



jHepWork

**S.Chekanov
(ANL)**

**DESY computing seminar
January 21, 2008**

HEP choices for data analysis



- **PAW/CERNLIB: FORTRAN / C:**
 - *Almost dead. No support. Many senior physicists like it*
 - *Not object-oriented*
 - *Cannot be easily ported to new Linux distributions (with gcc4.2)*
- **OpenScientist (<http://openscientist.lal.in2p3.fr/>): C/C++**
 - *Smooth transition from legacy HEP tools*
 - *Support for paw (opaw). Support for AIDA. OpenGL and Inventor for visualization*
- **ROOT: C++**
 - *Main choice for LHC (but everyone complains: it too complicated!)*
 - *~50% recent physics papers at HERA done using ROOT*
- **JAS based on JAIDA: JAVA**
 - *Works on every platform without compilation.*
 - *Plenty of HEP libraries (read ROOT files, jminuit etc)*
 - *R&Ds for ILC in USA based on JAS and FreeHEP*
 - *JAS graphics is still behind ROOT (although JAVA has superior graphics!)*
- **Do we need something else for data analysis? If we do need, this has to be with good graphics. Can this be JAVA?**

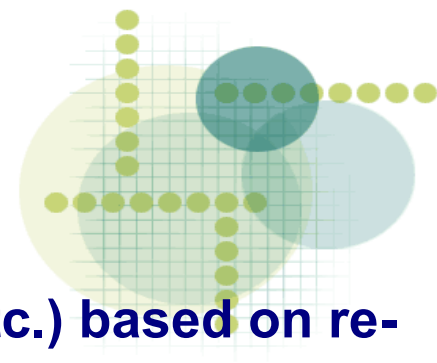
See : Tony Johnson:
The Role of Java in High Energy Physics
SLAC Colloquium - March 8th 1999

JAVA vs C++: I



- **Java is multiplatform: “Write once, run anywhere”**
 - *Once written, works everywhere where JAVA is installed*
 - *Unlike ROOT/C++. Should be compiled for every platform (I'm compiling ROOT several times every year for each version of FC and SUSE)*
- **Most popular programming language:**
 - *Freshmeat.net: 5800 JAVA vs 4800 C++ projects*
 - *“popularity” was one of the argument for moving from FORTRAN to C++!*
- **Better structured, clean, efficient:**
 - *Every variable, constant, and function must be inside some class*
 - *Everything in “packages”*
 - *Easier to understand a code*
- **Simpler than C++ (no pointers!) Advantageous for physicists?**
- **Java does not have operator overloading!**
 - *Do we really need it?*
 - *Some “overloading” done via reprocessing (example: "a" + "b", a, b are strings)*
 - *If we really need it, use Jython (see later)*

JAVA vs C++: II



- **Java reflection technology. Expected in C++ in >5 years.**
- **Powerful, intelligent and often free IDE (Eclipse, NetBean etc.) based on reflection**
 - *Tells what method to use, proposes fixes, refactoring etc.*
 - *Absolutely essential for complicated software projects*
 - *Or we still like “VI” or Emacs?*
- **Automatic garbage collection**
 - *a programmer does not need to perform memory management*
- **Extensive compile-time and run-time checking**
- **Can be embedded to the WEB:**
 - *important for distributed analysis environment (i.e. HEP)*
 - *plugins, applets*
- **SUN released JAVA under an open source initiative**

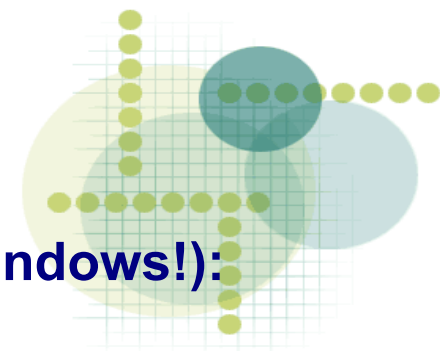
JAVA vs C++: III



- **Java is VERY stable and robust**
 - *I have still running java programs written 10 years ago, without need for recompilation*
 - *Very good chance that once your JAVA (jar) library is created, it will work in 10, 20.. years from now without need for any support. I suspect that it can even be recompiled from sources..*
- **C/C++?**
 - **Often no backward compatibility**
 - **Nowadays HEP uses specific (and usually old) gcc versions**
 - *HEP community has to develop “Scientific Linux” to support HEP libraries*
 - *SL is often behind mainstream Linux distros*
 - *need specific hardware, SL4 often cannot be installed on modern PC's (unlike Suse 10.3, Ubuntu 7.10)*
 - *security issues have to be fixed*
 - **HEP community needs a significant man power to support and develop the entire computing chain: “Hardware – OS – HEP specific software”**
 - *LHC data using SL4 and gcc3, while all modern distros have been moved to gcc4 long time ago*
- **For Java, we do not care which PC & operating system is used!**

LHC computing

(useful tips for newcomers)



- **4 different environments (for Linux, of course, no Mac or Windows!):**
 - *SL3 or SL4 (gcc323, gcc34) + 64 bit versions*
 - *Many versions of libraries. Some hardware supported only by SL4*
 - *64 libraries are not recommended (at least for ATLAS)*
 - *some packages functional only for gcc322*
 - *some packages cannot be compiled for gcc322. But grid needs them!*
 - *some shared libraries are not compatible with other.*
 - *At the end, even “vim7.1” does not exist (with bracket matching)*
 - *Soon SL5? Redo all libraries (and rewrite some packages)!*
 - *For any user coming from Ubuntu7.10, suse 10.3 (with gcc4.2) , all looks old-fashion and archaic!*
- **A simple test: Go to Sourceforge and pick up any jar file (or java source) created 7-8 years ago and run or compile on a modern JVM (6) and modern PC on 32 or 64 bit platform. I guarantee: it will work!**
- **Do not even try to do this with a C/C++ code written 4-5 years ago!**

Java for ILC?



- It is going to be a long lasting project, and we should be sure that software created now can run in 10-15 years from now without significant maintenance - **Java is an ideal for this**
- Many Java packages for LC detector studies:
 - *Offline Simulation and Reconstruction*
 - *hep.lcd package for Linear Collider Detector studies*
 - *Event display (Wired)*
 - *Extensive java libraries from FreeHEP (Jaida)*
 - *Finally, JAS (Java analysis studio and many other tools)*

Note: CMS and ATLAS event displays are written in JAVA as well and include FreeHEP libraries

JAS



- **JAS – JAVA analysis framework**
- **All usual HEP software components exist:**
 - *histograms, data containers,*
 - *Wired plugin to integrated with event display*
 - *fits, rewrite of Minuit,*
 - *some CERNLIB are available*
- **Drawbacks:**
 - *Plotting part of JAS is less advanced than that of ROOT*
 - *In many cases, you need a lot of typing to make a simple plot*

Typical 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();
```

From JAS to jHepWork



- I've started making some improvements for JAS, but then realized I had to completely rewrite the code (but still using FreeHEP libraries)
- Main aim:
 - *To build a high-level object-oriented, dynamically-typed analysis framework with numerous high-level constructs on top of FreeHep libraries*
 - *No installation (just unpack and run on any platform)*
 - *No external dependences*
 - *Programs must be shorter than in JAS or ROOT (program length should be close to PAW macros for most common tasks). Programs should be as short as in PAW*
 - *Graphics should be as good as in ROOT*
 - *Plotted components should be fully interactive*
 - *Build-in help based on reflection*
 - *Auto-update*

jHepWork



- a full-featured multiplatform data-analysis framework written in JAVA
- Based on FreeHEP java libraries (JAIDA)
- Seamlessly integrated with Java-based Linear Collider Detector (LCD) software concept
- Powerful Python/Jython friendly IDE with a code assist based on reflection. Also BeanShell and plain Java is supported
 - Integrated compiler
 - editor with syntax highlighting, bracket matching
 - a code assist based on reflection
- No installation! Works on any platform. Self update feature.
- Self-contained, does not require external packages
- Many java libraries and graphical packages for data analysis included
- Many graphics packages for displaying and interacting with data
- Many math packages for histograming and math operations

jHepWork and Jython



- **Main programming language is Jython**
 - object-oriented, multiplatform data analysis environment
 - 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
- VERY intelligent free Java IDE focused on productivity (Eclipse, NetBeans etc.)
- Can be embedded to the WEB: Good for distributed analysis environment
- Significantly shorter programs compared to ROOT/Cint
- High-level operations for common HEP
- Auto-update

Jython is slow?



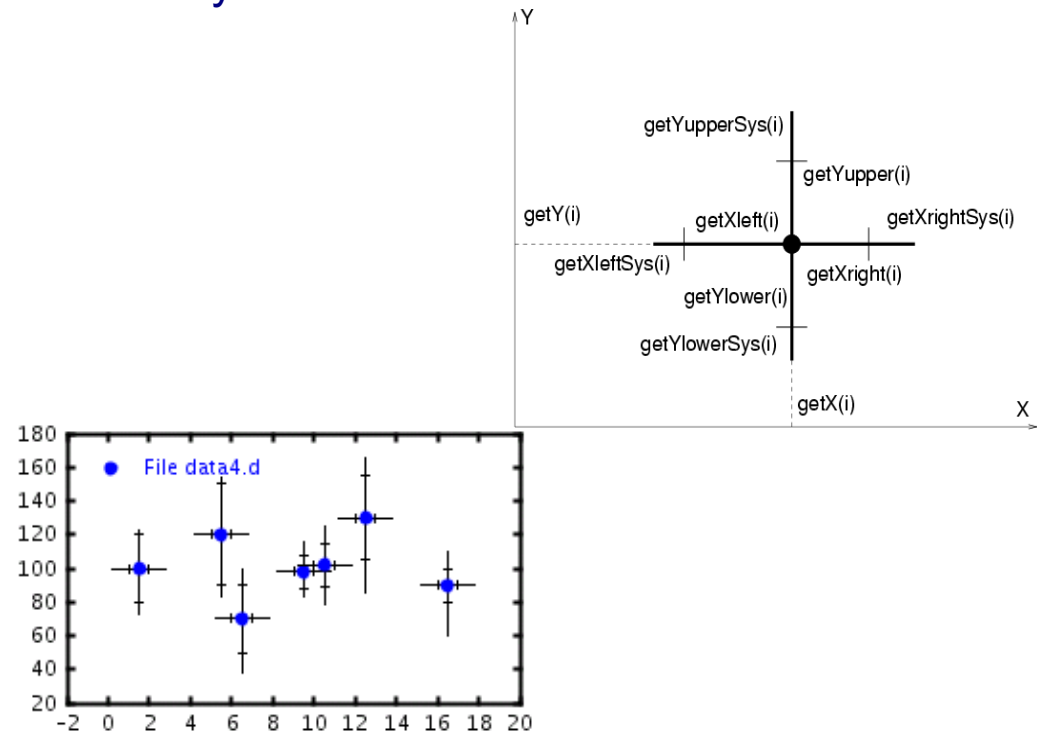
- Do not care, especially for interactive experimentation, debugging, rapid program development (i.e. exactly what we do for final analysis and visualization)
- All depends on how you write programs. If you use high-level data structures from Jython, there is much higher chance that your program will be faster than in C/C++
- For common operations on primitive data types (loops etc), Jython is typically slower than CINT (factor ~2-3) and factor 5-10 than equivalent java programs
 - *Difference with CINT is mainly due to longer start-up time*
- CPU extensive tasks can be moved to jar byte-code libraries, which can dynamically be linked. jHepWork allows automatically build and link such libraries with a Jython code.
 - Programs based on java jar libraries faster than those in Jython by a factor 5-10
 - similar to compiled C++ code
 - Extensive jHepWork libraries have to be used as much as possible.

Example: high-level constructs

- **P1D: a container class for data points (i.e. List):**
 - Extends DrawOptions class
 - include 2-level errors (usually statistical and systematic errors). In most general case, each data point is represented by 10 numbers

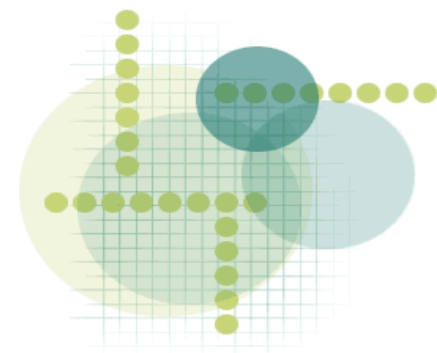
Simple data manipulation:

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



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

Some methods

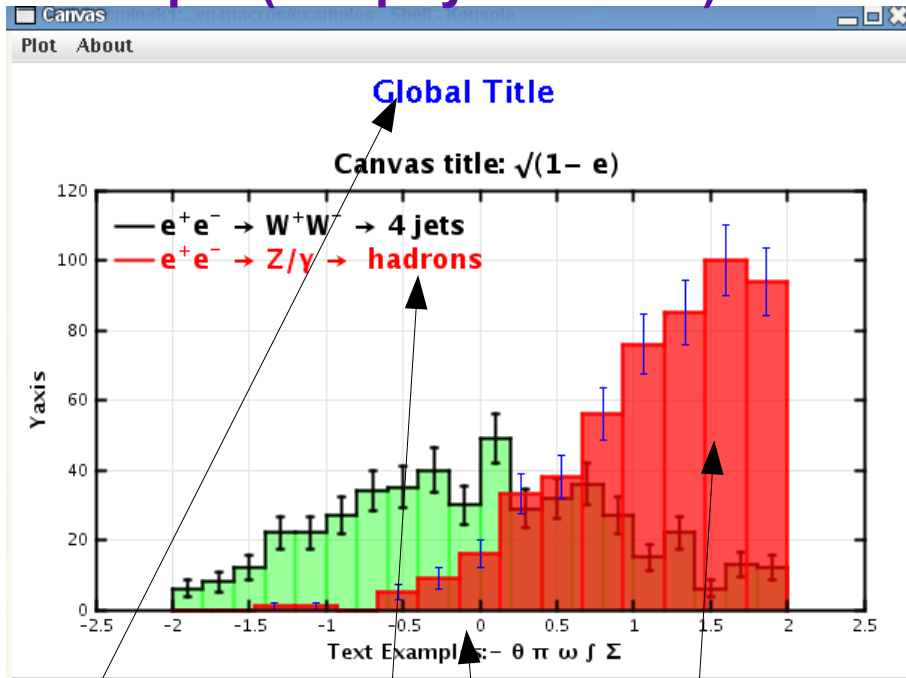


```
>>> p1=P1D("show points", "data.d")
>>> p1.size()          # size of the data
>>> p1.clear()         # clear the P1D
>>> p1.getMin(int)    # min value for axis=0,1,..
>>> p1.getMax(int)    # max value for axis=0,1,..
>>> p1.mean()         # mean value
>>> p3=p1.merge(p2)    # merge 2 containers into 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 includes 1st and 2nd level errors

Simple example-I: plot histograms

Example (not physics one!)



interactive global title

interactive labels

zoomable axis
(middle mouse button)

left-click on mouse
change axis, styles

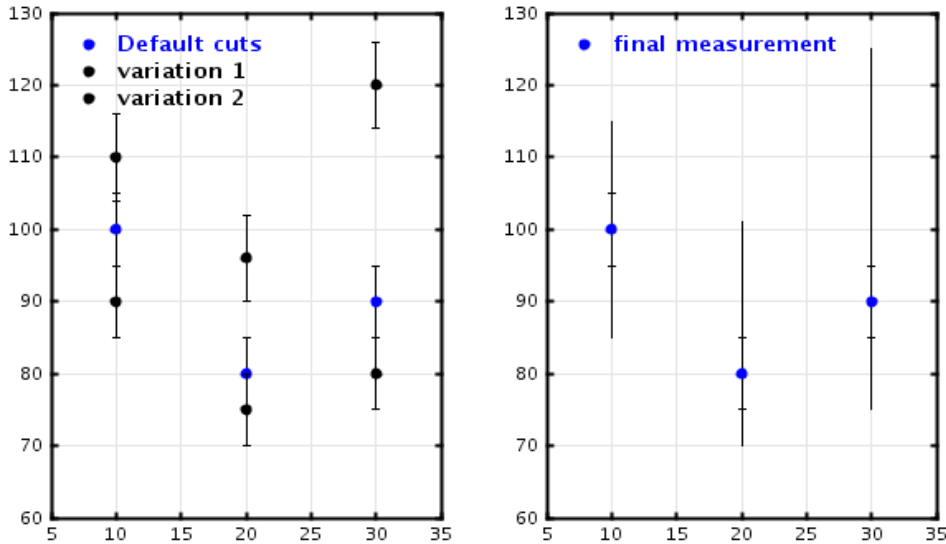
```
c1 = HPlot("Canvas",600,400,0.1)
c1.gTitle("Global Title", Color.blue)
c1.visible(1)
c1.setAutoRange()
h1 = H1D("e^{+}e^{-} &rarr; W^{+}W^{-}
&rarr; 4 jets",20, -2.0, 2.0)
rand = Random()
```

```
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)
```

Simple example-II: do systematics

Systematical uncertainties



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

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

```
pp=[]  
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)
```

Supported symbols for legends and labels and all Text Objects



jHPlot symbols I

jHPlot symbols II

used by all strings. Add '&' in front and terminate by ';'

used by all strings. Add '&' in front and terminate by ';'

d ₃ subs	± plusmn	Á Aring	Ù Ugrave	í iacute	A Alpha	Φ Phi
d ³ super	² sup2	Æ AElig	Ú Uacute	í icirc	B Beta	Χ Chi
đ s̄ ā	³ sup3	Ç Ccedil	Û Ucirc	ï iuml	Γ Gamma	Ψ Psi
nbsp	´ acute	È Egrave	Ü Uuml	ð eth	Δ Delta	Ω Omega
¡ iexcl	μ micro	É Eacute	Ý Yacute	ñ ntilde	E Epsilon	α alpha
¢ cent	¶ para	Ê Ecirc	Þ THORN	ò ograve	Z Zeta	β beta
£ pound	· middot	Ë Euml	ß szlig	ó oacute	H Eta	γ gamma
¤ curren	¸ cedil	Ì Igrave	à agrave	ô ocirc	Θ Theta	δ delta
¥ yen	¹ sup1	Í Iacute	á aacute	õ otilde	I Iota	ε epsilon
¦ brvbar	º ordm	Î Icirc	â acirc	ö ouml	K Kappa	ζ zeta
§ sect	» raquo	Ï Iuml	ã atilde	÷ divide	Λ Lambda	η eta
¨ uml	¼ frac14	Ð ETH	ä auml	ø oslash	M Mu	θ theta
© copy	½ frac12	Ñ Ntilde	å aring	ù ugrave	N Nu	ι iota
ª ordf	¾ frac34	Ò Ograve	æ aelig	ú uacute	Ξ Xi	κ kappa
« laquo	¿ iquest	Ó Oacute	ç ccedil	û ucirc	Ο Omicron	λ lambda
¬ not	À Agrave	Ô Ocirc	è egrave	ü uuml	Π Pi	μ mu
– shy	Á Aacute	Õ Otilde	é eacute	ý yacute	Ρ Rho	ν nu
® reg	Â Acirc	Ö Ouml	ê ecirc	þ thorn	Σ Sigma	ξ xi
¬ macr	Ã Atilde	× times	ë euml	ÿ yuml	T Tau	ο omicron
° deg	Ä Auml	Ø Oslash	ì igrave	ƒ fnof	Υ Upsilon	π pi

τ tau	← larr	Σ sum	□ sub	” quot
υ upsilon	↑ uarr	– minus	□ sup	& amp
φ phi	→ rarr	□ lowast	□ nsub	< lt
χ chi	↓ darr	√ radic	□ sube	> gt
ψ psi	↔ harr	□ prop	□ supe	Œ OElig
ω omega	□ crarr	∞ infin	□ oplus	œ oelig
ϑ thetasym	□ lArr	□ ang	□ otimes	Š Scaron
Υ upsilh	□ uArr	□ and	□ perp	š scaron
ϖ piv	□ rArr	□ or	□ sdot	Ÿ Yuml
• bull	□ dArr	∩ cap	□ lceil	ˆ circ
… hellip	□ hArr	□ cup	□ rceil	˘ tilde
´ prime	□ forall	∫ int	□ lfloor	ensp
“ Prime	∂ part	□ there4	□ rfloor	emsp
– oline	□ exist	□ sim	□ lang	thinsp
/fracl	□ empty	□ cong	□ rang	zwnj
⊖ weierp	□ nabla	≈ asymp	◊ loz	zwj
♠ image	□ isin	≠ ne	♣ spades	lrm
℔ real	□ notin	≡ equiv	♣ clubs	rlm
™ trade	□ ni	≤ le	♥ hearts	- ndash
κ alefsym	Π prod	≥ ge	♦ diams	– mdash

jHepWork in action



file browser and structure browser

or F8 to run a script

The screenshot shows the jHepWork interface. At the top, a file browser and structure browser are visible. The main canvas displays a plot titled "Linear regression" with data points, a blue fit line, and confidence intervals. The jython shell at the bottom shows the execution of a script that performs linear regression on the data.

```
1 # Linear regression example. Show predictions and confidence r
2 #
3 from jhplot import HPlot,P1D,F1D
4 from jhplot.regression import *
5
6
7 print "\njhplot.regression:"
8 c1 = HPlot("Canvas",600,400,0.12,1, 1)
9 c1.visible(1)
10 c1.gTitle("Linear regression")
11 c1.setAutoRange()
12
13
14 p1= P1D("data")
15 p1.add(20,6)
16 p1.add(1,3)
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

BeanShell JythonShell

```
<-- 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, zoomable axis etc.

jython shell based on JyConsole with object completion

F4 to view all methods associated with this object

jHepWork v1.6 can:



- **Display 1D and 2D functions**
- **Display 1D and 2D histograms**
- **Operations with histograms (via interface to JAIDA)**
- **Display data with 2-level errors using many options**
- **Interface with JAIDA histograms**
- **Can read ROOT histograms / trees (via interface to JAIDA)**
- **Operations with high-level data containers**
- **Display interactive graphs. Draw Feynman diagrams**
- **Show interactive labels, legends, titles. Zooming plot regions**
- **Plot graphical 2D primitives**
- **Fit histograms (via interface to JAIDA)**
- **Cluster analysis (k-means , fuzzy etc..)**
- **Overload for most common operators for Jython histograms**
- **Neural network. Linear regression analysis**

jHepWork data structure



- Unlike plain JAVA (i.e. vector, list, map), extend “DrawOptions” class
- Histograms:
 - **H1D** - one-dimensional histogram
 - **H2D** - two-dimensional histogram
 - Use JAIDA for other histogram types
- Data holders:
 - **P0D** – vector type
 - **P1D** – holds data in (x,y) with 1st and 2nd level errors
 - **P2D** – holds data in (x,y,z)
 - **PND** – holds data in N dimensions
- Functions
 - **F1D** – one-dimensional function
 - **F2D** – two-dimensional function
- + all FreeHEP/JAIDA containers
- + all JAVA data containers (arrays, list, map etc)
- + all Python/Jython containers (map, list, dictionary, tuple etc)

jHepWork graphical canvas

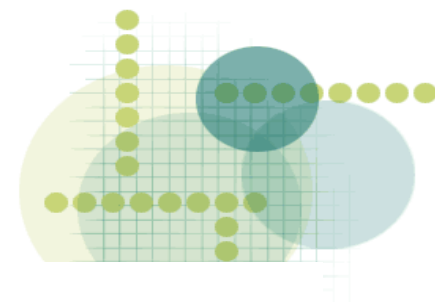


- All methods are rather similar for all graphical canvas
 - For example: data can be shown using “`canvas.draw(object)`”
- **HPlot** – canvas to show plots with several pads
 - **SHPlot** - a singleton, i.e. only one instance is allowed
- **SPlot** – similar to HPlot but simpler and less memory consuming (good for applets)
- **JaPlot** – similar to HPlot, but allows manipulations with pads and drawing Feynman diagrams (**SJaPlot** – singleton). Based on **JaxoDraw**
- **HPlot3D** – to draw histograms and data in 3D
- **HGraph** - to draw interactive graphs (interconnected boxes etc.)
- **HChart** - for charts (based on **jFreeChart** package)
- **HView3D** – interactive 3D objects
- **HTable** – sortable table
- **SPsheet** - a spreadsheet class to display the data
- **IEditor** – image editor and viewer (based on **ImageJ**)

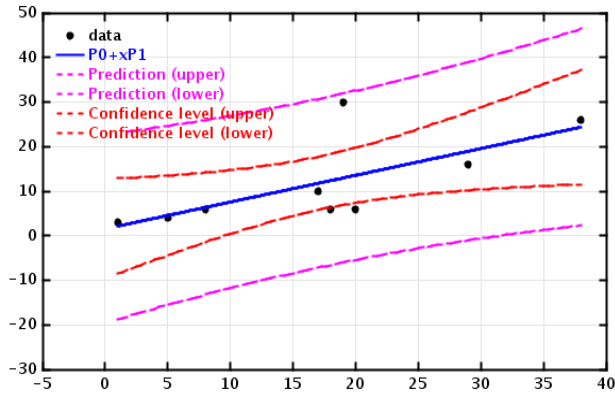
Each class usually has different origin. All are GNU licensed

Usually significantly modified for better integration

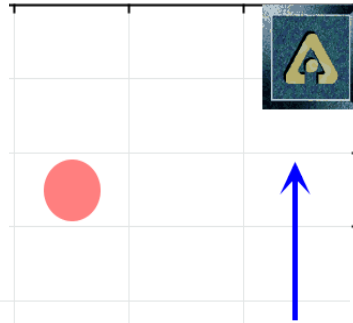
Random examples:



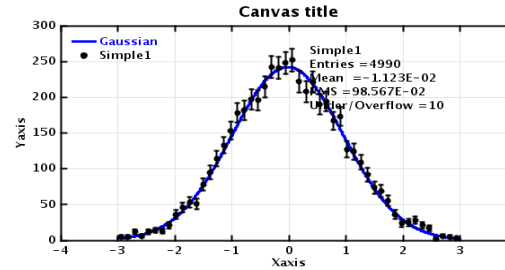
Linear regression



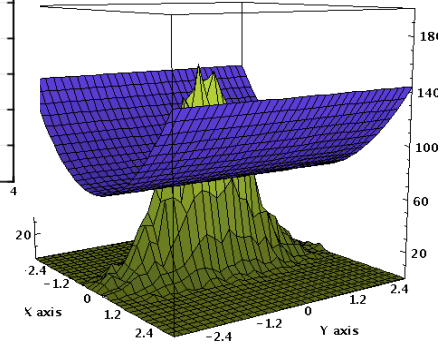
je to draw Java 2D objects



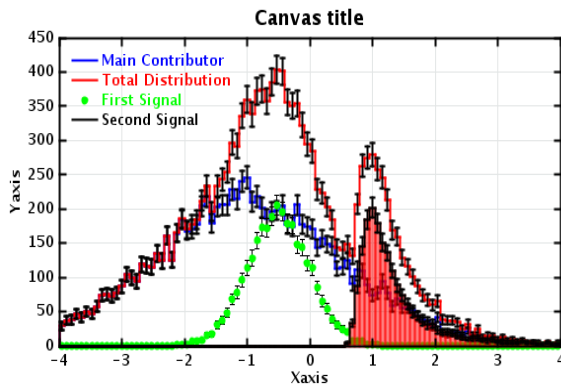
Fit example



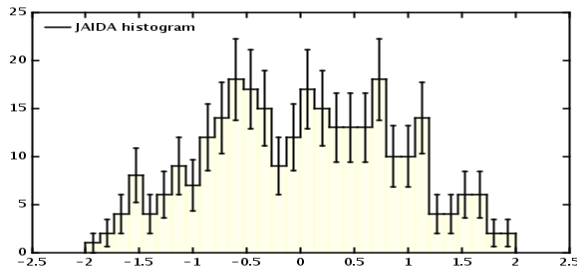
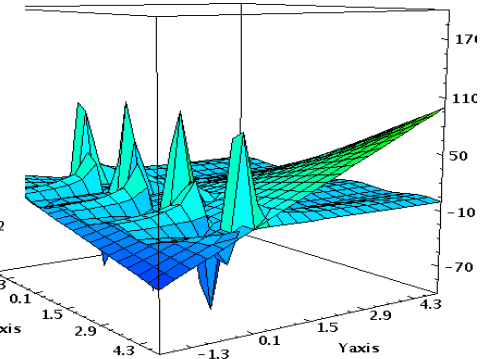
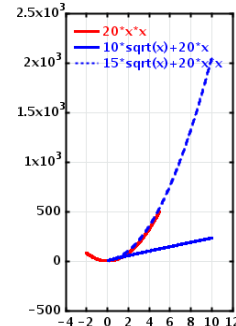
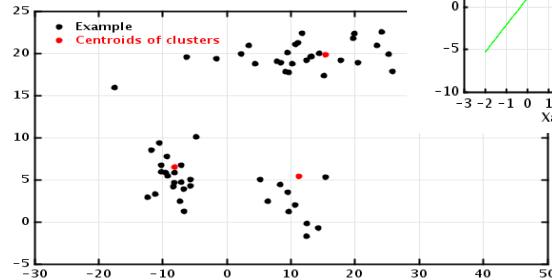
Example of functions



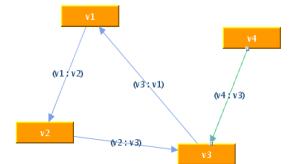
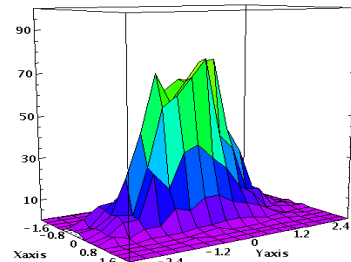
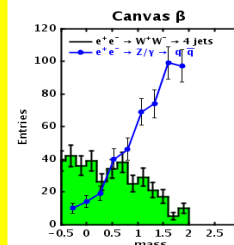
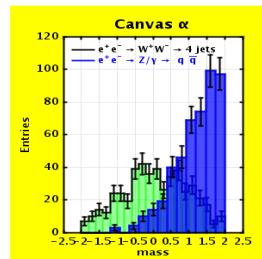
Show ROOT histograms from Example.root



HLabel in NDC



Global Title



jHepWork in action



- **jHepWork scripting (divx4 avi file)**

http://projects.hepforge.org/jhepwork/doc/tutorial_movie/promt.avi

jHepWork in action



- **jHepWork script examples**

http://projects.hepforge.org/jhepwork/doc/tutorial_movie/graphics_examples.avi

jHepWork in action



- **Opening ROOT files (histograms)**
Based on JaPlot and JaxoDraw

http://projects.hepforge.org/jhepwork/doc/tutorial_movie/root_file.avi

jHepWork in action



- **Opening AIDA files (data points)**

http://projects.hepforge.org/jhepwork/doc/tutorial_movie/aida.avi

jHepWork in action



- **SHPlot (Jython package)**

http://projects.hepforge.org/jhepwork/doc/tutorial_movie/shplot.avi

jHepWork in action



- **Building Java jar libraries**

http://projects.hepforge.org/jhepwork/doc/tutorial_movie/java_library.avi

jHepWork in action



- **Doing analysis..**

http://projects.hepforge.org/jhepwork/doc/tutorial_movie/physics.avi

Status and summary



- **jHepWork version 1.6:**
 - <http://projects.hepforge.org/jhepwork/>
 - Short manual (70 pages) or long manual + classes (700 pages)
 - Online API for HPlot package
 - ~50 examples (Jython source files + generated figures)

Contributions:

The main JHPlot, jMinHEP packages were developed by myself.

Many packages have been re-factored and modified:

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 VIsolution etc. Includes FreeHEP, JaxoDraw, jgraph, jgraphT, jFreeChart, Jsci libraries...