jHepWork
a java-based analysis framework

S.Chekanov (DESY/ANL)

HERA-LHC workshop
March 2007
DESY

2 choices for data analysis

- ROOT/C++: Main choice for LHC:
  - Very powerful, based on C++
  - Drawback: Not multiplatform. Should be compiled for every platform (I'm compiling ROOT several times every year for each version of FC and SUSE and for every major version of ROOT).
  - Too difficult to find out which method belongs to which object.
- JAS based on JAIDA:
  - JAVA based. Multiplatform. No need to compile.
  - Plenty of HEP libraries. Can read ROOT files
  - R&D for ILC in USA use JAS and FreeHEP
  - Drawback: graphics still behind ROOT
- Do we need something else for data analysis? If we do need, this has to be with good graphics. And this will likely to be JAVA..

JAVA vs C++

- **Multiplatform and VERY stable:**
  - Once compiled, works everywhere and every time (I have still running java programs written 10 years ago, without need for recompilation)
  - Very good chance that once your JAVA library is written and compiled, it will work in 10, 20.. years from now without need for any support. I suspect that it can even be recompiled from sources..
- **Most popular language:**
  - Freshmeat.net: 5300 JAVA vs 4500 C++ projects
  - Remember: “popularity” was among motivations to move to C++
- **Better structured and simpler than C++**
- **Garbage collection. Powerful Java reflection technology**
- **Can be embedded to the WEB:**
  - important for distributed analysis environment. This is what we need now!
- **Python can be used with JAVA (i.e. Jython)**
- **Java and Python and currently used in several domains at LHC:**
  - **Example: ATLAS event display (JAVA), Athenaeum (Distributed Athena). See Java in ATLAS Wiki (WWW)**

JAS

- **JAS – JAVA analysis framework**
- **Very powerful fit package**
  - interactive set up of the initial conditions for the fit
- **All usual HEP software components exist (rewrite of Minuit, some CERNLIB etc. are available)**
- **Plotting part of JAS is less advanced than that of ROOT**
  - **this is one of the reasons why JAS is not as popular as ROOT**
- **In many cases, you need a lot of typing to make a simple plot due to long names of “factories” (i.e. “producers” of objects)**

Typical JAS example:

```
af = IAnalysisFactory.create();
hf = af.createHistogramFactory(af.createTreeFactory().create());
h1 = hf.createHistogram1D("test 1d",50,-3,6);
plotter = af.createPlotterFactory().create("Plot");
plotter.show();
```

can be shortened by a factor 3 using shorter names!

JAS

- I've started making some improvements for JAS, but then realized that one could do something better and efficient
- Aim:
  - a high-level object-oriented, dynamically-typed analysis framework with numerous high-level constructs on top of FreeHep library
  - access to all methods of requested objects (without looking at a manual)
  - programs must be shorter than in JAS or ROOT
  - program length should be close to PAW macros for most common tasks
  - programming language for macros should be as popular as possible.
  - macros should be compiled without modifications (unlike ROOT “Cint”, macros or Fortran COMIS)
  - graphics should be as good as in ROOT
  - plotted components should be interactive

jHepWork

- **Main programing language is Jython**
  - object-oriented, multiplatform
  - similar to other high-level languages (MatLab, Maple, S-plus)
  - shorter programs than in C++/JAVA (factor of 2)
  - higher-level constructions than in C++/Java
  - dynamically typed. Dynamic object completion & help system
  - extensive build-in run-time checks
  - easy integration with C++/Java libraries (for CPU extensive tasks)
  - easy integration with GUI & WWW
- **Faster programming, less bugs**
  - concentrate on physics, not on low-level programming!
  - very high-level constructions for data manipulation which have no analogy in ROOT or JAS
- **BeanShell and plain JAVA can also be used**

Advantages compared to JAS

- More powerful graphics than in JAS
  - output plots are very close “publishable” standard.
  - statements are more similar to ROOT. 3D graphics
- More interactive plots than in JAS. More choices for labels (support for overline, Greek and math symbols), legends, titles etc.
- High-level operations on data for common HEP tasks
- Programs are shorter than in JAS.
- Powerful editor (jEdit-like) with a browser for object methods and a structure browser + LaTeX support (LaTeX & BibTex tools etc..)

Advantages compared to ROOT

- Multiplatform. No installation. No compilation. Access to JAVA/Jython
- Java reflection technology looks inside a Java object at runtime
- Jython macros can be compiled using jythonc (unlike Cint macros)
- VERY intelligent Java IDE focused on productivity (Eclipse, NetBeans etc.)
- Can be embedded to the WEB for distributed analysis environment
- Significantly shorter programs compared to ROOT/Cint
- High-level operations for common HEP

Jython is slow?

- Do not care, especially for front-end analysis environment, interactive experimentation, debugging, rapid program development (i.e. exactly what we do for final analysis and visualization)

- All depends on how do you write programs. If you use high-level data structures from Jython, there would be much higher chance that your programs will be faster than those implemented in C/C++

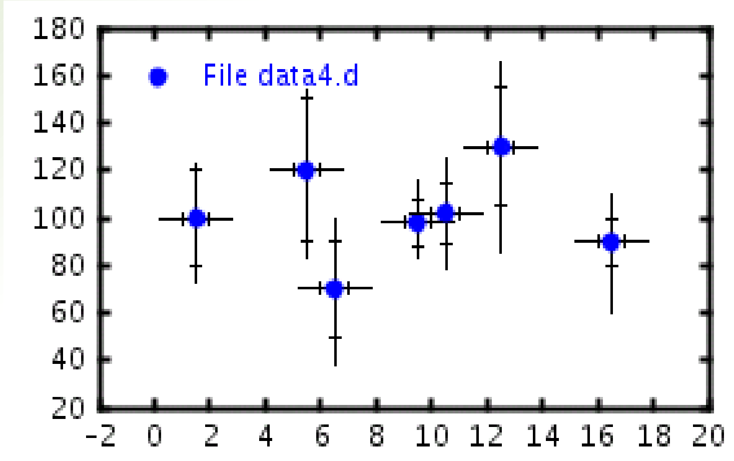
- CPU extensive tasks can be moved to compiled jar libraries, which can dynamically by linked

Example: High-level constructs

- P1D: a container class for data points which includes 2-level errors (usually statistical and systematical errors). In most general case, each data point is represented by 10 numbers

To read and to display P1D object, you need only a few statements:

```
>>> c1=HPlot("Main canvas")
>>> p=P1D( "File data4.d","data4.d")
>>> c1.show(p)     # plot it
>>> p.toTable()     # show in a table
>>> p.toFile("file") # write back to a file
```



jHepWork has many methods to manipulate with P1D data holder. Each method takes into account 1st and 2nd level errors

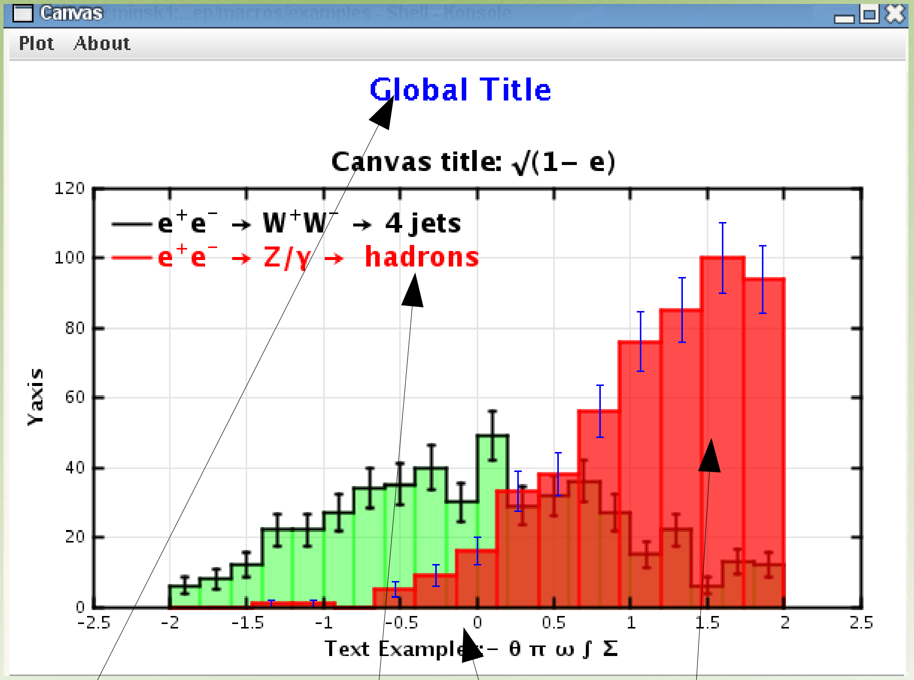
Some P1D methods

```
>>> p1=P1D("show points", "data.d")
>>> p1.size()            # size of the data
>>> p1.clear()           # clear the P1D
>>> p1.getMin(int)       # min value for axis=0,1,..
>>> p1.getMax(int)       # max value for axis=0,1,..
>>> p1.mean()            # mean value
>>> p3=p1.merge(p2)                          # merge 2 containers into p3
>>> p3=p1.oper(p2,"New Title","+")       # add  p1 and p2
>>> p3=p1.oper(p2,"New Title","-")       # subtract p2 from p1
>>> p3=p1.oper(p2,"New Title","*")       # multiply p1 by p2
>>> p3=p1.oper(p2,"New Title","/")       # divide   p1 by p2
>>> p3=p1.oper("New Title", scaleFactor )
>>> p1=p1.move("function", "axis" ) # move to ”exp”, ”log”, ”sqrt”, ”cos”, ”sin”
>>> p3=p1.oper(p2,"added with 50% corr.","+","Y",corr) # operations assuming correlations
between p1 and p2
>>> p3.toTable()          # move to a pop-up table
>>> p3.Spsheet()          # move to a spreadsheet
>>> p3.toFile(“name.d”) # write to a file
>>> c1.show(p3)           # show again in a canvas (including all errors)
```

Each operation propagates 1st and 2nd level errors

Example-I: histograms

Example (ignore legend meaning!)



interactive global title

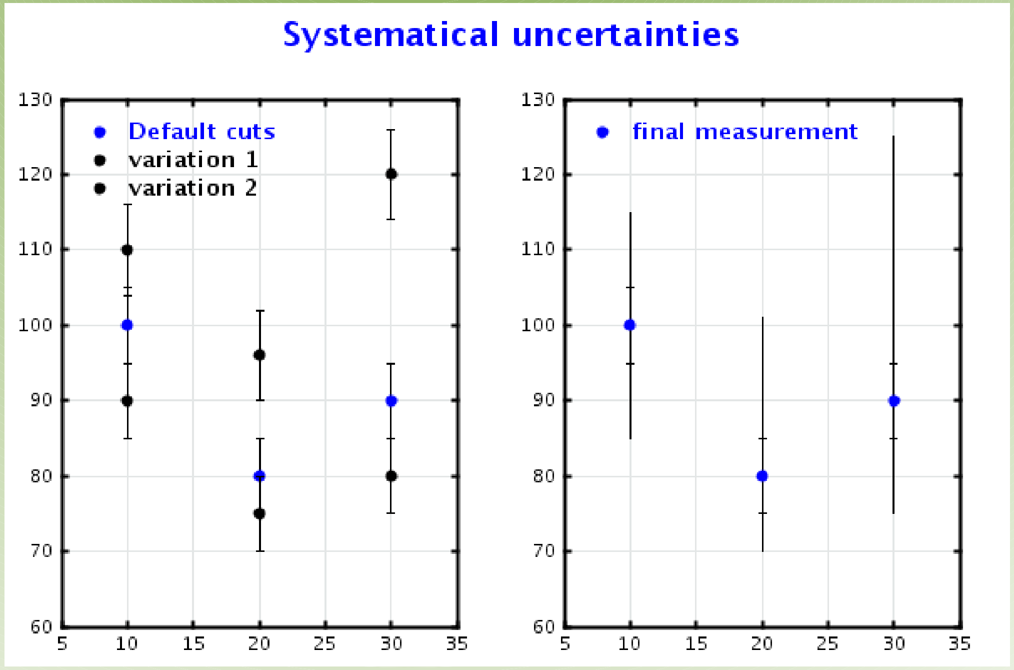
left-click on mouse change axis, styles

interactive labels (with overlines, Greeks etc)

zoomable axis (middle mouse button)

```
c1 = HPlot("Canvas",600,400,0.1)
c1.gTitle("Global Title", Color.blue)
c1.visible(1)
c1.setAutoRange()
h1 = H1D("e^{+}e^{-} &rarr; W^+{}W^{-}
&rarr; 4 jets",20, -2.0, 2.0)
rand = Random()

for i in range(500):
     h1.fill(rand.nextGaussian())

h1.setFill(1)
h1.fillColor(Color.green)
h1.errX(0)
h1.errY(1)
h1.setPenWidthErr(2)
c1.draw(h1)
```

Example-II: do systematics



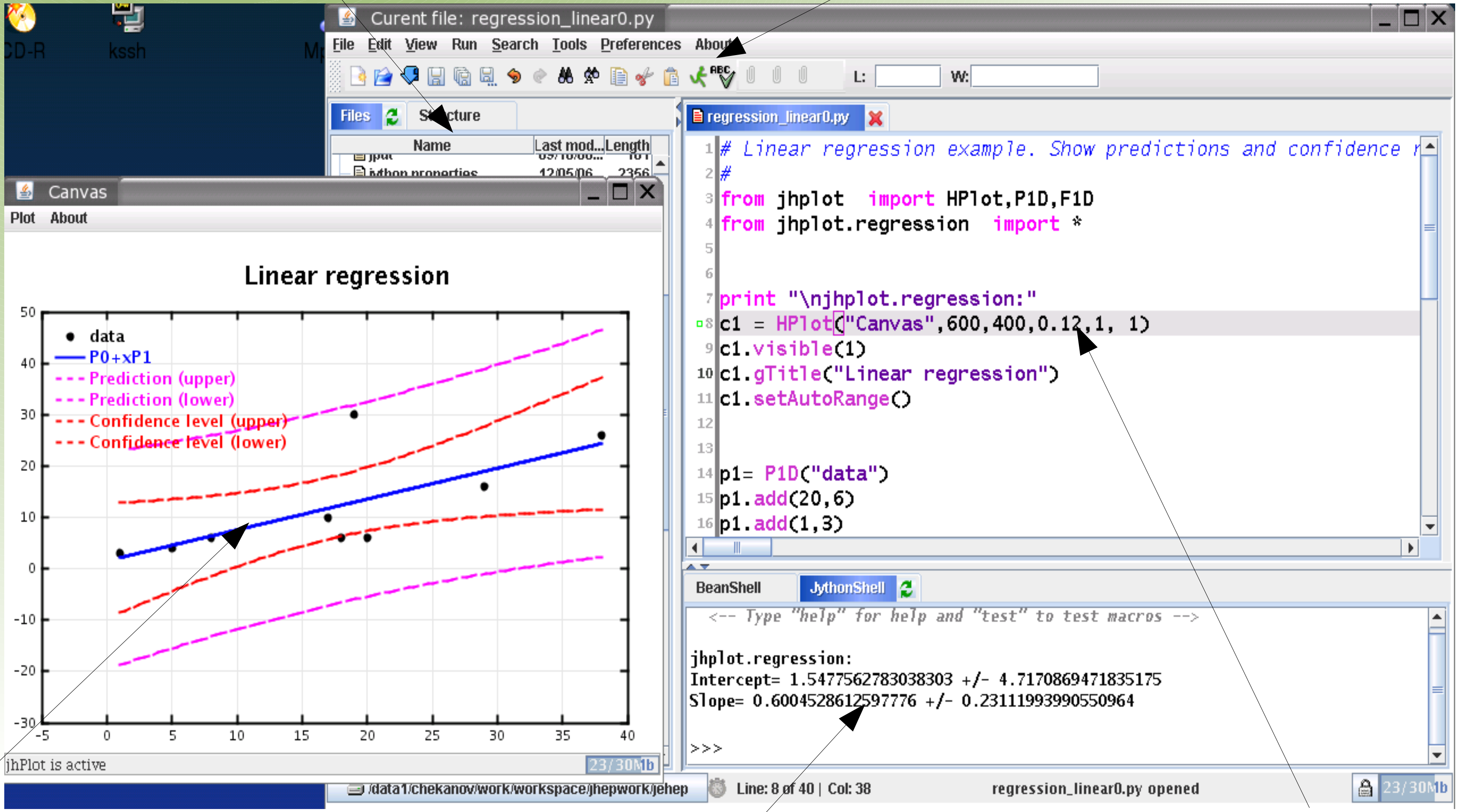
```
p1= P1D("Default cuts")

pp=[]  # build list of objects
p2= P1D("variation 1")
pp.append(p2)
p3= P1D("variation 2")
pp.append(p3)

c1.draw(p1)
c1.draw(p2)
c1.draw(p3)

psys=p1.getSys(pp)
psys.setTitle("final measurement")

c1.cd(1,2)
c1.setAutoRange()
psys.showErrors(1)
c1.draw(psys)
```

evaluate systematics by adding all variations in quadrature and returns a new P1D object and display it

jHepWork in action

file browser and structure browser

or F8 to run a script



```
# Linear regression example. Show predictions and confidence r
#
from jhplot  import HPlot,P1D,F1D
from jhplot.regression  import *


print "\njhplot.regression:"
c1 = HPlot("Canvas",600,400,0.12,1, 1)
c1.visible(1)
c1.gTitle("Linear regression")
c1.setAutoRange()


p1= P1D("data")
p1.add(20,6)
p1.add(1,3)
```

```
  <-- Type "help" for help and "test" to test macros -->

jhplot.regression:
Intercept= 1.5477562783038303 +/- 4.7170869471835175
Slope= 0.6004528612597776 +/- 0.23111993990550964


>>>
```

main canvas with interactive legends, titles, labels, zoomble axis etc.

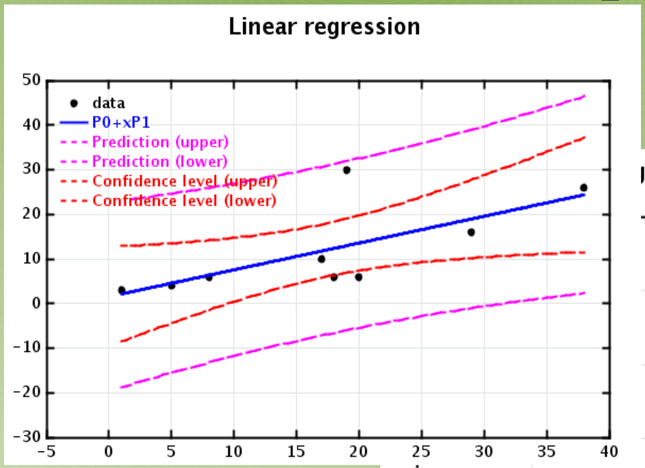
jython shell based on JyConsole with object completion

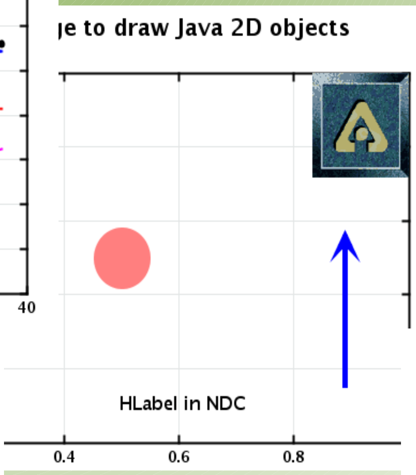
F4 to view all methods associated with this object (while you are typing!)

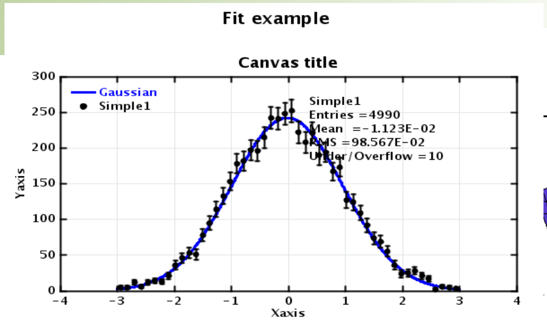
jHepWork can:

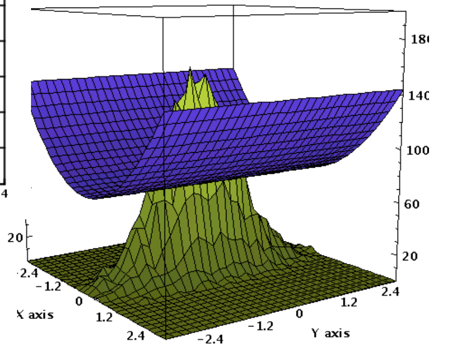
- Display 1D and 2D functions
- Display 1D and 2D histograms
- Operations with histograms (via interface with JAIDA)
- Display data with 2-level errors using many options
- Interface with JAIDA histograms
- Can read ROOT histograms / trees (via interface to JAIDA)
- Perform math operations with high-level data containers
- Display interactive graphs
- Show interactive labels, legends, titles. Zoomable plot regions
- Plot graphical 2D primitives
- Fit histograms (via interface to JAIDA)
- Cluster analysis (k-means , fuzzy etc..)
- Neural network
- Linear regression analysis

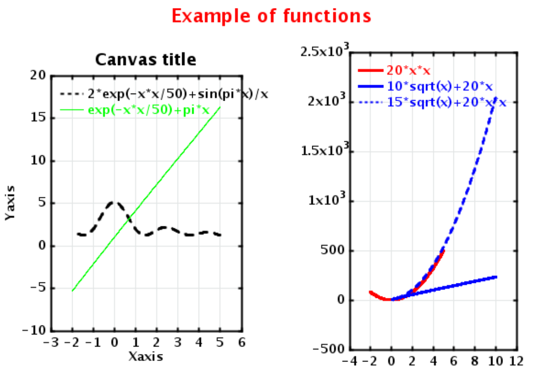
Random examples:

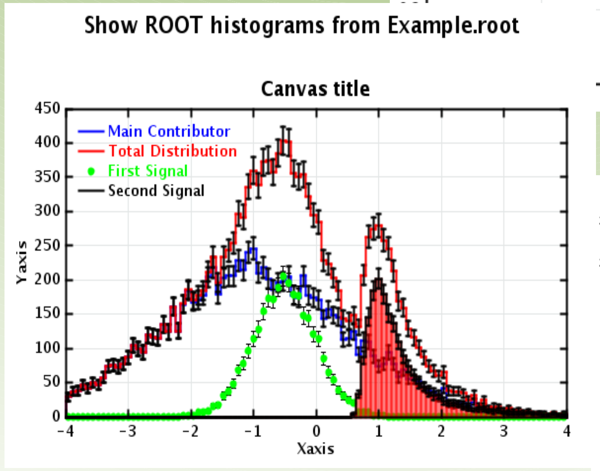


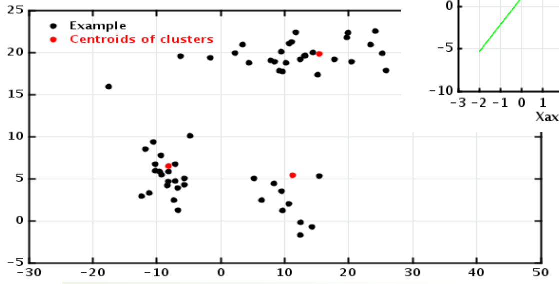


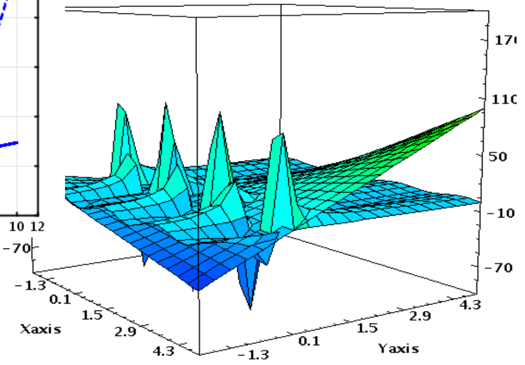


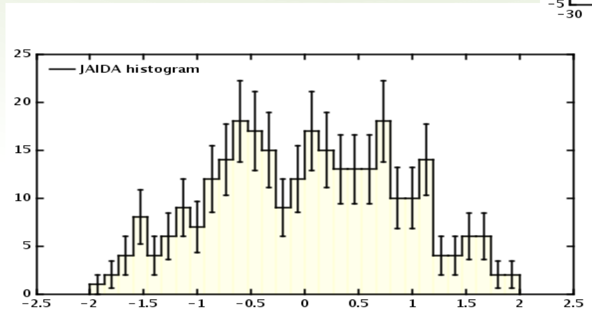


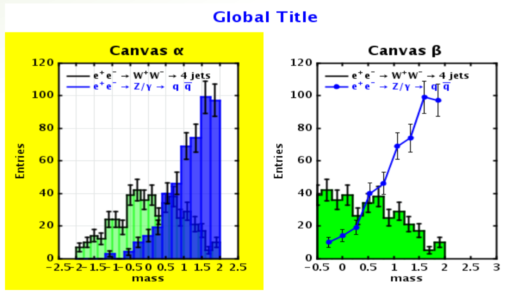


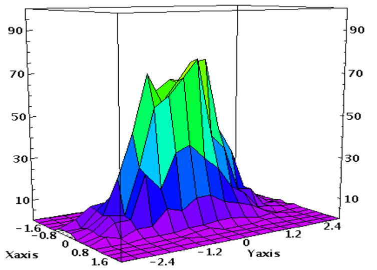


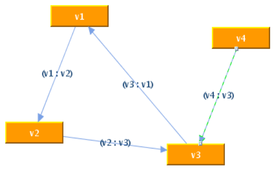












Status and summary

- **jHepWork version 1.1:**
  - **http://projects.hepforge.org/jhepwork/**
  - **Online manual with description of ~200 methods**
  - **Online API for the core HPlot package**
  - **~40 examples (Jython source files + generated figures)**
  - **Installation: None.**
    - **unpack jhepwork-1.1.run and run jhepwork.sh or jhepwork.bat (windows)**

Contributions:
Core packages (JEHep, JHPlot, jMinHEP) were developed by myself.

+ many packages have been refactored and modified (GNU license):
jEdit TextArea (S.Pestov), jMySpell project (DreamTagnerine), JabRef project, Jext project, JyConsole by Artenum, parts of jpEdit editor,jPlot, by Jan van der Lee, Surface Plotter packages, by Yanto Suryono,VLJTable from Vlsolution etc. Includes FreeHEP, jgraph, jgrapht, jFreeChart, Jsci libraries, Flanagan's Math Library,Jakarta Common Math library

+ JAIDA from FreeHEP