`nnlojet-combine` documentation

Alexander Huss

19th January 2018

Contents

1 Introduction

This document is intended to describe the new combine script to merge multiple runs and different parts of the calculation into a final result.

2 Prerequisites & Setup

The combine script is written entirely in `python` and requires:

- `python3`
- `numpy` (python module)[1]

Auto-generated plotscripts (`*.plt`) need `gluplot` version ≥ 5.

All relevant files are in `/path/to/nnlojet/driver/bin` and the executable files *should not be copied* elsewhere to perform the combine[2]. Instead, add the bin directory to your `PATH` variable:

```
export PATH=/path/to/nnlojet/driver/bin:$PATH
```

to make the combine script available globally. It is recommended to make this setting permanent by adding the above line to your bashrc file (`~/.bashrc`).

If everything is set up properly, running `nnlojet-combine.py -h` in the terminal should give you a help screen like this:

```
$ nnlojet-combine.py -h
usage: nnlojet-combine.py [-h] [-C CONFIG] [-j [JOBS]] [--APPLfast APPLFAST]

Merge histogram files

optional arguments:
  -h, --help            show this help message and exit
  -C CONFIG, --config CONFIG
                        configuration file for the combine
  -j [JOBS], --jobs [JOBS]
                        Specifies the number of jobs to run simultaneously.
  --APPLfast APPLFAST   Read in an APPLfast weight table and combine.
```

3 Quick Start

The `nnlojet-combine.py` script *must not be edited* and instead all settings will be provided in a separate steering file which is read in by the script. A minimal steering file is given in `/path/to/nnlojet/driver/bin/combine.ini` and this is the file that should be copied into the folder where you want to do the combination. For the purpose of this quick-start tutorial we assume a folder structure as shown in Fig. 1.

The `combine.ini` steering file contains the information for the script to find the relevant datasets and how to assemble them into the final result:

[1] Install using "`pip install numpy`" or "`pip3 install numpy`" if not available on your system.

[2] Any settings that are "local" to your combination are instead stored in a steering file (default: `combine.ini`) which will be read in by the combine script.

```
[Paths]
raw_dir = raw
out_dir = combined

[Observables]
ALL

[Parts]
LO
V
R
VV
RV
RR

[Final]
LO    = LO
NLO   = LO + V + R
NNLO  = LO + V + R + VV + RV + RR
NLO_only  = V + R
NNLO_only = VV + RV + RR
```

The steering file is divided into different sections:

`[Paths]` In this block we specify two paths (absolute or relative to the location where the combine script is executed): The location of the raw data files (`raw_dir`) and the output directory (`out_dir`).

`[Observables]` Specifying "`ALL`" here will make the script scan the histogram file names for observables automatically. Alternatively, one can give a list of observables (one on each line) which should be considered. This is useful if only a subset is of interest and then we can save time in the merge.

`[Parts]` Here, we specify the folders within the `raw_dir` directory, which correspond to the different parts of the calculation. Each of these folders will then be scanned for `*.dat` files and combined. To scan the directory *recursively* for histogram files, see Sect. 6.2.

`[Final]` File names for the final histogram files and how the different parts are assembled into them is defined here. Names appearing on the r.h.s. must be items defined in the "`[Parts]`" section (or in the "`[Merge]`" section, see Sect. 5).

So far, only "+" supported, maybe "−" useful too?

Running "`nnlojet-combine.py`" will then create the output directory ("`combined`" here) with further sub-directories as shown in Fig. 1:

`./combined/Parts` A merge is performed for the different parts separately using all the the runs found in the corresponding folder. The output of this is stored in the "Parts" sub-folder.

`./combined/Final` The sum of the different parts, as defined in the "`[Final]`" section of the steering file, is performed using the histograms generated in the "Parts" sub-folder. The resulting histograms are stored in the "Final" sub-folder.

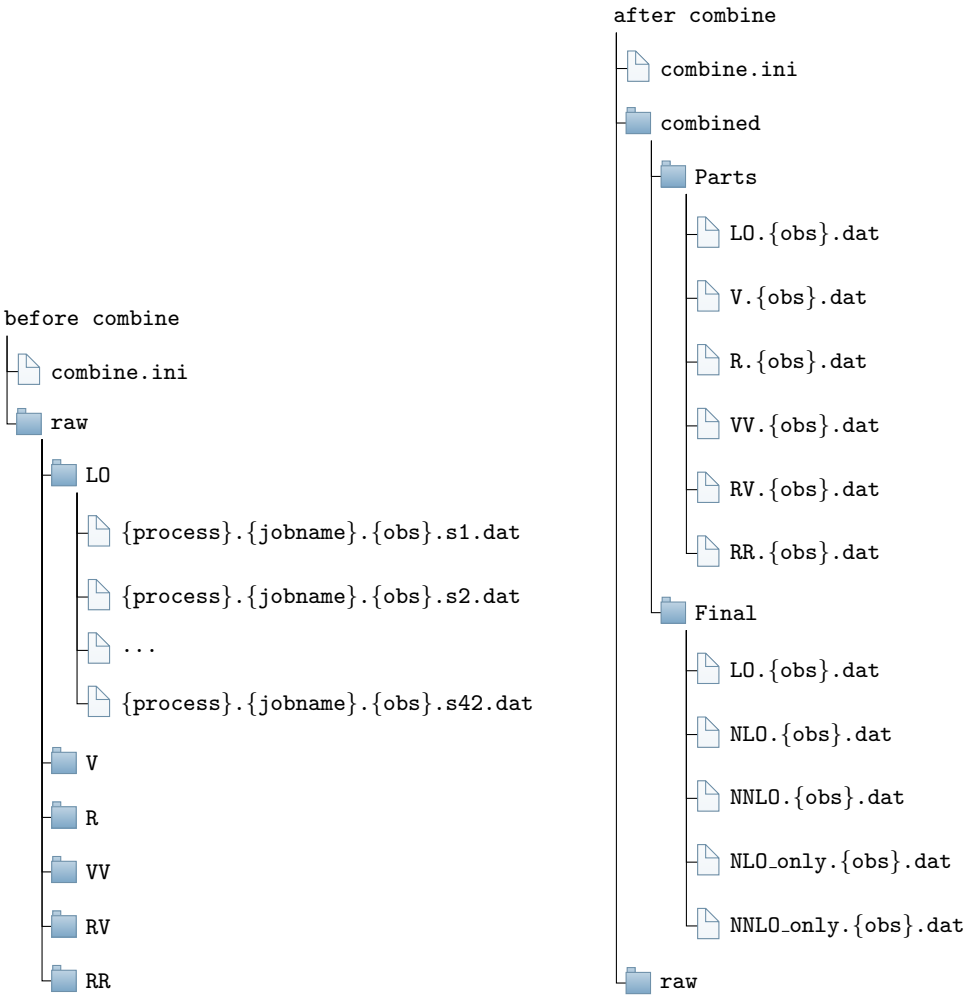

Figure 1: Example folder structure before (left) and after (right) running the combination.

4 More Details

4.1 Comments

Comments in the steering file start with `#`. With an Observables block like this:

```
[Observables]
# cross
xm12_y1
# xm12_y2
```

only a merge for the observable `xm12_y1` is performed. Similarly, one can comment out lines in the "`[Parts]`" block if the raw data has not changed and one can skip the merging step of the corresponding part.

4.2 Sub-directories

Elements in the "`[Parts]`" block are in fact paths (relative to `raw_dir`) and we can therefore also write something like this

```
[Parts]
H1-LQall-8.V/ptavg_12_q2_5p5_8
H1-LQall-8.R/ptavg_12_q2_5p5_8
```

In this case, the corresponding merged files written into `combined/Parts` will re-create the same directory structure.

4.3 Alias

The use of sub-directories, as described in the previous subsection, might not be something we actually want. It certainly makes the declaration of the "`[Final]`" block more ugly with path names on the r.h.s. However, everyone has a different way of organising their runs into folders and we prefer to have maximal flexibility on the combine side than forcing the user to move raw datasets into new folder structures. For this, we can assign an alias for the folder given in the "`[Parts]`" block and use the alias in the "`[Final]`" part like this:

using paths directly:

```
[Parts]
H1-LQall-8.V/ptavg_12_q2_5p5_8
H1-LQall-8.R/ptavg_12_q2_5p5_8

[Final]
NLO_only  = H1-LQall-8.V/
    ptavg_12_q2_5p5_8 + H1-LQall
    -8.R/ptavg_12_q2_5p5_8
```

using aliases:

```
[Parts]
H1-LQall-8.V/ptavg_12_q2_5p5_8 : V
H1-LQall-8.R/ptavg_12_q2_5p5_8 : R

[Final]
NLO_only  = V + R
```

The histogram files generated in the `./combined/Parts` folder will also use the alias name as the file prefix.

4.4 Steering File

By default, the script will look for a file `combine.ini` as the steering file. Using the flag "`-c`" we can manually provide the name of the steering file:

```
nnlojet-combine.py -C another_combine.ini
```

This can become useful if there are multiple combines to be performed on the datasets contained in the same raw-data directory (different PDFs, low-Q^2 & high-Q^2 in DIS, ...).

4.5 Multi-threading

Add the option `-j {num_workers}` or `--jobs {num_workers}` to the execution of the script to spawn `{num_workers}` processes to perform the combine in parallel. Dropping the number of workers as an argument will automatically determine the number of (physical) cores on your machine and spawn that many worker processes.

4.6 Auto-generate skeleton for the steering file

todo: auto-generate a skeleton by scanning a raw folder. Something along the line of `nnlojet-combine.py --raw_dir {raw_dir} -o combine.ini`

4.7 Plots

todo: auto-generate plots like in my old toolchain: differential breakdown into the parts, differential breakdown of absolute errors, compatibility of merged files, etc. These are essential plots to assess the consistency of the final results and to determine Parts that are require more statistics...

5 Merging

Often times, multiple data sets are produced for the same Part using different run conditions. Typical examples are:

(A) data sets using different settings

- variation of `y0cut` for checks
- different reweighting functions to have dedicated datasets that are more precise in certain kinematic regions
- ...

(B) division of a distribution into distinct / overlapping slices

- p_{T} distribution divided into individual runs for: [2,7], [5,15], [10,100] GeV
- ...

Merging such datasets is a very common task and therefore it makes sense to directly provide this within the combine script. This is specified in an optional section "`[Merge]`" of the steering file.

5.1 Weighted Combination

For case (A), we consider the situation of different `y0cut` values:

```
[Parts]
V
R/y0cut_1d-6 : R6
R/y0cut_1d-7 : R7
R/y0cut_1d-8 : R8

[Merge]
R_mrg = R6 & R7 & R8

[Final]
NLO_only = V + R_mrg
```

Here, `R6`, `R7`, and `R8` are first combined separately. The operator "`&`" used in the "`[Merge]`" section then specifies that a *weighted* average should be performed using these datasets and saved as `R_mrg`. We can then use this alias in the "`[Final]`" section to assemble our final result.

The script will further use the specification "`R_mrg = R6 & R7 & R8`" of the steering file to auto-generate debug plots. These plots can be used to check for the compatibility of the components that are merged together. In the example above, if the dataset `R/y0cut_1d-6` turns out to be inconsistent with the rest due to a too large `y0cut` value, one can then drop it from the merge: `R_mrg = R7 & R8` and re-run the combine script.

plot generation still to do

5.2 Combine Slices

For case (B), we would like to stitch together datasets that correspond to different patches of a distribution. It's typically a good idea to do these runs in such a way that there are overlapping bins between the datasets to check for consistency. We use the example of the p_{T} distribution divided into runs for: [2,7], [5,15], [10,100] GeV

```
[Parts]
V
R/pt_2_7     : R_2_7
R/pt_5_15    : R_5_15
R/pt_10_100 : R_10_100

[Merge]
R_mrg = R_2_7 | R_5_15 | R_10_100

[Final]
NLO_only = V + R_mrg
```

Here, `R_2_7`, `R_5_15`, and `R_10_100` are first combined separately. The operator "`|`" used in the "`[Merge]`" section then specifies that we wish to stitch these partial results together into a final `R_mrg` result. For the overlapping regions, the dataset that comes "later" (is to the right) has precedence. The order in which we specify the merge is therefore important and for the example above we have:

2 – 5 GeV: dataset "`R_2_7`"

5 – 10 GeV: dataset "`R_5_15`"

10 – 100 GeV: dataset "`R_10_100`"

ATTENTION: If the cuts in the runs are not aligned to bin edges in the histogram it can happen that the first/last bin in the dataset is not correct. TODO: For this, we implement a post-processing feature that allows the user to make the script strip away the first and/or last bin of a dataset.

todo: auto-generation of debug plots.

5.3 Do Both

It is not possible to specify a single line in the "`[Merge]`" section that uses *both* the operator "`&`" and "`|`". There are cases where such a thing is needed though and there are two ways of doing it:

```
[Parts]
R/y0cut_1d-6_pt_2_7     : R6_2_7
R/y0cut_1d-7_pt_2_7     : R7_2_7
R/y0cut_1d-6_pt_5_15    : R6_5_15
R/y0cut_1d-7_pt_5_15    : R7_5_15

[Merge]
R6_mrg = R6_2_7 | R6_5_15
R7_mrg = R7_2_7 | R7_5_15
R_mrg = R6_mrg & R7_mrg
```

```
[Parts]
R/y0cut_1d-6_pt_2_7     : R6_2_7
R/y0cut_1d-7_pt_2_7     : R7_2_7
R/y0cut_1d-6_pt_5_15    : R6_5_15
R/y0cut_1d-7_pt_5_15    : R7_5_15

[Merge]
R_2_7_mrg   = R6_2_7  & R7_2_7
R_5_15_mrg = R6_5_15 & R7_5_15
R_mrg = R_2_7 | R_5_15
```

5.4 Simple Sum

There are cases where a sum of Parts becomes necessary in the pre-processing step, which can be done in the "`[Merge]`" section like this:

```
[Parts]
V_sub
V_p2b
R_sub
R_p2b

[Merge]
SUB_NLO_only = V_sub + R_sub    ! NLO using antenna subtraction
P2B_NLO_only = V_p2b + R_p2b    ! NLO using projection-to-Born
NLO_only = SUB_NLO_only & P2B_NLO_only    ! "best" NLO prediction
```

6 Optional settings

In addition to the compulsory sections in the steering file described in Sect. 3, we can further control the behaviour of the combine script by an optional section "`[Options]`".

6.1 Combine settings

Although the default settings of the combine script should be "good" (more on the conservative side), there will be cases where the user will want to adjust parameters to control

the merging prescription. This is useful to see if the default settings are too aggressive for the given dataset or if one might be able to squeeze out more from the raw the data. The merging proceeds in three steps, which are applied for each histogram separately on a bin-by-bin basis:

1. 'trim': We apply an outlier-rejection procedure based on the inter-quartile-range (robust statistics) to discard data-points which are identified as outliers.

2. 'k-scan': on the trimmed dataset, we successively merge individual jobs (unweighted) into pseudo-datasets until we observe a plateau.

3. 'weighted': The pseudo-runs are then merged using a weighted average.

All internal parameters can be controlled through declarations in the "`[Options]`" block:

`trim = (threshold, max_frac)`

- `threshold`: Threshold above which data points are trimmed away. Larger values means less trimming (default = 4).
- `max_frac`: Dynamically increase threshold until the ratio of trimmed data is below this value (default = 0.1, can also be 'None').

`k-scan = (maxdev_unwgt, nsteps, maxdev_steps)`

- `maxdev_unwgt`: If this variable is not '`None`' we first produce a reference result using an unweighted average and perform a successive k-merging until the result of the k-merging lies within `maxdev_unwgt`$\times\sigma_{\text{unwgt}}$ of the unweighted reference (default = `None`).
- `nsteps, maxdev_steps`: In the k-merging, we successively combine pairs of pseudo-runs. This means that at each step the number of pseudo-runs decreases by a factor of $\frac{1}{2}$ at each step. If `nsteps, maxdev_steps` are not `None`, we keep track of the *previous* `nsteps` results and check for their consistency and therefore look for a plateau. More precisely, we check that the previous `nsteps` results are within `maxdev_steps`$\times\sigma_{\text{current}}$ of the current result of the k-merging. We therefore need at least `nsteps`+1 steps to be able to check for this termination condition (default = 2,0.5).

`weighted = <bool>` After we have combined individual runs into pseudo-runs, such that we actually trust the error estimates, we perform a weighted average over them. This can be turned off by setting this flag to "`False`" (default = `True`). Clearly, the whole k-merging is completely pointless if we combine things in an unweighted manner in the end. If this flag is set to `False` one should also switch off the k-merging by setting the respective values to `None` in order to save computing time.

Let me stress one more time that this procedure is applied on each single bin of every histogram *separately*. Depending on the kinematic region, certain bins can be more prone to outliers and also receive more/less statistics and the optimal value for k can differ between bins.

6.2 Recursive search

If we want to search the paths given in the "`[Parts]`" section *recursively* for `*.dat` files, we can set the flag as follows:

```
[Options]
recursive = True
```

By default, recursive search is turned off for performance reasons.

6.3 Plot

If we want to auto-generate some debug gnuplot scripts, we can turn this feature on by adding the corresponding option:

```
[Options]
plot = True
```

The gnuplot scripts (`*.plt` files) are generated inside the "`Plot`" folder inside `out_dir`.

6.4 Output weight table

In order to fully mimic our combine procedure on the `APPLfast` side, we require the output of a weight table containing bin-by-bin factors for each raw run. We can switch on this output with an optional flag like this:

```
[Options]
weights = True
```

By default, the output of weight tables is off for performance reasons.

The script will produce two types of files containing the weight information:

`*.npz`: This file is generated for every entry in the "`[Parts]`", "`[Merge]`", and "`[Final]`" section. It contains the *full* weight information (including all columns/channels) in a binary file as generated through `numpy.savez`.

`*.APPLfast.txt`: This file outputs the weight table for the `APPLfast` merging and for the final results defined in the "`[Final]`" section. Each line in this text file starts with the raw-data file name followed by a white-space-separated list of weight factors to be applied on the corresponding x-bin. Note that the weight factor is derived from the first data column in the histograms, i.e. for the total cross section at the central scale. Using this set of weight factors for the other scales reproduces the "real" final result to single-precision accuracy (~ 6 digits).

You can read in an `*.APPLfast.txt` file and perform a combination using the weights stored in there by running `nnlojet-combine.py --APPLfast {filename}.APPLfast.txt`. The resulting histogram output will be displayed into the standard output and can be easily redirected to a file if necessary.

6.5 Restrict merge to certain columns

To further speed up the combine, we might want to only merge certain columns of the data files. An example is when we're not interested in the full channel breakdown. In this case we can gain an order-of-magnitude speedup by only picking the "`tot`"-type columns:

```
[Options]
columns = ['tot_scale01', 'tot_scale02', 'tot_scale03', 'tot_scale04', '
    tot_scale05', 'tot_scale06', 'tot_scale07']
```

Only the column name associated with the values are specified *but not* the associated `*_Err` column names. The errors will be automatically read in as the adjacent column in the data file.

6.6 Rebin histograms

A rebinning of histograms is useful if we want to improve the statistics in some distributions by combining the bins. For this, we can register a "new" observable

```
[Observables]
observable   ! with the original binning
observable > observable_rebin : [x0, x1, x2, ... xN]
```

which uses the data-file for "observable" and performs a re-binning according to the array specified on the right. A new name is given to the re-binned observable using the operator "`>`" in order to avoid overwriting the files of the original observable. The re-binning is performed at the level of individual raw data files.

better to do this on the final merged files?!

6.7 More features

todo: ...