Weighting toolkits in Jpp

From the user-perspective

To reiterate

- What's possible / what's new:

1. Ordering sets of MC-files based on header-info
2. Clear, unambiguous mapping:
   weight-functions (with specifiable fluxes!) ⟺ **MC-files**

I.e.: 'low-level' interface to be used in conjunction with high-level software

- Potential applications:

1. Combining MC-files with compatible headers into one
2. Creating single run-by-run MC-files for all primaries
   - Add / modify total weight (w3 / 'w4') for each event, based on ordered header-info
3. User-friendly reweighting of (set of) MC-productions
   - Scaling weights by single user-specifiable function already possible
   - Current tools can be extended to allow arithmetic operations with flux-functions

Neutrino event reweighting

Not a new idea!

Currently everyone needs to write their own implementation

Still needed:

- Coupling to oscillation software
- Flux function arithmetics
- Automatic testing

[Simulations_git issue #3]

working_groups › Simulations › Issues › #3

Closed Opened 1 year ago by Alba Domi

Function to re-weight neutrino events in a MC file

I think it would be really useful to write a function (or probably a class?) to re-weight MC neutrino events (for example in a run by run mc simulation). I think it would be useful for 2 reasons:

1. so that everyone do not to waste too much time on doing that
2. so that everyone can use the same functionality from Jpp instead of having different codes (which of course will include more checks and possible bugs)

This procedure is not so immediate up to now because it is needed to know:

1. the run duration (in years) of the equivalent data run
2. we need the oscillation probability to re-weight neutrinos (use of OscProb or Jpp probabilities?)
3. the production flux is also needed

So, my idea is simply to have a function in which you give as input:

1. the name of the DATA file
2. the neutrino flux you want to consider at the end
3. the neutrino flux generation spectrum

and it does all the job.

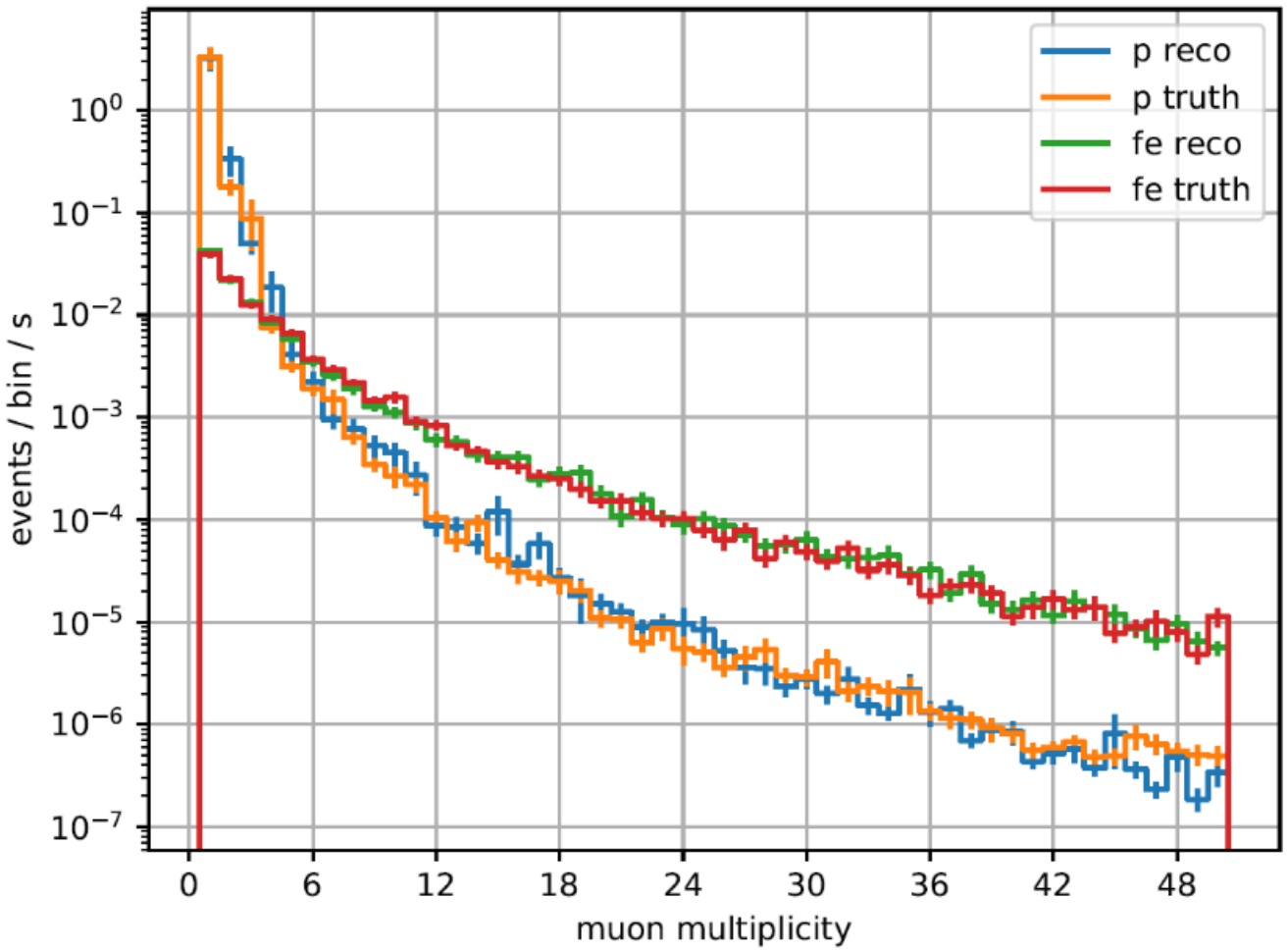
What do you think? Maybe it is something more "high level" but honestly I don't like too much the idea to use different languages/frameworks/codes to do the same thing. If there is something complete in one framework I think people will use it more easily.

\assign @vkulikovskiy

Muon multiplicities

From Stefan Reck's collabo presentation

all p+Fe events, weighted

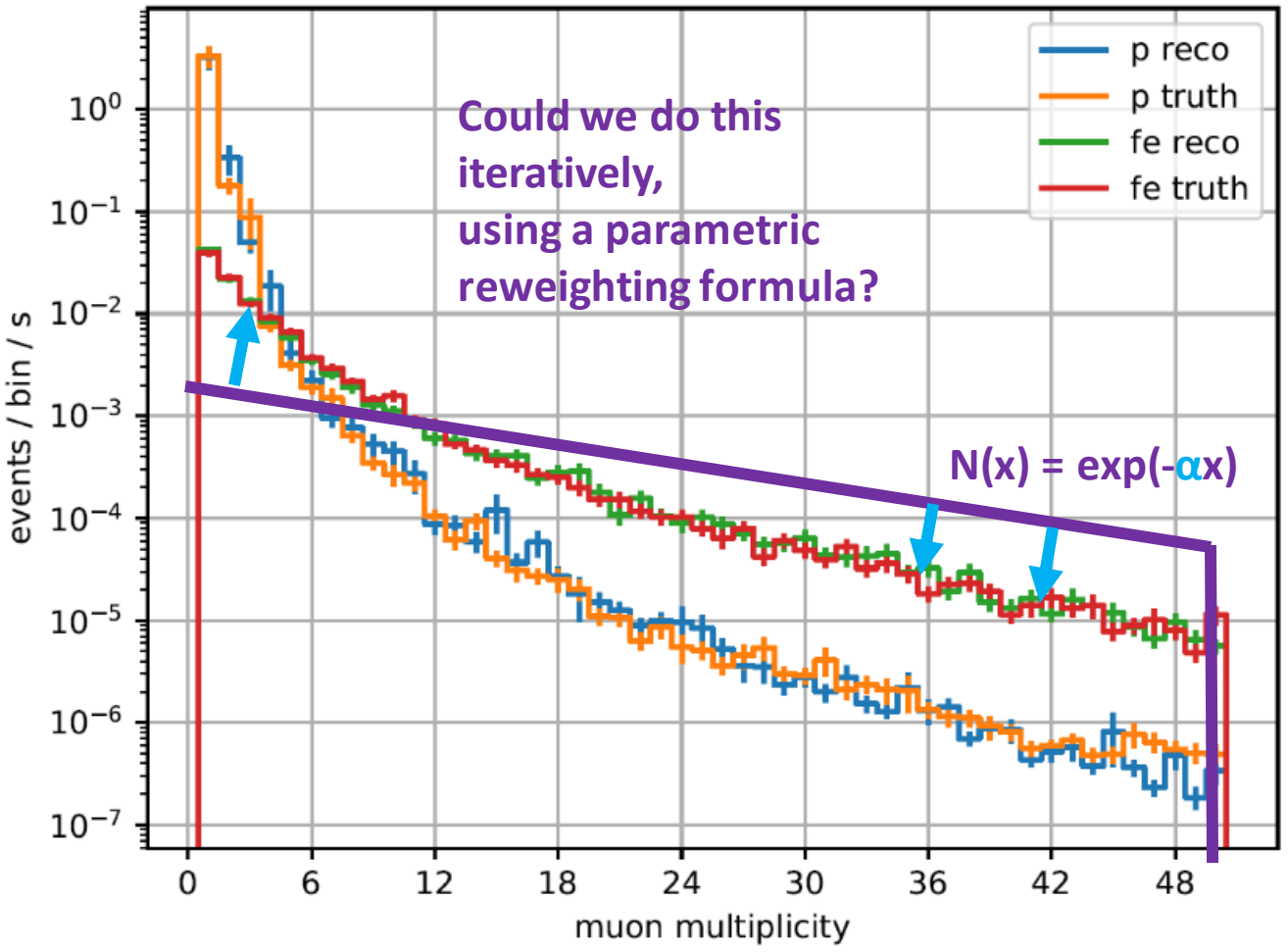


ML-based muon multiplicity reconstruction

Muon multiplicities

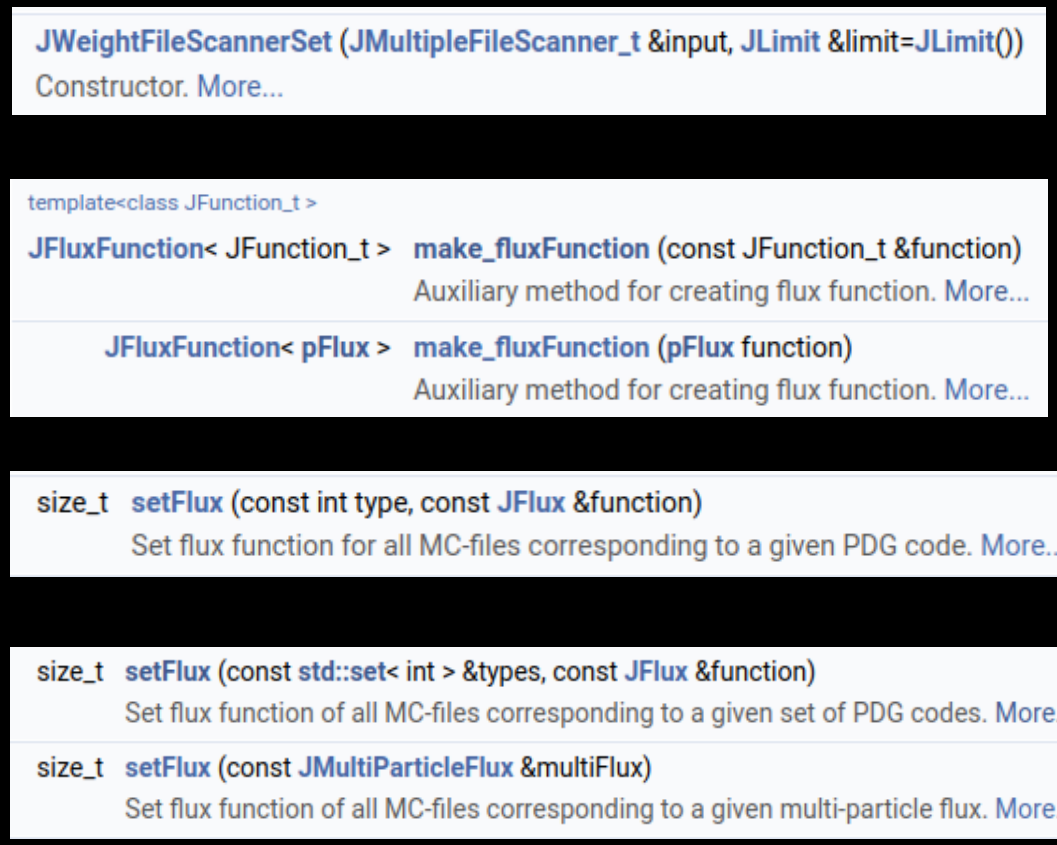
From Stefan Reck's collabo presentation

all p+Fe events, weighted



How to use

- For external user only few functions important:



member methods of JWeightFileScannerSet

Create set of ordered (MC/DAQ) file-scanners, with corresponding weight-functions

Create a flux-function wrapper that can be interfaced with one or multiple event-weighters

Assign given flux-function wrapper to all files, which contain the given PDG type as primary

Assign given flux-function wrapper only to those files, which contain *all* PDG types in a given set

How to use

- Example:
  <jpp>/examples/JAAnet/JMultiParticleFlux.cc

```
JMultiParticleFlux multiFlux;

for (vector<int>::const_iterator i = zeroFluxes.cbegin(); i != zeroFluxes.cend(); ++i) {
  multiFlux.insert(*i, make_fluxFunction(zeroFlux));
}

for (map<int, JFlatFlux>::const_iterator i = flatFluxes.cbegin(); i != flatFluxes.cend(); ++i) {
  multiFlux.insert(i->first, make_fluxFunction(i->second));
}

for (map<int, JPowerLawFlux>::const_iterator i = powerlawFluxes.cbegin(); i != powerlawFluxes.cend(); ++i) {
  multiFlux.insert(i->first, make_fluxFunction(i->second));
}


// Set event weighter

JWeightFileScannerSet<> scanners(inputFiles, numberOfEvents);

size_t n = scanners.setFlux(multiFlux);

if (n == 0) {
  WARNING("No file found containing all given primaries; Flux function not set." << endl);
}
```

Three example flux-functions:

1. ZeroFlux
2. JFlatFlux
3. Power-law flux

But should also work with your favourite self-defined / imported flux-function!