Documentation of Double Beamforming Package

Eileen R. Martin

June 6, 2020

This is the documentation of the doublebeamforming package, which can be downloaded from `https://github.com/eileenrmartin/doubleBeamforming` for ambient noise interferometry double beamforming transforms, as described in the paper "A Linear Algorithm for Ambient Seismic Noise Double Beamforming Without Explicit Crosscorrelations," currently under review in Geophysics (will update this statement after review process). Questions about the code can be directed to `eileenrmartin@vt.edu`.

This package can be installed using the `installDBF` rule in the `Makefile`, and all examples and scalability tests in the paper can be reproduced using rules in the `Makefile`. The package contents are all in the `doublebeamforming` folder:

- `__init__.py` to include all functions and classes in the package.
- `arrays.py` with the arrayPatch class
- `distFromAvg.py` with the calcDistFromAvg function used to create a local coordinate system
- `newDBFFuncs.py` with the new proposed algorithm, broken up into the phase1 function, the phase2 function, and the helper function called shiftFrqData
- `traditionalXcorrsDBF.py` with the traditional algorithm for performing cross-correlations in the xCorrsAcrossArrays function, and the traditional algorithm for performing a double beamforming tranform on cross-correlations in the DBFAfterXcorrs function

More details of the arrayPatch class and functions are described below.

1 The arrayPatch class

An arrayPatch is a class that keeps track of a group of stations near each other, which have a set of slownesses and angles of interest for beamforming, a common time sampling rate, and a list of files where data and sensor-specific metadata are stored (a single file per sensor), as well as one file that specifies the coordinates. The data associated with this class are:

- stations, a list of station names (list of strings)
- n, the number of sensors in the array patch (integer)
- u, the slownesses of interest (numpy 1D array)
- Nu, the number of slownesses of interest (integer)
- Th, the angles of interest (numpy 1D array)
- NTh, the number of angles of interest (integer)
- dtValue, the time in seconds between samples (float)
- filenames, a list of file names with input data in mseed format, one file per station
- coordinatesFile, a string representing a file name that contains the coordinates of each channel (one row per channel: station-name latitude longitude)

2 The xCorrsAcrossArrays function

This is one of the old, slow algorithms which should be used only for comparison purposes. After preprocessing of ambient noise is done, this function can be used to calculate cross-correlations between all sensors in arrayPatchA with all sensors in arrayPatchB at time lags between -maxTau and +maxTau seconds (assuming the files all cover the same time window). Note: If you have many separate time windows split up by separate files, create a new arrayPatch for each time window and call this for each time window (averaging as you go along).

HOW TO CALL:
`xcorrs = xCorrsAcrossArrays(arrayPatchA, arrayPatchB, maxTau)`

INPUTS:

- arrayPatchA, the first arrayPatch object (note that if angles/slownesses aren't correct, it won't affect the cross-correlations)
- arrayPatchB, the second arrayPatch (should have same dtValue as A, and note that if angles/slownesses aren't correct, it won't affect the cross-correlations)
- maxTau, a positive float representing max time lag of cross-correlations in seconds

OUTPUTS:

- xcorrs, a numpy array with (number stations A) x (number stations B) x (number of time lags including negative, 0 and positive)

DEPENDENCIES:

- arrayPatch class in this package
- numpy
- obspy

3 The DBFAfterXcorrs function

This is one of the old, slow algorithms which should be used only for comparison purposes. After a set of crosscorrelations has been calculated between all sensors in two array patches (i.e. one sensor in one patch and the other sensor in the other patch), then this function can calculate the double beamforming transform between the two array patches.

HOW TO CALL:
```
B = DBFAfterXcorrs(arrayPatchA, arrayPatchB, xcorrs, Nt)
```

INPUTS:

- arrayPatchA, the first arrayPatch
- arrayPatchB, the second arrayPatch (should have same dtValue as A)
- maxTau, a float representing max time lag of cross-correlations in seconds
- xcorrs, a numpy array with (num. stations A) x (num. stations B) x (num. of time lags including negative, 0 and positive) representing all cross-correlations
- Nt, an integer representing number of starting time lags to consider

OUTPUTS:

- B, a numpy array with (num. slowness A) x (num. angles A) x (num. slowness B) x (num. angles B) x (num. start times) representing the double beamforming transform

DEPENDENCIES:

- numpy
- arrayPatch class in this package
- calcDistFromAvg function in this package

4 The calcDistFromAvg function

This function converts an approximate coordinate system from latitude longitude to the average distance in meters (both directions) from the center of the array of sensors collecting the data in listOfFilenames.

HOW TO CALL:
If the MSEED files don't contain latitude and longitude in stats.coordinates, specify in another text file: `distFromAvg = calcDistFromAvg(listOfFilenames, fileWithCoords)` or if the MSEED files each contain stats.coordinates info with latitude and longitude: `distFromAvg = calcDistFromAvg(listOfFilenames)`

INPUTS:

- listOfFilenames, a list of MSEED filename strings, each file containing data for one seismic sensor in a patch
- (optional) fileWithCoords, a string representing the path to a text file (0th column name, first column latitude, 2nd column longitude). This is only needed if stats.coordinates is missing in the files specified by listOfFilenames.

OUTPUTS:

- distFromAvg, 2D numpy array storing floats representing the distance in meters of each sensor from centroid of array of sensors. Column 0 contains the lattitude difference converted to meters, and column 1 contains the longitude difference converted to meters.

DEPENDENCIES:

- obspy
- numpy
- math

5 The shiftFrqData function

This function calculates the R factor for an array due to a particular sensor. The R factor for all sensors is then added up outside this function to create the overall single beamforming result for this array patch.

HOW TO CALL:
`RUpdate = shiftFrqData(thisLoc,slownesses,frqs,dataF,Nu,NTh,nFrq)`

INPUTS:

- thisLoc, a 1D numpy array with 2 entries (x and y coordinates in meters) describing location of the sensor
- slownesses, a 3D numpy array with slownesses of interests (number of slownesses, number of angles, 2) where slownesses[iu,ith,0] = u*cos(theta) and slownesses[iu,ith,1] = u*sin(theta)
- frqs, a 1D numpy array of the frequencies associated with each frequency bin in dataF
- dataF, a 1D numpy array (complex) of the FFT of the data at the particular sensor
- Nu, integer representing number of slownesses (redundant but allows jit to work for performance improvement)
- NTh, integer representing number of angles
- nFrq, integer representing number of entries in frqs

OUTPUTS:

- RUpdate, a 3D numpy array (number of slownesses, number of angles, number of frequencies) representing the contribution of this sensor to the whole array's R factor.

DEPENDENCIES:

- numpy
- numba jit compiler

6 The phase1 function

This calculates the R factor for one array patch for a given time window (equation 4/5 in the paper).

HOW TO CALL:

```
R = phase1(stations,frqs,nTrunc,locations,slownesses,
    funcToGenerateFilenames,startWindow)
```

INPUTS:

- stations, a list of station names (often this is a 4 letter/digit code)
- frqs, a 1D numpy array of the frequencies associated with each frequency bin in dataF
- nTrunc, number of entries to use for FFT (nearest power of 2 below number of time samples)

- locations, a 2D numpy array (number of sensors in array, 2) where each row contains the x,y location (in meters) of a sensor
- slownesses, a 3D numpy array with slownesses of interests (number of slownesses, number of angles, 2) where slownesses[iu,ith,0] = u*cos(theta) and slownesses[iu,ith,1] = u*sin(theta)
- funcToGenerateFilenames, a function that takes two arguments (station name, a starting time described by obspy UTCTime object) to generate filename for each sensor (to be read w/ obspy.read())
- startWindow, obspy UTCTime representing the starting time of this window of data (for generating file name)

OUTPUTS:

- R, a 3D numpy array (number of slownesses, number of angles, number of frequencies) representing the whole array's R factor for this time window

DEPENDENCIES:

- obspy
- numpy

7 The phase2 function

This function calculates the double beamforming transform for one time window given the R factors from phase 1 of the A and B arrays.

HOW TO CALL:
```
BTemp = phase2(RA,RB,Nt)
```

INPUTS:

- RA, 3D numpy array (NuA,NThA,number of frequencies), R factor for A array in this time window
- RB, 3D numpy array (NuB,NThB,number of frequencies), R factor for B array in this time window
- Nt, integer number of time lags of interest in the double beamforming transform

OUTPUTS:

- BTemp, a 5D numpy array (NuA,NThA,NuB,NThB,Nt) that describes the double beamforming transform for this time window

DEPENDENCIES:

- numpy
- scipy