# the APPLgrid project: hands on

Mark Sutton
University of Sussex

22nd October 2012

# What is APPLgrid?

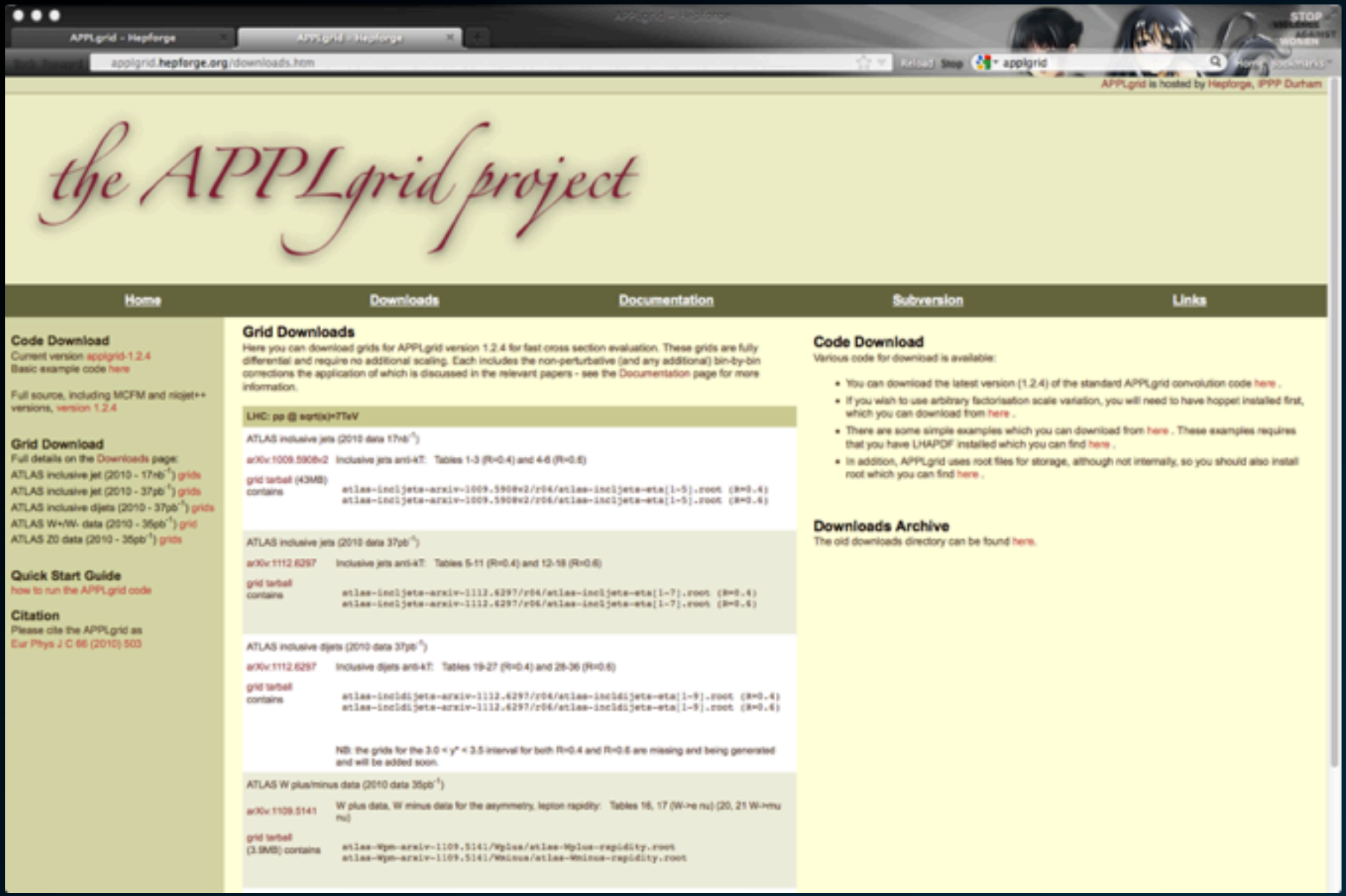
- APPLgrid is a fully open source package to build a library of utility classes for performing fast (N)NLO convolutions with PDFs written in C++

- Can be used for fast cross section production with pre-existing grids
  - Arbitrary renormalisation and factorisation scale variation
  - Arbitrary beam-energy rescaling
  - Different PDF sets

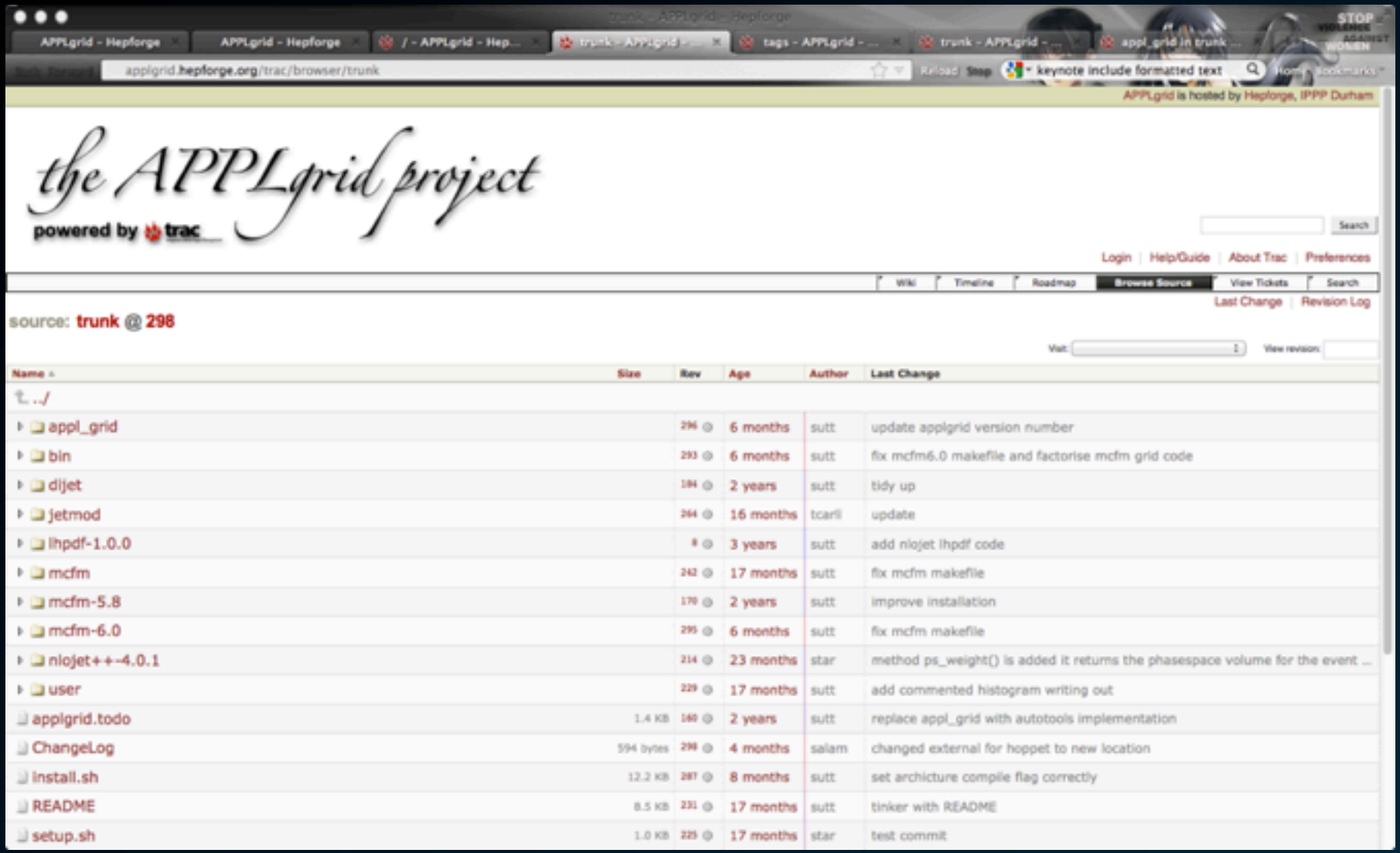
- Can be used by the user to generate custom grids for different cross sections and processed, using
  - NLOjet++ for jet production
  - MCFM for Electroweak boson production

# applgrid.hepforge.org

# Downloads


APPLgrid - Hepforge
applgrid.hepforge.org/downloads.htm
APPLgrid is hosted by Hepforge, IPPP Durham
the APPLgrid project
Home
Downloads
Documentation
Subversion
Links
Code Download
Current version applgrid-1.2.4
Basic example code here
Full source, including MCFM and nlojet++ versions, version 1.2.4
Grid Download
Full details on the Downloads page:
ATLAS inclusive jet (2010 - 17nb-1) grids
ATLAS inclusive jet (2010 - 37pb-1) grids
ATLAS inclusive dijets (2010 - 37pb-1) grids
ATLAS W+/W- data (2010 - 35pb-1) grid
ATLAS Z0 data (2010 - 35pb-1) grids
Quick Start Guide
how to run the APPLgrid code
Citation
Please cite the APPLgrid as
Eur Phys J C 66 (2010) 503
Grid Downloads
Here you can download grids for APPLgrid version 1.2.4 for fast cross section evaluation. These grids are fully differential and require no additional scaling. Each includes the non-perturbative (and any additional) bin-by-bin corrections the application of which is discussed in the relevant papers - see the Documentation page for more information.
LHC: pp @ sqrt(s)=7TeV
ATLAS inclusive jets (2010 data 17nb-1)
arXiv:1009.5908v2 Inclusive jets anti-kT: Tables 1-3 (R=0.4) and 4-6 (R=0.6)
grid tarball (43MB) contains
atlas-incljets-arxiv-1009.5908v2/r04/atlas-incljets-eta[1-5].root (R=0.4)
atlas-incljets-arxiv-1009.5908v2/r06/atlas-incljets-eta[1-5].root (R=0.6)
ATLAS inclusive jets (2010 data 37pb-1)
arXiv:1112.6297 Inclusive jets anti-kT: Tables 5-11 (R=0.4) and 12-18 (R=0.6)
grid tarball contains
atlas-incljets-arxiv-1112.6297/r04/atlas-incljets-eta[1-7].root (R=0.4)
atlas-incljets-arxiv-1112.6297/r06/atlas-incljets-eta[1-7].root (R=0.6)
ATLAS inclusive dijets (2010 data 37pb-1)
arXiv:1112.6297 Inclusive dijets anti-kT: Tables 19-27 (R=0.4) and 28-36 (R=0.6)
grid tarball contains
atlas-incldijets-arxiv-1112.6297/r04/atlas-incldijets-eta[1-9].root (R=0.4)
atlas-incldijets-arxiv-1112.6297/r06/atlas-incldijets-eta[1-9].root (R=0.6)
NB: the grids for the 3.0 < y* < 3.5 interval for both R=0.4 and R=0.6 are missing and being generated and will be added soon.
ATLAS W plus/minus data (2010 data 35pb-1)
arXiv:1109.5141 W plus data, W minus data for the asymmetry, lepton rapidity: Tables 16, 17 (W->e nu) (20, 21 W->mu nu)
grid tarball (3.9MB) contains
atlas-Wpm-arxiv-1109.5141/Wplus/atlas-Wplus-rapidity.root
atlas-Wpm-arxiv-1109.5141/Wminus/atlas-Wminus-rapidity.root
Code Download
Various code for download is available:
You can download the latest version (1.2.4) of the standard APPLgrid convolution code here .
If you wish to use arbitrary factorisation scale variation, you will need to have hoppet installed first, which you can download from here .
There are some simple examples which you can download from here . These examples requires that you have LHAPDF installed which you can find here .
In addition, APPLgrid uses root files for storage, although not internally, so you should also install root which you can find here .
Downloads Archive
The old downloads directory can be found here.

# Code


the APPLgrid project
powered by trac
source: trunk @ 298
Name
Size
Rev
Age
Author
Last Change
../
appl_grid 296 6 months sutt update applgrid version number
bin 293 6 months sutt fix mcfm6.0 makefile and factorise mcfm grid code
dijet 184 2 years sutt tidy up
jetmod 264 16 months tcarli update
lhpdf-1.0.0 8 3 years sutt add nlojet lhpdf code
mcfm 262 17 months sutt fix mcfm makefile
mcfm-5.8 170 2 years sutt improve installation
mcfm-6.0 295 6 months sutt fix mcfm makefile
nlojet++-4.0.1 214 23 months star method ps_weight() is added it returns the phasespace volume for the event
user 229 17 months sutt add commented histogram writing out
applgrid.todo 1.4 KB 160 2 years sutt replace appl_grid with autotools implementation
ChangeLog 594 bytes 298 4 months salam changed external for hoppet to new location
install.sh 12.2 KB 287 8 months sutt set archicture compile flag correctly
README 8.5 KB 231 17 months sutt tinker with README
setup.sh 1.0 KB 225 17 months star test commit

# Prerequisites

- APPLgrid requires some additional packages
  - **`root`** : ubiquitous histogramming package - used only for grid file storage and cross-section output **not** used for any operations internally.
  - **`hoppet`** : (optional) QCD evolution code used for QCD splitting functions to allow arbitrary factorisation scale variation. If you do not want to vary the factorisation scale, hoppet is not required
- Links to download both of these can be found from the **`Downloads`** page on the APPLgrid web site
- LHAPDF is **not** required - although the convolution interface uses the LHAPDF convention for the PDF evolution and $\alpha_s$ routines, and is needed for the examples

# class appl::grid

- Almost all interaction is via the **appl::grid** class

- A grid contains everything it (except the PDF) needed to perform the convolution and can be interrogated for any required information
  - Number of bins in the observable,
  - Bin limits i nthe observable
  - Number of subprocesses
  - Lowest order of calculation, loop order of calculation etc

- For each order of the calculation, the **grid** contains a number of subclasses to store the event weights
  - For each observable bin from the cross section, it has an array of subclasses - one instance for each "subprocess contribution"
  - The actual storage of the weights is in a custom sparse data structure to reduce the memory footprint and speed up the convolution
    - The class knows which elements are empty and does not store them, nor interogates them if they are requested.

- Can also include additional multiplicative corrections
  - bin-wise hadronisation corrections, K-factors etc

# The rest of the session ...

- Rest of the sessions divided as follows …
  - Downloading and installing the applgrid code
  - Running a simple convolution example
    - Including the multiplicative corrections
    - Different PDFs ...
  - More involved examples
    - Centre-of-mass energy rescaling
    - Renormalisation and factorisation scale variation
    - Subprocess contributions
  - fastNLO interface
  - If there is time…
    - Generating user grids

# Download and installation

- From the **applgrid.hepforge.org** or
  - **wget www.hepforge.org/archive/applgrid/applgrid-1.2.6.tar.gz**

- Compile ...
  - **tar -xzf applgrid-1.2.6.tar.gz**
  - **cd applgrid-1.2.6**
  - **./configure --prefix=/usr/local**
  - **make install**

- This builds the
  - **libAPPLgrid** and **libfAPPLgrid** libraries - **libfAPPLgrid** contains a FORTAN interface
  - It also installs the **applgrid-config** utility …

```
% applgrid-config --help
applgrid-config: configuration tool for the APPLgrid
                 fast cross section convolution code
                 http://projects.hepforge.org/applgrid/

Usage: applgrid-config [[--help|-h] | [--prefix] | [...]]
Options:
  --help | -h    : this help

  --prefix       : installation prefix (cf. autoconf)
  --incdir       : path to the APPLgrid header directory
  --libdir       : path to the APPLgrid library directory
  --cxxflags     : compiler flags for the C preprocessor
  --ldflags      : compiler flags for the linker including the fortan interface
  --ldcflags     : compiler flags for the linker just for c code

  --version      : release version number
```

# Simple example code

- Simple example code - example.tgz can be downloaded from the applgrid web pages or
  - **`wget www.hepforge.org/archive/applgrid/example.tgz`**
  - **`tar -xzf example.tgz`**
  - **`cd example`**
  - **`make`**

- Contains a simple example…
  - **`stand.cxx`**

```
#include "appl_grid/appl_grid.h"
```

```
appl::grid g("atlas-incljets-arxiv-1009.5908v2/r04/atlas-incljets-eta1.root");
g.trim(); // trim away uneeded memory
```

# Convolution

- Any convolution with a PDF set requires a PDF routine and a consistent $\alpha_S$ routine
  - APPLgrid uses the LHAPDF interface standard …

```
// lhapdf routines
#include "LHAPDF/LHAPDF.h"
extern "C" void evolvepdf_(const double& x, const double& Q, double* xf);
extern "C" double alphaspdf_(const double& Q);
```

  - Any routines that conform to this interface can be used with APPLgrid - user defined function can be used for PDF fitting
  - Actual convolution with a particular PDF set is performed by passing the PDF routines to the grid
    - More straightforward than requiring a user defined custom class
    - Output into a vector …

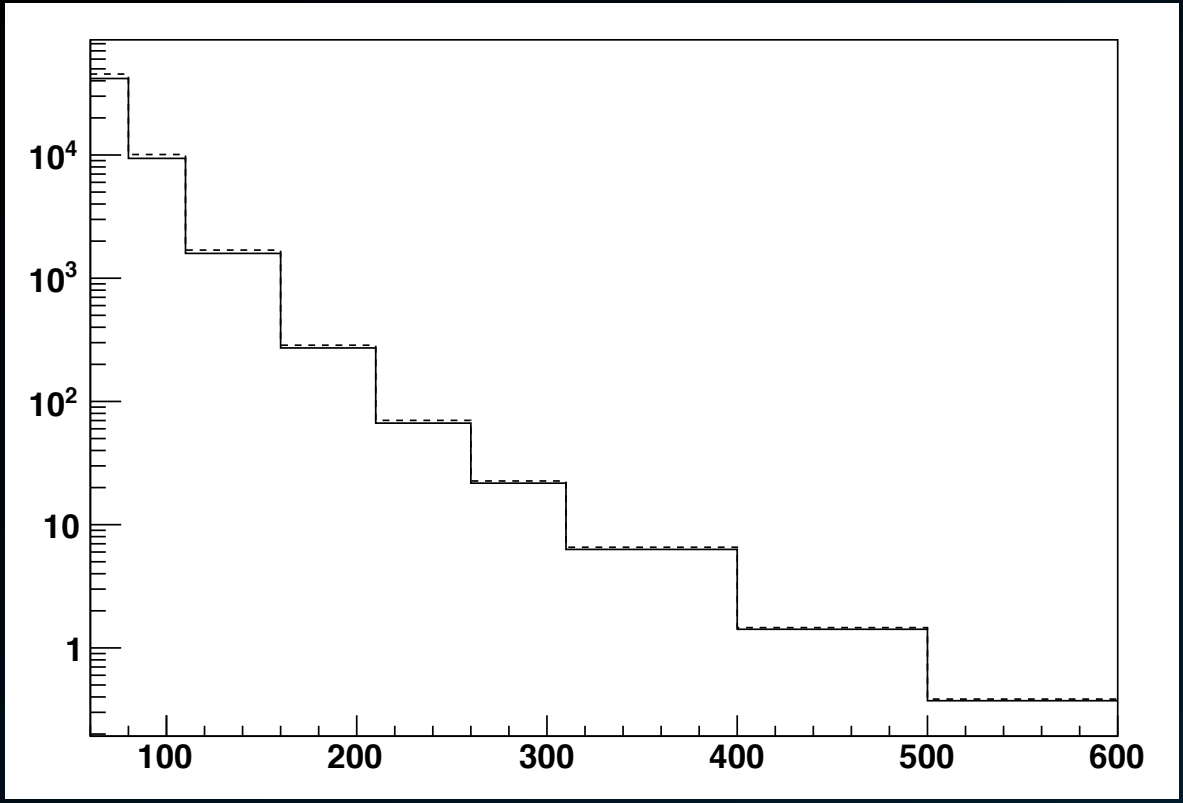
```
std::vector<double>  xsec = g.vconvolute( evolvepdf_, alphaspdf_ );
```

    - Or a root TH1D …

```
TH1D*  hxsec = g.convolute( evolvepdf_, alphaspdf_ );
```

- Different PDFs can be trivially used by passing in the pdf routine for the appropriate PDF
  - Limited only by your PDF package

# Multiplicative corrections




- Multiplicative corrections (disabled by default) can be enabled by first telling the grid to do so ...

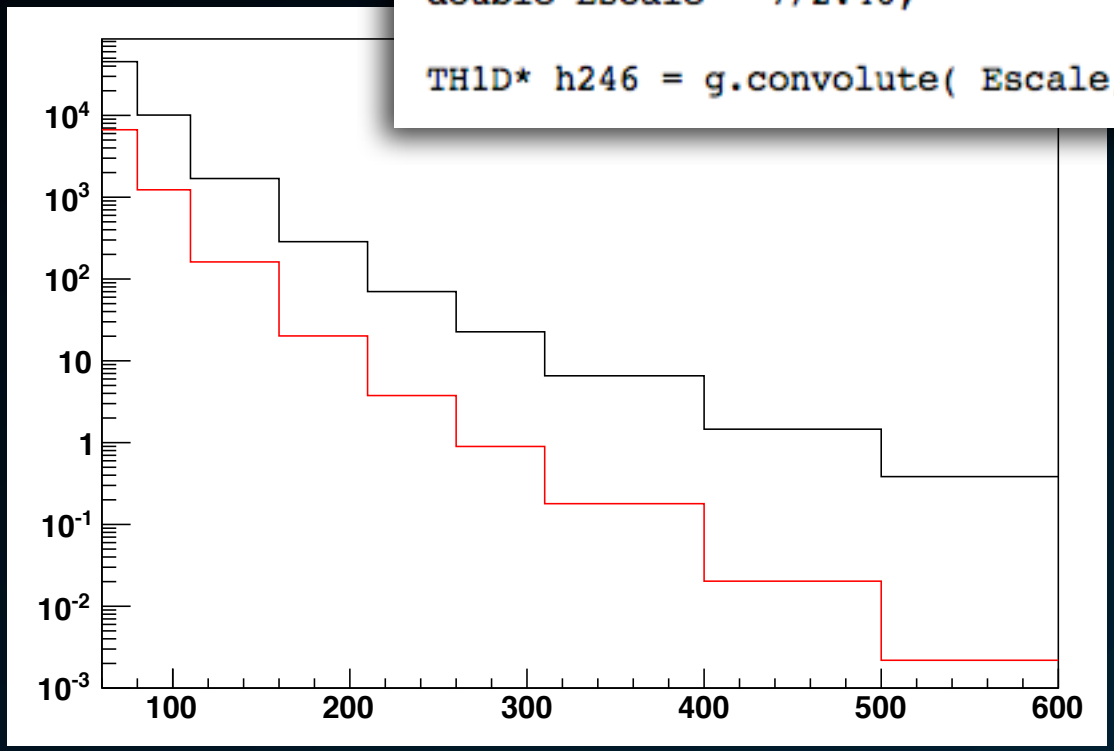
```
std::cout << g.getApplyCorrections() << std::endl;

g.setApplyCorrections(true);
```

# Centre-of-mass rescaling

- Performs the convolution as if the cross section were at a different centre-of-mass energy
  - NB: if the centre-of-mass energy is increased, the cross section might not be reliable since the grid might be missing contributions from the phase space outside the phase space with the lower centre of mass energy

- We (unfortunately) scale by 1/scale factor eg to scale from 7 TeV to 2.46 TeV use the scale factor 7/2.46

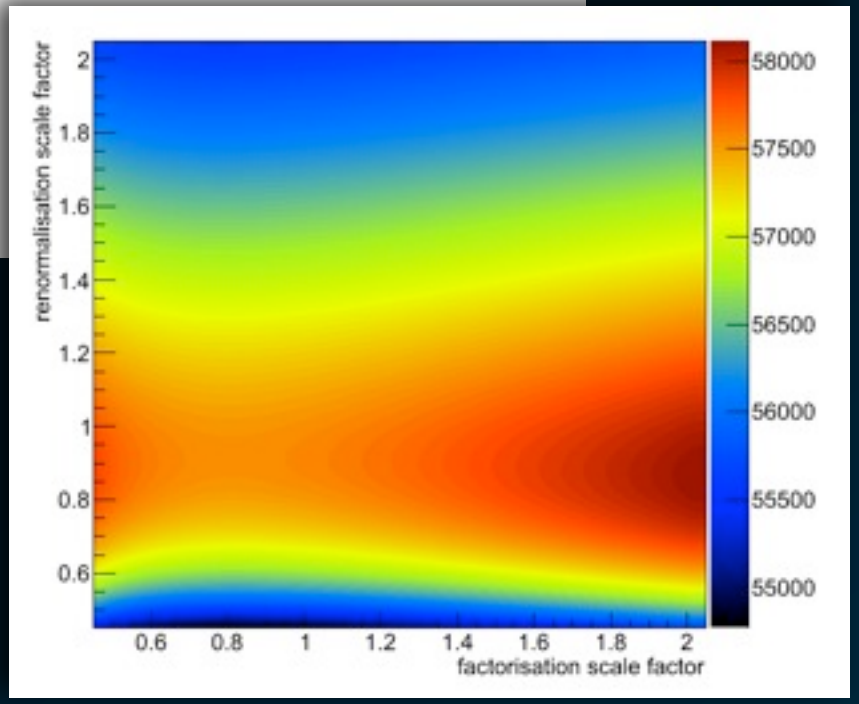
```
double Escale = 7/2.46;

TH1D* h246 = g.convolute( Escale, evolvepdf_, alphaspdf_ );
```



# Renormalisation and factorisation scales

- use a "scale factor" for each of the renormalisation and factorisation scales

```
int nloops = 1; // loop order

int Nbins = 50;
TH2D* hscale = new TH2D("scale", "", Nbins, 0.45, 2.05,  Nbins, 0.45, 2.05 );

for ( int i=1 ; i<=hscale->GetNbinsX() ; i++ ) {
  double fscale = hscale->GetXaxis()->GetBinCenter(i);

  for ( int j=1 ; j<=hscale->GetNbinsY() ; j++ ) {
    double rscale = hscale->GetYaxis()->GetBinCenter(j);

    std::vector<double> xs = g.vconvolute( evolvepdf_, alphaspdf_, nloops, rscale, fscale );

    double total = 0;
    for ( int k=xs.size() ; k-- ; ) total += xs[k];

    hscale->Fill( fscale, rscale, total );

  }
}
```



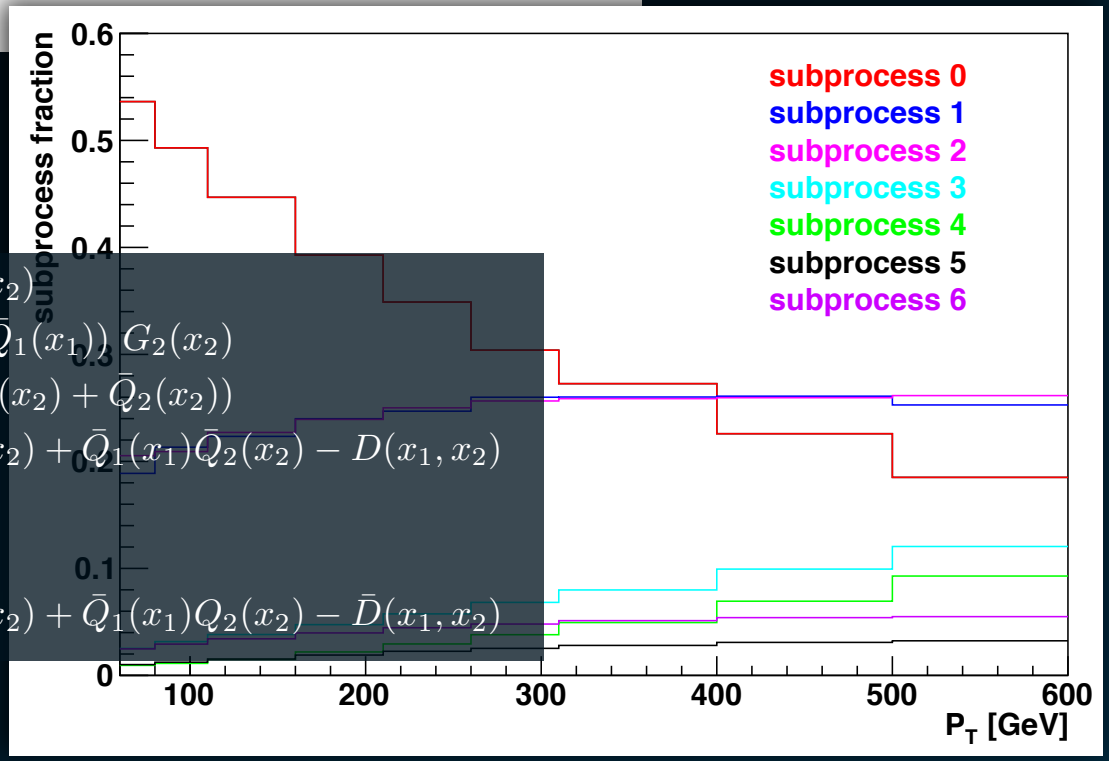
# Subprocess

- methods for the internal subprocess

```
std::vector<double> xsec = g.vconvolute( evolvepdf_, alphaspdf_);

std::vector<TH1D*> xsec_sub(g.subProcesses());

for ( int i=0 ; i<g.subProcesses() ; i++ ) {
  xsec_sub[i] = g.convolute_subproc( i, evolvepdf_, alphaspdf_);

  for ( int j=0 ; j<xsec_sub[i]->GetNbinsX() ; j++ ) {
    xsec_sub[i]->SetBinContent( j+1, xsec_sub[i]->GetBinContent(j+1)/xsec[j] );
  }
}
```

- Which actuals partons correspond to which subprocess depends on physics process under study, jets $Z^0$ etc ...

$$
\begin{aligned}
gg: \quad & F^{(0)}(x_1,x_2,Q^2) &=& \; G_1(x_1)G_2(x_2) \\
qg: \quad & F^{(1)}(x_1,x_2,Q^2) &=& \; (Q_1(x_1)+\bar{Q}_1(x_1))\,G_2(x_2) \\
gq: \quad & F^{(2)}(x_1,x_2,Q^2) &=& \; G_1(x_1)\,(Q_2(x_2)+\bar{Q}_2(x_2)) \\
qq': \quad & F^{(3)}(x_1,x_2,Q^2) &=& \; Q_1(x_1)Q_2(x_2)+\bar{Q}_1(x_1)\bar{Q}_2(x_2)-D(x_1,x_2) \\
qq: \quad & F^{(4)}(x_1,x_2,Q^2) &=& \; D(x_1,x_2) \\
q\bar{q}: \quad & F^{(5)}(x_1,x_2,Q^2) &=& \; \bar{D}(x_1,x_2) \\
q\bar{q}': \quad & F^{(6)}(x_1,x_2,Q^2) &=& \; Q_1(x_1)\bar{Q}_2(x_2)+\bar{Q}_1(x_1)Q_2(x_2)-\bar{D}(x_1,x_2)
\end{aligned}
$$



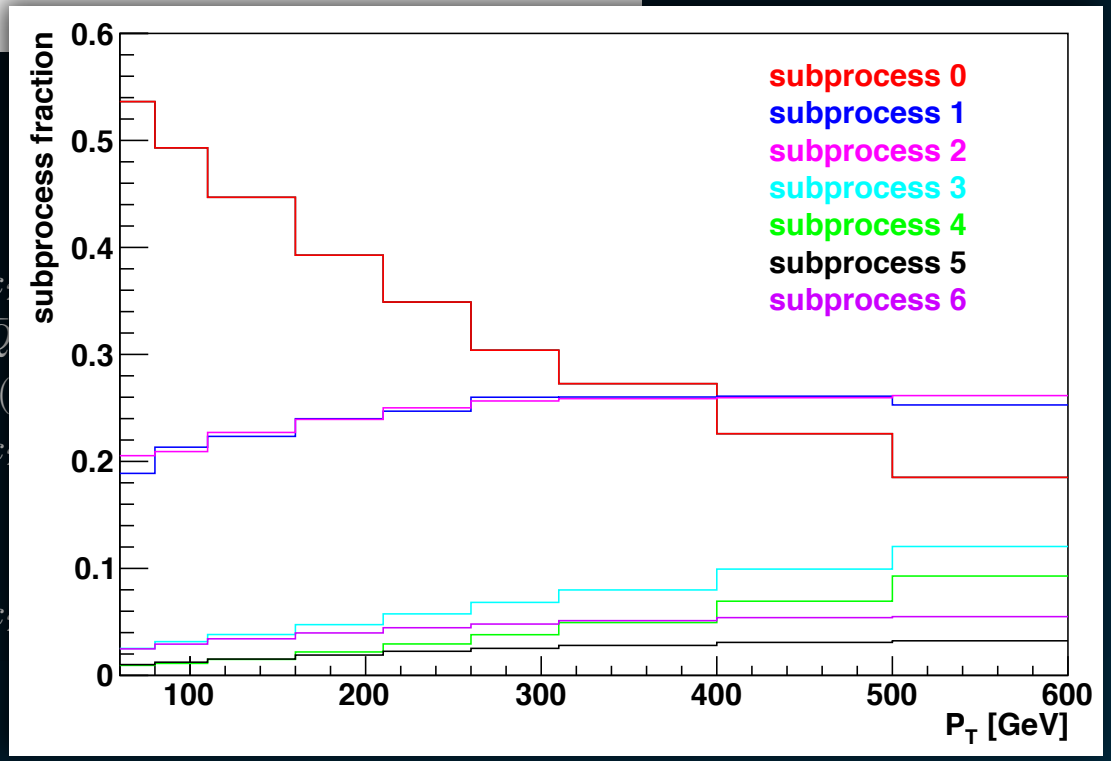
# Subprocess

- methods for the internal subprocess

```
std::vector<double> xsec = g.vconvolute( evolvepdf_, alphaspdf_);

std::vector<TH1D*> xsec_sub(g.subProcesses());

for ( int i=0 ; i<g.subProcesses() ; i++ ) {
  xsec_sub[i] = g.convolute_subproc( i, evolvepdf_, alphaspdf_);

  for ( int j=0 ; j<xsec_sub[i]->GetNbinsX() ; j++ ) {
    xsec_sub[i]->SetBinContent( j+1, xsec_sub[i]->GetBinContent(j+1)/xsec[j] );
  }
}
```

- Which actuals partons correspond to which subprocess depends on physics process under study, jets $Z^0$ etc ...

$$
\begin{aligned}
gg: \quad & F^{(0)}(x_1,x_2,Q^2) &=& \; G_1(x_1)G_2(x_2) \\
qg: \quad & F^{(1)}(x_1,x_2,Q^2) &=& \; (Q_1(x_1)+\bar{Q}_1(x_1))G_2(x_2) \\
gq: \quad & F^{(2)}(x_1,x_2,Q^2) &=& \; G_1(x_1)\,(Q_2(x_2)+\bar{Q}_2(x_2)) \\
qq': \quad & F^{(3)}(x_1,x_2,Q^2) &=& \; Q_1(x_1)Q_2(x_2)+\bar{Q}_1(x_1)\bar{Q}_2(x_2) \\
qq: \quad & F^{(4)}(x_1,x_2,Q^2) &=& \; D(x_1,x_2)+\bar{D}(x_1,x_2) \\
q\bar{q}: \quad & F^{(5)}(x_1,x_2,Q^2) &=& \; \bar{D}(x_1,x_2) \\
q\bar{q}': \quad & F^{(6)}(x_1,x_2,Q^2) &=& \; Q_1(x_1)\bar{Q}_2(x_2)+\bar{Q}_1(x_1)Q_2(x_2)
\end{aligned}
$$



# fastNLO interface

- There is an interface for fastNLO version 1 grids to allow centre-of-mass rescaling and arbitrary renormalisation and factorisation scale variation
  - simple example in **`fnmain.cxx`**

```
#include "appl_grid/fastnlo.h"
```

  - This reads the fastNLO file, and generates the appropriate number of **`appl::grids`** which can be used as usual

```
fastnlo f( gridname );

std::vector<appl::grid*> g = f.grids();
```

  - The number of differential distributions in the fastNLO scenario, or the number of bins in the cross-section need not be known in advance.

# How to use APPLgrid: FORTRAN interface

- But a FORTRAN interface exists…
  - First need to define PDF and $\alpha_S$ routines to be called by convolution

```
double precision function fnalphas(Q)
double precision Q
double precision alphaspdf
fnalphas = alphaspdf(Q)
return
end

subroutine fnpdf(x, Q, xf)
double precision x, Q
double precision xf(13)
call evolvePDF(x, Q, xf)
return
end
```

- No user types in FORTRAN
  - Need to identify grid somehow
- Returns an integer grid "id"
  - Needed to allow more than one grid
  - Full C++ grid interface not available

```
C---  lhapdf set
      integer iset
      integer idata

      double precision xsec(100)

C---  grid controlled grid id - needed to access specific grid
      integer igrid

      integer Nbins
      integer getnbins

      iset = 0

C---  set up pdf
      call initPDFSet("/usr/local/share/lhapdf/PDFsets/cteq6mE.LHgrid")
      call initpdf(iset)

C---  cross section
      call readgrid( igrid, "atlas-incljets04-eta1.root"//char(0) )

C---  how many bins in cross section for this grid ---
      Nbins = getnbins(igrid)

C---  convolute this grid ---
      call convolute( igrid, xsec)

C---  print out the results ---
      do idata=1, Nbins
         write(6,*) "xsec(", idata, ")=", xsec(idata)
      end do

C---  free the grid storage ---
      call releasegrids
```

- Better off using the native C++ interface
- Implement simple C interface for your FORTRAN code - Best of both worlds

# Grid generation using MCFM

- To generate grids, users must run an (N)NLO calculation
  - NLOjet++ for jet production
  - MCFM for Electroweak physics

- The intricacies of running either of these is far beyond the scope of this presentation, so here will concentrate only on the common APPLgrid techniques, with a brief example using MCFM

- One of the ideas with grid generation is to first run a test run to determine the phase space, and then do a full run to actually fill the grids, with the grid dimensions optimised from the test run
  - Caveat: The optimised grids are determined from the test run, so if breaking the full run into several parallel jobs, each must start from an **identical** copy of the grid from the test run to ensure the optimisation is the same so the grids can be combined later.

- Basics - six stages
  - Test run: i) create the grid, ii) fill the phase space, iii) write out
  - Full run: i) read in test grid, ii) fill with event weights, iii) write out

- Will discuss each in turn.

# `appl::grid()` constructor

```
grid(int NQ2=50,  double Q2min=10000.0, double Q2max=25000000.0,  int Q2order=5,
     int Nx=50,   double xmin=1e-5,     double xmax=0.9,          int xorder=5,
     int Nobs=20, double obsmin=100.0,  double obsmax=7000.0,
     string genpdf="mcfm_pdf",
     int leading_order=0, int nloops=1,
     string transform="f2");

grid( int Nobs, const double* obsbins,
      int NQ2=50,  double Q2min=10000.0, double Q2max=25000000.0, int Q2order=5,
      int Nx=50,   double xmin=1e-5,     double xmax=0.9,         int xorder=5,
      string genpdf="mcfm_pdf",
      int leading_order=0, int nloops=1,
      string transform="f2" );

grid( const vector<double> obs,
      int NQ2=50,  double Q2min=10000.0, double Q2max=25000000.0,   int Q2order=5,
      int Nx=50,   double xmin=1e-5,     double xmax=0.9,           int xorder=5,
      string genpdf="mcfm_pdf",
      int leading_order=0, int nloops=1,
      string transform="f2" );
```

- Can define the observable bins using either number of bins and limits, a limit C style array, or an std::vector
- Specify the number and range of $x_1$ (and $x_2$) and $Q^2$
- The (physics process dependent) function to determine the PDF independent subprocesses
- The leading order of the process
- The number of loops
- The transform to use for mapping from $x$ to the internal storage variable

# Filling and writing

- Writing the filled grid is essentially trivial ...

```
// save grid to specified file
void Write(const string& filename="weightgrid.root", const string& dirname="grid");
```

- Before writing, when running, once you have the weights in a C style array, you simply fill the phase space routine

```
void fill_phasespace(const double x1, const double x2, const double Q2,
                     const double obs,
                     const double* weight, const int iorder);
```

- To fill for the full run,

```
// update grid with one set of event weights
void fill(const double x1, const double x2, const double Q2,
          const double obs,
          const double* weight, const int iorder);
```

- Of course, obtaining the weights for each subprocess from the NLO code is far from trivial - code has been produced for this and is naturally distinct for MCFM and NLOjet++
  - Customised versions of MCFM and NLOjet++ are available where this is done
  - Discussion of the code beyond the scope of this tutorial - for the time being use as a black box

# MCFM example: Compilation and execution

- Compilation "should be" straightforward, on the virtual machine the code should already be instralled and compiled, but to compile it from scratch you would need
  - **`cd ~/applgrid-full/mcfm-6.0`**
  - **`./Install`**
  - **`make install`**

- This builds the executables in the **`mcfm-6.0/Bin`** directory …

```
% cd Bin
/home/school/applgrid-full/mcfm-6.0/Bin
% ls -l
total 20508
-rw-r--r-- 1 school school     2630 2012-10-20 23:00 BBbarinput.DAT
-rw-r--r-- 1 school school    79157 2012-10-20 23:00 br.sm1
-rw-r--r-- 1 school school    79157 2012-10-20 23:00 br.sm2
-rw-r--r-- 1 school school     2622 2012-10-20 23:00 CCbarinput.DAT
-rw-r--r-- 1 school school     2627 2012-10-20 23:00 input.DAT
-rwxrwxr-x 1 school school 20748700 2012-10-22 17:18 mcfm
drwxr-xr-x 2 school school     4096 2012-10-20 23:00 output
lrwxrwxrwx 1 school school       22 2012-10-20 23:24 PDFsets -> /usr/local/lib/PDFsets
-rw-r--r-- 1 school school    14260 2012-10-20 23:00 process.DAT
-rwxrwxr-x 1 school school    29361 2012-10-20 23:30 standSimple
-rw-r--r-- 1 school school     2621 2012-10-20 23:00 TTbarinput.DAT
-rw-r--r-- 1 school school     2626 2012-10-20 23:00 winput.DAT
-rw-r--r-- 1 school school     2625 2012-10-20 23:00 Wminput.DAT
-rw-r--r-- 1 school school     2625 2012-10-20 23:00 Wpinput.DAT
-rw-r--r-- 1 school school     2619 2012-10-20 23:00 Z0input.DAT
```

- The **`DAT`** files are the configuration files, you need to run at least twice (once to determine the phase space, once to fill the optimised phase space with actual event weights)
  - **`./mcfm Wpinput.DAT ; ./mcfm Wpinput.DAT`**

- It will read in an update the grid.

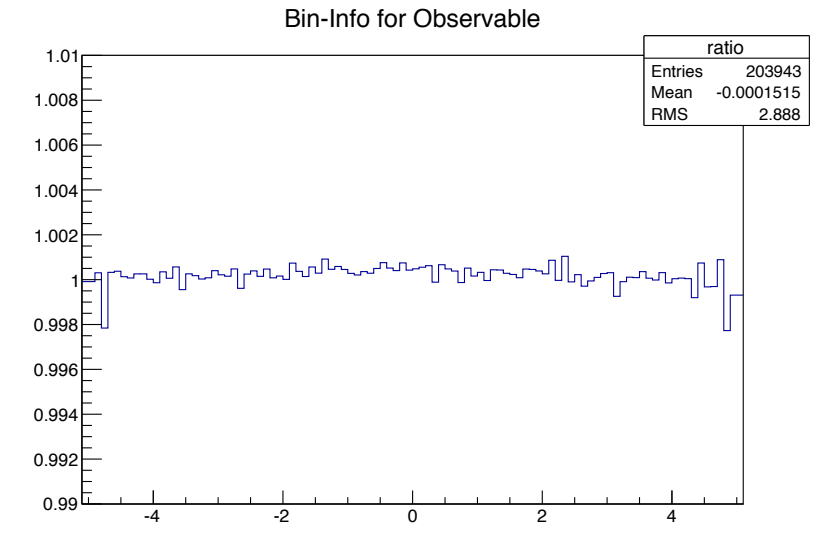
# Output

- The output grid files go into the **`Bin/output`** directory

```
% ls output
total 16560
-rw-r--r-- 1 school school 4286036 2012-10-22 17:37 grid-30-Wplus_eta3.root
-rw-r--r-- 1 school school 1383528 2012-10-22 17:36 grid-30-Wplus_eta3.root-save
-rw-r--r-- 1 school school 4251185 2012-10-22 17:37 grid-30-Wplus_eta4.root
-rw-r--r-- 1 school school 1381805 2012-10-22 17:36 grid-30-Wplus_eta4.root-save
-rw-r--r-- 1 school school 2062577 2012-10-22 17:37 grid-30-Wplus_pt3.root
-rw-r--r-- 1 school school  747334 2012-10-22 17:36 grid-30-Wplus_pt3.root-save
-rw-r--r-- 1 school school 2082743 2012-10-22 17:37 grid-30-Wplus_pt4.root
-rw-r--r-- 1 school school  749369 2012-10-22 17:36 grid-30-Wplus_pt4.root-save
```

- You can use the same executable **`stand`** as the previous examples, since the grids contain everything needed for the convolution, but we build another in the **`Bin`** directory which compares with a reference histogram …
  - **`./standSimple output/grid-30-Wplus_eta3.root`**

- This produces an output file with the fast convolution, a reference and a comparison …




- For the use who wants to define their own grids, or change the observable binning passed into the **`appl::grid(…)`** constructors etc, the code is in
  - **`src/User/gridwrap.cxx`**

# Combining output …

- Reading in a grid, or a number of grids are trivial
- Useful operators are defined to allow easy combination of grids…

```
// very lovely algebraic operators
grid& operator=(const grid& g);
grid& operator*=(const double& d);
grid& operator+=(const grid& g);
```

# Outlook

- Hopefully, this has been a useful introduction to the features of APPLgrid
- Installing and using the code to obtain cross sections is simple and straightforward
- A FORTRAN interface exists, but is only partially featured
  - Use of the FORTRAN interface (and indeed FORTRAN!) is strongly discouraged
- An increasing number of cross sections for LHC (ATLAS) jet and Electroweak data are available
- Happy cross section generation ...