Jet measurements

Beyond inclusive jet and inclusive b jet

Patrick L.S. Connor

Deutsches Elektronen-Synchrotron
Hamburg

19 November 2020

Introduction.

Introduction I

History

- My contributions to the jet measurements with Run-2 data started early 2016 with the inclusive b jet analysis.
- Then, Radek and I started the development of a pretty powerful framework, and we decided to abandon the old one.
- Although the focus was on the inclusive jet analysis, we tried to develop the software in a generic way, so that it could be used for the inclusive b jet at least, as well as for Luis Ignacio's analyses.
- Finally, at least colleagues from Greece have already adopted the framework.

Outline

- Quick description of the framework.
- Some recent results beyond the inclusive jet and inclusive b jet.

Framework.

How the project came up
Tools
Documentation

How the project came up

DAS = *Data Analysis School*

- Successfully used for long exercise at DAS18 at DESY and DAS19 at PKU (2.5 days)
- Using full 2016 & 2017 datasets, able to run within a few minutes

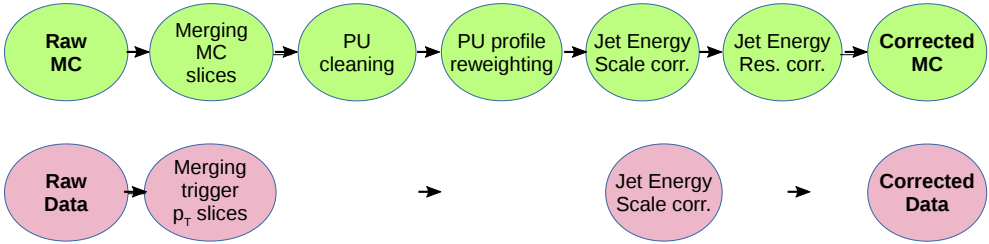
$\longrightarrow$ newcomers were able to perform analysis, from *n*-tuples to unfolded results with systematics

DAS = *Das Analysis System*

- ROOT-based environment
  - inspired from advantages and limitations from the former SMPJ *n*-tupliser
  - developed as a CMSSW module for convenience (ROOT compatibility, etc.) although no fundamental component from CMSSW is used (except in *n*-tupliser itself)
- Basic principles in appendix.
- Currently developed and used at DESY, nearly fully ported to CERN.

$\longrightarrow$ `http://gitlab.cern.ch/DasAnalysisSystem`

Tools



Existing tools

- Modular corrections of CMS samples (see figure).
- Generic macro for unfolding without regularisation.
- MC calculations.
- Tests of smoothness ("Greta" fits).

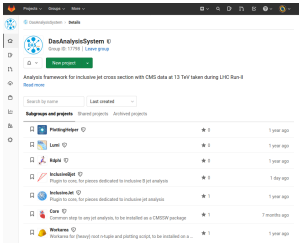
Tools in development

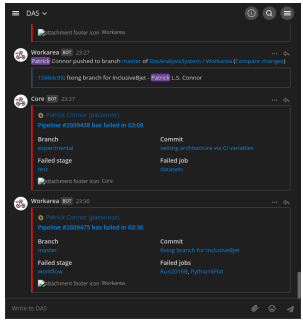
- Toy RM based on double Crystal-Ball function.
- Smoothing of systematic correlated uncertainties.

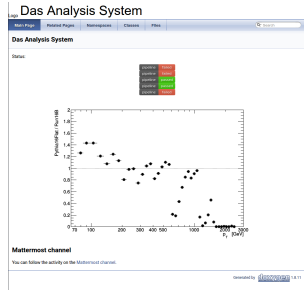
Documentation

Status

- I avoid TWiki: hard to maintain or to keep up to date.
- The AN of the inclusive jet analysis includes references to the pieces of code.
- Otherwise using Doxygen and `README.md` files.







Practice.

Dijet mass

$R_{\Delta\phi}$

Practice

The two measurements driving the developments

- Inclusive jet
- Inclusive b jet

Measurements with Luis Ignacio

- Dijet decorrelations
- Single-jet spectra

Two recent measurements discussed in the next slides...

- Dijet mass

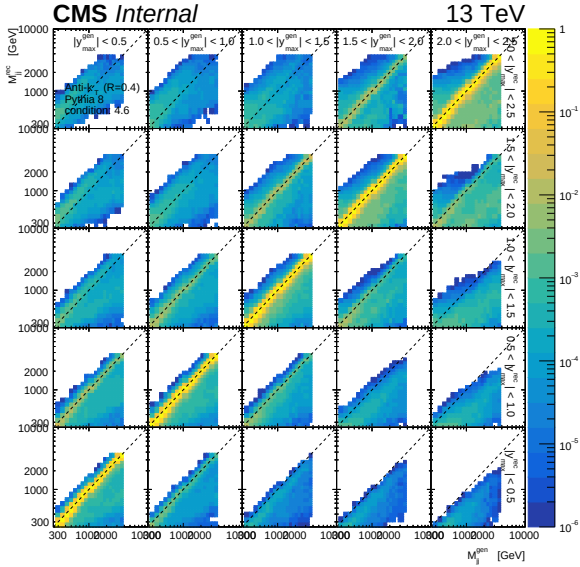
$$\frac{\mathrm{d}^2\sigma}{\mathrm{d}m_{jj}\,\mathrm{d}y_{\mathsf{max}}} \tag{1}$$

- $R_{\Delta\phi}$

$$R_{\Delta\phi}(p_T) = \frac{\Sigma_{n=0}^{\infty} n N(p_T, n)}{\Sigma_{n=0}^{\infty}\ N(p_T, n)} \tag{2}$$

$\longrightarrow$ In collaboration with colleagues from Greece & KIT.

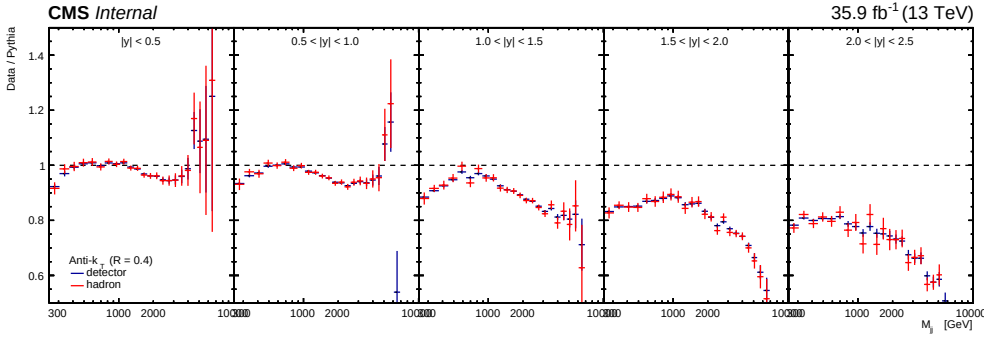
Dijet mass



Strategy

- Re-use (nearly) exact same samples as in inclusive jet analysis.
- Leading and subleading jets must be matched within a cone (swapping is allowed).
- 2D unfolding is used to treat migrations among $|y_{\text{max}}$ bins (especially important when swapping).
- Using pure matrix inversion (small condition number, no need for regularisation).

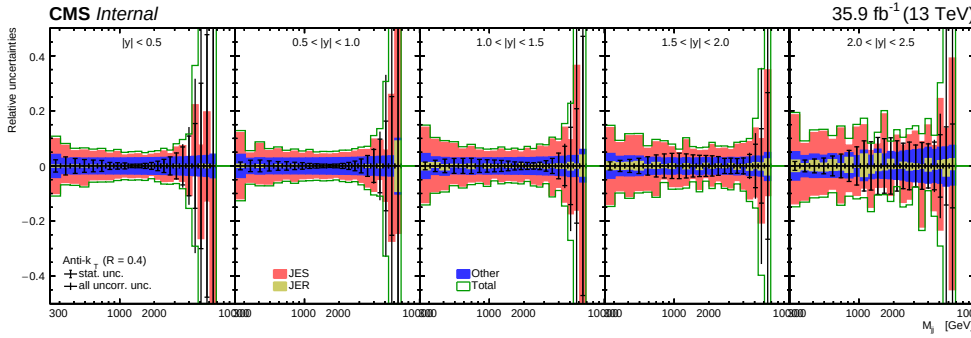
Dijet mass



Figure

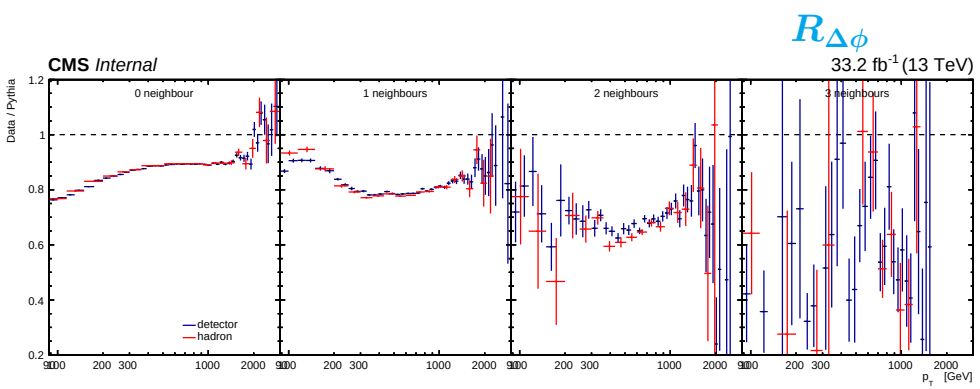
- Showing ratio to Pythia before and after unfolding.
- We use single-jet triggers based on p_T, just like in the inclusive jet analysis.
- Unfolding was straightforward, including all systematic uncertainties (except impact of model in unfolding).

Dijet mass



Figure

- Statistical uncertainties are only dominated by JES uncertainties.
- No model uncertainty.

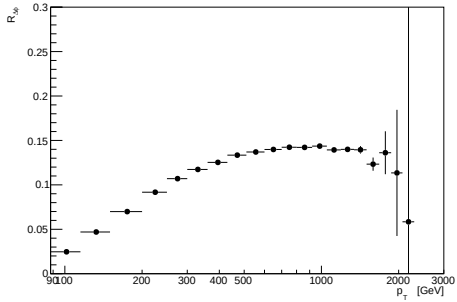


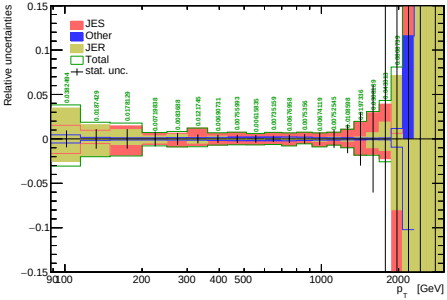
Figure

$$R_{\Delta\phi}(p_T) = \frac{\Sigma_{n=0}^{\infty} n N(p_T, n)}{\Sigma_{n=0}^{\infty} N(p_T, n)} \quad (3)$$

- $R_{\Delta\phi}$ is sensitive to α_S (see last presentation on QCD meeting of 7 June 2019).
- n is the number of « neighbours » of a given jet.
- We extract $R_{\Delta\phi}$ from $N(p_T, n)$ after unfolding.
- In practice, there are rarely more than 2 neighbours per jet.

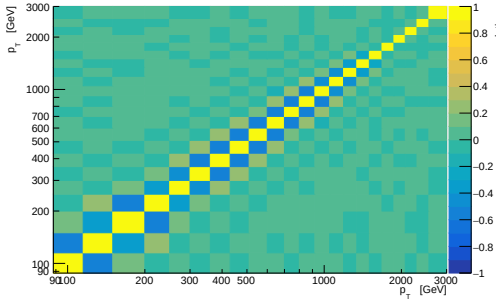
$R_{\Delta\phi}$





Figures

- We recycle the piece of code as for the extraction of the b jet fraction.
- I only show data here: the fit to obtain α_S is performed by colleagues from Ioannina.



Summary & Conclusions.

Summary & Conclusions

- A good & unique framework brings several advantages:
  - Less repetition, less time spent at re-inventing the wheel.
    $\longrightarrow$ The inclusive b jet analysis took just a few days, similarly for the dijet mass measurement.
  - Avoid artificial tensions among measurements.
  - More time to spend on novel developments or on physics.

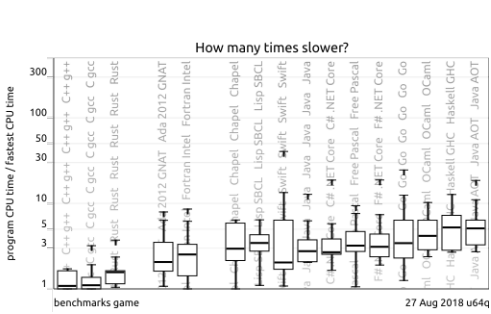
  $\longrightarrow$ Benefiting from the many developments for the inclusive jet analysis.
- Prospects:
  - Extend to full Run-2.
    $\longrightarrow$ Actually, we developed the framework with 2017 and we also tested once 2018...
  - Investigate correlations among different measurements.

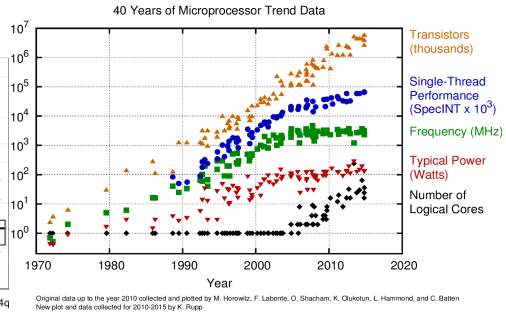
  $\longrightarrow$ Less time spent on technicalities would mean more time for physics.

Thank you for your attention!

Back-up.

Modern computing





Crucial questions

- How long does it take to compile a code?
- How long does it take to execute a code?
- How large are the root files? $\longrightarrow$ parallelisation?
- How readable is the code?
- Is it easy to maintain the code?
- Could anyone join the analysis easily?
  $\longrightarrow$ re-usability?

A few principles

The ~~ten~~ six commandments

- Thou shalt not make more than one or two operations at a time.
- Thou shalt not write codes much longer than a few hundreds of lines.
- Thou shalt not multiply layers of scripts and of classes.
- Thou shalt not mix plotting and advanced (slow) operations.
- Thou shalt not use too many flags.
- Thou shalt not over-engineer your code.

Prerequisites

- Decent knowledge of C++
  $\longrightarrow$ basic oriented object programming, basic template programming, ROOT
- Basics with Git (and GitLab)
  $\longrightarrow$ be familiar with branches and commits at least, with forks and merge requests would be nice

$\longrightarrow$ all standard things for physicist (no fundamental need to be a real C++ advanced expert)

Organisation

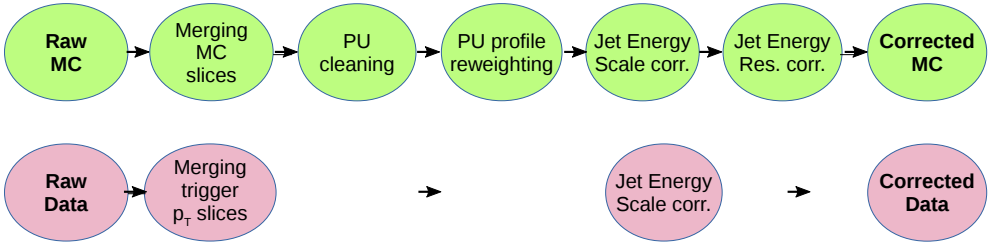
Software (on a fast storage system)

Clean & compilable C++17 code:

1. Create *n*-tuples
   $\longrightarrow$ from `MiniAOD` to light format using ROOT dictionary
2. Apply PU corrections,JECs, etc. to *n*-tuples
   $\longrightarrow$ transform a *n*-tuple into another *n*-tuple (with same structure)
3. Project *n*-tuples
   $\longrightarrow$ onto basic histograms without any cosmetics
4. Unfold projections
   $\longrightarrow$ only `TUnfold` at the moment

$\longrightarrow$ keep the software light and the code rather generic (remember that it is sent to the grid each time you submit CRAB jobs...)

Organisation (continued)



Work-area (on a large-file system, *e.g.* eos)

Make physics from the terminal or from short script (examples on next slides)

- intending to host the "heavy" *n*-tuples (at most a few tens of GB)
- still some logical structure of the directories
- also with luminosity and calibration tables
- scripts for ratio, fractions, simple projections & plotting only

$\longrightarrow$ don't waste time writing too clean code, since it should be rather dependent on the analysis

Running

Practice

The programs to analyse or modify events have the following general structure:

```
analyser input output [...] nSplit nNow
```

`input`	The input root files with events inside `TTree`
`output`	The output root file (with either histograms or `TTree`)
`...`	any other arguments (depending on the command)
`nSplit`	To how many slices to split the input sample
`nNow`	The index of slice to run over

$\longrightarrow$ just enter the command in the prompt to see the required arguments.

Running

Examples

1. run on the whole sample

```
analyser input output
analyser input output 1 0
```

⟶ the two lines are equivalent

2. run on the first half of events

```
analyser input output 2
analyser input output 2 0
```

3. run on the first thousandth of events

```
analyser input output 1000
analyser input output 1000 0
```

⟶ good for testing

4. run over all 10 slices in parallel

```
for i in `seq 0 9`
do
    analyser in out$i 10 $i &
done
```

Beyond testing: `parallel` & `submit`

- Use all cores on current machine:

```
parallel analyser input output
```

- Use `HTCondor` (might need a quick fix):

```
submit analyser input output
```

Long-term safeties

Continuous Integration (CI)

Using GitLab environment at CERN, testing after every `git push`:

- compilation
- *n*-tupliser
- a few executables
- documentation

⟶ `https://mattermost.web.cern.ch/cms-exp/channels/das18`

Doxygen

- Easiest way to document/comment within the code
- Automatically producing documentation with CI

⟶ `http://desy.de/~connorpa/DAS/doxygen/`

Acronyms I

CERN European Organisation for Nuclear Research. 5, 24

CI Continuous Integration. 24

CMS Compact Muon Solenoid. 6

DESY *Deutsches Electroknen-Synchrotron*. 5

JEC Jet Energy Correction. 20

JES Jet Energy Scale. 12

PU pile-up. 20

RM Response Matrix. 6

References I

Patrick L.S. Connor
Deutsches Elektronen-Synchrotron
Tel.: +49 40 8998-2617
Geb.: 01a/O2.104
https://www.desy.de/~connorpa