dtwhaclustering

Release 1.0

Utpal Kumar

Aug 22, 2021

USAGE

CHAPTER

ONE

DYNAMIC TIME WARPING BASED HIERARCHICAL AGGLOMERATIVE CLUSTERING

Codes to perform Dynamic Time Warping Based Hierarchical Agglomerative Clustering of GPS data

author Utpal Kumar

date 2021/08

CHAPTER

TWO

INSTALLATION

2.1 Requirements

1. `dtaidistance`: For computing DTW distance
2. `pygmt`: For plotting high-resolution maps
3. `pandas`: Analyze tabular data
4. `numpy`: Computation
5. `matplotlib`: Plotting time series
6. `scipy`: Interpolating data
7. `xarray`: Multilayered data structure

2.2 Install from PyPI

This package is available on PyPI (requires Python 3):

```
pip install dtwhaclustering
```

CHAPTER

THREE

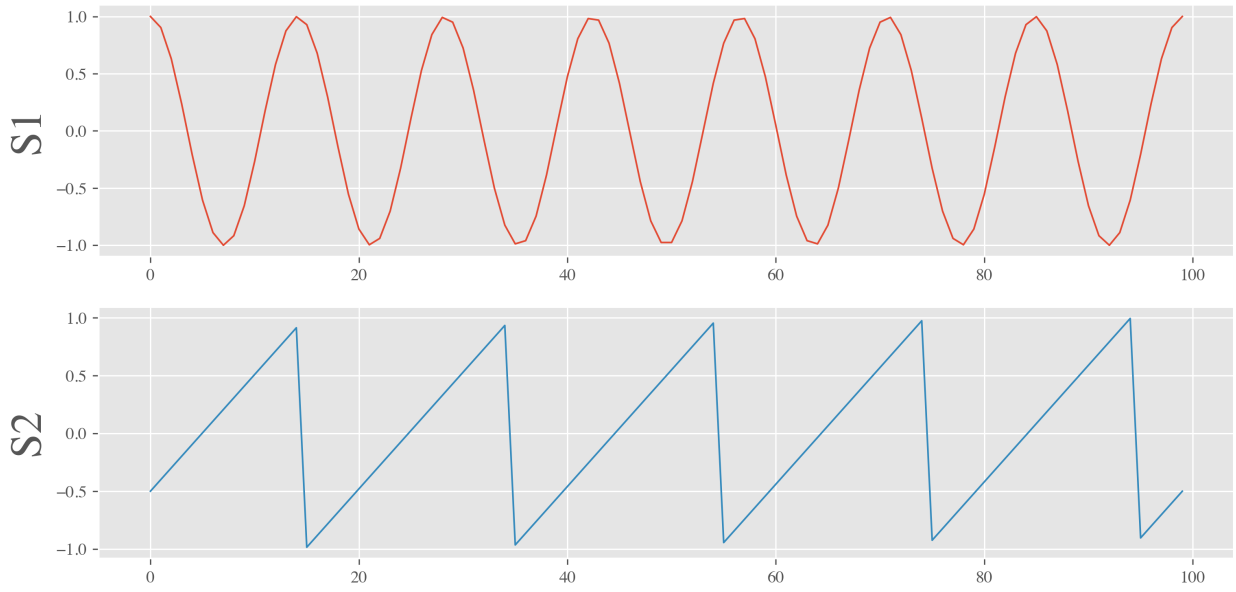
SIGNAL ANALYSIS

3.1 Create signals for analysis

```
np.random.seed(0)
# sampling parameters
fs = 100   # sampling rate, in Hz
T = 1      # duration, in seconds
N = T * fs  # duration, in samples

# time variable
t = np.linspace(0, T, N)

SNR = 0.2 #noise

XX0 = np.sin(2 * np.pi * t * 7+np.pi/2) #+ np.random.randn(1, N) * SNR
XX1 = signal.sawtooth(2 * np.pi * t * 5+np.pi/2) #+ np.random.randn(1, N) * SNR
s1, s2 = XX0, XX1
```

3.2 Inspect the DTW distance between two signals

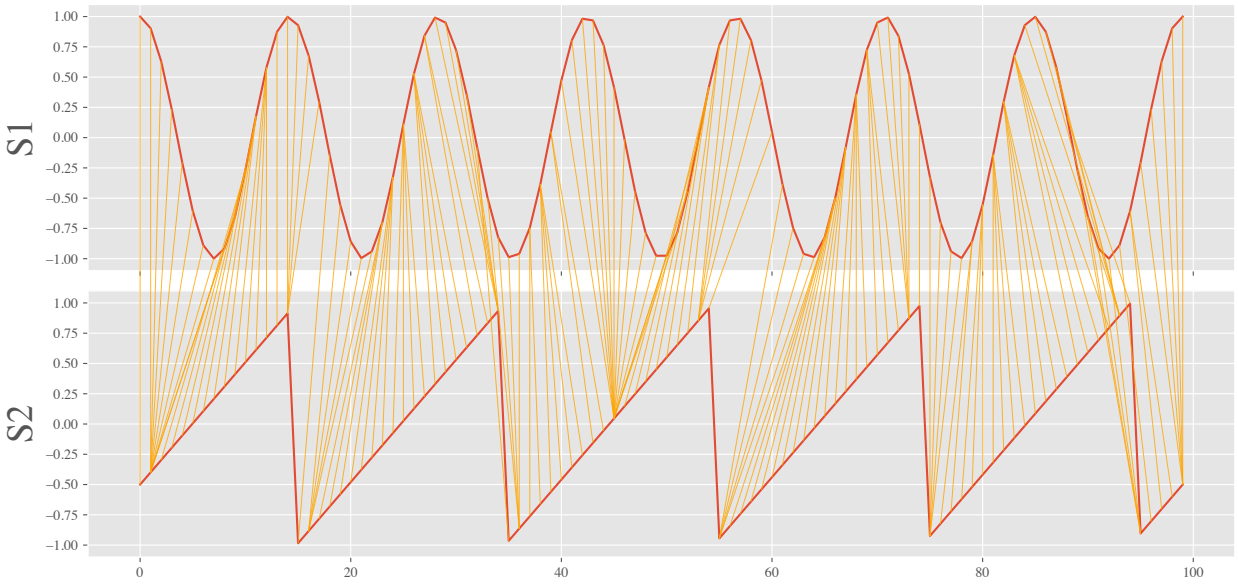
```
dtwsig = dtw_signal_pairs(s1, s2, labels=['S1', 'S2'])
dtwsig.plot_signals()
plt.show()
```



3.3 Plot warping path

```
matplotlib.rcParams['pdf.fonttype'] = 42
distance,_,_ = dtwsig.plot_warping_path()
print(f"DTW distance between signals: {distance:.4f}")

plt.savefig("warping_path_s1_s2.pdf", bbox_inches='tight')
```

DTW distance between signals: 5.2093

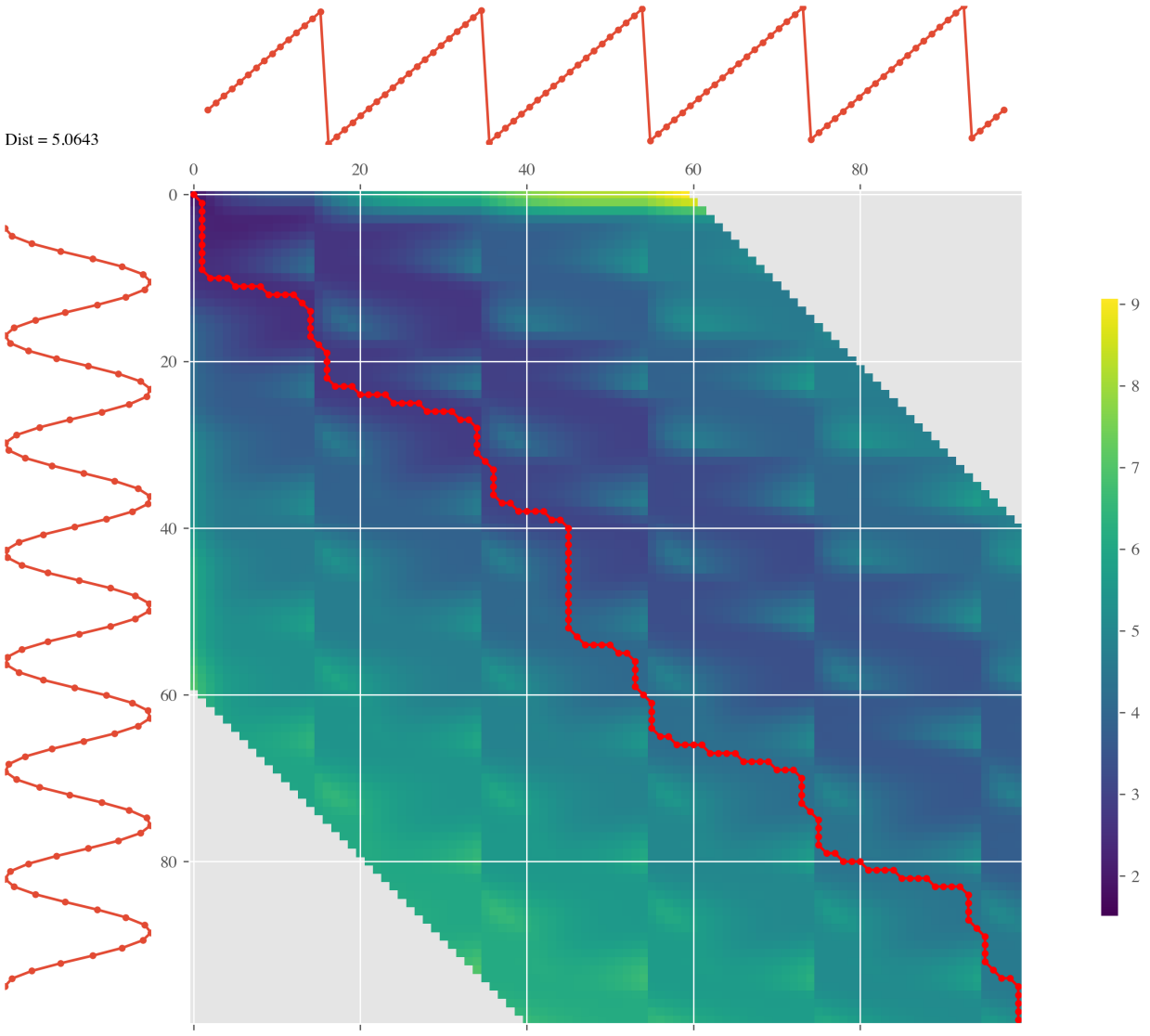



```
dtwsig.plot_matrix(windowfrac=0.6, psi=None) #Only allow for shifts up to 60% of the␣
↪minimum signal length away from the two diagonals.
```

(continues on next page)

(continued from previous page)

```
plt.show()
```



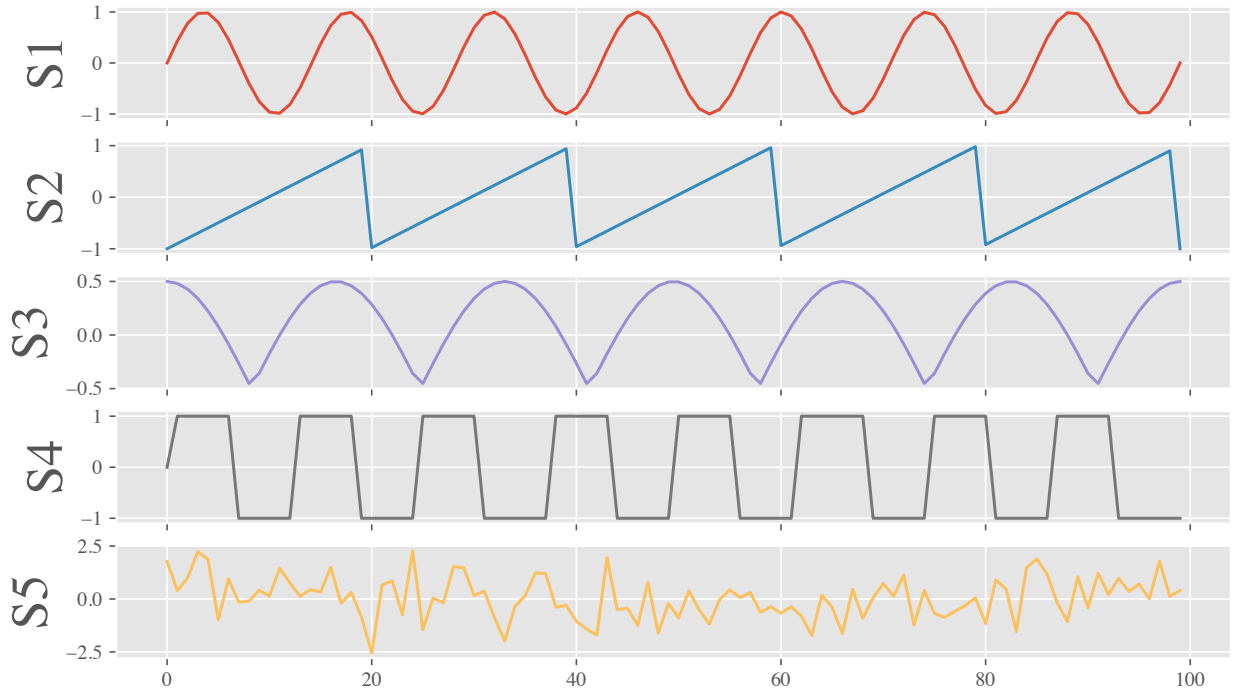
3.4 Create multiple signals

```
fs = 100   # sampling rate, in Hz
T  = 1     # duration, in seconds

N  = T * fs # duration, in samples
M  = 5      # number of sources
S1 = np.sin(2 * np.pi * t * 7)
S2 = signal.sawtooth(2 * np.pi * t * 5)
S3 = np.abs(np.cos(2 * np.pi * t * 3)) - 0.5
S4 = np.sign(np.sin(2 * np.pi * t * 8))
S5 = np.random.randn(N)
```

(continues on next page)

(continued from previous page)

```
time_series = np.array([S1, S2, S3, S4, S5])

## instantiate the class
dtw_cluster = dtw_clustering(time_series,labels=['S1','S2','S3','S4','S5'])

matplotlib.rcParams['pdf.fonttype'] = 42
dtw_cluster.plot_signals()
# plt.show()
plt.savefig("base_functions.pdf", bbox_inches='tight')
```



3.5 Compute the relative DTW distance between the signals

```
ds = dtw_cluster.compute_distance_matrix(compact=False)
```

```
array([[0.        , 5.15998322, 4.19080907, 5.77875263, 7.95685039],
[5.15998322, 0.        , 4.74413601, 7.71110741, 9.31343712],
[4.19080907, 4.74413601, 0.        , 8.75201301, 8.51048008],
[5.77875263, 7.71110741, 8.75201301, 0.        , 9.18406086],
[7.95685039, 9.31343712, 8.51048008, 9.18406086, 0.        ]])
```

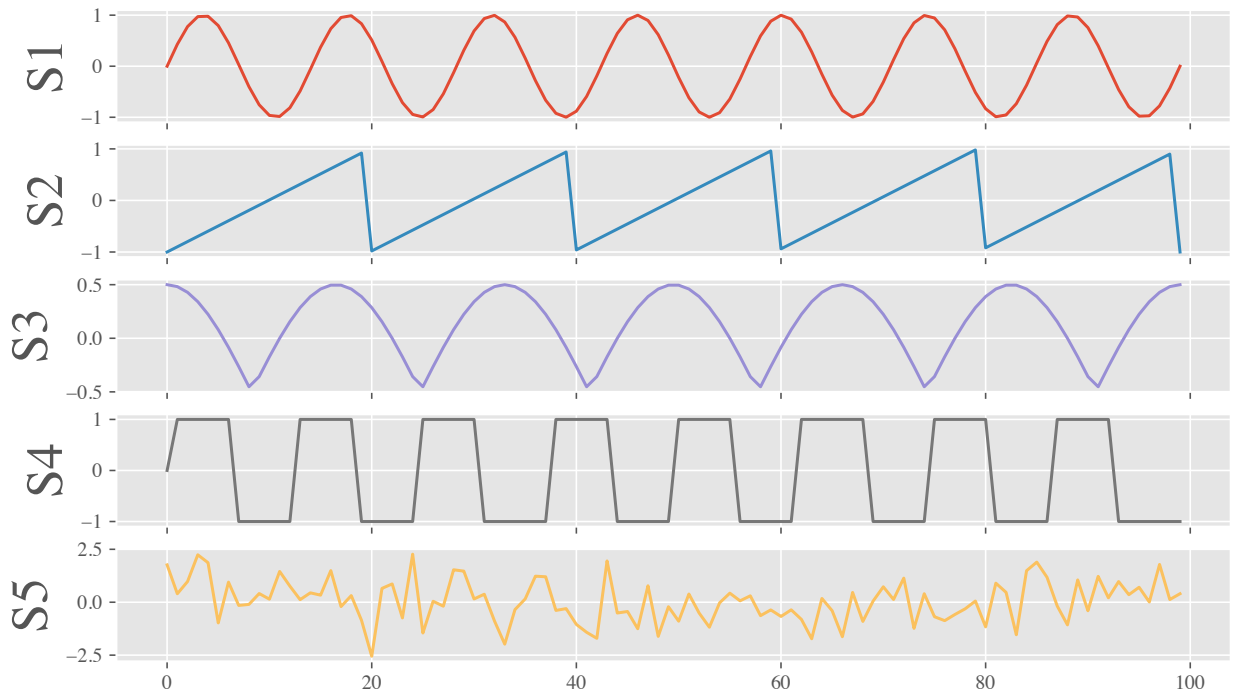
CHAPTER

FOUR

CLUSTERING ANALYSIS

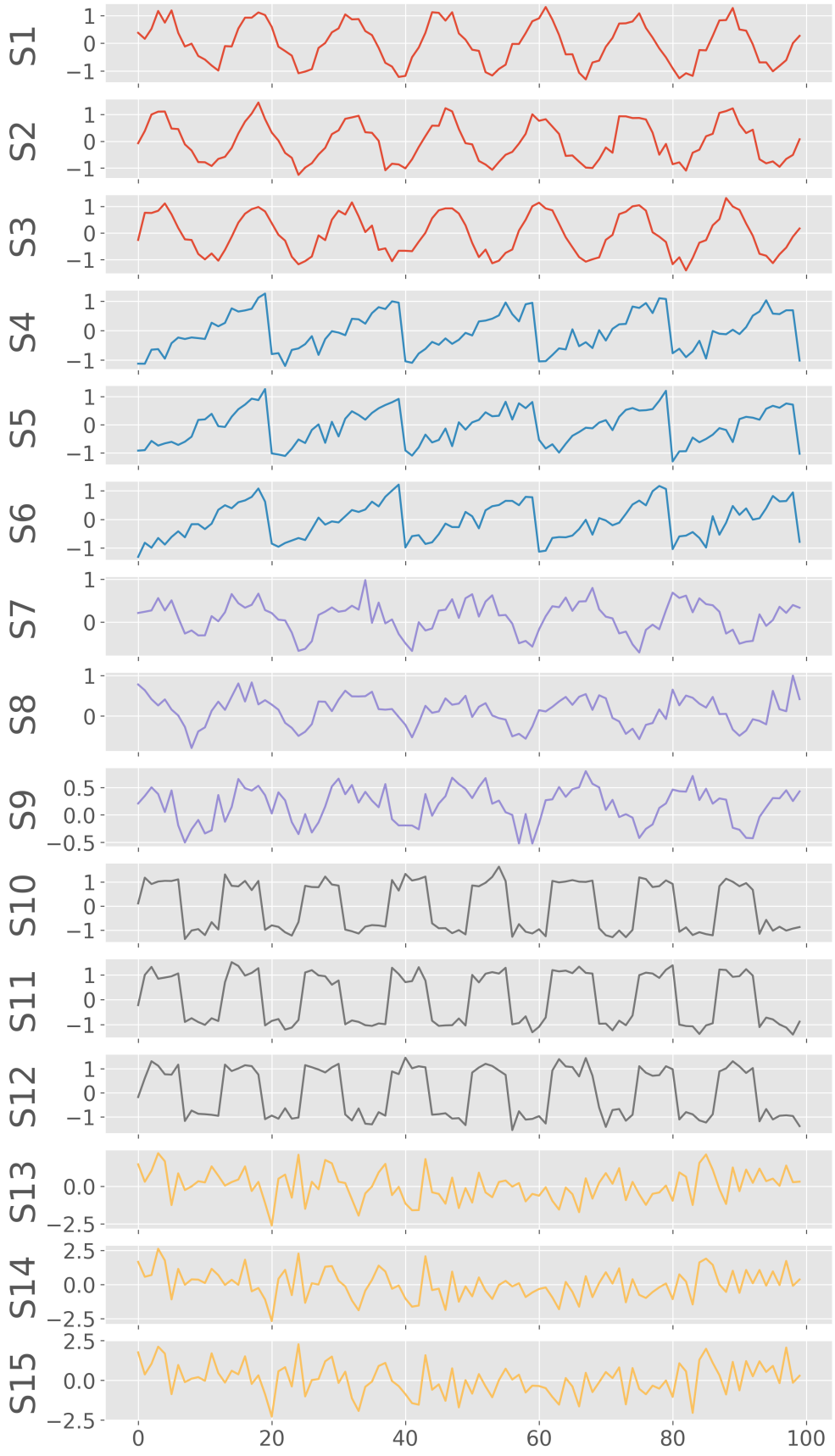
4.1 Create signals for analysis

```
from dtwhaclustering.dtw_analysis import dtw_signal_pairs, dtw_clustering, plot_signals,␣
↪shuffle_signals, plot_cluster
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
from dtaidistance import dtw
from scipy.cluster.hierarchy import fcluster

from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering

np.random.seed(0)
# sampling parameters
fs = 100   # sampling rate, in Hz
T  = 1      # duration, in seconds
N  = T * fs # duration, in samples
M  = 5      # number of sources
R  = 3      # number of copies
MR = M * R


# time variable
t = np.linspace(0, T, N)

S1 = np.sin(2 * np.pi * t * 7)
S2 = signal.sawtooth(2 * np.pi * t * 5)
S3 = np.abs(np.cos(2 * np.pi * t * 3)) - 0.5
S4 = np.sign(np.sin(2 * np.pi * t * 8))
S5 = np.random.randn(N)

time_series = np.array([S1, S2, S3, S4, S5])

fig, ax = plot_signals(time_series)
plt.show()
```



4.2 Add noise and make 3 copies of each signal

```
SNR = 0.2
X0 = np.tile(S1, (R, 1)) + np.random.randn(R, N) * SNR
X1 = np.tile(S2, (R, 1)) + np.random.randn(R, N) * SNR
X2 = np.tile(S3, (R, 1)) + np.random.randn(R, N) * SNR
X3 = np.tile(S4, (R, 1)) + np.random.randn(R, N) * SNR
X4 = np.tile(S5, (R, 1)) + np.random.randn(R, N) * SNR
X = np.concatenate((X0, X1, X2, X3, X4))

color = ['C0']*3+['C1']*3+['C2']*3+['C3']*3+['C4']*3
fig, ax = plot_signals(X,figsize=(10,20), color=color)
plt.show()
```

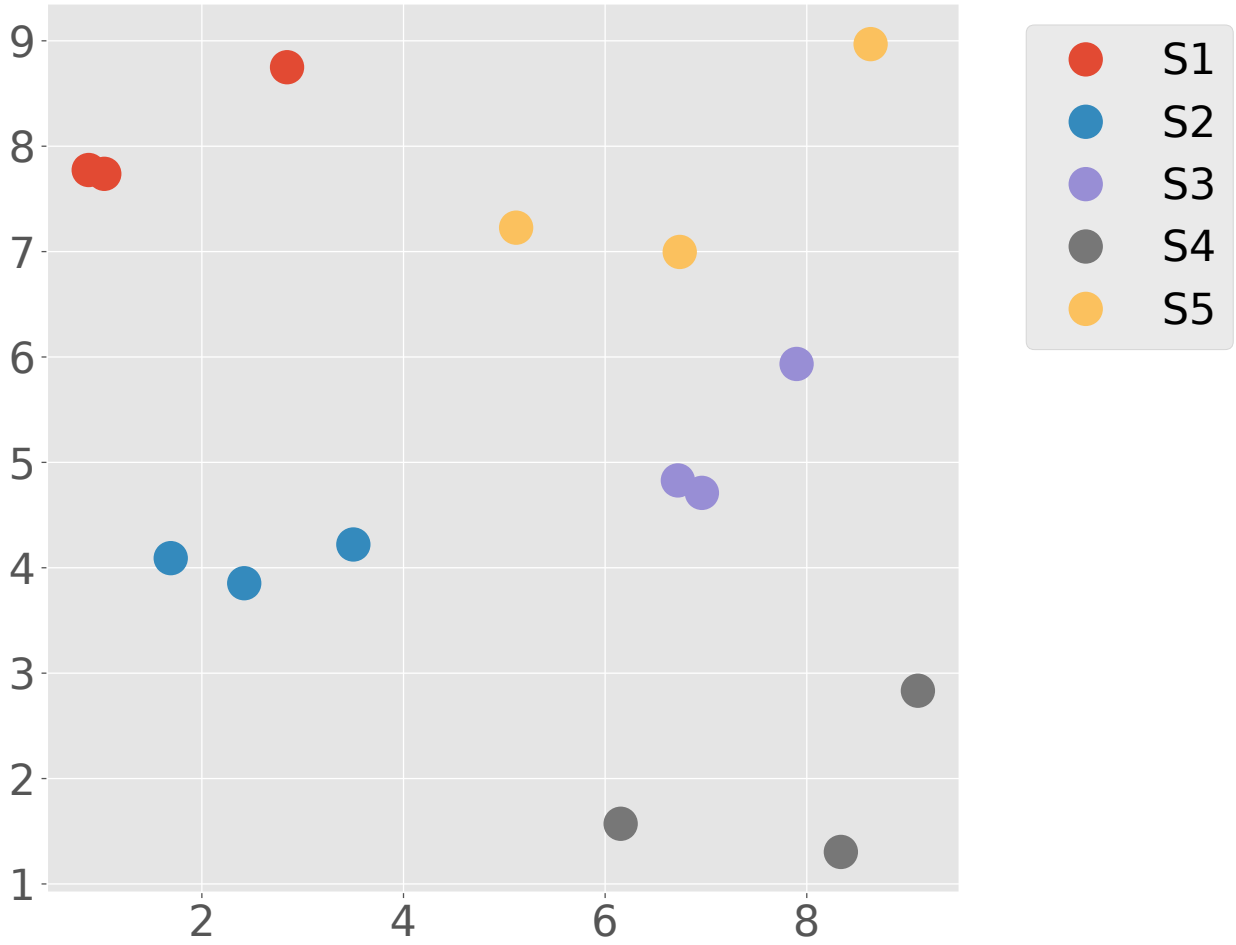

S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
0
20
40
60
80
100

4.3 Geographically distribute the signals

Now, we have 15 signals in total. Let us also randomly make these signals distributed in geographical space by assigning them longitudes and latitudes. We assume that the signals with similar waveforms are geographically co-located.

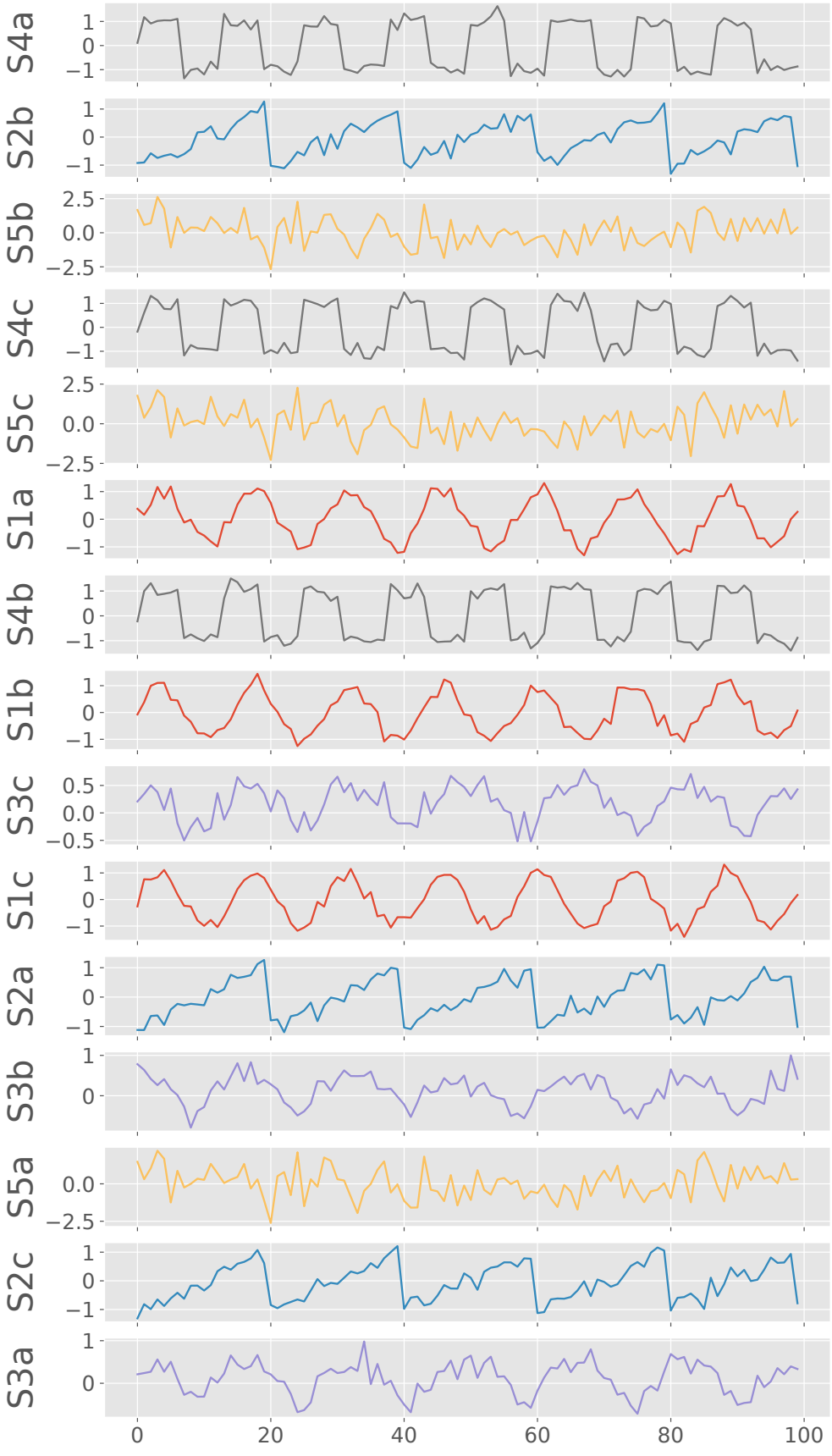
```
S0 variants (S0, S1, S2) -> xrange(0-3) yrange(7-10)
S1 variants (S3, S4, S5) -> xrange(1-4) yrange(3-5)
S2 variants (S6, S7, S8) -> xrange(4-8) yrange(4-6)
S3 variants (S9, S10, S11) -> xrange(5-10) yrange(0-4)
S4 variants (S12, S13, S14) -> xrange(5-9) yrange(6-9)
```

```
S0_lons = np.random.uniform(0, 3, 3)
S0_lats = np.random.uniform(7, 10, 3)

S1_lons = np.random.uniform(1, 4, 3)
S1_lats = np.random.uniform(3, 5, 3)

S2_lons = np.random.uniform(4, 8, 3)
S2_lats = np.random.uniform(4, 6, 3)

S3_lons = np.random.uniform(5, 10, 3)
S3_lats = np.random.uniform(0, 4, 3)

S3_lons = np.random.uniform(5, 10, 3)
S3_lats = np.random.uniform(0, 4, 3)

S4_lons = np.random.uniform(5, 9, 3)
S4_lats = np.random.uniform(6, 9, 3)

lons = np.concatenate((S0_lons, S1_lons, S2_lons, S3_lons, S4_lons))
lats = np.concatenate((S0_lats, S1_lats, S2_lats, S3_lats, S4_lats))

plot_cluster(lons,lats)
# plt.show()
plt.savefig("signals_locations.pdf", bbox_inches='tight')
```



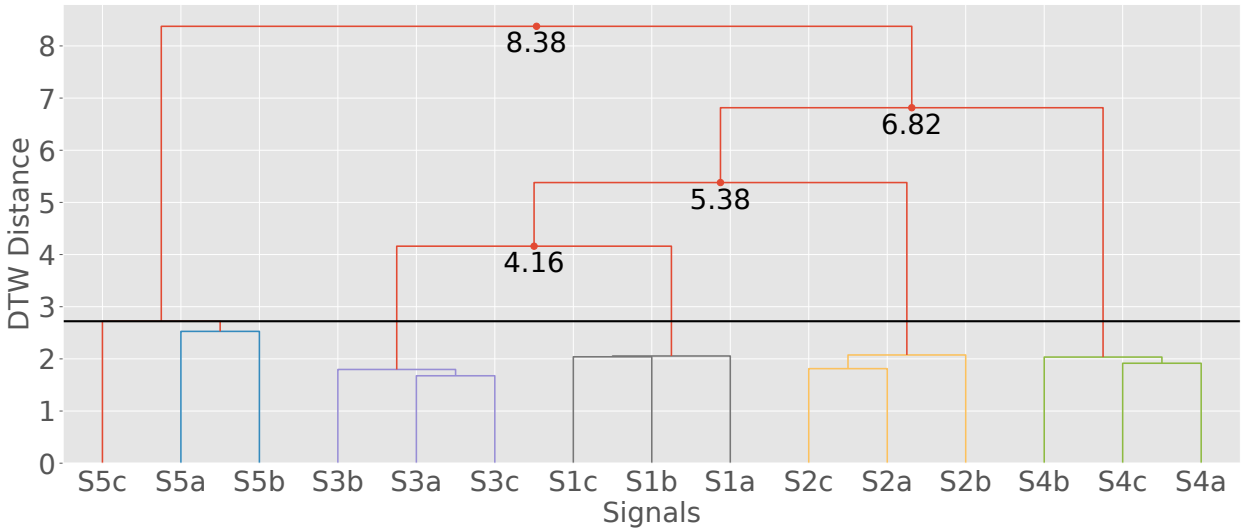
4.4 Reshuffle the noisy signals

```
shuffled_idx, shuffled_matrix = shuffle_signals(X, labels=[], plot_signals=False,␣
↪figsize=(10, 20))
shuffled_lons = lons[shuffled_idx]
shuffled_lats = lats[shuffled_idx]

labels = np.array(['S1a', 'S1b', 'S1c','S2a', 'S2b', 'S2c','S3a', 'S3b', 'S3c','S4a',
↪'S4b', 'S4c','S5a', 'S5b', 'S5c'])
newlabels = labels[shuffled_idx]

color = np.array(color)
color = color[shuffled_idx]
fig, ax = plot_signals(shuffled_matrix,figsize=(10,20), color=color, labels=newlabels)
plt.savefig("shuffled_signals.pdf", bbox_inches='tight')
```


S4a
S2b
S5b
S4c
S5c
S1a
S4b
S1b
S3c
S1c
S2a
S3b
S5a
S2c
S3a
0
20
40
60
80
100

4.5 Cluster reshuffled signals

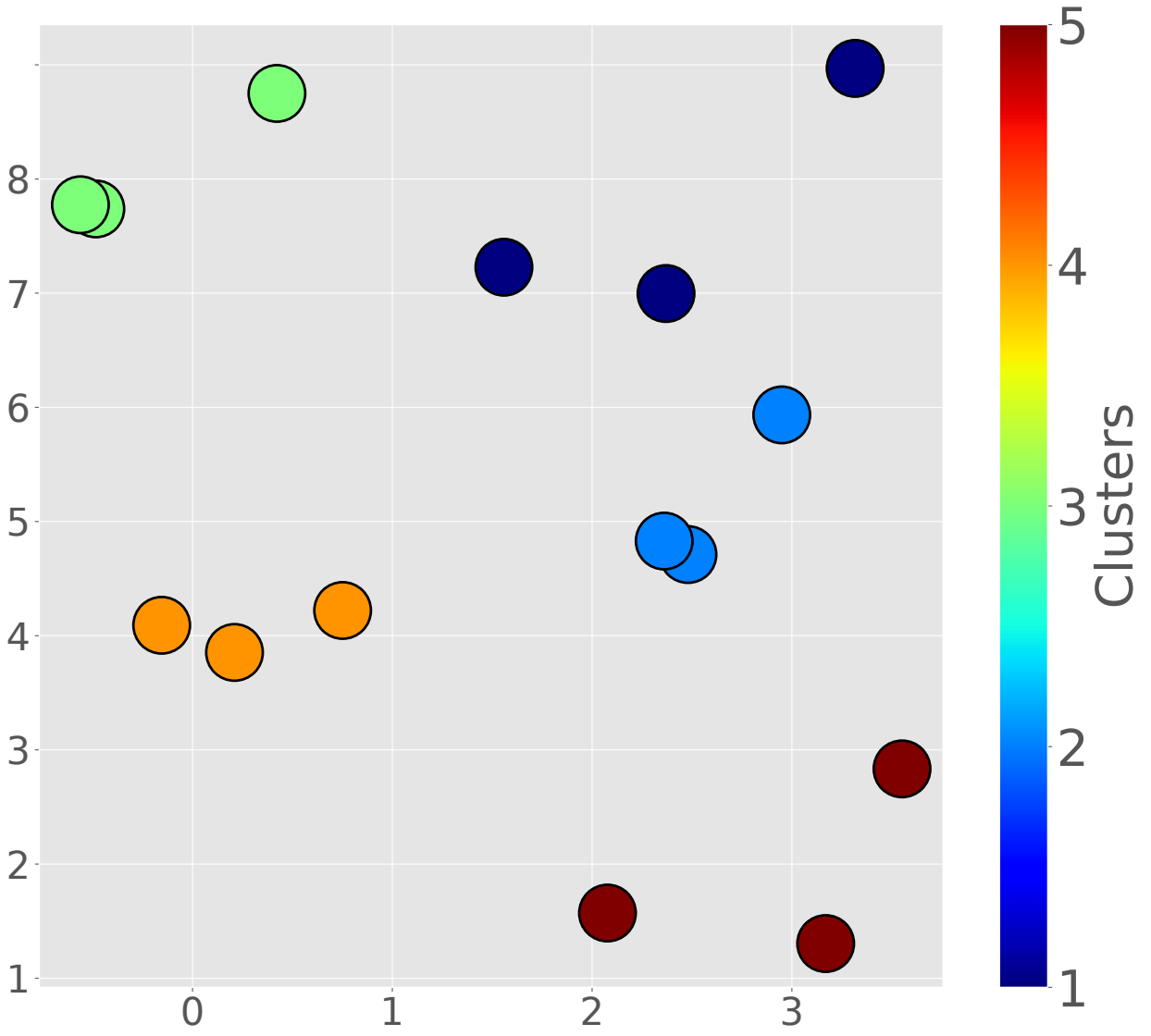
```
dtw_cluster2 = dtw_clustering(shuffled_matrix, labels=newlabels, longitudes=shuffled_
↪lons, latitudes=shuffled_lats)
dtw_cluster2.plot_dendrogram(annotate_above=3,xlabel="Signals", figname="example_dtw_
↪cluster.png",distance_threshold="optimal")
```



In the above dendrogram, we manually selected the threshold distance to be 3 to find the best clusters

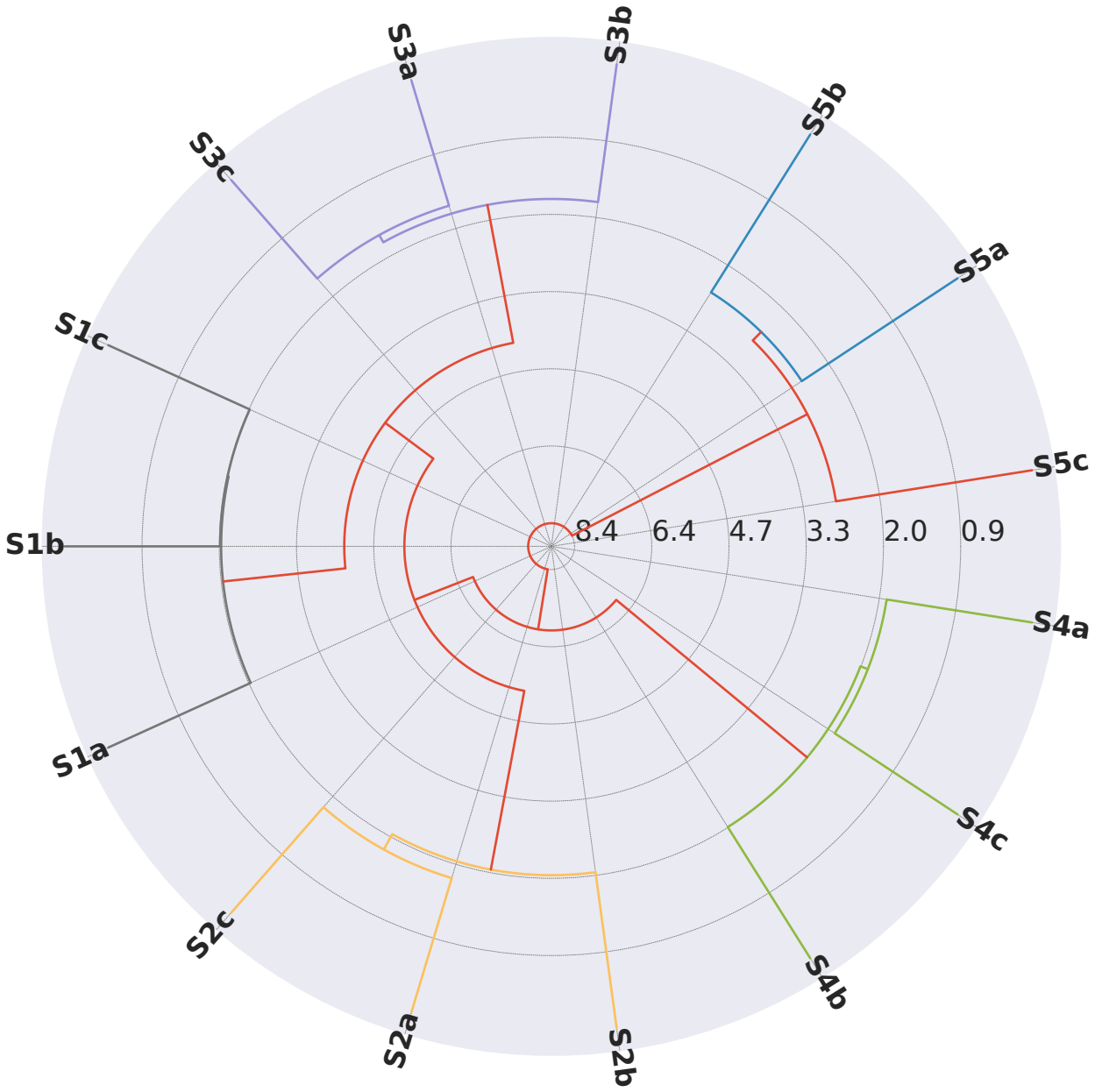
4.6 Plot the geographical locations of the clusters

```
dtw_cluster2.plot_cluster_xymap(dtw_distance=3, figname=None, xlabel='', ylabel='',␣
↪fontsize=40, markersize=200, tickfontsize=30, cbarsize=40)
plt.savefig("signals_cluster_xy_map.pdf", bbox_inches='tight', edgecolors='black',␣
↪linewidths=5)
```



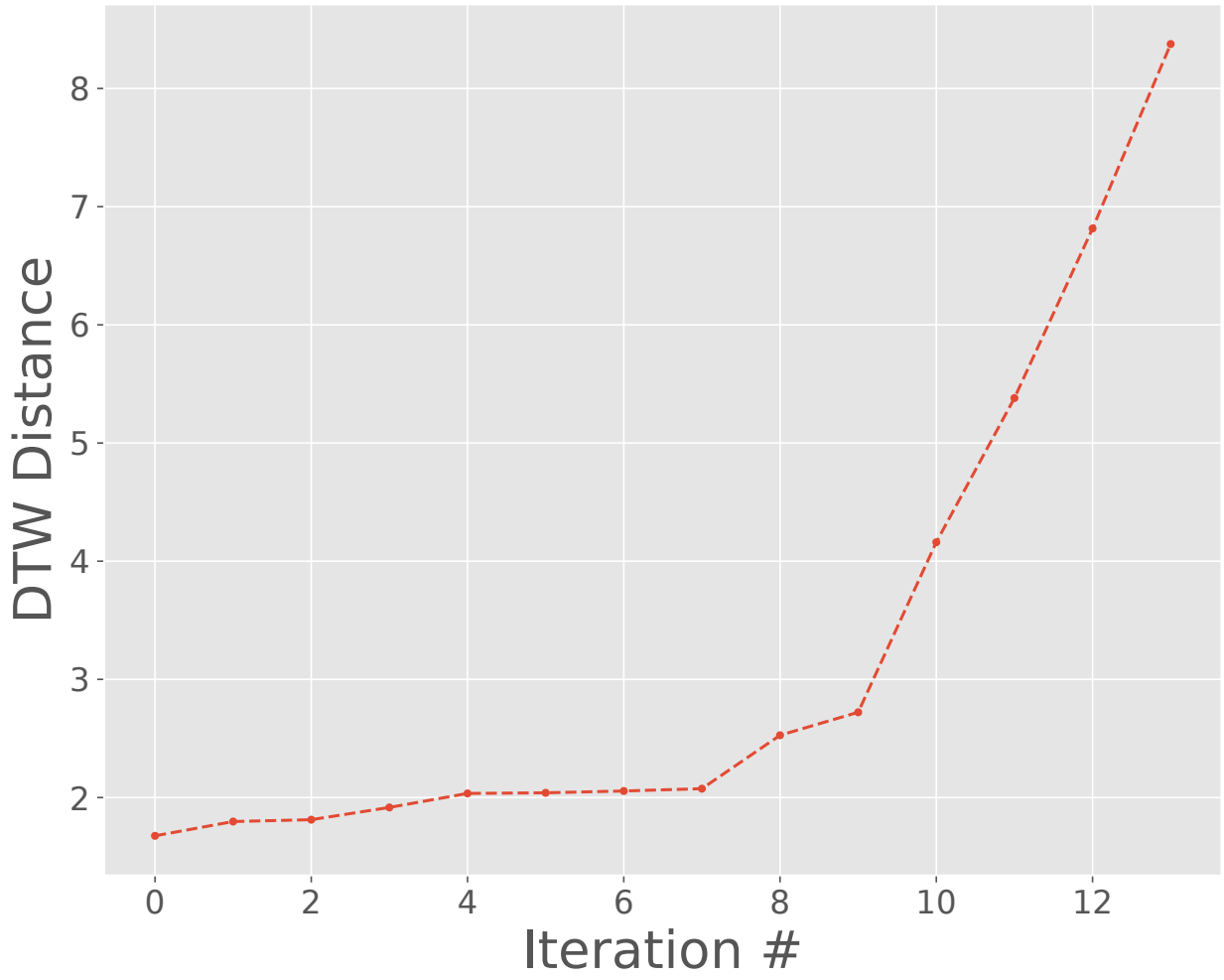
4.7 Polar dendrogram

```
kwargs_dendro={
"plotstyle":'seaborn',
"linewidth":5,
"gridwidth":0.8,
"gridcolor":'gray',
'xtickfontsize':60,
'ytickfontsize':60,
"figsize":(40,40),
"distance_threshold":"optimal" #use optimal number of clusters estimated by elbow method
}

dtw_cluster2.plot_polar_dendrogram(**kwargs_dendro)
plt.savefig("example_polar_dendro.pdf", bbox_inches='tight')
```

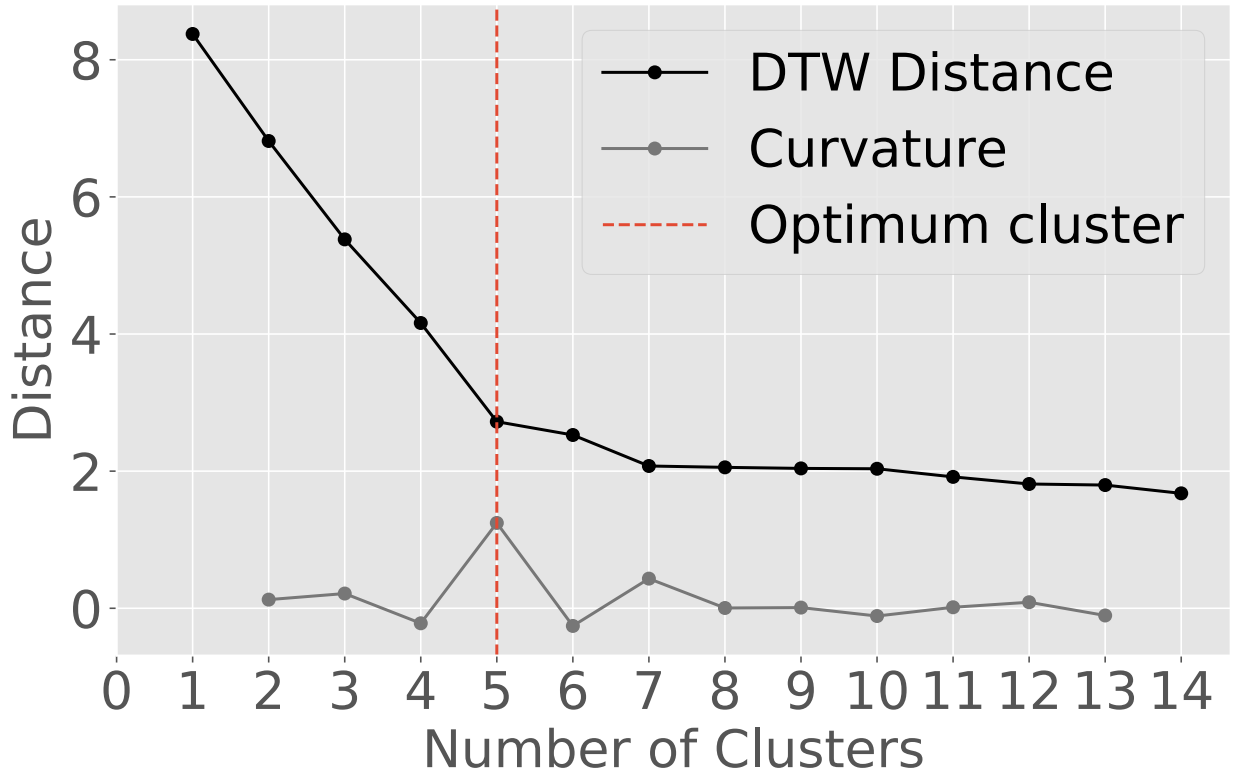


4.8 How the DTW distance changes with iterations to obtain the dendrogram?

```
dtw_cluster2.plot_hac_iteration()
plt.show()
```




```
dtw_cluster2.plot_optimum_cluster(legend_outside=False)
plt.savefig("optimum_clusters.pdf", bbox_inches='tight')
```



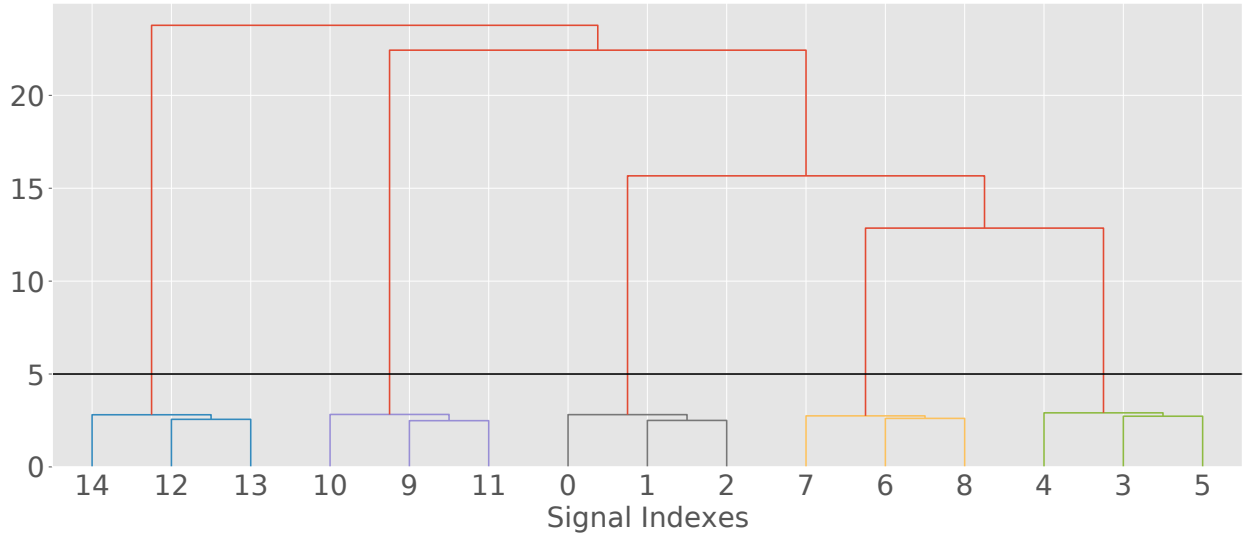
4.9 Euclidean distance-based cluster

```
def compute_linkage(model):
    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)
    return linkage_matrix

def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    linkage_matrix = compute_linkage(model)

    # Plot the corresponding dendrogram
```

(continues on next page)

(continued from previous page)

```
    dendrogram(linkage_matrix, **kwargs)


#'ward' minimizes the variance of the clusters being merged
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None, affinity=
↪'euclidean',linkage='ward')

model = model.fit(X)

plt.figure(figsize=(20, 8))
# plot the top three levels of the dendrogram
plot_dendrogram(model, p=5, color_threshold=5)
plt.xticks(fontsize=26)
plt.yticks(fontsize=26)
plt.axhline(y=5, c='k')


plt.xlabel("Signal Indexes")
plt.savefig('example_euclidean_cluster.pdf',bbox_inches='tight')
plt.close()
```



Both Euclidean and DTW based clustering results are similar. However, we can see some obvious differences. Let us list some of the similarity and differences for the above example.

- Both the results found 5 significant clusters.
- Both Euclidean and DTW based HAC found that the random function based time series (12, 13, 14) are most dissimilar
- The two closest clusters with DTW is sawtooth (6,7,8) and sine func(0,1,2). While that with the Euclidean, it is abs_cosine (6,7,8) and sawtooth fn (3,4,5)

The two results are similar because the signals considered for this example are stationary in nature.

CHAPTER

FIVE

DTWHACLUSTERING.ANALYSIS_SUPPORT

DTW HAC analysis support functions (*analysis_support*)

author Utpal Kumar, Institute of Earth Sciences, Academia Sinica

dtwhaclustering.analysis_support.**dec2dt**(*start*)
Convert the decimal type time array to the date-time type array

Parameters **start** (*list*) – list or numpy array of decimal year values e.g., [2020.001]

Returns date-time type array

Return type list

dtwhaclustering.analysis_support.**dec2dt_scalar**(*st*)
Convert the decimal type time value to the date-time type

Parameters **st** – scalar decimal year value e.g., 2020.001

Returns time as datetime type

Return type str

dtwhaclustering.analysis_support.**toYearFraction**(*date*)
Convert the date-time type object to decimal year

Parameters **date** – the date-time type object

Returns decimal year

Return type float

CHAPTER

SIX

DTWHACLUSTERING.PLOT_LINEAR_TREND

Plot linear trend values on a geographical map

author Utpal Kumar, Institute of Earth Sciences, Academia Sinica

dtwhaclustering.plot_linear_trend.**compute_interpolation**(*df*, *method='nearest'*, *lonrange=(120.0, 122.0)*, *latrange=(21.8, 25.6)*, *step=0.01*)

Interpolate linear trend values using Scipy's griddata and returns xarray object

Parameters

- **df** (*pandas.DataFrame*) – pandas dataframe containing the columns *lon*, *lat* and *slope*
- **method** (*str*) – Method of interpolation. One of {'linear', 'nearest', 'cubic'}. For more, see *scipy.interpolate.griddata*
- **lonrange** (*tuple*) – minimum and maximum values of the longitude to be interpolated
- **latrange** (*tuple*) – minimum and maximum values of the latitude to be interpolated
- **step** (*float*) – stepsize to interpolate data spatially

Returns *xarray.DataArray* of dims, and coords ("lat", "long"), maximum of the absolute interpolated array values

dtwhaclustering.plot_linear_trend.**plot_linear_trend_on_map**(*df*, *maplonrange=(120.0, 122.1)*, *maplatrange=(21.8, 25.6)*, *intrp_lonrange=(120.0, 122.0)*, *intrp_latrange=(21.8, 25.6)*, *outfig='Maps/slope-plot.png'*, *frame=['a1f0.25', 'WSen']*, *cmap='jet'*, *step=0.01*, *stn_labels=None*, *justify='left'*, *labelfont='6p,Helvetica-Bold,black'*, *offset='5p/-5p'*, *markerstyle='i10p'*, *defpen='1p,black'*, *stn_labels_color='black'*, *rand_justify=False*, *water_color='skyblue'*)

Plot the interpolated linear trend values along with the original data points on a geographical map using PyGMT

Parameters

- **df** (*pandas.DataFrame*) – Pandas dataframe containing the columns *lon*, *lat* and *slope*
- **maplonrange** (*tuple*) – longitude min/max of the output map
- **maplatrange** (*tuple*) – latitude min/max of the output map

- **intrp_lonrange** (*tuple*) – longitude min/max for the interpolation of data
- **intrp_latrange** (*tuple*) – latitude min/max for the interpolation of data
- **step** (*float*) – resolution of the interpolation
- **cmap** (*str*) – colormap for the output map
- **frame** (*list*) – frame of the output map. See PyGMT docs for details
- **outfig** (*str*) – output figure name with extension, e.g., *slope-plot.png*
- **water_color** (*str*) – color of the water, default: skyblue
- **stn_labels** (*array*) – label the station names. Only the stations provided will be labeled
- **justify** (*str*) – justification of the station labels - 'left', 'right'
- **rand_justify** (*boolean*) – randomly decide the justification for each labels
- **markerstyle** (*str*) – marker style and size

Returns None

```
kwargs={
    "labelfont":"16p,Helvetica-Bold,black",
    "justify":'right',
    "offset":"-10p/-1p",
    "maplonrange":(119.8, 122.1),
    "maplatrange":(21.8, 25.6),
    'markerstyle': 'i16p',
    'rand_justify':True,
    'water_color': 'gray'
}
comp = "U"
all_labels = {}
all_labels[comp] = ['PKGM','YMSM', 'VR02', 'HUWE', 'LNCH','CHEN','TUNH']

slopeFile=f'stn_slope_res_{comp}.txt'
df = pd.read_csv(slopeFile, names=['stn','lon','lat','slope'], delimiter='\s+')
figname = f"{outloc}/slope-plot_{comp}.png"
plot_linear_trend_on_map(df, outfig=figname,stn_labels=all_labels[comp], **kwargs)
```

CHAPTER

SEVEN

DTWHACLUSTERING.LEASTSQUAREMODELING

Least square modeling of GPS displacements for seasonality, tidal, co-seismic jumps.

author Utpal Kumar, Institute of Earth Sciences, Academia Sinica

class dtwhaclustering.leastSquareModeling.**LSQmodules**(*dUU*, *sel_eq_file=None*, *station_loc_file='helper_files/stn_loc.txt'*, *comp='U'*, *figdir='LSQOut'*, *periods=(13.6608, 14.7653, 27.5546, 182.62, 365.26, 6793.836)*)

Bases: object

compute_lsq(*plot_results=False*, *remove_trend=True*, *remove_seasonality=True*, *remove_jumps=True*, *plotformat=None*)

Compute the least-squares model using multithreading

Parameters

- **plot_results** (*boolean*) – plot the final results
- **remove_trend** (*boolean*) – return the time series after removing the linear trend
- **remove_seasonality** (*boolean*) – return the time series after removing the seasonal signals
- **remove_jumps** (*boolean*) – return the time series after removing the co-seismic jumps
- **plotformat** (*str*) – plot format of the output figure, e.g. "png". "pdf" by default.

jump(*t*, *t0*)

heaviside step function

Parameters

- **t** (*list*) – time data
- **t0** – earthquake origin time

dtwhaclustering.leastSquareModeling.**lsqmodeling**(*dUU*, *dNN*, *dEE*, *stnlocfile*, *plot_results=True*, *remove_trend=False*, *remove_seasonality=True*, *remove_jumps=False*, *sel_eq_file='helper_files/selected_eqs_new.txt'*, *figdir='LSQOut'*)

Least square modeling for the three component time series

Parameters

- **dUU** (*pandas.DataFrame*) – Vertical component pandas dataframe time series
- **dNN** (*pandas.DataFrame*) – North component pandas dataframe time series
- **dEE** (*pandas.DataFrame*) – East component pandas dataframe time series

- **plot_results** (*boolean*) – plot the final results
- **remove_trend** (*boolean*) – return the time series after removing the linear trend
- **remove_seasonality** (*boolean*) – return the time series after removing the seasonal signals
- **remove_jumps** (*boolean*) – return the time series after removing the co-seismic jumps
- **sel_eq_file** (*str*) – File containing the earthquake origin times e.g., *2009,1,15* with header info, e.g. *year_val,month_val,date_val*
- **stnlocfile** (*str*) – File containing the station location info, e.g., *DAWU,120.89004,22.34059*, with header *stn,lon,lat*

Returns Pandas dataframe corresponding to the vertical, north and east components e.g., final_dU, final_dN, final_dE

Return type pandas.DataFrame, pandas.DataFrame, pandas.DataFrame

```
from dtwhaclustering.leastSquareModeling import lsqmodeling
final_dU, final_dN, final_dE = lsqmodeling(dUU, dNN, dEE,stnlocfile="helper_files/
↪stn_loc.txt",  plot_results=True, remove_trend=False, remove_seasonality=True,␣
↪remove_jumps=False)
```

CHAPTER

EIGHT

DTWHACLUSTERING.DTW_ANALYSIS

Classes and functions for the DTW analysis and plotting maps and figures (*dtw_analysis*) This module is built around the dtaidistance package for the DTW computation and scipy.cluster

author Utpal Kumar, Institute of Earth Sciences, Academia Sinica

note See dtaidistance for details on HierarchicalTree, dtw, dtw_visualisation

class dtwhaclustering.dtw_analysis.**dtw_clustering**(*matrix*, *labels=[]*, *longitudes=[]*, *latitudes=[]*)

Bases: object

compute_cluster(*clusterMatrix=None*, *window='constrained'*, *windowfrac=0.25*)

Parameters **compute_cluster** – data matrix to cluster

Returns model, cluster_idx

compute_cut_off_inconsistency(*t=None*, *depth=2*, *criterion='inconsistent'*, *return_cluster=False*)

Calculate inconsistency statistics on a linkage matrix following *scipy.cluster.hierarchy.inconsistent*. It compares each cluster merge's height *h* to the average *avg* and normalize it by the standard deviation *std* formed over the depth previous levels

Parameters

- **t** (*scalar*) – threshold to apply when forming flat clusters. See scipy.cluster.hierarchy.fcluster for details
- **depth** (*int*) – The maximum depth to perform the inconsistency calculation

Returns maximum inconsistency coefficient for each non-singleton cluster and its children; the inconsistency matrix (matrix with rows of avg, std, count, inconsistency); cluster

compute_dendrogram(*color_thresh=None*)

compute_distance_accl(*clusterMatrix=None*)

compute_distance_matrix(*compact=True*, *block=None*)

get_linkage(*clusterMatrix=None*)

optimum_cluster_elbow(*minmax=False*, *plotloc=False*)

Gives the optimum number of clusters required to express the maximum difference in the similarity using the elbow method

plot_cluster_geomap(*dtw_distance='optimal'*, *minlon=None*, *maxlon=None*, *minlat=None*, *maxlat=None*, *figname='dtw_cluster.pdf'*, *colorbar=True*, *colorbarstep=1*, *doffset=1*, *dpi=720*, *topo_data='@earth_relief_15s'*, *plot_topo=False*, *markerstyle='c0.3c'*, *cmap_topo='topo'*, *cmap_data='jet'*, *projection='M4i'*, *topo_cpt_range='-8000/8000/1000'*, *landcolor='#666666'*)

Plot the cluster points on a geographical map

Parameters

- **figname** (*str*) – output figure name
- **colorbar** (*bool*) – plot colorbar
- **colorbarstep** (*int*) – step for the colorbar
- **cmap** (*str*) – colormap for the cluster points
- **projection** (*str*) – map projection
- **topo_data** (*str*) – topographic data resolution, see pygmt docs for details
- **topo_cpt_range** – min/max/step for topographic color
- **landcolor** (*str*) – color for land region
- **dpi** (*int*) – output figure resolution

plot_cluster_geomap_interpolated(*dtw_distance='optimal'*, *lonrange=(120.0, 122.0)*, *latrange=(21.8, 25.6)*, *gridstep=0.01*, *figname='dtw_cluster_interp.pdf'*, *minlon=None*, *maxlon=None*, *minlat=None*, *maxlat=None*, *markerstyle='c0.3c'*, *dpi=720*, *doffset=1*, *plot_data=True*, *plot_intrp=True*)

Parameters

- **dtw_distance** (*str or float*) – use *dtw_distance* value to obtain the cluster division. If *optimal* then the optimal *dtw_distance* will be calculated
- **lonrange** (*tuple*) – minimum and maximum of longitude values for interpolation
- **latrange** (*tuple*) – minimum and maximum of latitude values for interpolation
- **gridstep** (*float*) – step size for interpolation

plot_cluster_xymap(*dtw_distance='optimal'*, *figname=None*, *xlabel='x'*, *ylabel='y'*, *colorbar=True*, *colorbarstep=1*, *scale=2*, *fontsize=26*, *markersize=12*, *axesfontsize=20*, *xtickstep=1*, *tickfontsize=20*, *edgecolors='k'*, *cmap='jet'*, *linewidths=2*, *cbarsize=18*)

Plot the cluster points in a rectangular coordinate system

Parameters

- **dtw_distance** (*str or float*) – use *dtw_distance* value to obtain the cluster division. If *optimal* then the optimal *dtw_distance* will be calculated
- **figname** (*str*) – output figure name
- **colorbar** (*bool*) – plot colorbar
- **colorbarstep** (*int*) – step for the colorbar
- **scale** (*int*) – figure scale
- **markersize** – cluster points size
- **edgecolors** – marker edge color

- **cmap** (*str*) – colormap for the cluster points

Returns figure, axes

plot_dendrogram(*figname=None*, *figsize=(20, 8)*, *xtickfontsize=26*, *labelfontsize=26*, *xlabel='3-D Stations'*, *ylabel='DTW Distance'*, *truncate_p=None*, *distance_threshold=None*, *annotate_above=0*, *plotpdf=True*, *leaf_rotation=0*)

Parameters **truncate_p** – show only last truncate_p out of all merged branches

plot_hac_iteration(*figname=None*, *figsize=(10, 8)*, *xtickfontsize=26*, *labelfontsize=26*, *xlabel='Iteration #'*, *ylabel='DTW Distance'*, *plot_color='C0'*, *plotpdf=True*)

plot_optimum_cluster(*max_d=None*, *figname=None*, *figsize=(10, 6)*, *plotpdf=True*, *xlabel='Number of Clusters'*, *ylabel='Distance'*, *accl_label='Curvature'*, *accl_color='C10'*, *dist_label='DTW Distance'*, *dist_color='k'*, *xlim=None*, *legend_outside=True*, *fontsize=26*, *xlabelfont=26*, *ylabelfont=26*)

Parameters **xlim** (*list*) – x limits of the plot e.g., [1,2]

plot_polar_dendrogram(*figsize=(20, 20)*, *normfactor=None*, *Nyticks=7*, *gap=0.05*, *Nsmooth=100*, *linewidth=1*, *xtickfontsize=20*, *ytickfontsize=20*, *plotstyle='seaborn'*, *figname=None*, *plotpdf=True*, *gridcolor=None*, *gridstyle='--'*, *gridwidth=1*, *tickfontweight='bold'*, *distance_threshold=None*)

Plot polar dendrogram of the clustering result

Parameters

- **figsize** – figure size
- **normfactor** – (optional) normalization factor for the log spacing between the yticks, -np.log(dcoord+normfactor)
- **Nyticks** – number of y ticks
- **gap** – gap for the yticks
- **plotstyle** – matplotlib plot style, *plotstyle* by default
- **distance_threshold** – str, float. threshold for the coloring of branches. If str=”optimal”, then the optimal number of clusters based on elbow method will be used

plot_signals(*figname=None*, *figsize=(10, 6)*, *fontsize=26*)

Parameters

- **figname** (*str*) – output figure name
- **figsize** (*tuple*) – output figure size

significance_test(*numsimulations=10*, *outfile='pickleFiles/dU_accl_sim_results.pickle'*, *fresh_start=False*)

class dtwhaclustering.dtw_analysis.**dtw_signal_pairs**(*s1*, *s2*, *labels=['s1', 's2']*)

Bases: object

compute_distance(*pruning=True*, *best_path=False*)

Returns the DTW distance

Parameters **pruning** (*boolean*) – prunes computations by setting max_dist to the Euclidean upper bound

compute_warping_path(*windowfrac=None*, *psi=None*, *fullmatrix=False*)
Returns the DTW path

Parameters

- **windowfrac** – Fraction of the signal length. Only allow for shifts up to this amount away from the two diagonals.
- **psi** – Up to psi number of start and end points of a sequence can be ignored if this would lead to a lower distance
- **full_matrix** – The full matrix of all warping paths (or accumulated cost matrix) is built

plot_matrix(*windowfrac=0.2*, *psi=None*, *figname=None*, *shownumbers=False*, *showlegend=True*)
Plot the signals with the DTW matrix

Parameters **figname** (*str*) – output figure name

plot_signals(*figname=None*, *figsize=(12, 6)*)
Plot the signals

Parameters

- **figname** (*str*) – output figure name
- **figsize** (*tuple*) – output figure size

Returns figure and axes object

plot_warping_path(*figname=None*, *figsize=(12, 6)*)
Plot the signals with the warping paths

Parameters

- **figname** (*str*) – output figure name
- **figsize** (*tuple*) – output figure size

dtwhaclustering.dtw_analysis.**noise_robustness_test**(*clean_df*, *scale=0.1*)

dtwhaclustering.dtw_analysis.**plot_cluster**(*lons*, *lats*, *figname=None*, *figsize=(10, 10)*, *plotpdf=True*, *labels=[]*, *markersize=20*)

Parameters

- **figname** (*str*) – output figure name
- **figsize** (*tuple*) – output figure size

dtwhaclustering.dtw_analysis.**plot_signals**(*matrix*, *labels=[]*, *figname=None*, *plotpdf=True*, *figsize=(10, 8)*, *ylabelsize=26*, *color=None*)
Plot signals

Parameters

- **color** – list of colors. If None then matplotlib defaults color sequence will be used
- **figname** (*str*) – output figure name
- **figsize** (*tuple*) – output figure size

dtwhaclustering.dtw_analysis.**shuffle_signals**(*matrix*, *labels=[]*, *plot_signals=False*, *figsize=(10, 12)*, *figname=None*, *plotpdf=True*)

Parameters

- **figname** (*str*) – output figure name
- **figsize** (*tuple*) – output figure size

CHAPTER

NINE

DTWHACLUSTERING.PLOT_STATIONS

Plot topographic station map

author Utpal Kumar, Institute of Earth Sciences, Academia Sinica

dtwhaclustering.plot_stations.**plot_station_map**(*station_data*, *minlon=None*, *maxlon=None*, *minlat=None*, *maxlat=None*, *outfig='station_map.png'*, *datacolor='blue'*, *topo_data='@earth_relief_15s'*, *cmap='etopo1'*, *projection='M4i'*, *datalabel='Stations'*, *markerstyle='i10p'*, *random_station_label=False*, *stn_labels=None*, *justify='left'*, *labelfont='6p,Helvetica-Bold,black'*, *offset='5p/-5p'*, *stn_labels_color='red'*, *rand_justify=False*)

Plot topographic station map using PyGMT

Parameters

- **station_data** – Pandas dataframe containing columns *lon*, *lat*
- **minlon** – Minimum longitude of the map (optional)
- **maxlon** – Maximum longitude of the map (optional)
- **minlat** – Minimum latitude of the map (optional)
- **maxlat** – Maximum latitude of the map (optional)
- **datacolor** – color of the data point to plot
- **topo_data** – etopo data file
- **cmap** – colormap for the output topography
- **projection** – projection of the map. Mercator of width 4 inch by default
- **datalabel** – Label for the data
- **markerstyle** – Style of the marker. Inverted triangle of size 10p by default.
- **outfig** – Output figure path
- **random_station_label** – int. label randomly selected *random_station_label* of stations

```
from dtwhaclustering import plot_stations

outloc=os.path.join("Figures","Maps")
figname = f'{outloc}/station_map.pdf'
kwargs={
    "labelfont":"10p,Helvetica-Bold,black",
```

(continues on next page)

(continued from previous page)

```
    "outfig":figname,
    "stn_labels":['PKGM','YMSM', 'VR02', 'HUWE', 'LNCH','CHEN','TUNH','ERPN','FALI',
↪'WANS'],
    "justify":'right',
    "offset":"-10p/-1p",
    'markerstyle': 'i12p',
    'stn_labels_color': 'red',
    'rand_justify': True
}
plot_stations.plot_station_map(station_data = 'helper_files/selected_stations_info.
↪txt',**kwargs)
```

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

P

S

T