In this thesis, I discussed several methods and applications of data matching in seismic data analysis. Chapter **??** introduced on introducing the three data matching operators that are used in this thesis—shifting, scaling, and filtering. Chapter **??** introduced 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 on the assumption that the data we observe can be represented by a superposition of Ricker wavelets. This method worked well in certain situations, but was not robust enough to work for all of the given data sets. In the second method, I introduced an iterative algorithm to find the smoothing radius. In my experiments, this method converges quickly and works well in several presented situations. Finally, a modification to this algorithm was shown that allows adaptive smoothing for more complex data sets.

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

Chapter **??** went 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, I demonstrated the method on a 2D line from the Gulf of Mexico. Then, I applied the method to a 3D seismic volume from a different part of the Gulf of Mexico.

Chapter **??** discusses another application of improving migration resolution by approximating the least-squares Hessian using non-stationary data matching operations. I showed that 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. I applied the proposed approach to a 2D synthetic Sigsbee data set, a traditional benchmark for imaging algorithms.

# FUTURE WORK

In the future, the work presented in Chapter **??** should be extended to involve real data and 3D examples. It could also benefit from comparing the results of the proposed approach to 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 **??**. However, other data matching problems could benefit from these corrections.

The newly developed field of deep learning may also benefit from including these data matching techniques in the parameterization, as specifically tailoring the neural networks to work with geophysical data in this manner may produce better results.

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 using 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 be directly reproducible.

For brevity in this thesis, code is only included for one example of the main frequency balancing algorithm presented in Chapter **??**. The code for the rest of the examples in this thesis are available online at the URL above.

Table 1: List of figures in this thesis and the locations of scripts and programs 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/ | — |

Listing 1: chapter-locfreq/merge/SConstruct

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

# must have 'legacy.rsf' and 'hires.rsf' initial data sets in same directory
```

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

11 # frequency content
Flow('legacy-spec','legacy','spectra all=y')
13 Result('nspectra-orig','high-spec legacy-spec',
     '''cat axis=2 ${SOURCES[1]} | scale axis=1 | window max1=180 |
15       graph title="Normalized Spectra" label2="Amplitude" unit2=""''')

17 # Balance local frequency
flol=40
19 corrections = 5
Flow('legacyfilt','legacy','bandpass flo=%d '%(flol))
21 radius('hires','legacyfilt', corrections, [0.13,0.2,0.3,0.5,0.5],
       bias=0, clip=90, rect1=80, rect2=16, maxval=90 )
23
End()
```

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

```
from rsf.proj import *
2
def radius(high, low,                # initial high and low frequency images
4           niter,                   # number of corrections
            c,                       # 'step length' for radius corrections. Can
6                                    #       be type int or float for constant c
                                     #       or type array for changing c.
8           bias=-15, clip=30,       # bias and clip for display
            rect1=40, rect2=80,      # radius for local frequency calculation
10          maxrad=1000,             # maximum allowed radius
            theor=True,              # use theoretical smoothing radius
12          scale=9,                 # scale for theoretical smoothing radius
            initial=10,              # initial value for contant smoothing radius
14          minval=0, maxval=25,     # min and max local frequency for display
            titlehigh="Hires",
16          titlelow="Legacy"):

18      if type(c) is float or type(c) is int:
            c = [c]*niter
20
        # plot image
22      def seisplot(name):
            return 'grey title="%s"'%name
24
        # local frequency
26      locfreq = '''iphase order=10 rect1=%d rect2=%d hertz=y complex=y |
                  put label="Frequency" unit=Hz'''%(rect1,rect2)
28
        def locfreqplot(name):
30          return 'grey mean=y color=j scalebar=y title="%s" '%name

32      # difference in local frequencies
        freqdif = 'add scale=-1,1 ${SOURCES[1]} | put label=Frequency'
34
        def freqdifplot(num):
36          return '''grey allpos=y color=j scalebar=y mean=y
                    title="Difference in Local Frequencies %s"
38                  clip=%d bias=%d minval=%d
                    maxval=%d''' %(num,clip,bias,minval,maxval)
40
        # plot spectral content
42      specplot = '''cat axis=2 ${SOURCES[1]} |
                  scale axis=1 | window max1=180 |
44                graph title="Normalized Spectra" label2="Amplitude" unit2=""'''
```

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

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

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

    # plot images
    Result(high, seisplot(titlehigh))
    Result(low, seisplot(titlelow))

    # initial local frequency
    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))

    # initial difference in local frequency
    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"))

    # smoothing using intial smoothing radius guess
    Flow('high-smooth0','%s rect0' % high,smooth)
    Result('high-smooth0', seisplot("%s Smooth 0"%titlehigh))

    # frequency spectra
    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

        # update smoothing radius
        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)

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

        # smoothed spectra
```

```
        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)

        # smoothed local frequency
        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)))

        # frequency residual
        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);
```

```
54 }
```