# Rivet and RivetGun

## Toolkit for MC generator validation and tuning

Andy Buckley

Institute for Particle Physics Phenomenology
Durham University, UK

HERA-LHC, 2007-11-01

Durham University

# Introduction

# Outline



Durham University

# Generator parameters

Since generator models can't be exact, uncalculable parameters are unavoidable:

- Lund string params / cluster fragmentation mass
- Flavour composition of hadronization model
- Jet definition in multi-jet / ME merging
- Endpoint of parton cascade ($p_{T\text{min}}$ / $m_{\text{min}}$ / angular cutoff)
- Parton density functions (PDFs)
- Proton matter distribution (for UE modelling)

Models need to be validated/tuned using real data.

Durham University

# The real data: HepData

- Established archive of published HEP data from $\mathcal{O}(30yrs)$
- Concentrates on cross-sections and similar distributions: complementary to PDG
- Legacy database is being upgraded from FORTRAN-accessed BDB to a modern relational database by CEDAR
- **`http://projects.hepforge.org/hepdata`**

Durham University

# Rivet: a new tuning framework

- Rivet is a C++ replacement for FORTRAN HZTool
- Combination of tools, analysis handler and analyses
- Structure based on auto-cached **`Projections`** acting on HepMC events
- **`Analysis`** routines use projections to make distributions
- Histogramming etc. via AIDA interfaces
- AGILe/RivetGun in place of HZSteer
- http://projects.hepforge.org/rivet

Durham University

# Getting started

# Getting Rivet

Requirements: LHAPDF, HepMC, HepPDT, (KtJet, CLHEP) + gens

- Downloadable from **`http://www.hepforge.org/downloads/`**
- Development in public SVN on HepForge
- Easy way: use bootstrap script from **`http://projects.hepforge.org/rivet/rivet-bootstrap`**
- This downloads dependencies, builds and installs them in the right order.

Durham University

# Using rivet-bootstrap

```
$ ./rivet-bootstrap $PWD/local
$ export LD_LIBRARY_PATH=$PWD/local/lib:$LD_-
LIBRARY_PATH
$ export PATH=$PWD/local/bin:$PATH
$ which rivetgun
/home/buckley/testbootstrap/local/bin/rivetgun
$ rivetgun -h
```

Durham University

# Building and installing Rivet

- Rivet (and other CEDAR packages) are built with GNU autotools and libtool
- (If from SVN, **`autoreconf -i`**)
- **`./configure -prefix=/prefix/path && make && make install`**
- That's all!

# Available Projections in Rivet

Projections do the calculations.

- Thrust, sphericity, C & D params...
- FinalState, VetoedFS, ChargedFS, HadronicFS
- KtJets, D0 legacy cone, FastJet
- Multiplicities, DIS...

Durham University

# Available Analyses in Rivet

An analysis corresponds to an experimental paper.

- LEP: ZPhys73C11 (event shapes), PL273B181 (mults.)
- HERA: HepEx9506012 (energy flow), HepEx0112029 (dijet photoproduction)
- Tevatron: HepEx0107012 (W/Z diff $\sigma$), HepEx0409040 (jet ang. decorr.), HepEx0505013 (jet shapes), HepEx0701051, PRD65092002 (UE)
- TestAnalysis (demo), ExampleTree (ROOT demo)

Durham University

# Available Generators in AGILe

AGILe (A Generator Interface Library) provides uniform C++ interfaces to various generators.

- FHerwig (+ Jimmy), FPythia
- (+ AlpGen and Charybdis)
- Pythia 8, (Herwig++, Sherpa)

Dynamic loading means that multiple generators from the same "series" can be swapped at runtime.

# A RivetGun session

**`rivetgun`** steers generators via AGILe and runs Rivet analyses

- Generate events and run generator using RivetGun:

```
$ rivetgun -g FPythia:6411 -n 50000
-a HEPEX0409040 -H MyHistos
-beam1 PROTON -mom1 980
-beam2 ANTIPROTON -mom2 980
-P fpythia.params -l RivetGun:WARN
```

- Read event files with JAS, PAIDA or other tool:

```
$ jas3 Rivet.aida
```

ROOT output is optionally available.

Durham University

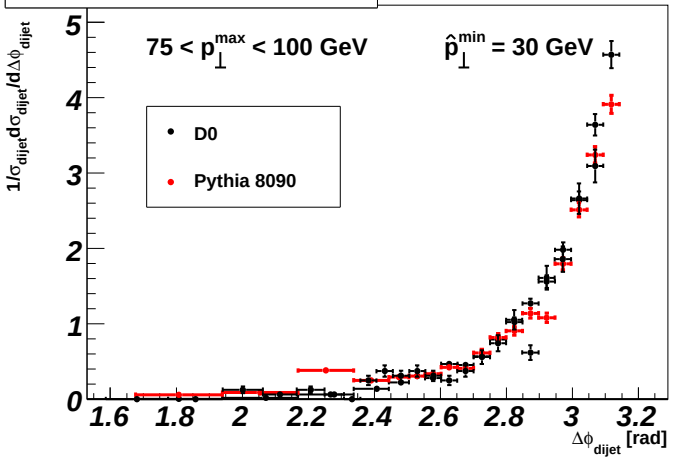
# More RivetGun flags

- `-h`: help, including available analyses and gens
- `-P file`: read in params from a text file
- `-p key=val`: specify a param on the command line
- `-H name`: choose output histo file base name
- `-o file`: write outHepMC events to file
- `-l hier.name:LEVEL`: control logging level

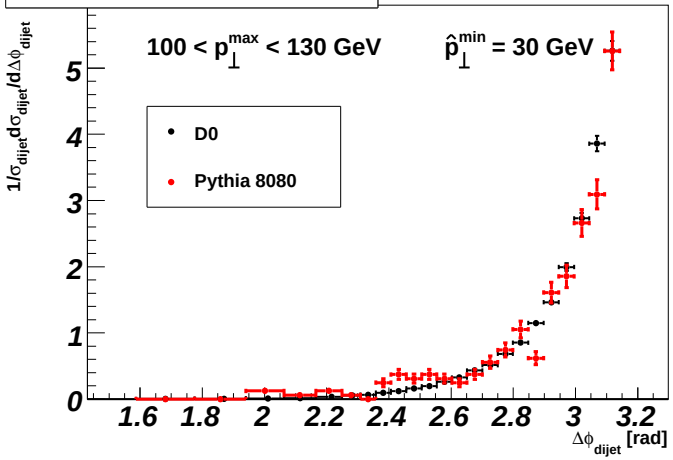
Durham University

# Demo plots (Pythia 8)

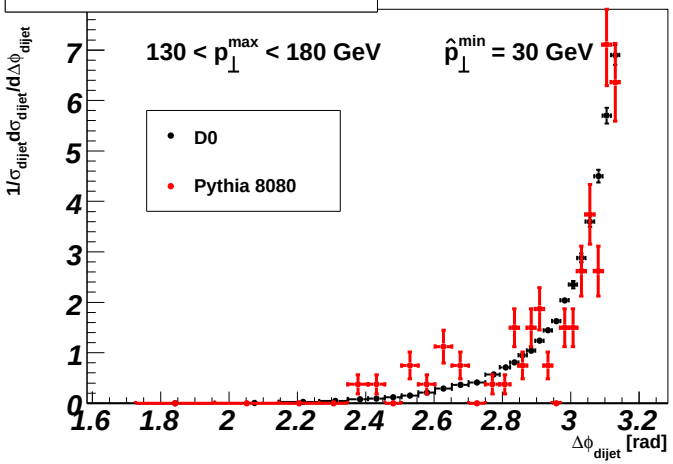
HepData/HepEx0409040/ds1-x2-y1



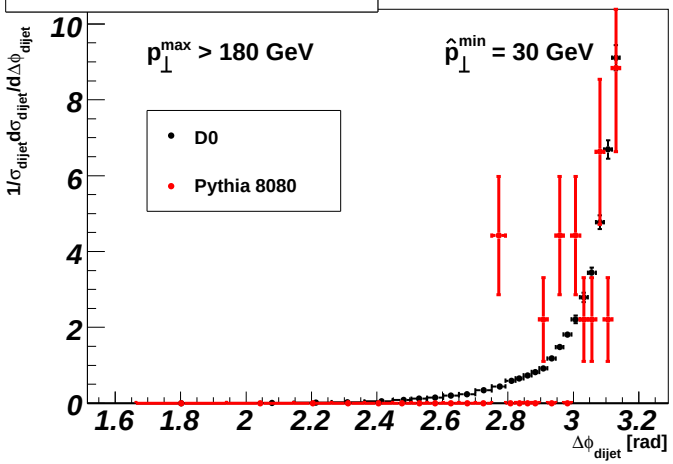

HepData/HepEx0409040/ds2-x2-y1




HepData/HepEx0409040/ds3-x2-y1




HepData/HepEx0409040/ds4-x2-y1




Durham University

# Details

# How Rivet works

1. Event reference handed to **`AnalysisHandler`**
2. First event applied: applies projections
3. Each projection is attached to the event, after running **`analyze`**
4. Subsequent projections will be compared against attached projs. of same type (RTTI)
5. **`MyProj::compare(MyProj)`** defines comparison behaviour
6. Fill histos. . . next event.

Durham University

# A Rivet `Projection`

## include/Rivet/Projections/Multiplicity.hh

```
#include "Rivet/Projection.hh"
#include "Rivet/Projections/FinalState.hh"

/// Count the final-state particles in an event.
class Multiplicity : public Projection {
public:

  inline Multiplicity(FinalState& fsp)
    : _totalMult(0), _totalChMult(0), _totalUnchMult(0),
      _hadMult(0), _hadChMult(0), _hadUnchMult(0),
      _fsproj(fsp) {
    addProjection(fsp);
  }

  /// Return the name of the projection
  inline string getName() const { return "Multiplicity"; }
```

# A Rivet `Projection`

## `include/Rivet/Projections/Multiplicity.hh`

```
protected:
    void project(const Event& e);
    int compare(const Projection& p) const;
    inline const unsigned int totalMultiplicity() const { return _tota

private:
  unsigned int _totalMult, _totalChMult, _totalUnchMult;
  FinalState _fsproj;
};
```

Durham University

# A Rivet `Projection`

## `src/Projections/Multiplicity.cc`

```
int Multiplicity::compare(const Projection& p) const {
  const Multiplicity& other = dynamic_cast<const Multiplicity&>(p);
  return pcmp(*_fsproj, *other._fsproj);
}

void Multiplicity::project(const Event& e) {
  _totalMult = 0;
  const FinalState& fs = e.applyProjection(_fsproj);

  for (ParticleVector::const_iterator p = fs.particles().begin();
       p != fs.particles().end(); ++p) {
    ++_totalMult;
    ...
  }
}
```

Durham University

# A Rivet **Analysis**

## include/Rivet/Analyses/TestAnalysis.hh

```
#include "Rivet/Analysis.hh"
#include "Rivet/Projections/Multiplicity.hh"
#include "Rivet/Projections/Thrust.hh"

class TestAnalysis : public Analysis {
public:
  TestAnalysis()
    : _multproj(_fsproj), _thrustproj(_fsproj)
  {
    addProjection(_fsproj);
    addProjection(_multproj);
    addProjection(_thrustproj);
  }

  static Analysis* create() { return new TestAnalysis(); }
  string getName() const { return "Test"; }
  void init();
  void analyze(const Event& event);
  void finalize();
```

Durham University

# A Rivet **Analysis**

## include/Rivet/Analyses/TestAnalysis.hh

```
private:
    FinalState _fsproj;
    Multiplicity _multproj;
    Thrust _thrustproj;

private:
    TestAnalysis& operator=(const TestAnalysis&);
    AIDA::IHistogram1D* _histTot;
  ...
};
```

Durham University

# A Rivet **Analysis**

## src/Analyses/TestAnalysis.cc

```
void TestAnalysis::init() {
  _histTot          = bookHistogram1D("TotalMult",
    "Total multiplicity", 100, -0.5, 99.5);
  _histChTot        = bookHistogram1D(1, 1, 3, "Total charged multiplic
  ...
}

void TestAnalysis::analyze(const Event& event) {
  Log log = getLog();

  const Multiplicity& m = event.applyProjection(_multproj);
  log << Log::DEBUG << "Total multiplicity = "
      << m.totalMultiplicity() << endl;
  const Thrust& t = event.applyProjection(_thrustproj);
  log << Log::DEBUG << "Thrust = " << t.thrust() << endl;

  const double weight = event.weight();
  _histTot->fill(m.totalMultiplicity(), weight);
  ...
}
```

Durham University

# A Rivet `Analysis`

## `src/Analyses/TestAnalysis.cc`

```
void TestAnalysis::finalize() {
  normalize(_histTot);
}
```

Durham University

# Cuts and BeamConstraints

- Projections and analyses specify cuts and beam constraints
- Cuts: binary cut ranges on named variables
- BeamConstraint: order-independent pair of allowed beams, inc. `ALL` wildcard
- Each projection and analysis contains an acyclic graph of other projections
- Analyses can be queried about minimal cut ranges, allowed biam types, internal consistency etc.
- Useable by e.g. RivetGun. Extension planned.

Durham University

# Histogramming

- Histogramming primarily via AIDA (for now)
- 1D histos, 1D profile, 2D `DataPointSet` supported
- Output as AIDA XML files or “flat” text: easily readable
- Optional ROOT I/O

Durham University

# Histogram autobooking

- Histos can be booked by
  - unifomly dividing a range
  - manually specifying bin edges
- This sucks: manual binning is tedious and fragile
- Better: book binnings from reference data!
- Each analysis comes with ref data as AIDA XML: book by specifying dataset, x-axis and y-axis

Durham University

# External (plugin) analyses

- You now don't need to modify **`libRivet`** to write an analysis
- At runtime, `.`, **`LD_LIBRARY_PATH`** and **`RIVET_LIBRARY_PATH`** are scanned for **`libRivet*.so`**
- If such a file contains **`dlopen`**able analyses, they are added to the list available with **`rivetgun -h`**
- Projections are not "pluggable"
- See Rivet wiki for more developer info

Durham University

# AGILe

- AGILe **`Generator`** interface defines basic messages to which gens respond
- **`setInitialState(...)`**, **`setSeed(...)`**, **`setParam(...)`**, **`makeEvent(...)`**...
- Plus some “behind the scenes” behaviours
- Interface now stabilising: generator state I/O remains to be done
- Rather annoying that this is necessary at all!

Durham University

# The bigger picture

# Sociology

- What we need is users!
- GENSER, CEDAR, Professor so far — grass-roots interest?
- Migrating from HZTool now? YES!
- How to minimise the lethargy/inertia barrier? :-)
- More HERA analyses. RHIC data? EW, hadronisation...

Durham University

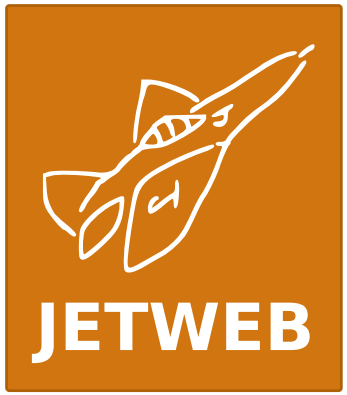
# Rivet/RivetGun development site

**http://projects.hepforge.org/rivet**

# User 1: JetWeb

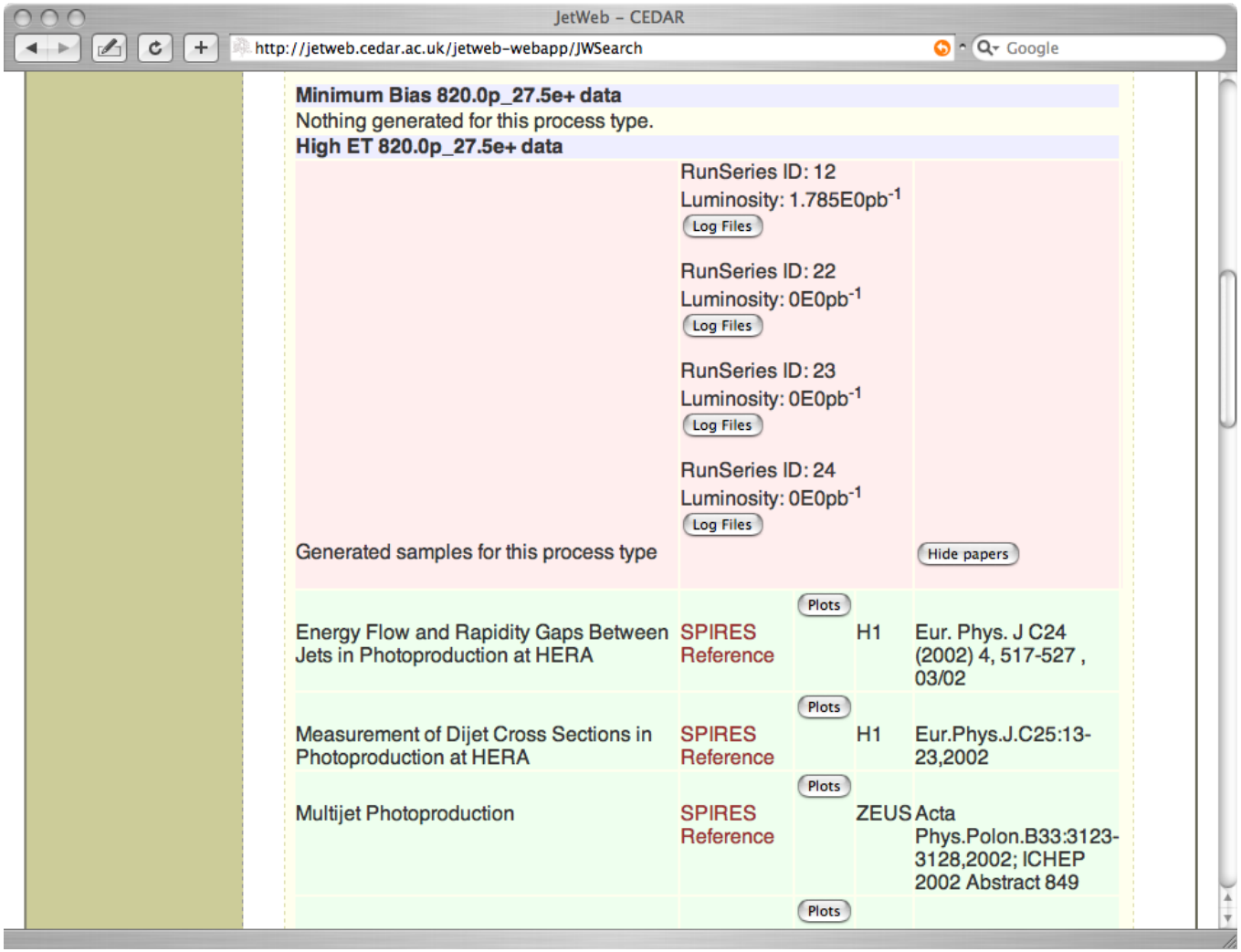
The last part of “CEDAR-proper” is an archive of simulated analysis results, indexed by model parameters: JetWeb

- Java-based: uses MySQL and Apache Tomcat as a back-end
- AIDA rendering of distributions
- Obtains reference data direct from HepData using HD object model
- Machinery for generating mixed-run distributions
- Can generate processes / extra stats on Web-user request
- Grid-based distribution of simulation jobs

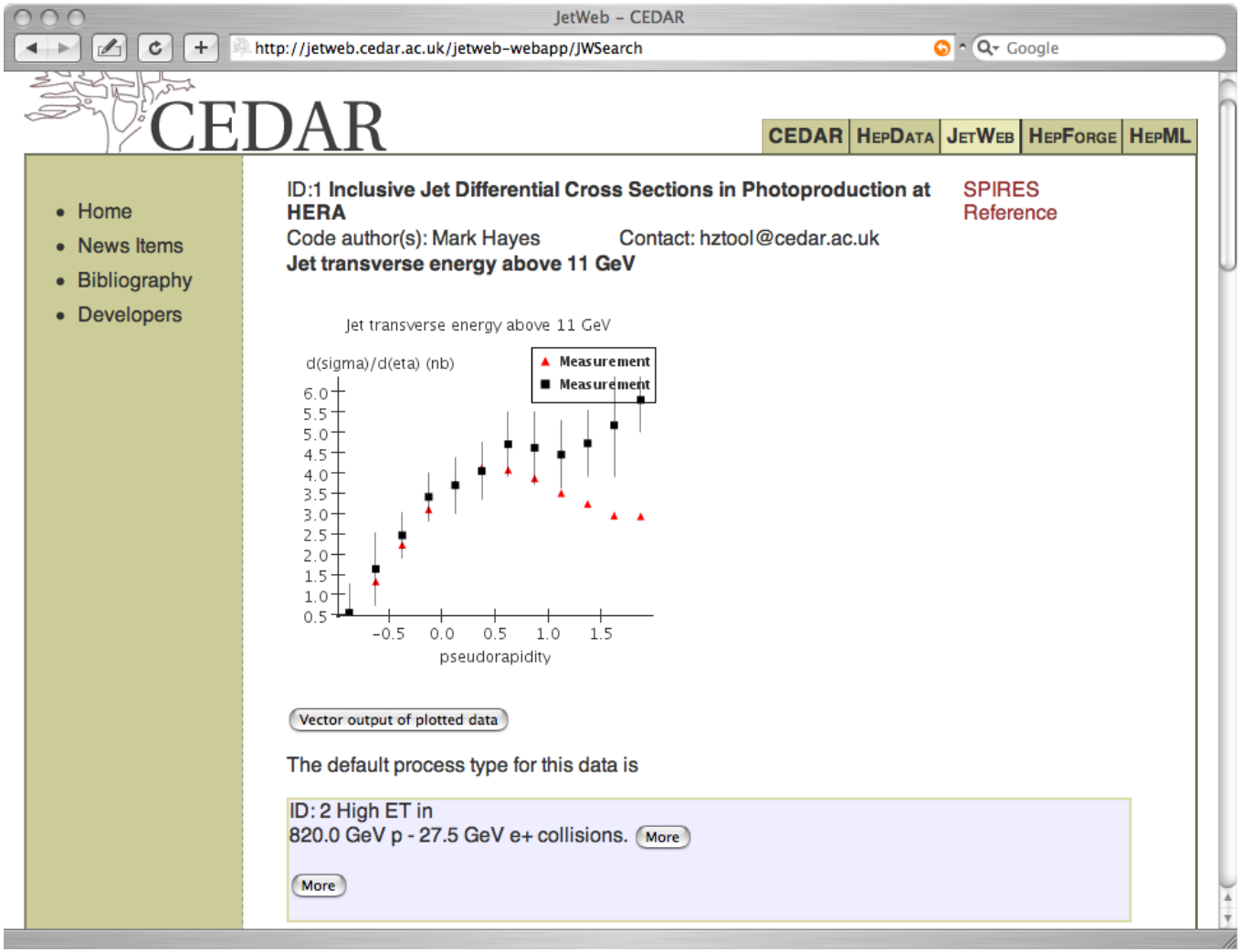



Durham University

# JetWeb screenshots - papers



# JetWeb screenshots - plots



# User 2: Professor tuner

- As so often happens, params are highly correlated
- So no point in tuning one param at a time...
- We have a high-dimensional parameter space, $n > \mathcal{O}(10)$
- Data from LEP, RHIC, HERA, (Tevatron) useful
- Delphi tuning: fit MC results to quadratic in $n$ variables:

$$X_{MC}(\vec{p}) = A_0 + \sum_{i=1}^{n} B_i p_i + \sum_{i=1}^{n} \sum_{j=i}^{n} C_{ij} p_i p_j$$

- Being reimplemented with Rivet machinery: AB + H. Hoeth (Wuppertal) + H. Lacker et al (Dresden)

Durham University

# Try it for yourself

# Playing with Rivet

- The world is your oyster!
- Start by running some simple built-in analyses and view with JAS3/gnuplot
- Change gen params, e.g. Pythia **`CKIN(3)`** ($k_\perp^{\text{min}}$), **`PARJ(41-42)`** (Lund params), **`PARJ(81-82)`** ($\Lambda_{\text{QCD}}$, $m_{\text{min}}$)
- Scan a param grid with a shell one-liner

Durham University