LHAPDF

Recent developments, current issues and future plans

Andy Buckley

IPPP, Durham University

PDF4LHC meeting, 2009-05-28

In this talk…

- A bit of history
- Current issues with LHAPDF
  - Memory occupancy
  - Maintainability
  - Extensibility / standards
- Current situation: coping strategies
- Re-development plans
- Summary

I first gave this talk to CMS MC; there will be tech details, but it affects everyone so no apology!

LHAPDF history

pre-2001: Original HEP PDF library was PDFLIB, maintained as part of CERNLIB (in the LEP era). PDFLIB contained all set data in the code, and became unmaintainable

2001/2002: LHAPDF started development as a "Les Houches Accord" PDF library at the Les Houches HEP workshop. Main work by W. Giele. PDF parameterisations, runnings and boundary conditions stored in `.LHpdf` data files. DGLAP PDF RGE evolution in x & Q^2 via QCDNUM (by M. Botje) and CTEQ codes.

2003/2004: Management handed to Mike Whalley at Durham, as a component of the HepData grant. PDFLIB-a-like interface, LHAGLUE, by D. Bourilkov. Extensions to grids and multi-set initialisation, many more PDFs, drift towards larger `.LHgrid` interpolation grids rather than `.LHpdf` files, particularly for sets with large error eigensets. C++ interface by S. Gieseke.

LHAPDF history (ctd.)

2005–2007: AB starts contributing: build system improvements as part of CEDAR. Move to HepForge, re-written & portable C++ API, Python interface, regression testing… Extensions to QED NLO & gluino PDFs: hacky/inextensible.

2008/9: LCG deployments encountering memory difficulties with LHAPDF: “lite” versions. Proliferation of new sets continues: too many / too large to bundle, who makes the *worthiness* decision? Every new set requires code changes — a lot of work for the maintainers. **Time for PDF library version 3!**

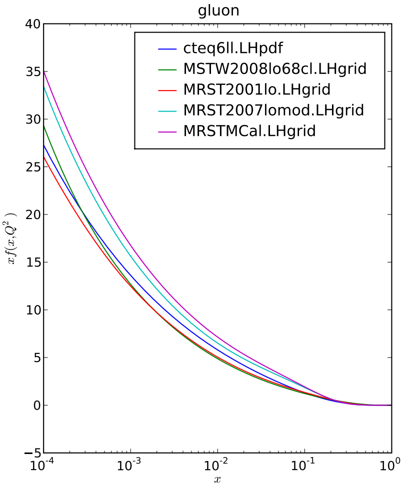
Recent additions: Thorne-Sherstnev mLO, HERA, NNPDF & MSTW 2008 (plus one more last week). Expect more MSTW, CTEQ etc. PDFs, plus LHC collaborations’ equivalents (LHCb, CMS, Atlas)…

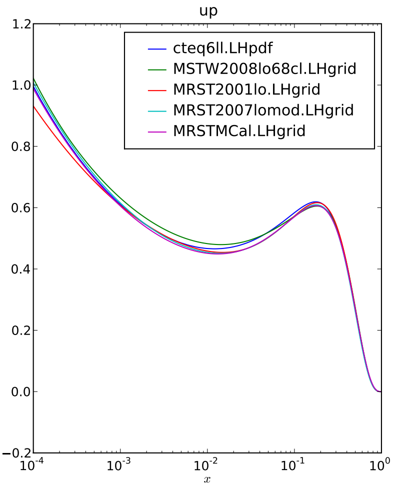
Coming in LHAPDF 5.7.1: bug fixes, no more set bundling (download via `lhapdf-getdata` script), mpl plotting examples

Some recent PDFs

mLO vs. LO $f(x, Q^2)_i$

From LHAPDF 5.7.0, $Q^2 = 10$ GeV^2:

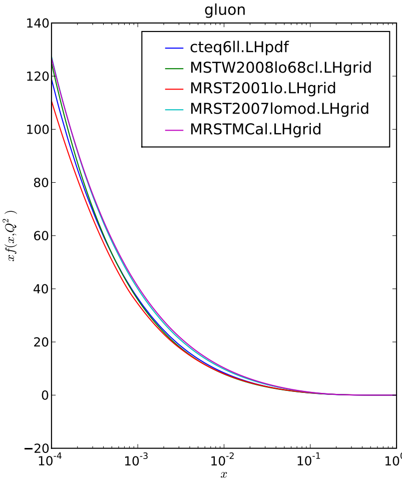


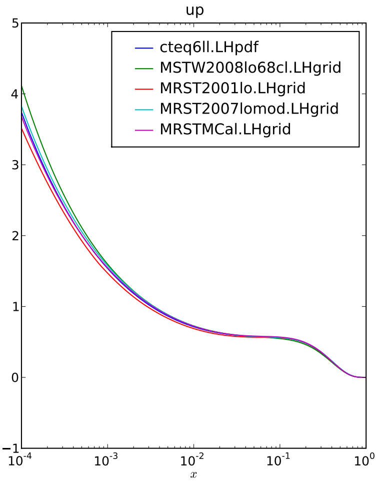


Some recent PDFs

mLO vs. LO $f(x, Q^2)_i$

From LHAPDF 5.7.0, $Q^2 = 6400\ \mathrm{GeV}^2$:





Current issues with LHAPDF

Memory occupancy

Fortran only has static allocation. Interpolation grids stored as huge working arrays; allocated as common blocks, i.e. huge (uninitialised) global array variables in C parlance.

Number of concurrent sets defined at compile time $\Rightarrow$ static memory footprint. Pre-5.5.1 used NSET = 3, without option to change. 3 sets allocated concurrently: BSS $>$ 400 MB.

Common blocks defined on an approx. set-by-set basis
i.e. even if using only one member of one set, global arrays are allocated for other (uninitialised) sets. Worse, arrays are sized for full sets, i.e. 40+ members for CTEQ6 and recent MRST/MSTW: 40 times larger than you probably want!

Current issues with LHAPDF

Memory occupancy (ctd.)

Is this really a problem?
Not for interactive users!
Uninitialised data areas declared as “BSS” in library, but a good OS won’t allocate that memory in RAM or swap to disk until code tries to use it (try it!) But falls foul of LCG process accounting: makes Grid MC production with LHAPDF difficult.

Response (LHAPDF 5.5.1): make number of concurrent sets specifiable at compile time. Can reduce BSS by $\times \sim 3$ with usability loss. “lite” mode can be set at compile time: reduce main multi-sets to single member array sizes.

Best: BSS ~ 130 MB. QCDNUM uses about 20 MB/set, CTEQ LHpdf uses about 3 MB/set.

Current issues with LHAPDF

Maintainability

LHAPDF currently supported as part of HepData grant + donated time. Sustainable, but not with the current work model.

High workload for maintainers +
limited development/maintenence resources = trouble!
Maintainer(s) have to add new code to the framework for each new PDF set. Grid file formats not standardised: each ipol code reads a different format.

How do we decide what sets are most important to devote time to integrating into the framework? How do we deal with a higher rate of new PDFs than the maintenence team can handle?

Plus, the original framework was not really designed carefully: plagued with global state, a maddening common block based configuration system etc.

Current issues with LHAPDF

Extensibility / standards

There are sets with an extra photon, extra gluino (!), etc. These have had to be hacked in rather inextensibly, by adding extra members to the returning $xf(x, Q^2)$ array for those sets if the right function is called. If you get the wrong set for that function call, LHAPDF will exit ungracefully! No programmatic way to protect against this.

To make it really extensible, a more general PID scheme (of which the PDG MC ID code is the only obvious contender) would be needed, with the possibility to check whether a particular flavour is supported (and to exit gracefully in C++ at least, via exceptions.)

Current situation: coping strategies

Any rewrite is going to take some time. How to cope with the Fortran version for now?

Be aware there is a fundamental size to PDF sets. If you want to use n sets at the same time, you will need at least $n \times$ set-size. x and Q^2 values vary sufficiently in MC code that just storing one part of the grid at any time won't work: code will be continually re-reading the grid file.

Typically, set-size = 30 MB. Larger sets (NNPDF_1000!) will need more, on the rough scale of the grid file size! Evolved sets, especially CTEQ evolution, use less runtime memory than gridded sets, but same BSS due to common arrays.

Current situation: coping strategies (ctd.)

Bearing this in mind, running in “lite” mode will reduce memory occupancy to $\sim$130 MB for 3 concurrent sets. 40 MB overhead isn’t *so* bad, and getting it any lower in Fortran will be hard: fragile work in legacy code.

Drastic action: “chopped” LHAPDF, only compiled with particular sets supported. Is it worth it for 40 MB! We won’t do this, encourage it or validate chopped versions for you. And no code forks of the Fortran version, i.e. you would be on your own.

But we have development plans of our own…

Re-development plans

For almost a year, I have been planning a rewrite to address the problems. Support has gradually built both inside and outside developer group: rewrite will involve contributions from AB, M. Whalley, D. Grellscheid, G. Watt. Design discussions and early coding work started.

Rewrite will be a **C++ core**, allowing dynamic memory allocation, wrapped by Fortran and other APIs (i.e. an inversion of the current design).

This will be a major, major upgrade. Probably a different library name to make it clear.

PDFLIB API fixed; others flexible to some degree — expect small user community for current non-LHAGLUE Fortran, C++ and Python interfaces. User community surveyed to assess degree of flexibility. MCnet MC authors prepared to move to new C++ interface.

Re-development plans

Memory management

Current "slots" system $\Rightarrow$ proper dynamic allocation.
Scope to use even more memory, but default will be smaller: you use what you ask for, and no more.

Set members rather than full sets as central PDF objects: scope for reducing memory further? Must still allow efficient set-wise initialisation.

Constraints: C++ design must be compatible with "the Fortran way", especially how the LHAGLUE interface works, because of legacy code expecting a PDFLIB-compatible API. So it must appear to PDFLIB client code that global, static set assignments are still happening, even though the true footprint is dynamic.

Re-development plans

Move to pure grid interpolation system

Standardise grid format. α_s ipol in grid files? Scope for extra ipol dimensions needed? Many issues if we don't want to re-re-design.

Ability to specify ipol scheme away from set defaults, by string (or function pointer in C++ API.) Several will be implemented, starting with MSTW 2008 ipol code. Onus is on set authors who want a particular interpolation algorithm to supply such an algorithm that matches the defined C++ `LHAPDF::Interpolator` API.

Memory structuring for most efficient use of main use case: all flavours at given x & Q^2. Use single cache line for all flavours.

Re-development plans

Open issues

Use PDG IDs and return flavour-indexed maps of $xf(x, Q^2)$ values. Possibility of asking a set about what flavours it supports.

Handling nuclear and other PDFs — special extra args. Set transformations.

Metadata in data files: set info, authors, description, URLs, contact email etc. more semantically that at present. Use code to generate documentation.

Reading files: file-per-member or single file per set, with member read pointers set during initial scan?

Sketch redesign

Survey results indicated that most users are using LHAPDF indirectly via MC event generators, i.e. LHAGLUE interface without direct common block access; some are using a subset of the Fortran LHAPDF API. Propose that we provide compatibility wrappers for these methods.

Core will be C++ `Pdf` class, providing interface to ask about supported flavours and calculate flavour-indexed maps of $xf(x, Q^2)$ values. Extra metadata also supported. One `Pdf` object will represent a single member.

Memory management will be dynamic, but a system will be provided to hide the dynamic (de)allocation details by initialising `Pdf` s with a name, e.g.

```
Pdf* mypdf = LHAPDF::initPdf(0, "cteq66");
```

This will be used to provide dynamic allocation from Fortran, and to emulate the current behaviour without the NSET restriction.

Summary

LHAPDF in its current form is unsustainable
Both for users and developers: memory issues, amount of work required per new set, etc.

Memory issues particularly a problem for Grid-based production
Due to Fortran static initialisation and historic use of common blocks (without actually using them “commonly”). Recent versions have provided compile time control of NSET and NHESS, allowing reductions in BSS at the cost of usability.

A rewrite is underway which will solve the problems
(or at least, pass the blame over: in future if you use too much memory, it’s because you requested too much concurrently!) But the rewrite will take time: maybe something for testing by mid/end of the summer.
Sorry, I’d have liked it to happen sooner, too.

Summary (ctd.)

In the meantime, workarounds will be needed.
Best option is probably to hack **`src/Makefile.am`** to only build the interpolation/evolution code for the set(s) that you are using.
Memory gains from ~ 30 MB to > 100 MB depending on how hard you work and what sets you are using.

But it will no longer be a general purpose PDF library and you won't be supported as if it was a vanilla install. Sorry!