morpho Documentation

Release v2.1.4-0-g940e68c

The Project 8 Collaboration

Aug 01, 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

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.processors package

Submodules:

morpho.processors.BaseProcessor module

Some template vars

Members: BaseProcessor

Functions:

Classes: BaseProcessor

Base processor for sampling-type operations Authors: J. Johnston, M. Guigue, T. Weiss Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.BaseProcessor.**BaseProcessor**(*name*, **args*, ***kwargs*)
Bases: object

Base Processor All Processors will be implemented in a child class where the specifics are encoded by overwriting Configure and Run.

name

delete

Configure(*params*)
This method will be called by nymph to configure the processor

InternalConfigure(*params*)
Method called by Configure() to set up the object. Must be overridden by child class.

Run()
This method will be called by nymph to run the processor

InternalRun()
Method called by Run() to run the object. Must be overridden by child class.

Subpackages:

morpho.processors.IO package

Submodules:

morpho.processors.IO.IOCVSProcessor module

Some template vars

Members: IOCVSProcessor

Functions:

Classes: IOCVSProcessor

CVS IO Processor Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.IO.IOCVSProcessor.**IOCVSProcessor**(*name*, **args*, ***kwargs*)
Bases: *morpho.processors.IO.IOProcessor.IOProcessor*

Base IO CVS Processor The CVS Reader and Writer

Reader()
Need to be defined by the child class

Writer()
Need to be defined by the child class

morpho.processors.IO.IOJSONProcessor module

Some template vars

Members: IOJSONProcessor IOYAMLProcessor

Functions:

Classes: IOJSONProcessor IOYAMLProcessor

JSON/Yaml IO processors Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.IO.IOJSONProcessor.**IOJSONProcessor**(*name*)
Bases: *morpho.processors.IO.IOProcessor.IOProcessor*

Base IO JSON Processor

module_name = 'json'

dump_kwargs = {'indent': 4}

Reader()
Need to be defined by the child class

Writer()
Need to be defined by the child class

class morpho.processors.IO.IOJSONProcessor.**IOYAMLProcessor**(*name*)
Bases: *morpho.processors.IO.IOJSONProcessor.IOJSONProcessor*

IO YAML Processor: uses IOJSONProcessor as basis

module_name = 'yaml'

morpho.processors.IO.IOProcessor module

Some template vars

Members: IOProcessor

Functions:

Classes: IOProcessor

Base input/output processor for reading and writing operations Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.IO.IOProcessor.**IOProcessor**(*name*, **args*, ***kwargs*)
Bases: *morpho.processors.BaseProcessor.BaseProcessor*

IO_Processor All Processors will be implemented in a child class where the specifics are encoded by overwriting Configure and Run.

Reader()
Need to be defined by the child class

Writer()
Need to be defined by the child class

InternalConfigure(*params*)
This method will be called by nymph to configure the processor

InternalRun()
This method will read or write an file

morpho.processors.IO.IOROOTProcessor module

Some template vars

Members: IOROOTProcessor

Functions:

Classes: IOROOTProcessor

ROOT IO processor Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.IO.IOROOTProcessor.**IOROOTProcessor**(*name*, **args*, ***kwargs*)

Bases: *morpho.processors.IO.IOProcessor.IOProcessor*

Base IO ROOT Processor The ROOT Reader and Writer

InternalConfigure(*params*)
This method will be called by nymph to configure the processor

Reader()
Read the content of a TTree in a ROOT File. Note the use of the uproot package. The variables should be a list of the "variable" to read.

Writer()
Write the data into a TTree in a ROOT File. The variables should be a list of dictionaries where

- "variable" is the variable name in the input dictionary,
- "root_alias" is the name of the branch in the tree,
- "type" is the type of data to be saved.

morpho.processors.IO.IORProcessor module

Some template vars

Members: IORProcessor

Functions:

Classes: IORProcessor

R IO processor Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.IO.IORProcessor.**IORProcessor**(*name*, **args*, ***kwargs*)

Bases: *morpho.processors.IO.IOProcessor.IOProcessor*

Base IO R Processor The R Reader and Writer use pystan.misc package

Reader()
Need to be defined by the child class

Writer()
Need to be defined by the child class

morpho.processors.diagnostics package

Submodules:

morpho.processors.diagnostics.StanDiagnostics module

Some template vars

Members: StanDiagnostics

Functions:

Classes: StanDiagnostics

Creates Stan diagnostic plots. Authors: T. Weiss Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.diagnostics.StanDiagnostics.**StanDiagnostics**(**args*, ***kwargs*)

Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Describe.

InternalConfigure(*params*)
Configures by reading in list of names of divergence plots to be created and dictionary containing fit object

InternalRun()
Method called by Run() to run the object. Must be overridden by child class.

morpho.processors.misc package

Submodules:

morpho.processors.misc.ProcessorAssistant module

Some template vars

Members: ProcessorAssistant

Functions:

Classes: ProcessorAssistant

Create a wrapping processor from a function given in a python script Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.misc.ProcessorAssistant.**ProcessorAssistant**(*name*, **args*, ***kwargs*)

Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Describe.

InternalConfigure(*config_dict*)
Configure

InternalRun()
Method called by Run() to run the object. Must be overridden by child class.

morpho.processors.plots package

Submodules:

morpho.processors.plots.APosterioriDistribution module

Some template vars

Members: APosterioriDistribution

Functions:

Classes: APosterioriDistribution

Plot a posteriori distribution of the variables of interest Authors: J. Jonhston, M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.plots.APosterioriDistribution.**APosterioriDistribution**(*name*, **args*, ***kwargs*)

Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Generates an a posterior distribution for all the parameters of interest TODO: - Use the RootHistogram class instead of TH1F itself...

data

InternalConfigure(*param_dict*)
Configure

InternalRun()
Method called by Run() to run the object. Must be overridden by child class.

morpho.processors.plots.Histogram module

Some template vars

Members: Histogram

Functions:

Classes: Histogram

Plot an histogram of the variables of interest Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.plots.Histogram.**Histogram**(*name*, **args*, ***kwargs*)
Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Processor that generates a canvas and a histogram and saves it. TODO: - Add the possibility to plot several histograms with the same binning on the same canvas - Generalize this processor so it understands if if should be a 1D or a 2D histogram

InternalConfigure(*params*)
Configure

InternalRun()
Method called by Run() to run the object. Must be overridden by child class.

morpho.processors.plots.RootCanvas module

Some template vars

Members: RootCanvas

Functions:

Classes: RootCanvas

Root-based canvas class Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.plots.RootCanvas.**RootCanvas**(*input_dict*, *optStat='emr'*)
Bases: object

cd(*number=0*)
Go to frame 'number' of the TCanvas

Divide(*cols*, *rows*)
Divide the TCanvas

Draw()
Draw the TCanvas

Save()
Save the TCanvas

morpho.processors.plots.RootHistogram module

Some template vars

Members: RootHistogram

Functions:

Classes: RootHistogram

Root-based histogram class Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.plots.RootHistogram.**RootHistogram**(*input_dict*, *optStat='emr'*)

Bases: object

Fill(*input_data*)

SetBinsContent(*a_list*)

Draw(*arg='hist'*)

morpho.processors.plots.TimeSeries module

Some template vars

Members: TimeSeries

Functions:

Classes: TimeSeries

Plot a time series of the variables of interest Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.plots.TimeSeries.**TimeSeries**(*name*, **args*, ***kwargs*)

Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Describe.

data

InternalConfigure(*param_dict*)

Configure

InternalRun()

Method called by Run() to run the object. Must be overridden by child class.

morpho.processors.sampling package

Submodules:

morpho.processors.sampling.GaussianSamplingProcessor module

Some template vars

Members: GaussianSamplingProcessor

Functions:

Classes: GaussianSamplingProcessor

Gaussian distribution sampling processor Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.sampling.GaussianSamplingProcessor.**GaussianSamplingProcessor**(*name*, **args*, ***kwarg*

Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Sampling processor that will generate a simple gaussian distribution. Does not require input data nor model (as they are define in the class itself)

InternalConfigure(*input*)

Method called by Configure() to set up the object. Must be overridden by child class.

InternalRun()

Method called by Run() to run the object. Must be overridden by child class.

morpho.processors.sampling.LinearFitRooFitLikelihoodProcessor module

Some template vars

Members: LinearFitRooFitLikelihoodProcessor

Functions:

Classes: LinearFitRooFitLikelihoodProcessor

Processor for linear fitting Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.sampling.LinearFitRooFitLikelihoodProcessor.**LinearFitRooFitLikeliho**

Bases: *morpho.processors.sampling.RooFitLikelihoodSampler.RooFitLikelihoodSampler*

Linear fit of data using RootFit Likelihood sampler. We redefine the _defineDataset method as this analysis requires datapoints in a 2D space. Users should feel free to change this method as they see fit.

InternalConfigure(*config_dict*)
Method called by Configure() to set up the object. Must be overridden by child class.

definePdf(*wspace*)
Define the model which is that the residual of the linear fit should be normally distributed.

morpho.processors.sampling.PyStanSamplingProcessor module

Some template vars

Members: PyStanSamplingProcessor

Functions:

Classes: PyStanSamplingProcessor

PyStan sampling processor Authors: J. Formaggio, J. Johnston, M. Guigue, T. Weiss Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.sampling.PyStanSamplingProcessor.**PyStanSamplingProcessor**(*name*)
Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Sampling processor that will call PyStan

data

gen_arg_dict()

InternalConfigure(*params*)
Method called by Configure() to set up the object. Must be overridden by child class.

InternalRun()
Method called by Run() to run the object. Must be overridden by child class.

morpho.processors.sampling.RooFitLikelihoodSampler module

Some template vars

Members: RooFitLikelihoodSampler

Functions:

Classes: RooFitLikelihoodSampler

Base processor for RooFit-based samplers Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.processors.sampling.RooFitLikelihoodSampler.**RooFitLikelihoodSampler**(*name*, **args*, ***kwargs*)
Bases: *morpho.processors.BaseProcessor.BaseProcessor*

Base class for RooFit-based Likelihood sampling. A new class should inheritate from this one and have its own version of "definePdf". The input data are given via the attribute "data"<

definePdf(*wspace*)
Defines the Pdf that RooFit will sample and add it to the workspace. The Workspace is then returned by the user. Users should always create their own method.

data

InternalConfigure(*config_dict*)
Method called by Configure() to set up the object. Must be overridden by child class.

InternalRun()
Method called by Run() to run the object. Must be overridden by child class.

9.1.2 morpho.utilities package

Submodules:

morpho.utilities.morphologging module

Some template vars

Members: getLogger

Functions: getLogger

Classes:

Morpho logging utilities Authors: J. Johnston, M. Guigue Date: 02/22/18

Summary

Functions:

Reference

morpho.utilities.morphologging.**getLogger**(*name*, *stderr_lb=40*, *level=10*, *propagate=False*)

Return a logger object with the given settings that prints messages greater than or equal to a given level to stderr instead of stdout name: Name of the logger. Loggers are conceptually arranged

in a namespace hierarchy using periods as separators. For example, a logger named morpho is the parent of a logger named morpho.plot, and by default the child logger will display messages with the same settings as the parent

stderr_lb: Messages with level equal to or greaterthan stderr_lb will be printed to stderr instead of stdout

level: Initial level for the logger propagate: Whether messages to this logger should be passed to

the handlers of its ancestor

morpho.utilities.parser module

Some template vars

Members: change_and_format merge parse_args update_from_arguments

Functions: change_and_format merge parse_args update_from_arguments

Classes:

Definitions for parsing the CLI and updating the Toolbox configuration dictionary Authors: J. Johnston, M. Guigue, T. Weiss Date: 06/26/18

Summary

Functions:

Reference

morpho.utilities.parser.**parse_args**()
Parse the command line arguments provided to morpho :param None:

Returns Namespace containing the arguments

Return type namespace

morpho.utilities.parser.**update_from_arguments**(*the_dict*, *args*)
Update a dictionary :param the_dict: Dictionary to update :param args: Dictionary to merge into the_dict

Returns Dictionary with args merged into the_dict

Return type dict

morpho.utilities.parser.**change_and_format**(*b*)
Try to convert a string into a boolean or float :param b: String containing a boolean or float

Returns If b == 'True' or 'False', then the corresponding boolean is returns. Otherwise, if b can be converted into a float, the float is returned. Otherwise b is returned.

Return type bool, float, or str

morpho.utilities.parser.**merge**(*a*, *b*, *path=None*)
Merge two dictionaries :param a: Base dictionary :param b: Dictionary to merge into a :param path: Location to merge b at

Returns Merged dictionary

Return type dict

morpho.utilities.plots module

Some template vars

Members:

Functions:

Classes:

Definitions for plots Authors: J. Johnston, M. Guigue, T. Weiss Date: 06/26/18

morpho.utilities.pystanLoader module

Some template vars

Members: extract_data_from_outputdata

Functions: extract_data_from_outputdata

Classes:

Definitions for interfacing with pyStan IO Authors: M. Guigue Date: 06/26/18

Summary

Functions:

Reference

morpho.utilities.pystanLoader.**extract_data_from_outputdata**(*conf*, *theOutput*)

morpho.utilities.reader module

Some template vars

Members: add_dict_param read_param

Functions: add_dict_param read_param

Classes:

Interface between config files and processors config dictionaries Authors: J. Johnston, M. Guigue, T. Weiss Date: 06/26/18

Summary

Functions:

Reference

morpho.utilities.reader.**read_param**(*yaml_data*, *node*, *default*)

morpho.utilities.reader.**add_dict_param**(*dictionary*, *key*, *value*)

This method checks if a key already exists in a dictionary, and if not, it adds the key and its corresponding value to the dictionary.

Could be changed to take as input a list of tuples (key, value), so multiple parameters may be added at once.

morpho.utilities.toolbox module

Some template vars

Members: ToolBox

Functions:

Classes: ToolBox

Toolbox class: create, configure and run processors Authors: M. Guigue Date: 06/26/18

Summary

Classes:

Reference

class morpho.utilities.toolbox.**ToolBox**(*args*)

Manages processors requested by the user at run-time. Via a configuration file, the user defines which processor to use, how to configure them and how to connect them.

Run()

9.1.3 Summary

Data:

9.1.4 Reference

Python Module Index

m

Index

L

M

N

P

R

S

T

U

W