stochprop Documentation

Release 1.0

P. Blom

Jul 29, 2020

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 Authorship & License Info

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

License info here…

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

- Discussion of stochastic propagation analysis approach. . .

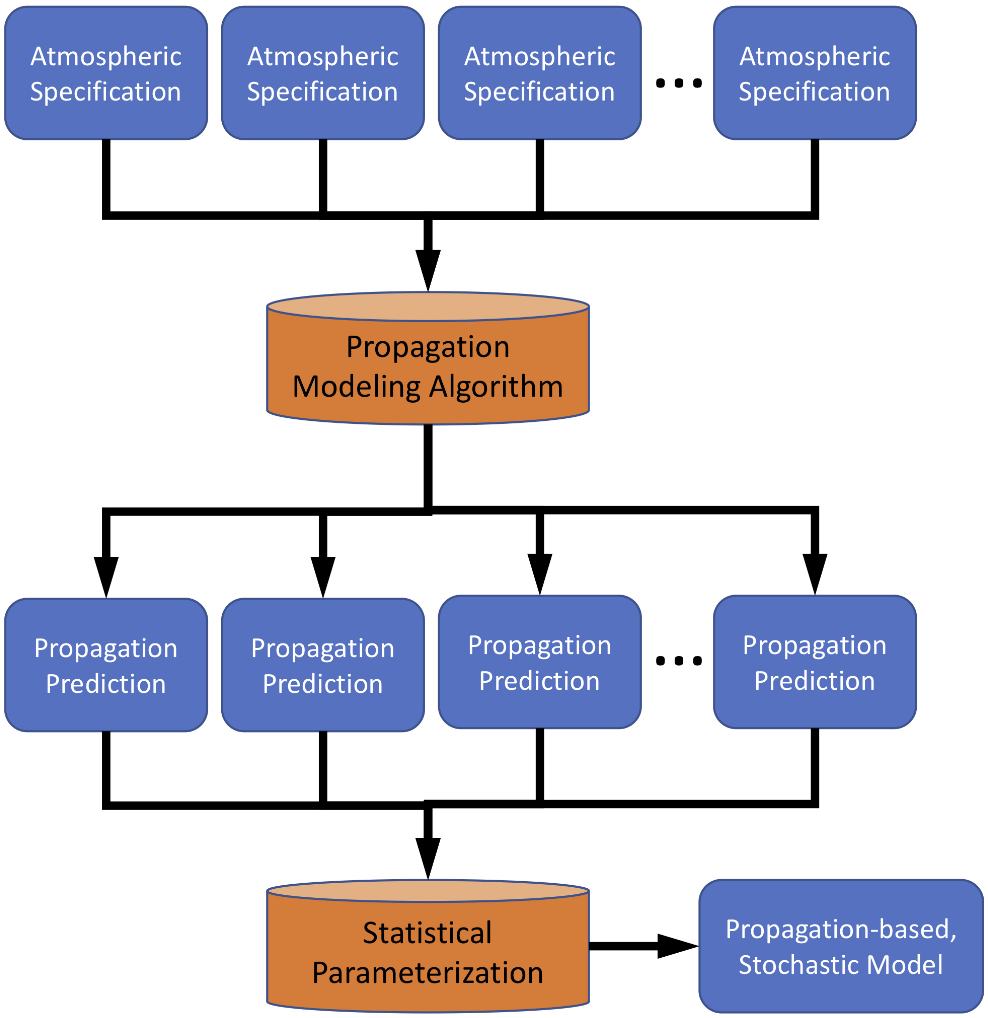

Fig. 1: Figure caption

- More discussion. . .

1.3.1 Empirical Orthogonal Function Analysis

Empirical Orthogonal Function Analysis

- Discussion of empirical orthogonal function expansions and use in quantifying atmospheric variability. . .

1.3.2 Atmospheric Sampling, Fitting, and Perturbation

Atmospheric Sampling, Fitting, and Perturbation

- Discussion of sampling, fitting, and perturbing atmospheric specifications…

1.3.3 Propagation Statistics

Propagation Statistics

- Discussion of building propagation statistics for path geometry and transmission loss…

1.3.4 Section Links

Empirical Orthogonal Function Analysis

- Atmospheric specifications are available through a number of repositories, but the most up to date is maintained by University of Mississippi's National Center for Physical Acoustics (NCPA) at http://g2s.ncpa.olemiss.edu
- Pull atmospheric specifications…

Define Run Parameters

- Discussion…

```
prof_dir = "dir/of/g2s/"
prof_prefix = "g2stxt_"
year_lims = [2010, 2016]
run_id = "example"
eof_cnt = 50
```

Load Atmosphere Specifications and Building EOFs

- Discussion

Compute Coefficients and Determine Seasonality

- Discussion…

Generate Samples from a Coefficient Set

- Discussion…

Atmospheric Sampling, Fitting, and Perturbation

- Overview of building propagation statistics and their use in BISL and SpYE

Fitting an Atmospheric Specification using EOFs

- Stuff…
- Test math input:

$$y = mx + b$$

- Test inline math input, $y = mx + b$, and then it stops?

Sampling Specifications using EOF Coefficient Distributions

- Stuff…

Perturbing Specifications to Account for Uncertainty

- Stuff…

Propagation Statistics

- Overview of building propagation statistics and their use in BISL and SpYE

Path Geometry Models (PGMs)

- Stuff…

Transmission Loss Models (TLMs)

- Stuff…

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_svd

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_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_svd

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

stochprop.eofs.**compute_svd**(*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.**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_svd

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_svd

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_svd

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_svd

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])	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*)

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

display(*file_id=None, subtitle=None*)

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])	Display the transmission loss statistics
eval(rng, tloss, az)	Evaluate TLoss model at specified range, transmission loss, and azimuth

Continued on next page

Table 2 – continued from previous page

load(model_file)	Load a transmission loss file for use

build(*tloss_file*, *output_file*, *show_fits=False*, *use_coh=False*)

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

display(*file_id=None*, *title='Transmission Loss Statistics'*)

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=''*)

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

stochprop.propagation.**run_modess**(*profs_path, results_path, pattern='*.met', cpu_cnt=None, azimuths=[-180.0, 180.0, 3.0], freq=0.1, z_grnd=0.0, rng_max=1000.0, ncpaprop_path=''*)

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

PYTHON MODULE INDEX

S