DE LA RECHERCHE À L'INDUSTRIE

GENERAL OVERVIEW OF THE URANIE PLATFORM

HPC and Uncertainty Treatment with Open TURNS and Uranie, 10/05/2021

Jean-Baptiste Blanchard (jean-baptiste.blanchard@cea.fr), Fabrice Gaudier (fabrice.gaudier@cea.fr)

CEA DES/ISAS/DM2S/STMF/LGLS SACLAY

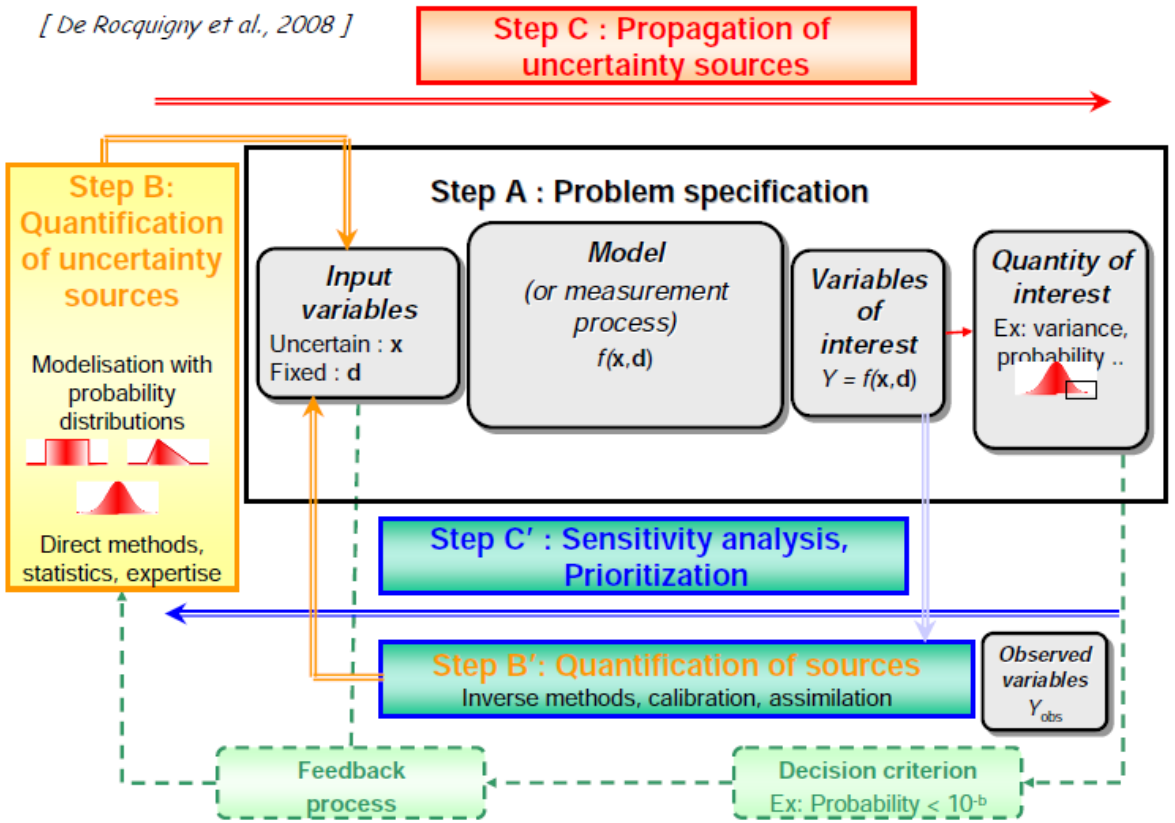
Outline

Reminder

In a nutshell

Focusing on Uranie

Workflow: breakdown into steps



Main steps:

- A: problem definition
  - Uncertain input variables
  - Variable/quantity of interest
  - Model construction
- B: uncertainty quantification
  - Choice of pdfs
  - Choice of correlations
- C: uncertainty propagation
  - Evolution of output variability w.r.t input ones
- B': quantification of sources
  - Inverse methods using data to constrain input values and uncertainties
- C': sensitivity analysis
  - Uncertainty source sorting

These steps are usually model dependent, it might be useful to iterate to help converging to proper conclusions

A generic analysis is shown in our platform description paper (published in EPJ-N): `https://doi.org/10.1051/epjn/2018050`

Technical aspects

Outline

Reminder

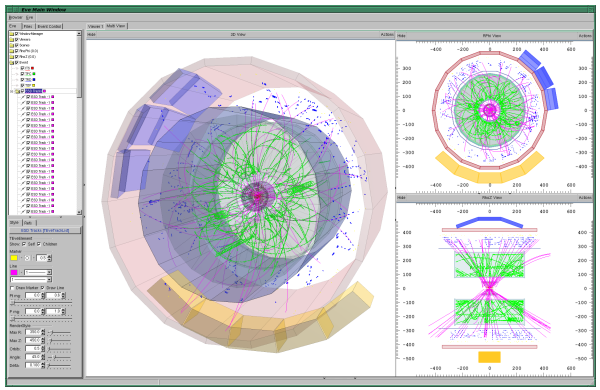
In a nutshell
- ROOT
- Uranie

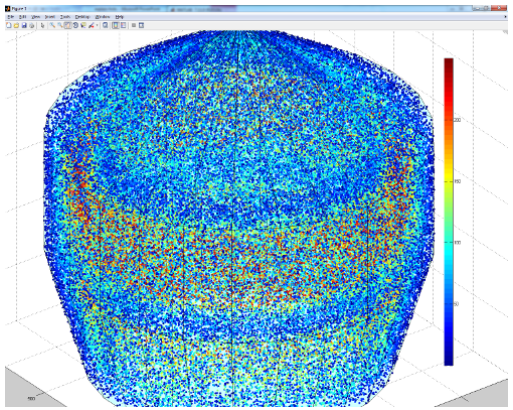
Focusing on Uranie

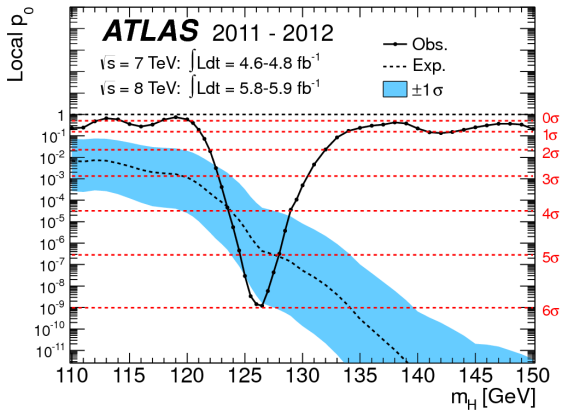
Developed at CERN to help analyse the huge amount of data delivered by the successive particle accelerators

- Written in C++ (3/4 releases a year)

- Multi platform (Unix/Windows/Mac OSX)
- Started and maintained over more than 20 years
- It brings:
  - a C++ on-the-flight compiler, but also Python and Ruby interface
  - a hierarchical object-oriented database (machine independent and highly compressed)
  - advanced visualisation tool (graphics are very important in HEP)
  - statistical analysis tools (*RooStats*, *RooFit*...)
  - and many more (3D object modelling, distributed computing interface...)
- LGPL







Many sources for documentation

Online

- Reference guide: https://root.cern.ch/root/html/ClassIndex.html
  → Details all the methods (inherited or not) of a given class
- User-guide:
  https://root.cern.ch/root/html/guides/users-guide/ROOTUsersGuideA4.pdf
  → What can be done from installation to high level usage. Nicely illustrated !
- How-to: https://root.cern.ch/howtos
  → Example to answer most answered questions
- A dedicated forum: https://root-forum.cern.ch/
  → Very reactive forum, to help people with the many different usage one can do with ROOT.

On your machine, once installed

- User guide and manual: They are provided in markdown, ready to be compiled
  → `$ROOTSYS`/documentation/users-guide and `$ROOTSYS`/documentation/primer
- Tutorials: plenty of examples to be run
  → `$ROOTSYS`/tutorials
- Macros: place to store your own macros that you might call from anywhere
  → `$ROOTSYS`/macros

This is a structure that we acknowledge and try to follow as well

Developed at CEA/DES to help partners handling sensitivity, meta-modelling and optimisation problems.

- Written in C++ (∼2 releases a year), based on ROOT
- Multi platform (developed on Unix and tested on Windows)
- It brings simple data access:
  - Flat ASCII file, XML, JSON ...
  - TTree (internal ROOT format)
  - SQL database access
- Provides advanced visualisation tools (on top of ROOT's one)
- Allows some analysis to be run in parallel through various mechanism
  - simple fork processing
  - shared-memory distribution (pthread)
  - split-memory distribution (mpirun)
  - through graphical card (GPU)
- Main purpose is tools for:
  - construction of design-of-experiment
  - uncertainty propagation
  - surrogate models generation
  - sensitivity analysis
  - optimisation problem
  - reliability analysis
- LGPL

Outline

Reminder

In a nutshell

Focusing on Uranie
- Organisation
- Communication and interoperability
- The modular organisation

General organisation: version 4.5.0

General description:

- ROOT version: 6.22.02
- 13 modules / $\sim$ 280 classes
  $\sim$ 154 000 lines of code
- Compilation using CMAKE

Unit Testing Report

	DataServer	Launcher	Relauncher	Sampler	Sensitivity	Optimizer	reOptimizer	Modeler	UncertModeler	reLiability	XMLProblem
Status	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED
Duration											
Num. test	328	112	39	176	115	139	46	429	53	2	13
Total Failures	0	0	0	0	0	0	0	0	0	0	0
Num. Errors	0	0	0	0	0	0	0	0	0	0	0
Num. Failures	0	0	0	0	0	0	0	0	0	0	0
Start	2018-01-09 20:15:10	2018-01-09 20:16:38	2018-01-09 20:31:36	2018-01-09 20:32:26	2018-01-09 20:33:03	2018-01-09 20:59:22	2018-01-09 21:11:42	2018-01-09 21:38:09	2018-01-09 22:09:47	2018-01-09 22:09:51	2018-01-09 22:09:51
End	2018-01-09 20:16:38	2018-01-09 20:31:35	2018-01-09 20:32:26	2018-01-09 20:33:03	2018-01-09 20:59:19	2018-01-09 21:11:40	2018-01-09 21:38:07	2018-01-09 22:09:45	2018-01-09 22:09:50	2018-01-09 22:09:51	2018-01-09 22:12:45

Regularly tested:

- 7 Linux platforms and Windows 10 every night
- $\sim$ 1650 unitary tests with CPPUNIT
- $\sim$ 83% coverage with GCOV (without logs)
- Memory leak check with VALGRIND

Documentation: 3 different levels

- Methodological reference ($\sim$ 90 pages)
- User manual: $\sim$ 1050 pages
  $\sim$ 340 pages: describing methods and their options.
  $\sim$ 350 pages: C++ macros ($\sim$ 130 examples)
  $\sim$ 350 pages: Python macros ($\sim$ 130 examples)

Developer's guide using DOXYGEN (HTML only)

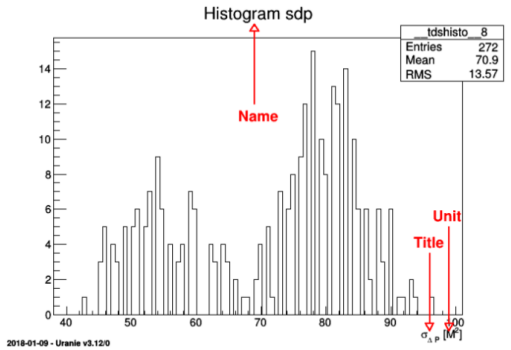
- Name + title: constructor defined from the name and the title of the variable

```
TAttribute *psdp = new TAttribute("sdp", "#sigma_{#Delta P}");
psdp->setUnity("M^{2}");
```

A pointer **psdp** to a variable **"sdp"** is available with title being *#sigma_{#Delta P}*. The command **setUnity()** precises the unit. In this case, by default, the field *key* is identical to the field *name*. We will use the ability given by ROOT to write LaTeX expressions in graphics to improve graphics rendering without weighing down the manipulation o variables: as a matter of fact, we can plot the histogram of the variable *sdp* by:

```
tdsGeyser->addAttribute("newx2","x2","#sigma_{#Delta P}","M^{2}");
tdsGeyser->draw("newx2");
```

The result of this piece of code is shown in Figure II.3



Uranie's approach

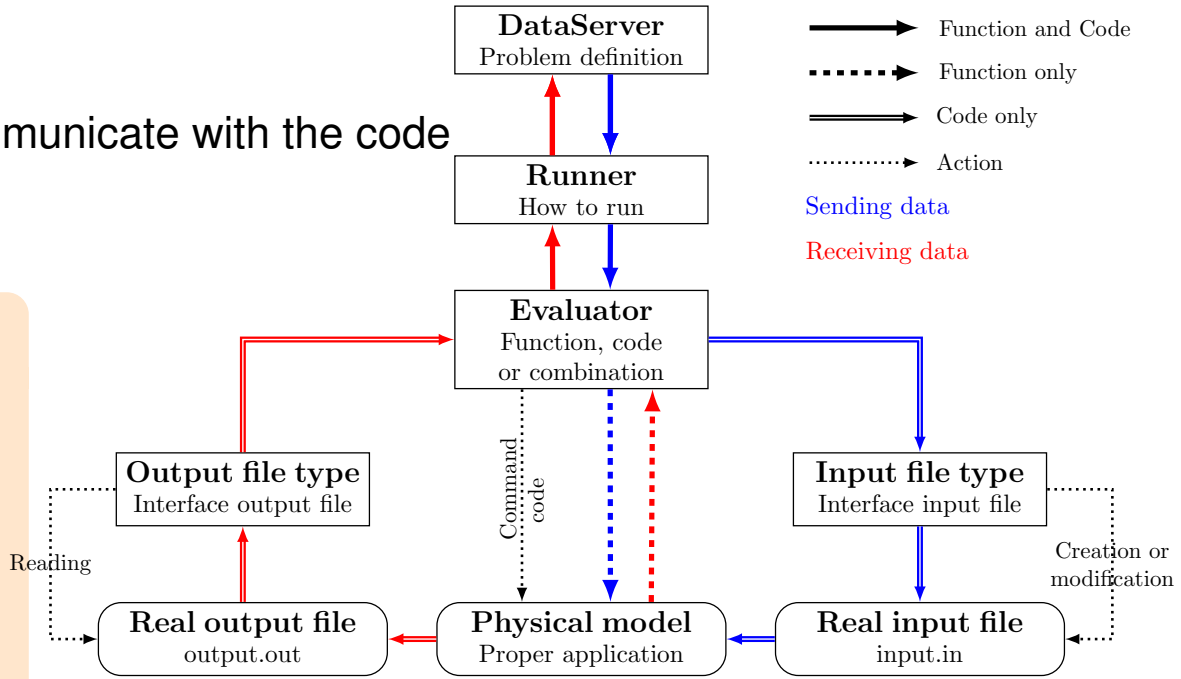
- Non-intrusive: code is a black box that cannot be modified but for some allowed parameters

Nature of Evaluators

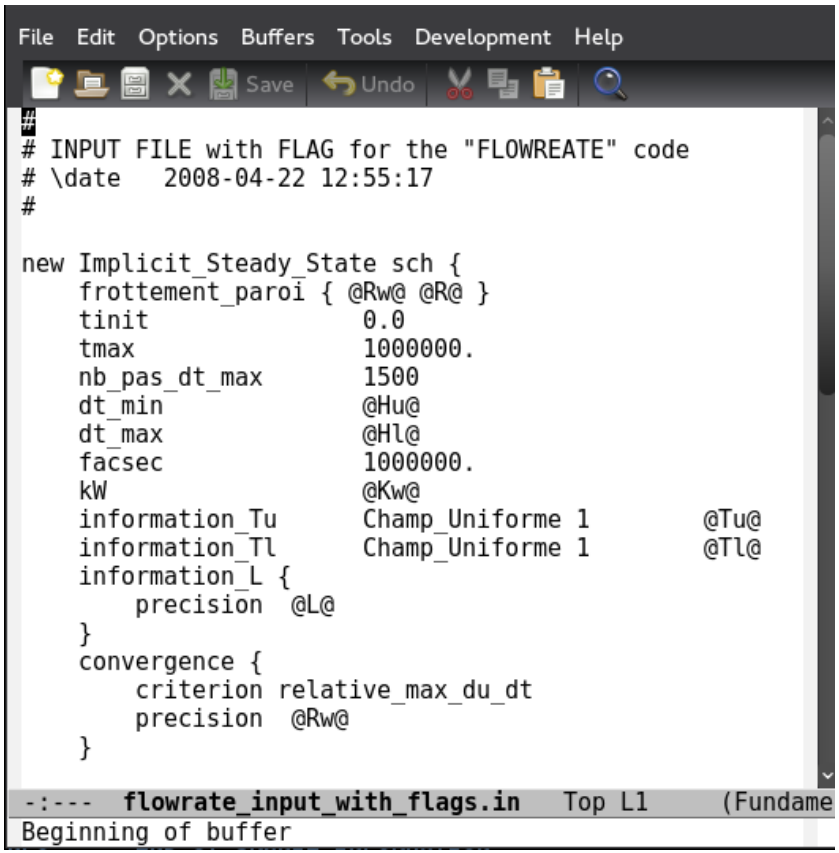
- C++ compiled function
- python function
- external code
  - → Need input / output files to communicate with the code
- chain of all aforementioned types

Ways of submitting jobs

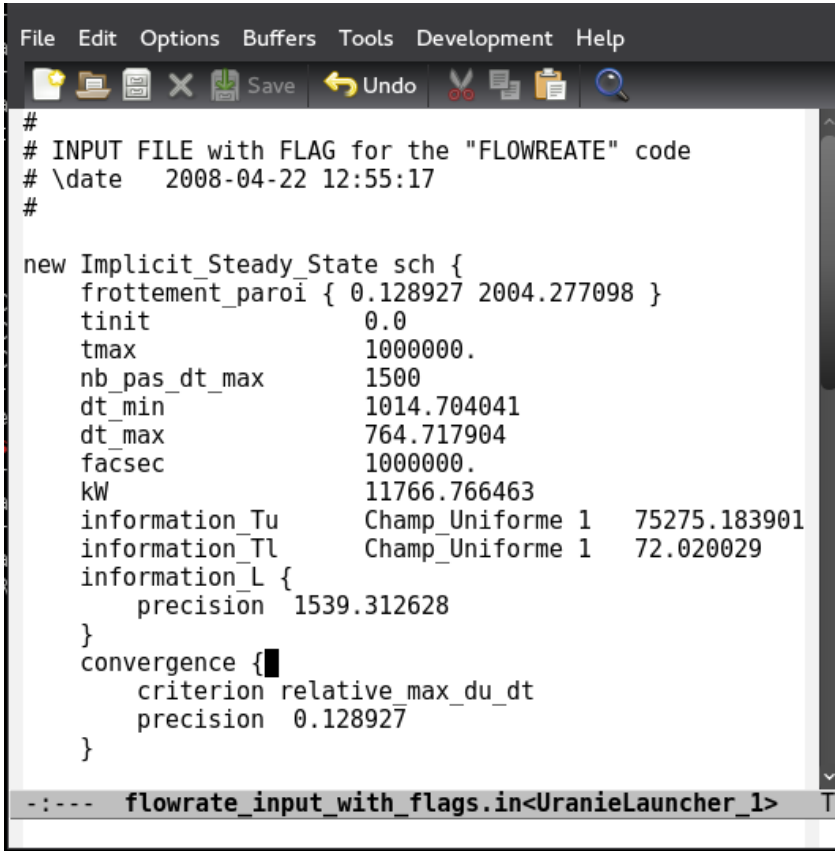
- Sequentially
- Forking the code
- Shared-memory distribution `pthread`
- Split-memory distribution `mpirun`
- Distributed on certain clusters



Example of flag format



File containing flags

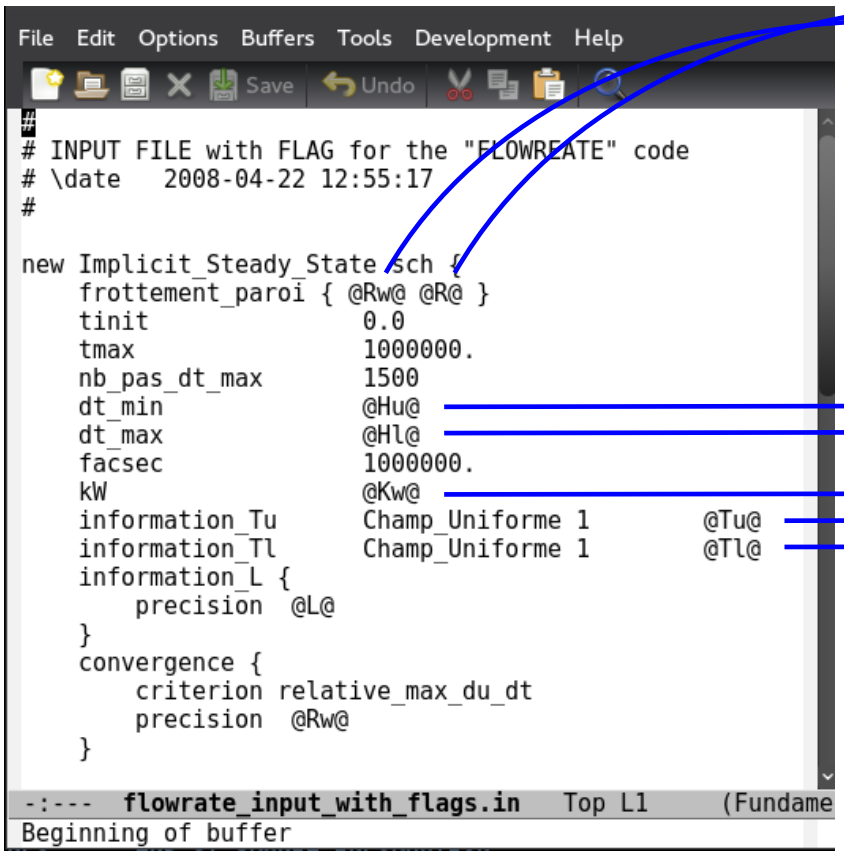


Modified file

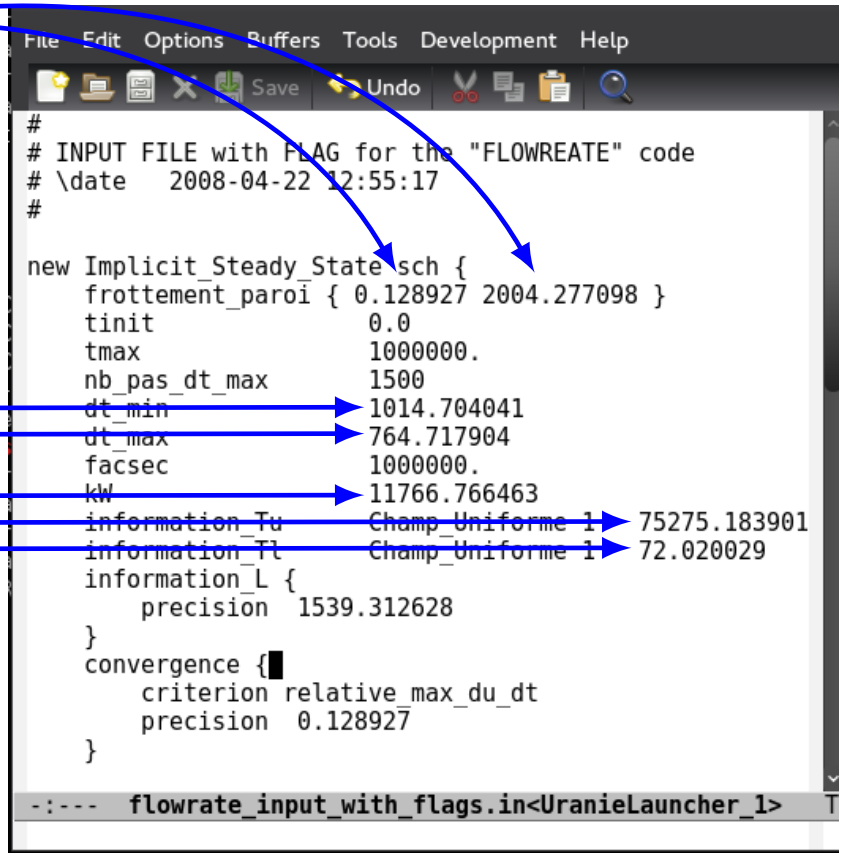
Advantage

Allow to keep a complicated input file, as long as its structure does not change

Example of flag format



File containing flags



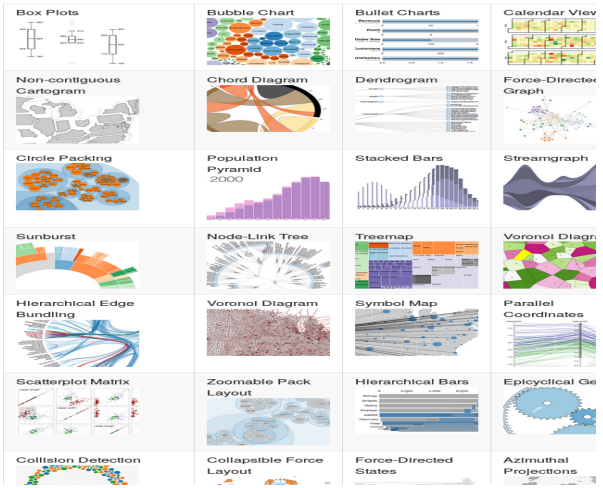
Modified file

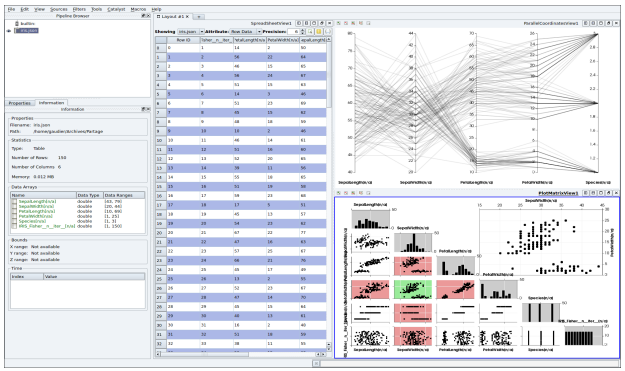
Advantage

Allow to keep a complicated input file, as long as its structure does not change

Use standard input/output language to import/export data and models, to help communicate with other platforms (XML, PMML, JSON. . .)


```
{
    "_metadata" : {
        "table_name" : "IRIS_Fisher",
        "table_description" : "Fisher Iris Data Set",
        "short_names" : [
            "SepalLength", "SepalWidth",
            "PetalLength", "PetalWidth", "Species" ],
        "date" : "Thu Mar 17 11:40:48 2016"
    }
    "items" : [ {
    "PetalLength" : 14, "PetalWidth" : 2,
    "SepalLength" : 50, "SepalWidth" : 33, "Species" : 1
    }, "items" : { ...
```



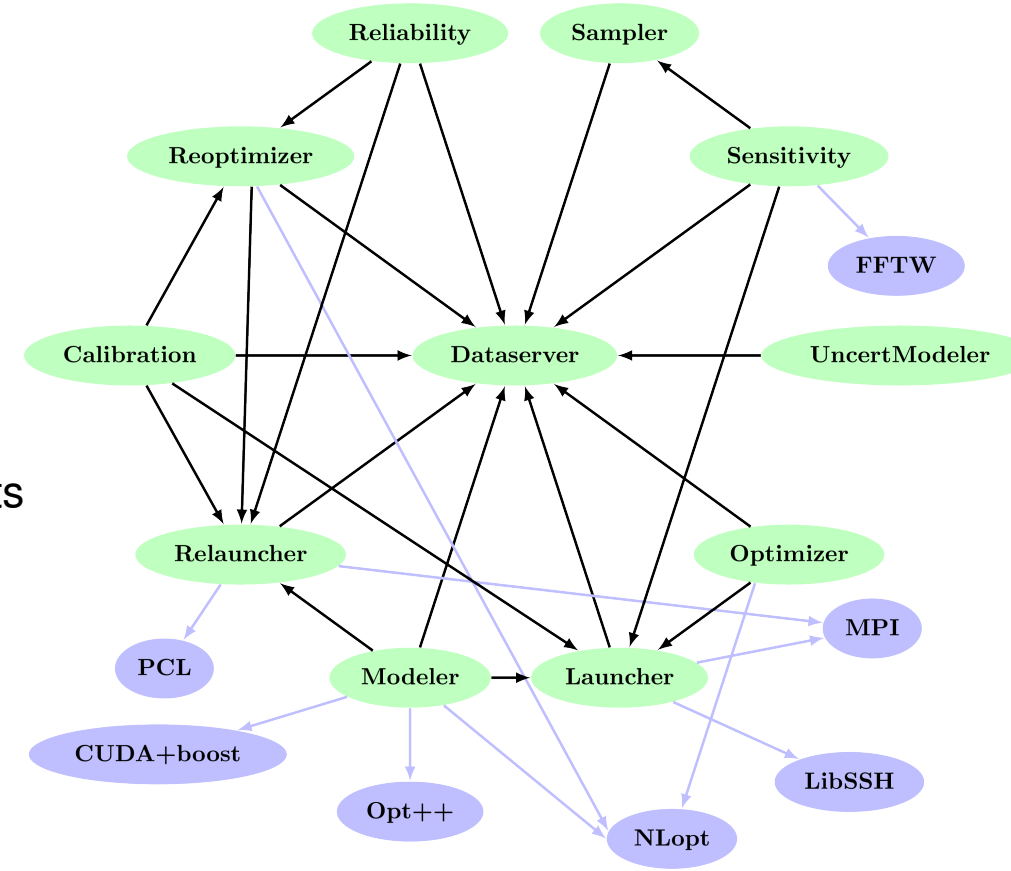
Import/Export data in Json format in order to :

- Benefit the features of D3 (D3js.org)
  - Interactive visualisation into a browser
  - Several available graphics (Cobweb, Sun-Burst, Treemap,..)
- Visualize the same data file in ParaView / Paravis module of Salomé
- Proposal as a common format for data with OpenTURNS

The platform organisation

Organised in modules

- Some are more technical ones:
  - `DataServer`: data handling and first statistical treatment
  - `Launcher/ReLauncher`: interface to code/ functions
- Many are dedicated ones:
  - `Sampler`: creation of design of experiments
  - `Modeler`: surrogate model generation
  - `Optimizer/Reoptimizer`: mono/multi criteria optimisation problems
  - `Sensitivity`: input variable sorting w.r.t impact on the output
  - `Reliability`: estimate rare probabilities
  - `Calibration`: estimate the parameter model values

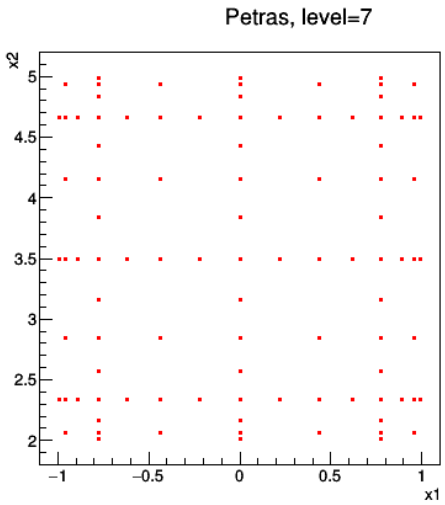


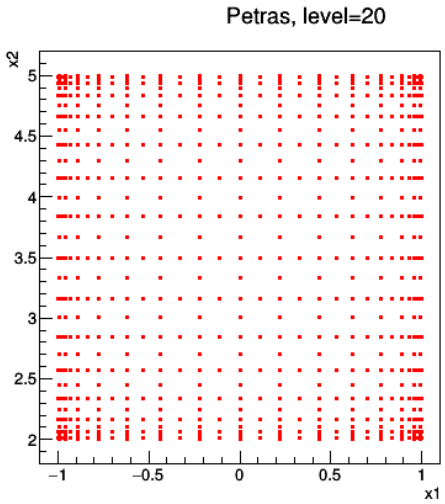
A glimpse at the main modules

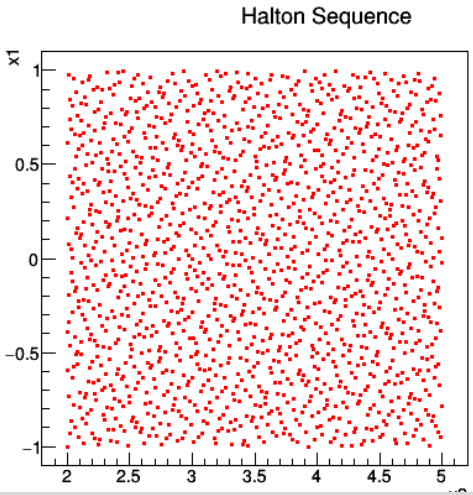
Used to generate the design-of-experiments, basis of many analysis.
Some methods can deal with correlation as well.

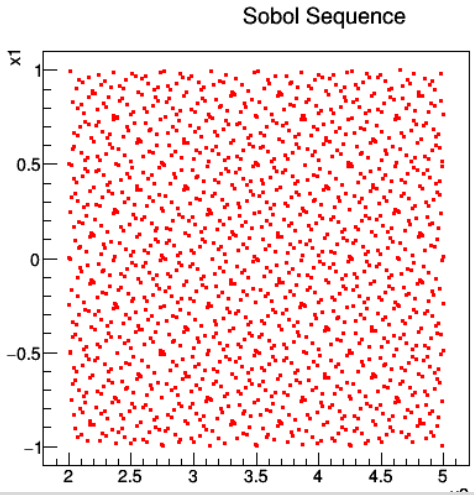
Two main categories

- Stochastic designs:
  - Simple Random Sampling (SRS)
  - Latin Hypercube Sampling (LHS)
  - One-At-a-Time Sampling (OAT)
  - Archimedian copulas
  - Random fields...
- Deterministic designs:
  - Regular quasi Monte-Carlo: Halton/Sobol sequence
  - Sparce grid sampling: Petras
  - Space filling design





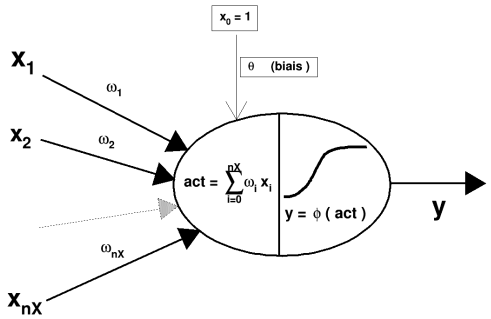


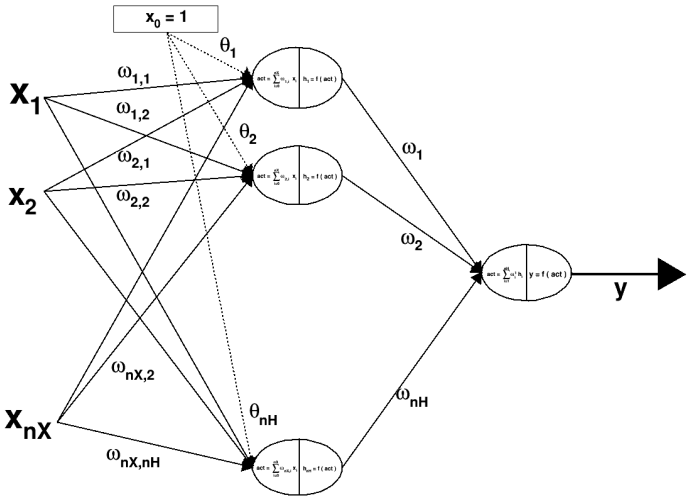


Create a surrogate-model: a numerical model reproducing the behaviour of provided data

Several possible models to be chosen:

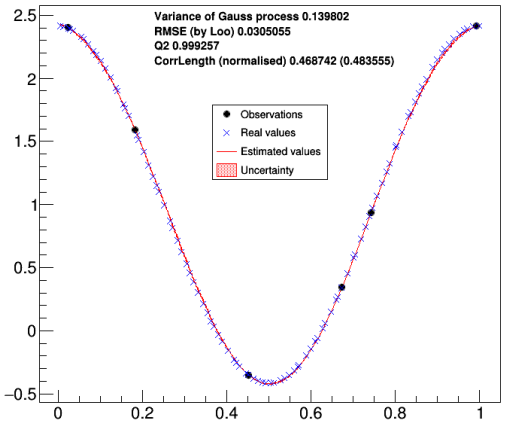
- Polynomial regressions
- Generalised linear models
- k-nearest neighbours
- Artificial Neural Networks (ANN/MLP)
- Chaos Polynomial + ANISP
- Kriging



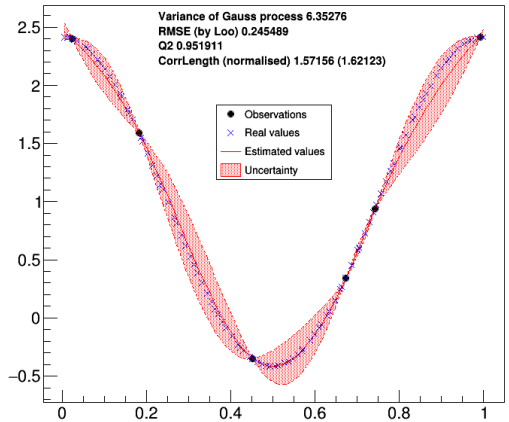


→ Models can be exported in different format (C++, fortran, PMML) to be re-used later on.

gauss_LOO_Subplexe__100_1000



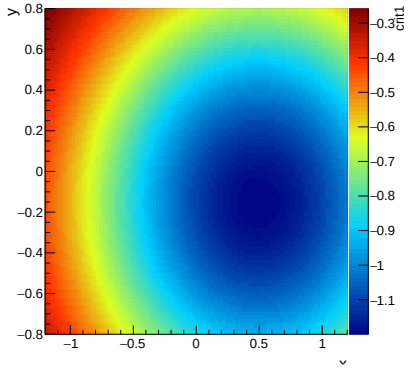
matern3-2_LOO_Subplexe__100_1000



Optimizer module

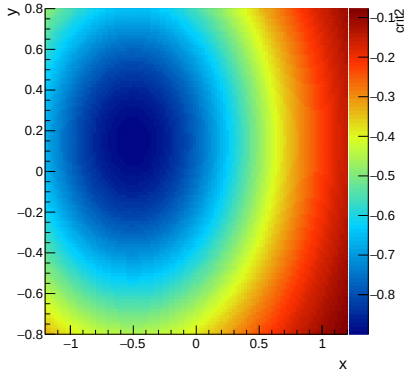
Dealing with optimisation problem usually means:

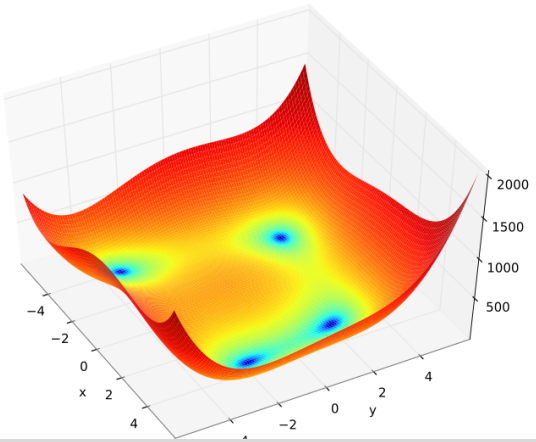
- Minimise Single Objective (SO) / Multi Objectives
- parameters that have an impact on objective
- possible constraint on these parameters

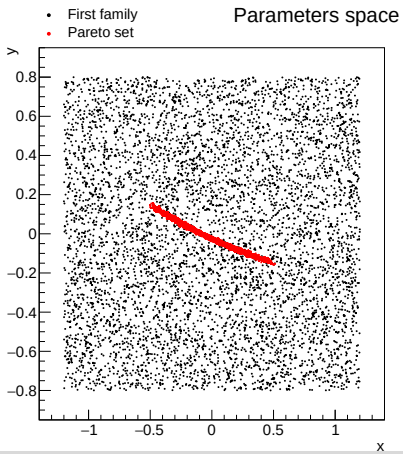


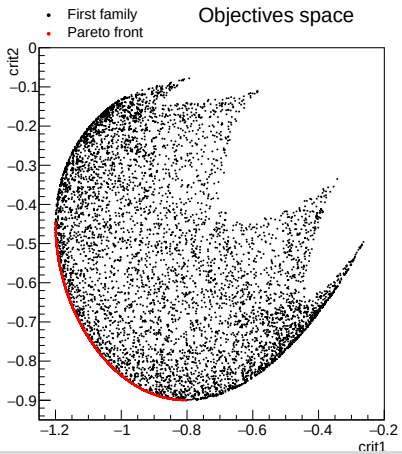
Many possible implementation for this, based on:

- Minuit: ROOT's SO optimisation library without constraint
- Opt++: SO optimisation library with/without constraint
- NLopt: SO optimisation library with/without constraint
- **Vizir**: CEA's MO optimisation library with/without constraint, based on stochastic algorithms (*e.g.* genetic ones)





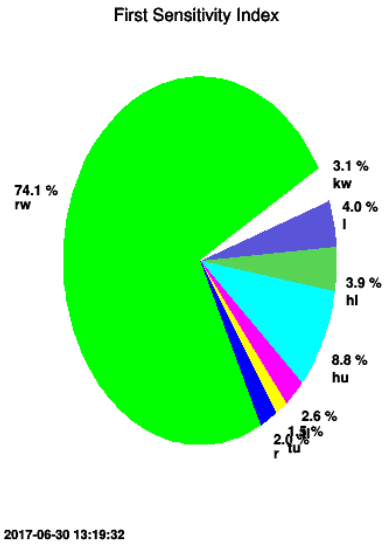


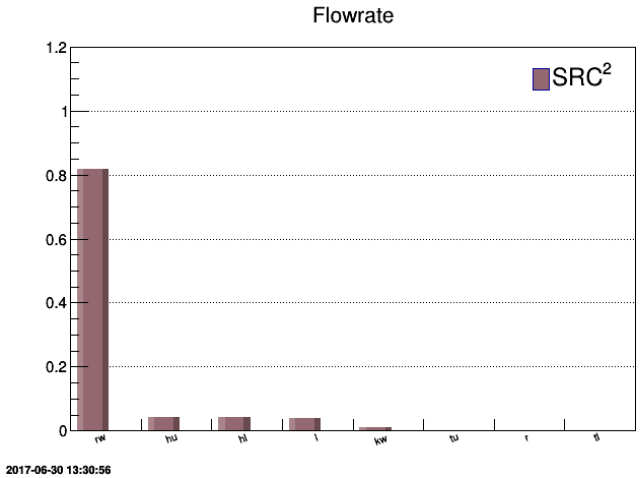


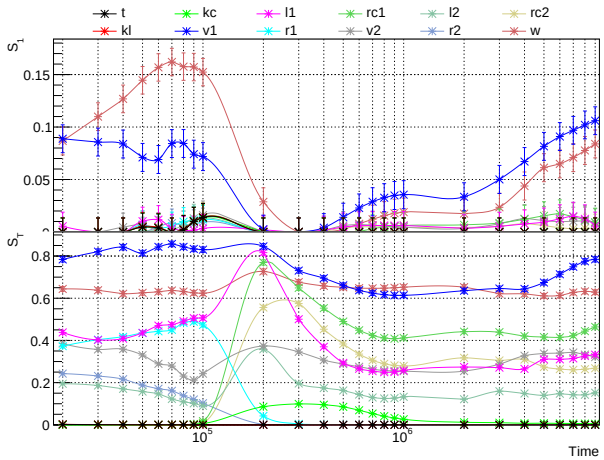
Tools to evaluate the sensitivity of outputs to the function/code inputs.

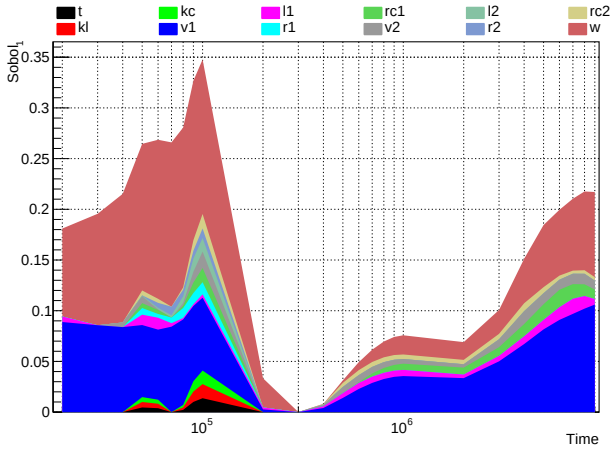
Several kinds of methods available:

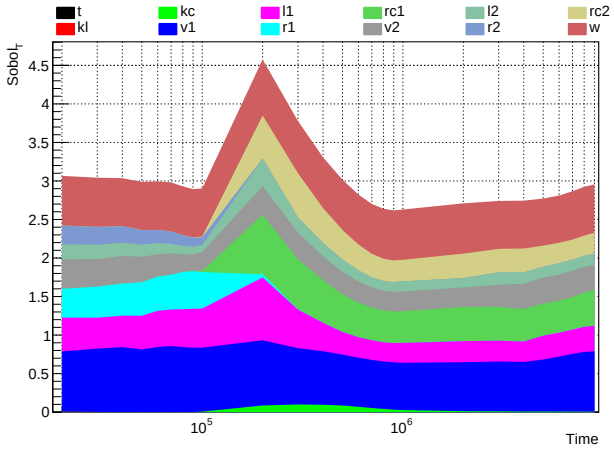
- Screening: OAT, Morris. . .
- Regression: Pearson / Spearman
- Sobol indexes:
  - Sobol/Saltelli Methods
  - Fourier-based: FAST, RBD. . .
- Correlated issues:
  - Johson relative weight
  - Shapley indices











Plans for the future

Technical improvements

- Simplify interfacing with IA platforms
- Porting more methods on GPU (kNN and ANN so far)

Methodological improvements

- Combine Hamiltonian Markov-chain and ANN
- Improve our range of sensitivity techniques
- Improve our Bayesian calibration (through various approaches)
- Improve many-criteria algorithms from VIZIR

Feel free to test the platform

The code is available here: http://sourceforge.net/projects/uranie

- All documentations are embedded in the archive
- We give 2-3 formation sessions a year
  → Dedicated session also on specific modules once every 18 month (roughtly)
- Can contact us at support-uranie@cea.fr

More information can be found in our recent paper (published in EPJ-N):
https://doi.org/10.1051/epjn/2018050

Thanks! Any questions?