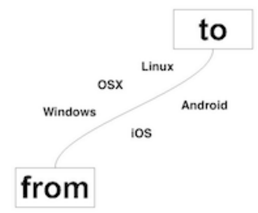
GOPAW

G.Barrand, CNRS/IN2P3/LAL



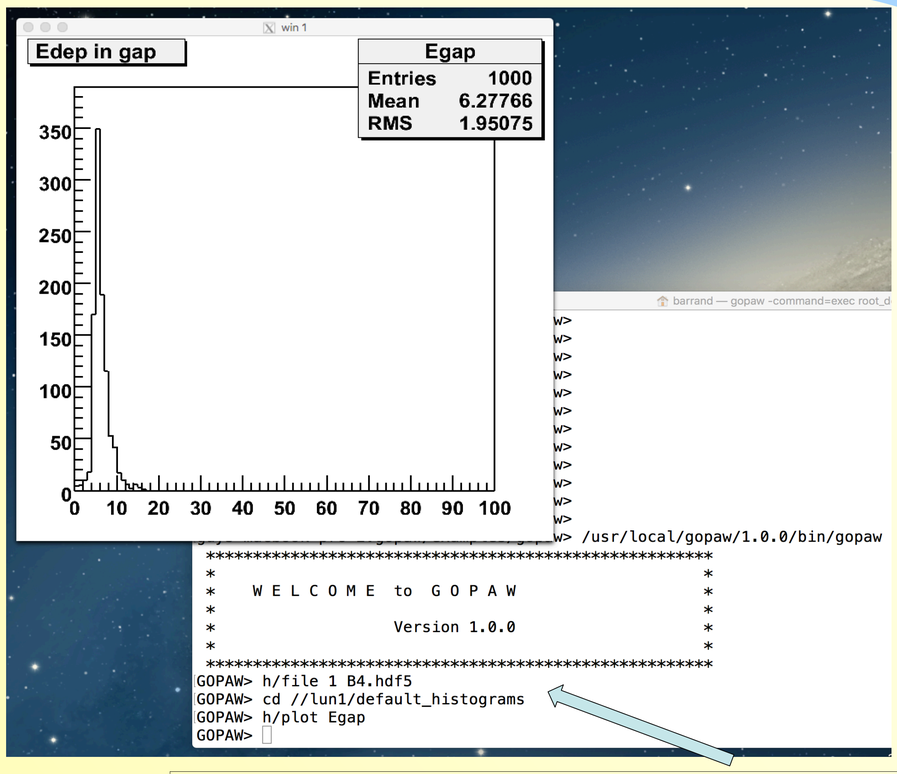
https://github.com/gbarrand/gopaw.git

https://gbarrand.github.io gopaw section.

gopaw, for “Good Old PAW”, is a refactoring of CERN-PAW done with softinex tools.

Motivation :

- One motivation to do **gopaw** came from the introduction of the **HDF5** file format in the **Geant4/Analysis** category (through the backend **g4tools** library) to store histograms and ntuples (see dedicated poster for this). Fine to produce files, but having interactive analysis tools that understand them would be highly welcome.
- And a tool that permits to open a file and plot an histo in a couple of **dedicated commands** would be great, especially if these commands are familiar to physicists.
- A **user API** comes naturally : **the one of PAW**.



B4.hdf5 file with histograms stored according the g4tools HDF5 data schema.

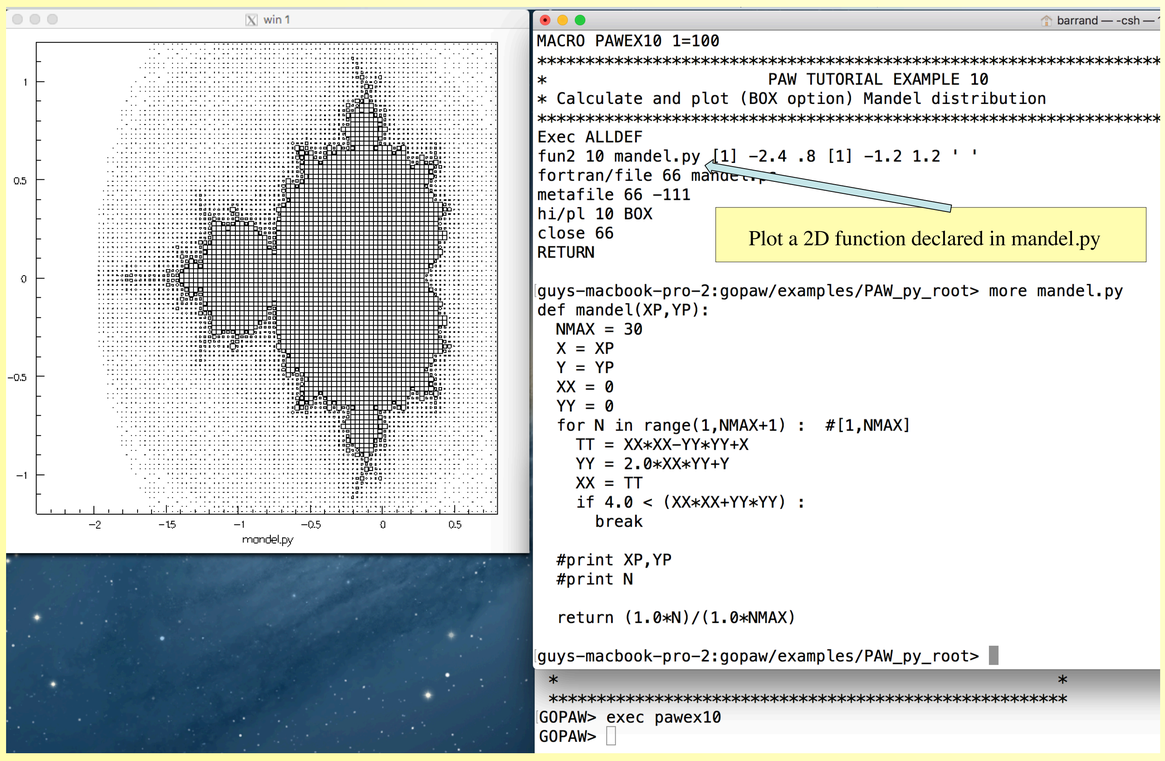
Commands :

- As the command engine is the **KUIP C** code extracted from CERNLIBs and that the command description files, the **paw.cdf** and **kuip.cdf**, are the same than the original PAW implementation, then we have in gopaw the **SAME** command syntax than PAW.
- A lot (but not all) commands and options are implemented.
- The examples pawex1.kumac up to pawex24.kumac are emulated with quite the same rendering than PAW.

Technology :

- **C/C++**.
- **GL-ES** for rendering engine.
- Our **inlib/sg** scene graph manager for high level graphics (same logic as OpenInventor).
- Getline to capture terminal commands.
- **KUIP** as a command engine.
- **Python** (2.7) (as a replacemenet for COMIS) used to define functions to be plotted or to fill histograms, etc...
- But “on the fly compilation and plugin loading” also available to define functions if a compiler is correctly declared to gopaw. (In fact if a fortran compiler is declared, someone can even recover PAW COMIS way of doing). This permit to have effective (since compiled) functions.
- **Can read .root files** (by using the **light inlib** classes for that).
- Then can read **HDF5** files but also **FITS** files of **astronomy**.
- Fitting is done by using the code of ROOT/TMinuit, arranged to depend only on STL, and put in the **inlib/f2cmn** class (copyright respected).
- We are not able to read .hbook files (we have no C library around to do that...).

With **Python** : here examples/PAW_py_root/pawex10.kumac and **mandel.py** :

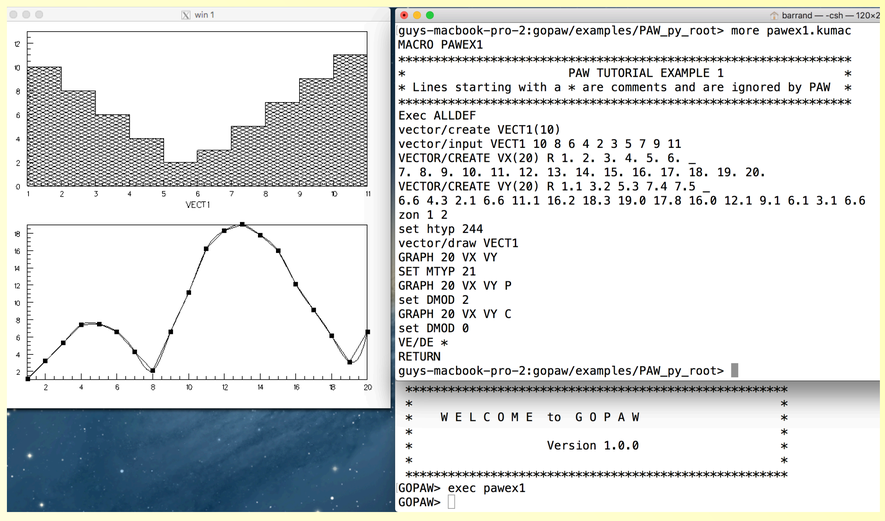


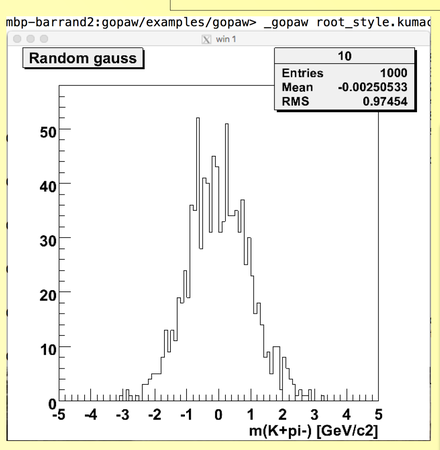
Plot a 2D function declared in mandel.py

VECTOR handling and **SIGMA** done with :

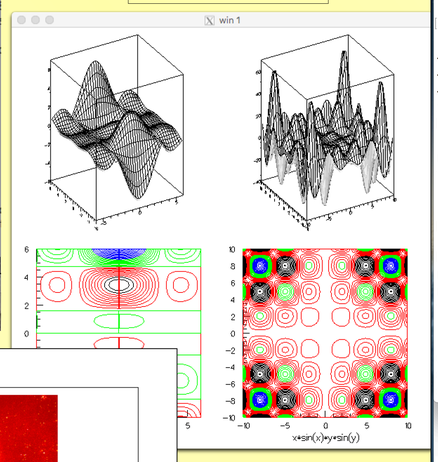
- inlib::array template class. A multidimensional array template over std::vector (then fast).
- SIGMA command done with the little exlib::yacc::cexpr_eval single C instruction interpreter. Something like V1 *V2 loops directly within std::vectors. Fast.

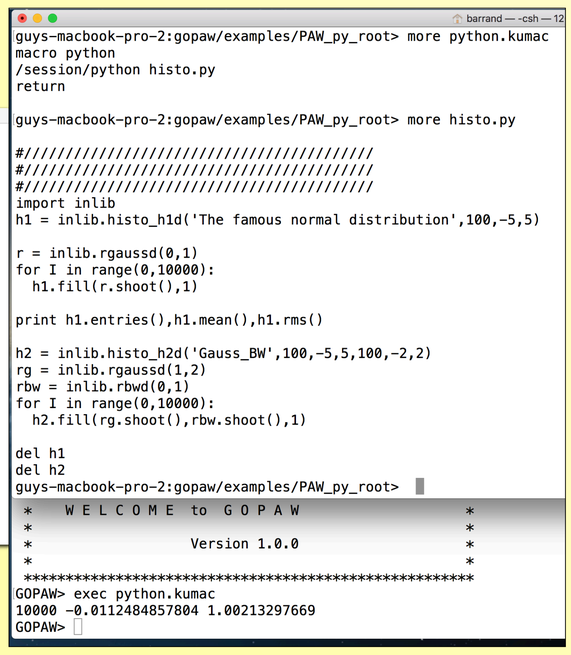
VECTOR manipulations : pawex1.kumac :





ROOT style





Astro hst.fits

on an **iPad**

Platforms :

- Portability is a high priority for us.
- **gopaw runs on macOS, Linux, Windows-10**. (We have also a docker on hub.docker/gbarrand).
- The core engine (including KUIP and Python) can be built also on **Android** and **iOS** and we have embedded a **“kumac reader”** in our **ioda** application.
- On Android and iOS, we have not yet finished the work to have an effective way to input commands from a virtual keyboard and then have not yet a gopaw app on the stores. Having a gopaw app on an iPad makes definitely sense for us and it remains a nice target to work on.

Why doing gopaw ? Also to try to answer a more fundamental question :

What is the best user API to do physics ? A dedicated (command ?) language with physics keywords or a C++ or Python prompt ?

23nd International Conference on Computing in High Energy and Nuclear Physics, July 9-13 2018, Sofia