Software "enhancements" (how to hack your way to a Better/Faster analysis)

G Niculescu
James Madison University

June 28, 2019

Outline:

Time-permitting I shall talk about...

- Monte Carlo Simulation Considerations
- Data Analysis & MC analysis
- Monte Carlo Considerations (II)
- ...
- Outlook

Disclaimer:

This is just GN's $0.02 worth...

- **First:** Like any piece of advice the points/suggestions made in this talk are only worth what you make of them.
- **Second:** If you already know/use some of these: Sorry to have wasted 15 min of your time!
- **Third (and most important!):** Do not think for 0.001 s that I am bashing on the (many) wonderful people that contributed to the Hall C software.
- ... especially since I happen to be one of them!
-
- That said, onward to the **Hall C simulation**...

Hall C/A simulation

Fact:

With very few exceptions the analysis of all Hall C (A) experiments requires some simulation. Usually "a lot" of simulation!!

As you (hopefully) all know...

- The "standard" Hall C (and A) simulation software comes in two main "flavors":
- **simc** - two arm simulation. physics–aware, radiation and detector effects, matrix elements, etc. Spectrometer available: HMS, SOS, SHMS, HRS(l) , HRS(r).
- **mc_single_...** Single arm Monte Carlo. Detector effects, matrix elements. Generally non–physics aware. Intended to get acceptance* & solid angle. HMS, SHMS, others ?

Common Threads...

Q:

What do simc and the single arm MC have in common?

Answer(s):

- written in **f77** (w/ extensions).
- output in **hbook (.rzdat)** format. Text too (summaries...)
- needs **cernlib** for output, and for some inner level routines.
-
- "SIMC is NOT [...] Not **hard** to modify" (!!)
  (from simc on github)

mc_hms_single Readme.md excerpt

You do not have to take my word for it:

... Readme.md excerpt

Running code

---

mc_hms_single
(ask for input file name (assumed in infiles subdirectory with .inp extension)
* Input file : infile_name
* Output file is at outfiles/infile_name.out
* The hbook file is at worksim/infile_name.rzdat

Therefore...

Problems:

- cernlib was last updated $\sim$ 1995!!!...
- maintaining it off–site (where the lab encourages you to do your simulation/analysis!) is problematic.
- output size is limited in size! ("c mkj [...] the file size is limited to $\sim$260M no matter how I change iquest !" - mc_shms_single.f)
- "standard" simc/mc_... (github, wiki) don't have an **obvious** mechanism for setting the rng seed! Is there but it is hidden!
- So running several simulation w/o changing seed gives you just **copies** of the first one!!!

Getting simc/mc_... to compile/run w/o simc

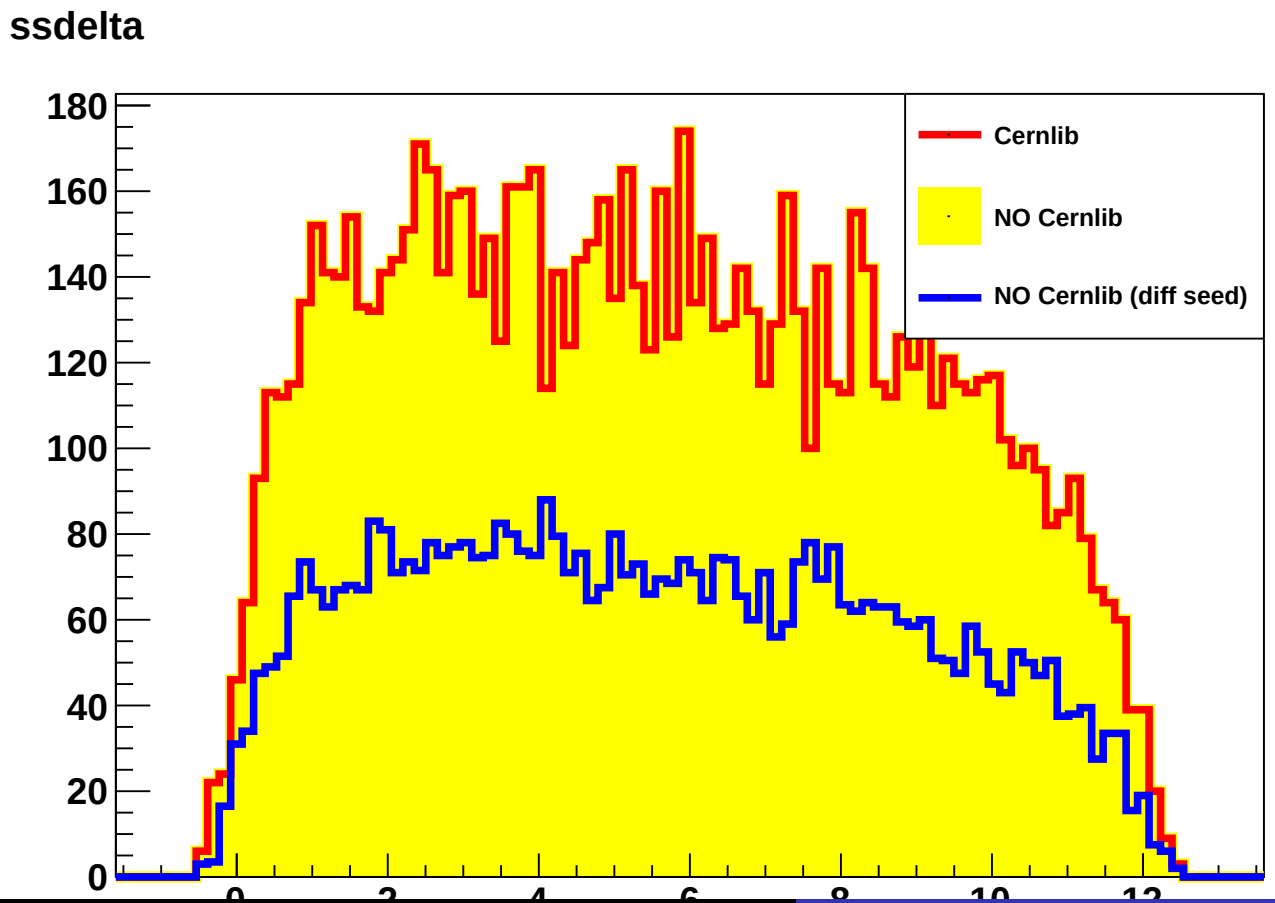
to compile & run w/o cernlib we:

- identify the routines (besides HBOOK) the code needs
  - **lfit** - linear fit subroutine (simc, mc_single...)
  - **fint** - multidim. interpolation (simc only)
- **simple hack:** get orig. lfit.f and fint.f (KEK?), remove calls to error–formatting funcs, force "real*4" instead of "real".
- Remove HBOOK references from CTP (simc only).
- For output I went w/ the ROOT tree format...
- Took only one C++ function (w/ switch to change between book/fill/close) to take care of the output.
- Added a "run_number" in the MC input while at it.

You want proof? Here it is:

Verify (ifarm/offsite) that the hack gives the same #s as cernlib.

- it DOES! Look 4 it on /volatile and, if approved, on github.



OK. I got data and some MC. Now what?

GN's $0.02: Review/Make/Adopt* a **plan (workflow)**.

- ...that takes you from the data & MC files to
- ... wherever you want/need 2g.
- ... whatever is needed for your experiment.
- time spent here (as opposed to jumping and producing some uncooked spaghetti code!) will pay for itself in no time at all.
- Old programmer's quote (author escapes me, sorry!): "...spend 30% of the time making (software) tools..."

"no plan ever survives ..."

Of course you will have to adapt the plan, possibly several times.

Sample single arm exp. workflow.

Preprocessing (JLab):

- drop variables from .root file ($\sim$ x2 reduction)
- Check data quality (detectors, optics, etc.) - lots of .pdf files!
- Move data to JMU (Globus is it!)

Simulation(JMU):

- Generate single arm MC events
- Typically x10 more successes than in the data.
- Check for DATA/MC offsets (lots of pdf files!)
- Run RC, pi0 codes.
- Get distributions, tables, etc. as needed.

Scan:

- All Settings taken @ the same θ and Z

Setting:

- all COMPATIBLE runs taken w/ the same (E,E',θ, cryo...)
- includes DATA, MC, MT (as in eMpTy)

Run:

- a bunch of real (or MC) data @ a given (E, E', θ, Z, A...)
- efficiencies applied at this level
  - acc. too (if using...) - indexed by the same N

(Main) Processing:
All done in Compiled C++/ROOT.

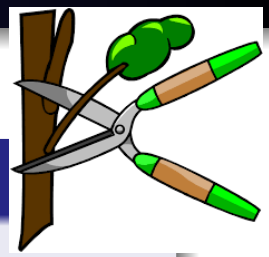
$\sim$.

$\sim$.

PostProcessing:

- Form d/p ratios
- Apply various corrections. Produce plots and tables...
- All/most done in Python (numpy, pandas, matplotlib)

Implementing the plan



Whatever strategy you adopt...

- Chances are you will have to:
  - define cuts for the data (PID, timing, fiducial...)
  - define cuts for the simulation (see above).
  - apply cuts to data a/o MC and fill (many) histograms.
- Do the above many–many times (syst. studies, oops-ies, etc).

G.N.'s $0.02: you can start by...

- **prune** your trees! eliminate superfluous variables. (Q: # of...)
- **flatten** your trees! (no: nHits; abc[nHits]; ...)
- the leaner the better!

Tree traversing strategies

...more than one way of skinning...

- **tree→Draw()**
- loop over entries (**MakeClass**...)
- **RDataFrame**
-
- +es and -es of these below

"classic" Tree Draw

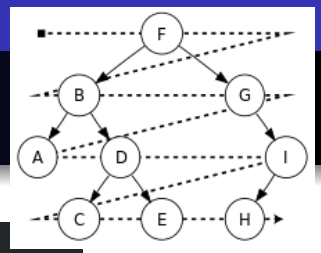
...

myTree→Draw("avar>>ahist", aCut);

complete (short) test program

```
// GN 2019. A short test drive of
// <classic> tree drawing in ROOT
void test_hfill0() {
  TStopwatch ss;  // time keeps on ticking...
  // Define the limits and labels of the histograms.
  TFile *f=new TFile("jmuMC/shms_2488_mc.root");
  TTree *h1411=(TTree *)f->Get("h1411");
  TString aCut="stop_id==0 && hsdelta >=-10.
 && hsdelta <=22.";
  h1411->Draw("hsdelta>>hh1(200, -20., 40.)",aCut);
  // ... add more histogram drawing here
  ss.Stop();
  cout << "Timing for <classic> 1D
  histogram projection (x10): ";
  ss.Print();
}
```

Looping and MakeClass...

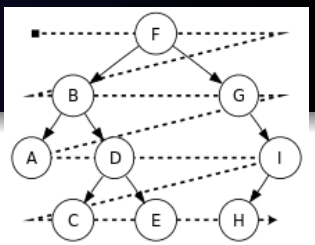



```
[gabriel@DESKTOP-GOFCKT6 uproot]$ root -l
Gabriel Niculescu: Welcome to ROOT
root [0] TFile *f=new TFile("jmuMC/shms_2488_mc.root");
root [1] h1411->Ma
MakeClass
MakeCode
MakeProxy
MakeSelector
MayNotUse
root [1] h1411->MakeClass(
Int_t MakeClass(const char* classname = 0, Option_t* option = "")
root [1] h1411->MakeClass("myTreeExample");
Info in <TTreePlayer::MakeClass>: Files: myTreeExample.h and myTreeExample.C generated from TTree: h1411
root [2]
```

Notes:

- use code like the one above to generate a skeleton analysis class for your tree.
- methods of interest: init, book, notify, Cut, Loop
- use class to loop over events, apply Cut(s), fill histograms.
- **G.N.'s $0.02: Best if compiled!**

RDataFrame...

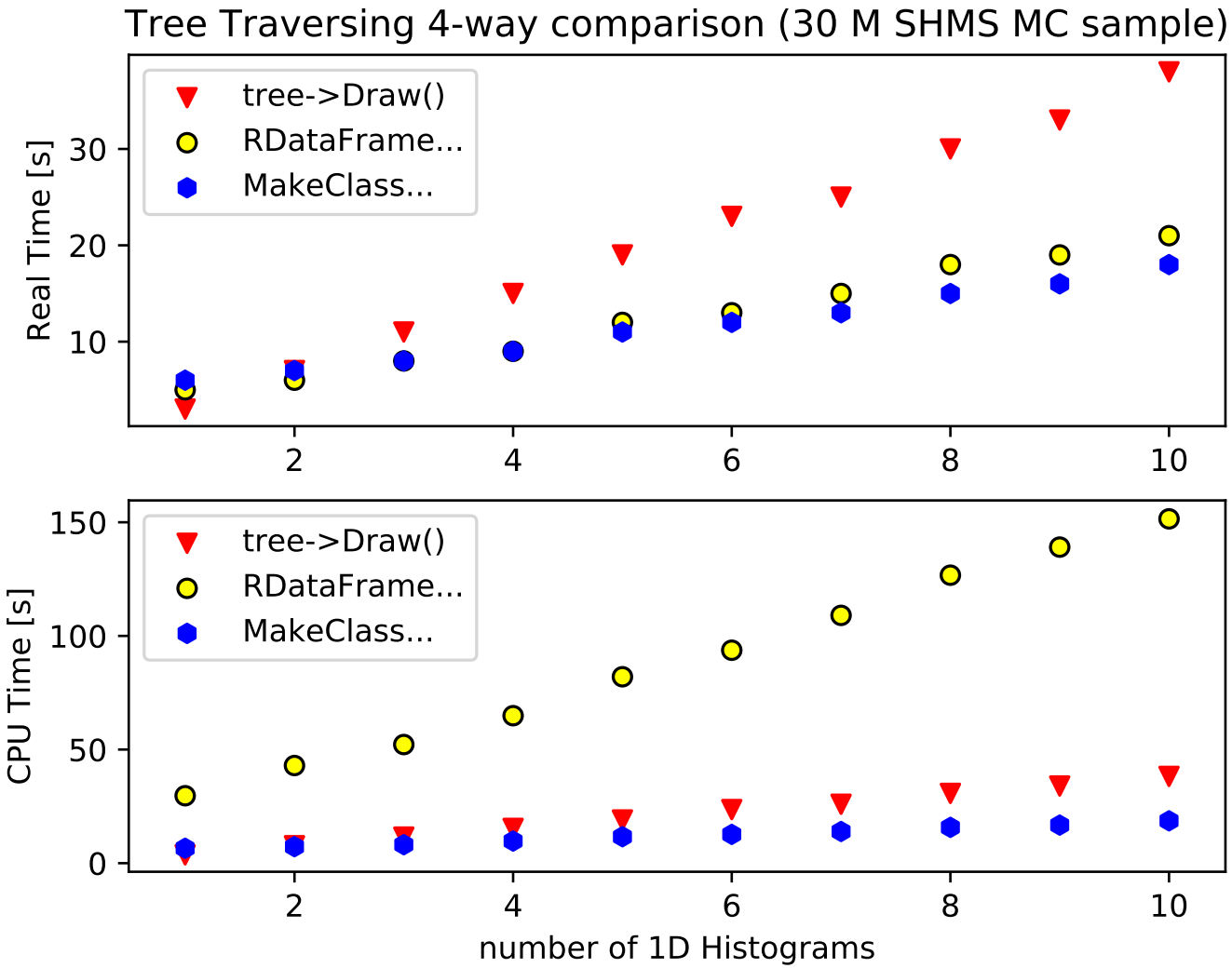



```
//
// GN 2019. A short test drive of ROOT's version
// of R DFs.
//
void test_rdf01b() {
  TStopwatch ss;                      // time keeps on ticking...
  ROOT::EnableImplicitMT();           // Multi-threaded, baby!
  // open TTree (or TChain). Specify shms MC tree.
  ROOT::RDataFrame df("h1411", "jmuMC/shms_2488_mc.root");
  auto aCut = "stop_id == 0 && hsdelta >=-10. && hsdelta<=22."; // here is a cut
  // Histogram definitions. complete with cuts, if using.
  // NOTE: these are all "lazy" actions so the DF is not looped trough just yet
  auto hh1 = df.Filter(aCut).Histo1D({"hh11","SHMS delta; hsdelta; Counts", 200, -20.,40.}, "hsdelta");
  auto hh2 = df.Filter(aCut).Histo1D({"hh12","SHMS delta; hsdelta; Counts", 200, -20.,40.}, "hsxpfp");
  // ... skipped some hists here
  auto hh11 = df.Histo1D({"hh21","SHMS delta; hsdelta; Counts", 200, -20.,40.}, "hsdelta");
  // This is in ||, of course.
  hh1->DrawCopy(); // The first time one of the histograms defined above sees an IMMEDIATE Action
  hh2->DrawCopy(); // like Draw, that's when the event loop gets run!
  // ... skipped some histograms so the text fits
  hh1->DrawCopy(); // GN: DrawCopy - otherwise they disappear at the end of the program!!
  hh11->SetLineColor(kRed); // to verify that two diff. cuts get applied when looping...
  hh11->DrawCopy("same");
  ss.Stop(); // stop the clock. inform the human.
  cout << "Timing for RDF 1D histogram projection (x10): ";
  ss.Print();
}
```

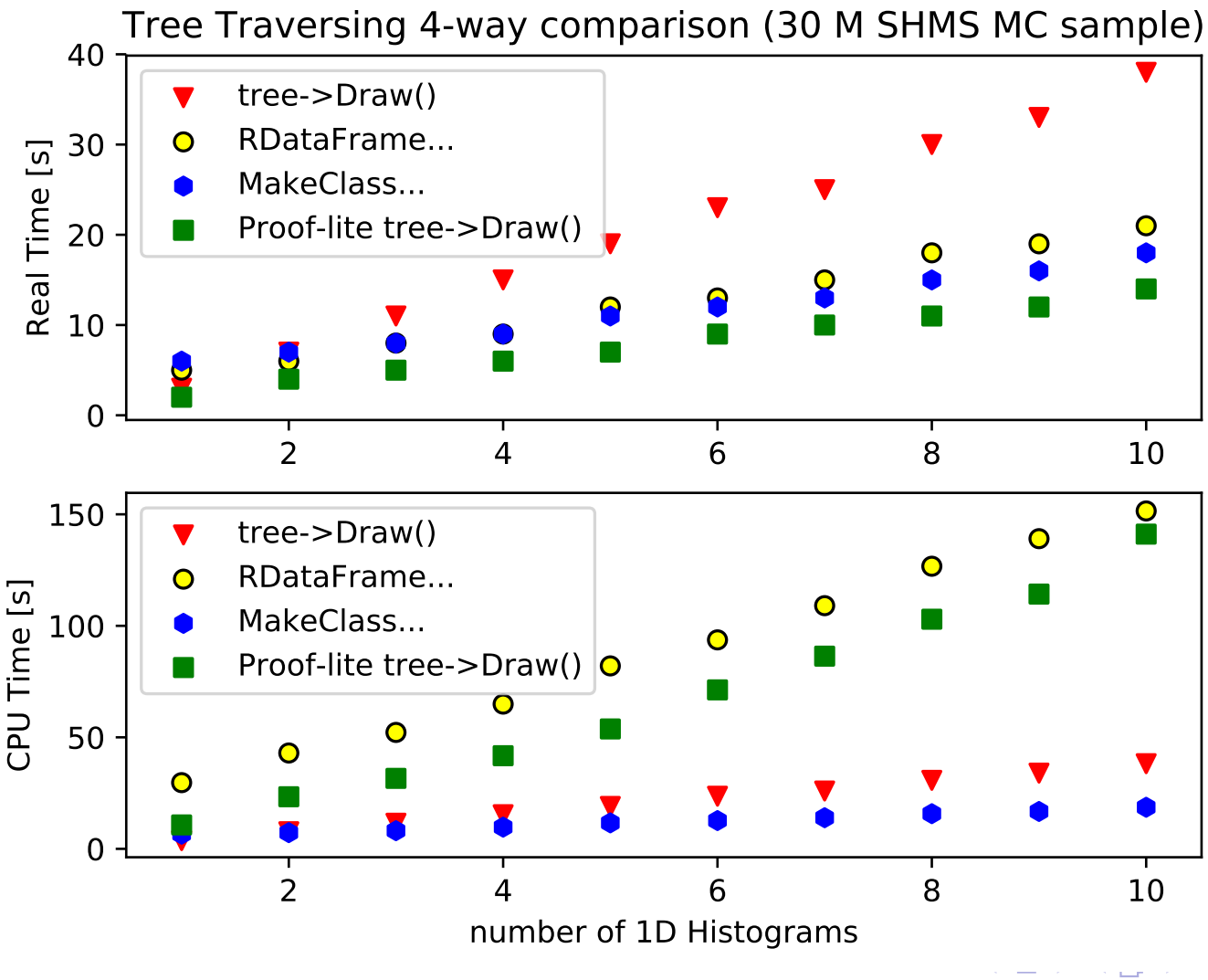
RDF:

- ROOT implementation of R's DataFrame.
- Lazy vs immediate action. Multithreaded.
- **G.N.'s $0.02: Good intro to pipelines, DA...**

Comparisons (I)



Comparisons (II)



...and the winner is...

G.N.'s $0.02:

- winner unclear. sub–optimal strategies seem obvious:
  - non–compiled looping over events.
  - lots of tree→Draw calls (while not doing Proof!)
- **Go parallel** (on one/more machines))!!
- **Proof–lite:** very easy. (++) on single user systems. ordering not guaranteed... (-)
- **Compiling loops:** (acLiC) also straightforward.
  More control (++), more code (-). OK single thread.
- **RDF:** opens the door to a "whole new world" (industry).
  easy MT, adding columns. piping... "lazy action"

Typically...

NOTE:

- this is not a finished product!
- maybe the beginning of a mature, civil, and hopefully fact–based discussion?

Acceptance

- one uses the simulation to obtain (and subsequently apply) the "Acceptance Correction" (acceptance, eff. phase space).
- opening a (random) thesis one finds things like:

$$Acceptance(i) = \frac{N_{recon}(i)}{N_{gen}(i)} \quad i - bin$$

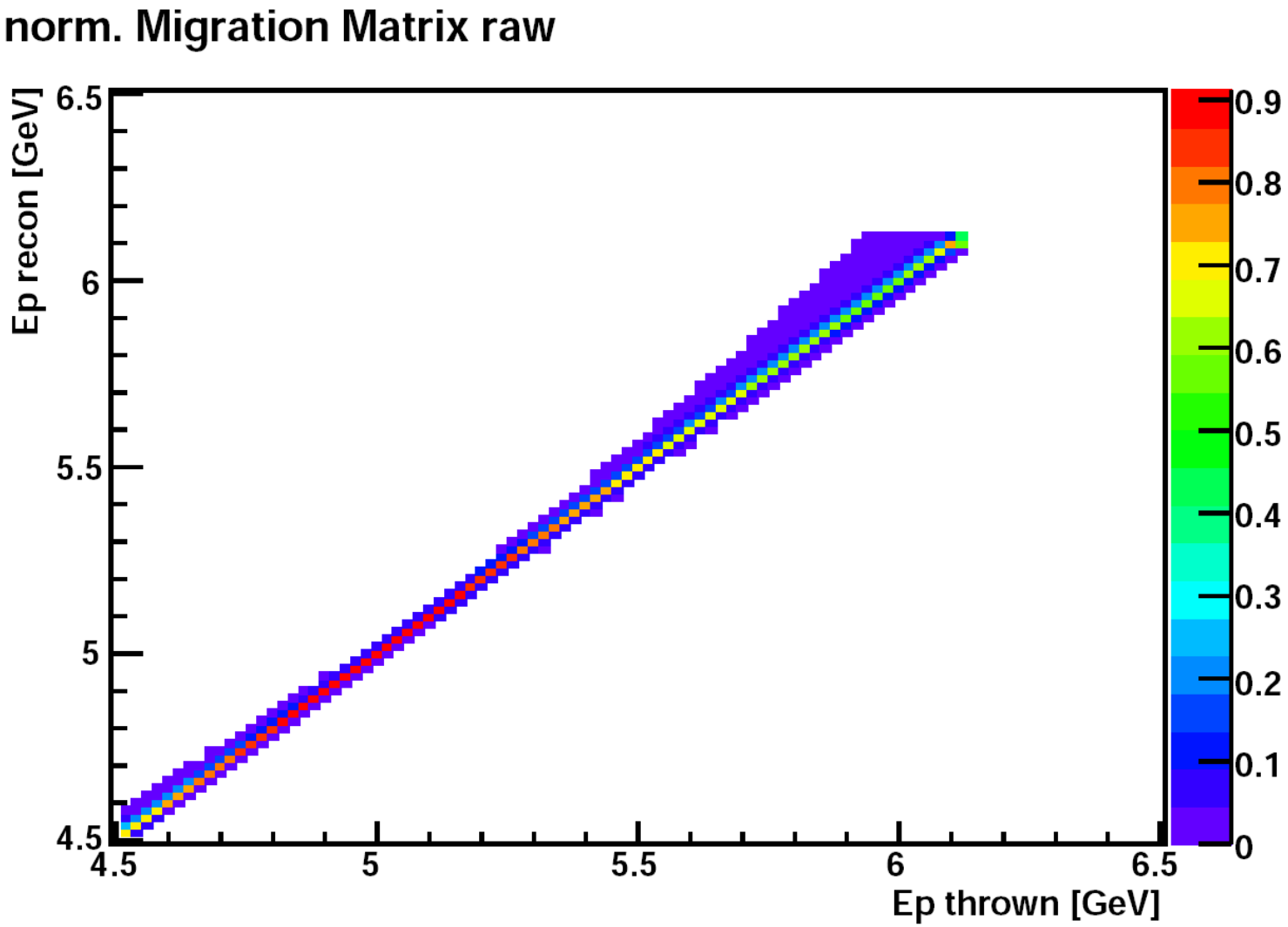
- One applies this acceptance to the data and thinks one has the "acceptance corrected yield" and proceeds further.

Acceptance (cont.)

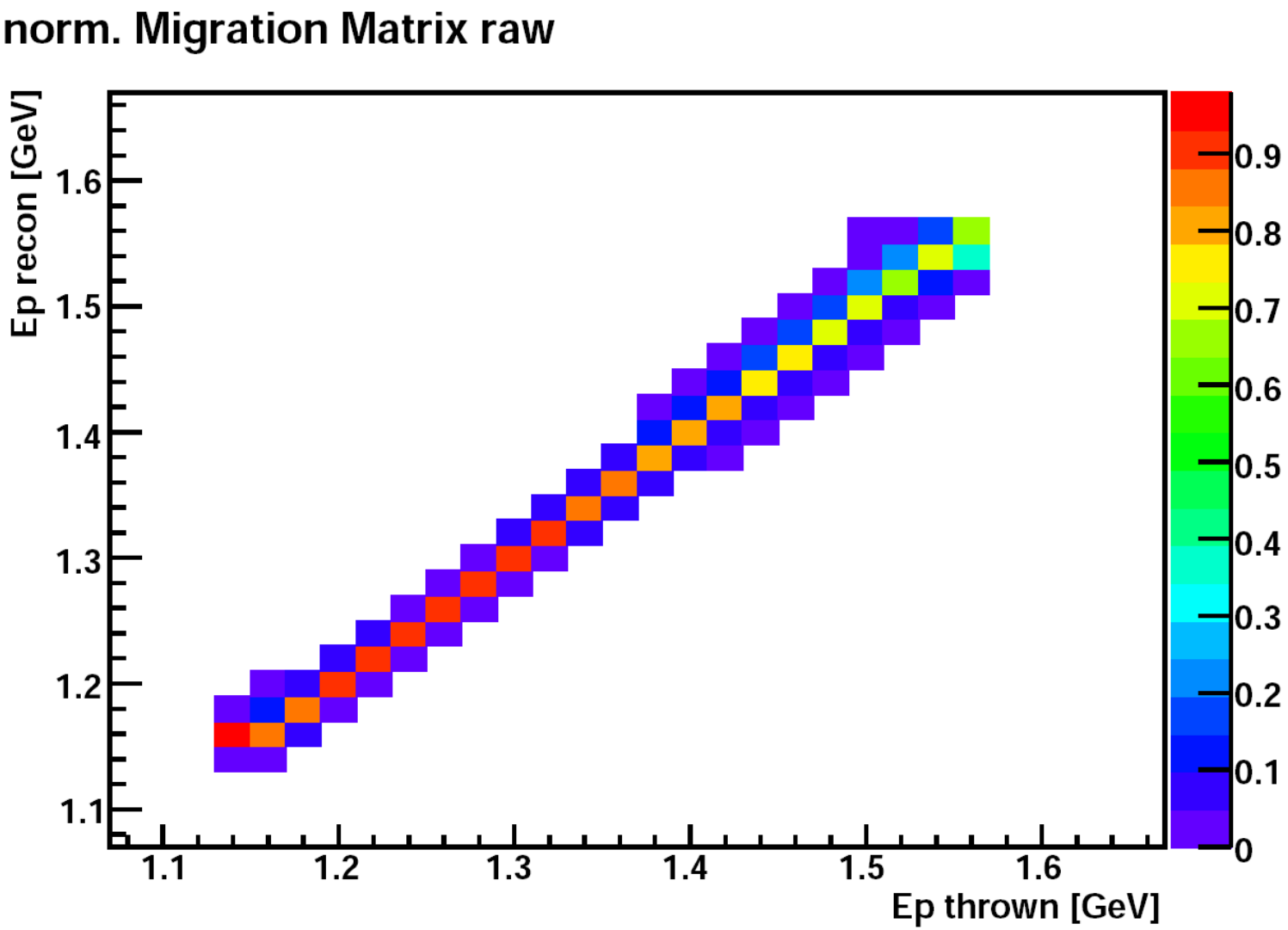
...

- One applies this acceptance to the data and thinks one has obtained the **"acceptance corrected yield"** and proceeds further.
- ... except that this is only **partially** true.
- would have been true if events generated in a bin end up in the same bin.
- there are many **"detector effects"** that make the statement above invalid: multiple scattering, detector resolution, round-offs, etc.
- the end result is "bin migration".
- What? in Hall C? Well...

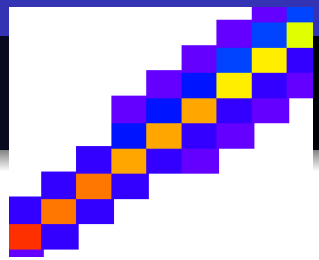
SHMS bin migration (I)



SHMS bin migration (II)



Bin Migration

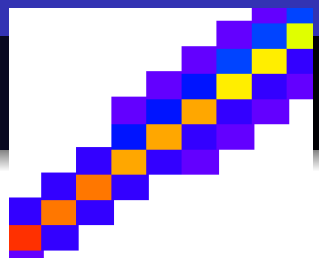


What can one do about this?

- **Note:** Situation does not improve going to higher dim. spaces - it gets worse*. It is not Hall C specific either.
- **Option 1:** Ignore it.
  - keep worrying about small(er) effects (BC) when much larger errors (of the same kind!) are pretty obvious.
  - keep quoting the uncertainties we've been quoting.
- **Options 2: Unfolding (deconvolution).**
  - Attempt to redistribute events to "undo" the "det. effects".
  - This is a "tricky" statistical problem.
  - It will require much thought and (for sure) software development.

Unfolding (II)

Unfolding (III)



G.N.'s $0.02:

- If we do go this route one expects the unfolding to take place fairly early in the analysis pipeline.
- bins no longer stat.–indep. (error matrix, anyone?).
- Fortunately there is a lot of HEP **literature & software** (that could possibly be adapted/adopted) on this topic:
  - S. Biondi: "Experience using unfolding prod. in ATLAS"
  - K. Dutta, D. Kar, D. Roy: "Unfolding with Generative Adversarial Networks"
  - V. Blobel: "Unfolding methods in Particle Physics"
  - S. Schmitt: "TUnfold, an algorithm for correcting migration effects in high energy physics". many others**

Conclusion (sort of)

G.N.'s $0.02:

- **simc** and **mc_single...** no longer need cernlib!
- **.root** output. size no longer a prob. rng **seed**.
- several tree–traversing options. **go parallel/MT!**
- like it (or not) bin migration is a problem, especially if we aim for small uncertainties. Further discussion/volunteers??
- ...
- **THANK YOU!**