# stochprop Documentation

***Release 1.0***

**P. Blom**

**Aug 28, 2020**

# CONTENTS

Simulations of infrasonic propagation in the atmosphere typically utilize a single atmospheric specification describing the acoustic sound speed, ambient winds, and density as a function of altitude. Due to the dynamic and sparsely sampled nature of the atmosphere, there is a notable amount of uncertainty in the atmospheric state at a given location and time so that a more robust analysis of infrasonic propagation requires inclusion of this uncertainty. This Python library, stochprop, has been implemented using methods developed jointly by infrasound scientists at Los Alamos National Laboratory (LANL) and the University of Mississippi's National Center for Physical Acoustics (NCPA). This software library includes methods to quantify variability in the atmospheric state, identify typical seasonal variability in the atmospheric state and generate suites of representative atmospheric states during a given season, as well as perform uncertainty analysis on a specified atmospheric state given some level of uncertainty. These methods have been designed to interface between propagation modeling capabilities such as InfraGA/GeoAc and NCPAprop and signal analysis methods in the LANL InfraPy tool.

CHAPTER

# ONE

# CONTENTS

## 1.1 Authorship & License Info

stochprop is being developed and maintained by Dr. Philip Blom (pblom at lanl.gov)

The software license for this Python library is being processed by the Los Alamos National Laboratory's Feynman Center for Innovation and will be a limited release "Academic and Research"/"US Government" license until a manuscript in progress detailing the methods has been published. Please contact Philip Blom to obtain the user survey needed to gain access to the software.

*License text here once finalized.*

## 1.2 Installation

### 1.2.1 Anaconda

The installation of stochprop is ideally completed using pip through Anaconda to resolve and download the correct python libraries. 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/.

### 1.2.2 Installing Dependencies

#### Propagation Modeling Methods

A subset of the stochprop methods require access to the LANL InfraGA/GeoAc ray tracing methods as well as the NCPAprop normal mode methods. Many of the empirical orthogonal function (EOF) based atmospheric statistics methods can be used without these propagation tools, but full usage of stochprop requires them.

- InfraGA/GeoAc: https://github.com/LANL-Seismoacoustics/infraGA
- NCPAprop: https://github.com/chetzer-ncpa/ncpaprop

#### InfraPy Signal Analysis Methods

The propagation models constructed in stochprop are intended for use in the Bayesian Infrasonic Source Localization (BISL) and Spectral Yield Estimation (SpYE) methods in the LANL InfraPy signal analysis software suite. As with the InfraGA/GeoAc and NCPAprop linkages, many of the EOF-based atmospheric statistics methods can be utilized without InfraPy, but full usage will require installation of InfraPy (https://github.com/LANL-Seismoacoustics/infrapy).

### 1.2.3 Installing stochprop

Once Anaconda is installed, you can install stochprop using pip by navigating to the base directory of the package (there will be a file there named setup.py). Assuming InfraPy has been installed within a conda environment called infrapy_env, it is recommended to install stochprop in the same environment using:

```
>> conda activate infrapy_env
>> pip install -e .
```

Otherwise, a new conda environment should be created with the underlying dependencies and pip should be used to install there (work on this later):

```
>> conda env create -f stochprop_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 stochprop_env
```

To deactivate an active environment, use

```
>> conda deactivate
```

### 1.2.4 Testing stochprop

Once the installation is complete, you can test the methods by navigating to the /examples directory located in the base directory, and running:

```
>> python eof_analysis.py
>> python atmo_analysis.py
```

A set of propagation analyses are included, but require installation of infraGA/GeoAc and NCPAprop. These analysis can be run to ensure linkages are working between stochprop and the propagation libraries, but note that the simulation of propagation through even the example suite of atmosphere takes a significant amount of time.

## 1.3 Stochastic Propagation Analysis

- The atmospheric state at a given time and location is uncertain due to its dynamic and sparsely sampled nature
- Propagation effects for infrasonic signals must account for this uncertainty in order to properly quantify uncertainty in analysis results
- A methodology of constructing propagation statistics has been developed that identifies a suite of atmospheric states that characterize the possible space of scenarios, runs propagation simulations through each possible state, and builds statistical distributions for propagation effects

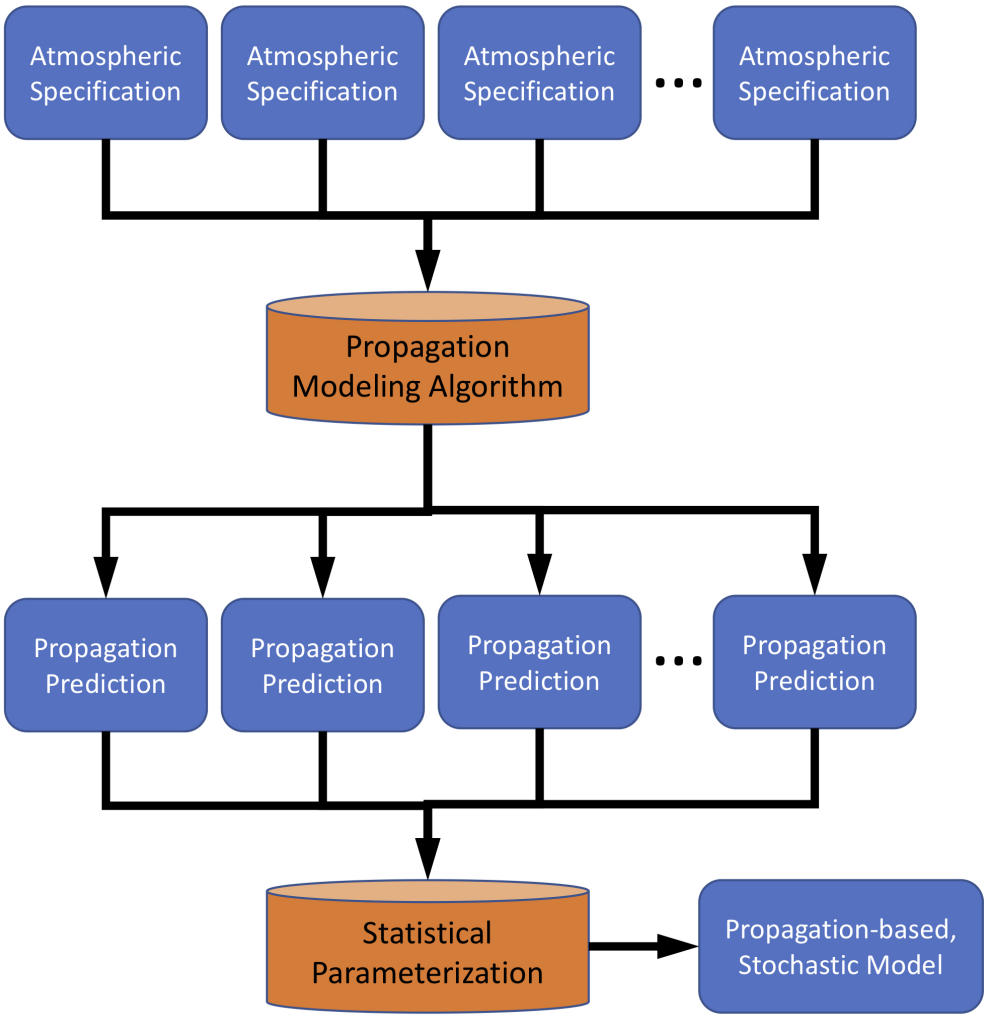


Fig. 1: Stochastic propagation models are constructing using a suite of possible atmospheric states, propagation modeling applied to each, and a statistical model describing the variability in the resulting set of predicted effects

- The tools included here provide a framework for constructing such models as well as perform a number of analyses related to atmospheric variability and uncertainty

### 1.3.1 Empirical Orthogonal Function Analysis

- Empirical Orthogonal Functions (EOFs) provide a mathematical means of measuring variations in the atmospheric state
- Methods measure EOF statistics to reduce the number of atmospheric samples necessary to characterize the atmosphere at a given location during a specified time period

### 1.3.2 Atmospheric Fitting, Sampling, and Perturbation

- EOFs can be used to fit a specified atmosphere by computing coefficients for each EOF
- Statistics of the coefficients for a suite of atmospheric states can be used to generate a set of characteristics samples
- Randomly generated EOF coefficients can be used to generate perturbations to an initial atmospheric specification and construct a suite of atmospheric states that fall within expected uncertainty

### 1.3.3 Propagation Statistics

- InfraGA/GeoAc ray tracing analysis can be applied to a suite of atmospheric states to predict geometric propagation characteristics such as arrival location, travel time, and direction of arrival needed to estimate the source location
- NCPAprop modal simulations can be applied to a suite of atmospheric states to predict finite frequency transmission loss needed to characterize the infrasonic source

#### Empirical Orthogonal Function Analysis

- Empirical orthogonal functions (EOFs) are a mathematical tool useful for characterizing a suite of vectors or functions via construction of basis vectors or functions.
- Consider $N$ fields, $a_n(\vec{z})$, sampled at $M$ points, $z_m$ that define a matrix,

$$A\left(\vec{z}\right) = \begin{pmatrix} a_1\left(z_1\right) & a_2\left(z_1\right) & \cdots & a_N\left(z_1\right) \\ a_1\left(z_2\right) & a_2\left(z_2\right) & \cdots & a_N\left(z_2\right) \\ \vdots & \vdots & \ddots & \vdots \\ a_1\left(z_M\right) & a_2\left(z_M\right) & \cdots & a_N\left(z_M\right) \end{pmatrix}$$

- Analysis of this $N \times M$ matrix to compute EOFs entails first extracting the mean set of values and then applying a singular value decomposition (SVD) to define singular values and orthogonal functions,

$$A\left(\vec{z}\right) \xrightarrow{\text{SVD}} \bar{a}\left(z_m\right), \mathcal{S}_n^{(a)}, \mathcal{E}_n^{(A)}\left(z_m\right)$$

- The resulting EOF information can be used to reproduce any other other field sampled on the same set of points,

$$\hat{b}\left(z_m\right) = \bar{a}\left(z_m\right) + \sum_n \mathcal{C}_n^{(b)} \mathcal{E}_n^{(A)}\left(z_m\right),$$

$$\mathcal{C}_n^{(b)} = \sum_m \mathcal{E}_n^{(A)}\left(z_m\right)\left(b\left(z_m\right) - \bar{a}\left(z_m\right)\right),$$

- Note that the coefficients, $\mathcal{C}_n^{(b)}$, are defined by the projection of the new function onto each EOF (accounting for the mean, $\bar{a}$)

- Consider a second matrix, $B(\vec{z})$ defined by a set of $K$ fields, $b_k(\vec{z})$. Each of these columns produces a set of coefficients that can be used to define a distribution via a kernel density estimate (KDE),

$$\left\{\mathcal{C}_n^{(b_1)}, \mathcal{C}_n^{(b_2)}, \ldots, \mathcal{C}_n^{(b_K)}\right\} \xrightarrow{\text{KDE}} \mathcal{P}_n^{(B)}(\mathcal{C}).$$

- Comparison of the distributions for various matrices, $B_1, B_2, B_3, \ldots$, allows one to define the relative similarity between differen sets by computing the overlap and weighting each term by the EOF singular values,

$$\Gamma_{j,k} = \sum_n \mathcal{S}_n^{(\text{all})} \int \mathcal{P}_n^{(B_j)}(\mathcal{C})\, \mathcal{P}_n^{(B_k)}(\mathcal{C})\, d\mathcal{C}$$

- In the case of EOF analysis for atmospheric seasonality and variability, each $a_m(\vec{z})$ is an atmospheric specification sampled at a set of altitudes, $\vec{z}$, and the set of atmospheric states in $A$ includes all possible states for the entire year (and potentially multiple years). The sets of atmospheres in each matrix, $B_j$, is a subset of $A$ corresponding to a specific month or other interval. The coefficient overlap can be computed for all combinations to identify seasonality and determine the grouping of intervals for which propagation effects will be similar.

### EOF methods in stochprop

- Empirical Orthogonal Function analysis methods can be accessed by importing `stochprop.eofs`
- Although analysis can be completed using any set of user defined paths, it is recommended to build a set of directories to hold the eof results, coefficient analyses, and samples produced from seasonal analysis. This pre-analysis set up can be completed manually or by running:

```
import os
import subprocess
import numpy as np

from stochprop import eofs

if __name__ == '__main__':
        eof_dirs = ["eofs", "coeffs", "samples"]
        season_labels = ["winter", "spring", "summer", "fall"]

        for dir in eof_dirs:
                if not os.path.isdir(dir):
                        subprocess.call("mkdir " + dir, shell=True)

        for season in season_labels:
                if not os.path.isdir("samples/" + season):
                        subprocess.call("mkdir samples/" + season,␣
↪shell=True)
```

### Load Atmosphere Specifications

- Atmospheric specifications are available through a number of repositories including the Ground-to-Space (G2S) system, the European Centre for Medium-Range Weather Forecasts (ECMWF), and other sources
- A convenient source for G2S specifications is the University of Mississippi's National Center for Physical Acoustics (NCPA) G2S server at http://g2s.ncpa.olemiss.edu
- The current implementation of EOF methods in stochprop assumes the ingested specifications are formatted such that the columns contain altitude, temperature, zonal winds, meridional winds, density, pressure (that is, `zTuvdp` in the infraGA/GeoAc profile options), which is the default output format of the G2S server at NCPA. Note: a script is included in the infraGA/GeoAc methods to extract profiles in this format from ECMWF netCDF files.
- The atmosphere matrix, $A(\vec{z})$ can be constructed using `stochprop.eofs.build_atmo_matrix` which accepts the path where specifications are located and a pattern to identify which files to ingest.
    - For seasonal analysis, it is useful to initially separate atmospheric specifications by month. Assuming subdirectories labeled "01", "02", "03", ... "12" contain profiles for January through December (see the subdirectories in the examples directory of the package), atmospheres can be ingested and combined using `numpy.vstack`
    - The optional argument, ref_alts, should be used to ensure a common vertical sampling if multiple directories are being ingested in this manner.

```
A, z0 = eofs.build_atmo_matrix("profs/01/", "g2stxt_*")
for n in range(2, 13):
        A_temp, _ = eofs.build_atmo_matrix("profs/{:02d}/".
↪format(n), "g2stxt_*", ref_alts=z0)
        A = np.vstack((A, A_temp))
```

    - Alternately, if all profiles are contained within a common directory, ingestion can be completed using a single call,

```
A, z0 = eofs.build_atmo_matrix("profs/", "g2stxt_*")
```

### Computing EOFs

- Once the atmosphere matrix, $A(\vec{z})$ has been ingested, EOF analysis can be completed using:

```
eofs.compute_eofs(A, z0, "eofs/examples")
```

- The analysis results are written into files with prefix specified in the function call ("eofs/examples" in this case). The contents of the files are summarized is the below table.

| EOF Output File | Description |
|---|---|
| eofs/example-mean_atmo.dat | Mean values, $\bar{a}\left(\vec{z}\right)$ in the above discussion |
| eofs/example-singular_values.dat | Singular values corresponding each EOF index |
| eofs/example-adiabatic_snd_spd.eofs | EOFs for the adiabatic sound speed, $c_{\text{ad}} = \sqrt{\gamma \frac{p}{\rho}}$ |
| eofs/example-ideal_gas_snd_spd.eofs | EOFs for the ideal gas sound speed, $c_{\text{ad}} = \sqrt{\gamma R T}$ |
| eofs/example-merid_winds.eofs | EOFs for the meridional (north/south) winds |
| eofs/example-zonal_winds.eofs | EOFs for the zonal (east/west) winds |

- The EOF file formats is such that the first column contains the altitude points, $\vec{z}$, and each subsequent column contains the $n^{th}$ EOF, $\mathcal{E}_n^{(A)}(\vec{z})$
- As discussed in Waxler et al. (2020), the EOFs are computed using stacked wind and sound speed values to conserve coupling between the different atmospheric parameters and maintain consistent units (velocity) in the EOF coefficients
- The resulting EOFs can be used for a number of analyses including atmospheric updating, seasonal studies, perturbation analysis, and similar analyses

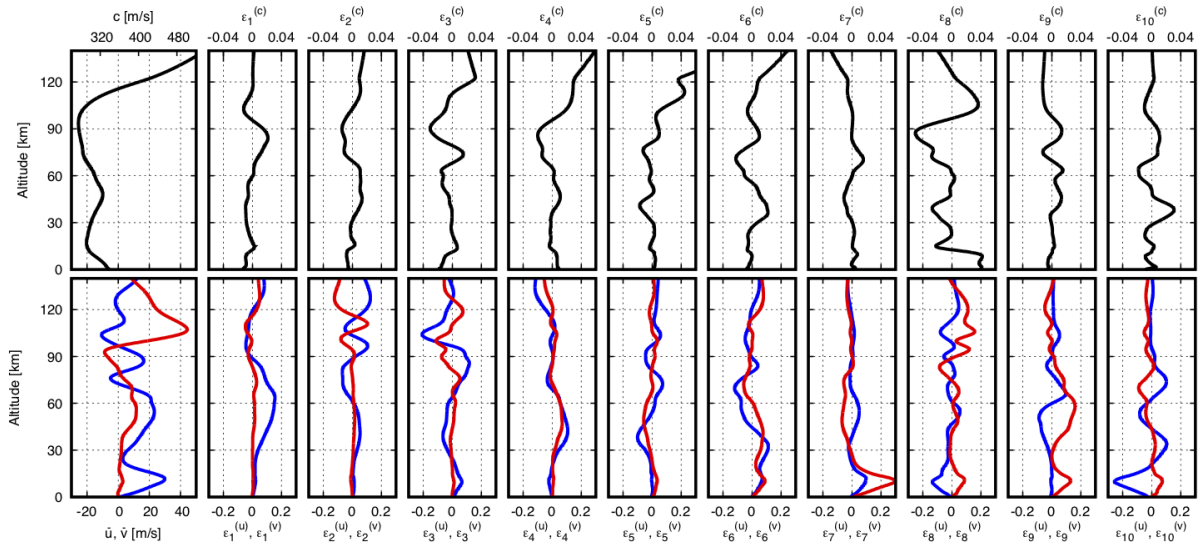


Fig. 2: Mean atmospheric states (left) and the first 10 EOFs for the adiabatic sound speed (upper row) and zonal and meridional winds (lower row, blue and red, respectively) for analysis of the atmosphere in the northeastern US

### Compute Coefficients and Determine Seasonality

- Using the EOFs for the entire calendar year, coefficient sets can be defined for individual months (or other sub-intervals) using the `stochprop.eofs.compute_coeffs` function.
- For identification of seasonality by month, the coefficient sets are first computed for each individual month using:

```
coeffs = [0] * 12
for m in range(1, 13):
    Am, zm = eofs.build_atmo_matrix("profs/{:02d}/".format(m), g2stxt_*")
    coeffs[m - 1] = eofs.compute_coeffs(Am, zm, "eofs/example", "coeffs/
↪example_{:02d}".format(m), eof_cnt=eof_cnt)
```

- The resulting coefficient sets are analyzed using `stochprop.eofs.compute_overlap` to identify how similar various month pairs are:

```
overlap = eofs.compute_overlap(coeffs, eof_cnt=eof_cnt)
eofs.compute_seasonality("coeffs/example-overlap.npy", "eofs/example",
↪"coeffs/example")
```

- The output of this analysis is a dendrogram identifying those months that are most similar. In the below result, May - August is identified as a consistent "summer" season, October - March as

"winter", and September and April as "spring/fall" transition between the two dominant seasons

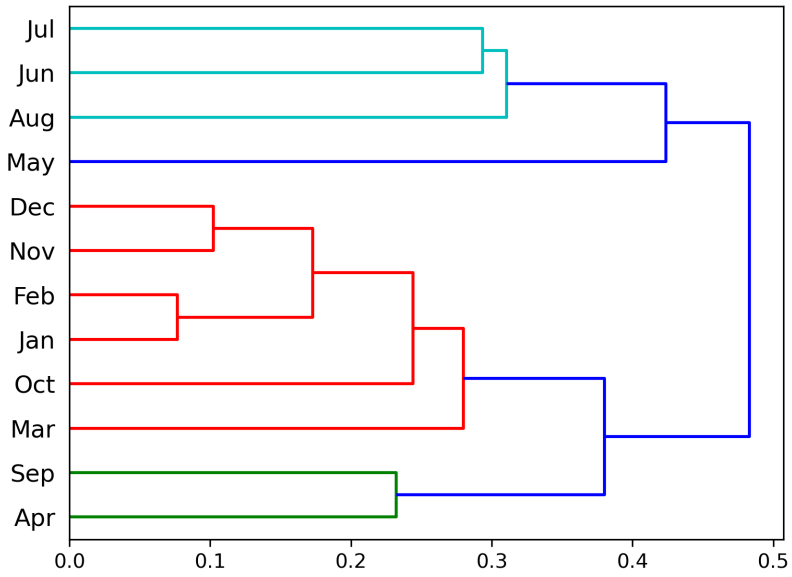


Fig. 3: Clustering analysis on coefficient overlap is used to identify which months share common atmospheric structure

### Atmospheric Fitting, Sampling, and Perturbation

- The Empirical Orthogonal Functions (EOFs) constructed using a suite of atmospheric specifications can be utilized in a number of different analyses of the atmospheric state
- In general, an atmospheric state can be constructed by defining a reference atmosphere, $b_0\left(z_m\right)$, and a set of coefficients, $\mathcal{C}_n$,

$$\hat{b}\left(z_m\right) = b_0\left(z_m\right) + \sum_n \mathcal{C}_n \mathcal{E}_n\left(z_m\right),$$

### Fitting an Atmospheric Specification using EOFs

- In the case that a specific state, $b\left(z_m\right)$, is known, it can be approximated using the EOF set by using the mean state pulled from the original SVD analysis and coefficients defined by projecting the atmospheric state difference from this mean onto each EOF,

$$b_0\left(z_m\right) = \bar{a}\left(z_m\right), \qquad \mathcal{C}_n^{(b)} = \sum_m \mathcal{E}_n\left(z_m\right)\left(b\left(z_m\right) - \bar{a}\left(z_m\right)\right),$$

- These coefficient calculations and construction of a new atmospheric specification can be completed using `stochprop.eofs.fit_atmo` with the path to specific atmospheric state, a set of EOFs, and a specified number of coefficients to compute,

```
prof_path = "profs/01/g2stxt_2010010100_39.7393_-104.9900.dat"
eofs_path = "eofs/example"

eofs.fit_atmo(prof_path, eofs_path, "eof_fit-N=30.met", eof_cnt=30)
```

- This analysis is useful to determine how many coefficients are needed to accurately reproduce an atmospheric state from a set of EOFs. Such an analysis is shown below for varying number of coefficients and convergence is found at 50 - 60 terms.

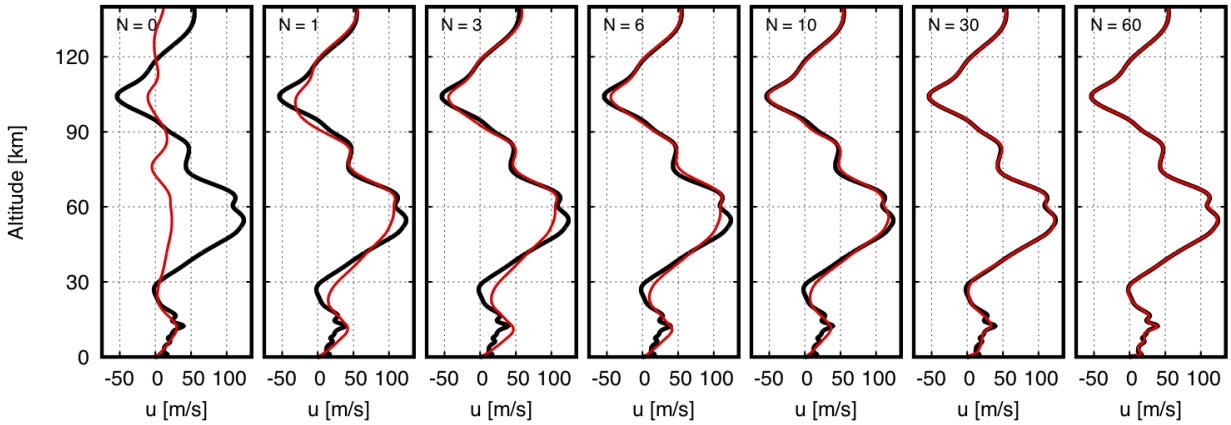


Fig. 4: Accuracy of fitting a specific atmospheric state (black) using varying numbers of EOF coefficients (red) shows convergence for approximately 50 - 60 terms in the summation

### Sampling Specifications using EOF Coefficient Distributions

- Samples can be generated that are representative of a given coefficient distributions as constructed using `stochprop.eofs.compute_coeffs` or a combination of them.
- In such a case, the reference atmosphere is again the mean state from the SVD analysis and the coefficients are randomly generated from the distributions defined by kernel density estimates (KDE's) of the coefficient results

$$b_0^{(B)}(z_m) = \bar{a}(z_m), \qquad \mathcal{C}_n \longleftarrow \mathcal{P}_n^{(B)}(\mathcal{C})$$

- In addition to sampling the coefficient distributions, the maximum likelihood atmospheric state can be defined by defining each coefficient to be the maximum of the distribution,

$$b_0^{(B)}(z_m) = \bar{a}(z_m), \qquad \mathcal{C}_n = \text{argmax}\left[\mathcal{P}_n^{(B)}(\mathcal{C})\right]$$

- This sampling and maximum likelihood calculation can be run by loading coefficient results and running,

```
coeffs = np.load("coeffs/example_05-coeffs.npy")
coeffs = np.vstack((coeffs, np.load("coeffs/example_06-coeffs.npy")))
coeffs = np.vstack((coeffs, np.load("coeffs/example_07-coeffs.npy")))
coeffs = np.vstack((coeffs, np.load("coeffs/example_08-coeffs.npy")))

eofs.sample_atmo(coeffs, eofs_path, "samples/summer/example-summer", prof_
↪cnt=25)
eofs.maximum_likelihood_profile(coeffs, eofs_path, "samples/example-summer
↪")
```

- This analysis can be completed for each identified season to generate a suite of atmospheric specifications representative of the season as shown in the figure below. This can often provide a significant amount of data reduction for propagation studies as multiple years of specifications (numbering in the 100's or 1,000's) can be used to construct a representative set of 10's of atmospheres that characterize the time period of interest as in the figure below.

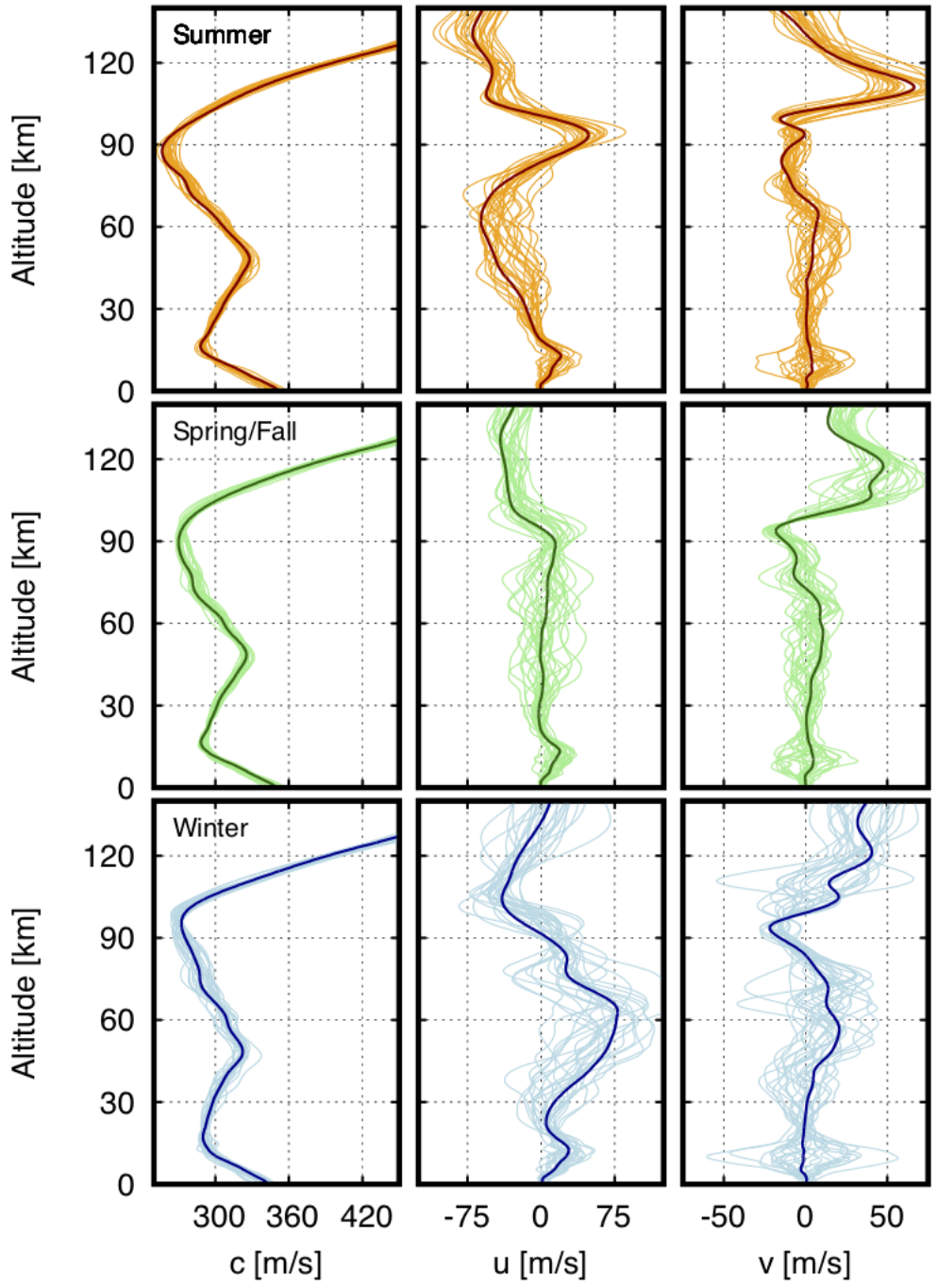


Fig. 5: Samples for seasonal trends in the western US show the change in directionality of the stratospheric waveguide in summer and winter

### Perturbing Specifications to Account for Uncertainty

- In most infrasonic analysis, propagation analysis through a specification for the approximate time and location of an event doesn't produce the exact arrivals observed due to the dynamic and sparsely sampled nature of the atmosphere
- Because of this, it is useful to apply random perturbations to the estimated atmospheric state covering some confidence level and consider propagation through the entire suite of "possible" states
- In such a case, the reference atmosphere, $c_0(z_m)$ defines the initial states, coefficients are randomly generated from a normal distribution, and weighting is applied based on the singular values and mean altitudes of the EOFs,

$$b_0(z_m) = c_0(z_m), \qquad \mathcal{C}_n \longleftarrow \mathcal{N}(0, \sigma^*), \qquad w_n = \mathcal{S}_n^{\gamma}\, \bar{z}_n^{\eta}$$

- The set of perturbations is scaled to match the specified standard deviation after summing over coefficients and averaged over the entire set of altitudes
- Unlike the above methods, in this analysis a weighting is defined by the singular value of the associated EOF and the mean altitude of the EOF, $\bar{z}_n = \sum_m z_m \mathcal{E}_n(z_m)$ in order to avoid rapidly oscillating EOFs from contributing too much noise and to focus perturbations at higher altitudes where uncertainties are larger, respectively. The exponential coefficients have default values of $\gamma = 0.25$ and $\eta = 2$, but can be modified in the function call.
- This perturbation analysis can be completed using `stochprop.eofs.perturb_atmo` with a specified starting atmosphere, set of EOFs, output path, uncertainty measure in meters-per-second, and number of samples needed,

```
eofs.perturb_atmo(prof_path, eofs_path, "eof_perturb", uncertainty=5.0,␣
↪sample_cnt=10)
```

- The below figure shows a sampling of results using uncertainties of 5.0, 10.0, and 15.0 meters-per-second. The black curve is input as the estimated atmospheric state and the red curves are generated by the perturbations.

### Propagation Statistics

- Propagation statistics for path geometry (e.g., arrival location, travel time, direction of arrival) and transmission loss can be computed for use in improving localization and yield estimation analyses, respectively.
- In the case of localization, a general celerity (horizontal group velocity) model is available in InfraPy constructed as a three-component Gaussian-mixture-model (GMM). This model contains peaks corresponding to the tropospheric, stratospheric, and thermospheric waveguides and has been defined by fitting the parameterized GMM to a kernel density estimate of a full year of ray tracing analyses.
- More specific models can be constructed from a limite suite of atmospheric states describing a location and seasonal trend (e.g., winter in the western US) or using an atmospheric state for a specific event with some perturbation analysis. In either case, propagation simulations are run using the suite of atmospheric states and a statistical model is defined using the outputs to quantify the probability of a given arrival characteristic.

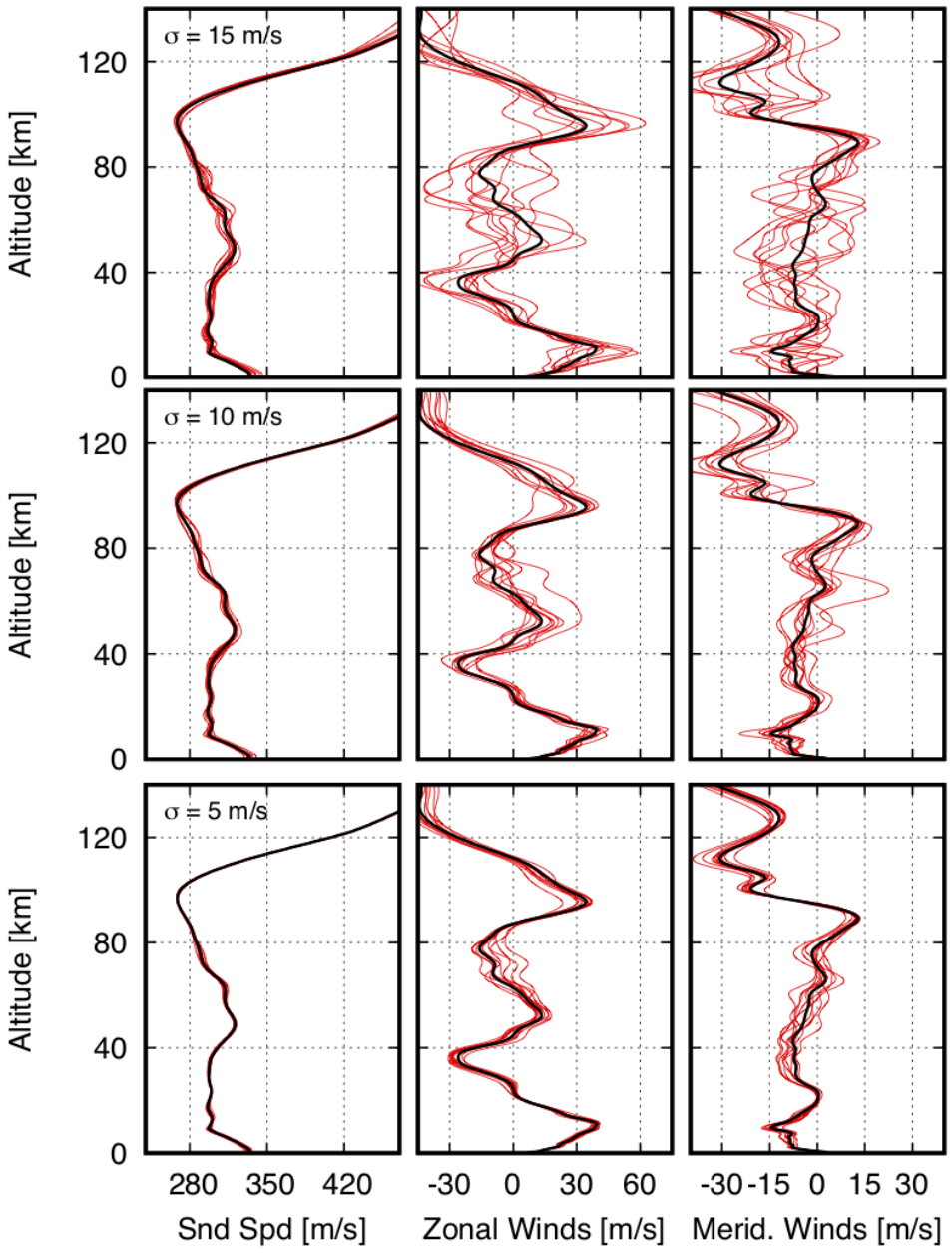


Fig. 6: Perturbations to a reference atmospheric state can be computed using randomly generated coefficients for a suite of EOFs with specified standard deviation

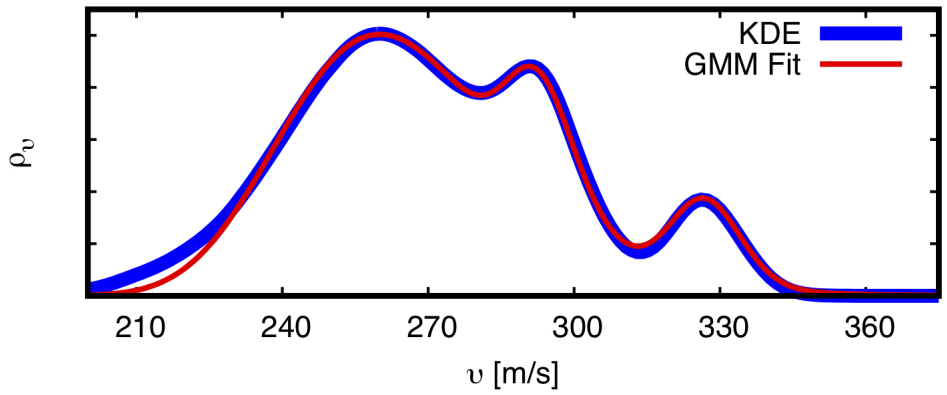


Fig. 7: A general travel time model includes three components corresponding to the tropospheric, stratospheric, and thermospheric waveguides.

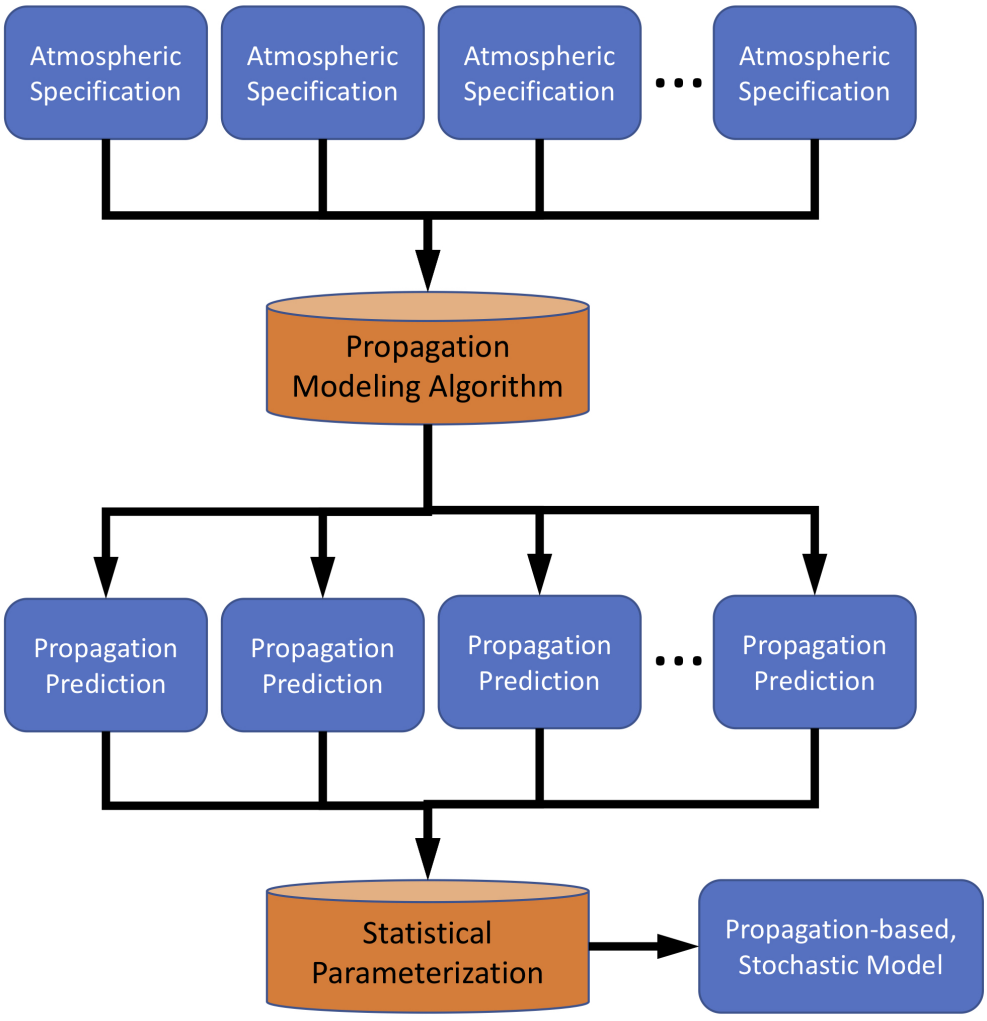


Fig. 8: Stochastic propagation models are constructing using a suite of possible atmospheric states, propagation modeling applied to each, and a statistical model describing the variability in the resulting set of predicted effects

### Path Geometry Models (PGMs)

- Path geometry models describing the arrival location, travel time, direction of arrival (back azimuth, inclination angle) can be computed using geometric modeling simulations such as those in the InfraGA/GeoAc package.
- Ray tracing simulations can be run for all atmospheric specification files in a given directory using the `stochprop.propagation.run_infraga` method by specifying the directory, output file, geometry (3D Cartesian or spherical), CPU count (if the infraGA/GeoAc OpenMPI methods are installed), azimuth and inclination angle ranges, and source location
  - Note: the source location is primarily used in the spherical coordinate option to specify the latitude and longitude of the source, but should also contain the ground elevation for the simulation runs as the third element (e.g., for a source at 30 degrees latitude, 100 degrees longitude, and a ground elevation of 1 km, specify `src_loc=(0.0, 0.0, 1.0)` or `src_loc=(30.0, 100.0, 1.0)` for the `geom="3d"` or `geom="sph"` options, respectively).

```
from stochprop import propagation

propagation.run_infraga("samples/winter/example-winter", "prop/winter/
↪example-winter.arrivals.dat", cpu_cnt=12, geom="sph", inclinations=[5.0,
↪ 45.0, 1.5], azimuths=azimuths, src_loc=src_loc)
```

- The resulting infraGA/GeoAc arrival files are concatenated into a single arrivals file and can be ingested to build a path geometry model by once again specifying the geometry and source location.

```
pgm = propagation.PathGeometryModel()
pgm.build("prop/winter/example-winter.arrivals.dat", "prop/winter/example-
↪winter.pgm", geom="sph", src_loc=src_loc)
```

- The path geometry model can later be loaded into a `stochprop.propagation.PathGeometryModel` instance and visualized to investigate the propagation statistics.

```
pgm.load("prop/winter/example-winter.pgm")
pgm.display(file_id="prop/winter/example-winter", subtitle="winter")
```

- The path geometry models constructed here can be utilized in the InfraPy Bayesian Infrasonic Source Localization (BISL) analysis by specifying them as the `path_geo_model` for that analysis.

```
from infrapy.location import bisl

det_list = lklhds.json_to_detection_list('data/detection_set2.json')
result, pdf = bisl.run(det_list, path_geo_model=pgm)
```

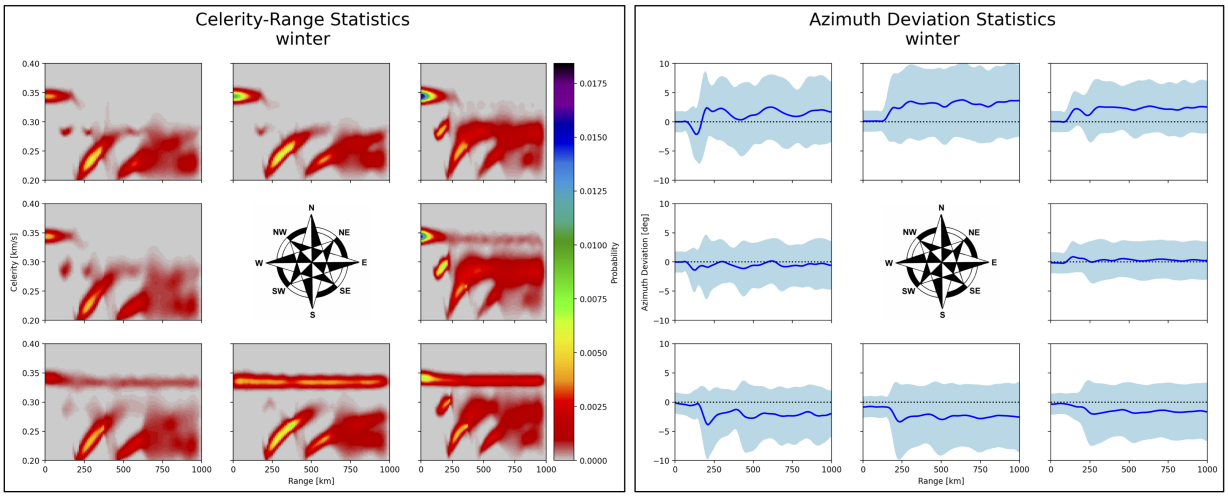


Fig. 9: Stochastic propagation-based path geometry model examples for a winter shows the expected stratospheric waveguide for propagation to the east and azimuth deviations to the north and south due to the strong stratospheric cross winds.

### Transmission Loss Models (TLMs)

- Analysis of source characteristics includes estimation of the power of the acoustic signal at some reference distance from the (typically) complex source mechanism
- Such analysis using regional signals requires a propagation model that relates the energy losses along the path, termed the transmission loss and in the case of infrasonic analysis, several methods are available in the NCPAprop software suite from the University of Mississippi
- The NCPAprop modal analysis using the effective sound speed, `modess`, can be accessed from `stochprop.propagation.run_modess` to compute transmission loss predictions for all atmospheric specifications in a directory in a similar fashion to the methods above for infraGA/GeoAc.

```
from stochprop import propagation

f_min, f_max, f_cnt = 0.01, 1.0, 10
f_vals = np.logspace(np.log10(f_min), np.log10(f_max), f_cnt)

for fn in f_vals:
    propagation.run_modess("samples/winter/example-winter", "prop/winter/
↪example-winter", azimuths=azimuths, freq=fn)
```

- Each run of this method produces a pair of output files, `prop/winter/example-winter_0.100Hz.nm` and `prop/winter/example-winter_0.100Hz.lossless.nm` that contain the predicted transmission loss with and without thermo-viscous absorption losses.
- The transmission loss predictions are loaded in frequency by frequency and statistics for transmission as a function of propagation range and azimuth are constructed and written into specified files,

```
for fn in f_vals:
    tlm = propagation.TLossModel()
```

(continues on next page)

(continued from previous page)

```
    tlm.build("prop/winter/example-winter" + "_%.3f" %fn + ".nm", "prop/
↪winter/example-winter" + "_%.3f" %fn + ".tlm")
```

– The transmission loss model can later be loaded into a `stochprop.propagation.TLossModel` instance and visualized to investigate the propagation statistics similarly to the path geometry models.

```
tlm.load("prop/winter/example-winter_0.359Hz.tlm")
tlm.display(file_id=("prop/winter/example-winter_0.359Hz), title=(
↪"Transmission Loss Statistics" + '\n' + "winter, 0.359 Hz"))
```

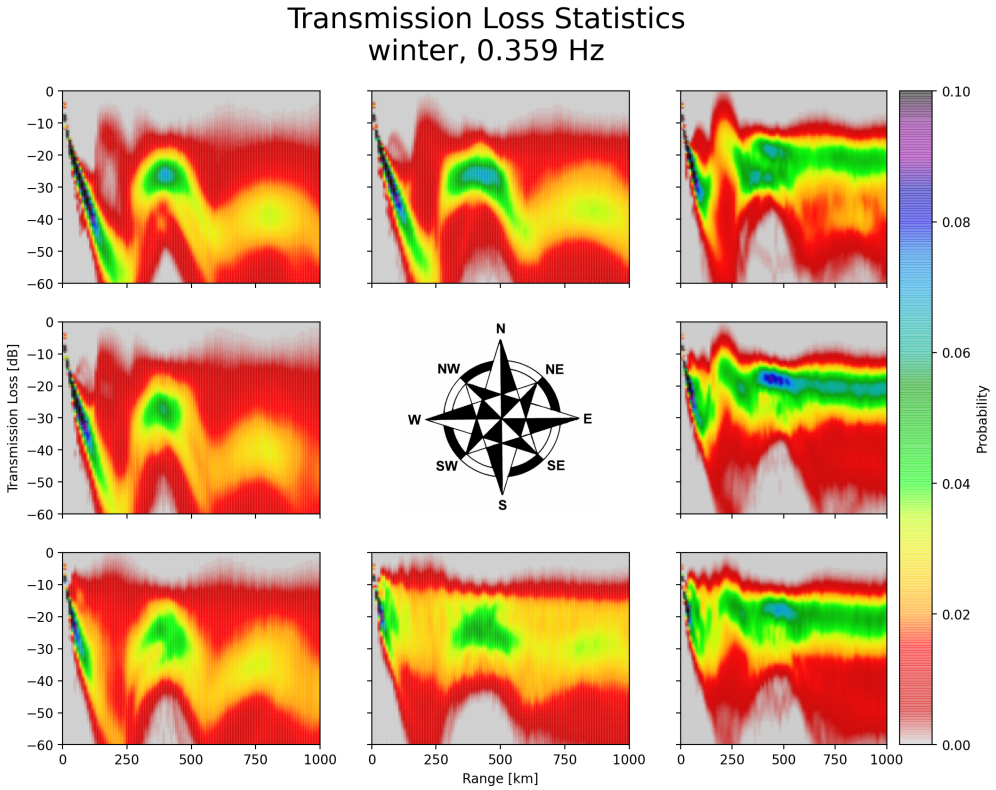


Fig. 10: Transmission loss statistics used for source characterization can be constructed using analysis of NCPAprop normal mode algorithm output.

– The transmission loss models constructed in `stochprop` can be utilized in the InfraPy Spectral Yield Estimation (SpYE) algorithm by specifying a set of models and their associated frequencies (see InfraPy example for detection and waveform data setup),

```
from infrapy.characterization import spye

# Define detection list, signal-minus-signal spectra,
# source location, and analysis frequency band

tlms = [0] * 2
tlms[0] = list(f_vals)
tlms[1] = [0] * f_cnt
```

(continues on next page)

(continued from previous page)

```
for n in range(f_cnt):
    tlms[1][n] = propagation.TLossModel()
    tlms[1][n].load("prop/winter/example-winter_" + "%.3f" % models[0][n]␣
↪+ "Hz.tlm")

yld_vals, yld_pdf, conf_bnds = spye.run(det_list, smn_specs, src_loc,␣
↪freq_band, tlms)
```

## 1.4 API

### 1.4.1 Empirical Orthogonal Function Analysis

stochprop.eofs.**build_atmo_matrix**(*path*, *pattern='*.met'*, *skiprows=0*, *ref_alts=None*)

Read in a list of atmosphere files from the path location matching a specified pattern for continued analysis.

**Parameters**

**path: string** Path to the profiles to be loaded

**pattern: string** Pattern defining the list of profiles in the path

**skiprows: int** Number of header rows in the profiles

**ref_alts: 1darray** Reference altitudes if comparison is needed

**Returns**

**A: 2darray** Atmosphere array of size M x (5 * N) for M atmospheres where each atmosphere samples N altitudes

stochprop.eofs.**build_cdf**(*pdf*, *lims*, *pnts=250*)

Compute the cumulative distribution of a pdf within specified limits

**Parameters**

**pdf: function** Probability distribution function (PDF) for a single variable

**lims: 1darray** Iterable containing lower and upper bound for integration

**pnts: int** Number of points to consider in defining the cumulative distribution

**Returns**

**cfd: interp1d** Interpolated results for the cdf

stochprop.eofs.**compute_coeffs**(*A*, *alts*, *eofs_path*, *output_path*, *eof_cnt=100*, *pool=None*)

Compute the EOF coefficients for a suite of atmospheres and store the coefficient values.

**Parameters**

**A: 2darray** Suite of atmosphere specifications from build_atmo_matrix

**alts: 1darray** Altitudes at which the atmosphere is sampled from build_atmo_matrix

**eofs_path: string** Path to the .eof results from compute_eofs

**output_path: string** Path where output will be stored

**eof_cnt: int** Number of EOFs to consider in computing coefficients

**pool: pathos.multiprocessing.ProcessingPool** Multiprocessing pool for accelerating calculations

**Returns**

**coeffs: 2darray** Array containing coefficient values of size prof_cnt by eof_cnt. Result is also written to file.

stochprop.eofs.**compute_eofs**(*A*, *alts*, *output_path*, *eof_cnt=100*)

Computes the singular value decomposition (SVD) of an atmosphere set read into an array by stochprop.eofs.build_atmo_matrix() and saves the basis functions (empirical orthogonal functions) and singular values to file

**Parameters**

**A: 2darray** Suite of atmosphere specifications from build_atmo_matrix

**alts: 1darray** Altitudes at which the atmosphere is sampled from build_atmo_matrix

**output_path: string** Path to output the SVD results

**eof_cnt: int** Number of basic functions to save

stochprop.eofs.**compute_overlap**(*coeffs*, *eof_cnt=100*)

Compute the overlap of EOF coefficient distributions

**Parameters**

**coeffs: list of 2darrays**

**List of 2darrays containing coefficients to consider** overlap in PDF of values

**eof_cnt: int** Number of EOFs to compute

**Returns**

**overlap: 3darray** Array containing overlap values of size coeff_cnt by coeff_cnt by eof_cnt

stochprop.eofs.**compute_seasonality**(*overlap_file*, *eofs_path*, *file_id=None*)

Compute the overlap of EOF coefficients to identify seasonality

**Parameters**

**overlap_file: string** Path and name of file containing results of stochprop.eofs.compute_overlap

**eofs_path: string** Path to the .eof results from compute_eofs

**file_id: string** Path and ID to save the dendrogram result of the overlap analysis

stochprop.eofs.**define_coeff_limits**(*coeff_vals*)

Compute upper and lower bounds for coefficient values

**Parameters**

**coeff_vals: 2darrays** Coefficients computed with stochprop.eofs.compute_coeffs

**Returns**

**lims: 1darray** Lower and upper bounds of coefficient value distribution

stochprop.eofs.**draw_from_pdf**(*pdf*, *lims*, *cdf=None*, *size=1*)

Sample a number of values from a probability distribution function (pdf) with specified limits

**Parameters**

**pdf: function** Probability distribution function (PDF) for a single variable

**lims: 1darray** Iterable containing lower and upper bound for integration

**cdf: function** Cumulative distribution function (CDF) from stochprop.eofs.build_cfd

**size: int** Number of samples to generate

**Returns**

**samples: 1darray** Sampled values from the PDF

stochprop.eofs.**fit_atmo**(*prof_path*, *eofs_path*, *output_path*, *eof_cnt=100*)

Compute a given number of EOF coefficients to fit a given atmophere specification using the basic functions. Write the resulting approximated atmospheric specification to file.

**Parameters**

**prof_path: string** Path and name of the specification to be fit

**eofs_path: string** Path to the .eof results from compute_eofs

**output_path: string** Path where output will be stored

**eof_cnt: int** Number of EOFs to use in building approximate specification

stochprop.eofs.**maximum_likelihood_profile**(*coeffs*, *eofs_path*, *output_path*, *eof_cnt=100*)

Use coefficient distributions for a set of empirical orthogonal basis functions to compute the maximum likelihood specification

**Parameters**

**coeffs: 2darrays** Coefficients computed with stochprop.eofs.compute_coeffs

**eofs_path: string** Path to the .eof results from compute_eofs

**output_path: string** Path where output will be stored

**eof_cnt: int** Number of EOFs to use in building sampled specifications

stochprop.eofs.**perturb_atmo**(*prof_path*, *eofs_path*, *output_path*, *uncertainty=10.0*, *eof_max=100*, *eof_cnt=50*, *sample_cnt=1*, *alt_wt_pow=2.0*, *sing_val_wt_pow=0.25*)

Use EOFs to perturb a specified profile using a given scale

**Parameters**

**prof_path: string** Path and name of the specification to be fit

**eofs_path: string** Path to the .eof results from compute_eofs

**output_path: string** Path where output will be stored

**uncertainty: float** Estimate of uncertainty in wind speeds; 95% confidence is set to this value

**eof_max: int** Higher numbered EOF to sample

**eof_cnt: int** Number of EOFs to sample in the perturbation (can be less than eof_max)

**sample_cnt: int** Number of perturbed atmospheric samples to generate

**alt_wt_pow: float** Power raising relative mean altitude value in weighting

**sing_val_wt_pow: float** Power raising relative singular value in weighting

stochprop.eofs.**profiles_qc**(*path*, *pattern='*.met'*, *skiprows=0*)

Runs a quality control (QC) check on profiles in the path matching the pattern. It can optionally plot the bad profiles. If it finds any, it makes a new direcotry in the path location called "bad_profs" and moves those profiles into the directory for you to check

**Parameters**

**path: string** Path to the profiles to be QC'd

**pattern: string** Pattern defining the list of profiles in the path

**skiprows: int** Number of header rows in the profiles

stochprop.eofs.**sample_atmo**(*coeffs*, *eofs_path*, *output_path*, *eof_cnt=100*, *prof_cnt=250*, *output_mean=False*)

Generate atmosphere states using coefficient distributions for a set of empirical orthogonal basis functions

**Parameters**

**coeffs: 2darrays** Coefficients computed with stochprop.eofs.compute_coeffs

**eofs_path: string** Path to the .eof results from compute_eofs

**output_path: string** Path where output will be stored

**eof_cnt: int** Number of EOFs to use in building sampled specifications

**prof_cnt: int** Number of atmospheric specification samples to generate

**output_mean: bool** Flag to output the mean profile from the samples generated

### 1.4.2 Propagation Statistics

**class** stochprop.propagation.**PathGeometryModel**

Bases: object

Propagation path geometry statistics computed using ray tracing analysis on a suite of specifications includes celerity-range and azimuth deviation/scatter statistics

#### Methods

| | |
|---|---|
| *build*(arrivals_file, output_file[, ...]) | Construct propagation statistics from a ray tracing arrival file (concatenated from multiple runs most likely) and output a path geometry model |
| *display*([file_id, subtitle, show_colorbar]) | Display the propagation geometry statistics |
| *eval_az_dev_mn*(rng, az) | Evaluate the mean back azimuth deviation at a given range and propagation azimuth |
| *eval_az_dev_std*(rng, az) | Evaluate the standard deviation of the back azimuth at a given range and propagation azimuth |
| *eval_rcel_gmm*(rng, rcel, az) | Evaluate reciprocal celerity Gaussian Mixture Model (GMM) at specified range, reciprocal celerity, and azimuth |
| *load*(model_file[, smooth]) | Load a path geometry model file for use |

**build**(*arrivals_file*, *output_file*, *show_fits=False*, *rng_width=50.0*, *rng_spacing=10.0*, *geom='3d'*, *src_loc=[0.0, 0.0, 0.0]*, *min_turning_ht=0.0*, *az_bin_cnt=16*, *az_bin_wdth=30.0*)

Construct propagation statistics from a ray tracing arrival file (concatenated from multiple runs most likely) and output a path geometry model

**Parameters**

**arrivals_file: string** Path to file containing infraGA/GeoAc arrival information

**output_file: string** Path to file where results will be saved

**show_fits: boolean** Option ot visualize model construction (for QC purposes)

**rng_width: float** Range bin width in kilometers

**rng_spacing: float** Spacing between range bins in kilometers

**geom: string** Geometry used in infraGA/GeoAc simulation. Options are "3d" and "sph"

**src_loc: iterable** [x, y, z] or [lat, lon, elev] location of the source used in infraGA/GeoAc simulations. Note: '3d' simulations assume source at origin.

**min_turning_ht: float** Minimum turning height used to filter out boundary layer paths if not of interest

**az_bin_cnt: int** Number of azimuth bins to use in analysis

**az_bin_width: float** Azimuth bin width in degrees for analysis

**display**(*file_id=None*, *subtitle=None*, *show_colorbar=True*)
Display the propagation geometry statistics

**Parameters**

**file_id: string** File prefix to save visualization

**subtitle: string** Subtitle used in figures

**eval_az_dev_mn**(*rng*, *az*)
Evaluate the mean back azimuth deviation at a given range and propagation azimuth

**Parameters**

**rng: float** Range from source

**az: float** Propagation azimuth (relative to North)

**Returns**

**bias: float** Predicted bias in the arrival back azimuth at specified arrival range and azimuth

**eval_az_dev_std**(*rng*, *az*)
Evaluate the standard deviation of the back azimuth at a given range and propagation azimuth

**Parameters**

**rng: float** Range from source

**az: float** Propagation azimuth (relative to North)

**Returns**

**stdev: float** Standard deviation of arrival back azimuths at specified range and azimuth

**eval_rcel_gmm**(*rng*, *rcel*, *az*)
Evaluate reciprocal celerity Gaussian Mixture Model (GMM) at specified range, reciprocal celerity, and azimuth

**Parameters**

**rng: float** Range from source

**rcel: float** Reciprocal celerity (travel time divided by propagation range)

**az: float** Propagation azimuth (relative to North)

**Returns**

**pdf: float** Probability of observing an infrasonic arrival with specified celerity at specified range and azimuth

**load**(*model_file*, *smooth=False*)
Load a path geometry model file for use

**Parameters**

**model_file: string** Path to PGM file constructed using stochprop.propagation.PathGeometryModel.build()

**smooth: boolean** Option to use scipy.signal.savgol_filter to smooth discrete GMM parameters along range

**class** stochprop.propagation.**TLossModel**
Bases: object

### Methods

| | |
|---|---|
| *build*(tloss_file, output_file[, show_fits, ...]) | Construct propagation statistics from a NCPAprop modess or pape file (concatenated from multiple runs most likely) and output a transmission loss model |
| *display*([file_id, title, show_colorbar]) | Display the transmission loss statistics |
| *eval*(rng, tloss, az) | Evaluate TLoss model at specified range, transmission loss, and azimuth |
| *load*(model_file) | Load a transmission loss file for use |

**build**(*tloss_file*, *output_file*, *show_fits=False*, *use_coh=False*, *az_bin_cnt=16*, *az_bin_wdth=30.0*)
Construct propagation statistics from a NCPAprop modess or pape file (concatenated from multiple runs most likely) and output a transmission loss model

**Parameters**

**tloss_file: string** Path to file containing NCPAprop transmission loss information

**output_file: string** Path to file where results will be saved

**show_fits: boolean** Option ot visualize model construction (for QC purposes)

**use_coh: boolean** Option to use coherent transmission loss

**az_bin_cnt: int** Number of azimuth bins to use in analysis

**az_bin_width: float** Azimuth bin width in degrees for analysis

**display**(*file_id=None*, *title='Transmission Loss Statistics'*, *show_colorbar=True*)
Display the transmission loss statistics

**Parameters**

**file_id: string** File prefix to save visualization

**subtitle: string** Subtitle used in figures

**eval**(*rng*, *tloss*, *az*)
Evaluate TLoss model at specified range, transmission loss, and azimuth

**Parameters**

**rng: float** Range from source

**tloss: float** Transmission loss

**az: float** Propagation azimuth (relative to North)

**Returns**

**pdf: float** Probability of observing an infrasonic arrival with specified transmission loss at specified range and azimuth

**load**(*model_file*)
Load a transmission loss file for use

**Parameters**

**model_file: string** Path to TLoss file constructed using stochprop.propagation.TLossModel.build()

stochprop.propagation.**find_azimuth_bin**(*az*, *bin_cnt=16*)
Identify the azimuth bin index given some specified number of bins

**Parameters**

**az: float** Azimuth in degrees

**bin_cnt: int** Number of bins used in analysis

**Returns**

**index: int** Index of azimuth bin

stochprop.propagation.**run_infraga**(*profs_path*, *results_file*, *pattern='*.met'*, *cpu_cnt=None*, *geom='3d'*, *bounces=25*, *inclinations=[1.0, 60.0, 1.0]*, *azimuths=[- 180.0, 180.0, 3.0]*, *freq=0.1*, *z_grnd=0.0*, *rng_max=1000.0*, *src_loc=[0.0, 0.0, 0.0]*, *infraga_path=''*, *clean_up=False*)
Run the infraga -prop algorithm to compute path geometry statistics for BISL using a suite of specifications and combining results into single file

**Parameters**

**profs_path: string** Path to atmospheric specification files

**results_file: string** Path and name of file where results will be written

**pattern: string** Pattern identifying atmospheric specification within profs_path location

**cpu_cnt: int** Number of threads to use in OpenMPI implementation. None runs non-OpenMPI version of infraga

**geom: string** Defines geometry of the infraga simulations (3d" or "sph")

**bounces: int** Maximum number of ground reflections to consider in ray tracing

**inclinations: iterable object** Iterable of starting, ending, and step for ray launch inclination

**azimuths: iterable object** Iterable of starting, ending, and step for ray launch azimuths

**freq: float** Frequency to use for Sutherland Bass absorption calculation

**z_grnd: float** Elevation of the ground surface relative to sea level

**rng_max: float** Maximum propagation range for propagation paths

**src_loc: iterable object** The horizontal (latitude and longitude) and altitude of the source

**infraga_path: string** Location of infraGA executables

**clean_up: boolean** Flag to remove individual [..].arrival.dat files after combining

stochprop.propagation.**run_modess**(*profs_path*, *results_path*, *pattern='*.met'*, *azimuths=[- 180.0, 180.0, 3.0]*, *freq=0.1*, *z_grnd=0.0*, *rng_max=1000.0*, *ncpaprop_path=''*, *clean_up=False*)
Run the NCPAprop normal mode methods to compute transmission loss values for a suite of atmospheric specifications at a set of frequency values

**Parameters**

**profs_path: string** Path to atmospheric specification files

**results_file: string** Path and name of file where results will be written

**pattern: string** Pattern identifying atmospheric specification within profs_path location

**azimuths: iterable object** Iterable of starting, ending, and step for propagation azimuths

**freq: float** Frequency for simulation

**z_grnd: float** Elevation of the ground surface relative to sea level

**rng_max: float** Maximum propagation range for propagation paths

**clean_up: boolean** Flag to remove individual .nm files after combining

## 1.5 References and Citing Usage

The Empirical Orthogonal Function (EOF) analyses available in `stochprop` are part of ongoing joint research between infrasound scientists at Los Alamos National Laboratory (LANL) and the University of Mississippi's National Center for Physical Acoustics (NCPA) and will be summarizing in an upcoming publication:

- Waxler, R., Blom, P., & Frazier, W. G. (2020) Spatial and seasonal trends in atmospheric structure and predicted infrasonic propagation for the continental US. *Geophysical Journal International*, In Preparation

Stochastic, propagation-based models for infrasonic signal analysis were initially introduced in analysis of the Bayesian Infrasonic Source Localization (BISL) and Spectral Yield Estimation (SpYE) framworks so that usage of path geometry and transmission loss models should be cited using:

- Blom, P. S., Marcillo, O., & Arrowsmith, S. J. (2015). Improved Bayesian infrasonic source localization for regional infrasound. *Geophysical Journal International*, 203(3), 1682-1693.
- Blom, P. S., Dannemann, F. K., & Marcillo, O. E. (2018). Bayesian characterization of explosive sources using infrasonic signals. *Geophysical Journal International*, 215(1), 240-251.

See the documentation for the supporting packages (InfraGA/GeoAc, NCPAprop, InfraPy) for guidance on citing usage of those methods.

# PYTHON MODULE INDEX

## S

# INDEX