LCIO Analysis in Jupyter Notebooks

Marcel Stanitzki (DESY)
Jan Strube (PNNL)

PNNL is operated by Battelle for the U.S. Department of Energy

LCWS 2018

Pacific Northwest NATIONAL LABORATORY

LCIO – Linear Collider Input / Output format

- Workhorse for linear collider studies
- Has been around since 2003
- Used for all event storage
  - Productions for strategic documents
  - Beam test analysis
  - Students' theses
- A couple of petabytes stored world-wide

Pacific Northwest NATIONAL LABORATORY

C++

- Default way to interact with LCIO
- Tried and trusted, production quality
- Modern C++ is becoming a bit more user friendly
  - But, there is still a lot of old C++ around
- Additional learning curve for installation
  - Download and unpack LCIO
  - Configure with CMake and build and install
- Additional learning curve for use
  - Need build tools for compiling the code

Pacific Northwest
NATIONAL LABORATORY

Python

- For new students, Python is a good option
  - Syntax easier to learn than C++
  - Lots of good documentation
  - Lots of good utilities for data analysis
- Great for quick-and dirty plots and cross-checks
- Installation can be a hassle
  - Does the ROOT version play with the Python version?
  - Has LCIO been compiled with Python bindings?
    - ✓ With the right ROOT version?

Pacific Northwest
NATIONAL LABORATORY

Julia

- Relatively new effort: just released version 1.0
  - But has been around in ILC-land for two years (cf. Talk at Morioka 2016)
- Just-in-time compiled:
  - No extra step before running your code
  - Allows better optimizations than interpreted code
- Syntax very similar to python
- User-friendliness built in
  - Package manager
  - Unicode support

Pacific Northwest NATIONAL LABORATORY

LCIO.jl – Overview

- Mostly a faithful representation of the LCIO API
  - Just go to `lcio.desy.de` and browse the online documentation of the EVENT namespace
- Strong typing: LCIO collections know their type. So do LCIORelation objects. No casting necessary. You will get the right type for both sides, `getRelationFrom()` and `getRelationTo()`, and for `getCollection()`
- Proper iteration over files and collections.
  - `for event in file do_something end`
  - `for particle in collection do_something end`
- Installation (no need for a compiler)
  - `Pkg.add("LCIO")`
  - That's all. If you have set up LCIO at this point, LCIO.jl will use it. If not, it will download and install LCIO.

Example C++ code

```
#include "lcio.h”
#include "IO/LCReader.h"
#include "EVENT/LCCollection.h"
#include "EVENT/ReconstructedParticle.h"
#include <string>
#include <iostream>
using namespace std;
using namespace lcio;

int main(int argc, char** argv ){
  int nEvents = 0;
  for (int nfiles=1; nfiles < argc; nfiles++) {
  string FILEN = argv[nfiles];
  cout << "Opening File " << nfiles << " "<<FILEN
<< endl;
  LCReader* lcReader = LCFactory::getInstance()-
>createLCReader();
  lcReader->open( FILEN );
  LCEvent* evt;
double visibleE = 0.0;
```

```
//----------- the event loop -----------
while ( (evt = lcReader->readNextEvent()) != 0 ) {
  LCCollection* col = evt-
>getCollection("PandoraPFOs");
  for (size_t i=0,N=col->getNumberOfElements(); i <
N; ++i) {
    ReconstructedParticle* p =
(ReconstructedParticle*) col->getElementAt(i);
    visibleE += p->getEnergy();
  }
  nEvents++;
}
cout << “visible E = “ << visibleE << endl;
lcReader->close();
delete lcReader;
}
cout << " read " << nEvents << " Events" << endl;
return 0;
}
```

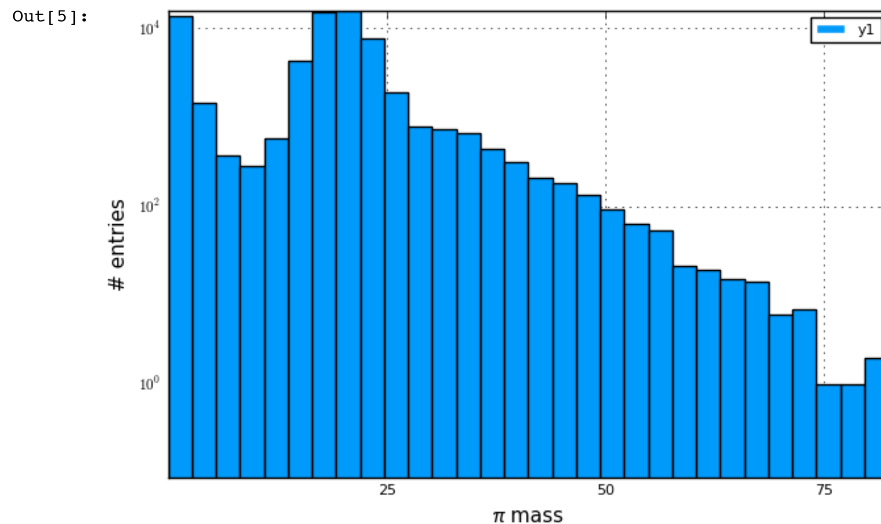
Pacific Northwest NATIONAL LABORATORY

Example LCIO code

```
using LCIO
nEvents = 0
for FNAME in ARGS
  visibleEnergy = 0.0
  LCIO.open("file.slcio") do reader
       for event in reader
      nEvents += 1
                for particle in getCollection(event, "PandoraPFOs")
          visibleEnergy += getEnergy(particle)
        end
      println("visible energy is: ", visibleEnergy)
    end
  end
  println("read ", nEvents, " Events")
end
```

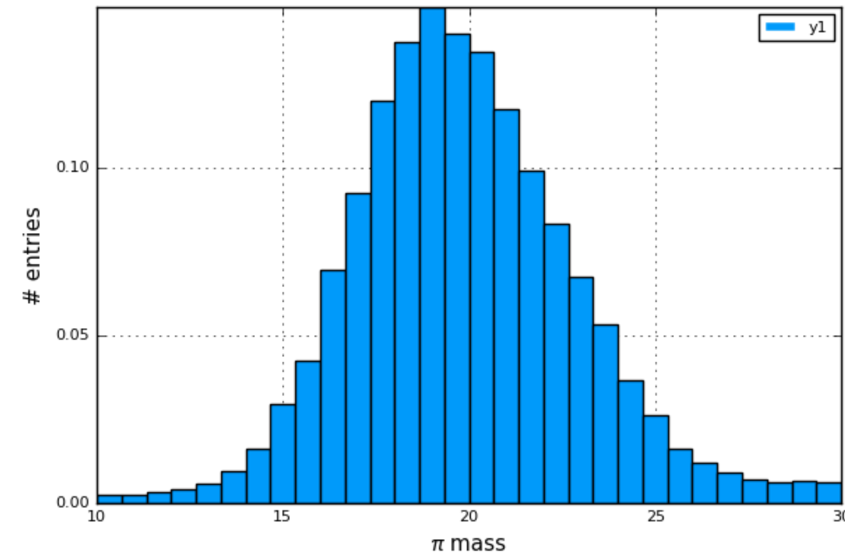
Jupyter notebooks

```
In [1]: using LCIO; using Plots; using Distributions; using StatsBase;

        WARNING: Method definition show(IO, Base.Multimedia.MIME{:text/plain}, Plots.Plo
        t) in module Plots at /Users/stru821/.julia/v0.5/Plots/src/output.jl:168 overwri
        tten at /Users/stru821/.julia/v0.5/Plots/src/output.jl:241.

In [2]: pyplot()

Out[2]: Plots.PyPlotBackend()

In [3]: filename = "/Users/stru821/Workdir/singlepi+_20__scint1x1_phi6_REC.slcio"

Out[3]: "/Users/stru821/Workdir/singlepi+_20__scint1x1_phi6_REC.slcio"

In [4]: energies = Float32[]
        LCIO.open(filename) do reader
            for event in reader
                particles = getCollection(event, "PandoraPFOCollection")
                for p in particles
                    push!(energies, getEnergy(p))
                end
            end
        end

In [5]: histogram(energies, yaxis = ("# entries", :log10, (.09,Inf)), xaxis = ("π mass"))

        [Plots.jl] Initializing backend: pyplot

Out[5]:
```

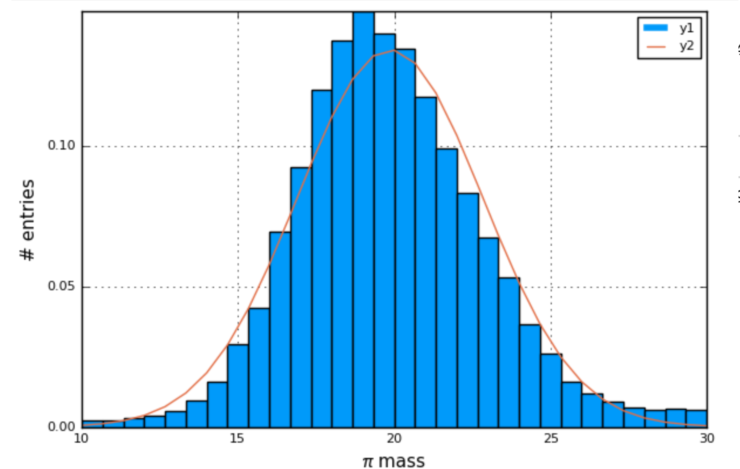


Let's focus on the interesting region between 10 and 30 GeV

```
In [6]: interesting = filter(x->10<x<30, energies);

In [7]: histogram(interesting, yaxis = ("# entries"), xaxis = ("π mass", (10,30)), bins=3
        0, normalize=true)

Out[7]:
```




```
In [8]: h = fit(Histogram, interesting, 10:(30-10)/30.:30)

Out[8]:   In [11]: plot!(h.edges[1], pdf(x, h.edges[1]))

          Out[11]:

In [9]:

Out[9]:
```




```
In [12]: x.μ

Out[12]: 19.867395401000977
```

A jupyter notebook is an interactive environment for data analysis

Code is interspersed with the results (plots) of that code.

→ A great tool for teaching and for data exploration

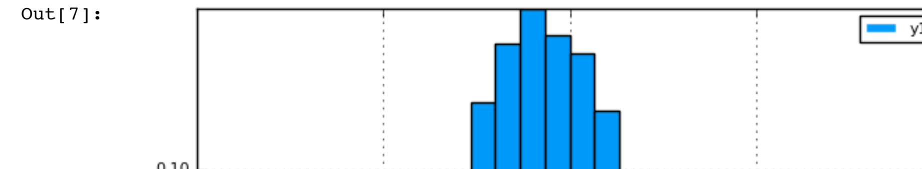
Jupyter notebooks are extensively used in the Belle II starter kit tutorials

Pacific Northwest NATIONAL LABORATORY

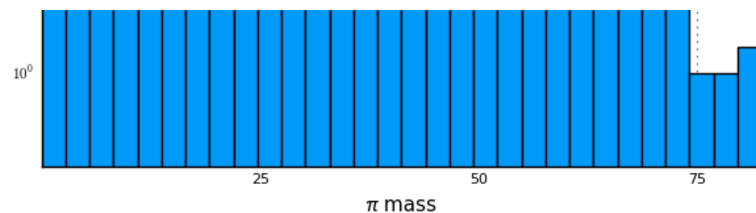
Jupyter notebooks

```
In [1]: using LCIO; using Plots; using Distributions; using StatsBase;

        WARNING: Method definition show(IO, Base.Multimedia.MIME{:text/plain}, Plots.Plo
        t) in module Plots at /Users/stru821/.julia/v0.5/Plots/src/output.jl:168 overwri
        tten at /Users/stru821/.julia/v0.5/Plots/src/output.jl:241.

In [2]: pyplot()

Out[2]: Plots.PyPlotBackend()
```

```
In [7]: histogram(interesting, yaxis = ("# entries"), xaxis = ("π mass", (10,30)), bins=3
        0, normalize=true)

Out[7]:
```

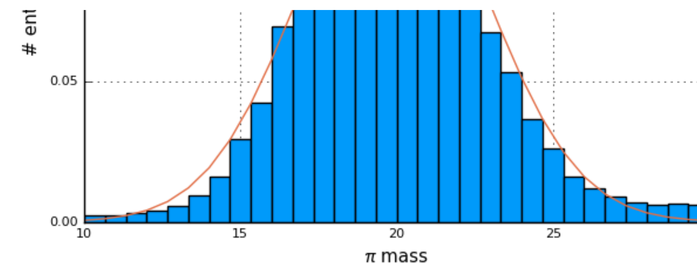



```
energies = Float32[]
LCIO.open(filename) do reader
  for event in reader
    particles=getCollection(event,"PandoraPFOCollection”)
    for p in particles
      push!(energies, getEnergy(p))
    end
  end
end
```



Let's focus on the interesting region between 10 and 30 GeV

```
In [6]: interesting = filter(x->10<x<30, energies);
```




```
In [12]: x.μ

Out[12]: 19.867395401000977
```

Pacific Northwest NATIONAL LABORATORY

What's in it for me? List of cool features

- No C++ → Much faster turnaround for quick analyses
  - Interactive access to the data
- Strong support for data science workflows
  - Documentation is readily available online
- Fewer black boxes
  - Fewer opaque recipes, no C/C++ behind the scenes (like numpy / tensorflow)
  - One consistent language throughout simplifies teaching
- Jupyter notebooks
  - Web technology for data analysis
  - Analyze data **on the remote server** without having to wait for graphics forwarding
- Building blocks for parallelization

Pacific Northwest NATIONAL LABORATORY

Performance measurements

- Simple test:
  - Find two muons from all reconstructed particles
  - Compute invariant mass of the muon pair
  - Plot the result
- Julia timing: up to 1030 events / second
  - Comparable with C++ speed
  - But: still start-up overhead for the julia compiler
- Second test: Implement Fox-Wolfram moments and thrust from pythia into Julia
  - Straightforward translation, basically just removed the type annotations
  - Timing comparable

Pacific Northwest NATIONAL LABORATORY

Performance comparison in numbers

Events	Julia	Python	C++	C++vs JULIA	C++ vs Python	Julia vs. Python
32400	35.66	41.34	26.59	34.09%	55.48%	15.95%
64800	64.12	79.75	52.09	23.10%	53.11%	24.37%
97200	92.53	118.93	77.37	19.60%	53.72%	28.53%
129600	121.38	157.65	103.19	17.63%	52.78%	29.88%
162000	149.55	195.88	128.46	16.42%	52.48%	30.98%
194102	178.68	234.86	154.48	15.66%	52.03%	31.44%

- Timings are for finding two reconstructed muons, and computing the Z mass.
- Very little computing, mostly sensitive to speed of the bindings and compiler / interpreter overheads.
- Timings for Fox-Wolfram moments forthcoming.

Pacific Northwest NATIONAL LABORATORY

Summary

- The Julia programming language offers a compelling alternative for data analysis
  - Speed comparable to C++
  - Simplicity comparable to Python
- Jupyter notebooks are useful tools for teaching and interactive data exploration
  - Allow access to data processing **on the remote server** without the need for graphics forwarding
- LCIO.jl is an implementation of LCIO in the julia programming language that allows using jupyter notebooks for the analysis of linear collider data sets
  - github.com/jstrube/LCIO.jl

Pacific Northwest
NATIONAL LABORATORY

Thank you