Summary

In this thesis, I discussed several methods and applications of data matching in seismic data analysis. Chapter 2 focuses on introducing the three data matching operators that are used in this thesis—shifting, scaling, and filtering. Chapter 3 introduces different methods of frequency balancing using non-stationary smoothing. The first method to find the non-stationary smoothing *radius*, or number of samples each data point is averaged over with a triangle weight, took a theoretical approach based off of the assumption that the data we observe can be modeled by a summation of Ricker wavelets. This method worked well in certain situations, but was not robust enough to work for any given data set. In the second method, I introduced an iterative algorithm to find the smoothing radius, and this method converges quickly and works well in several presented situations. Finally, a modification to this algorithm was shown and allows smoothing for more complex data sets.

This chapter also discusses two applications of these algorithms—the frequency balancing algorithm was demonstrated on an application of matching high-resolution and legacy seismic images, and the modified algorithm was demonstrated on an application of multicomponent seismic image registration.

Chapter 4 goes into more detail of the application of matching and merging high-resolution and legacy seismic images. This example takes two seismic volumes, acquired over the same area but using different technologies, and first matches them before merging them together to produce an optimized third image. First, the method is demonstrated on a 2D line from the Gulf of Mexico. Then, the method is applied to a 3D seismic volume from a different part of the Gulf of Mexico.

Chapter 5 discusses another application of improving migration resolution by approximating the least-squares Hessian using non-stationary data matching operations. An approximation to the least-squares Hessian can be calculated by solving a data matching problem between two conventionally migrated images, and the Hessian can be represented by the combination of amplitude and frequency balancing operations. An example is applied to a 2D synthetic Sigsbee data set.

Future work

In the future, the work presented in chapter 5 should be extended to involve real data and 3D examples. It also could benefit by comparing the results of the proposed approach taken in the chapter to other previous approaches presented to approximate the least-squares Hessian (**????????**), to see how it compares in different situations.

Another extension of this data matching procedure may be to incorporate the phase of the signal to be matched. Negligible improvements were made when trying to incorporate phase corrections into the high-resolution and legacy data matching problem of chapter 4, but other data matching problems could benefit from these corrections.

Several applications of data matching were discussed in this thesis. However, many applications remain unaddressed from a data matching standpoint. Problems such as seismic and well-log tying, deconvolution, automatic gain control (AGC), and surface-related multiple elimination (SRME) can also be recast as data matching problems. Looking at these problems in a new light may bring advancements to computational geophysics.

CODE

The examples in this thesis were implemented with the Madagascar open-source software environment for reproducible computational experiments (**?**). The package is available at `http://www.ahay.org/`.

The reproducible document for the results in this thesis, including code, is available at `http://www.sygreer.com/research/honorsThesis`. However, some of the data used in this thesis are proprietary, so those results may not directly be reproducible.

The scripts and programs used to produce the examples in this thesis are below.

Table 1: Figures and the scripts to generate them

Figures	**Directory**	**Listings**
??	chapter-locfreq/merge/	1, 2, 3
??, ??	chapter-background/dmExample/	??
??, ??, ??, ??	chapter-merge/apache/	??
??, ??, ??, ??	chapter-locfreq/merge/	1, 2, 3
??, ??	chapter-locfreq/vecta/	??, ??, ??, ??
??	chapter-locfreq/convergence/	??
??, ??, ??, ??	chapter-merge/apache/	??
??, ??	chapter-merge/pcable/	??
??	chapter-merge/pcable2/	??, ??, ??
??, ??, ??, ??, ??	chapter-merge/mighes/	??, ??, ??
??,	chapter-merge/triop/	??, ?? ??

For brevity in this thesis, code is only included for one example of the main frequency balancing algorithm presented in chapter 3.

Listing 1: chapter-locfreq/merge/SConstruct

```
from rsf.proj import *
from radius import radius

# Initial figures
Result('legacy','grey title="Legacy"')
Result('hires-agc','hires','agc rect1=2000 rect2=5 | grey title="High-resolution"')

Flow('legacy-spec','legacy','spectra all=y')
Result('nspectra-orig','high-spec legacy-spec',
    '''cat axis=2 ${SOURCES[1]} |
```

```
        scale axis=1 | window max1=180 |
        graph title="Normalized Spectra" label2="Amplitude" unit2=""'''')

# Balance local frequency
flol=40
corrections = 5
Flow('legacyfilt','legacy','bandpass flo=%d '%(flol))
radius('hires','legacyfilt', corrections, [0.13,0.2,0.3,0.5,0.5],
        bias=0, clip=90, rect1=80, rect2=16, maxval=90 )

End()
```

Listing 2: chapter-locfreq/merge/radius.py

```
from rsf.proj import *

def radius(high, low,                # initial high-resolution and legacy images
           niter,                    # number of corrections
           c,                        # 'step length' for radius corrections. Can
                                     # be type int or float for constant c
                                     # or type array for changing c.
           bias=-15, clip=30,        # bias and clip for display
           rect1=40, rect2=80,       # radius for local frequency calculation
           maxrad=1000,              # maximum allowed radius
           theor=True,               # use theoretical smoothing radius
           scale=9,                  # scale for theoretical smoothing radius
           initial=10,               # initial value for contant smoothing radius
           minval=0,
           maxval=25,
           titlehigh="Hires",
           titlelow="Legacy"):

    if type(c) is float or type(c) is int:
        c = [c]*niter

    def seisplot(name):
        return 'grey title="%s"'%name

    locfreq = '''iphase order=10 rect1=%d rect2=%d hertz=y complex=y |
                 put label="Frequency" unit=Hz'''%(rect1,rect2)

    def locfreqplot(name):
        return 'grey mean=y color=j scalebar=y title="%s" '%name

    freqdif = 'add scale=-1,1 ${SOURCES[1]} | put label=Frequency'

    def freqdifplot(num):
        return '''grey allpos=y color=j scalebar=y mean=y
                  title="Difference in Local Frequencies %s"
                  clip=%d bias=%d minval=%d
                  maxval=%d''' %(num,clip,bias,minval,maxval)

    specplot = '''cat axis=2 ${SOURCES[1]} |
                  scale axis=1 | window max1=180 |
                  graph title="Normalized Spectra" label2="Amplitude" unit2=""'''

    def rectplot(name):
        return '''grey color=j mean=y title="%s" scalebar=y barlabel=Radius
                  barunit=samples'''%name

    smooth = 'nsmooth1 rect=${SOURCES[1]}'

    Result(high, seisplot(titlehigh))
    Result(low, seisplot(titlelow))

    Flow('high-freq',high,locfreq)
    Result('high-freq',locfreqplot('%s Local Frequency'%titlehigh))
```

```
    Flow('low-freq',low,locfreq)
    Result('low-freq',locfreqplot('%s Local Frequency'%titlelow))

    locfreq2 = '''iphase order=10 rect1=1 rect2=1 hertz=y complex=y |
                 put label="Frequency" unit=Hz'''

    Flow('low-freq2',low,locfreq2)
    Result('low-freq2',locfreqplot('%s Instantaneous Frequency'%titlelow))

    Flow('freqdif','low-freq high-freq',freqdif)
    Result('freqdif',freqdifplot(''))

    # initial smoothing radius
    if (theor):
        from math import pi
        Flow('rect0','low-freq high-freq','''math f1=${SOURCES[1]}
        output="sqrt(%g*(1/(input*input)-1/(f1*f1)))/%g"'''%(scale,2*pi*0.001))
    else:
        Flow('rect0','low-freq','math output=%f'%initial)

    Result('rect0',rectplot("Smoothing Radius 0"))

    Flow('high-smooth0','%s rect0' % high,smooth)
    Result('high-smooth0', seisplot("%s Smooth 0"%titlehigh))

    Flow('high-spec',high,'spectra all=y')
    Flow('low-spec',low,'spectra all=y')
    Flow('high-smooth-spec0','high-smooth0','spectra all=y')
    Result('nspectra','high-spec low-spec',specplot)
    Result('high-smooth-spec0','high-smooth-spec0 low-spec',specplot)

    Flow('high-smooth-freq0','high-smooth0',locfreq)
    Result('high-smooth-freq0',
           locfreqplot("%s Local Frequency Smoothed %d" %(titlehigh,0)))

    Flow('freqdif-filt0','low-freq high-smooth-freq0',freqdif)
    Result('freqdif-filt0',freqdifplot('0'))

    prog=Program('radius.c')
    for i in range(1, niter+1):
        j = i-1
        Flow('rect%d'%i,'rect%d freqdif-filt%d %s'%(j,j,prog[0]),
             './${SOURCES[2]} freq=${SOURCES[1]} c=%f'%c[j])
        Result('rect%d'%i,rectplot("Smoothing Radius %d")%i)

        Flow('high-smooth%d'%i,'%s rect%d'%(high,i),smooth)
        Result('high-smooth%d'%i, seisplot('%s Smooth %d'%(titlehigh,i)))

        Flow('high-smooth-spec%d'%i,'high-smooth%d'%i,'spectra all=y')
        Result('high-smooth-spec%d'%i,'high-smooth-spec%d low-spec'%i,specplot)

        Flow('high-smooth-freq%d'%i,'high-smooth%d'%i,locfreq)
        Result('high-smooth-freq%d'%i,
               locfreqplot('%s Local Frequency Smoothed %d'%(titlehigh,i)))

        Flow('freqdif-filt%d'%i,'low-freq high-smooth-freq%d'%i,freqdif)
        Result('freqdif-filt%d'%i,freqdifplot(str(i)))
```

Listing 3: chapter-locfreq/merge/radius.c

```
/* smoothing radius (min = 1) */
#include <rsf.h>
#include <math.h>

int main (int argc, char* argv[])
{
```

```
    int n1, n1f, n2, n2f, i, n12, n12f;
    float *rect, *fr, maxrad, c, *rad;
    sf_file in, out, freq;

    sf_init (argc,argv);

    in = sf_input("in");
    freq = sf_input("freq");
    out = sf_output("out");

    if (!sf_histint(in,"n1",&n1)) sf_error("No n1= in input.");
    if (!sf_histint(freq,"n1",&n1f)) sf_error("No n1= in frequency difference.");

    n2 = sf_leftsize(in,1);
    n2f = sf_leftsize(freq,1);

    n12 = n1*n2;
    n12f = n1f*n2f;

    if (n1 != n1f) sf_error("Need matching n1");
    if (n2 != n2f) sf_error("Need matching n2");

    if (!sf_getfloat("c",&c)) c=1.;
    if (!sf_getfloat("maxrad",&maxrad)) maxrad=1000.;

    rect = sf_floatalloc(n12);
    sf_floatread(rect,n12,in);

    fr = sf_floatalloc(n12f);
    sf_floatread(fr,n12,freq);

    rad = sf_floatalloc(n12);

    for (i=0; i < n12; i++) {

        /* update radius */
      rad[i] = rect[i]+c*fr[i];

        /* set constraint conditions: [1, maxrad] */
        if (rad[i] > maxrad)
                rad[i] = maxrad;
        if (rad[i] < 1.0)
                rad[i] = 1.0;
    }

    sf_floatwrite(rad,n12,out);
    exit(0);
}
```