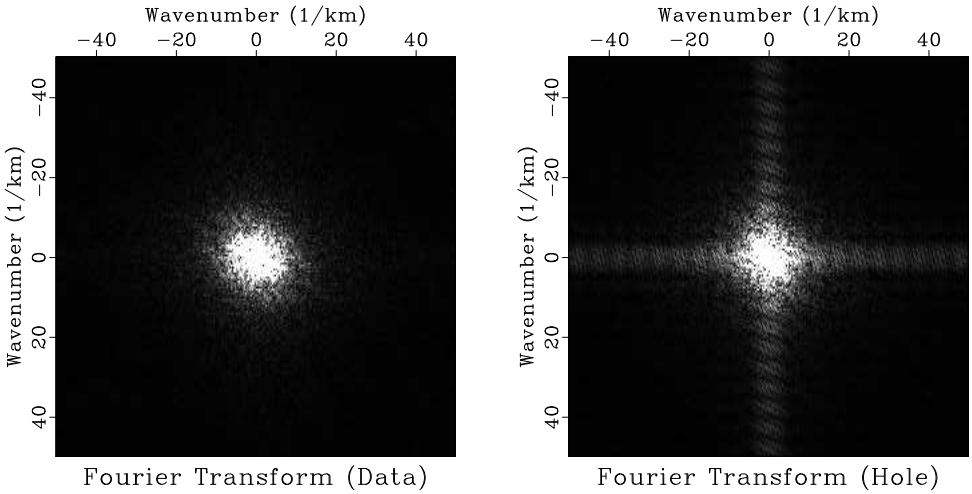


Figure 8: Seismic depth slice after removing selected parts of the data. pocs/ hole



(a) (b)

Figure 9: Fourier transform of the original data (a) and data with holes with holes (b). The absolute value is displayed pocs/ fft-data,fft-hole

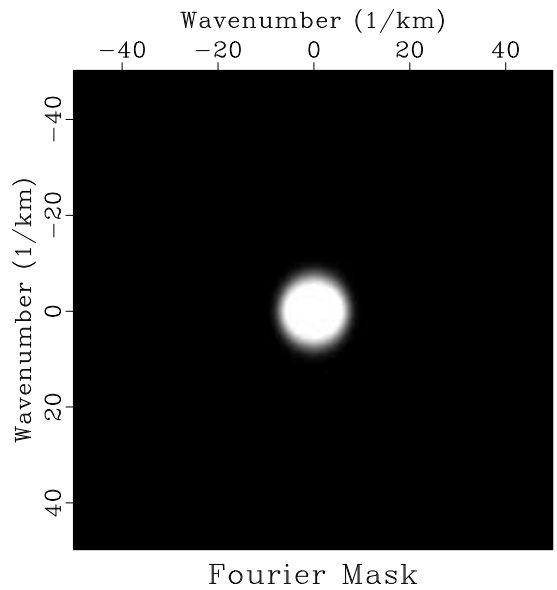


Figure 10: Fourier-domain mask for selecting a convex set. pocs/ fft-mask

support. Figure 10 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 $0 \leq \lambda \leq 1$) 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 (Youla and Webb, 1982).

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 11.

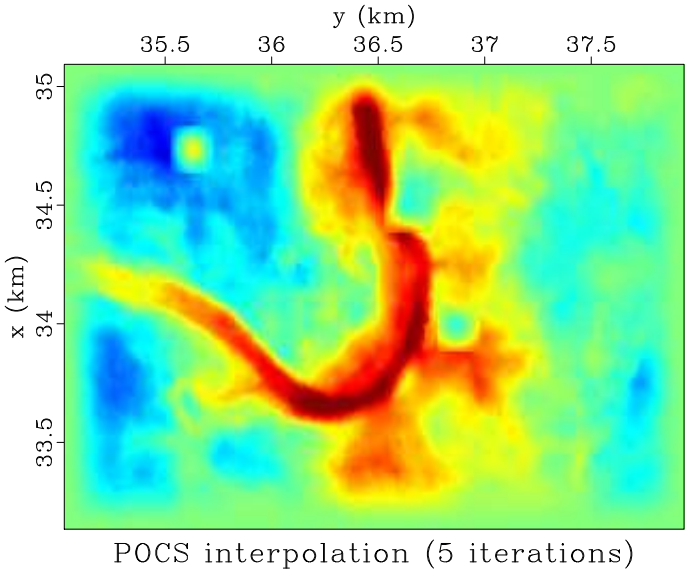


Figure 11: Missing data interpolated by iterative projection onto convex sets. pocs/ pocs

```
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 |
     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'))

# Cut three square holes (!!! CHANGE ME !!!)
cut = '''
cut n1=20 n2=20 f1=125 f2=150 |
```

```
cut n1=20 n2=20 f1=150 f2=50  |
cut n1=20 n2=20 f1=75  f2=175
'''

Flow('hole','data',cut)
Flow('mask','data',
     'math output=1 | %s | math output=1-input' % cut)
Plot('hole',plot('Input'))
Result('hole','Overlay')

# 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)
# Put frames in a movie
Plot('pocs',plots,'Movie',view=1)

# Last frame
Result('pocs',data,
       plot('POCS interpolation (%d iterations)' % niter))

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.

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`) by e-mail.

REFERENCES

Hall, M., 2007, Smooth operator: Smoothing seismic interpretations and attributes: The Leading Edge, **26**, 16–20.

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

Youla, D. C., and H. Webb, 1982, Image restoration by the method of convex projections: Part 1. Theory: IEEE transactions on medical imaging, **1**, 81–94.