pinefarm

the PineLine team

Nov 14, 2023

USAGE

This documentation is about the python package used to generate the PineAPPL grids out of the input pinecards. This program is called *pinefarm* (and its CLI `pinefarm`).

CHAPTER

ONE

INSTALL `PINEFARM`

There are two ways of installing `pinefarm`, that are:

- **production**: used by *final user*, simply run

  ```
  pip install pinefarm
  ```

  and then use `pinefarm` as a command available in `PATH`

- **develop**: used by the tools developer

  ```
  poetry install
  ```

  Then run with:

  ```
  poetry run pinefarm <args>
  ```

1.1 Non Python dependencies

Even if the bootstrap script and the installation management try to reduce as much as possible the amount of dependencies, still a few ingredients have to be available on the system.

To run the CLI:

- `python` itself
- `pip` available as a module of the `python` that is running `pinefarm` (as usually is)
- `curl`

To install `pineappl`:

- `pkg-config`
- `openssl.pc` (e.g. on Debian available in the `libssl-dev` package)

To install `mg5amc@nlo` and its dependencies:

- `gfortran`
- `wget`

1.1.1 Generate a PineAPPL grid

To generate a PineAPPL grid run:

```
pinefarm run <RUNCARD> <THEORY>
```

In order to get a list of available pinecards run:

```
pinefarm list pinecards
```

Note: Use *TEST_RUN_SH* in order to test the toolchain, it should run fairly quickly.

Analogously for theories:

```
pinefarm list theories
```

If any software is missing, it will be installed on the fly, including:

- Madgraph5_aMC@NLO
- PineAPPL

Note: Only the code relevant to run the selected pinecard will be installed.

1.1.2 What is all the output?

After having run `pinefarm run DATASET THEORY` (see *pinefarm command line interface*), the script prints a table, which is useful to quickly validate the MC uncertainty and the interpolation error of PineAPPL. The last line shows the directory where all results are stored, which has the form `DATASET-DATE`, where `DATASET` is the value given to the run script and `DATE` is a numerical date when the generation was started. The date is added so runs for the same dataset do not overwrite each other's output.

The most important file in the output directory is

```
DATASET-DATE/DATASET.pineappl.lz4
```

which is the final PineAPPL grid.

The remaining contents of this directory are useful for testing and debugging:

- `results.log`: The numerical results of the run, comparing the results of the grid against the results from `mg5_aMC`. The first column (PineAPPL) are the interpolated results, which should be similar to the Monte Carlo (MC) results in the second column. The third column gives the relative MC uncertainty (sigma). The next column gives the differences in terms of multiples of sigma. The final three columns give the per mille differences of the central, minimum, and maximum scale varied results. Ideally the first two columns are the same and the remaining columns are zero.
- `time.log`: Total `time` needed for the run

Note: The resulting PineAPPL grid will contain the metadata written in the `metadata.txt` file.

In addition, the script `run.sh` and PineAPPL will automatically add the following metadata:

- `initial_state_{1,2}`: The hadronic initial states of the grid, given as PDG ids, typically `2212` for protons, `-2212` for anti-protons, and so on.

- `lumi_id_types`: The meaning of the luminosities IDs in the definition of the luminosity function of a PineAPPL grid. This is set to `pdg_mc_ids` to signal they are PDG ids (with a possible exception of the gluon, for which `0` may be used).
- `mg5amc_repo` and `mg5amc_revno` (only *Mg5aMC@NLO* grids): The repository and revision number of the Madgraph5_aMC@NLO version that was used to generate the grid.
- `pineappl_gitversion`: The PineAPPL version that was used to generate the grid.
- `results`: The comparison of the HwU results against a convolution of the PineAPPL grid with the PDF selected in `launch.txt`. This is the same table printed at the end by `run.sh`, and is used to verify the contents of each grid. It also stores the MC uncertainties.
- `pinecard`: Madgraph5_aMC@NLO's pinecard that was used to generate the grid. Here all parameters are documented.
- `pinecard_gitversion`: The git version of this repository that was used to generate the grid.
- `yadism_version` (only *Yadism* grids): The yadism version used to generate the grid (if not a released version have been used it includes also git details).

Runner dependent output

Part of the output is specific to the selected runner, and described in the corresponding *section*.

1.1.3 `pinefarm` command line interface

You can get the full update help of the command line interface (CLI) with:

```
pinefarm --help
```

Below we present a brief recap of the subcommands and their goal.

`install`

Installs various programs, used to run pinecards.

`run`

It is the main command provided, and it runs the specified pinecard in the context of the selected theory.

The output will be stored in a directory in the current path, with the name `<RUNCARD>-<YYYYMMDDhhmmss>`, where:

- the first part is the name of the selected pinecard (that is also the name of the folder in which all the files are stored)
- the second part is the timestamp of the moment in which the command is issued

update

Update the metadata of the specified grid, with the content of the `metadata.txt` file in the current version of the pinecard.

merge

Merge the specified grids' content into a new grid.

1.1.4 Pinecards preparation

Part of the pinecards is specific by external and documented in dedicated page

Common structure

The following files are important for each data set; they must be in the folder `pinecards/DATASET`, where `DATASET` is the NNPDF identifier for the dataset.

- The *metadata.txt* file (optional).
- The *postrun.sh* file (optional, must be executable).

1.1.5 Metadata

This file collects all metadata, which is written into the grid after generation.

Arbitrary `key=value` pairs are supported, the most common are:

- `arxiv`: The arxiv number of the experimental analysis, or if there are more than one, comma-seperated numbers.
- `description`: A short description of the process/observables. Make sure to include the name of the experiment and the centre-of-mass energy.
- `hepdata`: The DOI pointing to the experimental data, or a comma-separated list of DOIs. Preferably this points to specific tables of the observables specified below, as the hepdata entries usually show many of them.
- `nnpdf_id`: The NNPDF ID, which can denote multiple datasets.
- `x1_label`: The name of the first (`x2` = second, `x3` = third, ...) observable.
- `x1_label_tex`: The name of the observable, written in LaTeX.
- `x1_unit`: The unit of the observable (typically `GeV`). If this key is not present, the corresponding observable is assumed to be dimensionless.
- `y_label`: The label for the differential cross section.
- `y_label_tex`: The label for the differential cross section, written in LaTeX. Use `\frac{a}{b}` instead of `a/b` for fractions.
- `y_unit`: The unit for the cross section (typically `pb` for dimensionless observables, or `pb/GeV` or `pb/GeV^2`).

This key-value pairs are written into the final PineAPPL, to allow the user to easily identify what is stored in the grid and how it was generated. It also allows for easily plotting the contents of the grids.

Note: Further metadata are specific for each external, you can find them in the respective external page, in the section "Additional metadata".

1.1.6 Postrun

This is a BASH script which is run after the successful generation of the PineAPPL grid and can be used to perform additional operations, such as rescaling. The environment variable `$GRID` contains the relative path the PineAPPL grid. Typically, this file contains instructions to remap the one-dimensional histograms generated by Madgraph5_aMC@NLO into higher dimensional ones with the proper limits.

1.1.7 External runners

The `pinefarm run` is mainly a uniform interface to some Monte Carlo (and non) generator that are able to produce PineAPPL grids.

Internally the runners are managed through a class system, with a base class *`External`*, that defines the basic steps and attributes, while implementing the common actions.

Attributes:

- `name`: name of the dataset
- `theory`: identifier of the theory
- `pdf`: PDF used for the comparison
- `timestamp`, *optional*: the timestamp of the previous run, if rerunning an already present grid

Computed attributes:

- `dest`: folder used for all the output
- `source`: folder containing pinecard
- `grid`: path of the computed grid
- `gridtmp`: path used for auxiliary grid (removed at the end of the run)

Steps:

- *`install()`*: further install steps, needed for the runner (not needed if the runner available as a python package on PyPI)
- *`run()`*: compute the actual predictions
- *`generate_pineappl()`*: collect predictions into a grid
- *`results()`*: provide runner results on chosen PDF, for comparison with `convolute` ones
- *`annotate_versions()`*: collect versions of all the program used to compute the results (for reproducibility)
- *`postprocess()`*: apply any further step specified in *postprocess file*, save *metadata*, and compress the final grid

1.1.8 Mg5aMC@NLO

Launch

Here is briefly recapped 's `launch.txt` special syntax that is allowed for (and introduced by) pinecards.

Special syntax

Since the parameter values are inserted by `run.sh`, do not insert the numerical values into the text file but rather the run variables. Supported are `@GF`, `@MH@`, `@MT@`, `@MW@`, `@MZ@`, `@WH@`, `@WT@`, `@WW@`, and `@WZ@`. The names are the same as chosen by `mg5_aMC`, but written in uppercase and surrounded with `@`. For details about more parameters, please see the `Template/NLO/Cards/run_card.dat` file in .

Cuts

They are implemented in two steps:

1. cuts relevant *variables* are defined
2. cuts *code* is implemented

List of cuts

- `abscoscsmax`
- `abscoscsmin`
- `atlas_1jet_8tev_r06`
- `atlas_2jet_7tev_r06_0005`
- `atlas_2jet_7tev_r06_0510`
- `atlas_2jet_7tev_r06_1015`
- `atlas_2jet_7tev_r06_1520`
- `atlas_2jet_7tev_r06_2025`
- `atlas_2jet_7tev_r06_2530`
- `atlas_dy3d_8tev`
- `atlas_wzrap11_cf`
- `cms_2jet_3d_8tev`
- `cms_2jets_7tev_0005`
- `cms_2jets_7tev_0510`
- `cms_2jets_7tev_1015`
- `cms_2jets_7tev_1520`
- `cms_2jets_7tev_2025`
- `dyjj`
- `minetal`
- `mjj`

- mmllmax
- mtw
- ptj1min
- ptl1min
- ptmiss
- ptzmax
- ptzmin
- yh
- yll
- yt
- yz
- yzmin

Cuts variables

- abscoscs
- atlas_1jet_8tev_r06
- atlas_dy3d_8tev
- atlas_wzrap11_cf
- cms_2jet_3d_8tev
- mtw
- ptmiss

Patches

For instance, to use a dynamical scale, a patch modifying `setscales.f` file should be included in the directory. To create patches use the command `diff -Naurb original new > patch.patch`. The patches are applied in an unspecified order, using `patch -p1 ...`.

List of available patches

`change_etaj_to_rapj`

`no_pole_cancellation_checks`

`set_tau_min`

Runcard structure

- The `output.txt` file (compulsory). This file contains the instructions to generate the source code for the relevant process. For details, please see arXiv:1804.10017 and arXiv:1405.0301. The variable `@OUTPUT@` must be used to generate the directory containing the source files.

- The `launch.txt` file (compulsory). This file contains the instructions to run the relevant process, including the relevant physical parameters and cuts, more info in *Launch*.
- The `analysis.f` file (compulsory). This Fortran file must fill the histograms from which the `HwU` files (histograms with uncertainties) and the PineAPPL grids are generated. Note that a single histogram must not contain more than 100 bins, otherwise will crash. However, big histograms can be split up into multiple histograms, for which the runner will merge the PineAPPL grids together.
- The `*.patch` file(s) (optional). These are one or more `.patch` files that are applied after has generated the sources.

Additional metadata

- `output.txt`: contains the generated `output.txt` script (after all substitutions have been done)
- `launch.txt`: contains the generated `launch.txt` script (after all substitutions have been done)
- `patch`: a list of patches' names, one per row (corresponding to those described in *Patches*)
- `tau_min`: the minimum τ value set by the user
- `user_cuts`: user defined cuts and cuts values, one per row in the format `cut=value` (cuts are those defined in *Cuts*)

Note: It is guaranteed that the keys listed above are always present in grid's metadata (even if some of the corresponding values might be empty).

Output

- `DATASET`: The directory created by `mg5_aMC`. A few interesting files in this subdirectory are:
  - `Events/-/MADatNLO.HwU`: histograms with uncertainties (HwU)
  - `Events/-/amcblast_obs_-.pineappl`: grids created by `mg5_aMC`, not yet merged together
- `output.txt`: Run card for the 'output' phase, with all variables substituted to their final values
- `output.log`: Output of the external runner during the 'output' phase
- `launch.txt`: Run card for the 'launch' phase, with all variables substituted to their final values
- `launch.log`: Output of the external runner during the 'launch' phase
- `pineappl.convolute`: Output of `pineappl convolute`
- `pineappl.orders`: Output of `pineappl orders`
- `pineappl.pdf_uncertainty`: Output of `pineappl pdf_uncertainty`

1.1.9 Yadism

Runcard structure

- The `observable.yaml` file (compulsory). This file contains the description of the observable requested (kind and kinematics), together with further parameters specifying the process, and the details of the yadism calculation

Output

- `DATASET.yaml`: is the other yadism output format, fully human readable (but a bit verbose)

1.1.10 pinefarm package

Produce theory predictions from external sources.

Subpackages

pinefarm.cli package

Provide CLI.

Submodules

pinefarm.cli.configs module

Print loaded configurations.

`pinefarm.cli.configs.`**`main`**`()`

Print loaded configurations.

pinefarm.cli.info module

Inspect runcards.

pinefarm.cli.install module

Install utilities.

pinefarm.cli.list module

List available resources.

`pinefarm.cli.list.`**`main`**`(`*path*, *files=False*, *prefix=''*`)`

List available resources.

pinefarm.cli.merge module

Merge multiple PineAPPL grids into a single one.

pinefarm.cli.merge.**main**(*grids*)

Merge multiple PineAPPL grids into a single one.

pinefarm.cli.run module

Compute a dataset and compare using a given PDF.

pinefarm.cli.run.**install_reqs**(*runner*, *pdf*)

Install requirements.

Parameters

- **runner** (interface.External) – runner instance
- **pdf** (*str*) – pdf name

pinefarm.cli.run.**main**(*dataset*, *theory*, *pdf*)

Compute a dataset and compare using a given PDF.

Parameters

- **dataset** (*str*) – dataset name
- **theory** (*dict*) – theory dictionary
- **pdf** (*str*) – pdf name

pinefarm.cli.run.**run_dataset**(*runner*)

Execute runner and apply common post process.

Parameters **runner** (interface.External) – runner instance

pinefarm.cli.update module

Update datasets metadata.

pinefarm.cli.update.**main**(*datasets*)

Update datasets metadata.

pinefarm.external package

Provided interfaces.

Subpackages

pinefarm.external.mg5 package

Madgraph interface.

pinefarm.external.mg5.**CONVERT_MODEL = '\nset auto_convert_model True\nimport model loop_qcd_qed_sm_Gmu\nquit\n'**

Instructions to set the correct model for MG5aMC@NLO.

class pinefarm.external.mg5.**Mg5**(**args*, ***kwargs*)

Bases: *pinefarm.external.interface.External*

Interface provider.

collect_versions()

Add versions.

generate_pineappl()

Generate grid.

static install()

Execute installer.

property mg5_dir

Return output dir.

property pdf_id

Convert PDF to SetIndex.

results()

Collect PDF results.

run()

Execute program.

pinefarm.external.mg5.**URL = 'https://launchpad.net/mg5amcnlo/{major}.0/{major}.{minor}.x/+download/MG5_aMC_v{version}.tar.gz'**

URL template for MG5aMC@NLO release

pinefarm.external.mg5.**VERSION = '3.4.2'**

Version in use

pinefarm.external.mg5.**apply_user_cuts**(*cuts_file*, *user_cuts*)

Apply a user defined cut, patching a suitable cuts file.

pinefarm.external.mg5.**find_marker_position**(*insertion_marker*, *contents*)

Find in file.

pinefarm.external.mg5.**url**()

Compute actual download URL.

Submodules

pinefarm.external.mg5.paths module

Define additional local paths.

Submodules

pinefarm.external.integrability module

Integrability interface.

class pinefarm.external.integrability.**Integrability**(**args*, ***kwargs*)

Bases: *pinefarm.external.interface.External*

Interface provider.

collect_versions()

Add the version defined by this file.

generate_pineappl()

Generate the pineappl grid for the integrability observable.

results()

Apply PDF to grid.

run()

Empty function.

pinefarm.external.integrability.**evolution_to_flavour**(*evol_fl*)

Use eko to transform an evolution pid (>100) to flavour basis in a pineappl-usable way (flavour, weight).

pinefarm.external.integrability.**is_integrability**(*name*)

Determine whether this is an integrability dataset.

The decision is based on the existence of *integrability.yaml*.

pinefarm.external.interface module

Abstract interface.

class pinefarm.external.interface.**External**(*name*, *theory*, *pdf*, *timestamp=None*)

Bases: abc.ABC

Interface class for external providers.

Parameters

- **name** (*str*) – dataset name
- **theory** (*dict*) – theory dictionary
- **pdf** (*str*) – PDF name
- **timestamp** (*str*) – timestamp of already generated output folder

annotate_versions()

Add version informations as meta data.

abstract collect_versions() → dict

Collect necessary version informations.

Returns program - version mapping related to programs specific to a single runner (common ones are already abstracted)

Return type dict

abstract generate_pineappl()

Generate PineAPPL grid and extract output.

Returns output of `pineappl convolute` on the generate grid and selected `pdf`

Return type str

property grid

Target PineAPPL grid name.

property gridtmp

Intermediate PineAPPL grid name.

static install()

Install all needed programs.

load_pinecard() → str

Load directory as b64encoded .tar.gz file.

postprocess()

Postprocess grid.

abstract results()

Results as computed by the program.

Returns standardized dataframe with results (containing `result`, `error`, `sv_min`, and `sv_max` columns)

Return type pandas.DataFrame

abstract run()

Execute the program.

property source

Runcard base directory.

update_with_tmp()

Move intermediate grid to final position.

pinefarm.external.positivity module

Positivity interface.

class pinefarm.external.positivity.**Positivity**(**args*, ***kwargs*)

Bases: *pinefarm.external.interface.External*

Interface provider.

collect_versions()

No additional programs involved.

generate_pineappl()

Generate grid.

results()

Apply PDF to grid.

run()

Open configuration.

pinefarm.external.positivity.**is_positivity**(*name: str*) → bool

Determine whether this a positivity dataset.

pinefarm.external.vrap module

Runner for vrap producing pineappl grids.

Uses a modified version of vrap https://github.com/NNPDF/hawaiian_vrap which produces pineappl grids. It has been tested for FT DY kinematics

A `vrap` dataset includes a single `vrap.yaml` file which defines how `vrap` will be run. A dataset can include many kinematic files (min. 1), `vrap` will be run once per kinematic file. Vrap datatasets can also include `cfactors` which need to match the name of the kinematic files and will be applied to the corresponding run ex: if the kinematic file is call "906_bin0.dat" the corresponding cfactors

are "ACC_906_bin0.dat" and "QCD_906_bin0.dat"

class pinefarm.external.vrap.**Vrap**(*pinecard*, *theorycard*, **args*, ***kwargs*)

Bases: *pinefarm.external.interface.External*

Interface provider.

collect_versions()

Add the version defined by this file.

generate_pineappl()

If the run contain more than one grid, merge them all.

static install()

Execute installer.

results()

Combinesthe results of the partial runs of vrap in order to compare with the generated grid.

run()

Run vrap for the given runcards.

After running vrap, the resulting grid will be optimized, cfactors (for instance, ACCEPTANCE factors) applied. The MC results for each run (writen to results.out) will be read.

pinefarm.external.vrap.**gen_pos_pdf**(*pdfname*, *base_pdf='NNPDF40_nnlo_as_01180'*)

Generate pdfname according to the rules in _POSITIVITY_PDFS.

pinefarm.external.vrap.**is_vrap**(*name: str*) → bool

Check whether this is a dataset to be run with vrap.

pinefarm.external.vrap.**yaml_to_vrapcard**(*yaml_dict*, *pdf*, *output_file*, *order='NLO'*)

Convert the dictionary from *vrap.yaml* file into a vrap runcard.

pinefarm.external.yad module

yadism interface.

class pinefarm.external.yad.**Yadism**(**args*, ***kwargs*)

Bases: *pinefarm.external.interface.External*

Interface provider.

collect_versions()

No additional programs involved.

generate_pineappl()

Generate grid.

property output

Return yadism output path.

results()

Apply PDF to output.

run()

Run program.

pinefarm.external.yad.**is_dis**(*name: str*) → bool

Determine whether this is a DIS dataset, i.e. is yadism needed to run.

The decision is based on the existance of the *observable.yaml* file.

Submodules

pinefarm.configs module

Configuration tools.

pinefarm.configs.**NAME = 'pinefarm.toml'**

Name of the config while (wherever it is placed).

pinefarm.configs.**PATHS_SECTIONS = ('paths', 'commands')**

Sections containing only paths.

pinefarm.configs.**basic_paths**(*root: pathlib.Path*) → dict

Build all default independent paths.

Independent on anything but `root`.

pinefarm.configs.**commands**(*paths: dict*) → dict

Set all default commands.

pinefarm.configs.**configs = {}**

Holds loaded configurations.

pinefarm.configs.**detect**(*path: Optional[os.PathLike] = None*) → pathlib.Path

Detect configuration files.

Parameters **path** (*os.PathLike or None*) – optional explicit path to file to be used as configs (default: *None*)

Returns configuration file path

Return type pathlib.Path

Raises **FileNotFoundError** – in case no valid configuration file is found

pinefarm.configs.**force_paths**()

Convert values in chosen sections to paths.

pinefarm.configs.**load**(*path: Optional[os.PathLike] = None*) → dict

Load configuration file.

If no path is explicitly passed, a minimal configuration is used instead (just setting root folder to the current one).

Parameters **path** (*os.PathLike or None*) – the path to the configuration file (default: *None*)

Returns loaded configurations

Return type dict

pinefarm.configs.**nestupdate**(*base: dict*, *update: dict*)

Merge nested dictionaries.

Pay attention, *base* will be mutated in place. So the second one will overwrite the first.

Note: Modifying in place avoids a lot of copies. But not being performance intensive, it would be possible to obtain a non in-place alternative just adding a first line:

```
base = copy.deepcopy(base)
```

but it would be called at every recursion (the lots of copies above). A simpler alternative is just to copy before calling, if needed:

```
mycopy = copy.deepcopy(mydict)
nestupdate(mycopy, update)
```

that will make a single copy.

Note: Another option could appear to be using something like `collections.ChainMap`. This is a smart way to implement cascade configurations, but it is not going to support nesting:

```
configs = ChainMap({'a': {'b': 0}}, {'a': {'c': 1}})
```

in this case, even if there is no clash for `configs['a']['c']`, this would result in a `KeyError` (since once `configs['a']` is executed, the result is just a normal `dict`, and the first one encountered). Any refinement would involve a custom `__geitem__()`, with even more complicate logic.

Parameters

- **base** (*dict*) – dictionary to be updated
- **update** (*dict*) – dictionary containing update

pinefarm.configs.**paths**(*paths: dict*) → dict

Build all default dependent paths.

pinefarm.configs.**raw**(*original: dict*) → dict

Convert configs (or dict) into serializable equivalent.

Parameters **original** (*dict*) – original dictionary to convert

Returns converted dictionary

Return type dict

See also:

rawscalar(), used

pinefarm.configs.**rawscalar**(*value*)

Turn scalar into serializable equivalent.

Available conversions:

```
pathlib.Path -> str
```

Parameters **value** (*any*) – value to convert

Returns converted value, if no converter available the original one

Return type any

pinefarm.info module

Provide inspection tools.

class pinefarm.info.**Info**(*color: str*, *external: Type[*pinefarm.external.interface.External*]*, *kind:* pinefarm.info.Kind)

Bases: `object`

Info type.

color: `str`

external: `Type[pinefarm.external.interface.External]`

kind: *pinefarm.info.Kind*

class pinefarm.info.**Kind**(*value*)

Bases: enum.Enum

Interface types.

dis = 1

ftdy = 3

hadronic = 4

integrability = 5

positivity = 2

pinefarm.info.**label**(*dataset: str*) → *pinefarm.info.Info*

Generate associated Info type.

pinefarm.install module

Install tools.

pinefarm.install.**LHAPDF_VERSION = 'LHAPDF-6.4.0'**

Version of LHAPDF to be used by default (if not already available).

pinefarm.install.**PINEAPPL_REPO = 'https://github.com/N3PDF/pineappl.git'**

Git repo location for pineappl.

pinefarm.install.**cargo**()

Initialize Rust and Cargo.

Returns path to *cargo*

Return type str

pinefarm.install.**hawaiian_vrap**()

Install a version of vrap flavoured with pineappl from https://github.com/NNPDF/hawaiian_vrap.

Returns whether vrap is now installed

Return type bool

pinefarm.install.**init_prefix**()

Set up paths.

pinefarm.install.**is_exe**(*command: os.PathLike*) → bool

Check if given path exists and is executable.

pinefarm.install.**lhapdf**()

Install LHAPDF C++ library.

Not needed:

- for *mg5*, since it's vendored
- for *yadism*, since we depend on the PyPI version

pinefarm.install.**lhapdf_conf**(*pdf*)

Initialize LHAPDF.

Parameters **pdf** (*str*) – LHAPDF name of the required PDF

pinefarm.install.**mg5amc**()

Initialize MadGraph5_aMC@NLO.

Returns whether the main executable is now existing.

Return type bool

pinefarm.install.**pineappl**(*capi=True*, *cli=False*)

Initialize PineAPPL.

Parameters

- **capi** (*bool*) – whether to install PineAPPl CAPI (by default *True*, since it's the only thing required)
- **cli** (*bool*) – whether to install even PineAPPL CLI (by default *False*, since it's not required to run)

Returns whether *pineappl* and *pineappl_capi* are now available.

Return type bool

pinefarm.install.**update_environ**()

Adjust necessary environment files.

pinefarm.install.**update_lhapdf_path**(*path*)

Update LHAPDF path, both in environment and lhapdf_management.

Parameters **path** (*str or pathlib.Path*) – path to LHAPDF data

pinefarm.log module

Logging tools.

class pinefarm.log.**ChildStream**(*parent*)

Bases: object

Inner stream.

write(*data*)

Write to stream.

class pinefarm.log.**Tee**(*name*, *stdout=True*, *stderr=False*)

Bases: object

Context manager to tee stdout to file.

Parameters **name** (*str or pathlib.Path*) – path to redirect stdout to

flush()

Flush stream.

write(*data*, *stream*)

Write to stream.

exception pinefarm.log.**WhileRedirectedError**(**args*, *file*, ***kwargs*)

Bases: RuntimeError

Error to signal a generic error, while stderr was redirected to file.

Parameters

- ***args** – arguments passed to RuntimeError
- **file** (*str*) – path to file to which stderr is redirected
- ****kwargs** – keyword arguments passed to RuntimeError

pinefarm.log.**subprocess**(**args*, *cwd*, *out*)

Wrap subprocess.Popen to print the output to screen and capture it.

Parameters

- **args** – positional arguments to pass to *subprocess.Popen*
- **cwd** (*path-like or str*) – directory where to execute the command
- **out** (*path-like or str*) – file to which (also) redirect the output

Returns output of the command run in the subprocess

Return type str

pinefarm.table module

Comparison tools.

pinefarm.table.**convolute_grid**(*grid*, *pdf_name*, *integrated=False*)

Call *convolute* via PineAPPL CLI.

Parameters

- **grid** (*str*) – grid path
- **pdf_name** (*str*) – PDF name
- **integrated** (*bool*) – whether the bins have to be integrated with bins normalizations

Returns (essential) output splitted by line

Return type list(str)

pinefarm.table.**print_table**(*pineappl_results*, *external_results*, *dest*)

Print comparison table to screen.

Parameters

- **pineappl_results** (*pandas.DataFrame*) – results from the generated grid
- **external_results** (*pandas.DataFrame*) – results from the external program
- **dest** (*pathlib.Path*) – path to output file

pinefarm.tools module

Auxilariy tools.

pinefarm.tools.**common_substring**(*s1*, *s2*, **sn*)

Return the longest common part of two iterables, starting from the begininng.

Parameters

- **s1** (*Sequence*) – first sequence to compare
- **s2** (*Sequence*) – second sequence to compare
- ***sn** (*Sequence*) – any further sequence to compare

Returns longest common subsequence

Return type Sequence

Examples

```
>>> common_substring("test test", "test toast")
"test t"
>>> common_substring("test test", "test test test")
"test test"
>>> common_substring("test test", "")
""
>>> common_substring("test test", "test test test", "test")
"test"
>>> common_substring("test test", "test test test", "test toast")
"test t"
```

pinefarm.tools.**compress**(*path*)

Compress a file into lz4.

Parameters **path** (*pathlib.Path*) – input path

Returns path to compressed file

Return type pathlib.Path

pinefarm.tools.**create_output_folder**(*name*, *theoryid*)

Create output folder.

Parameters **name** (*str*) – dataset name

Returns path to output folder

Return type pathlib.Path

pinefarm.tools.**decompress**(*path*)

Decompress a file from lz4.

Parameters **path** (*pathlib.Path*) – path to compressed file

Returns path to raw file

Return type pathlib.Path

pinefarm.tools.**git_pull**(*repo*, *remote_name='origin'*, *branch='master'*)

Pull a remote repository.

Parameters

- **repo** (*pygit2.Repository*) – repository
- **remote_name** (*str*) – remote name
- **branch** (*str*) – branch to get

pinefarm.tools.**nine_points = [(0.5, 0.5), (0.5, 1.0), (0.5, 2.0), (1.0, 0.5), (1.0, 1.0), (1.0, 2.0), (2.0, 0.5), (2.0, 1.0), (2.0, 2.0)]**

Nine points prescription for scale variations (as couples, referred to `(fact, ren)` scales).

pinefarm.tools.**parse_metadata**(*file*)

Parse metadata file.

Parameters **file** (*io.TextIOBase*) – the file to read

Returns the metadata entries

Return type dict

pinefarm.tools.**patch**(*patch*, *base_dir='.'*)

Apply patch.

Parameters

- **patch** (*str*) – patch to apply (text content, not path)
- **base_dir** (*str or pathlib.Path*) – path to dir where to apply patch (default: `.`)

pinefarm.tools.**print_time**(*t0*, *what=None*)

Report completion together with timing to the user.

Parameters

- **t0** (*int*) – start time
- **what** (*str*) – subject that is completed

pinefarm.tools.**set_grid_metadata**(*grid*, *entries=None*, *entries_from_file=None*)

Set metadata on a pineappl grid (in-place operation).

Parameters

- **input_grid** (*pineappl.grid.Grid*) – input grid on which to set metadata
- **entries** (*dict*) – mapping of key-value to store in the grid
- **entries_from_file** (*dict*) – mapping of key-value pairs, whose value are file paths of which storing the content

pinefarm.tools.**three_points = [0.5, 1.0, 2.0]**

Three points prescription for scale variations.

pinefarm.tools.**update_grid_metadata**(*input_file*, *output_file*, *entries=None*, *entries_from_file=None*)

Set metadata on a pineappl grid stored in a file, and save in a new one.

Parameters

- **input_file** (*path-like*) – file storing input grid
- **output_file** (*path-like*) – file to store the result
- **entries** (*dict*) – mapping of key-value to store in the grid
- **entries_from_file** (*dict*) – mapping of key-value pairs, whose value are file paths of which storing the content

1.1.11 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

K

L

M

N

O

P

R

S

T

U

V

W

Y