

Diagnosing Issues in Software/Code Development

IPPP Computing Club Seminar

Hitham Hassan

IPPP, Dept. of Physics
Durham University

August 9, 2022

“When you have eliminated all which is impossible, then whatever remains, however improbable, must be the truth.”

— Sir Arthur Conan Doyle, stated by Sherlock Holmes,
The Case-Book of Sherlock Holmes, 1927

A Pipeline from Ideas to Results

1. **The idea:** this could be a calculation, an algorithm, a procedure, or some combination that is planned to address a problem or a need.
2. **The implementation:** the specific software or script or collection of scripts developed to address the problem. This includes e.g. planning and testing.
3. **The run:** the process of using the program to produce results, involving specific settings and configuration.
4. **Processing of results:** results rarely leave the software in the most raw form, some processing is almost always required to ease interpretation e.g. plotting scripts, merging scripts etc.

Where in the above pipeline may a problem arise?

A Pipeline from Ideas to Results

1. **The idea:** does the calculation need a review? Does the algorithm not take into consideration certain important effects? Have you double counted some contributions?
2. **The implementation:** does the implementation accurately reflect the idea? Is the logic of some stage of the algorithm incorrect as implemented? Are all the factors the same as in the calculation? Are the tests comprehensive enough to find these issues?
3. **The run:** Are the correct input parameters used? Was the correct version of the software used?
4. **Processing of results:** Do the plotting scripts plot the correct quantities? Are results from different runs merged properly?

Where in the above pipeline may a problem arise - **EVERYWHERE**

Scale of Severity

These stages (**idea**, **implementation**, **run**, **processing**) could be seen as a descending scale of 'severity'.

For example, an error in the **processing** (e.g. a plotting script) means the plots will have to be reproduced. An error in the **idea** could mean the whole software **implementation** has to be reproduced (then **run** again, then have the data **processed**).

Misdiagnosing a stage 4 problem as a stage 1 problem can be catastrophic for the project and lead to significant reduction in productivity.

Typically, problems in the later stages are easier to diagnose and quicker to fix, it makes sense to approach the hunt for problems in reverse order.

Workflow for Addressing Issues

A Workflow for Addressing Issues - Processing of Results

Check with third party tools: are there other methods for processing results such as plotting scripts (e.g. `rivet-mkhtml`) or merging results from different runs (e.g. `yodastack`)? Check that these give the same output.

Check with your own tools: if you use third party tools, check with your own tools - write a simple plotting script to double check the output of e.g. `rivet-mkhtml`.

Ask the developers: if you use third party tools, get in touch with the authors (or look at the issues if the code is hosted with Git) to check if there are known issues.

Use dummy data: Create simple data that you could process manually (e.g. a single point or a straight line for the plotting script), does the processing give the expected output?

Example - Image Preprocessing I

A previous project involved pre-**processing** data from Sherpa on top quark jet profiles such that we could classify them with a machine learning algorithm.

First two stages:

1. A **translation** of all elements such that was at the centre of the jet.
2. A **rotation** to place the second hardest element directly above ($\phi = 0$) the centre

Snippet of my Python script:

```
# rotate to give max2 phi = 0
coords[i+1] = -np.sin(angle)*coords[i+1] + np.cos(angle)*coords[i]
coords[i] = np.sin(angle)*coords[i] + np.cos(angle)*coords[i+1]
```

The rotation seemed not to be correctly implemented, comparison on next slide

Example - Image Preprocessing II

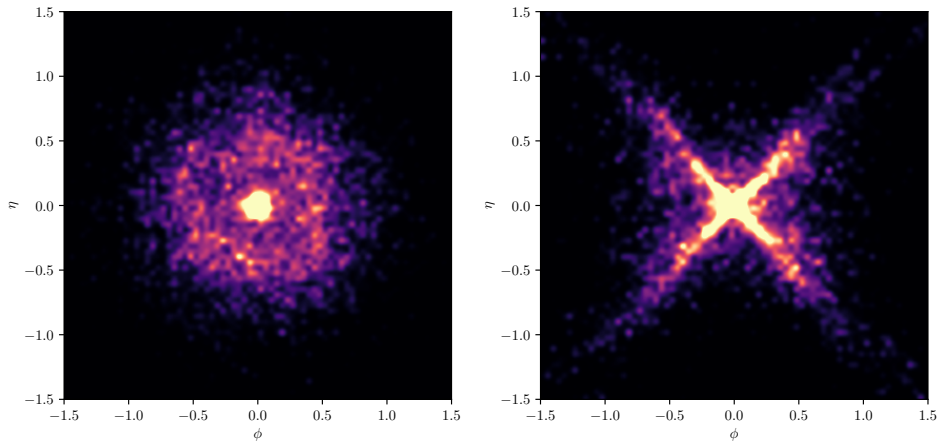


Figure: The p_{\perp} profile of a top-quark jet, from simulated Sherpa data. The left plot shows the figure as it should look, the right plot shows the figure after preprocessing with a tool I had developed.

Example - Image Preprocessing III

Issue was in rotating values we had **already rotated**:

```
coords[i+1] = -np.sin(angle)*coords[i+1] + np.cos(angle)*coords[i]  
coords[i] = np.sin(angle)*coords[i] + np.cos(angle)*coords[i+1]
```

Resolved by **storing previous values**:

```
eta_tmp = coords[i+1]  
phi_tmp = coords[i]  
  
# rotate to give max2 phi = 0  
coords[i+1] = -np.sin(angle)*eta_tmp + np.cos(angle)*phi_tmp  
coords[i] = np.sin(angle)*phi_tmp + np.cos(angle)*eta_tmp
```

I found this by plotting a dataset of just **two points** - the rotation was where the issue seemed to be hiding.

A Workflow for Addressing Issues - Running I

Document run parameters: keeping track of results **and** the input parameters (e.g. runcards for Sherpa or PYTHIA) **and** the version of the code used to produce them allows you to backtrack and check that you can reproduce old results.

A good method is to use a private Git(Hub)(Lab) repository - I organise mine with **directories** by **date** of the run, with the **input parameters** used, a (Markdown) text file showing the **commit ID** of my code used and a sample of the **results** from each run.

Automate! If you only make small changes to input parameters each time (e.g. for Sherpa only altering the minimum p_{\perp} by small amounts per run), write and test a script for generating the input.

A Workflow for Addressing Issues - Running II

I

Input Parameters for Weekly Meetings

Project ID: 776

🔔

☆ Star 0

🍴 Fork 0

🔗 8 Commits

🌿 1 Branch

🏷️ 0 Tags

💾 47.2 MB Project Storage

main

input-parameters-for-weekly-meetings /


+

Find file

Web IDE


📄


Clone





Add input parameters for meeting of 22-07-26
Hitham Hassan authored just now


258979f0





 README


 Add LICENSE

 Add CHANGELOG


 Add CONTRIBUTING

 Enable Auto DevOps

 Set up CI/CD

 Configure Integrations

| Name | Last commit | Last update |
|----------------------|--|-------------|
| 📁 22-05-24/follow-up | Add parameters for 22-05-24 | 1 month ago |
| 📁 22-06-01 | Add input parameters etc. for meeting of ... | 1 month ago |
| 📁 22-06-08 | Add input parameters for meeting of 22-0... | 1 month ago |
| 📁 22-06-14 | Add input parameters for meeting of 22-0... | 1 month ago |
| 📁 22-07-20 | Add input parameters for meeting of 22-0... | 5 days ago |
| 📁 22-07-26 | Add input parameters for meeting of 22-0... | just now |
| 📄 README.md | Modified README.md | 1 month ago |

 README.md

Input Parameters for Weekly Meetings



Parameters used to generate figures/results displayed in the HEJ+Pythia drupal [book](#) for weekly progress meetings.

Figure: A snippet of the GitLab repository I use to store my input parameters.

A Workflow for Addressing Issues - Running II

Hitham Hassan > **Input Parameters for Weekly Meetings**

main ▾ input-parameters-for-weekly-meetings / 22-06-01 History Find file Web IDE ⬇ ▾ Clone ▾
/ weight-checks / + ▾

 **Add input parameters etc. for meeting of 22-06-01** 8ff69433 
Hitham Hassan authored 1 month ago







| Name | Last commit | Last update |
|--|---|-------------|
| .. | | |
|  HEJmerging-unweighted.yoda | Add input parameters etc. for meeting of 22-06-01 | 1 month ago |
|  HEJmerging-weighted.yoda | Add input parameters etc. for meeting of 22-06-01 | 1 month ago |
|  Run.dat | Add input parameters etc. for meeting of 22-06-01 | 1 month ago |
|  config.yml | Add input parameters etc. for meeting of 22-06-01 | 1 month ago |
|  hej_merging.cmdnd | Add input parameters etc. for meeting of 22-06-01 | 1 month ago |
|  unweight-note.md | Add input parameters etc. for meeting of 22-06-01 | 1 month ago |

Figure: An example of one of the subdirectories, showing the input parameters (dat, cmdnd, yml), the output (yoda) and a note (unweight-note.md) in the Markdown format briefly outlining the checks made and a link to the revision of the code used.

A Workflow for Addressing Issues - Implementation

Debugging! Use tools such as `gdb` to trace through the program and analyse how variables evolve (we may have had a talk about this ...)

More output: alter your program to produce more output if you are handling large, complex data structures - e.g. print out an event at each stage in your algorithm.

Check test coverage: are your tests comprehensive? Do they cover as much of the code as practically possible? Do your tests perform the checks they are supposed to be performing?

Revert: check with previous versions of the code if the issue has recently arisen. If the code is hosted with `Git` this is simple and one needs only revert to previous commits.

A Workflow for Addressing Issues - Ideas

There is little to say here as this is the most specific to the problem you are trying to solve.

The main guiding principle here is **to not be too attached to your ideas** or the basis of your software implementation, they may be incorrect and need significant alteration.

Much time can be wasted by insisting the problem lies elsewhere in the pipeline - most of the time it will, but the possibility that the foundation was faulty **needs** to be accounted for.

Interlude - Questions?

A Case Study from My Work

My project (*HH*, *S. Jaskiewicz*, *J. R. Andersen*): produce a software **implementation** of an **algorithm** to combine events from one MC event generator (*High Energy Jets* or *HEJ*) with a parton shower (*PYTHIA*).

The **algorithm** is the **idea** - this is a complex procedure with many aspects of theory and principles underlying it.

Part of this is a *recoil strategy* - a scheme for reshuffling momentum added to an event by the parton shower.

Some results have been produced where the two jet rate is significantly underestimated by *HEJ+PYTHIA* compared to *HEJ* (which we know matches data well).

A Case Study - My Project

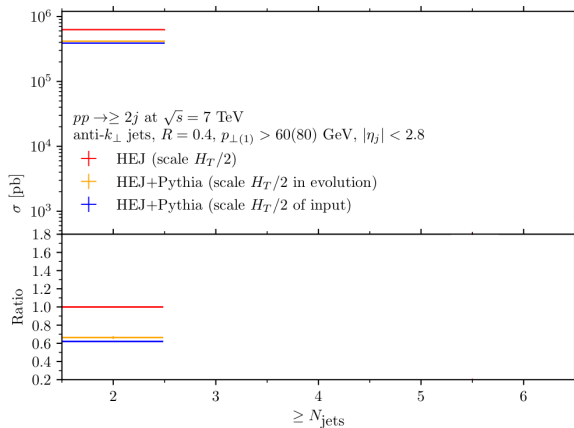


Figure: The underestimated $2j$ cross sections in HEJ+PYTHIA. The figure has been edited to remove results for the other bins which are not relevant to the discussion. The HEJ+PYTHIA lines are significantly beneath the HEJ line in the $2j$ bin.

Checking Processing of Results and the Run

Processing of Results: third party tools were used to process results (`rivet`, `YODA`). These were used for all of the results shown so any issue with these tools should be present in all lines.

These tools were used before without issue for different predictions.

Analysing the run: the `GitLab` repository was used to check against different input parameters from previous runs.

Within our group, other collaborators document their input parameters for different projects, additional checks were performed against these.

Checking the Implementation I

Added functionality to the implementation to produce optional debug output - i.e. a 'verbose' mode when running.

Made minor modifications to gauge the difference between our recoil strategy against that of PYTHIA on the same starting events.

Traced the evolution of a single event through the entire algorithm with the verbose option.

A sample of the output is attached.

Checking the Implementation II

This is a good start but there is still a problem - is this output easily comprehensible?

```
1717 no      id name      status mothers daughters colours p_x p_y p_z e m
1718 0        90 (system) -11 0 0 0 0 0 0.000 0.000 0.000 7000.000 7000.000
1719 1        2212 (p+) -12 0 0 3 0 0 0.000 0.000 3500.000 3500.000 0.938
1720 2        2212 (p+) -12 0 0 4 0 0 0.000 0.000 -3500.000 3500.000 0.938
1721 3        21 (g) -21 1 0 5 13 505 511 0.000 -0.000 360.882 360.882 0.000
1722 4        21 (g) -21 2 0 5 13 502 507 -0.000 -0.000 -1618.031 1618.031 0.000
1723 5        21 g 23 3 4 0 0 502 504 62.817 38.663 -27.638 78.770 0.000
1724 6        21 g 23 3 4 0 0 505 503 -79.088 -40.734 135.694 162.256 0.000
1725 7        21 g 23 3 4 0 0 510 506 6.801 4.282 -6.376 10.259 0.000
1726 8        21 g 23 3 4 0 0 506 507 1.722 -8.308 -1529.571 1529.594 0.000
1727 9        21 g 23 3 4 0 0 504 508 -6.401 -5.767 13.020 15.612 0.000
1728 10       -1 dbar 23 3 4 0 0 0 509 3.040 2.763 8.990 9.890 0.330
1729 11       21 g 23 3 4 0 0 503 510 7.487 11.308 -6.740 15.144 0.000
1730 12       21 g 23 3 4 0 0 509 511 3.290 0.262 154.734 154.769 0.000
1731 13       1 d 23 3 4 0 0 508 0 0.332 -2.470 0.737 2.620 0.330
1732 Charge sum: 0.000 Momentum sum: -0.000 0.000 -1257.149 1978.914 1528.292
1733
1734 ----- End PYTHIA Event Listing -----
1735 After appending:
1736
1737 ----- PYTHIA Event Listing (hard process-modified) -----
1738
1739 no      id name      status mothers daughters colours p_x p_y p_z e m
1740 0        90 (system) -11 0 0 0 0 0 0.000 0.000 0.000 7000.000 7000.000
1741 1        2212 (p+) -12 0 0 3 0 0 0.000 0.000 3500.000 3500.000 0.938
1742 2        2212 (p+) -12 0 0 4 0 0 0.000 0.000 -3500.000 3500.000 0.938
1743 3        21 (g) -21 1 0 5 13 505 511 0.000 -0.000 360.882 360.882 0.000
1744 4        21 (g) -21 2 0 5 13 502 507 -0.000 -0.000 -1618.031 1618.031 0.000
1745 5        21 g 23 3 4 0 0 502 504 31.504 27.063 7.232 42.157 0.000
1746 6        3 s 51 8 0 0 0 505 0 -2.189 -6.371 22.818 23.797 0.500
1747 7        21 g 23 3 4 0 0 510 506 3.372 3.046 -0.858 4.624 0.000
1748 8        21 g 23 3 4 0 0 506 507 -4.008 -14.572 -1588.190 1588.262 0.000
1749 9        21 g 23 3 4 0 0 504 508 -15.367 -11.095 52.369 55.694 0.000
1750 10       -1 dbar 23 3 4 0 0 0 509 1.119 2.353 10.152 10.486 0.330
1751 11       21 g 23 3 4 0 0 503 510 0.160 10.798 0.662 10.819 0.000
1752 12       21 g 23 3 4 0 0 509 511 2.074 -0.825 179.535 179.549 0.000
1753 13       1 d 23 3 4 0 0 508 0 -1.418 -4.322 4.228 6.219 0.330
1754 14       -3 sbar 51 8 0 0 0 503 -15.247 -6.075 54.904 57.307 0.500
1755 Charge sum: 0.000 Momentum sum: -0.000 0.000 -1257.149 1978.914 1528.292
1756
```

Data Visualisation

Checking the Implementation III

While some problems may be quickly identifiable, the huge data structures we work with are generally difficult to interpret in raw form.

To counter, I produced a Python module to read and **visualise** the events, the module plots the particles from an event in (y, ϕ) space and applies a colour grading to highlight their hardness in p_{\perp} .

This way the evolution of an event can be charted in a **visual format** and differences between our recoil scheme and the local recoil of PYTHIA can be better visualised.

We do exactly this and compare events after several trial emissions from the shower within both recoil schemes.

Comparison of $y - \phi$ Plots - Input v.s. Merged

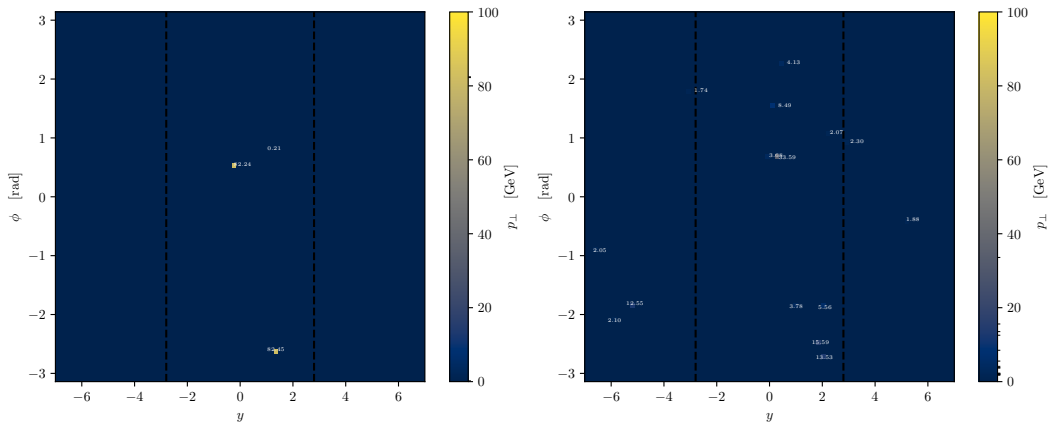


Figure: For the attached event trace, the input event is plotted on the left and the evolved event after merging in HEJ+PYTHIA is plotted on the right. The merged event clearly has had its jet multiplicity reduced.

Comparison of $y - \phi$ Plots - Trial 1

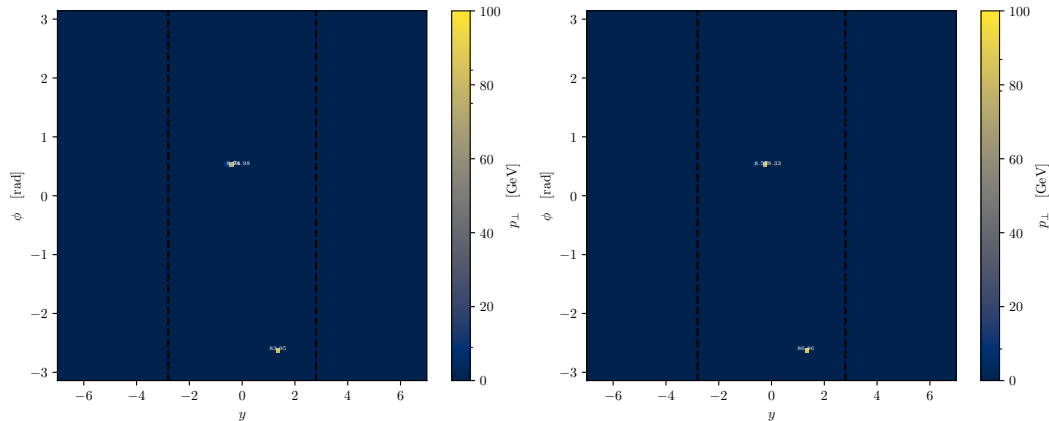


Figure: The first PYTHIA trial (left) compared to the result of recoiling the trial emission within our recoil strategy (right). The emission dynamics seem well reproduced by our recoil.

Comparison of $y - \phi$ Plots - Trial 4

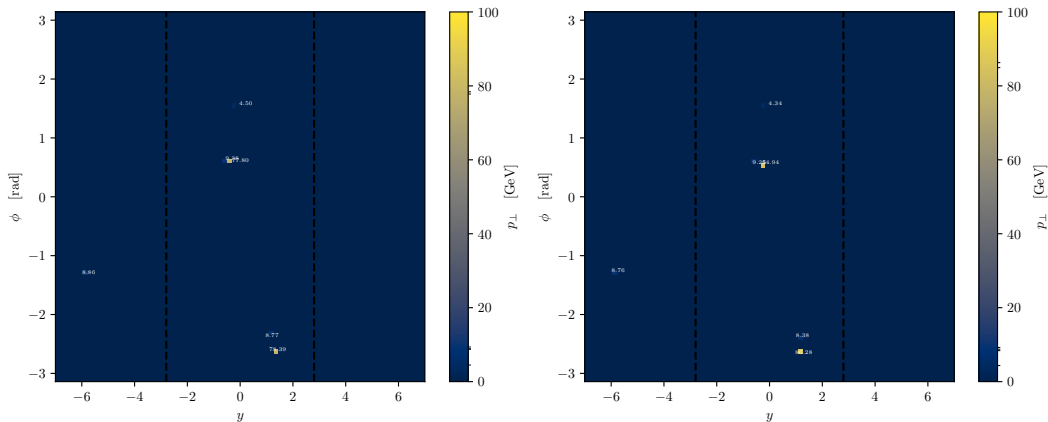


Figure: The fourth PYTHIA trial (left) compared to the result of recoiling the trial emission within our recoil strategy (right). The emission dynamics seem well reproduced by our recoil again, though with slight differences that appear subleading in shower accuracy.

Distribution of Trial 7 - the Base for Trial 8

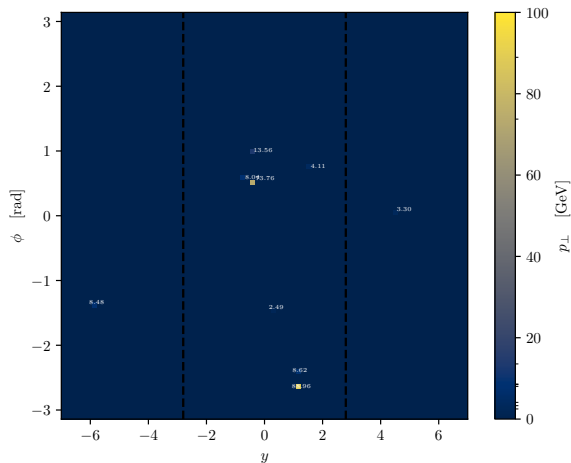


Figure: Trial 7 shown in PYTHIA - this will be the base for trial 8 which is where we see the biggest problems in our recoil strategy.

Comparison of $y - \phi$ Plots - Trial 8

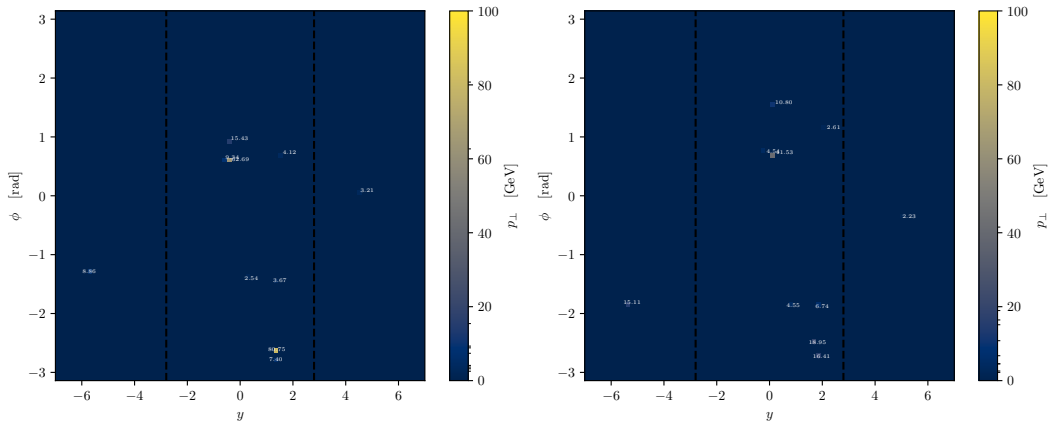


Figure: The eighth trial in PYTHIA (left) and the result of our recoil strategy (right), a $g \rightarrow s\bar{s}$ splitting. The mother appears to have been misidentified as the hard g with $p_{\perp} = 80.75$ GeV and replaced with a soft $s\bar{s}$ pair, recoiling the hard momentum excess across the event.

Interlude - Rapidity Search

When searching for a mother with our recoil strategy, we look to the closest parton in rapidity to e.g. split into a $q\bar{q}$ pair in the above case.

Neither the hardness, nor the ϕ of the partons were taken into account in our search.

Additionally, recoiling the momentum globally meant that the roughly 70 GeV of excess (vector) transverse momentum was spread across the event, **reducing the momentum of clustered jets**.

On the next slide are plotted again the base event for trial 8 and the trial itself in PYTHIA. The rapidities are labelled and the mother and daughter partons highlighted.

Comparison of $y - \phi$ Plots - Trial 8

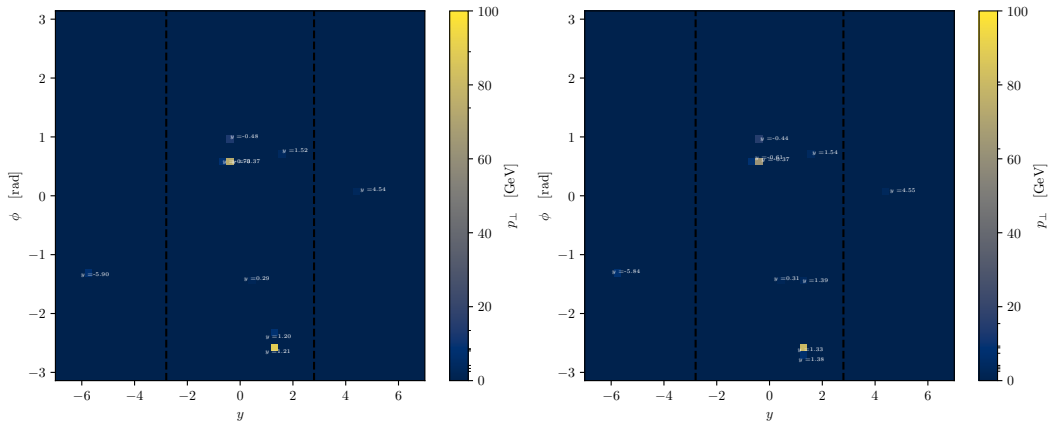


Figure: Base event for eighth trial (left) and the eighth trial in PYTHIA (right). Rapidities are labelled instead of p_{\perp} , though the colour grading is still performed in terms of hardness. The mother parton is highlighted in the left and the daughters on the right. The PYTHIA clustering identified the $y = 1.38$ parton (right) as the remnant of the emitter which is closer to the hard $y = 1.21$ gluon (left) than the actual mother.

Where was the Problem?

In this case, the problem seems to originate from stage 1, **the idea**. The rapidity search was part of our model that did not take into account other aspects of the event.

We have implemented a new search, which minimises the distance measure:

$$\Delta R_{\text{extended}}(p_m, p_c) = \sqrt{(y_c - y_m)^2 + (\phi_c - \phi_m)^2 + \left(\frac{p_{\perp,c} - p_{\perp,m}}{p_{\perp,m}} \right)^2},$$

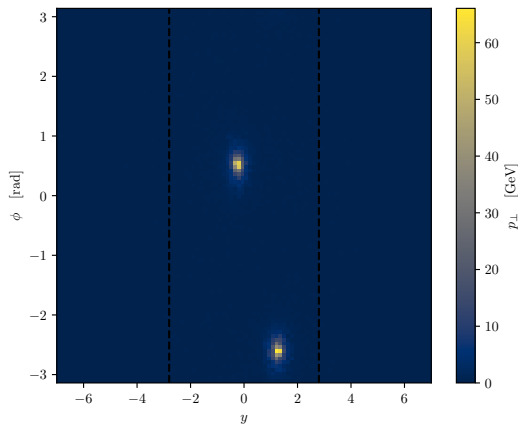
between a candidate c and the true mother m to find the closest parton.

These are big changes and it pays to be sure they are required before making them - hence why this process was essential and worthwhile.

This can be used to instruct new unit tests on the implementation (e.g. a more comprehensive check on the difference between recoil strategies).

Validation

As a preliminary validation test, we ran HEJ+PYTHIA on the same problem event 25,000 times with our new recoil strategy - the jet count appears retained for the majority of cases.



The feature of the $2j$ bin persists, though we are exploring different causes for this.

Final Points

Final Points I

This is a suggestion for a method that works for me - it is useful as a guiding principle but not intended to be applied blindly and without flexibility.

The discussion was centered around event generation and algorithms - this is just from my background.

These concepts can be applied to any sorts of projects - big or small. Consider a simple plotting script:

1. **The idea:** produce a convenient method for plotting data.
2. **The implementation:** produce a script to do this.
3. **The run:** using the script on your data.
4. **Processing of results:** maybe not so much processing - perhaps if multiple data are combined to produce plots?

Final Points II

This process can **importantly** be used to develop unit tests.

Unit tests are tools created to address a problem, which produce data that needs to be interpreted/processed by the developer (error messages, coverage reports, ...)

Some aspects are more difficult to diagnose - it may be that tests you have written do not actually test for the quantities you want.

See my [unit tests](#) talk for more detailed guiding principles in test-driven development.

Main point to take away:

**anything produced to fulfil a purpose should NOT be exempt from inspection,
issues can (and often will) arise everywhere.**

Thank You!

“When you have eliminated all which is impossible, then whatever remains, however improbable, must be the truth.”

— Sir Arthur Conan Doyle, stated by Sherlock Holmes,
The Case-Book of Sherlock Holmes, 1927