Seismic Array Correlation and Clustering with `CORAL`: User's Manual

Joshua D Carmichael, Ken C. Creager, Wes Thelen

May 5, 2010

Contents

Chapter 1

Quick Reference

This purpose of this chapter is to provide a reference for helping the user find a tool for a given processing task. I assume you have read the remainder of the document, or are familiar enough with `CORAL` in order to utilize this reference. It is likely that I did not anticipate every possible application, **nor** did I include all the redundant information already contained in `CORAL`.man. My hope is this initiates your search in the correct direction.

1.1 Loading SAC Seismograms

- `pascSac2Coral.m`: Load station or entire record section, given time window of desired record. Requires minimal input information. Cannot multiple days at once. Output options to extract file names for all files found matching time window input.
- `coralReadSac.m`: Loads single seismogram given complete file name. `pascSac2Coral.m` uses this function internally.
- `getFiles.m`: Multi-purpose function. Serves same function as `ls` in `Linux`, but provides file names as output for reference.

1.2 Plotting `CORAL` Seismograms

- `coralPlot.m`: Useful function for plotting time series in field `data`. For $M \times N$ `CORAL` structures, only the largest dimension of seismograms are plotted.

- `coralPlotLabel.m`: Labels plots of seismogram according to channel and station code fields in `CORAL` structure.
- `coralHelicorder.m`: Produces helicorder plot of seismograms, similar to the PNSN helicorder. Useful for single, long seismograms.
- `plotlRaClusterStacks.m`: Takes cell array input from CLUSTERHISTORY*mat to produce plots of seismograms. See help menu for specific implementation.

1.3 Arithmetic `CORAL` and Custodial Operations

- `normData.m`: Normalizes matrices of data by columns. Called by several `CORAL` functions.
- `coralNormColumns.m`: Normalizes data in $M \times N$ `CORAL` structure arrays according to Frobenius norm for record sections. Normalization is computed differently if N is 1. Normalization of each seismogram individually done via `arrayfun(@coralNormData, S)`
- `coralRMS.m`: Computes the root-mean-square amplitude of seismograms in $M \times N$ `CORAL` structure arrays.
- `coralExtract.m`: Extracts sub-array of $M \times N$ `CORAL` structure through identification of matching fields.
- `coralRaDistance.m`: Computes the distance between all elements in the $M \times N$ `CORAL` structure array.
- `coralSetNames.m`: Sets names in `CORAL` structure by matching input file names with input structure. Used in case staCode fields are not populated properly in SAC.
- `coralSetRefs.m`: Uses input 'reference' structure and reference field to copy field values from $M_1 \times N_1$ `CORAL` structure into those in a $M_2 \times N_2$ `CORAL` structure.

1.4 Correlating Record Sections

- `raCrossCorr.m`: Correlates record sections **or** seismogram pairs together. For $M \times N$ `CORAL` structure arrays, correlation is computed between N record sections by translation in one dimension, using a generalized cross correlation coefficient. Shift indices giving maximum correlation between N record sections are output as well.
- `raGramMat.m`: Computes the Gram matrix for $M \times N$ `CORAL` structure arrays. In other words, no shifting is done.

- `coralCrossCorr.m`: Correlates seismograms together as pairs (standard cross correlation).

1.5 Shifting Record Sections

- `coralCircShift.m`: Shifts data using input shift index. Takes shift output from raCrossCorr.m. See example in help menu.
- `coralSubShift.m`: Shifts data to subsample precision. More general version of `coralCircShift.m` and uses additional function `fshift.m`.
- `subRaCrossCorr.m`: Fine tunes maximum correlation between `CORAL` structure array of record sections to subsample precision. Outputs optimum shift index, and cell array of correlograms. Input record sections should first be aligned to integer sample precision.

1.6 Stacking Record Sections

- `coralRaStack.m`: Stacks an $M \times N$ `CORAL` structure array by columns. Output is an $M \times 1$ record section.
- `stackRaClusters.m`: Collects record section clusters and stacks seismograms. Saves `.mat` file of stacks.

1.7 Clustering Record Sections

When using some of these functions, I advise the user to first normalize their seismograms to give each time series a uniform weighting. To perform this operation on an $M \times N$ `CORAL` structure array use `S = arrayfun(@coralNormData,S)` for immediate results. The higher level functions that operate of `.mat` files do this internally, so no redundant processing needs to be executed.

- `xcorMutClust.m`: Identifies and aggregates multiplet clusters from $M \times N$ `CORAL` structure arrays. Uses `raCrossCorr.m` internally to identify clusters. Workhorse function for remainder of clustering operations.
- `coralCluster.m`: Uses `xcorMutClust.m` to identify cluster waveforms, remove possible redundancies, and output results of K clusters in an organized $K \times 1$ cell array.

- `raPicks2raCluster.m`: Takes pick date files (see chapter 4) and computes and saves cluster `.mat` files.
- `getRaClusters.m`: Combines cluster files (see chapter 7), aggregates, and saves cluster `.mat` files.

1.8 Synthesizing Record Sections

This section concerns methods for building or decomposing record sections into linear combinations (or nonlinear combinations) of other record sections. Some of these functions operate on vectors and matrices, not necessarily `CORAL` structures. I request you acknowledge the methodology if your publication is derived from the clustering and conic projection methods. I have not yet published the novel components of the conic projection method, so there is no literature reference for it as of 10, April 2010.

- `matchPurs.m`: Uses the matching pursuit algorithm to decompose a signal vector, or matrix (spectrogram, for example) using a 'dictionary' of signals into a linear combination of other signals. The energy of the signal is preserved, though dictionary elements do not have to be orthogonal. More general than Fourier analysis.
- `projCones.m`: Projects set of signal onto a convex cone.
- `projConesStruc.m`: Projects `CORAL` structure record section onto dictionary of record sections.
- `projOnClusters.m`: Projects a vector or signal onto a dictionary of cones/clusters.
- `coneDecompPursuit.m`: A matching pursuit algorithm generalized from subspace projections convex cone projections.

Chapter 2

Introduction

I wrote this tutorial to provide an instructional companion to a specific family of MATLAB functions that are used in seismology applications. These functions share a common feature in that they use input arguments in the form of "CORAL" structures. A structure is a MATLAB data type that provides the means to store a variety of data objects together. The family of `CORAL` functions were originally created by Ken Creager at the University of Washington. Several researchers and students have built upon this infrastructure and have developed `CORAL` functions in a community effort to create a set of seismic processing tools. This tutorial is *not* a comprehensive review of using `CORAL`. It is a user's manual for a subset of functions I have created to do specific clustering-based processing on sets seismic array data.

2.1 What These Functions Will Do For You

The user begins with a set of `.SAC` day volumes in a standard IRIS/PASSCAL file format, and after processing, will have compiled:

1. A set of `.mat` files containing dates and times for each seismic event observed in the record.
2. A set of `.mat` files containing array-picks. This is a set of `CORAL` structures containing the seismograms of those events that are observed on every seismometer in the array.
3. A set of `.mat` files containing the dates and times corresponding to a combinatorial count for all seismic event pairs, triplets, etc... observed between stations.
4. A set of `.mat` files containing cell arrays of `CORAL` structures for multiplet events, or clusters for each day. These clusters are identified by waveform similarity at all

stations.

5. A set of `.mat` files containing the seismic event count score over the entire record.
6. A set of `.mat` files containing the clusters observed over the entire record.

The package of `CORAL` functions furnished previously (Creager, Thelen) generally operate on structures containing individual seismic traces or single record sections. I take an array-processing approach in the design of `CORAL` functions. That is, I view an entire record section for an event as a single observation, conducive with my primary goal of identifying multiplet events and clusteirng. Thus, I treat the record section as the operand in the processing, as opposed to isolated receivers. Therefore, this supplement provides `CORAL` tools that generally take an $M \times N$ array of extttCORAL structures as a typical argument, so that N record sections consisting of simultaneous observation at M receivers may be processed at once. In abstract terms, I am taking the analysis chore to an M-fold Hilbert space. The computational consequence is invisible to the user, and the increase in complexity is low. I have made heavy use of vectorized operations in MATLAB, as well as array, structure, and cell based functions that utilize function handles to avoid using loops, whenever possible (within reason, of course).

2.2 Quick Examples

I have a provided a small data set in this package in order to provide tangible examples of output from the `CORAL` functions contained herein. In principle, the commands I provide in this section will produce a set of processed data files containing pick times, clusters of record sections, their stacks, and count statistics. In reality, few errors are likely to be produced prior to correct implementation. These errors may be due to your MATLAB path. I advise you initially ensure that the `.m` files `setCoralFields.m` and `defaultOptStruc` must be in the same directory as your data files, whereas the processing tools must be added to your path.

2.3 Produce Processed Data `.mat` Files

To produce the processed data files, type the following commands from the MATLAB command prompt, or better yet, in an `.m` file you can run the following commands. I have given a description of the command as a comment preceding the command.

```
%loads parameters structure to be used in following MATLAB functions
opt = defaultOptStruc;
year = 2005;
```

```
dayvec = 345:346

%generates set of p-wave picks
[pTime] = sac2RaPicks(2005,dayvec,opt);
for k = 345:346

%gathers seismograms that are observed across entire array
[S,T] = pickDates2arraypicks(year,k,opt);

%generates a count score for the number of events
[T,S] = pickDates2countmetric(year,k,opt);

%finds clusters, or multiplets
[Scluster,Sorphan] = raPicks2raCluster(year,day,opt);
end;

%collects clusters found with raPicks2raCluster across experiment period
Scluster = getRaCluster(opt);

%coherently stacks, at subsample precision, mulitplet waveforms within clusters
Sstack = getRaClusterStack(opt);

%collects history of seismic events generated by pickDates2countmetric
Scluster = getCountMetric(opt);
```

You should check the contents of the current directory to verify that new processed data files have been written.

2.4 Useful Plotting Commands

In order to visualize the processed data, the user can utilize some of the plotting functions such as `coralPlot`, `plotRaClusterStacks.m`, `plotCountMetric.m`, etc.... Some of these functions take different inputs, so as always, ensure to call `help` for the utility. `Example`: in order to visualize the results of counting events, try the following command:

```
% load countmetric structure for plotting
load 'COUNTHISTORY.ARRAY.024.2006.345.346.mat';
% plot countmetric time series, spectra, and statistics
[h] = plotCountMetric(countmetric);
```

To view the linear stack of the vertical channel seismograms populating the multiplet clusters and their timing for station JAN, type:

```
% load record section cluster cell, Scluster
load 'CLUSTERHISTORY.ARRAY.065.2006.345.346.mat';
% plot the JAN.EPZ seismograms in each cluster
[h] = plotRaClusterStacks(Scluster);
```

Note: I am giving you a small sample of example data, so the plots are providing a window for a sparse data set. I designed these functions for application to much larger data sets. My hope is these examples provide an illustration of how to use the functions in more general cases.

Chapter 3

Preparing Your Data

The following sections provide a sequential set of instructions for the user to prepare their data. Each includes a list of the concerning functions for quick reference.

3.1 SAC Day Volume Format

The seismic data must be in `.SAC` format. The file naming convention relies on the PASSCAL/IRIS file format, which is of the form:

STACODE.*.*.CHN.YYYY.*JJJ.*SAC

When extracting your miniseed or seed volumes from a data logger, you typically do not have to think about this process. However, the naming is important because the MATLAB functions that read in the seismic data identify the unique seismogram using the receiver name, channel, and the date provided by the file name. The * symbols indicate wildcat values, i.e., any text is permissable. The remaining symbols above are:

- **STACODE**: The station code for the receiver, sometimes called the receiver name. It is a maximum of 5 alpha-numeric characters long. **Example**: SHUK. This is the station code given for a Mt. Baker Ski Area seismometer.
- **CHN**: The channel name. These are three letter characters, giving the band code, instrument code, and orientation of the channel. **Example**: BHZ indicates the channel corresponds to broad band, high gain, vertical motion
- **YYYY**: The year the data was recorded. **Example**: 2006

- **JJJ**: The three-letter Julian day the data was recorded. **Example**: 345 is November 29 for regular years.

Associated Functions: `pascSac2Coral.m`, `coralReadSAC.m`

3.2 Directory Structure

The only requirement on the directory structure is that the `.SAC` files that you wish to process for a given experiment are in a single directory, together. The processed `.mat` files will be saved in this directory. You may have multiple experiments with multiple `.SAC` files populating the directory. It goes without saying that the `CORAL` functions must be in a directory that is part of the current MATLAB path.

3.3 Creating a Default Optional Structure

Many of the original `CORAL` functions use an optional "opt" structure as a variable function input. It is most convenient in my view to have one structure that can be loaded and used for all the `CORAL` functions. Then it is a matter of adding fields to the structure. For this reason, I have provided a function `defaultOptStruc.m` which can be edited and saved, much like a text file of model parameters. The difference here is that it is ran like a function, but no inputs are required. That is, it is used by simply typing:

```
opt = defaultOptStruc;
```

at the MATLAB command line. `defaultOptStruc.m` is used by many functions, for possibly different purposes. The structure `opt` has many fields, which can be edited and viewed in the m-file Editor. Each field of `opt` is commented. The current inventory of field, with default parameters, are:

```
opt =
```

```
thresh:         0.6500
maxNumClust:    200
localTime:      46800
cutTime:        5
fcut:           0.3500
chan:           'EPZ'
level:          6
rsec:           0.5000
tbin:           3600
nWins:          6
sWindow:        0.5000
lWindow:        2.5000
rCutoff:        3.2000
nInterval:      5
test:           0
eqLocPrec:      1.0000e-06
uncertainty:    [6x1 double]
eqOriginTime:   0.4000
recSampInt:     0.0050
maxResid:       1.0000e-06
nIteration:     100
plotopt:        'plot'
bitWeight:      2.3840e-06
staGain:        1
sensitivity:    30.4000
```

There several practical cases in which you would need to edit `opt` either as a structure in the command window, or quasi-permenantly in the function `defaultOptStruc.m`, in order to suit the format of your specific data set. Here are some **Examples:**

- You have data with BH* channels, not EP* channels. Edit the line `opt.chan = EPZ;` to read `opt.chan = BHZ;`.
- You would like to use a 5 second long term average parameter in the p-wave picker `pickerRatio.m` when picking (all) your data. Edit the line `opt.lWindow = 2.5;` to read `opt.lWindow = 5;`.

Other editing needs will become obvious in your processing, as each function has a verbose and complete help menu.

I advise that this function must be saved in the data directory, *not* the processing tools directory. Different sets of data with their own `.SAC` files will likely need a distinct opt

structure.

Associated Functions: `defaultOptStruc.m`

3.4 Creating a Default `CORAL` Structure

Sometimes the station names appearing in the `staCode` field or geographical locations of the receiver appearing in the `staLat`, `staLon` and `staElev` fields do not get read into the `CORAL` structure from the `.SAC` file headers. To populate the `CORAL` structure fields, you will create no-input function called `setCoralFields.m` and save it in the directory with the `.SAC` files. Like `defaultOptStruc.m`, I have provided a template. It can be viewed by simply typing:

```
S = setCoralFields;
```

from the MATLAB command line. The example I have provided gives a 6-by-1 `CORAL` structure array `S` with a set of (mostly blank) fields. `S` could provides the template for a 6 station seismic array with station codes `BOBBY`, `CINDY`, `GREG`, `JAN`, `MARSH` and `PETER`. Access a given row to view the structure contents. **Example**:

```
S(4,1)
ans =
```

```
staCode:            'JAN'
staChannel:         ''
staType:            ''
staNetworkCode:     'XA'
staLocationCode:    ''
staQualityCode:     ''
staLat:             -77.7300
staLon:             162.2432
staElev:            148.1130
staRespType:        ''
staGain:            1
staNormalization:   1.2741
staPoles:           [2x1 double]
staZeros:           [2x1 double]
eqLat:              ''
eqLon:              ''
eqDepth:            ''
eqOriginTime:       ''
eqComment:          ''
recNumData:         ''
extras:             ''
data:               ''
recSampInt:         0.0050
recMaxAmp:          ''
recStartTime:       ''
recComment:         ''
staEasting:         4.3406e+05
staNorthing:        1.3709e+06
```

The channel names are not provided here since these fields were correctly recorded. The `CORAL` structure for a complete triaxial array observation would be an 18-by-1 structure array. The user must open the function in the m-file editor and write in the station names, latitudes, longitudes, elevations, etc... into the corresponding fields of the `CORAL` structure for each receiver in the array; the channels do not have to be hard-written in for most cases. Again, the function serves the purpose of a parameter file, but makes use of the structure array data type in MATLAB. For example, the value recorded by the field `staElev` provides the station elevation. You will need to use the included `deg2utm.m` to add two additional fields `staNorthing` and `staEasting`, as in the template `setCoralFields.m` provided. These simplify distance calculations.

Associated Functions: `setCoralFields.m`, `coralSetNames.m`, `coralSetRefs.m`

Chapter 4

Identifying Seismic Events

This section provides the information necessary to compute p-wave picks for each receiver in your array. The challenge of picking first arrivals, sometimes called p-wave picking, continues to be addressed with different signal processing methods or algorithms. For most typical data sets, a methodology based upon the parameterized short-term average to long term average STA-to-LTA algorithm is sufficient. Certainly there are specialized applications in which this approach is less than ideal (non-volcanic tremor, high noise environments). I confront p-wave picking using the parameterized STA-to-LTA method due to my personal success and it's usage with volcanic seismology. Interested users should contact the author of the picker I implement here (Wes Thelen, Cascade Volcano Observatory, Vancouver WA) for questions on the utility of this genre of pickers.

4.1 Generating Pick Files

You now should have a set of `.SAC` day volumes, and two `.m` files in the same data directory, `defaultOptStruc.m` and `setCoralFields.m`. For simplicity, consider the 6 station array I have provided in the template `setCoralFields.m`. Suppose the data file directory containing the `.SAC` files have 2 days worth of data (pretty small data set!), recorded from November 29 2005 to November 30 2005. The contents of you data directory would then appear as (note the `.SAC` file naming convention appears as mentioned above):

```
defaultOptStruc.m              setCoralFields.m
BOBBY.XA.02.EPE.2005.345.SAC   JAN.XA.02.EPE.2005.345.SAC
BOBBY.XA.02.EPN.2005.345.SAC   JAN.XA.02.EPN.2005.345.SAC
BOBBY.XA.02.EPZ.2005.345.SAC   JAN.XA.02.EPZ.2005.345.SAC
CINDY.XA.02.EPE.2005.345.SAC   MARSH.XA.02.EPE.2005.345.SAC
CINDY.XA.02.EPN.2005.345.SAC   MARSH.XA.02.EPN.2005.345.SAC
CINDY.XA.02.EPZ.2005.345.SAC   MARSH.XA.02.EPZ.2005.345.SAC
GREG.XA.02.EPE.2005.345.SAC    PETER.XA.02.EPE.2005.345.SAC
GREG.XA.02.EPN.2005.345.SAC    PETER.XA.02.EPN.2005.345.SAC
GREG.XA.02.EPZ.2005.345.SAC    PETER.XA.02.EPZ.2005.345.SAC
BOBBY.XA.02.EPE.2005.346.SAC   JAN.XA.02.EPE.2005.346.SAC
BOBBY.XA.02.EPN.2005.346.SAC   JAN.XA.02.EPN.2005.346.SAC
BOBBY.XA.02.EPZ.2005.346.SAC   JAN.XA.02.EPZ.2005.346.SAC
CINDY.XA.02.EPE.2005.346.SAC   MARSH.XA.02.EPE.2005.346.SAC
CINDY.XA.02.EPN.2005.346.SAC   MARSH.XA.02.EPN.2005.346.SAC
CINDY.XA.02.EPZ.2005.346.SAC   MARSH.XA.02.EPZ.2005.346.SAC
GREG.XA.02.EPE.2005.346.SAC    PETER.XA.02.EPE.2005.346.SAC
GREG.XA.02.EPN.2005.346.SAC    PETER.XA.02.EPN.2005.346.SAC
GREG.XA.02.EPZ.2005.346.SAC    PETER.XA.02.EPZ.2005.346.SAC
```

To modify the picking criteria, type `help pickerRatio.m` to determine how the picking criteria is chosen, and modify the respective fields in the function `defaultOptStruc.m`. This picking criteria will now be implemented. To obtain the pick date `.mat` files over Julian days 345 through 346, type at the MATLAB command window:

```
opt = defaultOptStruc;
year = 2005;
dayvec = 345:346
[pTime] = sac2RaPicks(2005,dayvec,opt);
```

The verbose screen output illustrates the naming convention for the `.mat` files. This convention is designed to afford the minimal amount of information required for a unique file name. The function `sac2RaPicks.m` automatically determines picks for all available stations in your directory; there are options for picking individual stations, which can be viewed from the help menu. Once the function is through running (may take some minutes), view the pick dates on Julian day 345 for station `JAN` (for example) by loading the respective `.mat` file:

```
load PICKDATES.JAN.EPZ.2005.345.mat;
```

The pick date variable is a cell array `pTime`.

Associated Functions: `pascSac2Coral.m`, `sac2RaPicks.m`, `pickerRatio.m`, `defaultOptStruc.m`

Chapter 5

Gathering Array Observations

This section provides the information necessary to extract the `CORAL` structures for the large seismic events. Your pick dates were computed for each station on a single channel. You may be primarily interested in the largest seismic events. To extract the `CORAL` structure (seismograms) for the events that were observable on *every* array element for each day on every channel, you will use the function `pickDates2arraypicks.m`. I continue with the example provided and the 2 day data set. To extract the array observations, type:

```
year = 2005;
opt = defaultOptStruc;
for k = 345:346
[S,T] = pickDates2arraypicks(year,k,opt);
end;
```

This function generates a set of `.mat` files with the file format:

ARRAYPICKS.CHN.YYYY.JJJ.mat

The channel code CHN indicates which channel was used to generate the pick dates. In the example I have provided, only 2 `.mat` files are produced, one for each day:

```
ARRAYPICKS.EPZ.2005.345.mat
ARRAYPICKS.EPZ.2005.346.mat
```

Each `.mat` file is saved as a structure named `arraypicks` with 2 fields, `coralstruc` and `pTime`. To view the seismograms corresponding to day 345, for example, type the following commands at the MATLAB command prompt:

```
S = load('ARRAYPICKS.EPZ.2005.345.mat');
S = S.arraypicks;
S = S.coralstruc;
coralPlot(S(:,1));
```

In summary, a set of `CORAL` structures now exist for every event that was picked by all receivers in the array.

Associated Functions: `getPickDates.m`, `clusterdates.m`, `uniquePickDates.m`, `setCoralFields.m`, `coralSetNames.m`, `coralTaper.m`, `coralDetrend.m`, `pascSac2Coral.m`, `pickDates2arraypicks.m`, `defaultOptStruc.m`;

Chapter 6

Counting Events (Optional)

This section provides the information required to enumerate your seismic pick dates. This section is largely **optional** because clustering does not require any information on the number of events. I have included this section because I find the method particularly helpful in determining timing of seismic events within the array aperture, and have provided the requisite `CORAL` functions. You may elect to chose an alternative method to count events.

I have provided background information in the appendix to illustrate the quantitative definition of the count score. I think it sufficient to state here that it assigns a count score to each event based upon the geometry of the array, and the number of stations the event was picked on (detected). Events observed on a single station are not counted, and events observed on every station within the array are given a count of 1. The function `plotDistMetric.m` is a useful tool for visualizing the count weights, and includes explanation within the help menu.

6.1 Constructing a Count Score File

To generate a set of counts for each day, run the following commands from MATLAB prompt:

```
year = 2005;
opt = defaultOptStruc;
for k = 345:346
[T,S] = pickDates2countmetric(year,k,opt);
end;
```

Notice this function's implementation is identical to that of `pickDates2arraypicks.m`. Upon completion, it generates a set of .mat files with the file format:

COUNTMETRIC.ARRAY.NNN.YYYY.JJJ.mat

The triplet NNN has fixed field width of 3, and indicates the samples-per-day the counts were binned over. In the example I provided, the `opt` structure specifies that the count be binned hourly. Two `.mat` files are produced, one for each day:

```
COUNTMETRIC.ARRAY.024.2005.345.mat
COUNTMETRIC.ARRAY.024.2005.346.mat
```

Each `.mat` file contains a structure `countmetric` with fields `binLoc`, `hist`, `stations`. The separate structures are most conveniently interpreted after they are processed with functions `getCountMetric.m`, then `plotCountMetric.m`, and/or `sumCountmetric.m`. To run `getCountMetric.m`, type:

```
opt = defaultOptStruc;
countmetric = getCountMetric(opt);
```

This generates another `.mat` file with file format:

COUNTHISTORY.ARRAY.YYYY.JJJ.JJJ.mat

The first triplet JJJ describes the Julian day corresponding to the first `COUNTMETRIC*.mat` file and the latter JJJ describes the Julian day corresponding to the last `COUNTMETRIC*.mat file` used to generate the count metric that is saved in the `.mat` file. The example I have provided generates the file:

```
COUNTHISTORY.ARRAY.024.2005.345.346.mat
```

This file contains a structure `countmetric` whose fields are discussed in the help menu. A complete description is illustrated using `help getCountMetric.m`

6.2 Plotting the Count Score

To plot the resulting countmetric, type:

```
opt = defaultOptStruc;
```

```
h = plotCountMetric(opt);
```

If you have cleared variale `countmetric` you must reload it via:

```
load('COUNTHISTORY.ARRAY.024.2005.345.346.mat');
```

To extract the count score time series and respective dates, type:

```
[t,x] = sumCountMetric(countmetric);
```

Again, here `countmetric` is the output extracted from `getCountMetric` above. I recommend usage of the Statistics Toolbox functions to implement additional processing on the count metric time series.

`Notes`: Whenever working the `datenum` values, it is useful to convert to date vectors via `datevec` and use functions `timeadd` or `timediff`. **Do not** use `datenum` on data where milli-second level precision is necessary.

Associated Functions: `getPickDates.m`, `clusterdates.m`, `uniquePickDates.m`, `setCoralFields.m`, `coralSetNames.m`, `stationList2cluster.m`, `picks2distmetric.m`, `pickDates2countmetric.m`, `defaultOptStruc.m`, `plotDistMetric.m`, `getCountMetric.m`, `plotCountMetric.m`, `sumCountmetric.m`, `nchoose.m`

Chapter 7

Unsupervised Clustering of Record Sections

This section provides the information necessary to obtain the seismograms for mutliplet seismic events. In short, the methodology implements a partition-based, agglomerative clustering method to recognize seismic multiplets within event record sections. My purpose here is ensure that the user understands what aspect of the data is being clustered. I defer the mathematics and geometry to the appendix; here I provide a *quantitative* characterization of this clustering via the geometry of convex set projections

The clustering is performed on the `ARRAYPICKS.*.mat` files that contain the seismograms for the array-wide observations, which were constructed in chapter 5. Each `ARRAYPICKS.*.mat` stores an $M \times N$ `CORAL` structure `S` for the N events observed that day. These structures contain the MN seismograms recorded at M receivers. The structure containing the record section for event k is then `S(:,k)`. The cross-correlation between the record section described by `S(:,k)` and another described by `S(:,n)` gives a measure of the similarity in both the waveform geometry and the differential phasing between the M receivers. Thus each *pair of record sections*, not seismic traces, has an associated cross-correlation coefficient $C(k, n)$. If any number of record sections *mutually* cross correlate above a fixed absolute value, then these record sections belong to a cluster. I employ a partition-based clustering methodology. This mean that no record section with membership to a cluster populates any another cluster, so that there is no ambiguity in cluster uniqueness. Again, I have provided quantitative details in the appendix.

7.1 Clustering Tools

To implement this strategy, the user must have the obtained `ARRAYPICKS.*.mat` files as I stated above. To cluster over days, using the example files I have provided, type:

```
opt = defaultOptStruc;
year = 2005;
for k = 345:346,
[Scluster,Sorphan] = raPicks2raCluster(year,day,opt); end;
```

As with functions like `pickDates2arraypicks.m`, there is no `CORAL` structure input, just the `opt` structure. The function looks in the current directory for the `ARRAYPICKS.*.mat` files. Upon completion, a set of files are written to the current directory of the form:

CLUSTERPICKS.ARRAY.NNN.YYYY.JJJ.mat

The triplet of numbers NNN indicates the normalized corrleation value used to cluster the array picks, expressed as a percentage. In the example I have provided, the input structure `opt` contains a field `thresh` that stores the correlation coefficient. By default, `opt.thresh = 0.65`. Thus, the following files are written to the current directory:

```
CLUSTERPICKS.ARRAY.065.2005.345.mat
CLUSTERPICKS.ARRAY.065.2005.346.mat
```

Each of these `.mat` files contains a $P \times 1$ cell array. A cell array is a organization of elements called cells, in which distinct types of data can be stored. Row k of the cell array stored in `CLUSTERPICKS.ARRAY.*.mat` contains an $M \times N_k$ `CORAL` structure. The N_k columns correspond to distinct record sections, which all mutually correlate above the threshold value specified in `opt.thresh`. The clustering may be taken a step further. To obtain clusters for the entire experiment period, the function `getRaClusters.m` combines the cluster sets together and aggregates any record sections together that belong to the same cluster. Type:

```
[Scluster,Sorphan] = getRaClusters(opt);
```

Upon completion, the following file is written into the current directory:

```
CLUSTERHISTORY.ARRAY.065.2005.346.mat
```

This one file describes the set of *distinct* multiplets observed during the experiment.

Associated Functions: `raPicks2raCluster.m`, `coralCluster.m`, `xcorMutClust.m`, `xcorMutClustOld.m`, `coralNormData.m`, `raCrossCorr.m`, `raGramMat.m`, `raGramMat.m`, `plotRaClusterStacks.m`

Chapter 8

Appendix

This chapter details the quantitative details of partition-based, mutually exclusive, record section clustering. I use concepts from functional analysis, but it can be understood from a linear algebra perspective to less seasoned signal processors.