DE LA RECHERCHE À L'INDUSTRIE

General overview of the Uranie platform

Uncertainty PRACE formation session

J-B. Blanchard, F. Gaudier
jean-baptiste.blanchard@cea.fr, support-uranie@cea.fr

16/05/2018

www.cea.fr

In a nutshell

- ROOT
- Uranie

Focusing on Uranie

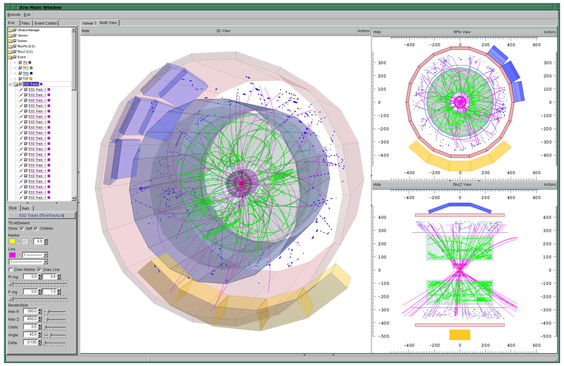
- Schematic workflow examples
- The modular organisation

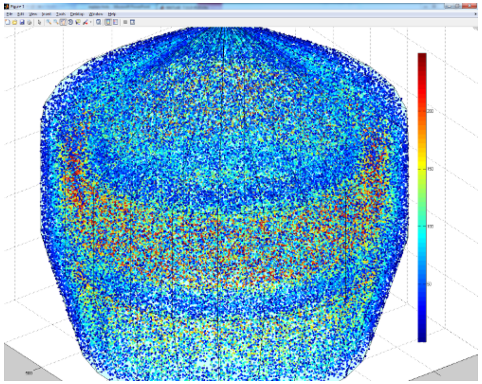
Module description

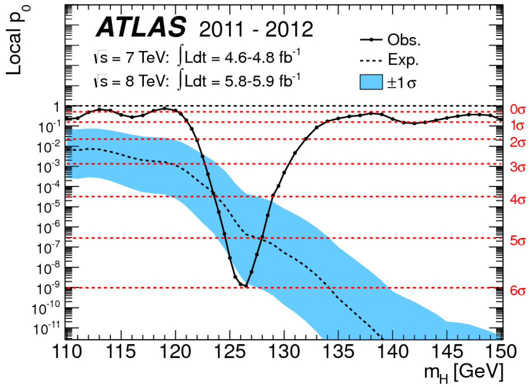
The ROOT platform

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++ interpreter, 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/html534/ClassIndex.html
  - ➜ Details all the methods (inherited or not) of a given class
- User-guide: https://root.cern.ch/root/html534/guides/users-guide/ROOTUsersGuideA4.pdf
  - ➜ Description of 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

The Uranie platform

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

- Written in C++ ($\sim$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

General organisation: version 3.12

General description:

- ROOT version: 5.34.36
- 11 modules / 246 classes
  ~ 134 000 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 7 every night
- ~ 1500 unitary tests with CPPUNIT
- ~ 83% coverage with GCOV (without logs)
- Memory leak check with VALGRIND

Documentation: 3 different levels

2 using DOCBOOK, generating both PDF and HTML formats.

- Methodological reference (~ 60 pages)
- User manual: ~ 550 pages
  ~ 250 pages: describing methods and their options.
  ~ 250 pages: use-case macros (~ 100 examples)

Developer's guide using DOXYGEN (HTML only)

- describing methods from comments in the code

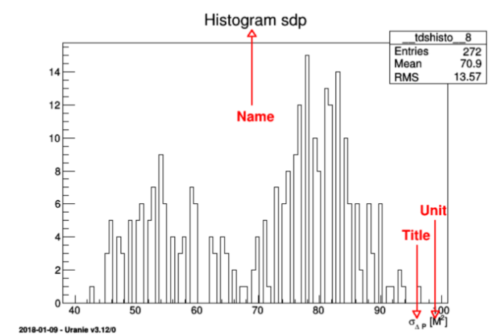
- Name + title: constructor defined from the name and the title of the variable

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

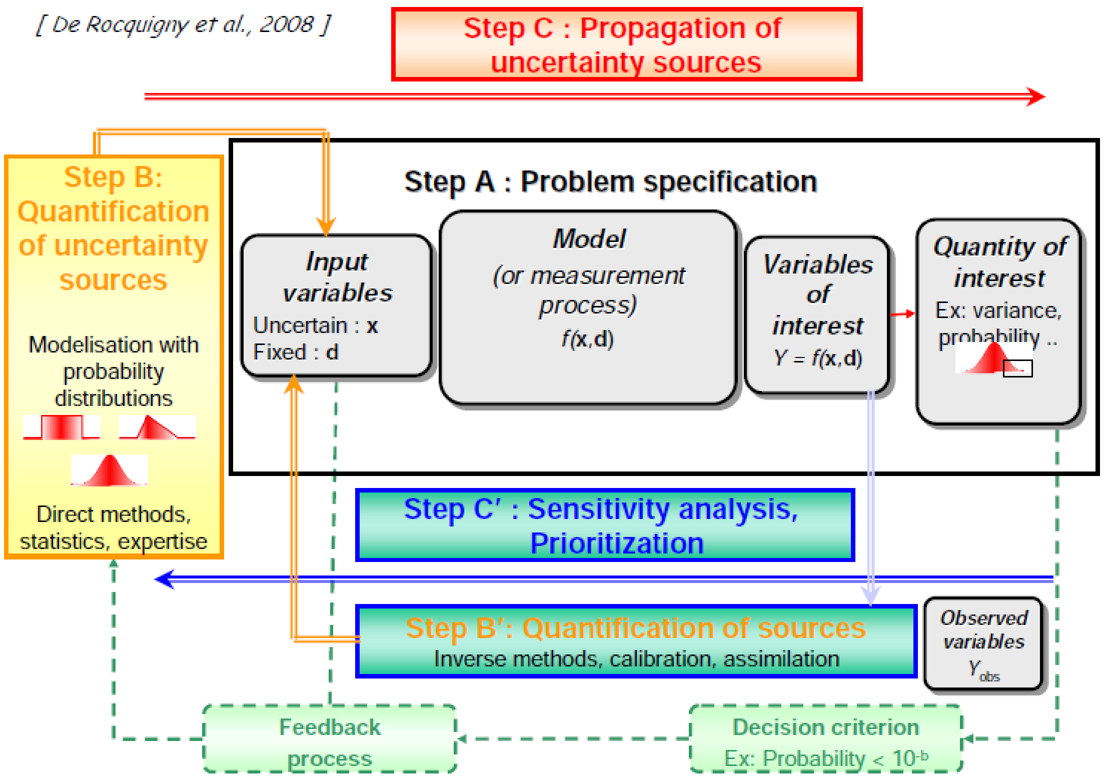
A pointer **psdp** to a variable **"sdp"** is available with title being *#sigma_{#Delta P}*. The command **setUnity()** precises the unit. In this case, by default, the field *key* is identical to the field *name*. We will use the ability given by ROOT to write LaTeX expressions in graphics to improve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



Workflow: breakdown into steps

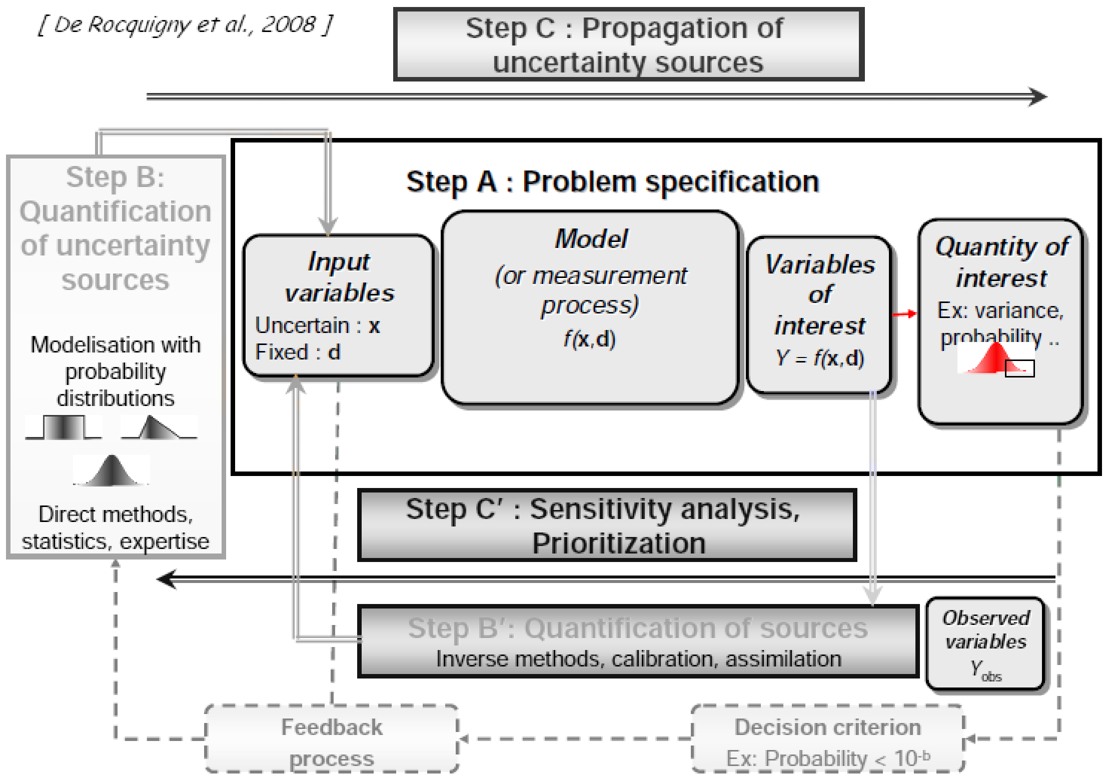


Main steps:

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

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

Workflow: breakdown into steps

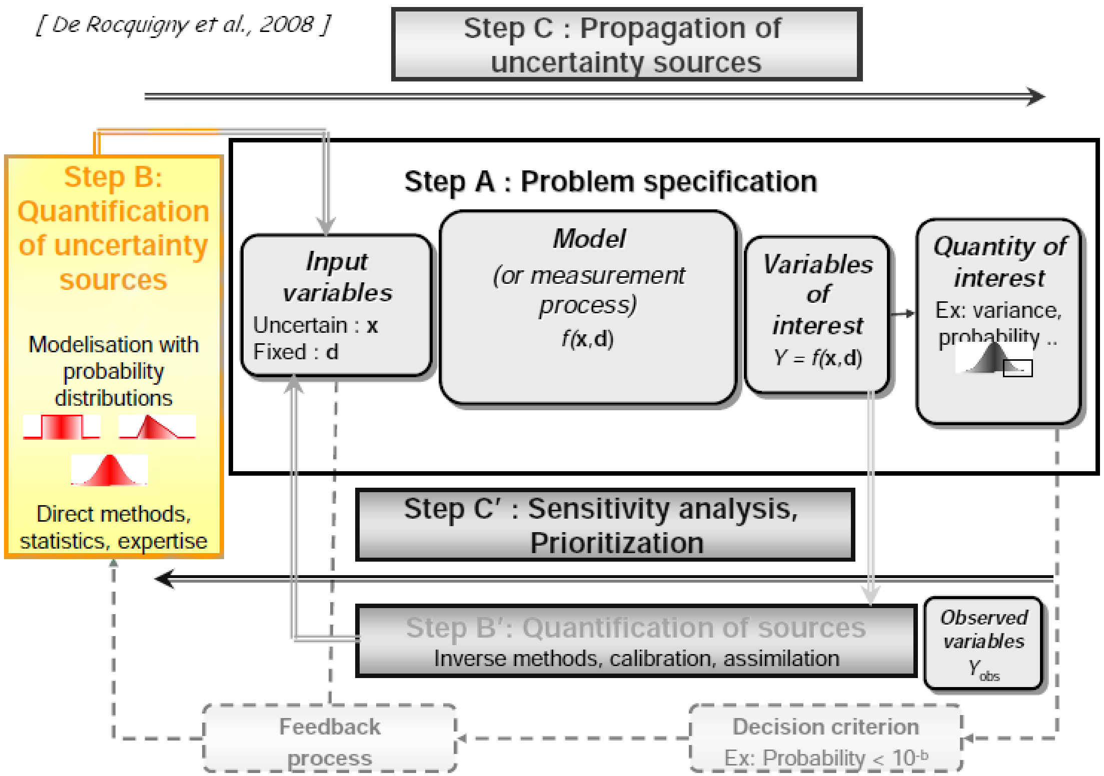


Main steps:

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

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

Workflow: breakdown into steps

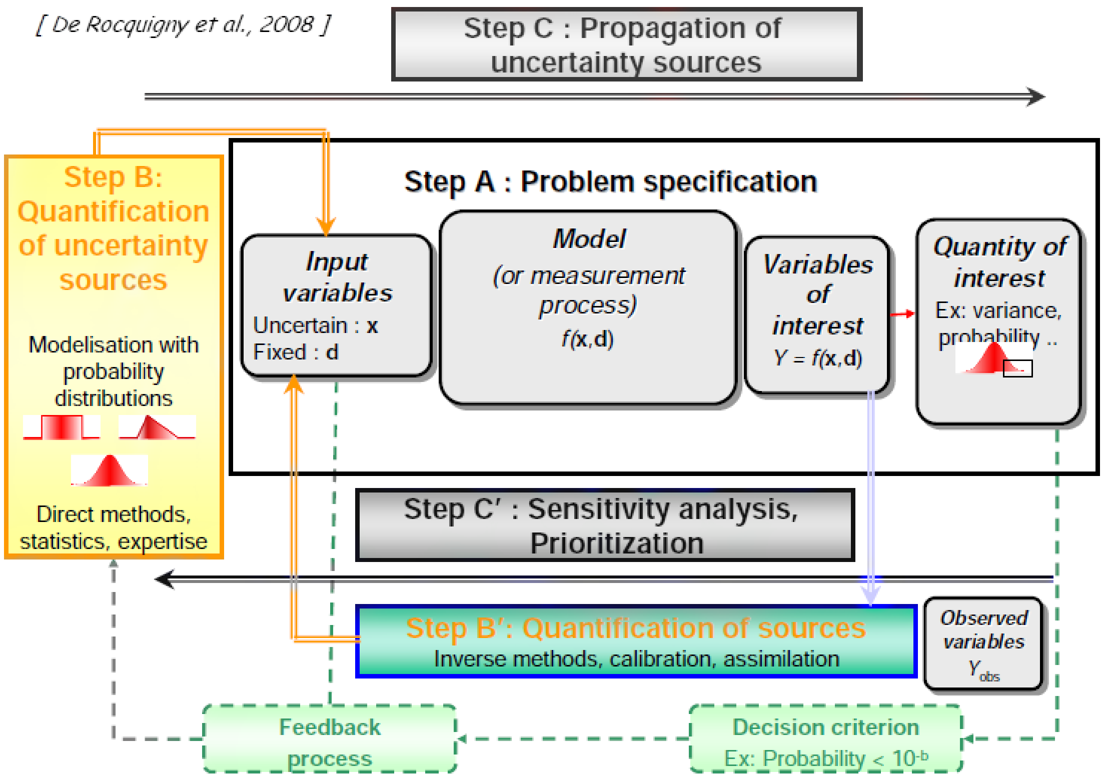


Main steps:

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

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

Workflow: breakdown into steps

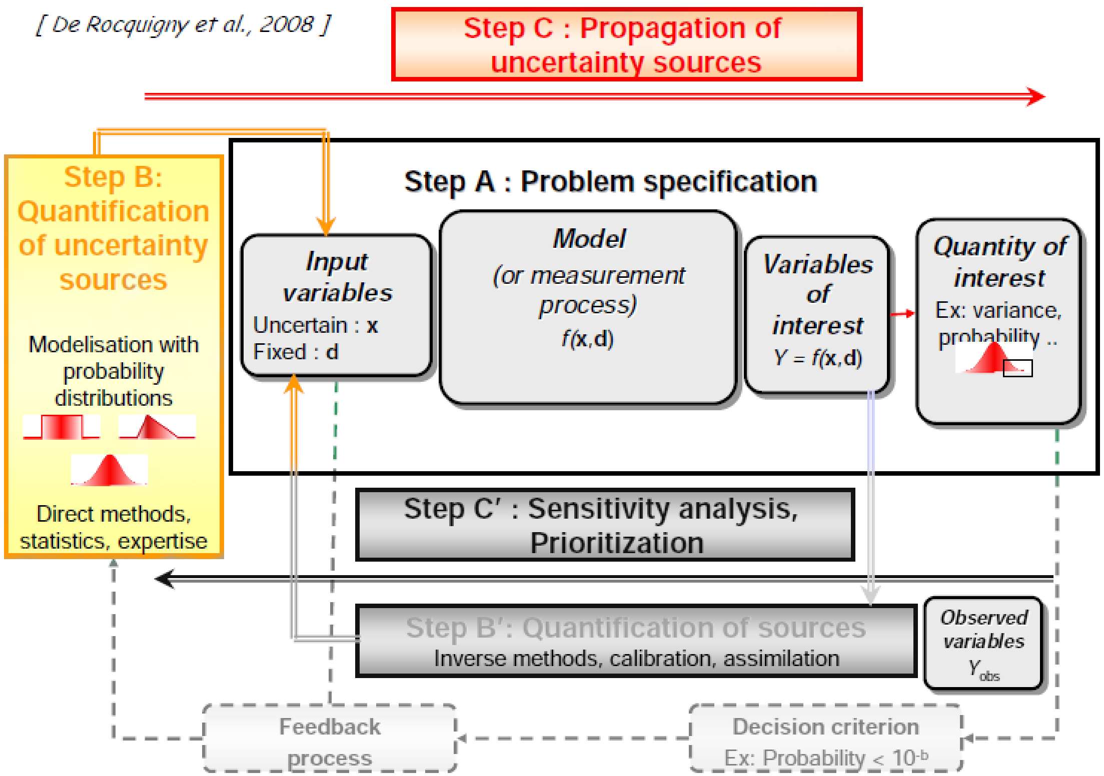


Main steps:

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

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

Workflow: breakdown into steps

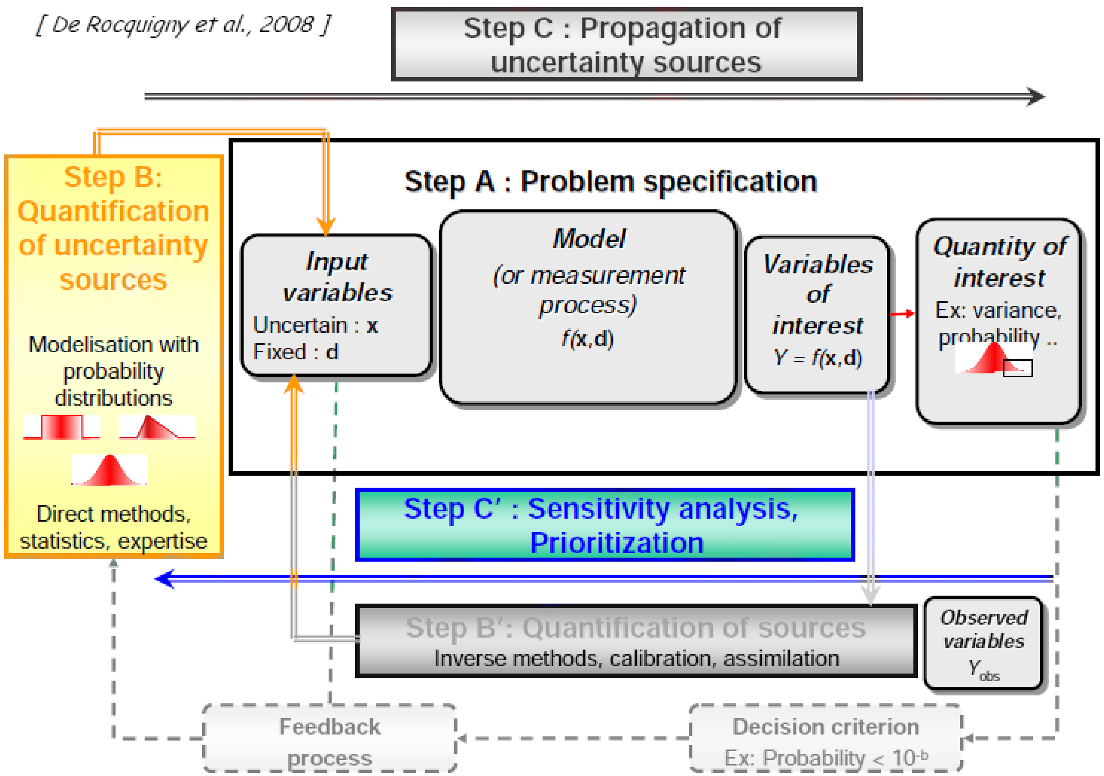


Main steps:

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

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

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
- B': quantification of sources
  - Inverse methods using data to constrain input values and uncertainties
- C: uncertainty propagation
  - Evolution of output variability w.r.t input uncertainty
- C': sensitivity analysis
  - Uncertainty source sorting

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

The module point of view

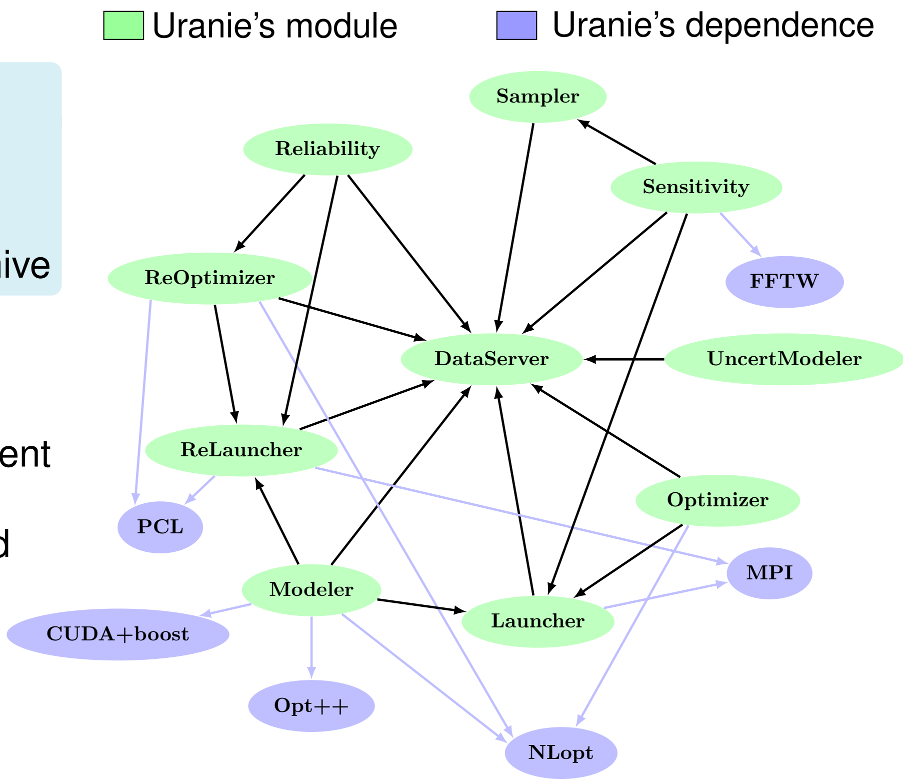
Few dependencies:

- Compulsory: ROOT, CPPUNIT, CMAKE
- Optional: PCL, NLOPT, OPT++*, MPI, FFTW, CUDA

(*) a patched version of OPT++ is brought along in the archive

Organised in modules:

- Some are more technical ones:
  - DataServer: data handling and first statistical treatment
  - (Re)Launcher: interfaces to code/function handling. Can deal with code, PYTHON-function, C++-interpreted and compiled functions
- Many are dedicated ones:
  - Sampler: creation of design-of-experiments
  - Modeler: surrogate-model generation
  - (Re)Optimizer: mono/multi criteria optimisation
  - Sensitivity: ranking inputs w.r.t impact on the output



The next following slides will discuss the content of the main dedicated modules

A glimpse at the main modules

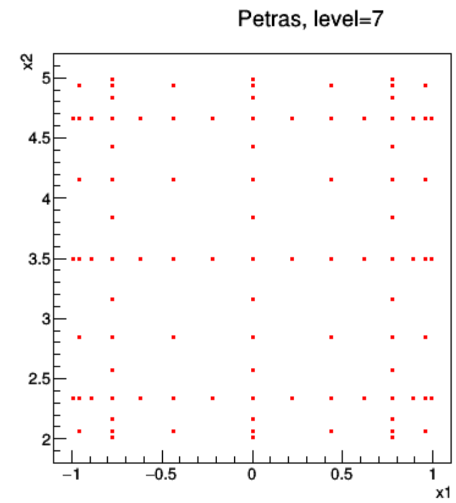
DE LA RECHERCHE À L'INDUSTRIE

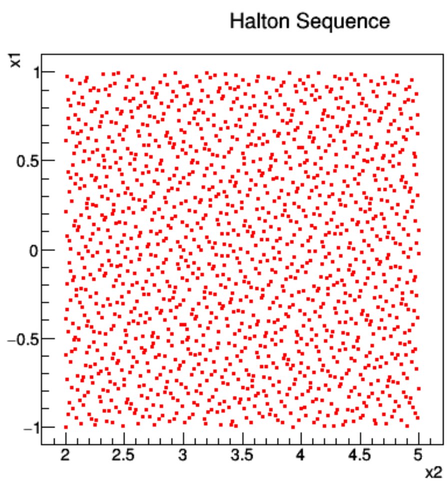
The sampler module

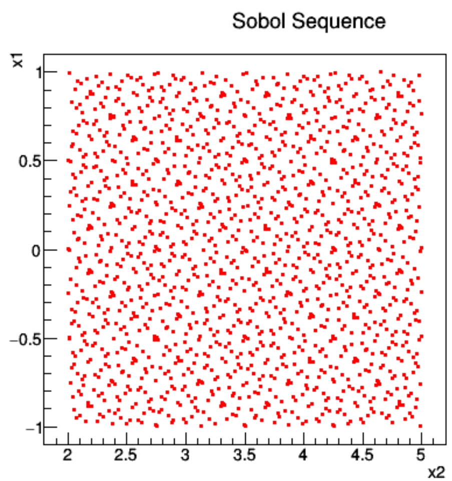
Used to generate the design-of-experiments, basis of many analysis.
Some methods can deal with correlation as well.

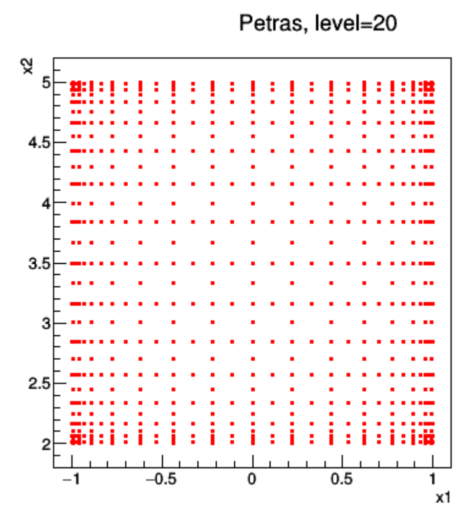
Two main categories

- 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







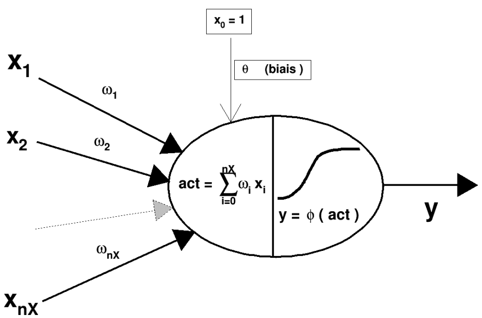


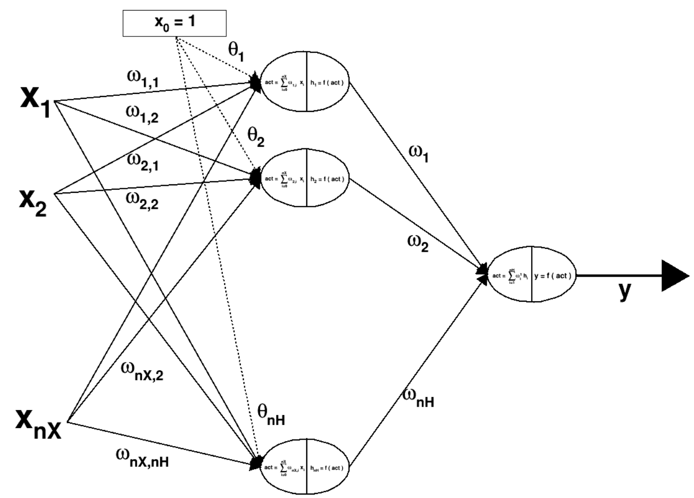
The modeler module

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

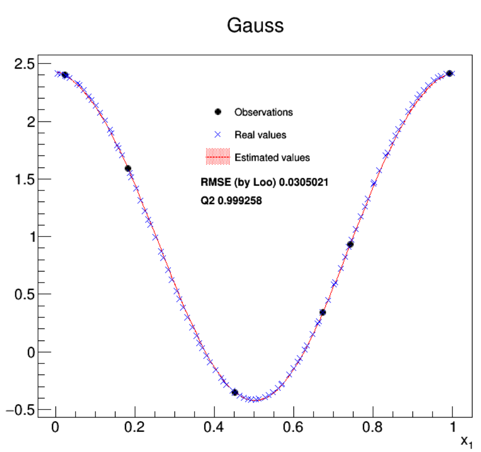
Several possible models to be chosen:

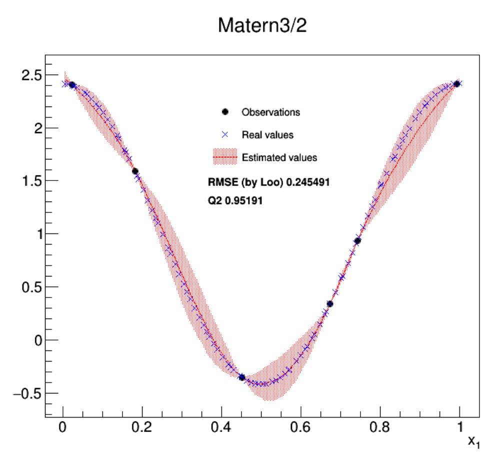
- Polynomial regressions
- Generalised linear models
- k-nearest neighbours
- Artificial Neural Networks (ANN/MLP)
- Chaos Polynomial + ANISP
- Kriging





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



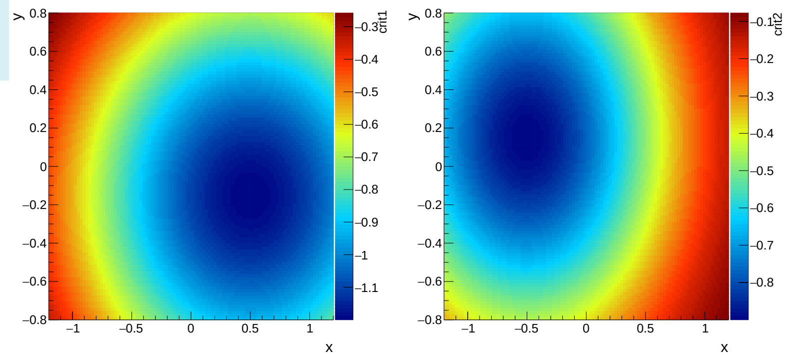


2017-06-30 13:16:31

Optimizer module

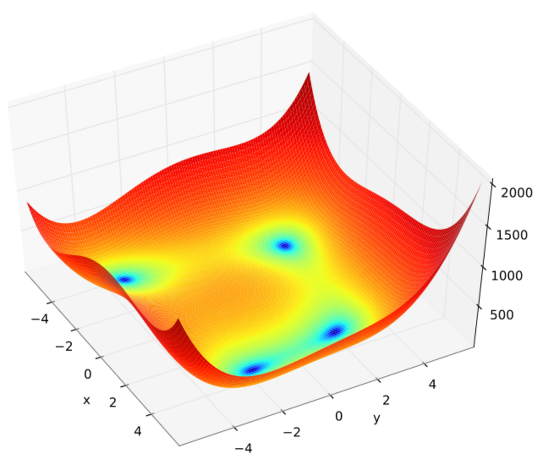
Dealing with optimisation problem usually means:

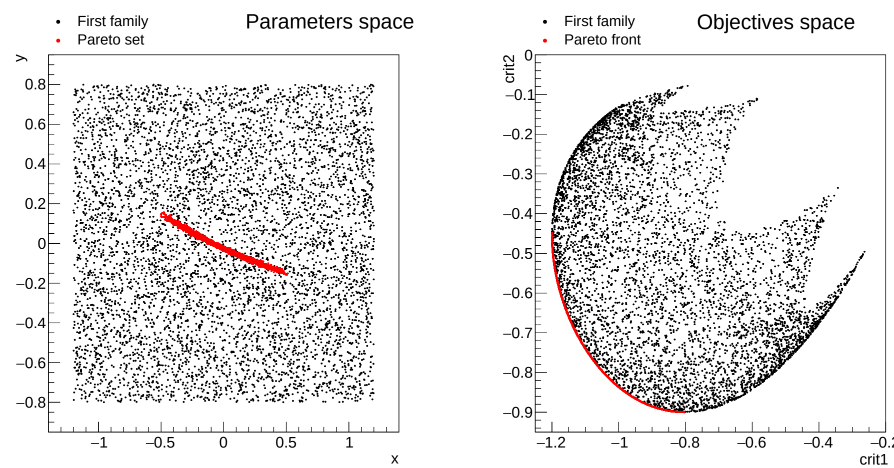
- Single Objective (SO) or Multi Objectives (MO) to be minimised
- parameters that have an impact on objective
- possible constraint on these parameters



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 algorithms)



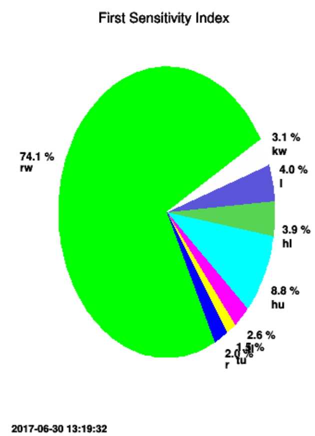


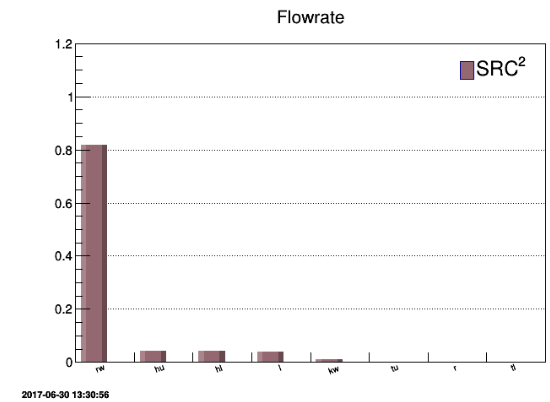
Sensitivity module

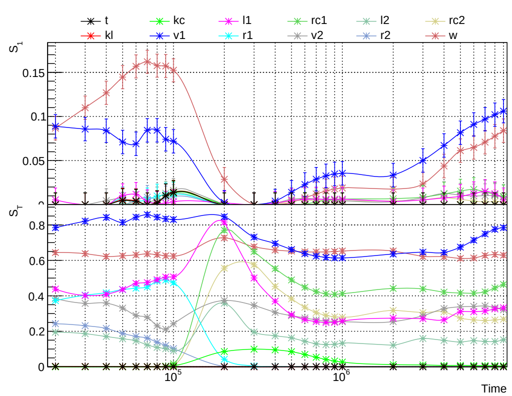
Tools to evaluate the sensitivity of the outputs of a code/function to its inputs.

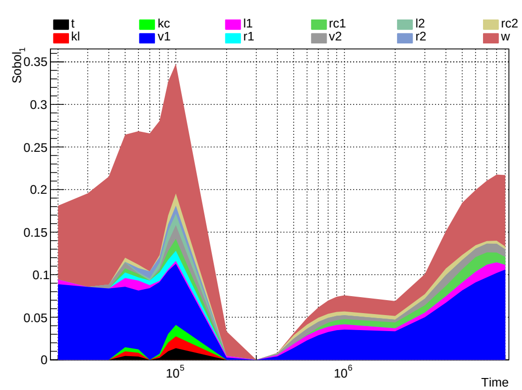
Several kinds of methods available:

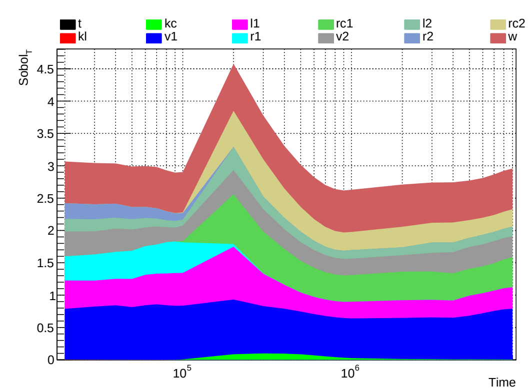
- Local: finite differences $\left(\frac{\delta Y_i}{\delta X_j}(x_0)\right)$
- Regression:
  - Pearson (values)
  - Spearman (ranks)
- Screening: OAT, Morris...
- Sobol indexes:
  - FAST (Fourier Amplitude Sensitivity Test)
  - RBD (Random Balance Design)
  - Sobol/Saltelli Methods











DE LA RECHERCHE À L'INDUSTRIE

Eyes-on: a simple example

emacs@jbpc

File Edit Options Buffers Tools C++ Help

Save Undo

```
void OneKrigingOnly()
{
  /*Reading a database (y vs x1) from a simple function.
  Informations are stored in the TDataServer object*/
  TDataServer *tdsObs = new TDataServer("tdsObs","observations");
  tdsObs->fileDataRead("utf-1D-train.dat");

  /*Defining a kriging model with the training database
   by defining a certain number of options */
  TGPBuilder *gpb = new TGPBuilder(tdsObs,"x1","y","maternII");
  //Find the best possible parameters by optimisation
  gpb->findOptimalParameters("ML", 20, "BFGS", 100);
  //Build the best obtained kriging model.
  TKriging *kg = gpb->buildGP();

  /*Reading now a test basis (constructed with the same dummy function)
  This is mostly for x-check and illustration purposes*/
  TDataServer *tdsEstim = new TDataServer("tdstest","base de test");
  tdsEstim->fileDataRead("utf-1D-test.dat");

  //Applying the kriging on the test basis => launching the model on every points
  TLauncher2 *lkrig = new TLauncher2(tdsEstim, kg, "x1", "yEstim:vEstim");
  lkrig->solverLoop();

  //Plotting the results
  gROOT->LoadMacro("PlottingKriging.C");
  PlottingKriging(tdsObs, tdsEstim);

}
```

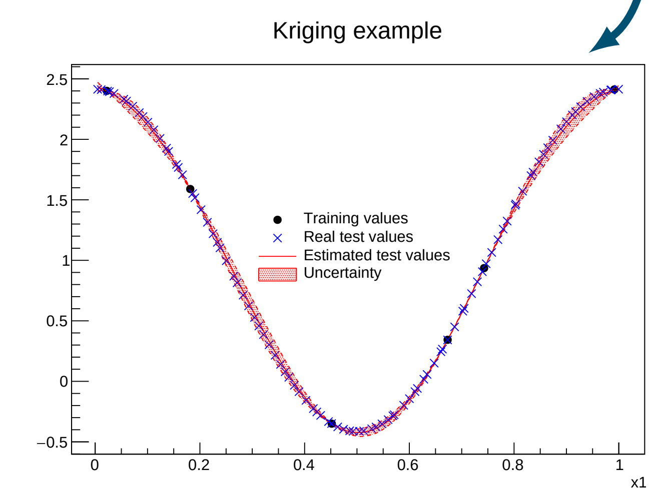
```
-:---  OneKrigingOnly.C    All L1     (C++/l Abbrev)
Loading cc-langs...done
```

called by

```
(mar.10mai 0:37)-(blanchard@jbpc:...ipts/modeler)-(1|%) root OneKrigingOnly.C
root [0]
Processing /home/blanchard/.root_logon...
Processing OneKrigingOnly.C...
--- Uranie v3.7/0 --- Developed with ROOT (5.34/23) by Fabrice Gaudier
                    Copyright (C) 2013  CEA/DEN
                    Version : v3.7/0 -   Date : Thu Jul 23, 2015
                    All rights reserved, please read http://root.cern.ch/

TGPBuilder::findOptimalParameters: starting screening procedure (20 evaluations)
TGPBuilder::findOptimalParameters: starting optimisation procedure (BFGS algorithm
!,,,,;,,,,!,,,,
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [2]
```

gives



Plans for the future

Technical improvements

- Parallelise the EGO estimation
- Porting more methods on GPU (kNN and ANN so far)
- Move to ROOT v6, to get the new C++ on the flight-compiler

Methodological improvements

- Combine Hamiltonian Markov-chain and ANN
- Get new sensitivity indexes (Shapeley)
- Bayesian calibration (through MCMC algorithms in non linear settings)
- Test and 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 (submitted to CPC):
http://arxiv.org/abs/1803.10656

Commissariat à l'énergie atomique et aux énergies alternatives
Centre de Saclay | 91191 Gif-sur-Yvette Cedex
T. +33 (0)1 69 08 73 20 | F. +33 (0)1 69 08 68 86

Direction de l'énergie nucléaire
Département de modélisation des systèmes et structures
Service de Thermohydraulique et de mécanique des fluides

Etablissement public à caractère industriel et commercial | R.C.S Paris B 775 685 019