HyDe Documentation

Release 0.4.1a

Paul Blischak and Laura Kubatko

Jul 12, 2019

Contents

HyDe is a software package that detects hybridization in phylogenomic data sets using phylogenetic invariants. The primary interface for HyDe is a Python module called `phyde` (**P**ythonic **Hy**bridization **De**tection). `phyde` provides a suite of tools for performing hypothesis tests on triples of taxa to detect hybridization. To ensure that the necessary dependencies for `phyde` are available, we suggest using a Python distribution such as Miniconda.

To facilitate analyses using the Python module, three scripts are provided to conduct hybridization detection analyses directly from the command line:

- `run_hyde.py`: runs a standard hybridization detection analysis on all triples in all directions. Results are also filtered based on whether there is significant evidence for hybridization.
- `individual_hyde.py`: tests each individual within a putative hybrid population using a list of specified triples specified.
- `bootstrap_hyde.py`: conducts bootstrap resampling of the individuals within the putative hybrid lineages for each specified triple.

These last two scripts need to be given a three column table of triples (P1, Hybrid, P2) that you wish to test:

```
sp1 sp2 sp3
sp1 sp3 sp4
sp3 sp4 sp5
.
.
.
```

You can also use a results file from a previous analysis as a triples file. For example, you can use the filtered results from the `run_hyde.py` script so that you only run analyses on triples that have significant levels of hybridization. If you only have a few hypotheses that you want to test, then you can also pass a triples file to `run_hyde.py` and it will only test those hypotheses rather than testing everything.

Multithreaded versions of these scripts are also available (`run_hyde_mp.py`, `individual_hyde_mp.py`, and `bootstrap_hyde_mp.py`). Make sure you have the `multiprocess` module installed before you use them: `pip install multiprocess`.

CHAPTER 1

Getting help

If you have questions about running HyDe, please feel free to use the **gitter chatroom** to get help:

If you have a problem while running HyDe and you think it may be a bug, please consider filing an issue on GitHub:

CHAPTER 2

Features

- Conduct hypothesis tests using multiple individuals per population.
- Test each individual within a putative hybrid lineage to assess if hybridization is uniform.
- Test all possible triples of taxa and process results from within Python.
- Bootstrap individuals within taxa to assess patterns of hybrid speciation vs. introgression.
- Visualize the distributions of various quantities (Test Statistic, Hybridization Parameter, D-Statistic) using bootstrap replicates.
- Calculate the D-Statistic (ABBA-BABA) using site pattern counts returned during a hypothesis test.

CHAPTER 3

Documentation

3.1 Installation

HyDe requires Python v2.7 or v3.6, a C++ compiler, and several Python modules (listed below).

3.1.1 Miniconda

We recommend using a Python distribution such as Miniconda to make it easier to manage modules and environments.

```
# Get Miniconda for your operating system (Mac or Linux)
# Answer yes to the questions the Installer asks
# These commands will download Python 2.7 for Mac OSX
curl -O https://repo.continuum.io/miniconda/Miniconda2-latest-MacOSX-x86_64.sh
bash Miniconda2-latest-MacOSX-x86_64.sh
```

3.1.2 Required Python packages

With Miniconda installed, we can use `pip` (or `conda`) to install all of the Python modules that HyDe requires (Cython, Numpy, Matplotlib, Seaborn, and Multiprocess).

```
# Install packages with pip
pip install cython numpy matplotlib seaborn multiprocess
```

3.1.3 Installing HyDe

There are two ways that HyDe can be installed once you have all of the required Python modules: (1) install from PyPI using pip or (2) clone from GitHub and install manually.

To install from PyPI, all we need to do is type the following command:

```
#installing from PyPI
pip install phyde
```

Next, we'll take a look at how to install HyDe by cloning it from GitHub. The commands below take you through every step to accomplish this:

```
# Clone the HyDe repo from GitHub and cd into the repo
git clone https://github.com/pblischak/HyDe.git
cd HyDe/

# Builds and installs phyde module
python setup.py install

# Test installation
make test
```

Step-by-step

After cloning HyDe from GitHub and moving into the main HyDe directory, the next two steps accomplish the following tasks:

1. `python setup.py install`: this will build and install the HyDe Python module, including the compilation of any Cython files (C++ compiler required).
2. `make test`: this will test the installation by running a series of commands designed to check that the installation was completed successfully. The main tests are in the `test.py` script in the `test/` folder.

3.2 HyDe Data Files

The main data files for running HyDe are (1) the DNA sequences and (2) the map of sampled individuals to their respective taxa/populations. Examples of these files can be found in the `test/` and `examples/` directories within the main HyDe folder. Below we describe each file in more detail. If you installed HyDe from PyPI and do not have a clone of the GitHub repository, these files can be viewed here.

3.2.1 Sequence Data

The DNA sequence data should be in sequential Phylip format with individual names and sequence data all on one line and separated by a tab. In previous versions of HyDe (<= 0.3.1), the header information that is typically present in a Phylip file (# individuals and # sites) was not included. We maintain this backwards compatibility. Individual names do not have length restrictions (within reason) and do not all need to be the same number of characters.

Example: `data.txt`

```
<NumInd>  <NumSites>
Ind1  AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
Ind2  AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
Ind3  AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
.
.
.
Ind23 AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
Ind24 AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
```

(continues on next page)

(continued from previous page)

```
Ind25 AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
.
.
.
Ind(N-2)  AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
Ind(N-1)  AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
IndN      AGATTGAGCTAGCAGACGTGACAGACAGATGACAGTGACGA...
```

3.2.2 Taxon Map

The taxon map file is organized similarly to the sequence data file with one individual per row and a tab separating the individual's name from the name of the taxon/population it belongs to. The individuals should be in the same order as the DNA sequence data file with all individuals in a particular taxon grouped together sequentially.

Example: `map.txt`

```
Ind1  Pop1
Ind2  Pop1
Ind3  Pop1
.
.
.
Ind23 Pop3
Ind24 Pop3
Ind25 Pop3
.
.
.
Ind(N-2)  PopM
Ind(N-1)  PopM
IndN  PopM
```

3.2.3 Triples file

Each of the Python scripts that are part of HyDe take a file of triples that are to be tested for hybridization. The format of this triples file is a three column table where each row is a triple to be tested. Column one of the table is parent one, column two is the putative hybrid, and column three is parent two. You can name the populations anything you like as long as the name doesn't have spaces.

```
Pop1  Pop2  Pop3
Pop1  Pop3  Pop4
Pop2  Pop4  Pop5
.
.
.
Pop4  Pop5  PopM
```

We have also written the scripts to take previous output files from HyDe as input. It does this by ignoring the first-row header and using the triples specified in the rows that follow.

3.3 Analyzing Data with HyDe

Below we provide details on the three main scripts that are used to analyze data with HyDe. The multithreaded versions of these scripts behave in the exact same way, but have an added `--threads` (`-j`) option to specify how many threads to use.

We will be using the `data.txt` and `map.txt` files in the `test/` folder from the GitHub repo for HyDe. If you don't have a clone of the repo, you can download the files using the following commands:

```
curl -O https://raw.githubusercontent.com/pblischak/HyDe/master/test/data.txt
curl -O https://raw.githubusercontent.com/pblischak/HyDe/master/test/map.txt
```

Note: Recommended workflow:

When analyzing data with HyDe, we recomend the following workflow.

1. Use the `run_hyde.py` script to analyze all possible triples. This will produce two output files, one with all results and one with only significant results (see **Note** below on filtered results).
2. Next, if you want to see if certain individuals are hybrids, run the `individual_hyde.py` script and use the filtered results file output from the previous step as the triples file.
3. For the `bootstrap_hyde.py` script, we recommend using it when you don't have enough data (and therefore not enough power) to detect hybridization within a single individual. This will depend on how much hybridization has occurred, but it is typically difficult to detect hybridization with HyDe using less than 10,000 sites. Otherwise, we always recommend using the `individual_hyde.py` script.

3.3.1 Command Line Scripts

`run_hyde.py`

To run HyDe from the command line, we have provided a Python script (`run_hyde.py`) that will test all triples in all directions ("full" analysis). It can also be used to test a predefined set of hypothesis tests using a triples file. The arguments for the script are passed using command line flags, all of which can be viewed by typing `run_hyde.py -h`. Typing the name of the script with no arguments will print out a docstring with additional details.

```
# Run a full hybridization detection analysis
run_hyde.py -i data.txt -m map.txt -o out -n 16 -t 4 -s 50000
```

The results will be written to file with a prefix that can be supplied using the `--prefix` flag (`<prefix>-out.txt`; the default is 'hyde').

Note: Filtered results:

We also write a file (`<prefix>-out-filtered.txt`) that filters the results from the hybridization detection analysis to only include significant results with sensible values of γ ($0 < \gamma < 1$). Some values of γ in the original results file may be nonsensical because they will be either negative or greater than 1. However, γ does not have any theoretical limits with regard to the hypothesis test in HyDe: it can range from $-\infty$ to ∞. The reason that these values occur and may give a significant p-value is because they are testing a hypothesis that involves a hybrid but in the wrong direction (i.e., a hybrid is tested as one of the parental species). Testing all possible directions that hybridization can occur for three taxa is typically what causes these types of results to happen.

---

individual_hyde.py

Triples file required.

If you want to test for hybridization at the individual level within the populations that have significant levels of hybridization, you can use the filtered results file from an analysis with `run_hyde.py` as the input triples file for the `individual_hyde.py` script. The options for `individual_hyde.py` are the same as for `run_hyde.py`, and typing the name of the script without arguments will again print out a docstring with more details.

```
# Test all individuals for the hybrid populations specified
# in the file hyde-filtered-out.txt
individual_hyde.py -i data.txt -m map.txt -tr hyde-out-filtered.txt -o out
                   -n 16 -t 4 -s 50000
```

The results of the individual level tests will be written to a file called `<prefix>-ind.txt`, where `<prefix>` can be set using the `--prefix` flag (default='hyde').

bootstrap_hyde.py

Triples file required.

The `bootstrap_hyde.py` conducts bootstrap resampling of individuals within hybrid populations. The arguments are the same as the `individual_hyde.py` script with the addition of specifying the number of bootstrap replicates (`--reps=<#reps>`; default=100).

```
# Bootstrap resample individuals within hybrid populations
# specified in hyde-out-filtered.txt
bootstrap_hyde.py -i data.txt -m map.txt -tr hyde-out-filtered.txt -o out
                  -n 16 -t 4 -s 50000 --reps 200
```

The output file is named `hyde-boot.txt`, but again this can be changed using the `--prefix` argument. Bootstrap replicates for each triple are separated by a line with four pound symbols and a newline ("####\n"; match this pattern to split results).

3.3.2 Python Interface

Reading in Data

Reading data files (DNA sequences and taxon maps) into Python is done using the `HydeData` class. Making a new variable using the class requires passing six arguments to the constructor (in order): (1) the name of the data file, (2) the name of the map file, (3) the name of the outgroup taxon, (4) the number of individuals, (5) the number of taxa, and (6) the number of sites. Names should be provided in quotes. The code below will read in the `data.txt` and `map.txt` files for us to analyze.

```
# Import the phyde module. For simplicity, we always import it as `hd`
import phyde as hd

# Read in the data using the HydeData class
dat = hd.HydeData("data.txt", "map.txt", "out", 16, 4, 50000)
```

Conducting Individual Hypothesis Tests

With our data read in and stored using the variable `dat`, we can begin to run hypothesis tests using the methods provided in the `HydeData` class. The first of these is the `test_triple()` method, which will conduct a hypothesis

test at the population level for a specified triple of taxa (remember, the outgroup has already been specified when we read in the data).

```
# Using the `dat` variable from the previous code section,
# we'll run a hypothesis test on the triple (sp1, sp2, sp3).

res = dat.test_triple("sp1", "sp2", "sp3")
```

`res` is a variable that stores the results of our hypothesis test. More specifically, it is a Python dictionary. To see what it contains we can type `print res` (or `print(res)` if you have imported the `print_function` from the `__future__` module).

Using the same `HydeData` variable, we can test all of the individuals in the taxon "sp2" to see if they are all hybrids. We do this using the `test_individuals()` method.

```
# The code here should look very similar to the previous code block
# The only difference is that we are calling a different method

res_ind = dat.test_individuals("sp1", "sp2", "sp3")
```

`res_ind` stores the results of the hypothesis tests for each individual in population "sp2" (individuals "i5" through "i10"). Since the results for each test is a dictionary, the `res_ind` variable is a dictionary of dictionaries with the individual names as the keys and the results of the hypothesis test as the associated value. The code below shows a few examples of how to we can work with these dictionary results in Python.

```
# Look at results for individual i5 in res_ind
res_ind["i5"]

# To look at a specific value, say "Gamma", we need two keys: one for each nested␣
↪level of the dictionaries
res_ind["i5"]["Gamma"]

# To get all of the values of the test statistics ("Zscore"), we can use a dictionary␣
↪comprehension
zscores = {k:v["Zscore"] for k,v in res_ind.items()}
print zscores
```

The final method of note implemented by the HydeData class is the `bootstrap_triple()` function, which will randomly resample individuals in the hybrid population and run hypothesis tests for each replicate. The number of bootstrap replicates to run is specified using the `reps=#` argument.

```
res_boot = dat.bootstrap_triple("sp1", "sp2", "sp3", reps=100)
```

`res_boot` stores the results of this analysis as set of nested dictionaries. The easiest way to see its structure is to print it using `print(res_boot)`.

More detailed documentation on the HydeData class and other classes implemented in the `phyde` module can be found in the *API Reference*.

3.4 Visualization

This page is still in the works. Sorry!

3.4.1 Basic plotting

```
import phyde as hd

# Read in bootstrap replicates for plotting
boot = hd.Bootstrap("hyde-boot.txt")

# Make a density plot of the bootstrap reps for Gamma
hd.viz.density(boot, 'Gamma', 'sp1', 'sp2', 'sp3', title="Bootstrap Dist. of Gamma")
```

3.4.2 Advanced plotting

For plotting that goes beyond the basic methods implemented in `phyde`, direct interaction with the Seaborn package is required.

3.5 API Reference

3.5.1 Submodule: `core`

The `core` submodule provides the main Python interface for hybridization detection analyses and data exploration through the use of implemented classes.

File: `data.pyx`

The `data.pyx` source file is written in Cython, a superset of the Python language that allows the use of C/C++ code to speed up computationally intensive tasks (and many other things). This file is automatically translated into a C++ source file and is compiled into a shared library that can be called from Python.

class phyde.**HydeData**(*infile=None*, *mapfile=None*, *str outgroup=None*, *int nind=-1*, *int ntaxa=-1*, *int nsites=-1*, *bool quiet=False*, *bool ignore_amb_sites=False*)

Class for storing (1) a matrix of DNA bases as unsigned, 8-bit integers and (2) mapping of individuals to taxa.

Parameters

- **infile** (*str*) – the name of the input DNA sequence data file.
- **mapfile** (*str*) – the name of the individual mapping file.
- **outgroup** (*str*) – the name of the outgroup.
- **nind** (*int*) – the number of individuals.
- **ntaxa** (*int*) – the number of populations/taxa.
- **nsites** (*int*) – the number of sites.
- **quiet** (*bool*) – suppress printing output.
- **ignore_amb_sites** (*bool*) – ignore missing/ambiguous sites.

Example:

```
import phyde as hd
data = hd.HydeData("infile.txt", "mapfile.txt", "outgroup", 100, 6, 10000)
```

bootstrap_triple (*self*, *str p1*, *str hyb*, *str p2*, *int reps=100*) → dict
Performs bootstrap resampling of individuals within the specified hybrid population.

Parameters

- **p1** (*str*) – parent one.
- **hyb** (*str*) – the putative hybrid.
- **p2** (*str*) – parent two.
- **reps** (*int*) – number of boostrap replicates (default=100).

Return type dict

Example:

```
import phyde as hd
data = hd.HydeData("data.txt", "map.txt", "out", 16, 4, 50000)
res = data.bootstrap_triple("sp1", "sp2", "sp3")
```

list_triples (*self*) → list
List all possible triples based on the populations in the given map file.

Return type list

Returns a list of 3-tuples: `(p1, hyb, p2)`

resetOutgroup (*self*, *newOut*)
Reset outgroup population.

Parameters **newOut** (*str*) – Name of the new outgroup.

test_individuals (*self*, *str p1*, *str hyb*, *str p2*) → dict
Test all individuals in a given putative hybrid lineage (`hyb`).

Parameters

- **p1** (*str*) – parent one.
- **hyb** (*str*) – the putative hybrid.
- **p2** (*str*) – parent two.

Return type dict

Example:

```
import phyde as hd
data = hd.HydeData("data.txt", "map.txt", "out", 16, 4, 50000)
res = data.test_individuals("sp1", "sp2", "sp3")
```

test_triple (*self*, *str p1*, *str hyb*, *str p2*) → dict
Main method for testing a hypothesis on a specified triple. ((P1,Hyb),P2):γ and (P1,(Hyb,P2)):$1-\gamma$. It is a wrapper for the C++ based methods *_test_triple_c()*, which is not accessible from Python.

Parameters

- **p1** (*str*) – parent one.
- **hyb** (*str*) – the putative hybrid.
- **p2** (*str*) – parent two.

Return type dict

Returns a dictionary with the values for the Z-score, P-value, estimate of gamma, and all of the site pattern counts.

Example:

```
import phyde as hd
data = hd.HydeData("data.txt", "map.txt", "out", 16, 4, 50000)
res = data.test_triple("sp1", "sp2", "sp3")
```

File: result.py

class phyde.**HydeResult**(*infile*)

A class for reading in and working with results from a HyDe analysis. It is mostly a simple container for storing triples as tuples and using them as keys in a dictionary with values that are also dictionaries containing the results from the HyDe analysis.

Parameters **infile** (*str*) – name of results file.

Example:

```
import phyde as hd
res = hd.HydeResult("hyde-out.txt")
```

__call__(*attr*, *p1*, *hyb*, *p2*)

A callable method (the object can be called as a function) for accessing information in a phyde.HydeResult object.

Parameters

- **attr** (*str*) – name of hypothesis test attribute to plot (e.g., "Gamma", "Zscore", "Pvalue", etc.)
- **p1** (*str*) – parent one.
- **hyb** (*str*) – putative hybrid.
- **p2** (*str*) – parent two.

Example:

```
import phyde as hd
res = hd.HydeResult("hyde-out.txt")
res("Zscore", "sp1", "sp2", "sp3")
```

abba_baba(*p1*, *hyb*, *p2*)

Calculate Patterson's D-Statistic for the triple (p1, hyb, p2).

Parameters

- **p1** (*str*) – parent 1.
- **hyb** (*str*) – putative hybrid.
- **p2** (*str*) – parent 2.

Example:

```
import phyde as hd
res = hd.HydeResult("hyde-out.txt")
res.abba_baba("sp1", "sp2", "sp3")
```

File: bootstrap.py

class phyde.**Bootstrap**(*bootfile*)

A class representing a dictionary of species triplets with info for each bootstrap replicate stored as a dictionary containing parameter values.

Parameters **bootfile** (*str*) – name of file with bootstrapping results.

Example:

```
import phyde as hd
boot = hd.Bootstrap("hyde-boot.txt")
```

__call__(*attr*, *p1*, *hyb*, *p2*)

A callable method (the object can be called as a function) to access attributes from a bootstrapped HyDe analysis. Returned as a list.

Parameters

- **attr** (*str*) – name of hypothesis test attribute to plot (e.g., "Gamma", "Zscore", "Pvalue", etc.)
- **p1** (*str*) – parent one.
- **hyb** (*str*) – putative hybrid.
- **p2** (*str*) – parent two.

Return type list

Returns a list of the given attribute across all bootstrap replicates for the given triple `(p1, hyb, p2)`.

Example:

```
import phyde as hd
boot = hd.Bootstrap("hyde-boot.txt")
boot("Gamma", "sp1", "sp2", "sp3")
```

abba_baba(*p1*, *hyb*, *p2*)

Calculate Patterson's D-Statistic for all bootstrap replicates for the triple (p1, hyb, p2). Uses vectorization with numpy arrays.

Parameters

- **p1** (*str*) – parent 1.
- **hyb** (*str*) – putative hybrid.
- **p2** (*str*) –

Return type numpy.array

Example:

```
import phyde as hd
boot = hd.Bootstrap("hyde-boot.txt")
boot.abba_baba("sp1", "sp2", "sp3")
```

write_summary(*summary_file*)

Write a summary of the bootstrap replicates for each tested triple. The summary written to file includes the mean and standard deviation of the Zscore, P-value, and estimate of γ.

Parameters **summary_file** (*str*) – name of file.

3.5.2 Submodule: `visualize`

The `visualize` submodule uses the `matplotlib` and `seaborn` packages to provide basic plotting of results stored by the `phyde.Bootstrap` class.

File: `viz.py`

`phyde.visualize.viz.`**`density`**(*boot_obj*, *attr*, *p1*, *hyb*, *p2*, *title=''*, *xlab=''*, *ylab=''*, *shade=True*, *color='b'*)

Make a Seaborn density plot. You can turn shading off by setting `shade=False`.

Parameters

- **`boot_obj`** (`phyde.Bootstrap`) – An object of class `phyde.Bootstrap`.
- **`attr`** (*`str`*) – name of hypothesis test attribute to plot (e.g., "Gamma", "Zscore", "Pvalue", etc.)
- **`p1`** (*`str`*) – parent one.
- **`hyb`** (*`str`*) – putative hybrid.
- **`p2`** (*`str`*) – parent two.
- **`title`** (*`str`*) – plot title.
- **`xlab`** (*`str`*) – x-axis label.
- **`ylab`** (*`str`*) – y-axis label.
- **`shade`** (*`bool`*) – draw histogram.
- **`color`** (*`str`*) – plot color.

Example:

```
import phyde as hd
boot = hd.Bootstrap("hydeboot.txt")
hd.viz.density(boot, "Gamma", "sp1", "sp2", "sp3", title="Bootstrap distribution␣
↪of $\gamma$",
               xlab="$\gamma$", ylab="density")
```

`phyde.visualize.viz.`**`dist`**(*boot_obj*, *attr*, *p1*, *hyb*, *p2*, *title=''*, *xlab=''*, *ylab=''*, *hist=True*, *color='b'*)

Make a Seaborn distplot plot (density plot with histogram). Can turn histogram off by setting `hist=False`.

Parameters

- **`boot_obj`** (`phyde.Bootstrap`) – An object of class `phyde.Bootstrap`.
- **`attr`** (*`str`*) – name of hypothesis test attribute to plot (e.g., "Gamma", "Zscore", "Pvalue", etc.)
- **`p1`** (*`str`*) – parent one.
- **`hyb`** (*`str`*) – putative hybrid.
- **`p2`** (*`str`*) – parent two.
- **`title`** (*`str`*) – plot title.
- **`xlab`** (*`str`*) – x-axis label.
- **`ylab`** (*`str`*) – y-axis label.

- **hist** (*bool*) – draw histogram.
- **color** (*str*) – plot color.

Example:

```
import phyde as hd
boot = hd.Bootstrap("hydeboot.txt")
hd.viz.dist(boot, "Zscore", "sp1", "sp2", "sp3", title="Bootstrap distribution of␣
↪Z-score",
            xlab="Z-score", ylab="density")
```

phyde.visualize.viz.**violinplot**(*boot_obj*, *attr*, *p1*, *hyb*, *p2*, *title=''*, *xlab=''*, *ylab=''*, *color='b'*)

Make a Seaborn violinplot.

Parameters

- **boot_obj** (phyde.Bootstrap) – An object of class phyde.Bootstrap.
- **attr** (*str*) – name of hypothesis test attribute to plot (e.g., "Gamma", "Zscore", "Pvalue", etc.)
- **p1** (*str*) – parent one.
- **hyb** (*str*) – putative hybrid.
- **p2** (*str*) – parent two.
- **title** (*str*) – plot title.
- **xlab** (*str*) – x-axis label.
- **ylab** (*str*) – y-axis label.
- **color** (*str*) – plot color.

Example:

```
import phyde as hd
boot = hd.Bootstrap("hydeboot.txt")
hd.viz.violinplot(boot, "Pvalue", "sp1", "sp2", "sp3", title="Bootstrap␣
↪distribution of P-value",
                  xlab="P-value", ylab="density")
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

D

H

L

R

T

V

W