

Using pyjetty

Reynier Cruz Torres

November 17, 2021

Contents

1	Introduction	3
2	Processing	3
2.1	pp	3
2.2	Pb-Pb	4
3	Merging data root files	5
4	Scaling and merging MC root files	5
5	Analysis	6
5.1	Writing analysis code	6
5.1.1	Functions the user needs to implement	6
5.2	Running analysis code	7
5.3	What happens when you run the analysis code	7
5.3.1	Unfolding	8
5.3.2	Unfolding tests	9
5.3.3	Kinematic efficiency	9
5.4	Systematic Uncertainties	10
5.4.1	Track efficiency	10
5.4.2	Regularization parameter	10
5.4.3	Priors 1, 2	10
5.4.4	Truncation	10
5.4.5	Binning	10
6	Generating PYTHIA events within heppy	10

7	Folding theory calculations	11
7.1	Processing step	11
7.2	Analysis step	12
8	Plotting folded SCET calculations along with the data	12
9	Plotting ratio of Pb-Pb / pp distributions	12
10	Creating curves from generator data for theory comparison	13
11	Creating a txt file with all files in a subdirectory	13

1 Introduction

This document gives concrete examples on how to use pyjetty [1], which leverages the package heppy [2] and packages therein. Specifically, I will use the ‘jet_axis’ analysis as an example. For running other codes, replace the label ‘jet_axis’ with the appropriate identifier. Studies with pyjetty are done in two steps: 1) processing, and 2) analysis.

2 Processing

2.1 pp

Example on running a local pp data job:

```
python process/user/rej/process_data_jet_axis.py \  
-f /rstorage/alice/data/LHC17pq/448/20-06-2020/448_20200619-0610/unmerged/child_1/0001/AnalysisResults.root \  
-c config/jet_axis/rej_pp.yaml
```

Example on running a local pp MC job:

```
python process/user/rej/process_mc_jet_axis.py \  
-f /rstorage/alice/data/LHC18b8/520/child_1/TrainOutput/1/282008/0001/AnalysisResults.root \  
-c config/jet_axis/rej_pp.yaml
```

The measured dataset we are currently using corresponds to LHC17p and LHC17q pass1 AOD. We combine the FAST trigger cluster with the CENT woSDD trigger cluster, both of which are reconstructed without the SDD. Example on running a slurm data job:

```
cd slurm/sbatch/jet_axis/  
sbatch slurm_LHC17pq.sh
```

The analysis also uses the LHC18b8 Pythia8 $p_{T,\text{hard}}$ MC production (Monash 2013 tune) with the full GEANT3 ALICE detector simulation. The production consists of 20 $p_{T,\text{hard}}$ bins, each populated with approximately 6M events, with bin edges: [5, 7, 9, 12, 16, 21, 28, 36, 45, 57, 70, 85, 99, 115, 132, 150, 169, 190, 212, 235, 235+] GeV/ c . The MC is anchored run-by-run to LHC17pq runs. Example on running a slurm MC job:

```
cd slurm/sbatch/jet_axis/  
sbatch slurm_LHC18b8.sh
```

There are five different processing jobs that need to be submitted:

1. nominal processing for data

2. nominal processing for MC
3. processing for track efficiency (same as nominal MC, but randomly rejecting 4% of tracks for systematic uncertainty)
4. fast simulation with PYTHIA generator
5. fast simulation with HERWIG generator (these last two are combined to determine generator systematic uncertainty)

2.2 Pb-Pb

The measured dataset we are currently using corresponds to the LHC18q and LHC18r pass3 AOD. Example on running a slurm data job:

```
cd slurm/sbatch/jet_axis/PbPb/
sbatch slurm_LHC18qr.sh
```

The simulated dataset we are currently using corresponds to the LHC20g4 Pythia8 $p_{T,\text{hard}}$ MC production (Monash 2013 tune) with the full GEANT3 ALICE detector simulation. The production consists of 20 $p_{T,\text{hard}}$ bins, each populated with approximately 8M events, with bin edges: [5, 7, 9, 12, 16, 21, 28, 36, 45, 57, 70, 85, 99, 115, 132, 150, 101169, 190, 212, 235, 235+] GeV/ c . Example on running a slurm data job:

```
cd slurm/sbatch/jet_axis/PbPb/
sbatch slurm_LHC20g4_embedding.sh
```

Compared to pp, the Pb-Pb analysis is very heavy. First of all, we have much more Pb-Pb statistics than pp. Additionally, the MC embedding makes the process slower. Thus, for tests we can use 10% of the measured and simulated data. For these, use the following instead of the previous two sets of instructions:

```
cd slurm/sbatch/jet_axis/PbPb/
sbatch slurm_LHC18qr_10percent.sh
```

```
cd slurm/sbatch/jet_axis/PbPb/
sbatch slurm_LHC20g4_embedding_10percent.sh
```

There are six different processing jobs that need to be submitted:

1. nominal processing for data
2. nominal processing for MC (embedding PYTHIA event in PbPb background, rejecting 2% of tracks)

3. processing for track efficiency (same as nominal MC, but randomly rejecting 6% of tracks for systematic uncertainty)
4. fast simulation with PYTHIA generator
5. fast simulation with HERWIG generator (these last two are combined to determine generator systematic uncertainty)
6. thermal mode: embedding PYTHIA event in thermal background)

3 Merging data root files

- `cd pyjetty/pyjetty/alice_analysis/slurm/utils/rej`
- open the file `merge_data.sh`

```
#!/bin/bash
#
# Script to merge output ROOT files
JOB_ID=209383
FILE_DIR="/rstorage/alice/AnalysisResults/rej/$JOB_ID"
FILES=$( find "$FILE_DIR" -name "*.root" )
OUTPUT_DIR=/rstorage/alice/AnalysisResults/rej/$JOB_ID
hadd -f -j 20 $OUTPUT_DIR/AnalysisResultsFinal.root $FILES
```

- edit this file and replace `rej` with your username and edit the number in `JOB_ID=209383` with the correct run number that you would like to merge
- `source merge_data.sh`

4 Scaling and merging MC root files

The MC files are produced separate in \hat{p}_T bins. This is done to focus the generation in different bins and accrue similar amount of statistics even in the bins where the cross section is small. Consequently, these files need to be scaled by the cross section before combining them.

1. hadd all root files corresponding to the same \hat{p}_T bin. See, for example: [slurm_merge_LHC18b8.sh](#) and [merge_LHC18b8.sh](#). Edit both files and replace `rej` with the appropriate username. Also, modify the number in `JOB_ID=209384` in `merge_LHC18b8.sh` to reflect the right job number.

```
sbatch slurm_merge_LHC18b8.sh
```

2. cd into the directory containing the 1/, 2/, ... sub-directories and scale the combined files corresponding to a given \hat{p}_T bin by the appropriate scale factor. To do so, run scaleHistograms.py (with the correct file path) and config file associated with the simulation, e.g. -c /rstorage/alice/data/LHC18b8/scaleFactors.yaml. Here's an example:

```
python /home/rej/pyjetty/pyjetty/alice_analysis/slurm/utils/rej/scaleHistograms.py \  
-c /rstorage/alice/data/LHC18b8/scaleFactors.yaml
```

The path given above is specifically for the PYTHIA8 + GEANT3 simulations. The paths for the fast PYTHIA8 and fast HERWIG7 simulations are:

```
/rstorage/generators/pythia_alice/tree_fastsim/scaleFactors.yaml  
/rstorage/generators/herwig_alice/tree_fastsim/scaleFactors.yaml
```

respectively. For the Pb-Pb full simulations, the scale factors are here:

```
/rstorage/alice/data/LHC20g4/scaleFactors.yaml
```

In the case of the HERWIG7 fast simulation, we observed some outliers. To clean them, do:

```
python /home/rej/pyjetty/pyjetty/alice_analysis/slurm/utils/rej/scaleHistograms_fastHerwig.py
```

after the previous step.

3. After the histograms have been scaled, you should merge the \hat{p}_T bins. See for example [merge_pthat.sh](#). The number in the line JOB_ID=209384 and paths should be updated. Then do:

```
source merge_pthat.sh
```

5 Analysis

5.1 Writing analysis code

You need to begin by creating a code in: pyjetty/pyjetty/alice_analysis/analysis/user/. This code will inherit from [/substructure/run_analysis.py](#). For an example analysis code see: [run_analysis_jet_axis.py](#).

5.1.1 Functions the user needs to implement

There are three main functions the user needs to implement in the analysis code:

- `plot_single_result()`
- `plot_all_results()`
- `plot_performance()`

The function names are self-explanatory.

You also need to edit two functions in [analysis/user/substructure/analysis_utils_obs.py](#): `formatted_subobs_label` and `prior_scale_factor_obs`.

5.2 Running analysis code

```
python analysis/user/rey/run_analysis_jet_axis.py -c config/jet_axis/rey_pp.yaml
```

5.3 What happens when you run the analysis code

Right away, what the code does is it runs the ‘main’ function [run_analysis\(\)](#) defined in `run_analysis.py`. The first step in this function is to do unfolding (if the user requested it) through the function [perform_unfolding\(\)](#), also defined in `run_analysis.py`.

This function loops over the ‘systematic’ settings defined in the config file. For each setting, the code sets variables related to inputs and outputs:

```
output_dir = getattr(self, 'output_dir_{}'.format(systematic))
data = self.main_data
response = self.main_response
main_response_location = os.path.join(getattr(self, 'output_dir_main'), 'response.root')
rebin_response = self.check_rebin_response(output_dir)
```

It then initializes some variables:

```
prior_variation_parameter = 0.
truncation = False
binning = False
R_max = self.R_max
prong_matching_response = False
```

And it finally sets these variables depending on the systematic setting to be unfolded:

```

if systematic == 'trkeff':
    response = self.trkeff_response
elif systematic == 'prior1':
    prior_variation_parameter = self.prior1_variation_parameter
elif systematic == 'prior2':
    prior_variation_parameter = self.prior2_variation_parameter
elif systematic == 'truncation':
    truncation = True
elif systematic == 'binning':
    binning = True
elif systematic == 'subtraction1':
    R_max = self.R_max1
elif systematic == 'subtraction2':
    R_max = self.R_max2
elif systematic == 'prong_matching':
    prong_matching_response = True

```

Once these variables have been properly set, the code [creates an instance of the Roounfold_Obs class](#), and subsequently runs the function [roounfold_obs\(\)](#).

5.3.1 Unfolding

The Roounfold_Obs class and the roounfold_obs() function are defined in [roounfold_obs.py](#).

When the instance of the Roounfold_Obs class is created, the function `create_output_dirs()` is called. This function creates the following directories: 'RM', 'Data', 'KinematicEfficiency', 'Unfolded_obs', 'Unfolded_pt', 'Unfolded_ratio', 'Unfolded_stat_uncert', 'Test_StatisticalClosure', 'Test_Refolding', 'Correlation_Coefficients' and if the variable `thermal_model` is true, 'Test_ThermalClosure'. Also, two directories called 'Test_ShapeClosure{' are created. Here, {' corresponds to the prior variation parameters defined at the bottom of the config file (with periods removed). For instance, if the config file has:

```

prior1_variation_parameter: 0.5
prior2_variation_parameter: -0.5

```

then you will get the directories 'Test_ShapeClosure-05' and 'Test_ShapeClosure05'.

The unfolding procedure is done two-dimensionally in the observable and in p_T . The response matrix corresponds to:

$$\Lambda = (p_{T,\text{det}}, p_{T,\text{true}}, \text{observable}_{\text{det}}, \text{observable}_{\text{true}}). \quad (1)$$

This matrix is then used to unfold the data using Bayes' theorem:

$$P(T|O, \Lambda) = \frac{P(O|T, \Lambda) \cdot P(T)}{P(O)}, \quad (2)$$

where $P(T|O, \Lambda)$ is the likelihood of the truth (T) occurring given that the observation (O) is true.

Similarly, $P(O|T, \Lambda)$ is the likelihood of O occurring given that T is true. $P(T)$ and $P(O)$ are the marginal probabilities of observing T and O , respectively.

5.3.2 Unfolding tests

During the unfolding procedure, three validation tests are carried out:

1. **Refolding test:** the Response Matrix (RM) is multiplied by the unfolded result, and compared to the original detector-level distribution. This is done in `roounfold_obs.py` in the `refolding_test` function. Before doing the refolding, the kinematic-efficiency correction is reverted. Then, the output plots are created in `plot_obs_refolded_slice`. In these plots, the folded truth level and the detector-level (i.e. pre-unfolding) data are compared.
2. **Statistical closure test:**
 - MC det-level is smeared by an amount equal to the measured statistical uncertainty
 - the smeared det-level MC is then unfolded
 - unfolded smeared MC is compared to MC truth-level

This test checks whether the unfolding procedure is insensitive to statistical fluctuations of the measured spectra. This is done in `roounfold_obs.py` in the `statistical_closure_test` function. Then, the output plots are created in `plot_obs_closure_slice`.

3. **Shape closure test:**

- MC det-level and MC truth-level spectra are scaled
- scaled MC det-level spectrum is unfolded
- MC truth-level and unfolded scaled MC det-level spectra are compared

This test checks whether the unfolding procedure is insensitive to the shape of the measured distribution. This is done in `roounfold_obs.py` in the `shape_closure_test` function, which calls the `shape_closure_test_single` function twice, once with each shape-variation parameter. Then, the output plots are created in `plot_obs_closure_slice`.

5.3.3 Kinematic efficiency

The kinematic efficiency is calculated in `roounfold_obs.py` in the `plot_kinematic_efficiency` function. 1D slices are plotted in `plot_kinematic_efficiency_projections`. In the case of the jet axis analysis, this is defined as:

$$\varepsilon_{\text{kin}}(\Delta R_{\text{true}}, p_{\text{T,true}}) \equiv \frac{\frac{dN}{d\Delta R_{\text{true}}}(\Delta R_{\text{det}} \in [0, R/2], p_{\text{T,det}} \in [10, 80])}{\frac{dN}{d\Delta R_{\text{true}}}(\Delta R_{\text{det}} \in [0, R/2], p_{\text{T,det}} \in [0, \infty])} \quad (3)$$

5.4 Systematic Uncertainties

By default, pyjetty extracts the following sources of systematics:

- track efficiency
- regularization parameter
- priors 1, 2
- truncation
- binning

The explanation for each of these systematics is given below.

5.4.1 Track efficiency

The uncertainty on the tracking efficiency is approximately 4% for hybrid tracks [3, 4, 5, 6]. To assign a systematic uncertainty that accounts for this effect, we construct a response matrix by randomly rejecting 4% of tracks in jet finding. The resulting response matrix is then used to unfold the data.

5.4.2 Regularization parameter

5.4.3 Priors 1, 2

5.4.4 Truncation

5.4.5 Binning

6 Generating PYTHIA events within heppy

I will use the jet-axis analysis as an example.

cd into `pyjetty/pyjetty/alice_analysis/slurm/sbatch/jet_axis` and do:

```
sbatch pythia_gen_jet_axis_slurm.sh
```

This shell script is running jobs over the code:

`pyjetty/pyjetty/alice_analysis/process/user/re/pythia_parton_hadron.py` After the jobs are finished, cd into: `/home/re/pyjetty/pyjetty/alice_analysis/slurm/utls/re/gen/`. To merge the subjobs for each pT-bins, edit the run number in `merge_pythia.sh`, and then do:

```
sbatch slurm_merge_pythia.sh
```

If people are using the cluster and you cannot wait, then do this last step locally by editing the run number in `local_merge_gen.sh`, and then doing:

```
source source_local_merge_gen.sh
```

Finally, merge the different pT-hat-bin files using the same method described in the last step of section 4. No scaling is needed, since this is already done in the `pythia_parton_hadron.py` code.

7 Folding theory calculations

Theory calculations may need some corrections in order to be compared to data. For instance, while the data will correspond to full jets or charged jets, the calculations may be at parton level. Additionally, while the data may contain background effects, the calculations may be background-free.

Just like in the actual data analysis, there are two steps for these corrections: 1) a processing step and 2) an analysis step. See more details [here](#).

7.1 Processing step

Here's an example on running interactively for testing on Pythia:

```
python process/user/rei/process_parton_hadron_jet_axis.py \  
-c rei_pp_R0.4_pT_20_40_GeV_WTA_testing_theory_folding.yaml \  
-f /rstorage/alice/sim/pythia_gen/504913/1/45/AnalysisResults.root
```

and here's an example on running interactively for testing on Herwig:

```
python process/user/rei/process_parton_hadron_jet_axis.py \  
-c rei_pp_R0.4_pT_20_40_GeV_WTA_testing_theory_folding.yaml \  
-f /rstorage/alice/sim/herwig_gen/505074/1/142/AnalysisResults.root
```

Slurm submission scripts can be found in `slurm/sbatch/jet_axis/pp/gen`. Do, for example:

```
sbatch herwig_deltaR_slurm_R04.sh
```

7.2 Analysis step

```
python analysis/user/rei/run_fold_theory_jet_axis.py \  
-c config/jet_axis/rei_pp_processing_and_theory_folding_R04.yaml
```

```
th_fold_observable: 'jet_axis'  
do_theory_comp: True  
theory_dir: '/home/rei/jet_axis_theory_calculations/data/'  
pt_scale_factors_path: '/home/rei/jet_axis_theory_calculations/data/'  
response_files: ["path/to/response1/AnalysisResultsFinal.root", "path/to/response2/AnalysisResultsFinal.root"]  
response_labels: ["PYTHIA8", "Herwig7"]  
response_levels: [['h', 'ch', 'off'], ['p', 'ch', 'on']]  
th_subconfigs: ['config1', 'config2', 'config3']  
theory_pt_bins: [40,45,50,55,60,65,70,75,80,85,90,95,100,105,110,115,120,125,130,135,140,145,150]  
final_pt_bins: [40,60,80,100]
```

You can also choose to have a parameter `theory_obs_bins` in the config file, *e.g.*:

```
theory_obs_bins: [0., 0.005, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035, 0.04, 0.045, 0.05, 0.055, \  
0.06, 0.07, 0.08, 0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2]
```

If you include it, the code will return the observable with this binning. If you don't include it, whatever bin is in the RM matrix for the observable will be used.

8 Plotting folded SCET calculations along with the data

```
cd analysis/user/rei/  
source sourceme_plot_jet_axis_scet_comp.sh
```

That should execute commands like:

```
python plot_jet_axis_scet_comp.py \  
-c ../../../../config/jet_axis/configs_broken_down_in_pT_and_jetR/pp/rei_pp_R-0.4_pT-20_40_GeV.yaml
```

where, besides the information related to the folded SCET calculations presented in the previous section, a line pointing to the root file containing the final plots must be added:

```
final_data: '/path/to/final_results/fFinalResults.root'
```

9 Plotting ratio of Pb-Pb / pp distributions

```
cd analysis/user/rei/  
source sourceme_plot_jet_axis_raa.sh
```

That should execute commands like:

```
python plot_jet_axis_raa.py \  
-c ../../../../config/jet_axis/configs_broken_down_in_pT_and_jetR/PbPb/rey_PbPb_R0.2_pT_40_60_GeV_WTA.yaml
```

where the following lines must be added to the config file:

```
# Data  
file_pp: '/path/to/pp/final_results/fFinalResults.root'  
file_AA: '/path/to/PbPb/final_results/fFinalResults.root'
```

10 Creating curves from generator data for theory comparison

When running *e.g.* a fast simulation, we still construct a response matrix and other objects. For the theory comparison, these objects are not needed. We only need the truth spectra. Here's how to produce such spectra. To run a test locally over a Jewel PbPb job, do:

```
python process/user/rey/process_curve_from_generator_jet_axis.py \  
-c config/jet_axis/rey_PbPb_R02.yaml \  
-f /rstorage/generators/jewel_alice/tree_gen/746611/1/176/jewel.root \  
-t PWGHF_TreeCreator
```

To run over the entire dataset:

```
cd slurm/sbatch/jet_axis/PbPb  
sbatch slurm_JEWEL_R02.sh
```

To run over pythia pp, do:

```
cd slurm/sbatch/jet_axis/PbPb  
sbatch slurm_PYTHIA_R02.sh
```

In this case the resulting distributions are for pp, to use as a baseline for the PbPb predictions. The only difference is in the config files provided. The config file provided for the JEWEL events contains parameters for constituent subtraction, thus background subtraction is carried out, whereas we don't do this for pp.

11 Creating a txt file with all files in a subdirectory

```
ls -l */*/jewel.root > filename.txt
```

References

- [1] “pyjetty.” <https://github.com/matplo/pyjetty>.
- [2] “heppy.” <https://github.com/matplo/heppy>.
- [3] “ALICE analysis note, Measurement of charged jet cross section in pp collisions at $\sqrt{s_{NN}} = 5.02$ TeV.” <https://alice-notes.web.cern.ch/node/534>.
- [4] “ALICE analysis note, Measurement of charged jet spectra in Pb-Pb collisions at $\sqrt{s_{NN}} = 5.02$ TeV with ALICE at LHC (update including high interaction Pb-Pb runs).” <https://aliceinfo.cern.ch/Notes/node/818>.
- [5] “James’ analysis note.”
- [6] “Ezra’s analysis note.”