InfraPy Documentation

Release 1.0

F.Dannemann Dugick, O.Marcillo, P.Blom, J.Webster,

Jun 05, 2020

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 Authorship

Infrapy was built upon previous similar (InfraMonitor) tools and developed by the LANL SeismoAcoustic Team.

Omar Marcillo omarcillo at lanl.gov

Philip Blom pblom at lanl.gov

Jeremy Webster jwebster at lanl.gov

Fransiska Dannemann Dugick fransiska at lanl.gov

1.2 Installation

1.2.1 Operating Systems

Infrapy can currently be installed on machines running newer versions of Linux or Apple OSX. A Windows-compatible version is in development.

1.2.2 Anaconda

The installation of infrapy currently depends on Anaconda to resolve and download the correct python libraries. So if you don't currently have anaconda installed on your system, please do that first.

Anaconda can be downloaded from https://www.anaconda.com/distribution/. Either 3.x or 2.x will work since the numbers refer to the Python version of the default environment. Infrapy's installation will create a new environment and will install the version of Python that it needs into that environment.

1.2.3 Infrapy Installation

Once Anaconda is installed, you can install infrapy by navigating to the base directory of the infrapy package (there will be a file there named infrapy_env.yml), and run:

```
>> conda env create -f infrapy_env.yml
```

If this command executes correctly and finishes without errors, it should print out instructions on how to activate and deactivate the new environment:

To activate the environment, use:

```
>> conda activate infrapy_env
```

To deactivate an active environment, use

```
>> conda deactivate
```

1.2.4 Testing

Once the installation is complete, you can test some things by first activating the environment with:

```
>> conda activate infrapy_env
```

Then navigate to the /example directory located in the infrapy base directory, and run the test scripts via something like:

```
>> python test_beamforming.py
```

If infrapy was successfully installed, all of the test scripts should run and finish without any errors.

1.2.5 Running the InfraView GUI Application

Once installation is complete, and the new environment is activated, you can run the GUI with the command:

```
>> infraview
```

1.3 Quickstart

A series of scripts illustrating how to use infrapy subroutines as stand-alone modules are found in the /examples folder. The jupyter notebook documenting these steps is found in /tutorials/Quick Start.ipynb. The notebook can be run by installing jupyter notebook via conda.

```
>> conda install jupyter notebook
```

Beamforming:

1. Run Bartlett, Capon or Generalized Least Squares beamforming processes on an hour-long dataset from the BRP array in Utah

```
>> python test_beamforming.py
```

2. Visualize beamforming results in the sx/sy space

```
>> python test_slowness-grid.py
```

Detection:

1. Run detection on the series of beamforming results produced in the above step

```
>> python test_detection.py
```

Association

1. Associate a number of detections contained in a .dat file (/data/detection_set1.dat or /data/detection_set2.dat)

```
>> python test_assoc.py
```

Location

1. Test the Bayesian Infrasonic Source Localization (BISL) methodology using a set of provided detections (/data/detection_set1.dat or /data/detection_set2.dat). Location will be run twice, once assuming uniform atmospheric propagation and a second time applying provided atmospheric propagation priors for the Western US (see Blom et al., 2015 for further explanation)

```
>> python test_bisl.py
```

1.4 Interfacing with Pisces

Infrapy leverages pisces to connect with and process data in either local sqlite databases or oracle databases. More information about pisces can be found at https://jkmacc-lanl.github.io/pisces/.

1.4.1 Converting Data into Sqlite Databases

Data in miniseed or sac formats can be loaded into a sqlite database for pipeline processing using commands from pisces.

1. mseed to database (ms2db.py)

```
>> ms2db.py sqlite:///example.sqlite mslist.txt
```

2. sac to database (sac2db)

```
>> pisces sac2db sqlite:///example.sqlite *.sac
```

As infrapy is an array processing tool, after your sqlite database is created, you will need to update the REFSTA for each array using update_refsta.py

```
>> update_refsta.py sqlite:///example.sqlite <array name>
```

You can update the calibration for each array using update_calib.py

```
>> update_calib.py sqlite:///example.sqlite <array name> <calibration>
```

1.4.2 Connecting to a SQL Database

Infrapy employs two main methods for connecting to either Oracle or sqlite databases. Example files to facilitate these connections are found in tutorials/.

1.4.2.1 Defining Schema Specific Tables

Pipeline processing in infrapy utilizes information from CSS3.0 Site and Wfdisc tables. If your database schema differs from the CSS3.0 schema in any way, you can define the differences using a _global.py file. An example _global.py file is found in tutorial/ .

1.4.2.2 Connection within pipeline processing configuration file

The first three lines of your configuration file define the database you will connect to:

Example Configuration File for Sqlite Processing (Sqlite_Config.txt)

```
[database] # required
# url to database where you have the pointers to data and metadata
url = sqlite:///example.sqlite
# schema specific tables for your site and wfdisc files.  If you are processing in a␣
↪sqlite database, these variables will refer to schema specified in pisces. If you␣
↪are processing in an oracle database, these variables will refer to schema␣
↪specified in your global_.py file
site = pisces.tables.css3:Site
wfdisc = pisces.tables.css3:Wfdisc
```

Example Configuration File for Oracle DB Processing (Oracle_Config.txt)

```
[database] # required
url = oracle://<database name>:<port>
site = global_:Site
wfdisc = global_:Wfdisc_raw
```

1.4.2.3 Connection with a db.cfg file

Some modules in infrapy (db2sac) require a .cfg file to establish connection with a database. Examples are found in tutorial/ . More information can be found in the pisces documentation.

Example Configuration File for Oracle DB Processing (oracle_connection.cfg)

```
[database] # required
url = oracle://<db name>:<db port>
site = global_:Site
wfdisc = global_:Wfdisc_raw
origin = global_:Origin
```

Example Configuration File for Sqlite Processing (sqlite_connection.cfg)

```
[database] # required
url = sqlite:///example.sqlite
site = pisces.tables.css3:Site
wfdisc = pisces.tables.css3:Wfdisc
origin = pisces.tables.css3:Origin
```

1.5 Algorithms

- Analyst methods are modular so that results from other processing tools can be used in later analysis steps.
- Algorithms are written to be data agnostic so analysis can be performed regardless of data source once IO method is established

1.5.1 Station Level Processing

1.5.1.1 Array Processing

- Beamforming estimates parameters of coherent signals
- Capabilities include methods to characterize transients as well as persistent signals
- Transient signals are identified using standard Bartlett beaming
- Persistent signals can be investigated using Minimum Variance Distortionless Response (MVDR) or MUltiple Signal Classification (MUSIC) algorithms

1.5.1.2 The Adaptive F-Detector

- Adaptive Fisher statistics determine when to declare a detection

1.5.2 Network Level Processing

1.5.2.1 Association

- Events are identified using a pair-based Bayesian algorithm that defines the association between detection pairs from their joint-likelihood and identifies events via hierarchical clustering analysis
- Current implementation utilizes only spatial and temporal coincidence, but additional detection information can further improve event identification
- Evaluation using a synthetic data set shows some mixing of spatially similar events poorly resolved by network geometry and occasional inclusion of "noise" detections in event clustering

1.5.2.2 Bayesian Infrasonic Source Localization

- Event analysis using the Bayesian Infrasonic Source Localization (BISL) methodology to estimate both the spatial location of the event as well as the origin time with quantified uncertainty
- Preliminary analysis of the back projections can be used to define the spatial region of interest or it can be specified by the analysis
- Analysis identifies the maximum posteriori solution
- The marginalized spatial distribution is approximated as 2d-normal to define 95 and 99% confidence ellipse bounds
- The marginalized temporal distribution is analyzed to identify the exact 95 and 99% confidence bounds
- Likelihood definitions relating detection parameters to spatial and temporal source characteristics are shared between association and localization analysis for consistency

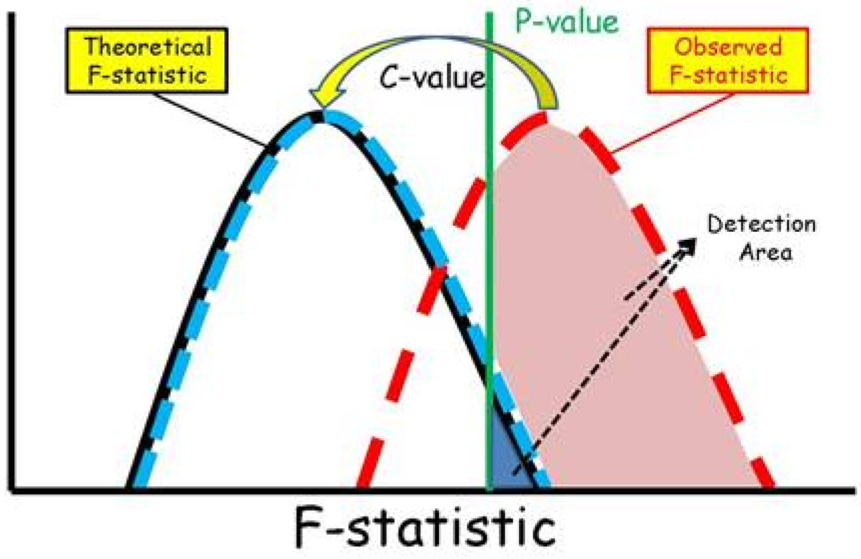
1.5.2.2.1 Array Processing

The use of infrasonic arrays, specifically for CTBT applications, is preferable due to the inherent reduction in signal-to-noise ratios (SNR) originating from the summation of four or more recordings at each array. The nature of a decision-rule based detector requires data that has been pre-processed using beamforming methods. Beamforming, a form of array processing, is the first step in the analysis of data from infrasonic arrays. Conventional beamforming methods (Bartlett, Capon) separate coherent and incoherent parts of a signal through the assumption of planar waves arriving at the array. A signal backazimuth and slowness can be estimated as signals are shifted to account for travel time differentials across array elements, bringing the signal into phase across as the noise deconstructively cancels out. In the classical, or Bartlett methodology , data records on each array element are time-shifted versions of the other with local noise,

See the following for more references on beamforming: Rost and Thomas 2002 Olson and Szuberla 2010 Costley 2013

1.5.2.2.2 The Adaptive F-Detector

The standard F-detector is based on decision rules for the F-statistic, which provides an estimate of the signal's beam power and is calculated as the power in the beam divided by the average over all channels of the power difference between the beam and the individual channels cite{Blandford:1982}. The AFD accounts for both correlated and uncorrelated noise through an increase in the value of the F-statistic required to declare a detection based upon background F-values being elevated from coherent or persistent noise sources. The figure below illustrates how the AFD remaps the F-statistic through the application of a C-value, which effective reduces the detection threshold (p-value) and decreases the number of noise-related detections.



See the following for more references on the Adaptive F-Detector:

Arrowsmith et al., 2008 Arrowsmith et al., 2009

1.5.2.2.3 Association

- Events are identified using a pair-based Bayesian algorithm that defines the association between detection pairs from their joint-likelihood and identifies events via hierarchical clustering analysis
- Current implementation utilizes only spatial and temporal coincidence, but additional detection information can further improve event identification
- Evaluation using a synthetic data set shows some mixing of spatially similar events poorly resolved by network geometry and occasional inclusion of "noise" detections in event clustering

See the following for more references on Association: Blom et al., 2020

1.5.2.2.4 Bayesian Infrasonic Source Localization

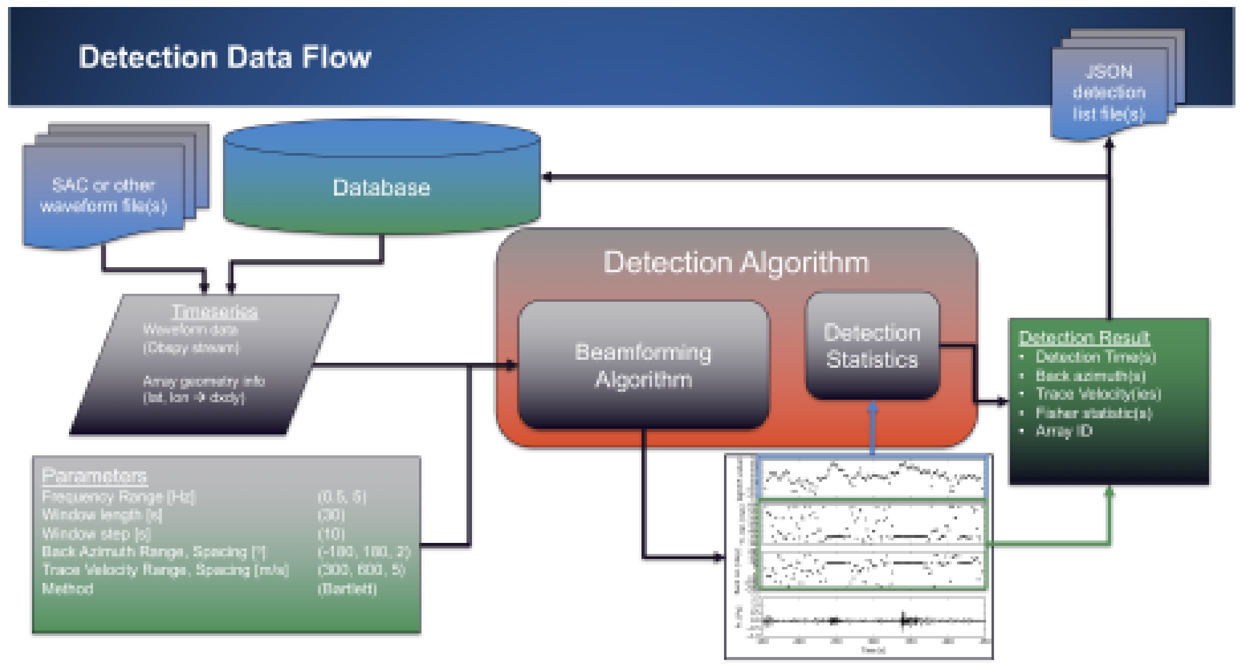
- Event analysis uses the Bayesian Infrasonic Source Localization (BISL) methodology to estimate both the spatial location of the event as well as the origin time with quantified uncertainty
- Preliminary analysis of the back projections can be used to define the spatial region of interest or it can be specified by the analysis
- Analysis identifies the maximum posteriori solution
- The marginalized spatial distribution is approximated as 2d-normal to define 95 and 99% confidence ellipse bounds
- The marginalized temporal distribution is analyzed to identify the exact 95 and 99% confidence bounds
- Likelihood definitions relating detection parameters to spatial and temporal source characteristics are shared between association and localization analysis for consistency

See the following for more references on BISL: Modrak et al., 2010 Marcillo et al., 2013 Blom et al., 2015

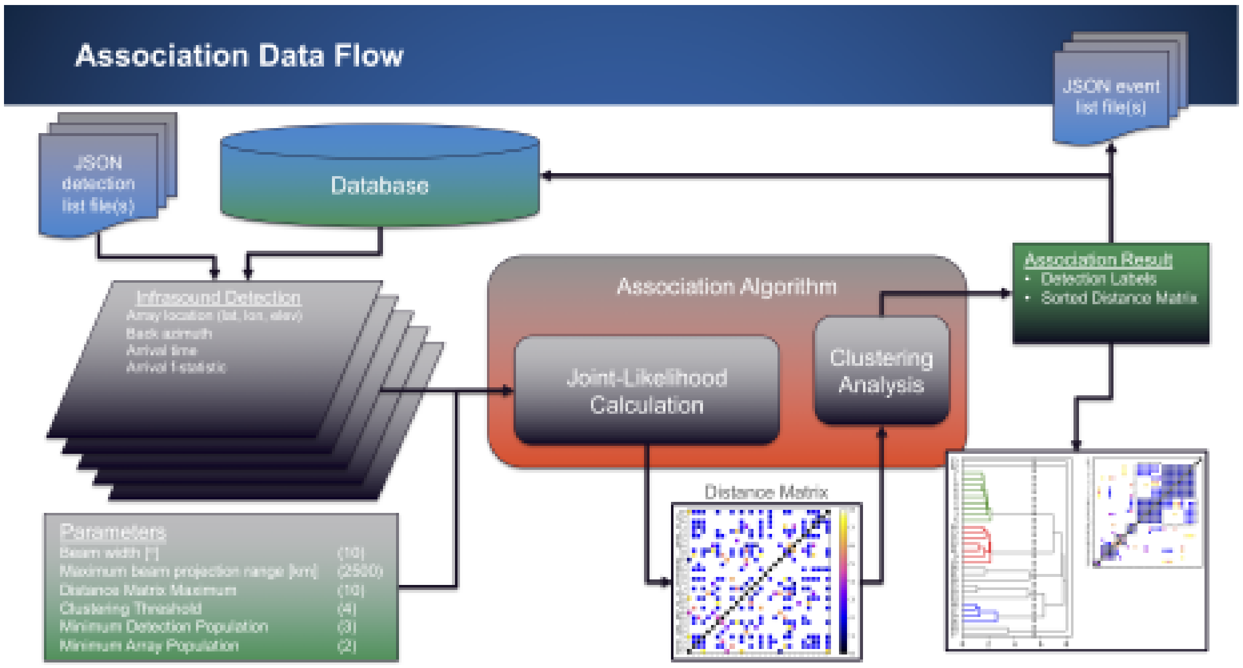
1.6 Data Processing Flow

Data can be processed using InfraPy in a variety of ways. See the images below for examples of data processing workflows.

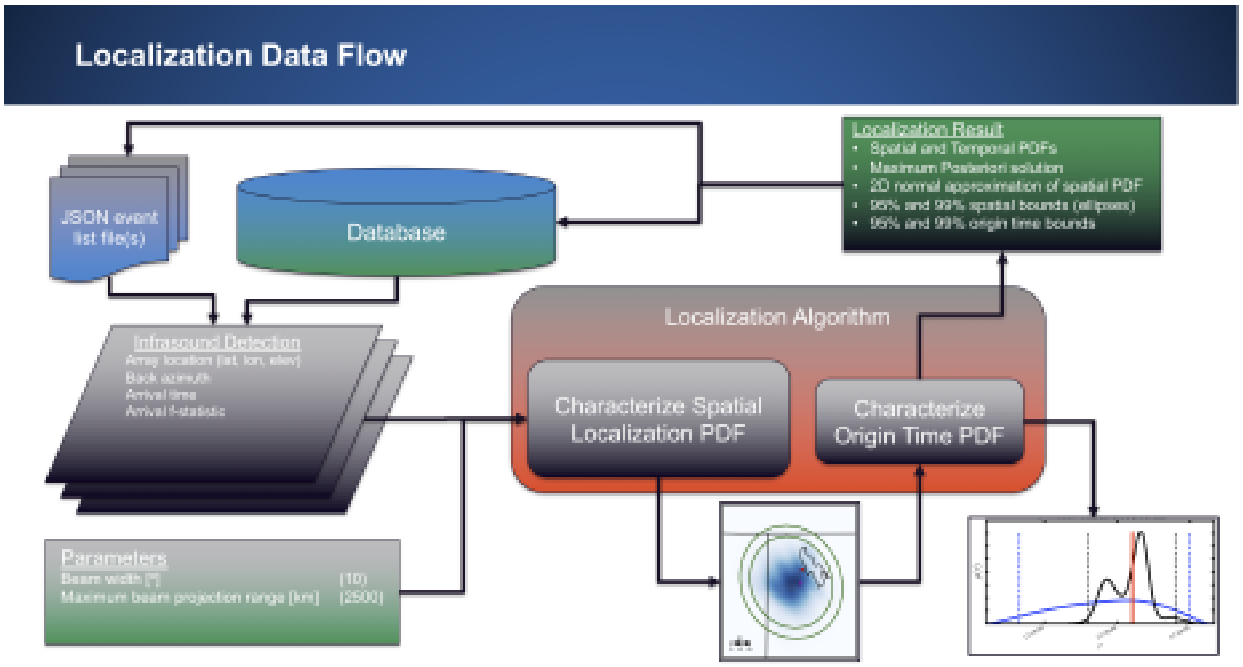
1.6.1 The Adaptive F-Detector



1.6.2 Association



1.6.3 Bayesian Infrasonic Source Localization


Localization Data Flow
JSON event list file(s)
Database
Localization Result
Spatial and Temporal PDFs
Maximum Posterior solution
2D normal approximation of spatial PDF
95% and 99% spatial bounds (ellipses)
95% and 99% origin time bounds
Infrasound Detection
Array location (lat, lon, elev)
Back azimuth
Arrival time
Arrival f-statistic
Localization Algorithm
Characterize Spatial Localization PDF
Characterize Origin Time PDF
Parameters
Beam width [°] (10)
Maximum beam projection range [km] (2500)

1.6.3.1 Stand-Alone Processing

Modules within Infrapy can be run 'stand-alone' as package imports following the scripts found in the /examples folder

1.6.3.2 Running Pipeline Processing in Infrapy

The folder tutorials/cli contains all necessary data and configuration files to begin utilizing the pipeline processing methodologies in Infrapy.

Once installed, the steps to run pipeline processing in Infrapy are:

1. Either load local waveform data into a sqlite database or connect to a Oracle database following instructions in *Interfacing with Pisces*.
2. Create a configuration file. See *Configuration Files* for a detailed description of the parameters that need to be included. Two example configuration files, one for connecting to the provided sqlite file and one for connecting to an oracle database are provided.
3. Run the FK analysis for a specific array:

```
>> infrapy run_fk --config_file BRPConfig.txt
```

4. Run the FD analysis for a specific array that has already FK results, remember to locate the parameter id from your FK analysis (you can use the script read_pfk.py to find the correct id).

```
  >> infrapy run_fd --config_file BRPConfig.txt

4. Once you have run detection on 2+ array, run association processing:

.. code-block:: python

    >> infrapy run_assoc --config_file BRPConfig.txt
```

1.6.3.2.1 Configuration Files

We run the different steps of the processing by running a script along with a configuration file. This applies to both station and network analysis levels. Configuration files set the parameters that are required to perform a specific analysis. The configuration file for the array processing (station level analysis) has the following structure:

An example configuration file is provided in tutorials/. Parameters for each field within the configuration file are outlined below.

```
# configuration file to run array (FK) processing and (FD) detection

[database] # required
url = sqlite:///example.sqlite
# database for processing
site = pisces.tables.css3:Site
wfdisc = pisces.tables.css3:Wfdisc
affiliation = pisces.tables.css3:Affiliation
# schemas for tables utilized in processing
# if processing in sqlite database, tables remain the same
# if processing in Oracle database, tables should point to your custom global_.py file

[GeneralParams]
year=2012
# year for processing
dayofyearini=100
# Julian Day to begin processing
dayofyearend=102
# Julian Day to stop processing
```

(continues on next page)

(continued from previous page)

```
station=BRP
# REFSTA of station for processing
channel=EDF
# channel
name=example
# name of processing parameters
cpucnt=30
# number of cpu cores to use for processing
domain=time
# domain (time or frequency) to run FK and FD processing


[FKParams]
name=mid band fk test
# name for fk processing
freqmin=0.5
# minimum frequency
freqmax=5.0
# maximum frequency
beamwinlen=60
# beam window length
beamwinstep=30
# beam window step
backazmin=-180.0
# minimum bz for processing
backazmax=180.0
# maximum bz for processing
backazstep=1.5
# bz step
trvelmin=300.0
# minimum trace velocity
trvelmax=600.0
# maximum trace velocity
trvelstep=2.5
# trace velocity step
beammethod=bartlett
# beam method
fkresults=fk_res_brp
# where fk processing results are stored
numsources = 1
func_fk = None

[FDetectParams]
back_az_lim=10
# limit of bz deviation between consecutive fk results
detwinlen=300.0
# window length for adaptive f detection
detthresh=0.99
# detection threshold
dsegmin=5
detmethod=fstat
tb_prod=4000
adaptivewlen=1200
#length of window for AFD
pthreshold=.01
#p-value for time domain detection
pfkid=0
```

(continues on next page)

(continued from previous page)

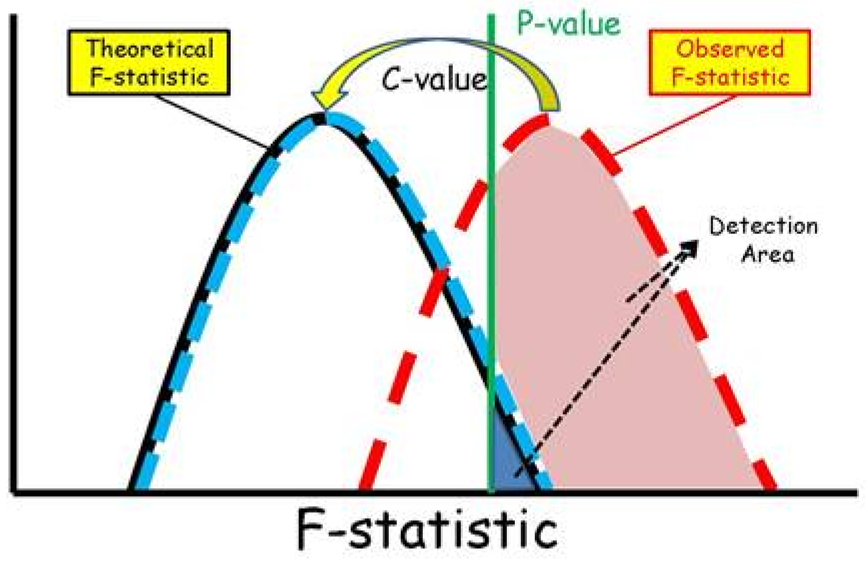
```
# pkfid for FK results
corrthreshold=0.5
# threshold for correlation values
mineventlength
# minimum event length in seconds
fkresults=fk_res_brp
# fk results to run detection on
fdresults=fd_res_example_brp
# where detection results are saved



[AssocLocParams]
network=YJ
# network for association
pfdetectid=0
# detection ID from detection processing (all arrays for assoc must have same detect␣
↪ID)
pfkid=2
# beamforming ID (all arrays must have same FK ID)
beamwidth=10.0
rangemax=1000.0
clusterthresh=4.0
trimthresh=None
eventdetmin=3
# minimum # of detections to form event
eventarrmin=2
# minimum number of arrays for event
duration = 60
# duration (minutes) for association windows

fdtable_1=fd_res_example_brp
#fdtable_2=fd_res_fsu
#fdtable_3=fd_res_wmu
# tables where detection results are stored
resultstable = test_assoc
# table where association results will be stored
```

1.6.3.2.2 Infrapy FK Processing

```
>> infrapy run_fk --config_file BRPConfig.txt
```

1.6.3.2.3 Infrapy Detection (FD) Processing

Infrapy detects signals using an Adaptive F-Detector (AFD). The adaptive F-detector was developed by Arrowsmith et al., (2009) to account for both correlated and uncorrelated noise through modification of the conventional F-statistic. The detector accounts for temporal changes in noise by applying an adaptive window to update the detection distribution, which allows for the distinction between signal and correlated noise.



Configuration file

Detection is run using the same configuration file as FK processing. Detection requires input from FK processing results

```
>> infrapy run_fd --config_file BRPConfig.txt
```

1.6.3.2.4 Infrapy Assoc Processing

```
>> infrapy run_assoc --config_file BRPConfig.txt
```

1.6.3.2.5 Scripts

Scripts to manipulate Infrapy FK results

Use “any_command”.py -h to get specific information to run the script

1. read_pfk.py to see the different set of configuration parameters used previously for FK analysis

-h –help show this help message and exit
-d SQ name of the database connection, e.g.: -d sqlite:///UT_tutorial.sqlite

```
>> read_pfk.py [-h] -d SQ
```

2. print_rfk.py to see fk results for a specific array and FK parameter ID

-h –help show this help message and exit

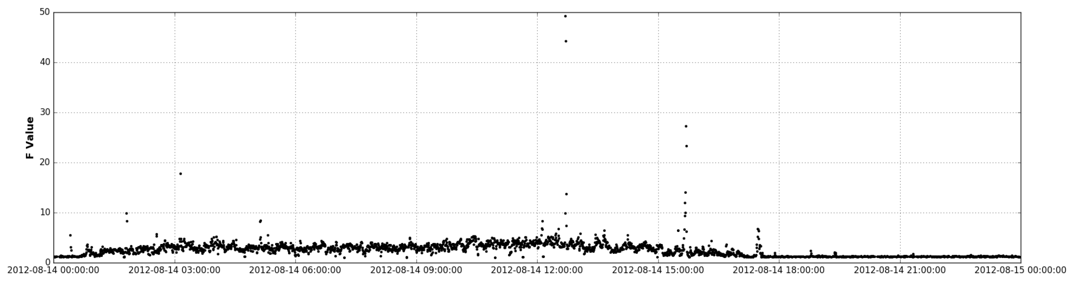
-d SQ name of the database connection, e.g.: -d sqlite:///UT_tutorial.sqlite
-a ARRAY array name, e.g.: -HWU4
-t FKRESULTS specific table with results, e.g.: -t fk_HWU
-s TS starttime plot, e.g.: -s /'2014-03-02T00:00:00/'
-e TE endtime plot, e.g.: -s /'2014-03-03T00:00:00/'
-F FVAL limit Fvalue, e.g.: -F 0
-o OUTTEXT print fk data, e.g.: -o res_FILE

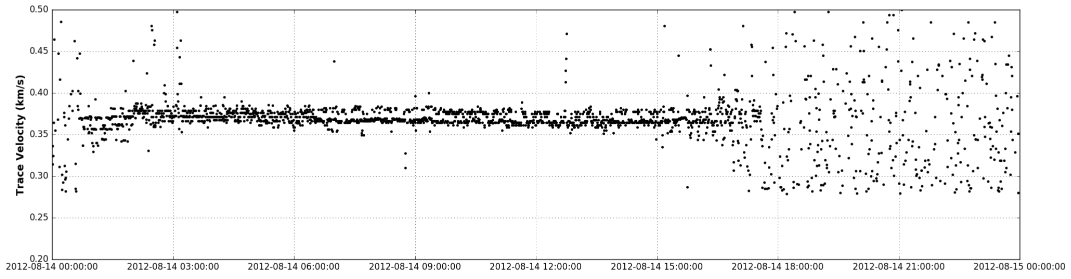
```
>> print_rfk.py [-h] -d SQ -a ARRAY -f PFK_ID -t FKRESULTS [-s TS] [-s TE] [-F FVAL] -
↪o res_FILE
```

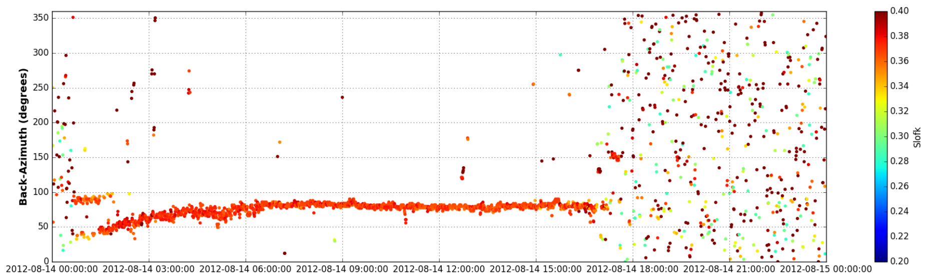
3. plot1_rfk.py to plot FK results

-h –help show this help message and exit
-d SQ name of the database connection, e.g.: -d sqlite:///UT_tutorial.sqlite
-a ARRAY array name, e.g.: -a HWU4
-f PFK_ID FK parameter ID to be plot, e.g.: -f 3
-t FKRESULTS specific table with results, e.g.: -t fk_res_HWU
-w WAVEPLOT plot waveforms, e.g.: -w 0
-s TS starttime plot, e.g.: -s /'2014-03-02T00:00:00/'
-e TE endtime plot, e.g.: -s /'2014-03-03T00:00:00/'
-F FVAL limit Fvalue, e.g.: -F 0
-slo SLOFK limit slofk, e.g.: -slo 0
-bzmin BZMIN limit min bz, e.g.: -bzmin 0
-bzmax BZMAX limit max bz, e.g.: -bzmax 360

```
>>plot1_rfk.py [-h] -d SQ -a ARRAY -f PFK_ID [-t FKRESULTS] [-w WAVEPLOT] [-s TS] [-e␣
↪TE] [-F FVAL] [-slo SLOFK] [-bzmin BZMIN] [-bzmax BZMAX]
```







Scripts to manipulate Infrapy FD results

1. read_pfd.py to see the different set of configuration parameters used previously for detection analysis

-h, –help show this help message and exit
-d SQ name of the database connection, e.g.: -d sqlite:///mydb.sqlite

```
>> read_pfd.py [-h] -d SQ
```

2. read_rfd.py to see the available detection results

-h, –help show this help message and exit
-d SQ name of the database connection, e.g.: -db sqlite:///UT_tutorial.sqlite
-a ARRAY array name, e.g.: -a HWU4
-f PFK_ID, –pfkid PFK_ID FK parameter ID to be plot, e.g.: -f 0
-j PFDID, –pfdid PFDID fd parameter id, e.g.: -j 0
-t FDRESULTS specific table with results, e.g.: -t fd_res_HWU

```
>> read_rfd.py [-h] -d SQ -a ARRAY -f PFK_ID -j PFDID [-t FDRESULTS]
fdid: 1  pfdid: 0 pfkid: 0   timeini: 12-08-14 00:24:30   timeend: 12-08-14 00:26:00 ␣
↪ maxf0: 5.54282460217 0.60800443287
fdid: 2  pfdid: 0 pfkid: 0   timeini: 12-08-14 00:47:00   timeend: 12-08-14 00:53:30 ␣
↪ maxf0: 3.67253815208 0.46716764663
fdid: 3  pfdid: 0 pfkid: 0   timeini: 12-08-14 00:54:30   timeend: 12-08-14 00:56:30 ␣
↪ maxf0: 2.61098286001 0.372113714177
fdid: 4  pfdid: 0 pfkid: 0   timeini: 12-08-14 01:47:30   timeend: 12-08-14 01:49:00 ␣
↪ maxf0: 9.91099311406 0.749628285504
```

3. read_rfd_fast.py to write the available detection results in text file

-h, –help show this help message and exit
-d SQ name of the database connection, e.g.: -d sqlite:///UT_tutorial.sqlite
-a ARRAY array name, e.g.: -a I37NO
-f PFKID, –pfkid PFKID fk parameter id, e.g.: -f 0
-j PFDID, –pfdid PFDID fd parameter id, e.g.: -j 0
-t FDRESULTS specific table with results, e.g.: -t fd_I37
-o OUTTEXT fd parameter id, e.g.: -o res_FILE

```
>> read_rfd_fast.py [-h] -d SQ -a ARRAY -f PFKID -j PFDID [-t FDRESULTS] [-o OUTTEXT]
```

1.7 Tutorial

A series of jupyter notebooks illustrating how to use infrapy subroutines are found in the /tutorial folder. You can run these by navigating to that folder and running:

```
>> jupyter notebook
```

A browser will launch with a webpage showing a directory listing, click on the InfraPyTutorial.ipynb link to open the main window, which will provide links to the various tutorials.

1.8 Schema

The purpose of this document is to define the schema used for the operation of the infrasound analysis tool, infrapy. The tables described by this document extend the CSS3.0 or KB core schema to include information required for the operation of infrapy. This document is divided into three sections, the first being this introduction. Section two defines eight new, infrasonic data processing-specific database tables. Both internal (ORACLE) and external formats for the attributes are defined, along with a short description of each attribute. Section three of the document shows the relationships between the different tables by using entity-relationship diagrams.

This schema is a work in progress and may be updated as development of infrapy continues.

1.8.1 Table Descriptions

This section describes the logical structure of each table used in the Infrapy software package. The name of the table is first, followed by a description of the purpose and use of the table. Below the description is a listing of the columns, in the order which they are defined in the tables. The storage column gives the actual ORACLE datatype for the column in question. The external format and character positions columns are provided for the convenience of database users who wish to transfer data between the ORACLE database tables and flat files.

1.8.1.1 Conventions

The following conventions are used, following Carr et al (2002):

Element	Appearance	Example
Database table	Bold	**arrival**
Database columns	Italic	*sta*
Database table and column when written in the dot notation	Bold.italic	**arrival**.*sta*
Value of a key or component of a key	Courier font	arid

1.8.1.2 Table Definitions: Infrapy-Specific Tables

Table descriptions can be found in the API section of the documentation.

1.9 API

1.9.1 Association

1.9.1.1 HJL

1.9.2 Characterization

1.9.2.1 SPYE

1.9.3 Database Processing

1.9.3.1 Database Processing Taskbase

1.9.3.2 Database Schema

1.9.4 Beamforming/Detection

1.9.4.1 Beamforming

1.9.4.2 Beamforming_new

1.9.5 Location

1.9.5.1 BISL

1.9.6 Propagation

1.9.6.1 Infrasound

1.9.6.2 Likelihoods

1.9.6.3 Seismic

1.9.7 Utils

1.9.7.1 infrapy.utils.cart2pol module

1.9.7.2 infrapy.utils.confidence module

1.9.7.3 infrapy.utils.db2db module

1.9.7.4 infrapy.utils.db2sac module

1.9.7.5 infrapy.utils.files2db module

1.9.7.6 infrapy.utils.get_arraywaveforms module

1.9.7.7 infrapy.utils.get_header_table module

1.9.7.8 infrapy.utils.get_mean_locations module

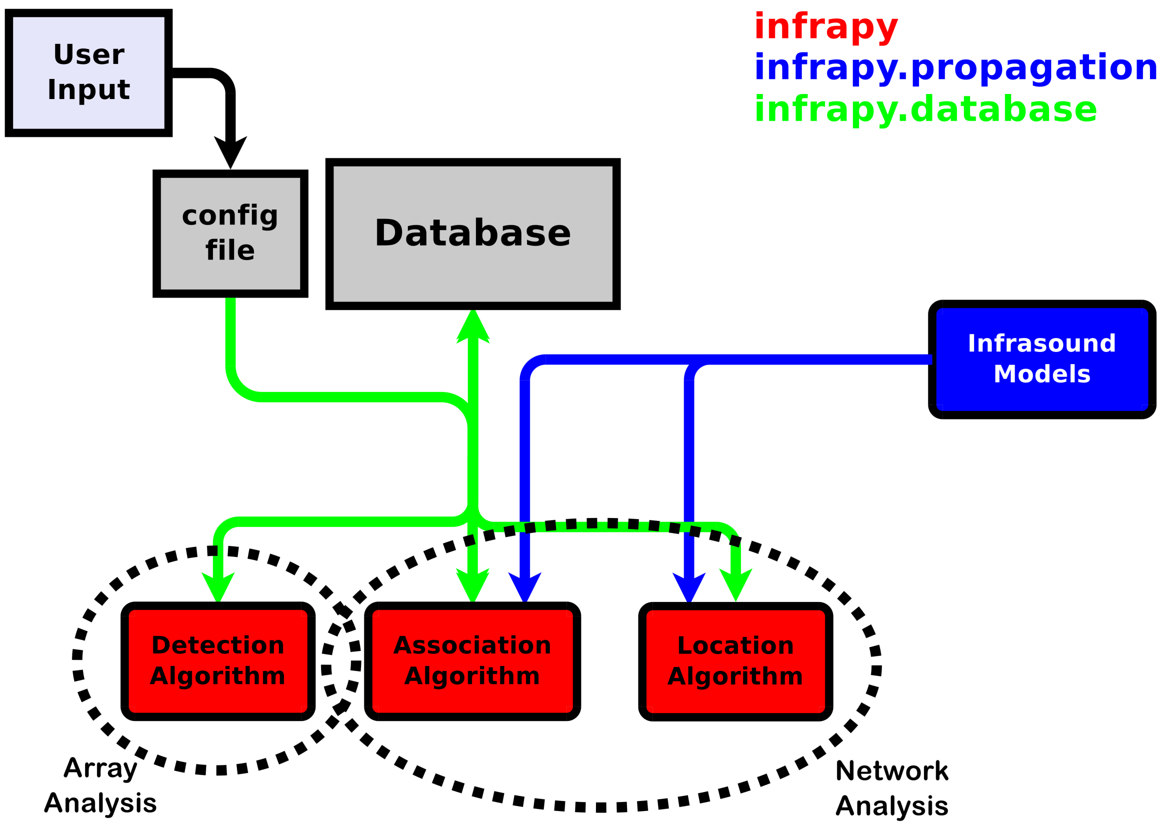
1.9.7.9 infrapy.utils.latlon module

1.9.7.10 infrapy.utils.ms2db module

[illegible]

docs describe each component in detail, with a full reference in the *API* section.

This document is a work in progress and may be updated as development of Infrapy continues.



CHAPTER

TWO

USER'S GUIDE

This part of the documentation, which is mostly prose, begins with some background information about Infrapy, then focuses on step-by-step instructions for data processing using Infrapy.

2.1 Overview

Infrapy is a tool for processing infrasound and seismic array data. Infrapy implements a database-centric approach for pipeline continuous near real-time analysis. The pipeline includes analysis at station and network levels (using beam-forming and clustering techniques, respectively) for the detection, association and location of events. The pipeline relies on the interaction of the algorithms with a relational database structure to organize and store waveform data, the parameters for the analysis, and results of both levels of analysis. Our implementation can interact seamlessly with traditional (e.g.: Oracle) and serverless (e.g.: SQLite) relational databases.