morpho Documentation

Release v1.5.0-0-g131fa05

The Project 8 Collaboration

Jun 07, 2018

Contents

Contents:

CHAPTER 1

What's New

This documentation, for one…

CHAPTER 2

Install

2.1 Dependencies

The following dependencies should be installed (via a package manager) before installing morpho:

- python (2.7.x; 3.x not yet supported)
- python-pip
- git
- python-matplotlib

Morpho reads and saves files in either **R** or **ROOT.** If you would like to use root, install root-system or see https://root.cern (and ensure that the same version of python is enabled for morpho and ROOT).

2.2 Virtual environment-based installation

We recommend installing morpho using pip inside a python virtual environment. Doing so will automatically install dependencies beyond the four listed above, including PyStan 2.15.

If necessary, install virtualenv, then execute:

```
bash
virtualenv ~/path/to/the/virtualenvironment
source ~/path/to/the/virtualenvironment/bin/activate #Activate the environment
#Use "bash deactivate" to exit the environment
pip install -U pip #Update pip to >= 7.0.0
cd ~/path/to/morpho
pip install .
pip install .[all]
```

2.3 Docker installation

If you would like to modify your local installation of morpho (to add features or resolve any bugs), we recommend you use a Docker container instead of a python virtual environment. To do so:

1. Install Docker: https://docs.docker.com/engine/installation/.
2. Clone and pull the latest master version of morpho.
3. Inside the morpho folder, execute `docker-compose run morpho`. A new terminal prompter (for example, `root@413ab10d7a8f:`) should appear. You may make changes to morpho either inside or outside of the Docker container. If you wish to work outside of the container,move morpho to the `morpho_share` directory that is mounted under the `/host` folder created by docker-compose.
4. You can remove the container image using `docker rmi morpho_morpho`.

If you develop new features or identify bugs, please open a GitHub issue.

2.4 Running Morpho

Once the relevant data, model and configuration files are at your disposal, run morpho by executing:

```
bash
morpho --config  /path/to/json_or_yaml_config_file --other_options
```

You can test morpho using the example in the morpho_test directory:

```
bash
morpho --config morpho_test/scripts/morpho_linear_fit.yaml
```

When you run morpho, it performs each of the following actions, in this order:

1. If the configuration file includes a `data` dictionary, morpho reads any Stan data parameter values under `type: mc` in that file and loads any named R or ROOT files.
2. If `do_preprocessing` is `true` in the configuration file, morpho executes the methods specified under `preprocessing` in that file. See preprocessing options here.
3. If `do_stan` is `true`, morpho searches for and uses a cached version of the compiled Stan model file. If the cache file does not exist, morpho compiles the model and creates a new cache file. Morpho then runs Stan, prints out summary statistics regarding posteriors (as well as basic diagnostic information), and outputs results to an R or ROOT file, as specified under `output` in the configuration file.
4. If `do_plots` is `true`, morpho executes the methods specified under `plot` in the configuration file to create and save plots. See plotting options here.
5. If `do_postprocessing` is `true`, morpho executes the methods specified under `postprocessing` in the configuration file and optionally saves results. See post-processing options here.

"Help will always be given to those who ask for it":

```
bash
morpho --help
```

CHAPTER 3

Morpho

Morpho is a python interface to the Stan/PyStan Markov Chain Monte Carlo package.

Morpho is intended as a meta-analysis tool to fit or generate data, organize inflow and outflow of data and models.

For more information, also see:

Stan: http://mc-stan.org

PyStan: https://pystan.readthedocs.io/en/latest/index.html

3.1 An Example File

The format allows the user to execute Stan using standarized scripts. Let us now take apart an example file to illustrate how morpho functions. You can find the example file in:

```
morpho/examples/morpho_test/scripts/morpho_linear_fit.yaml
```

Let us start with the initiation portion of the configuration.

```
morpho:
 do_preprocessing: False
 do_stan: True
 do_postprocessing: False
 do_plots: True
```

Under the morpho block, you can select how the processors will be run. In this case, it will run the main Stan function and produce plots at the end of processing.

Next, we come to the main Stan configuration block, where both running conditions, data and parameters can be fed into the Stan model.

```
stan:
name: "morpho_test"
```

(continues on next page)

(continued from previous page)

```
model:
   file: "./morpho_test/models/morpho_linear_fit.stan"
   function_file: None
   cache: "./morpho_test/cache"
 data:
   files:
   - name: "./morpho_test/data/input.data"
      format: "R"
   parameters:
    - N: 30
 run:
   algorithm: "NUTS"
   iter: 4000
   warmup: 1000
   chain: 12
   n_jobs: 2
   init:
    - slope : 2.0
      intercept : 1.0
      sigma: 1.0
 output:
   name: "./morpho_test/results/morpho_linear_fit"
   format: "root"
   tree: "morpho_test"
   inc_warmup: False
   branches:
   - variable: "slope"
     root_alias: "a"
   - variable: "intercept"
     root_alias: "b"
```

The model block allows you to load in your Stan model file (for more on Stan models, see PyStan or Stan documentations). The compiled code can be cached to reduce running time. It is also possible to load in *external* functions located in separated files elsewhere.

The next block, the data block, reads in data. File formats include R and root. One can also load in parameters directly using the parameters block, as we do for the variable *N*.

The next block, the run block, allows one to control how Stan is run (number of chains, warmup, algorithms, etc.). Initializations can also be set here. This block feeds directly into PyStan.

The last block within the Stan block is the output. In this example, we save to a root file, and maintain two variables, *a* and *b*.

Since we specified the configure file to also make some plots, we can set up those conditions as well. In our example again, we have:

```
plot:
 which_plot:
  - method_name: histo
    module_name: histo
    title: "histo"
    input_file_name : "./morpho_test/results/morpho_linear_fit.root"
    input_tree: "morpho_test"
    output_path: ./morpho_test/results/
    data:
      - a
```

The plot saves a PDF of the variable *a* based on the root file results.

The flow is thus as follows. Morpho is told to execute Stan and its plotting features. The Stan execution reads in external data and sets the running in much the same way as PyStan does. Results are then saved to the results folder (in this case, under root files). Plots are also executed to ensure the quality of results.

CHAPTER 4

Preprocessing

Preprocessing functions are applied to data in advance of executing the fitter. Typically this is done to prepare the data in some state in advance of fitting.

Preprocessing can be set as a flag in the beginning of the configuration file. As an example

```
morpho:
  do_preprocessing: true
```

Later in the configuration file, you can set up the commands to pre-process data

```
preprocessing:
 which_pp:
  - method_name: bootstrapping
    module_name: resampling
    input_file_name: ./my_spectrum.root
    input_tree: input
    output_file_name: ./my_fit_data.root
    output_tree: bootstrapped_data
    option: "RECREATE"
    number_data: 5000
```

In the above example, it will randomly sample 5000 data points from the root file “my_spectrum.root” (with tree input) and save it to a new data file called “./my_fit_data.root” with tree name ” bootstrapped_data”.

CHAPTER 5

Postprocessing

Postprocessing functions are applied to data after executing the fitter. Typically this is done examine the parameter information and check for convergence.

Postprocessing can be set as a flag in the beginning of the configuration file. As an example

```
morpho:
  do_postprocessing: true
```

Later in the configuration file, you can set up the commands to post-process data. For example, to reduce the data into bins

```
preprocessing:
 which_pp:
  - method_name: general_data_reducer
    module_name: general_data_reducer
    input_file_name: ./my_spectrum.root
    input_file_format: root
    input_tree: spectrum
    data:
     -Kinetic_Energy
    minX:
     -18500.
    maxX:
     -18600.
    nBinHisto:
     -1000
    output_file_name: ./my_binned_data.root
    output_file_format: root
    output_tree: bootstrapped_data
    option: "RECREATE"
```

In the above example, it will take data from the root file saved in the *Kinetic_Energy* parameter and rebin it in a 1000-bin histogram.

CHAPTER 6

Plots

Plotting is a useful set of routines to make quick plots and diagnostic tests, usualluy after the Stan main executable has been run.:

```
morpho:
  do_plots: true
```

Later in the configuration file, you can set up the commands to plot data after the fitter is complete.

```
plot:
which_plot:
 - method_name: histo
    title: "histo"
    input_file_name : "./morpho_test/results/morpho_linear_fit.root"
    input_tree: "morpho_test"
    output_path: ./morpho_test/results/
    data:
     - a
```

In the above example, it will take data from the root file saved in the *a* parameter plot and save it to ./morpho_test/results/histo_a.pdf

We have plotting schemes that cover a number of functions:

1. Plotting contours, densities, and matricies (often to look for correlations).
2. Time series to study convergences.

CHAPTER 7

Morpho 1 Example Scripts

The following are example yaml scripts for important Preprocessing, Postprocessing, and Plot routines in Morpho 1. The format of the yaml script for other methods can be obtained from the documentation for that method.

7.1 Preprocessing

"do_preprocessing : true" must be in the morpho dictionary. The dictionaries below should be placed in a "which_pp" dictionary inside the "preprocessing" dictionary.

7.1.1 bootstrapping

Resamples the contents of a tree. Instead of regenerating a fake data set on every sampler, one can generate a larger data set, then extract subsets.

```
- method_name: "boot_strapping"
  module_name: "resampling"
  input_file_name: "input.root" # Name of file to access
                                # Must be a root file
  input_tree: "tree_name" # Name of tree to access
  output_file_name: "output.root" # Name of the output file
                                  # The default is the same the input_file_name
  output_tree: "tree_name" # Tree output name
                           # Default is same as input.
  number_data: int # Number of sub-samples the user wishes to extract.
  option: "RECREATE" # Option for saving root file (default = RECREATE)
```

7.2 Postprocessing

"do_postprocessing : true" must be in the morpho dictionary. The dictionaries below should be placed in a "which_pp" dictionary inside the "postprocessing" dictionary.

7.2.1 general_data_reducer

Tranform a function defining a spectrum into a histogram of binned data points.

```
- method_name: "general_data_reducer"
  module_name: "general_data_reducer"
  input_file_name: "input.root" # Path to the root file that contains the raw data
  input_file_format: "root" # Format of the input file
                            # Currently only root is supported
  input_tree: "spectrum" #  Name of the root tree containing data of interest
  data: ["KE"] # Optional list of names of branches of the data to be binned
  minX:[18500.] # Optional list of minimum x axis values of the data to be binned
  maxX:[18600.] # Optional list of maximum x axis values of the data to be binned
  nBinHisto:[50] # List of desired number of bins in each histogram
  output_file_name: "out.root", # Path to the file where the binned data will be saved
  output_file_format: "root", # Format of the output file
  output_file_option: RECREATE # RECREATE will erase and recreate the output file
                               # UPDATE will open a file (after creating it, if it␣
↪does not exist) and update the file.
```

7.3 Plot

"do_plots : true" must be in the morpho dictionary. The dictionaries below should be placed in a "which_plot" dictionary inside the "plot" dictionary.

7.3.1 contours

contours creates a matrix of contour plots using a stanfit object

```
- method_name: "contours"
  module_name: "contours"
  read_cache_name: "cache_name_file.txt" # File containing path to stan model cache
  input_fit_name: "analysis_fit.pkl"# pickle file containing stan fit object
  output_path: "./results/" # Directory to save results in
  result_names: ["param1", "param2", "param3"] # Names of parameters to plot
  output_format: "pdf"
```

7.3.2 histo

Plot a 1D histogram using a list of data

```
- method_name: "histo"
  module_name: "histo"
```

7.3.3 spectra

Plot a 1D histogram using 2 lists of data giving an x point and the corresponding bin contents

```
- method_name: "spectra"
  module_name: "histo"
  title: "histo"
  input_file_name : "input.root"
  input_tree: "tree_name"
  output_path: "output.root"
  data:
      - param_name
```

7.3.4 histo2D

Plot a 2D histogram using 2 lists of data

```
- method_name: "histo2D"
  module_name: "histo"
  input_file_name : "input.root"
  input_tree: "tree_name"
  root_plot_option: "contz"
  data:
    - list_x_branch
    - list_y_branch
```

7.3.5 histo2D_divergence

Plot a 2D histogram with divergence indicated by point color

```
- method_name: "histo2D_divergence"
  module_name: "histo"
  input_file_name : "input.root"
  input_tree: "tree_name"
  root_plot_option: "contz"
  data:
    - list_x_branch
    - list_y_branch
```

7.3.6 aposteriori_distribution

Plot a grid of 2D histograms

```
- method_name: "aposteriori_distribution"
  module_name: "histo"
  input_file_name : "input.root"
  input_tree: "tree_name"
  root_plot_option: "cont"
  output_path: output.root
  title: "aposteriori_plots"
  output_format: pdf
  output_width: 12000
  output_height: 1100
  data:
    - param1
    - param2
    - param3
```

7.3.7 correlation_factors

Plot a grid of correlation factors

```
- method_name: "correlation_factors"
  module_name: "histo"
  input_file_name : "input.root"
  input_tree: "tree_name"
  root_plot_option: "cont"
  output_path: output.root
  title: "aposteriori_plots"
  output_format: pdf
  output_width: 12000
  output_height: 1100
  data:
    - param1
    - param2
    - param3
```

CHAPTER 8

Contribute

8.1 Branching Model

Morpho uses the git flow branching model, as described here. In summary, the master branch is reserved for numbered releases of morpho. The only branches that may branch off of master are hotfixes. All development should branch off of the develop branch, and merge back into the develop branch when complete. Once the develop branch is ready to go into a numbered release, a release branch is created where any final testing and bug fixing is carried out. This release branch is then merged into master, and the resulting commit is tagged with the number of the new release.

Currently Morpho has two development branches. develop is used for Morpho 1 development, while morpho2/develop is used for Morpho 2 development.

8.2 Style

Morpho loosely follows the style suggested in the Style Guide for Python (PEP 8).

Every package, module, class, and function should contain a docstring, that is, a comment beginning and ending with three double quotes. We use the Google format, because the docstrings can then be automatically formatted by sphinx and shown in the API.

Every docstring should start with a single line (<=72 characters) summary of the code. This is followed by a blank line, then further description is in paragraphs separated by blank lines. Functions should contain "Args:", "Returns:", and if necessary, "Raises" sections to specify the inputs, outputs, and exceptions for the function. All text should be wrapped to around 72 characters to improve readability.

8.3 Other Conventions

- __init__.py files:

In morpho 1, __init__.py files are set up such that

```
from package import *
```

will import all functions from all subpackages and modules into the namespace. If a package contains the subpackages "subpackage1" and "subpackage2", and the modules "module1" and "module2", then the __init__.py file should include imports of the form:

```
from . import subpackage1
from . import subpackage2
from ./module1 import *
from ./module2 import *
```

In morpho 2, __init__.py files are set up such that

```
from package import *
```

will import all modules into the namespace, but it will not directly import the functions into the namespace. For our package containing "subpackage1", "subpackage2", "module1", and "module2", __init__.py should be of the form:

```
__all__ = ["module1", "module2"]
```

In this case, functions would be called via module1.function_name(). If one wants all of the functions from module1 in the namespace, then they can include "from package.module1 import *" at the top of their code. This change to more explicit imports should prevent any issues with function names clashing as Morpho grows.

CHAPTER 9

morpho

9.1 morpho package

All modules and packages used by morpho

Subpackages:

- preprocessing: Process inputs before passing to stan
- loader: Load data for use by stan
- plot: Create plots from stan outputs
- postprocessing: Process stan outputs before or after plotting

Subpackages:

9.1.1 morpho.loader package

Import and format data for use by stan

Modules:

- pystanLoad: Load root or hdf5 files and format for use by stan

Submodules:

morpho.loader.pystanLoad module

Some template vars

Members: build_tree_from_dict extract_data_from_outputdata insertIntoDataStruct open_or_create readLabel stan_data_files stan_write_root theHack theTrick transform_list_of_dict_into_dict write_result_hdf5 Functions: build_tree_from_dict extract_data_from_outputdata insertIntoDataStruct open_or_create readLabel stan_data_files stan_write_root theHack theTrick transform_list_of_dict_into_dict write_result_hdf5 Classes:

Import root and hdf5 files for use by pystan

Functions:

- readLabel: Get an item from a dictionary
- insertIntoDataStruct: Insert item into dictionary
- theHack: Format a string
- stan_data_files: Read in a dictionary of data files
- extract_data_from_output_data: Extract Stan output into a dictionary
- open_or_create: Create a group in an hdf5 object
- write_result_hdf5: Write Stan results to an hdf5 file
- theTrick: Transform dict into a dict with depth indicated by "."
- transform_list_of_dict_into_dict: Transform elements of any lists inside a dict into separate dict entries
- build_tree_from_dict: Create a root TTree object from a dictionary
- stan_write_root: Save Stan inputs and outputs into a root file

Summary

Functions:

`morpho.loader.pystanLoad.`**`build_tree_from_dict`**(*treename*, *input_param*)
Create a root TTree object from a dictionary

Parameters

- **`treename`** – Desired name of the output tree
- **`input_param`** – Dictionary of values to place in tree. It must be depth 1.

Returns Tree containing data from input_param

Return type ROOT.TTree

`morpho.loader.pystanLoad.`**`extract_data_from_outputdata`**(*conf*, *theOutput*)
Extract the output data from a morpho object into a dictionary

Parameters

- **`conf`** – morpho object, after Stan has been run
- **`theOutput`** – stanfit object returned by Stan

Returns Contains all extracted data

Return type dictionary

`morpho.loader.pystanLoad.`**`insertIntoDataStruct`**(*name*, *aValue*, *aDict*)
Insert an item into a dictionary

Parameters

- **`name`** – Key used to insert item
- **`aValue`** – Value to insert
- **`aDict`** – Dictionary value is inserted into

Returns aDict with aValue inserted at name

morpho.loader.pystanLoad.**open_or_create**(*hdf5obj*, *groupname*)
Return a group from an hdf5 object

Parameters

- **hdf5obj** – hdf5 object to access
- **groupname** – group to access or create

Returns Creates a group within the given hdf5 object if it doesn't already exist, then returns the group.

Return type h5py.Group

morpho.loader.pystanLoad.**readLabel**(*aDict*, *name*, *default=None*)
Get an item from a dictionary, or return default

Parameters

- **aDict** – Dictionary to search
- **name** – Key used to search aDict
- **default** – Default value returned if the given key does not exist. Defaults to None.

Returns aDict[name], or default if name does not exist

morpho.loader.pystanLoad.**stan_data_files**(*theData*)
Read in a dictionary of R, hdf5 and root files

theData must be a dictionary containing information about the data that should be read in for use by stan. theData should contain up to two dictionaries, stored with the keys 'files' and 'parameters'. These dictionaries should be set up as follows:

Parameters

- **theData.files.name** – Name of the file to access
- **theData.files.format** – Filetype. Options are 'R', 'root' or 'hdf5'
- **theData.files.datasets** – List of datasets to be used when accessing an hdf5 file (required only if 'format'=='hdf5')
- **theData.files.tree** – Name of tree to access when accessing a root file (required only if 'format'=='root')
- **theData.files.branches** – Branches to be accessed in the given root tree (required only if 'format'=='root')
- **theData.files.cut** – String specifying the cut for a root file (optional)
- **theData.parameters** – All values from this list are added to the returned list

Returns All values from the given files are returned as a dictionary

Return type dictionary

morpho.loader.pystanLoad.**stan_write_root**(*conf*, *theFileName*, *theOutput*, *input_param*)
Save Stan inputs and outputs ina root files

Parameters

- **conf** – morpho object containing the configuration
- **theFileName** – Desired name of the output root file
- **theOutput** – stanfit object

- **input_param** – Dictionary containing any other parameters to be saved to the root tree

Returns The given theOutput and input_param values are saved to a root file named theFileName.

Return type None

morpho.loader.pystanLoad.**theHack**(*theString*, *theVariable*, *theSecondVariable=''*, *theThirdVariable=''*)
Format theString using the given variables

morpho.loader.pystanLoad.**theTrick**(*thedict*, *uppertreename=''*)
Transform a dictionary of dictionaries into a depth 1 dictionary

Takes a dictionary that contains other dictionaries and transforms it into a dictionary that has depth 1. The key for each value in the returned dictionary is that values path in the previous dictionary, with '.' indicating descent into a lower dictionary.

Parameters

- **thedict** – Dictionary to transform
- **uppertreename** – Prefix for all keys in the returned dictionary

Returns Depth one dictionary containing the same values, with key names equal to the path through thedict

Return type dictionary

morpho.loader.pystanLoad.**transform_list_of_dict_into_dict**(*thedict*)
Transform elements of lists into separate dict entries

First, look for any list of dicts inside the dict and transform it into lists. Then, flatten any list of list into new list. For example, {'xxx':[[1,2],[2,3]]} transforms into {'xxx_1':[1,2],'xxx_2':[2,3]})

Parameters **thedict** – Dictionary to transform

Returns the dict, with any elements that are lists separated into separate dictionary entries

Return type dictionary

morpho.loader.pystanLoad.**write_result_hdf5**(*conf*, *theFileName*, *stanres*, *input_param*)
Write the Stan result to an HDF5 file

Parameters

- **conf** – morpho object
- **theFileName** – Name of hdf5 output file, without the .h5 extension
- **stanres** – stanfit object containing the results to write
- **input_param** – No longer used

Returns None

End of modules condition End of data condition

9.1.2 morpho.plot package

Plotting routines for use after Stan

Modules:

- contours: Create a matrix of contour plots from a stanfit object

- histo: Plot 1D and 2D histograms
- neutrino_params: Example to plot quantities for a neutrino mass analysis
- plotting_routines: Private functions used in other plotting modules
- spectra: Plot spectra related to a neutrino mass analysis
- timeseries: Save multiple plots in a root file

Submodules:

morpho.plot.contours module

Some template vars

Members: contours Functions: contours Classes:

Create a matrix of contour/2D density plots

Functions:

- contours: Create contour plots matrix using a Stan model fit object

---

Todo:

- Resolve ROOT/scipy compatibility issue.
- Fix matrix formatting (including axes).
- Include histograms on top/left hand side.
- Color code histograms and create key.
- Contours at 1, 2, 3 sigma lines.

Summary

Functions:

`morpho.plot.contours.`**`contours`**(*param_dict*)
Create a matrix of contour plot using a stan fit object

Parameters **`param_dict`** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The resulting plot is stored in a file. The plot will be a grid of contour plots, with all given params plotted against one another.

Return type None

End of modules condition End of data condition

morpho.plot.histo module

Some template vars

Members: aposteriori_distribution correlation_factors histo histo2D histo2D_divergence spectra Functions: aposteriori_distribution correlation_factors histo histo2D histo2D_divergence spectra Classes:

Generic methods to display histograms with ROOT

Able to plot 1D and 2D histograms

Functions

- histo: plot a 1D histogram using a list of data
- spectra: plot a 1D histogram using two lists of data (x,bin_content)
- histo2D: plot a 2D histogram using two lists of data (x,y)
- histo2D_divergence: plot a 2D histogram with divergence indicated
- aposteriori_distribution: Plot a grid of 2D histograms
- correlation_factors: Plot a grid of correlation factors

Summary

Functions:

morpho.plot.histo.**aposteriori_distribution**(*param_dict*)
Plot a grid of 2D histograms

Plot a matrix of 2D histograms using a list of data names (>=3). It will form pairs of variables (x,y) and arrange the 2D histograms to get a view of the aposteriori distribution for every combination of parameters.

Parameters **param_dict** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The grid of plots is written to file

Return type None

morpho.plot.histo.**correlation_factors**(*param_dict*)
Plot the correlation factors for pairs of parameters

Create a plot with all variables along each axis, where the correlation factor between each pair of variables is represented by color.

Parameters **param_dict** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The grid is written to file

Return type None

morpho.plot.histo.**histo**(*param_dict*)
Create a 1D histogram using a list

Parameters **param_dict** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The histo plot is written to file

Return type None

morpho.plot.histo.**histo2D**(*param_dict*)
Plot 2D histogram from a list of (x,y) points

Parameters **param_dict** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The histo plot is written to file

Return type None

morpho.plot.histo.**histo2D_divergence**(*param_dict*)
Plot 2D histogram with divergence indicated by color

Parameters **param_dict** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The histo plot is written to file. Points with divergence==1 will be one color, and points with divergence==0 will be a different color.

Return type None

morpho.plot.histo.**spectra**(*param_dict*)
Plot a 1D spectrum using a (X,Y) list

Parameters **param_dict** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The spectrum plot is written to file

Return type None

End of modules condition End of data condition

morpho.plot.neutrino_params module

Some template vars

Members: neutrino_params Functions: neutrino_params Classes:

Example module to plot quantities related to a neutrino mass analysis

Create traceplots, sampling plots, and other related plots of neutrino mixing and mass parameters outputted by the analysis model.

In a configuration file plotting parameter dictionary, the user should specify which plots he or she wishes to create using the "plotting_options" list (e.g. "plotting_options": ["neutrino_masses", "mass_params", "mixing_params"]).

Functions:

- neutrino_params: Plot neutrino parameters

Todo: Allow for more flexible and/or user defined ranges for plots of parameters. Allow for more flexible contour level inputs. Clean up error messages.

Summary

Functions:

morpho.plot.neutrino_params.**neutrino_params**(*param_dict*)
Plot parameters related to neutrino mass analysis

Loads a Stan ModelFit from a pickle file. Then invokes whichever plotting functions are indicated by the 'plotting_options':[opt1, opt2 . . .] entry in param_dict.

Possible options: 'neutrino_masses', 'mass_params', 'mixing_params', 'contours'

param_dict is a dictionary containing the following plotting info

Parameters

- **output_path** – Path to save output
- **output_format** – Format of output file
- **read_cache_name** – name of a file containing the cache filename
- **input_fit_name** – name of a pickle file
- **data** – dictionary with names of Stan parameters
- **plotting_options** – List with some combination of 'neutrino_masses', 'mass_params', 'mixing_params', and 'contours'
- **hierarchy** – Specification of the mass hierarchy (either 'normal'or 'inverted')

End of modules condition End of data condition

morpho.plot.plotting_routines module

Some template vars

Members: Functions: Classes:

Private functions used in multiple plotting modules

End of modules condition End of data condition

morpho.plot.spectra module

Some template vars

Members: spectra Functions: spectra Classes:

Plot beta spectrum

Functions:

- spectra: Plot a beta spectrum

Summary

Functions:

morpho.plot.spectra.**spectra**(*param_dict*)
Plot a spectral shape

Takes as input a set of input parameters and/or by scatter plotting (KE, spectrum) pairs.

Unpickles a Stan fit object. Creates from data_names a new dictionary data_vals that is useful for spectral shape plotting. Then invokes whichever plotting functions are indicated by "plotting_options."

In a configuration file plotting parameter dictionary, the user should specify which plots he or she wishes to create using the "plotting_options" list (e.g. "plotting_options": ["spectrum_shape", "spectrum_scatter", "overlay"]).

param_dict is a dictionary containing the following fields:

Parameters

- **path** (*output*) – Path to save output
- **ouput_format** – Format of output file
- **read_cache_name** – Name of a file containing the cache filename
- **data_names** – dictionary with useful values, plot labels, and names of Stan parameters
- **plotting_options** – Plotting options
- **x_range** – List with x range
- **y_scale** – String with y_scale
- **num_x** – Number of x bins (optional, default 50)

End of modules condition End of data condition

morpho.plot.timeseries module

Some template vars

Members: timeseries Functions: timeseries Classes:

Generic methods to display histograms with ROOT

Functions:

- time_series: Save multiple plots in a root file

Summary

Functions:

morpho.plot.timeseries.**timeseries**(*param_dict*)
Save multiple plots in a root file

param_dict is a dictionary with the following fields:

Parameters

- **title** – Plot title
- **input_files_name** – Path to input file
- **input_file_format** – Input file format (only "root" is currently supported)
- **input_tree** – Name of input tree
- **output_path** – Path to output tree
- **output_format** – Output format, eg PDF

- **output_width** – Output width in pixels
- **output_height** – Output height in pixels
- **data** – Names of branches to plot
- **y_title** – Titles to label each branch with in the legend

Returns Canvas with the given plots

Return type TCanvas

End of modules condition End of data condition

9.1.3 morpho.postprocessing package

Perform preprocessing routines designed to run before Stan

Modules:

- general_data_reducer: Transform any spectrum into a histogram of binned data points
- stan_utility: Perform Stan diagnostic tests
- data_reducer: Transform a spectrum into a histogram of binned data points, for frequency, KE, or time spectra. (Soon deprecated by general_data_reducer).

Submodules:

morpho.postprocessing.data_reducer module

Some template vars

Members: data_reducer readTTree Functions: data_reducer readTTree Classes:

Transform a spectrum into a histogram of binned data points

data_reducer specifically looks for x values that are time, frequency, or kinetic energy

data_reducer will soon be deprecated by general_data_reducer

Functions:

- data_reducer: Convert a spectrum into a histogram
- readTTree: Retrieve specific branches from a tree

Todo:

easy tasks:

- **extend the number of nBinHisto to allow the user to have** different spectrum depending on the nature of the histo (time, frequency. . .)

harder tasks:

- **make this data reducer very generic (to be able to choose** between frequency, energ or time spectrum) or add the energy spectrum by default
- implement the h5 reader and writter

- **A possibility is also to make a tree with n branches with only** one element, these elements are then read in the analyzer as the number of bin/data to be analyzed

Summary

Functions:

`morpho.postprocessing.data_reducer.`**`data_reducer`**(*param_dict*)

Convert a spectrum into a histogram

Takes a set of x and y values defining a spectrum and creates a list of x and y values defining a histogram. The y values can optionally be poisson distributed in order to represent fake data.

param_dict is a dict containing all inputs. The following describes the elements of param_dict, along with exampl values.

Parameters

- **`param_dict.which_spectrum`** – ["frequency","time","KE"], #list of the reductions to be performed
- **`param_dict.Poisson_redistribution`** – True, #is a Poisson redistribution of the data required?
- **`param_dict.input_file_name`** – "input.root", #path to the root file which contains the raw data
- **`param_dict.input_file_format`** – "root", #format of the input file
- **`param_dict.input_tree`** – "stan_MC", # name of the tree (in case of root file)
- **`param_dict.minKE`** – 18500., #minimal energy used for generating the STAN data
- **`param_dict.maxKE`** – 18600., #maximal energy used for generating the STAN data
- **`param_dict.nBinHisto`** – 50, #number of bins wanted for the output spectrum
- **`param_dict.output_file_name`** – "out.root", #path to the root file where to save the spectrum data
- **`param_dict.output_file_format`** – "root", #format of the output file
- **`param_dict.output_file_option`** – RECREATE #give an option for the output file (RECREATE will erase and recreate the output file, UPDATE will open (after creating if not existing) and update the file)
- **`param_dict.output_freq_spectrum_tree`** – "spectrum", #name of the tree (in case of root file) which contains the frequency spectrum
- **`param_dict.output_KE_spectrum_tree`** – "spectrum", #name of the tree (in case of root file) which contains the KE spectrum
- **`param_dict.output_time_spectrum_tree`** – "time", #name of the tree (in case of root file) which contains the time spectrum
- **`param_dict.additional_file_name`** – "additionalData.out", #name of the file which contains the number of bins for the output histograms

Returns The created histogram is stored in a root file

Return type None

morpho.postprocessing.data_reducer.**readTTree**(*root_file_path*, *tree_name*)
Retrieve specific branches from a tree

Searches the given root TTree for branches named 'time_data', 'freq_data', 'spectrum_data', and 'KE_data' and returns them as lists.

Parameters

- **root_file_path** – Filepath to the root file containing the TTree
- **tree_name** – Name of the ttree to access

Returns (time_data, freq_data, spectrum_data, KE_data) where if the given branch was not in the tree, then an empty list is returned.

Return type (list, list, list, list)

End of modules condition End of data condition

morpho.postprocessing.general_data_reducer module

Some template vars

Members: general_data_reducer Functions: general_data_reducer Classes:

Transform a spectrum into a histogram of binned data points

general_data_reducer transforms a spectrum into a histogram of binned data points and saves the results in an output file. This module must be called in a dictionary under "postprocessing" within the configuration file.

Todo:

- Update this code to allow for data of the form (X, Y) as input

---

Summary

Functions:

morpho.postprocessing.general_data_reducer.**general_data_reducer**(*param_dict*)
Convert a spectrum into a histogram

Takes a set of x and y values defining a spectrum and creates a list of x and y values defining a histogram. The x and y values can be input via a root file or an hdf5. The resulting file can only curently be saved as a root file.

Parameters **param_dict** – dict containing all inputs. See "Morpho 1 Example Scripts" in the API for details.

Returns The resulting histogram is stored in a file.

Return type None

End of modules condition End of data condition

morpho.postprocessing.stan_utility module

Some template vars

Members: check_all_diagnostics check_div check_energy check_n_eff check_rhat check_treedepth partition_div
Functions: check_all_diagnostics check_div check_energy check_n_eff check_rhat check_treedepth partition_div
Classes:

Perform Stan diagnostic tests

Source: Michael Betancourt, https://github.com/betanalpha/jupyter_case_studies/blob/master/pystan_workflow/stan_utility.py Modified by Talia Weiss, 1-23-18

These tests are motivated here: http://mc-stan.org/users/documentation/case-studies/pystan_workflow.html

Functions:

- check_div: Check how many transitions ended with a divergence
- check_treedepth: Check how many transitions failed due to tree depth
- check_energy: Check energy Bayesian fraction of missing information
- check_n_eff: Check the effective sample size per iteration
- check_rhat: Check the potential scale reduction factors
- check_all_diagnostics: Check all MCMC diagnosticcs
- partition_div: Get divergent and non-divergent parameter arrays

Summary

Functions:

`morpho.postprocessing.stan_utility.`**`check_all_diagnostics`**(*fit*)
Checks all MCMC diagnostics

Parameters **`fit`** – stanfit object containing sampler output

Returns Returns the strings indicating the results of the checks for divergence, treee depth, energy Bayesian fraction of missing energy, effective sample size, and Rhat

Return type list of str

`morpho.postprocessing.stan_utility.`**`check_div`**(*fit*)
Check how many transitions ended with a divergence

Parameters **`fit`** – stanfit object containing sampler output

Returns States the number of transitions that ended with a divergence

Return type str

`morpho.postprocessing.stan_utility.`**`check_energy`**(*fit*)
Checks the energy Bayesian fraction of missing information (E-BFMI)

Parameters **`fit`** – stanfit object containing sampler output

Returns Warns that the model may need to be reparametrized if E-BFMI is less than 0.2

Return type str

`morpho.postprocessing.stan_utility.`**`check_n_eff`**(*fit*)
Checks the effective sample size per iteration

Parameters **fit** – stanfit object containing sampler output

Returns States whether the effective sample size indicates an issue

Return type str

morpho.postprocessing.stan_utility.**check_rhat**(*fit*)
Checks the potential scale reduction factors

Parameters **fit** – stan fit object containing sampler output

Returns States whether the Rhat values indicate an error

Return type str

morpho.postprocessing.stan_utility.**check_treedepth**(*fit*, *max_depth=10*)
Check how many transitions ended prematurely due to tree depth

A transition may end prematurely if the maximum tree depth limit is exceeded.

Parameters

- **fit** – stanfit object containing sampler output
- **max_depth** – Maximum depth used to check tree depth

Returns States the number of transitions that passed the given max_depth.

Return type str

morpho.postprocessing.stan_utility.**partition_div**(*fit*)
Returns parameter arrays for divergent and non-divergent transitions

Parameters **fit** – stanfit object containing sampler output

Returns The first dictionary contains all nondivergent transitions, the second contains all divergent transitions

Return type (dict, dict)

End of modules condition End of data condition

9.1.4 morpho.preprocessing package

Preprocessing methods to be called before invoking the fitter

Modules:

- resampling: Resample the contents of a tree

Submodules:

morpho.preprocessing.resampling module

Some template vars

Members: bootstrapping Functions: bootstrapping Classes:

Resample the contents of a tree

Functions:

- bootstrapping: Resample the contents using a bootstrap technique

Summary

Functions:

morpho.preprocessing.resampling.**bootstrapping**(*param_dict*)
Resample a tree using a bootstrap technique

Rather than regenerating fake data before every call to stan, one can generate a larger data data set and then extract a random subset before each call to stan

Parameters **param_dict** – Dictionary of parameters to search. See "Morpho 1 Example Scripts" in the API for details.

Returns Creates new root file with number_data sampled entries.

Return type None

End of modules condition End of data condition

morpho.preprocessing.sample_inputs module

Some template vars

Members: sample_inputs Functions: sample_inputs Classes:

Select Stan "data" parameter inputs by sampling from priors.

Below is an example preprocessing configuration dictionary.

preprocessing:

which_pp:

- method_name: sample_inputs module_name: sample_inputs params_to_sample:
  - Q
  - sigma_value

prior_dists:

- normal
- normal

priors:

- [18575., 0.2]
- [0.04, 0.004]

fixed_params: {'KEmin':18574., 'KEmax':18575.4, 'background_fraction':0.05, 'sigma_error':0.004, 'u_value':0, 'u_spread':70, 'mass':0.2} output_file_name: "./tritium_model/results/beta_spectrum_2-19_ensemble.root" tree: inputs

Summary

Functions:

morpho.preprocessing.sample_inputs.**sample_inputs**(*param_dict*)

End of modules condition End of data condition

Python Module Index

m

Index

R

S

T

W