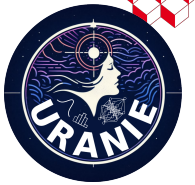


General overview of the Uranie platform

ExaMA general assembly, Aix en Provence, 20th of January, 2026

Rudy Chocat. For support please use `support-uranie@cea.fr`
CEA DES/ISAS/DM2S/SGLS/LIAD

Support designed by Jean-Baptiste Blanchard, Rudy Chocat and Gabriel Sarazin.

The Uranie (LIAD) Task Force®


Gilles Arnaud*


Jean-Baptiste Blanchard


Guillaume Damblin


Geoffrey Daniel


Rudy Chocat*


Clément Gauchy


Gauthier Fauchet


Aurore Lomet


Gabriel Sarazin*


Riccardo Finotello


Julien Nespoulous*


Inna Kucher

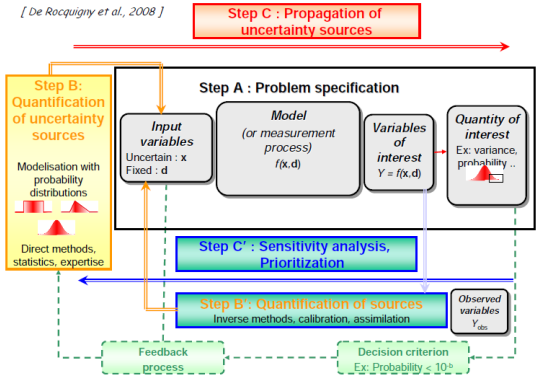

Salomon Chung*

Main missions of the UTF

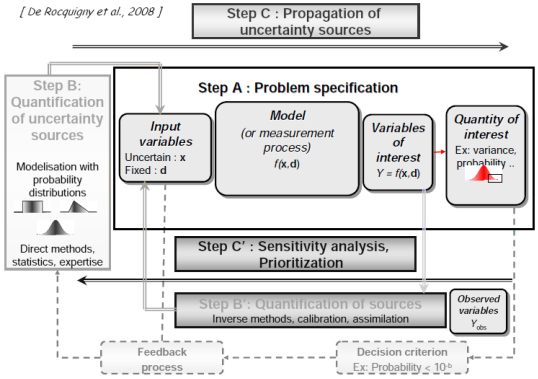
1. Develop methodologies to help solving problems related to uncertainties: **propagation and quantification of uncertainties, design and analysis of computer experiments, surrogate modeling, sensitivity analysis, parameter calibration, optimization,...**
2. Promotion of all the developed methods by **implementing them into the Uranie platform** and testing them on classic examples.
3. Dissemination of all new features by upgrading the documentation, providing regular training sessions and offering active support to all requesting research teams.
4. Upstream discussion with physicists when setting up their research projects to verify that their needs are covered by the URANIE scope, and return to step 1 otherwise.

Please contact support-uranie@cea.fr

Bird's eye view on the UQ methodology



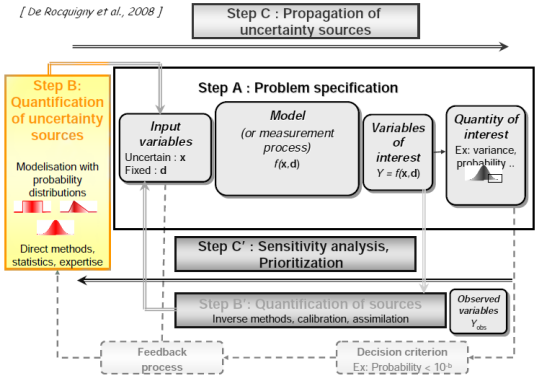
Bird's eye view on the UQ methodology



Main steps:

- **A: Problem specification.**
  - → Inventory of uncertain input variables.
  - → Choice of the output/quantity of interest.
  - → Identification of technical deadlocks.

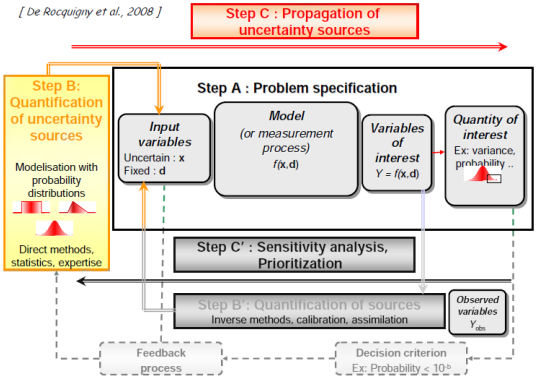
Bird's eye view on the UQ methodology



Main steps:

- A: Problem specification.
- **B: Quantification of uncertainty sources.**
  - → Choice of parametric families for each input.
  - → Specification of the dependence structure.
  - → Construction of the joint input distribution.

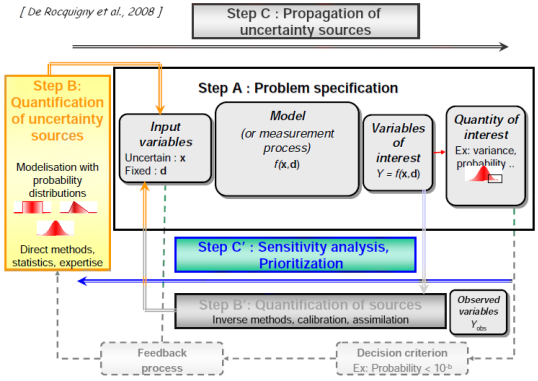
Bird's eye view on the UQ methodology



Main steps:

- A: Problem specification.
- B: Quantification of uncertainty sources.
- **C: Propagation of uncertainty sources.**
  - → Construction of a well-adapted DoE (sampling)
  - → Computation of all output values (Monte Carlo).
  - → Statistical expertise to derive robust estimates.

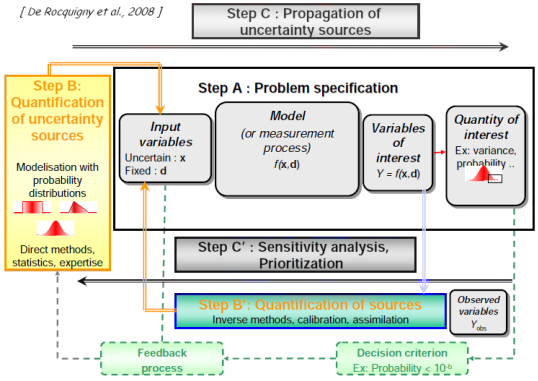
Bird's eye view on the UQ methodology



Main steps:

- A: Problem specification.
- B: Quantification of uncertainty sources.
- C: Propagation of uncertainty sources.
- **C': Sensitivity analysis.**
  - → Estimation of sensitivity measures.
  - → Variance-based methods for ranking.
  - → Kernel-based methods for screening.

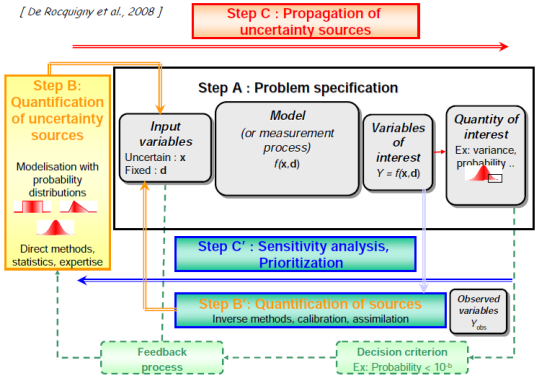
Bird's eye view on the UQ methodology



Main steps:

- A: Problem specification.
- B: Quantification of uncertainty sources.
- C: Propagation of uncertainty sources.
- C': Sensitivity analysis.
- **B': Feedback on the input uncertainties.**
  - → Robustness analysis.
  - → Inversion under uncertainty.
  - → Model calibration based on experimental data.

Bird's eye view on the UQ methodology

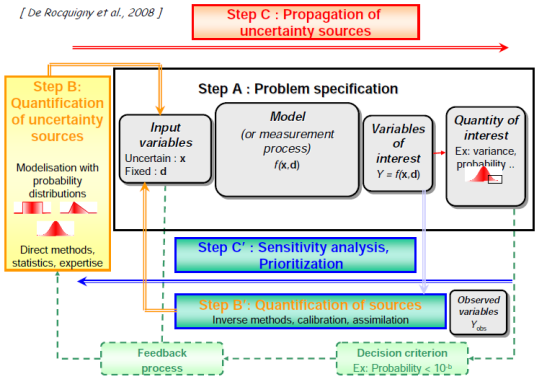


Main steps:

- A: Problem specification.
- B: Quantification of uncertainty sources.
- C: Propagation of uncertainty sources.
- C': Sensitivity analysis.
- B': Feedback on the input uncertainties.
- **A: Back to square one.**

Despite this well-established framework, there is no all-purpose solution. Sometimes, it might be useful to iterate several times before reaching a satisfactory level of uncertainty control.

Bird's eye view on the UQ methodology



Main steps:

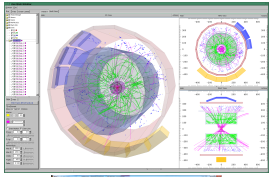
- **A: Problem specification.**
- **B: Quantification of uncertainty sources.**
- **C: Propagation of uncertainty sources.**
- **C': Sensitivity analysis.**
- **B': Feedback on the input uncertainties.**

As highlighted in a recent paper by the UTF, the URANIE platform offers a wide range of highly efficient tools to deal with many difficult problems in statistics and optimization:
https://doi.org/10.1051/epjn/2018050

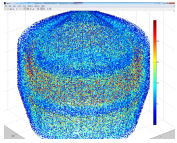
Technical aspects

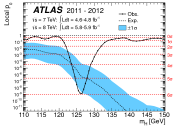
Based on the ROOT platform

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++ on-the-flight compiler and a Python interface (also Ruby)
  - ➙ 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 (`https://root.cern.ch` or on your machine, once installed)
- URANIE tries to follow the ROOT structure

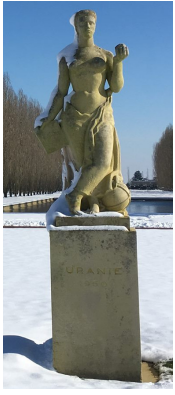




The Uranie platform

Developed at CEA/DES to help colleagues and partners become familiar with: uncertainty quantification, machine learning and optimization.

- **Named after the muse Urania.**
  - Symbol of inspiration and creativity.
- **Written in C++.**
  - Relies heavily on the ROOT framework.
- **Cross-platform software.**
  - Developed on Unix.
  - Tested on Unix and Windows.
- **Simple access to many data formats.**
  - Flat ASCII file, XML, JSON ...
  - TTree (internal ROOT format),
  - SQL database format.
- **Powerful visualization tools.**
  - Based on those provided by ROOT.
- LGPL



- **What URANIE excels at:**
  - Generating designs of experiments.
  - Making best use of parallel computing resources to propagate efficiently uncertainties throuh expensive computer codes.
  - Reliability analysis: estimation of rare-event probabilities or extreme quantiles.
  - Training surrogate models from very few data.
  - Sensitivity analysis (in many different ways).
  - Solving hard optimization problems.
  - Calibrating model parameters from experimental data (Bayesian methods fueled by MCMC simulation).
- **What URANIE cannot do:**
  - Outperform what is offered in scikit-learn, PyTorch, TensorFlow...

General organization: version 4.10

General description:

- ROOT version: 6.32
- 13 modules / $\sim$ 280 classes
  $\sim$ 154 000 lines of code
- Compilation using CMAKE.

Regularly tested:

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

Documentation: 3 different levels

- **Methodological guide** ($\sim$ 90 pages).
- **User manual** ($\sim$ 1050 pages):
  $\sim$ 340 pages: description of methods and their options,
  $\sim$ 350 pages: C++ macros ($\sim$ 130 examples),
  $\sim$ 350 pages: Python macros ($\sim$ 130 examples).
- **Developer guide** using DOXYGEN (HTML only).

Unit Testing Report

	DataServer	Launcher	Relauncher	Sampler	SmartEntry	Optimizer	reOptimizer	Modeler	ExpertModeler	ReLability	XMLTreater
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

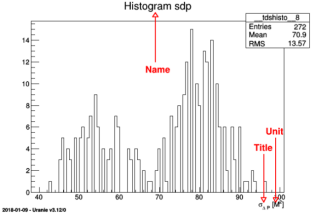
- Name + title: constructor defined from the name and the title of the variable

```
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");
```

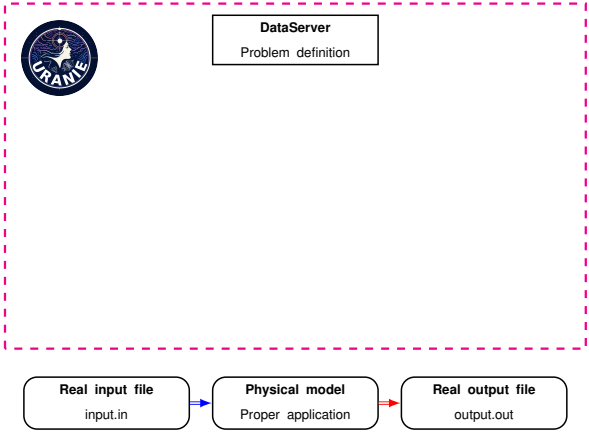
The result of this piece of code is shown in Figure II.3.



General discussion: introducing the concepts

Uranie's approach

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



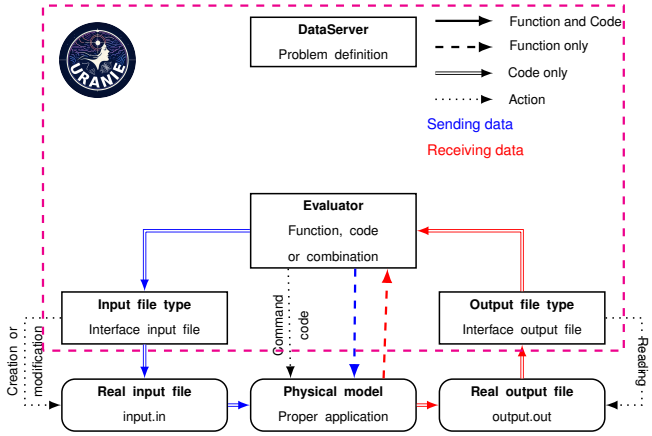
General discussion: introducing the concepts

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



General discussion: introducing the concepts

Uranie's approach

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

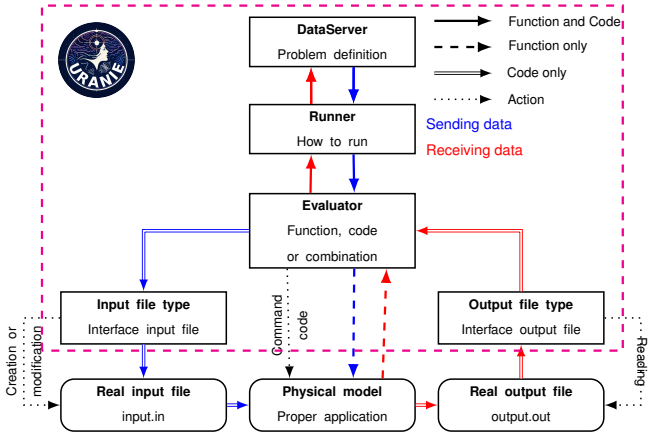
Nature of Evaluators

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

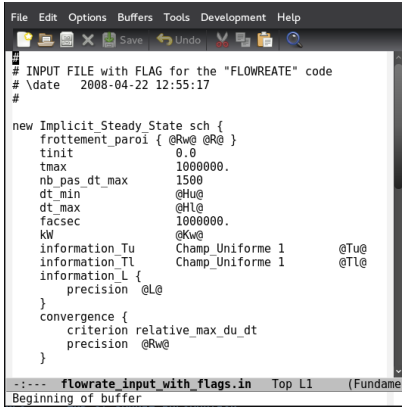
Ways of submitting jobs (Runner)

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

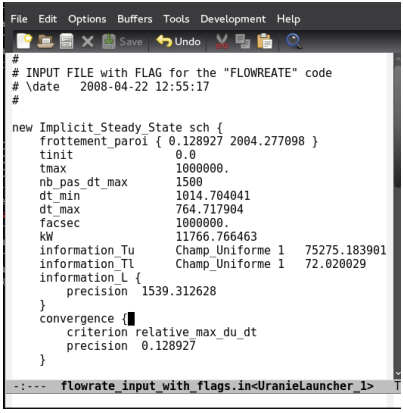
Very large number of use-case in the user manual to cover almost combination of runners/evaluators



Example of flag format



File containing flags

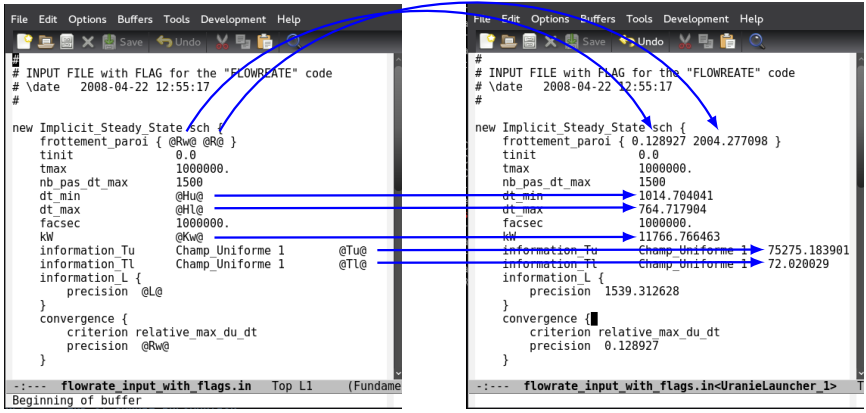


Modified file

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

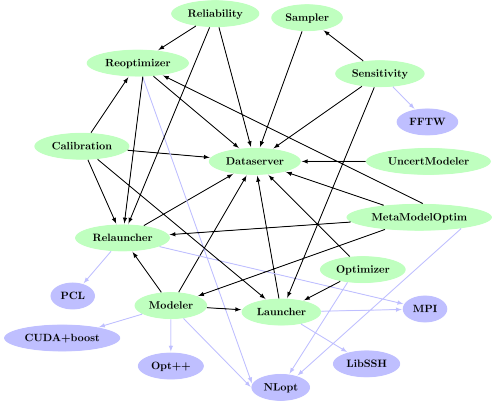
Advantage

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

The platform organization

Built upon interdependent modules.

- **Some modules are intended to build the IT architecture . . .**
  - `DataServer`: storage and manipulation of data.
  - `Launcher`/`ReLauncher`: dialog with computer codes.
- **. . . so that other focus on applied maths.**
  - `Sampler`: generation of designs of experiments.
  - `Modeler`: training of surrogate models.
  - `Optimizer`/`Reoptimizer`: solving of optimization problems.
  - `Sensitivity`: estimation of sensitivity indices.
  - `Reliability`: estimation of rare-event probabilities.
  - `Calibration`: reduction of parameter uncertainty.



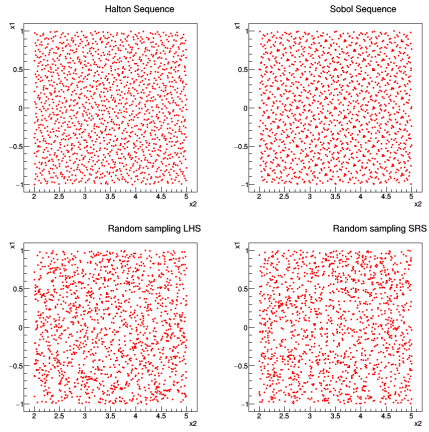
A glimpse at the most iconic modules

The **Sampler** module

Used to generate the design of-experiment (basement of any numerical study).
Some methods are able to deal with correlated input variables.

Two main categories:

- **Stochastic methods:**
  - Simple Random Sampling (SRS),
  - Latin Hypercube Sampling (LHS),
  - One-At-a-Time (OAT) sampling,
  - Elliptical and Archimedean copulas,
  - Random fields ...
- **Deterministic designs:**
  - Regular quasi Monte-Carlo (QMC): Halton / Sobol' sequences,
  - Sparse grid sampling: Petras / Smolyak quadrature rules,
  - Space-filling designs: optimized LHS.



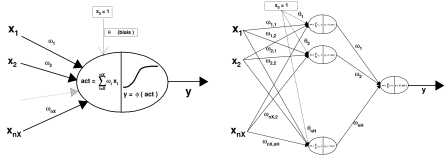
The **Modeler** module

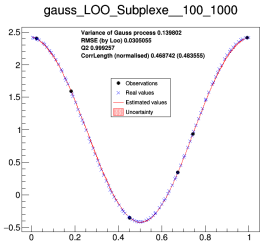
Construction of a surrogate model / metamodel / response surface / emulator.
Model selection, training (hyperparameter optimization), goodness-of-fit, validation.

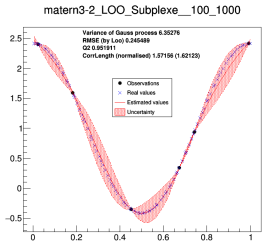
Models for supervised learning:

- Generalized linear models,
- Polynomial regression,
- k-nearest neighbors (kNN),
- Gaussian process regression (Kriging),
- Polynomial chaos expension (ANISP),
- Artificial neural networks (especially MLPs).

➙ Models can be exported in several formats (C++, Fortran, PMML) to be re-used later on.







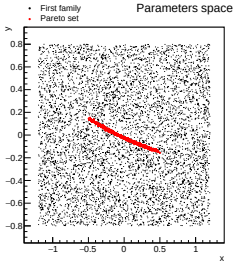
The **Optimizer** module

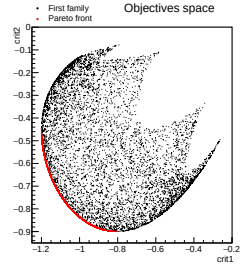
Dealing with optimisation problem usually means:

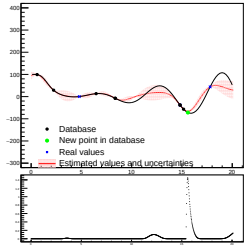
- One criterion (or several criteria) to minimize.
  - ➙ Single-objective (SO) vs. multi-objective (MO) optim.
- Parameters with significant impact on the objectives.
- Possible constraints on the input space.

Many possible implementations:

- Minuit: ROOT library for unconstrained SO optim.
- Opt++: SO optim. library with /without constraints.
- NLopt: SO optim. library with / without constraints.
- Vizir: MO optim. library with / without constraints.
  - ➙ Developed by Gilles Arnaud.
  - ➙ Stochastic methods (genetic algorithms, PSO, CMAES...).
- EGO: Gaussian process based optimization





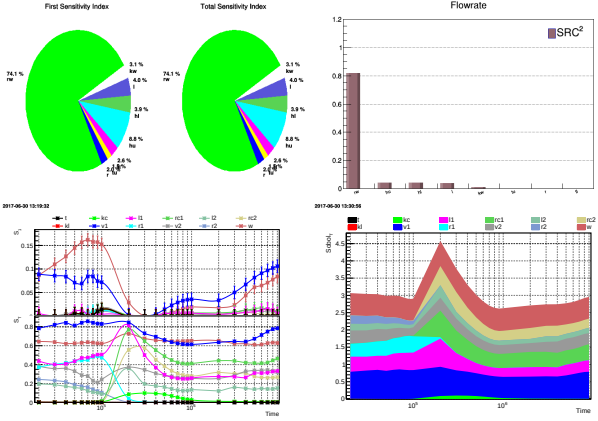


The **Sensitivity** module

Tools to measure how much influence each input variable has on the output distribution.

Many differents strategies:

- **Specific DOE** or **given data**
- **The old way.**
  - → Partial derivatives.
  - → The Morris method.
- **Regression-based methods.**
  - → SRC indices (for linear models).
- **Variance-based methods.**
  - → Sobol' indices (many schemes).
- **Kernel-based methods.**
  - → HSIC indices + p-values.
- **Probability-based methods.**
  - → Cramer Von Mises indices
- **Correlation issues.**
  - → Johnson's relative weights instead of SRC.
  - → Shapley values instead of Sobol'.



URANIE @ CEA

Availability

Software installation instructions:

- All the information you need is available here: `https://gitlab.com/uranie-cea/publication/-/wikis/home`

Binary packages are also available:

- Please click here.

Other installations:

- A conda package can be installed thanks to miniconda (Unix) but a new conda environment has to be created (because ROOT seeks to avoid conflicts).
- **For installation on Windows, local admin rights elevation is often required!**
- For further details, please click here.

Tuleap forge for bug tracking, wiki, doc...

To get account (or any other business): support-uranie@cea.fr

Become a user

How to use it?

We strongly recommand using URANIE through the Python interface:

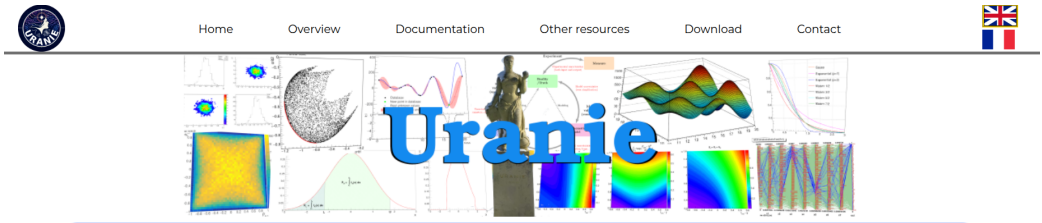
- Based on PyROOT (from ROOT).
- Please check the section "XIV.2. Macros Python HowTo" in the Python version of the user manual.
- Two approaches can be considered:
  - Incorporate all the the imports in a `rootlogon.py` file (available in the macros directory).
  - Import all necessary modules after importing ROOT.
- To run a Python function, use the `TPythonEval` object from the `Relauncher` module.

If you have any question or remark about URANIE, and more generally about uncertainty management or optimization, feel free to come back to us.

Please contact support-uranie@cea.fr

Hot news: our website

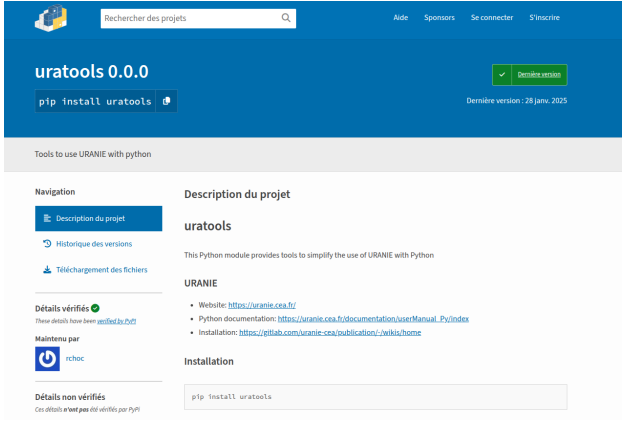
https://uranie.cea.fr/



It's not over, another hot news : **uratools**

Creation of the `Python` library **uratools** to enhance the user experience of `URANIE` in `Python` using pip (`https://pypi.org/project/uratools/`)

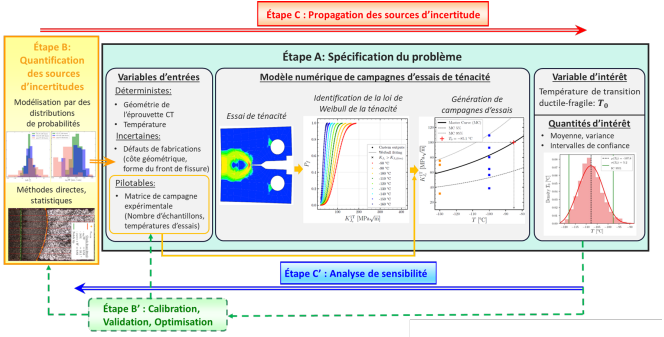
- Version 0.0.0 to simplify handling of Numpy objects from Uranie
- Many updates are coming



Some realizations with URANIE (1/3)

Uncertainty quantification of fracture toughness tests

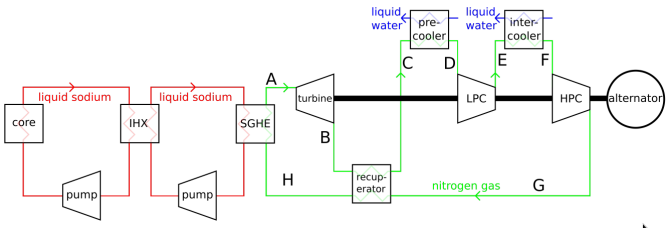
- PhD work of A. Quintin
- Numerical model based on `CAST3M`, bash script and Python ($\approx$ 1h for 3D model)
- Uncertainty Propagation, sampling, sensitivity analysis, surrogate model using `URANIE`
- Simulation of the fracture toughness test of steel tensile test used to characterize the remaining useful life of the vessel of a nuclear reactor

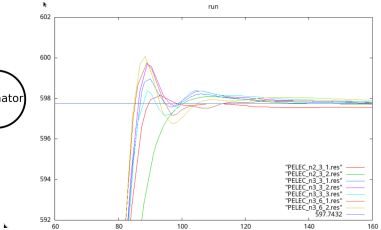


Some realizations with URANIE (2/3)

Optimization of the response of a system dedicated to reactor flexibility

- by G. Arnaud, G. Mauger and G. Fauchet
- Numerical model based on `CATHARE 3` (≈ 15 min)
- Efficient Global Optimization based on kriging to reduce the number of evaluations
- Specific adaptation for distributed jobs using asynchronous approach





Nuclear packaging

- A nuclear package is used to transport and store spent nuclear fuel
- It has to withstand radioactive radiation, mechanical load, heat from the residual power of the spent fuel...

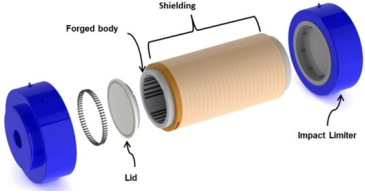
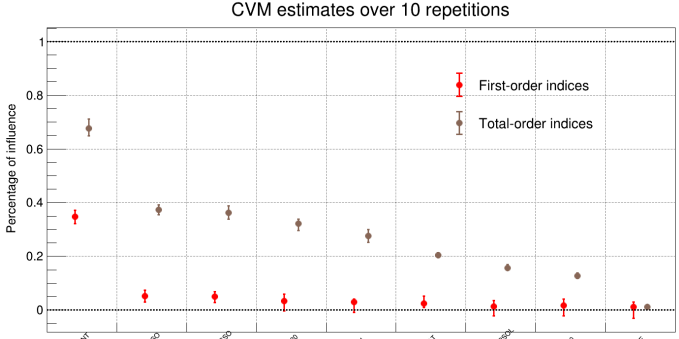


Figure 1: Example of a TN Eagle transport cask for nuclear waste, made by Orano

Some realizations with URANIE (3/3)

Sensitivity analysis of a simple reactive transport model at the package scale

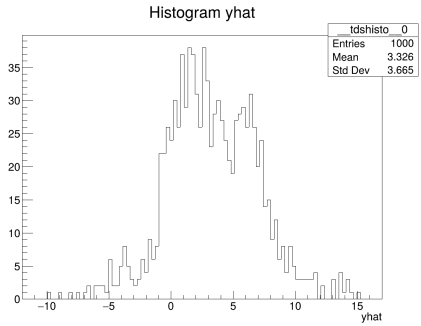
- by G. Sarazin, J.-M. Delaye
- Simulation of the chemical behavior of the nuclear package for nuclear safety studies
- 24 inputs, 3 outputs, fast model evaluations ($\approx 3s$) but outliers
- **Screening** using **HSIC** (U-stats) $\Rightarrow$ 9 random variables
- **Ranking** using **Cramer Von Mises indices** (new indices well adapted for the application)



A small example

- Uncertainty propagation in the Ishigami code

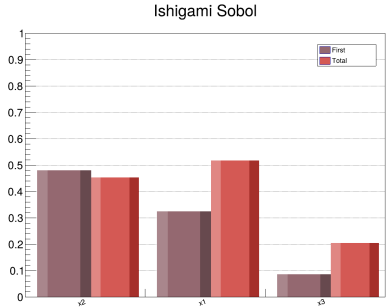
```
import ROOT; pi_ROOT = ROOT.TMath.Pi()
from ROOT.URANIE import DataServer, Sampler, Launcher
### === Definition of the problem in the DataServer ===
tds = DataServer.TDataServer()
tds.addAttribute(DataServer.TUniformDistribution("x1",-pi_ROOT,pi_ROOT))
tds.addAttribute(DataServer.TUniformDistribution("x2",-pi_ROOT,pi_ROOT))
tds.addAttribute(DataServer.TUniformDistribution("x3",-pi_ROOT,pi_ROOT))
### ===Coupling URANIE with the Ishigami code ===
sFileName=ROOT.gSystem.Getenv("PWD")+"/data/ishigami_input_with_flags.in";
tds.getAttribute("x1").setFileFlag(sFileName, "@x1@");
tds.getAttribute("x2").setFileFlag(sFileName, "@x2@");
tds.getAttribute("x3").setFileFlag(sFileName, "@x3@");
fout = Launcher.TOutputFileKey("_output_ishigami_withKey_.dat");
fout.addAttribute(DataServer.TAttribute("yhat"));
mycode = Launcher.TCode(tds, "ishigami -f >>/dev/null");
mycode.addOutputFile( fout )
### === Fill the dataserver by sampling ===
nS = 1000
sam = Sampler.TSampling(tds, "srs", nS)
sam.generateSample()
### === Propagation by evaluating the code ===
tlch = Launcher.TLauncher(tds, mycode)
tlch.run()
### === Plot the results ===
tds.draw("yhat")
```



A small example

- Sensitivity analysis of the Ishigami code

```
import ROOT; pi_ROOT = ROOT.TMath.Pi()
from ROOT.URANIE import DataServer, Sensitivity, Launcher
### === Definition of the problem in the DataServer ===
tds = DataServer.TDataServer()
tds.addAttribute(DataServer.TUniformDistribution("x1",-pi_ROOT,pi_ROOT))
tds.addAttribute(DataServer.TUniformDistribution("x2",-pi_ROOT,pi_ROOT))
tds.addAttribute(DataServer.TUniformDistribution("x3",-pi_ROOT,pi_ROOT))
### === Coupling URANIE with the Ishigami code ===
sFileName=ROOT.gSystem.Getenv("PWD")+"/data/ishigami_input_with_flags.in";
tds.getAttribute("x1").setFileFlag(sFileName, "@x1@");
tds.getAttribute("x2").setFileFlag(sFileName, "@x2@");
tds.getAttribute("x3").setFileFlag(sFileName, "@x3@");
fout = Launcher.TOutputFileKey("_output_ishigami_withKey_.dat");
fout.addAttribute(DataServer.TAttribute("yhat"));
mycode = Launcher.TCode(tds, "ishigami -f >>/dev/null");
mycode.addOutputFile( fout )
### === Sensitivity analysis by evaluating the code ===
nS=2000
tsobol = Sensitivity.TSobol(tds, mycode, nS)
tsobol.computeIndexes()
### === Plot the results ===
tsobol.drawIndexes("Ishigami Sobol","","all,hist,nonewcanv");
```



Thanks! Any questions?