DE LA RECHERCHE À L'INDUSTRIE

cea

The Uranie platform

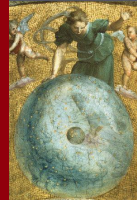
J-B. Blanchard, F. Gaudier, J-M. Martinez, G. Arnaud

jean-baptiste.blanchard@cea.fr

SIAM-UQ 2018 – 2018/04/18 –

www.cea.fr

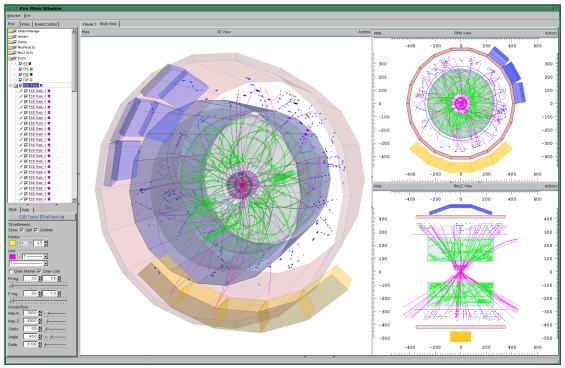
Outline

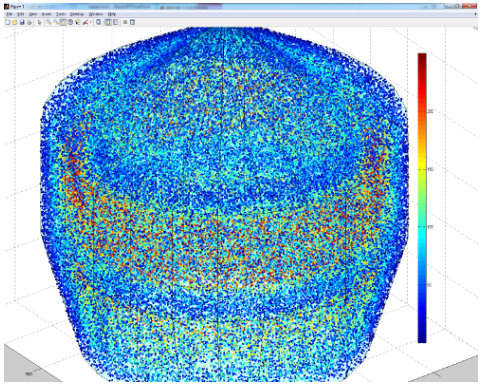


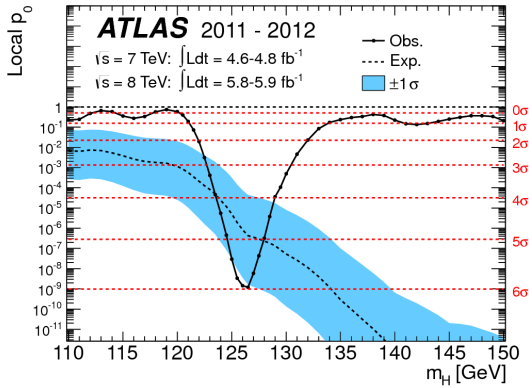
The ROOT platform

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 High energy physics)
  - statistical analysis tools (*RooStats*, *RooFit*...)
  - and many more (3D object modelling, distributed computing interface...)
- LGPL







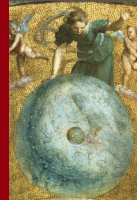
The Uranie platform

Developed at CEA/DEN 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

The Uranie project

General organisation: version 3.12 (1/2)



General description:

- ROOT version: 5.34.36
- 11 modules / 246 classes
  $\sim$ 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
- $\sim$ 1500 unitary tests with CPPUNIT
- $\sim$ 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 ($\sim$ 60 pages)
- User manual: $\sim$ 550 pages
  $\sim$ 250 pages: describing methods and their options.
  $\sim$ 250 pages: use-case macros ($\sim$ 100 examples)

Developer's guide using DOXYGEN (HTML only)

- describing methods from comments in the code

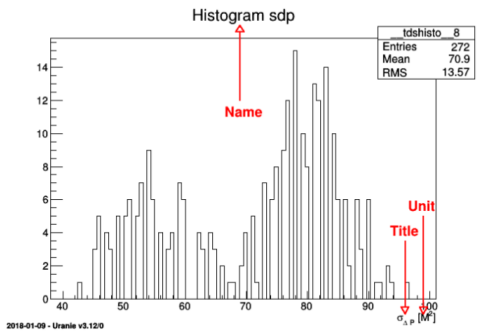
- Name + title: constructor defined from the name and the title of the variable

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

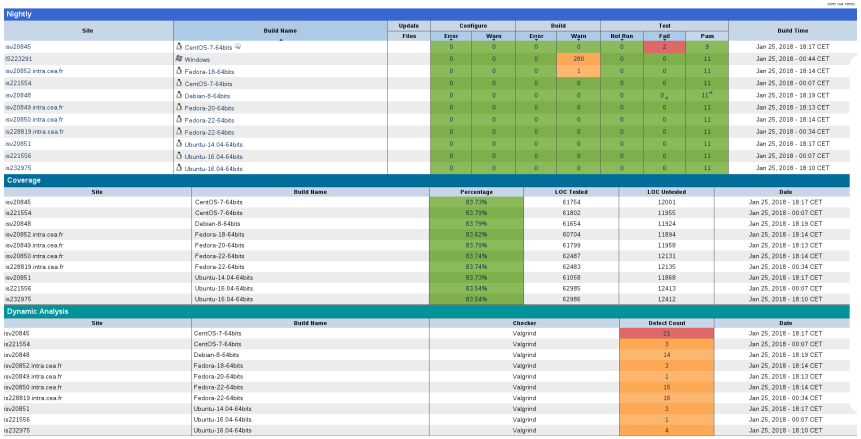
A pointer **psdp** to a variable **"sdp"** is available with title being *#sigma_{#Delta P}*. The command **setUnity()** precises the unit. In this case, by default, the field *key* is identical to the field *name*. We will use the ability given by ROOT to write LaTeX expressions in graphics to improve 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



General organisation: version 3.12 (2/2)



Nightly

Site	Build Name	Update Files	Configure Error	Configure Warn	Build Error	Build Warn	Test Not Run	Test Fail	Test Pass	Build Time
isv20845	CentOS-7-64bits		0	0	0	0	0	0	9	Jan 25, 2018 - 18:17 CET
IS223291	Windows		0	0	0	280	0	0	11	Jan 25, 2018 - 00:44 CET
isv20852.intra.cea.fr	Fedora-18-64bits		0	0	0	1	0	0	11	Jan 25, 2018 - 18:14 CET
is221554	CentOS-7-64bits		0	0	0	0	0	0	11	Jan 25, 2018 - 00:07 CET
isv20848	Debian-8-64bits		0	0	0	0	0	0	11	Jan 25, 2018 - 18:19 CET
isv20850.intra.cea.fr	Fedora-22-64bits		0	0	0	0	0	0	11	Jan 25, 2018 - 18:13 CET
is228819.intra.cea.fr	Fedora-22-64bits		0	0	0	0	0	0	11	Jan 25, 2018 - 18:14 CET
isv20851	Ubuntu-14.04-64bits		0	0	0	0	0	0	11	Jan 25, 2018 - 00:34 CET
is221556	Ubuntu-16.04-64bits		0	0	0	0	0	0	11	Jan 25, 2018 - 18:17 CET
is232975	Ubuntu-16.04-64bits		0	0	0	0	0	0	11	Jan 25, 2018 - 00:07 CET

Coverage

Site	Build Name	Percentage	LOC Tested	LOC Unteseted	Date
isv20845	CentOS-7-64bits	83.79%	61754	12001	Jan 25, 2018 - 18:17 CET
is221554	CentOS-7-64bits	83.79%	61802	11955	Jan 25, 2018 - 00:07 CET
isv20848	Debian-8-64bits	83.79%	61654	11924	Jan 25, 2018 - 18:19 CET
isv20852.intra.cea.fr	Fedora-18-64bits	83.62%	60704	11894	Jan 25, 2018 - 18:14 CET
isv20849.intra.cea.fr	Fedora-20-64bits	83.79%	61799	11958	Jan 25, 2018 - 18:13 CET
isv20850.intra.cea.fr	Fedora-22-64bits	83.74%	62487	12131	Jan 25, 2018 - 18:14 CET
is228819.intra.cea.fr	Fedora-22-64bits	83.74%	62483	12135	Jan 25, 2018 - 00:34 CET
isv20851	Ubuntu-14.04-64bits	83.73%	61058	11868	Jan 25, 2018 - 18:17 CET
is221556	Ubuntu-16.04-64bits	83.54%	62985	12413	Jan 25, 2018 - 00:07 CET
is232975	Ubuntu-16.04-64bits	83.54%	62986	12412	Jan 25, 2018 - 18:10 CET

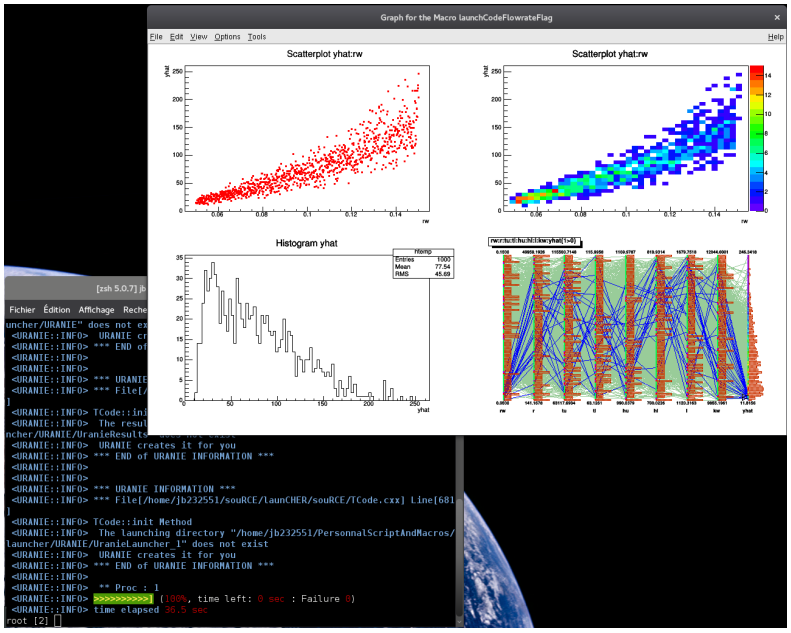
Dynamic Analysis

Site	Build Name	Checker	Defect Count	Date
isv20845	CentOS-7-64bits	Valgrind	21	Jan 25, 2018 - 18:17 CET
is221554	CentOS-7-64bits	Valgrind	3	Jan 25, 2018 - 00:07 CET
isv20848	Debian-8-64bits	Valgrind	14	Jan 25, 2018 - 18:19 CET
isv20852.intra.cea.fr	Fedora-18-64bits	Valgrind	3	Jan 25, 2018 - 18:14 CET
isv20849.intra.cea.fr	Fedora-20-64bits	Valgrind	1	Jan 25, 2018 - 18:13 CET
isv20850.intra.cea.fr	Fedora-22-64bits	Valgrind	15	Jan 25, 2018 - 18:14 CET
is228819.intra.cea.fr	Fedora-22-64bits	Valgrind	16	Jan 25, 2018 - 00:34 CET
isv20851	Ubuntu-14.04-64bits	Valgrind	3	Jan 25, 2018 - 18:17 CET
is221556	Ubuntu-16.04-64bits	Valgrind	1	Jan 25, 2018 - 00:07 CET
is232975	Ubuntu-16.04-64bits	Valgrind	4	Jan 25, 2018 - 18:10 CET

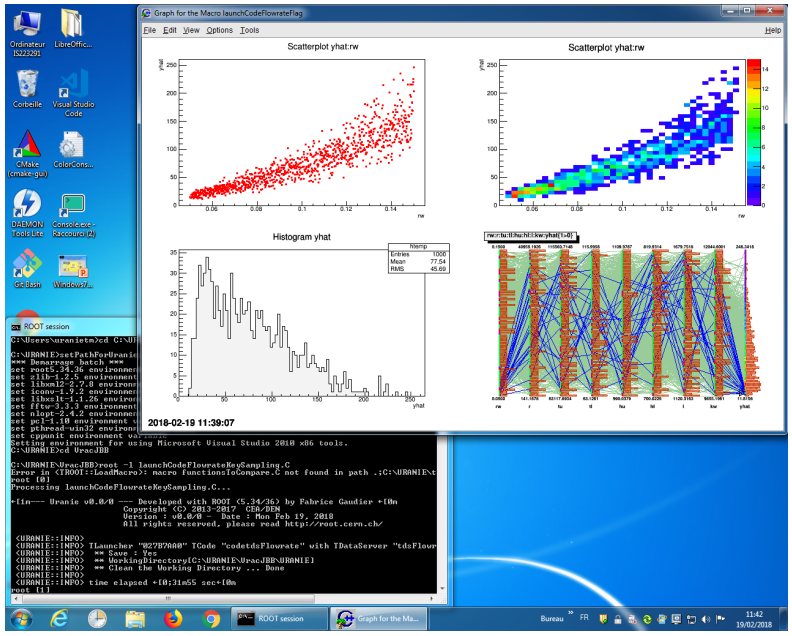
Developed in C++ on Linux, but

- Can be compiled on Windows as well
  - ➙ We provide (on-demand) a self-consistent binary archive to be put anywhere one needs (recommended).
- Very few “#ifdef WIN32”
  - ➙ Same macro can be run both on Linux and Windows
- Every macro in C++ can be written in PYTHON as well

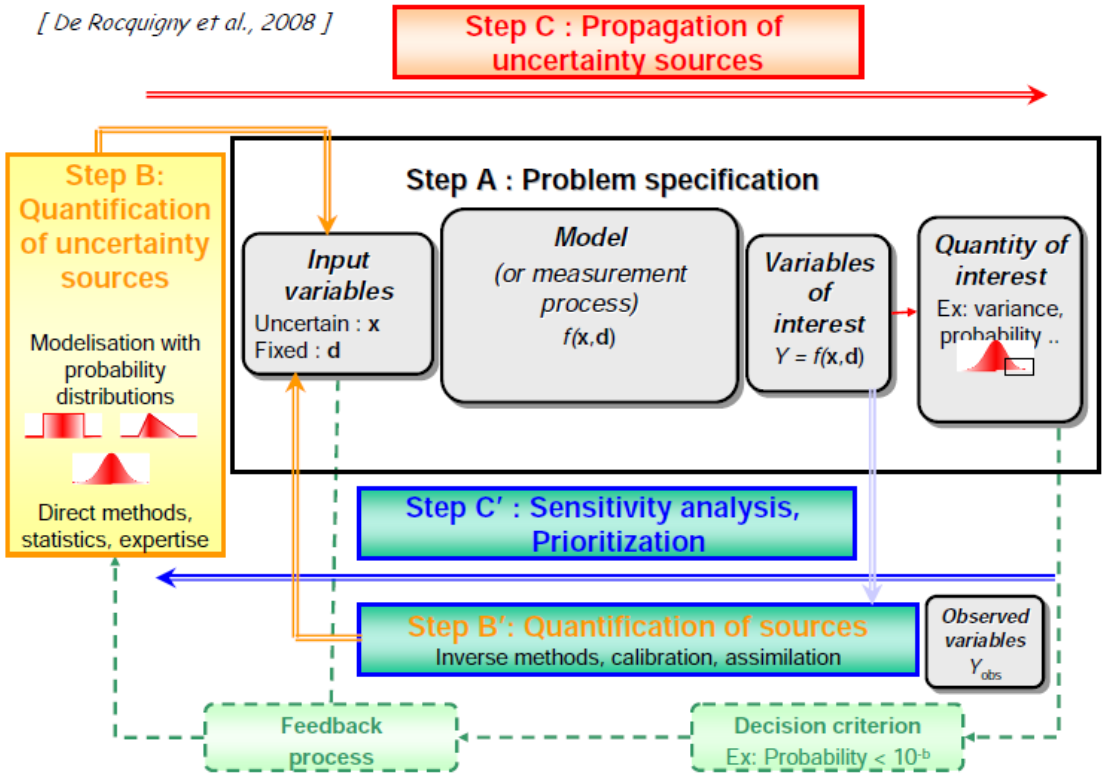
Linux



Windows



Workflow: breakdown into steps

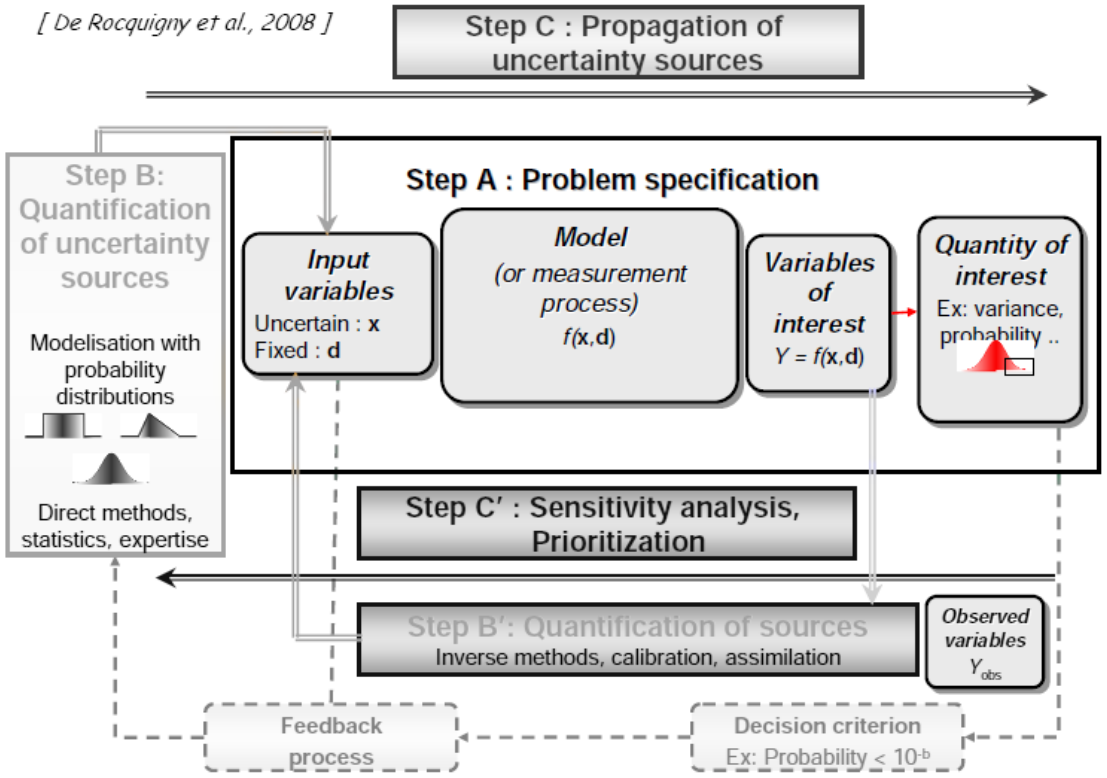


Main steps:

- 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

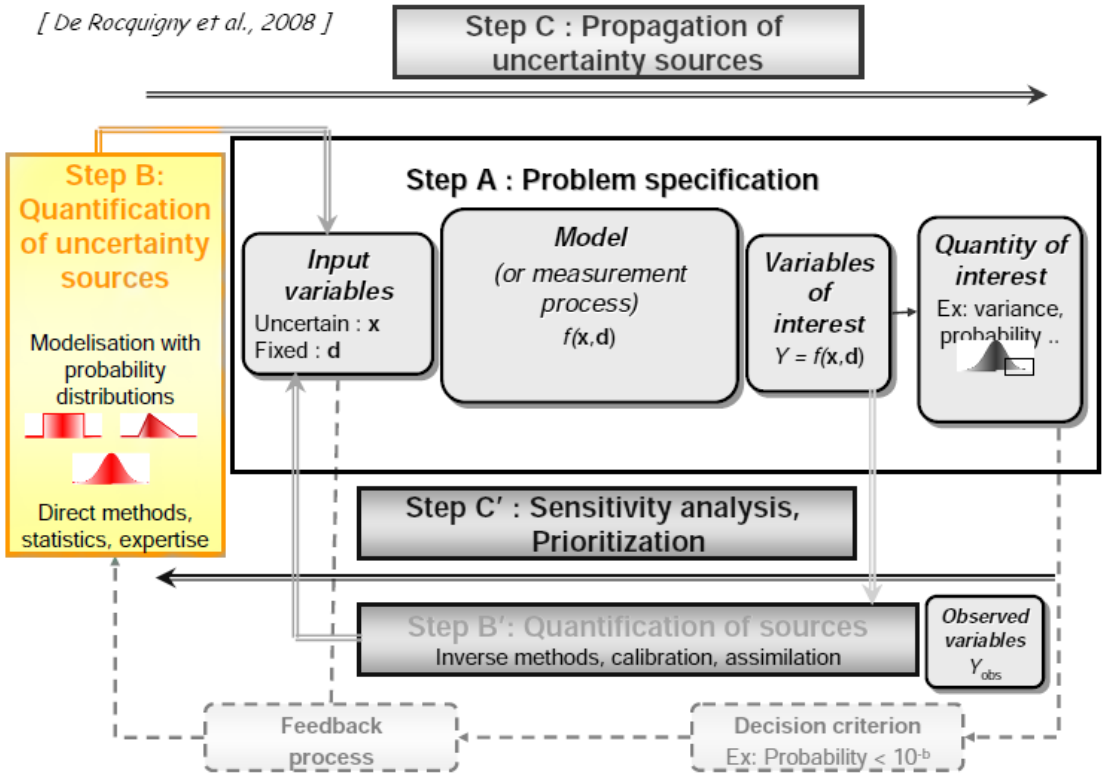


Main steps:

- 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

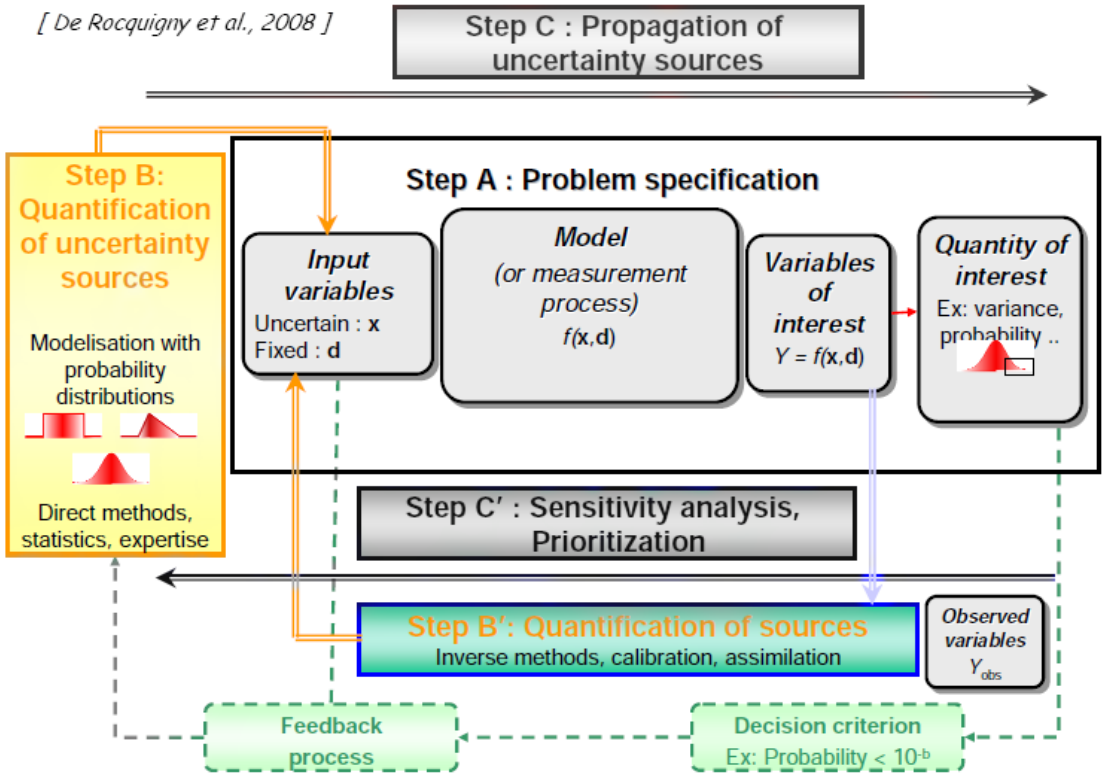


Main steps:

- 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

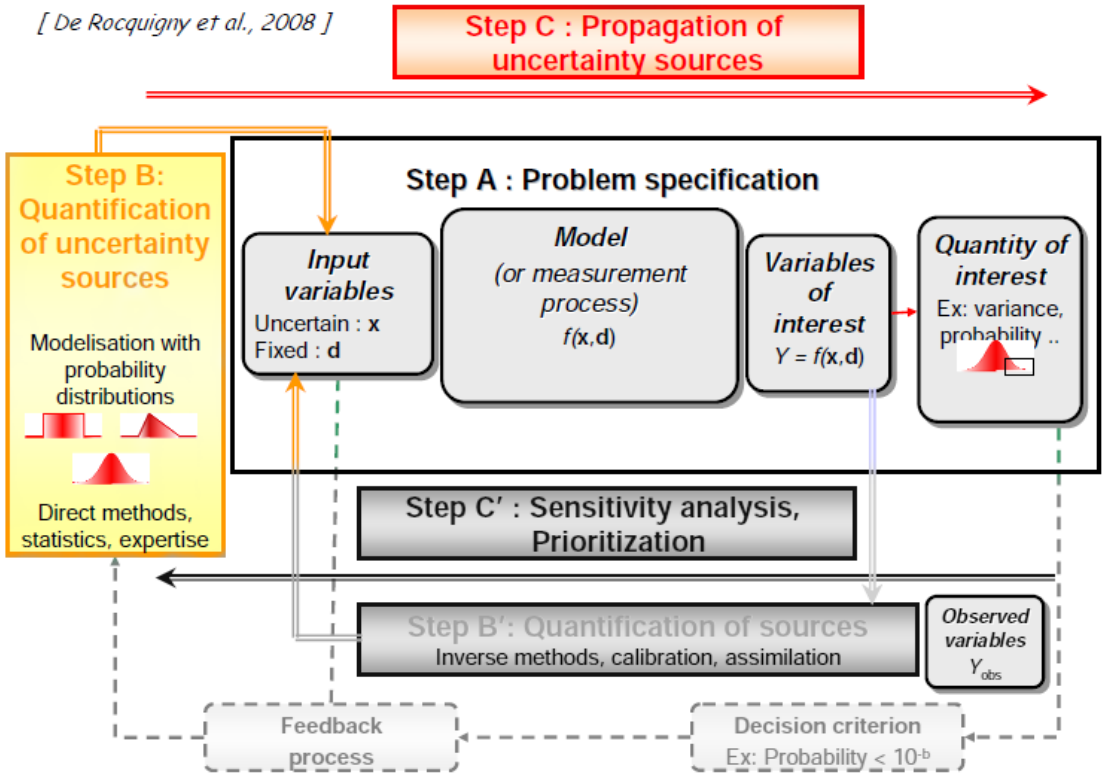


Main steps:

- 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

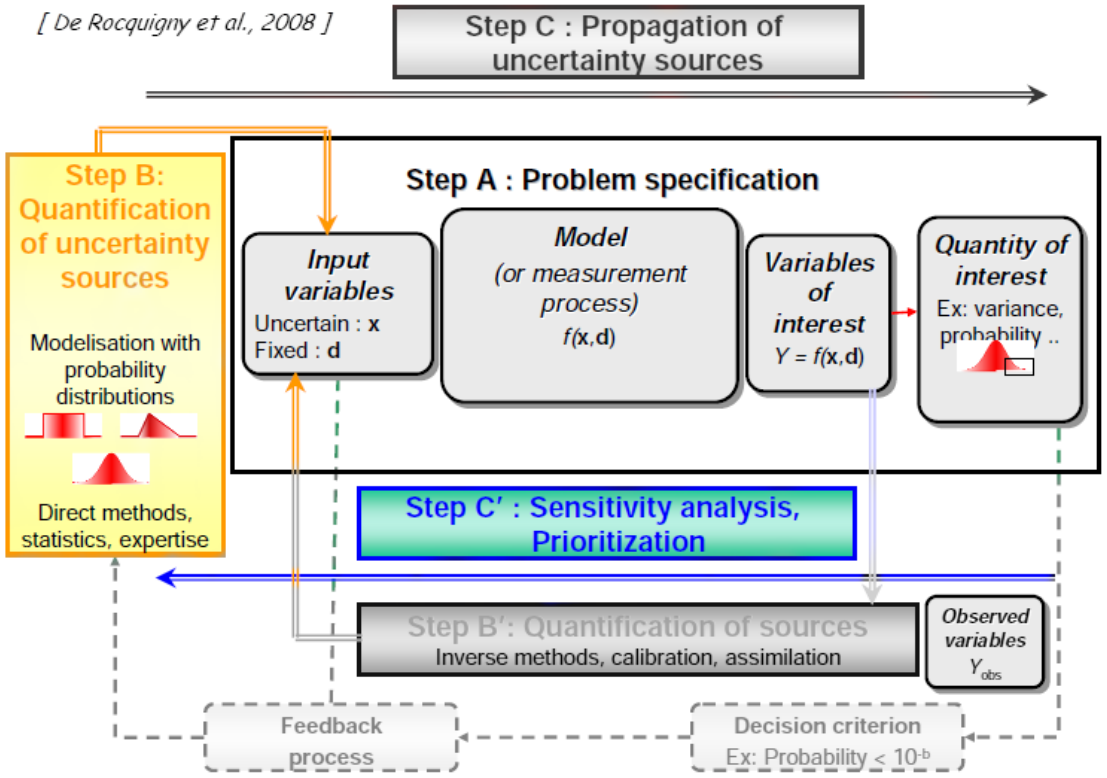


Main steps:

- 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

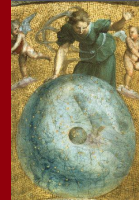


Main steps:

- 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



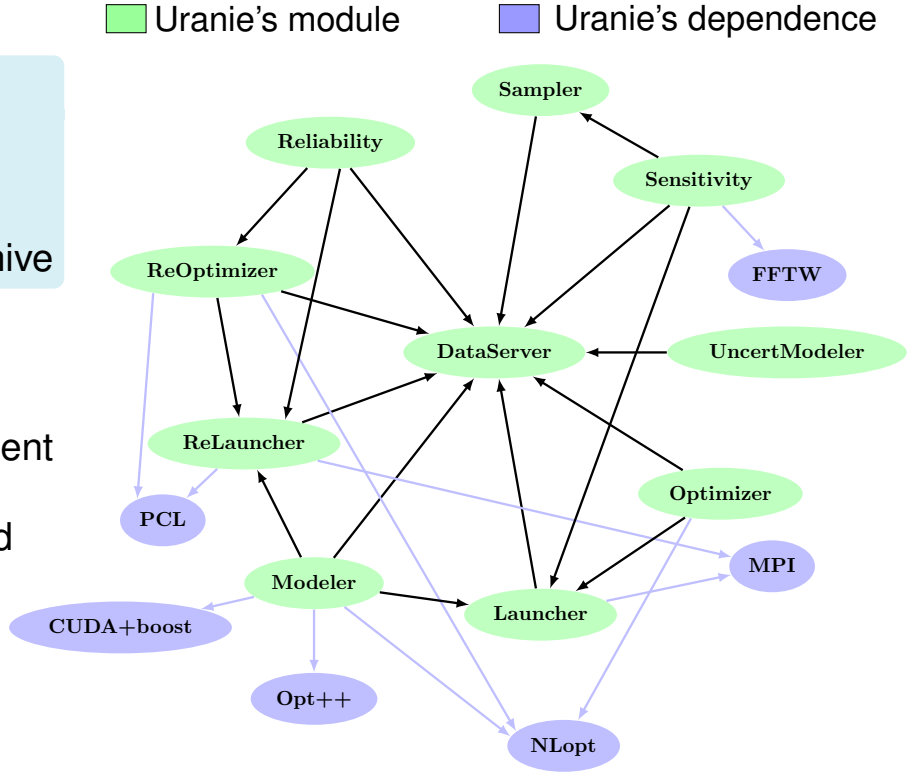
Few dependencies:

- 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:

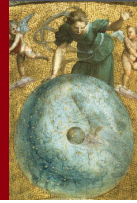
- Some are more technical ones:
  - DataServer: data handling and first statistical treatment
  - (Re)Launcher: interfaces to code/function handling. Can deal with code, PYTHON-function, C++-interpreted and compiled functions
- Many are dedicated ones:
  - Sampler: creation of design-of-experiments
  - 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

Tools for interoperability

Submitting code computations



Launching functions:

- Analytic C++ functions: `myFunction (double *x, double *y)`
  - → inputs/outputs are double-precision.
- Analytic PYTHON functions and compiled C++ functions
  - → inputs/outputs are double-precision, strings or varying-size vectors of double.

Launching external codes (considering them as black boxes):

→ inputs/outputs are double-precision, strings or varying-size vectors of double.

Non-intrusive approach: communication is done through input/output file with many possible formats

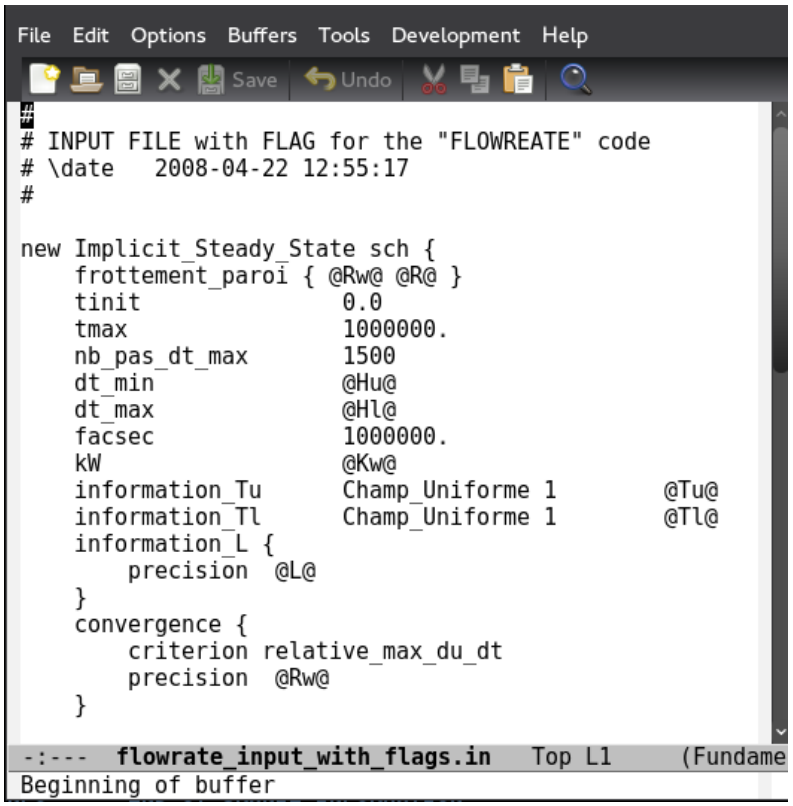
- line format: every input/output has its own line
- column format: every input/output has its own column
- XML format:
- key=value: every input/output has its own key and corresponding value is separated by "="
- flag format: input file is modified to put specific flags in the text ("@rw@" in next slide)

→ Can specify boundaries (vectors and string) and delimiters for two elements (vectors).

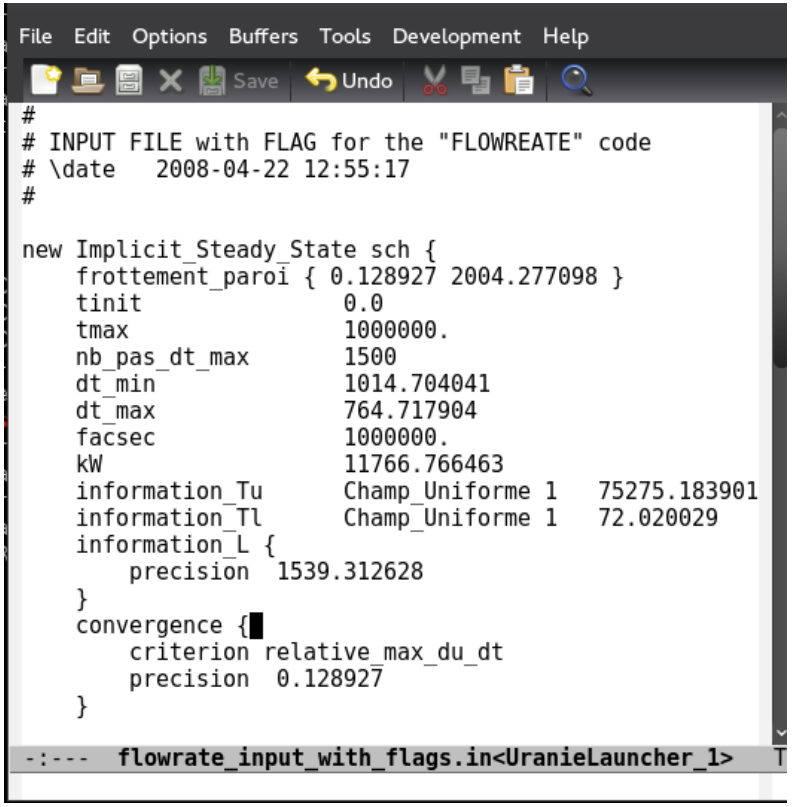
Distributing the computations

- simple **fork** processing
- shared-memory distribution: using **pthread**
- split-memory distribution: using **mpirun**

Example of flag format



File containing flags

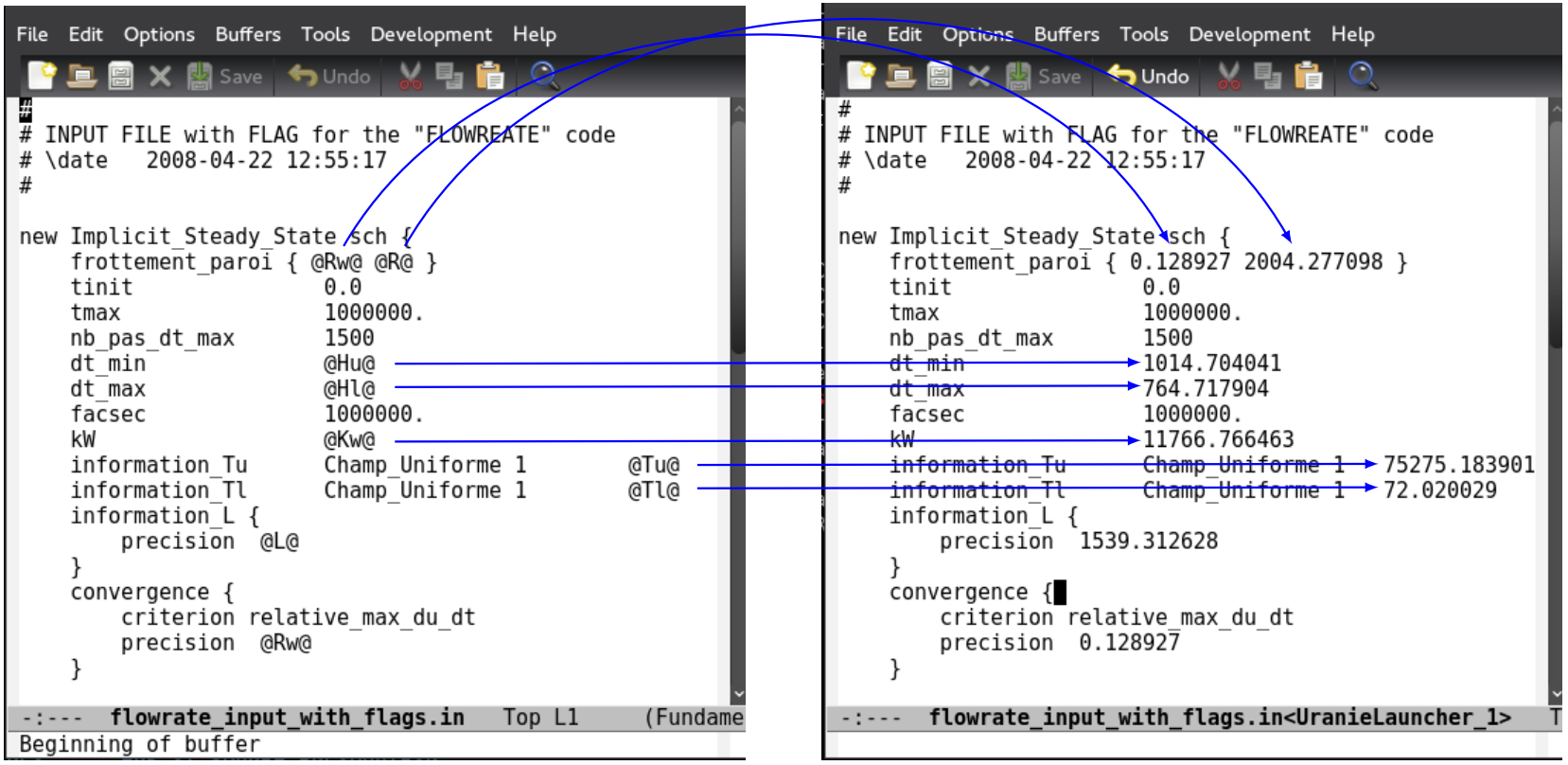


Modified file

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

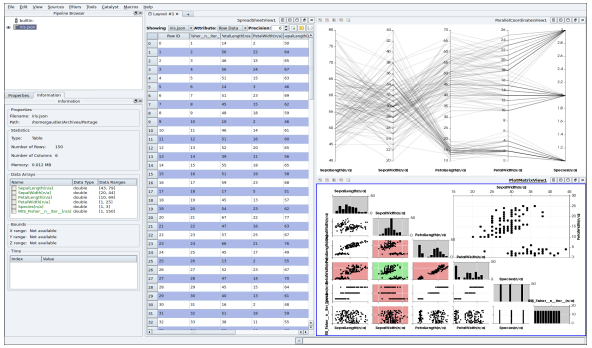
Advantage

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

Communication with other platforms

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 :

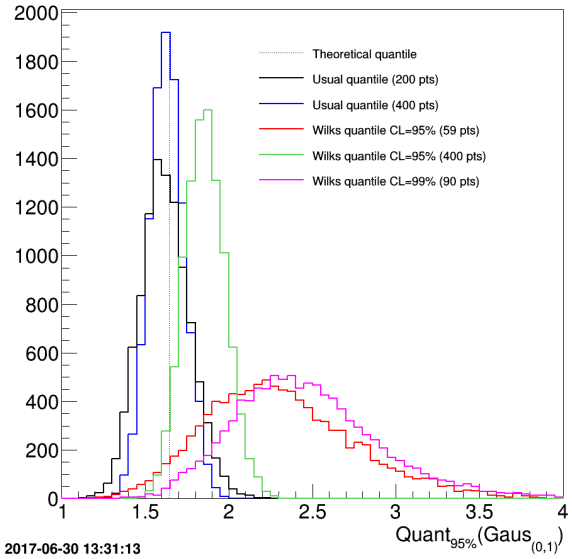
- Benefit the features of D3 (D3js.org)
  - Interactive visualisation into a browser
  - Several available graphics (Cobweb, Sun-Burst, Treemap,..)
- Visualize the same data file in ParaView / Paravis module of Salomé
- Proposal as a common format for data with OpenTURNS

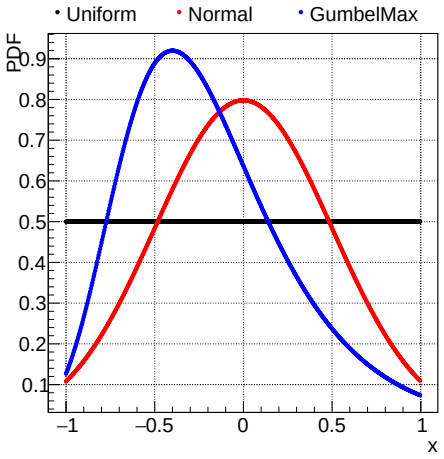
A glimpse at the main modules

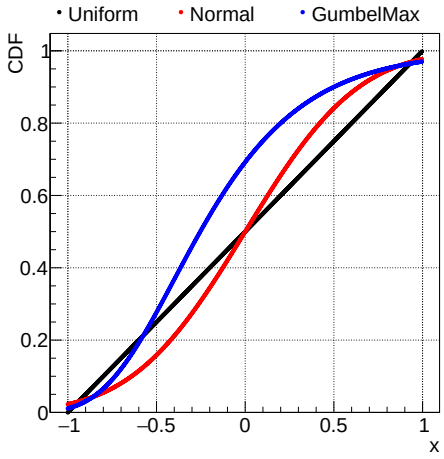
Dataserver module: create and handling variables

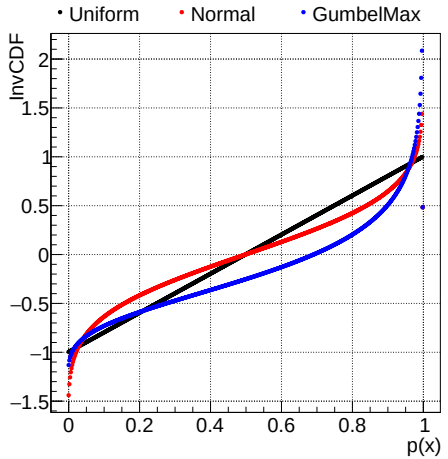
With the DataServer module, one can:

- create new variables from existing ones
- compute first statistical
  - Mean, standard deviation, minimum, maximum
  - Normalisation
  - Correlation matrices
  - Quantile (various definition, among which Wilks' ones)
- define variables using pre-defined statistical laws among: uniform, gaussian, exponential, triangular, beta, weibull. . .
- create plots and import/export data (ASCII, XML, JSON. . .)
  - ➙ See next slide.



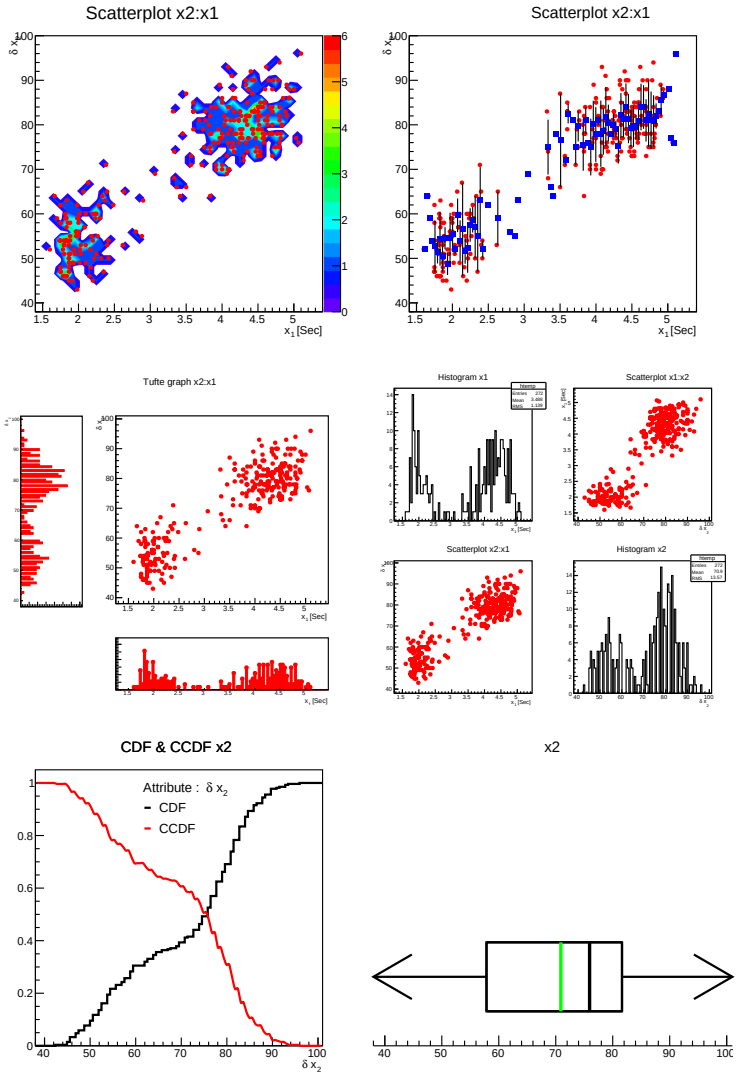






Dataserver module: import/export/represent data

```
//Loading namespaces to get rid of complicated names
using namespace URANIE::DataServer;

//Create dataserver and fill if with data file
TDataServer * tds = new TDataServer("Name", "Titre");
tds->fileDataRead("geyser.dat");

//Create the canvas on which plots will be laid
TCanvas *Can = new TCanvas("Can1","Can1",10,32,800,1200);
Can->Divide(2,3);//Divide the canvas into 6 pads

//2-dimensionnal plots with iso-level as color
Can->cd(1); tds->drawScatterplot("x2:x1");
//2-dimensionnal plots with average of x2 vs x1
Can->cd(2); tds->drawProfile("x2:x1","","same");

//2-dimensionnal plot with projection onto both axis
Can->cd(3); tds->drawTufte("x2:x1");
//All variables two-by-two and 1-dimensionnal plot in diagonal
Can->cd(4); tds->drawPairs();

//Plot CDF and CCDF curve for x2 variable
Can->cd(5); tds->drawCDF("x2","","ccdf");
//Plot BoxPlot (mean, standard deviation, mediane, quantiles...)
Can->cd(6); tds->drawBoxPlot("x2");
```



Can be used either

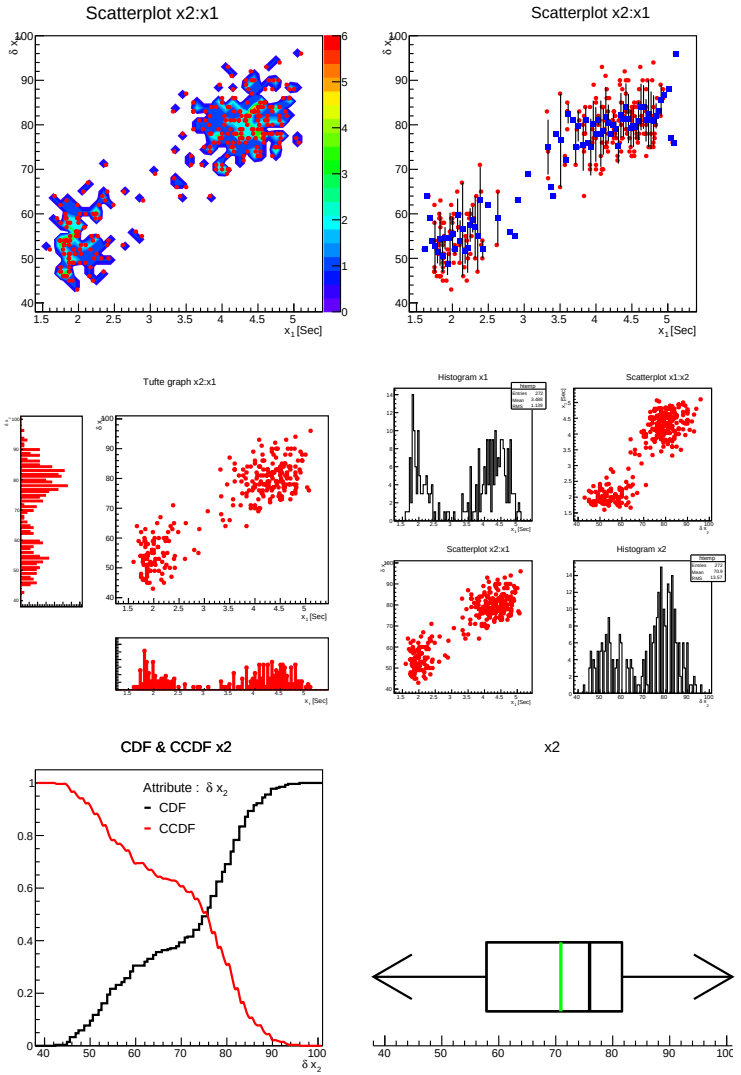
- interactively: `(%) root File.C`
- compiled:

  `(%) g++ -o Exec File.C ` echo $URANIECPPFLAG $URANIELDFLAG`

  `(%) ./Exec`
- interactively in PYTHON: `(%) python -i File.py`

Dataserver module: import/export/represent data

```
import ROOT
from ROOT.URANIE import DataServer as DS

## Create dataserver and fill if with data file
tds = DS.TDataServer("Name", "Titre");
tds.fileDataRead("geyser.dat");

## Create the canvas on which plots will be laid
Can = ROOT.TCanvas("Can1","Can1",10,32,800,1200);
Can.Divide(2,3);## Divide the canvas into 6 pads

## 2-dimensionnal plots with iso-level as color
Can.cd(1); tds.drawScatterplot("x2:x1");
## 2-dimensionnal plots with average of x2 vs x1
Can.cd(2); tds.drawProfile("x2:x1","","same");

## 2-dimensionnal plot with projection onto both axis
Can.cd(3); tds.drawTufte("x2:x1");
## All variables two-by-two and 1-dimensionnal plot in diagonal
Can.cd(4); tds.drawPairs();

## Plot CDF and CCDF curve for x2 variable
Can.cd(5); tds.drawCDF("x2","","ccdf");
## Plot BoxPlot (mean, standard deviation, mediane, quantiles...)
Can.cd(6); tds.drawBoxPlot("x2");
```



Can be used either

- interactively: `(%) root File.C`
- compiled:

```
(%) g++ -o Exec File.C ` echo $URANIECPPFLAG $URANIELDFLAG`
(%) ./Exec
```

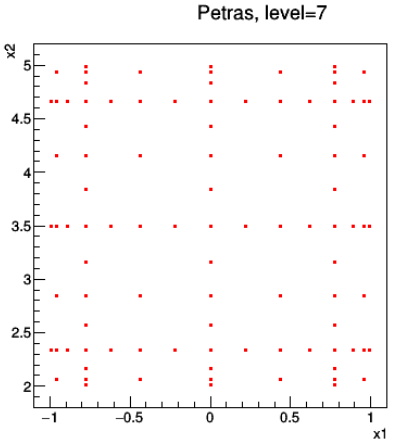
- interactively in PYTHON: `(%) python -i File.py`

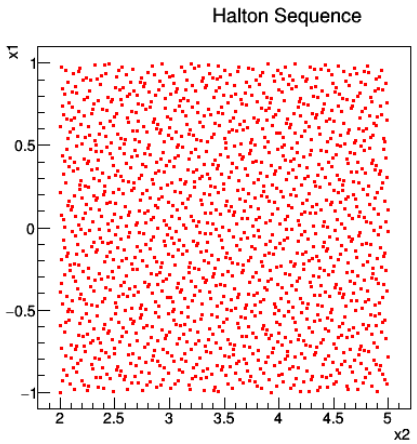
Sampler module

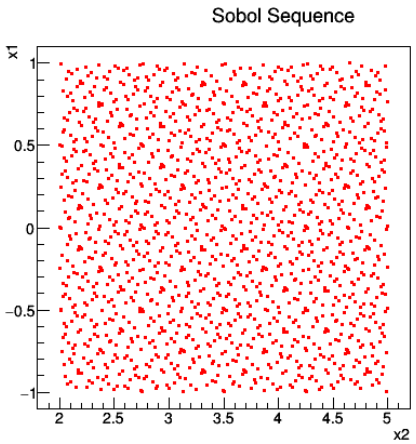
Used to generate the design-of-experiments, basis of many analysis.
Some methods can deal with correlation as well.

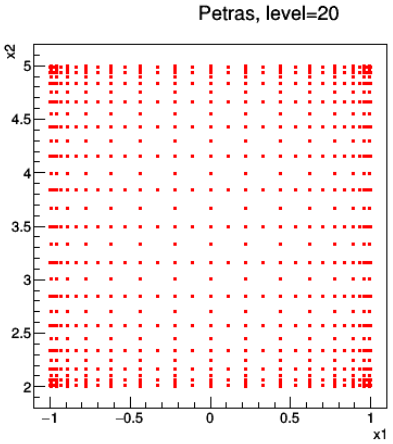
Two main categories

- Stochastic designs:
  - Simple Random Sampling (SRS)
  - Latin Hypercube Sampling (LHS), MaximLHS...
  - 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









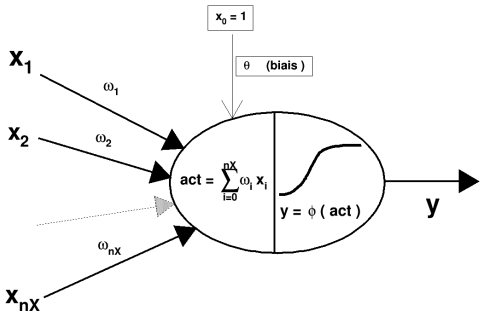
Modeler module

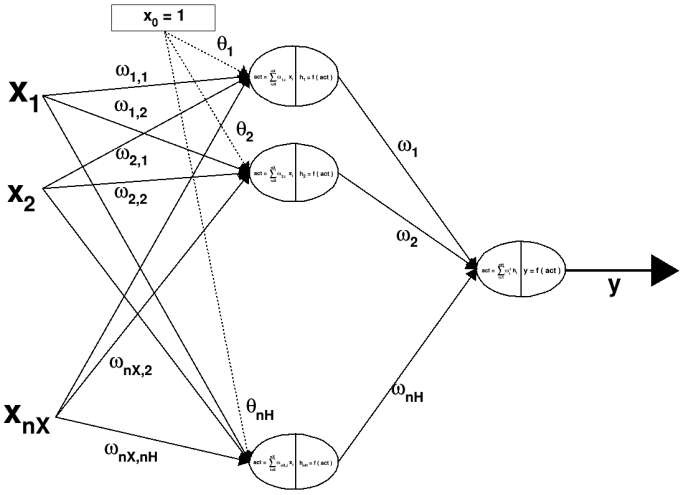
Create a surrogate-model to reproduce the behaviour of provided data

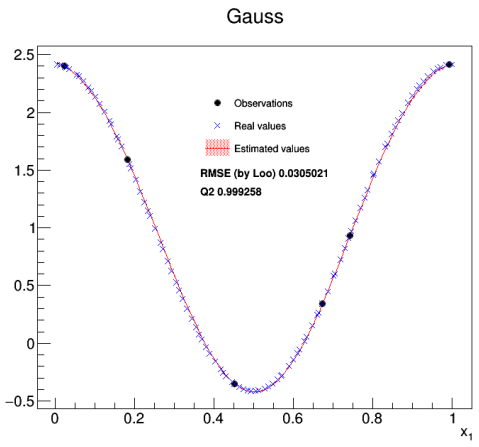
Several possible models to be chosen:

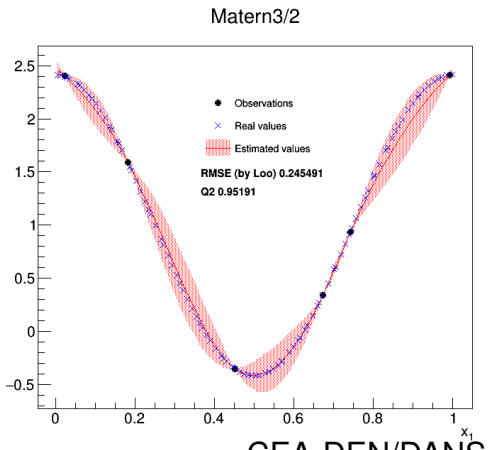
- Polynomial regressions
- Generalised linear models
- k-Nearest neighbour
- Kernel methods
- Artificial Neural Networks (ANN/MLP)
- Chaos polynomial expansion
- Gaussian process (kriging with gpLib)

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







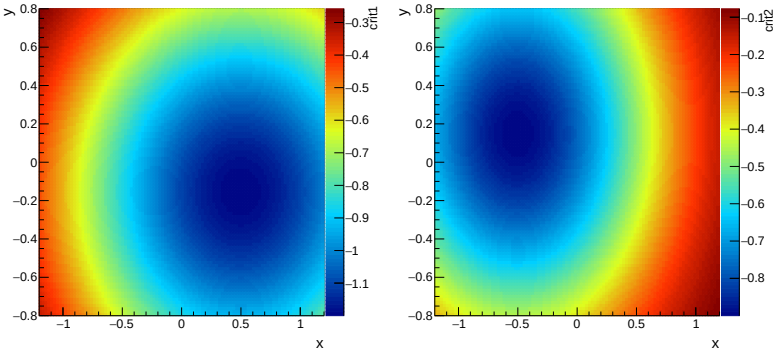


2017-06-30 13:16:31

Optimizer module

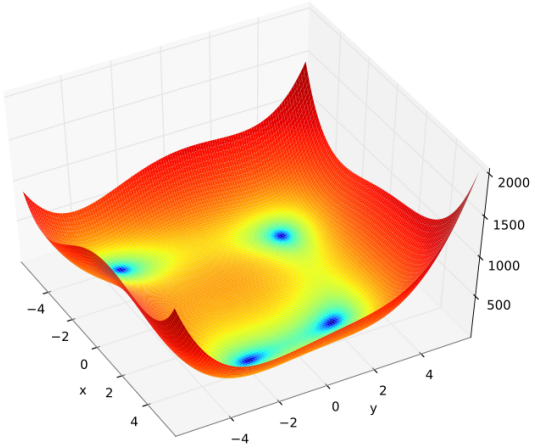
Dealing with optimisation problem usually means:

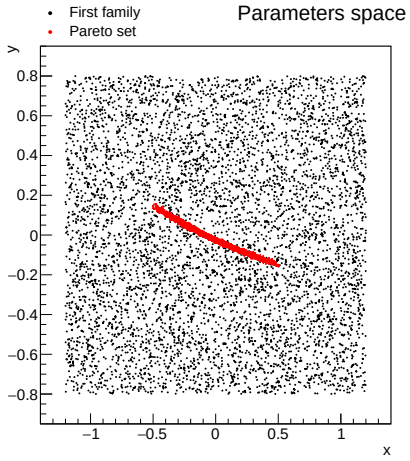
- Single Objective (SO) or Multi Objectives (MO) to be minimised
- parameters that have an impact on objective
- possible constraint on these parameters

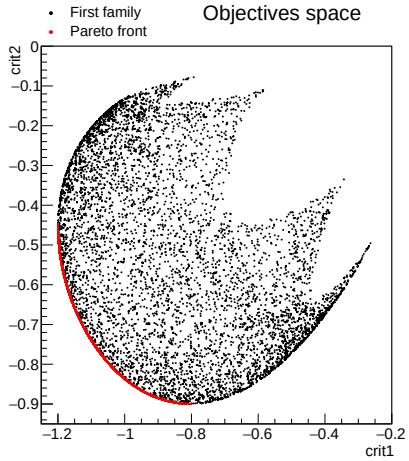


Many possible implementation for this, based on:

- 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)





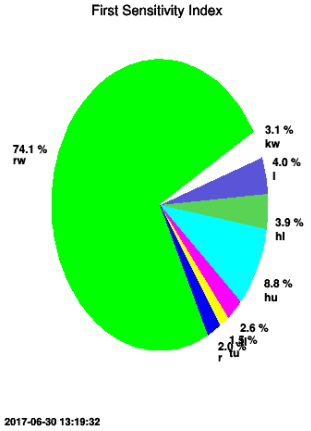


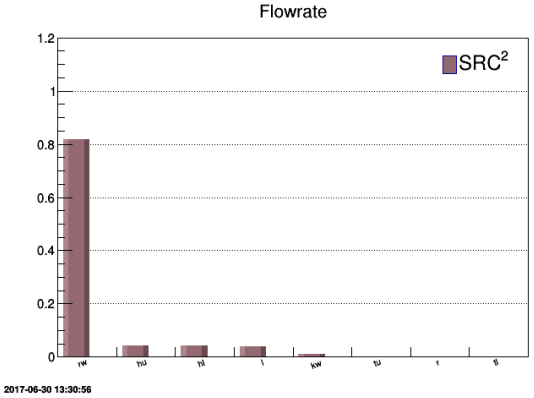
Sensitivity module

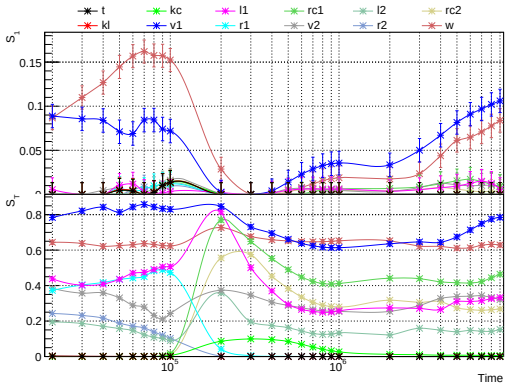
Tools to evaluate the sensitivity of the outputs of a code/function to its inputs.

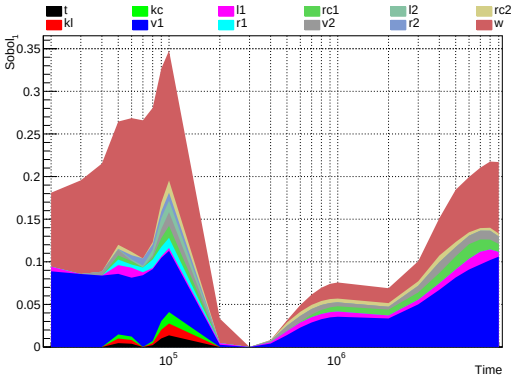
Several kinds of methods available:

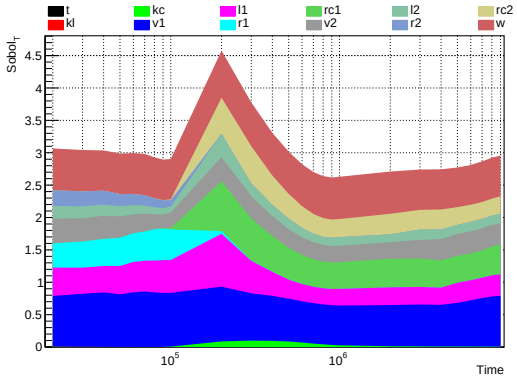
- Local: finite differences $(\frac{\delta Y_i}{\delta X_j}(x_0))$
- Regression:
  - Pearson (values)
  - Spearman (ranks)
- Screening: OAT, Morris...
- Sobol indexes:
  - FAST (Fourier Amplitude Sensitivity Test)
  - RBD (Random Balance Design)
  - Sobol/Saltelli Methods



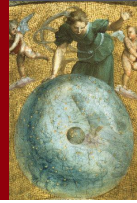








Eyes-on: a simple example



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)

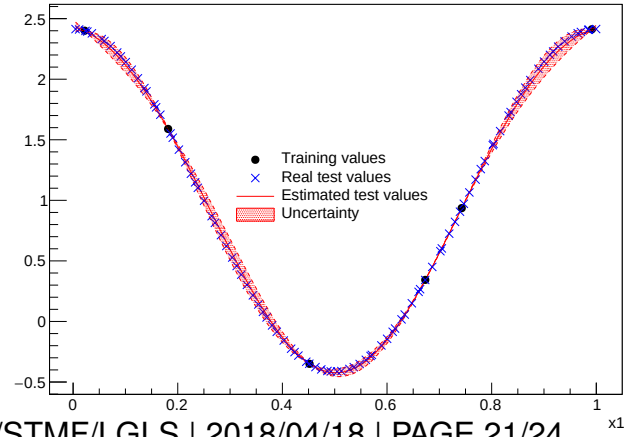
Loading cc-langs...done

called by

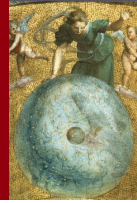
```
(mar.10mai 0:37)-(blanchard@jbpc:...ipts/modeler)-(1|%) root OneKrigingOnly.C
root [0]
Processing /home/blanchard/.root_logon...
Processing OneKrigingOnly.C...

--- 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

Kriging example



Combining techniques



Blocks as introduced previously can be combined to get new techniques.

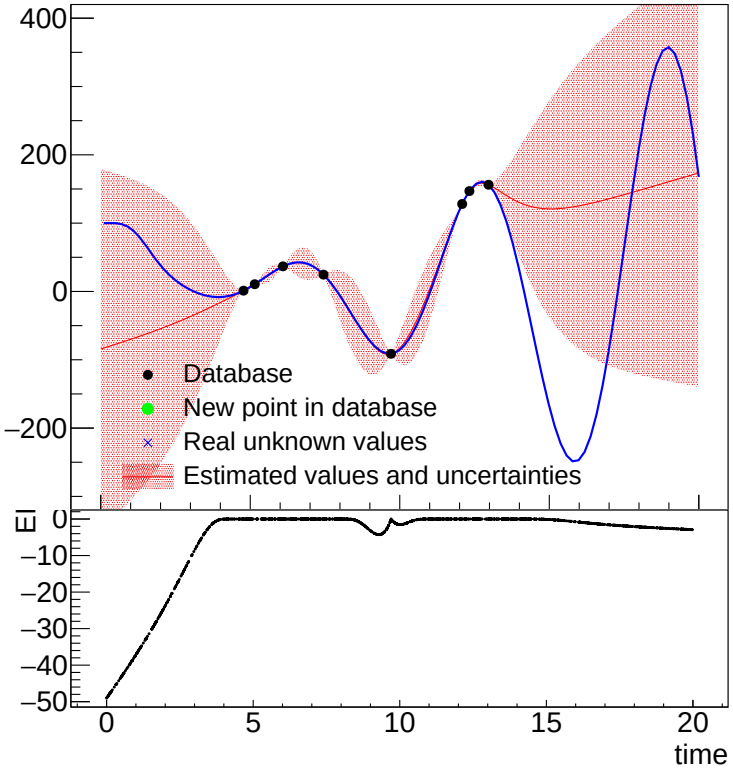
Efficient Global Optimisation (EGO)

From a small database (here 8 points)

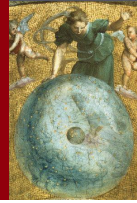
- Construct a kriging model
- Compute the Expected Improvement with the kriging model
  - → using genetic algorithm to get the minimum t^*
- Compute the real new value with the code at t^*
- Reconstruct the kriging on the database + t^*
- Continue this process iteratively. . .

Ongoing work to parallelise this process

Typically used for very time/cpu consuming code. Investigating different approaches to estimate the new points.



Combining techniques



Blocks as introduced previously can be combined to get new techniques.

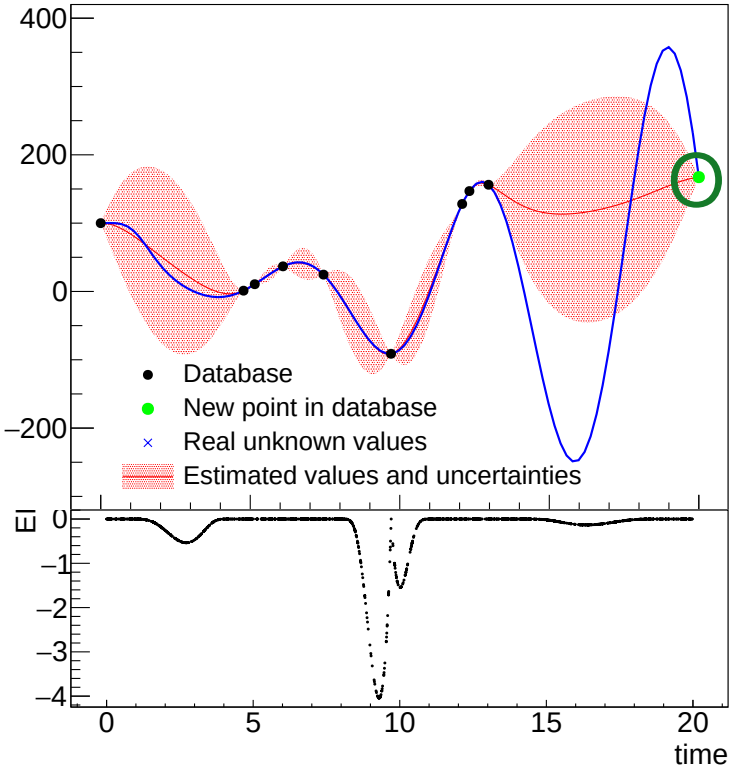
Efficient Global Optimisation (EGO)

From a small database (here 8 points)

- Construct a kriging model
- Compute the Expected Improvement with the kriging model
  - → using genetic algorithm to get the minimum t^*
- Compute the real new value with the code at t^*
- Reconstruct the kriging on the database + t^*
- Continue this process iteratively. . .

Ongoing work to parallelise this process

Typically used for very time/cpu consuming code. Investigating different approaches to estimate the new points.



Combining techniques

Blocks as introduced previously can be combined to get new techniques.

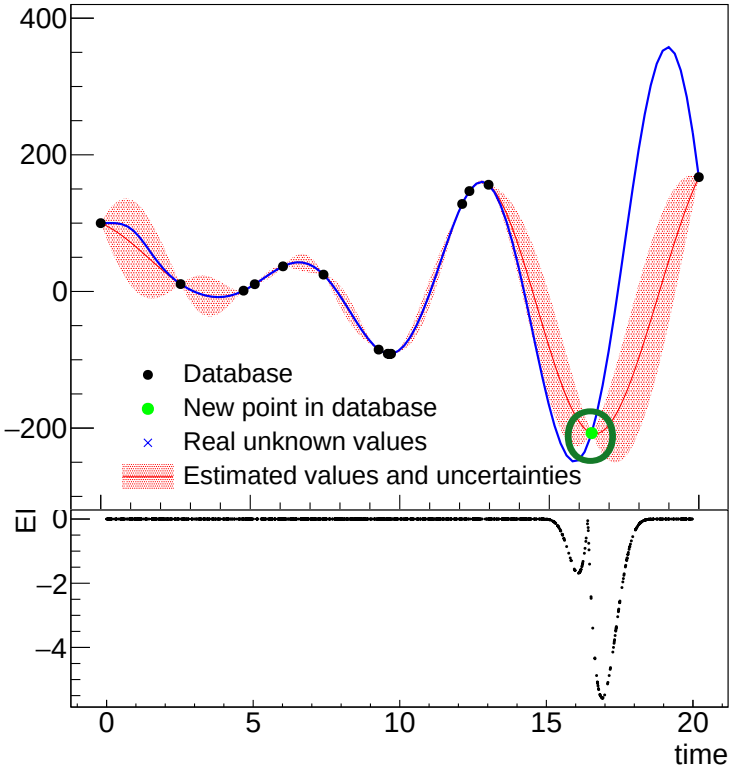
Efficient Global Optimisation (EGO)

From a small database (here 8 points)

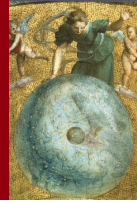
- Construct a kriging model
- Compute the Expected Improvement with the kriging model
  - → using genetic algorithm to get the minimum t^*
- Compute the real new value with the code at t^*
- Reconstruct the kriging on the database + t^*
- Continue this process iteratively. . .

Ongoing work to parallelise this process

Typically used for very time/cpu consuming code. Investigating different approaches to estimate the new points.



Combining techniques



Blocks as introduced previously can be combined to get new techniques.

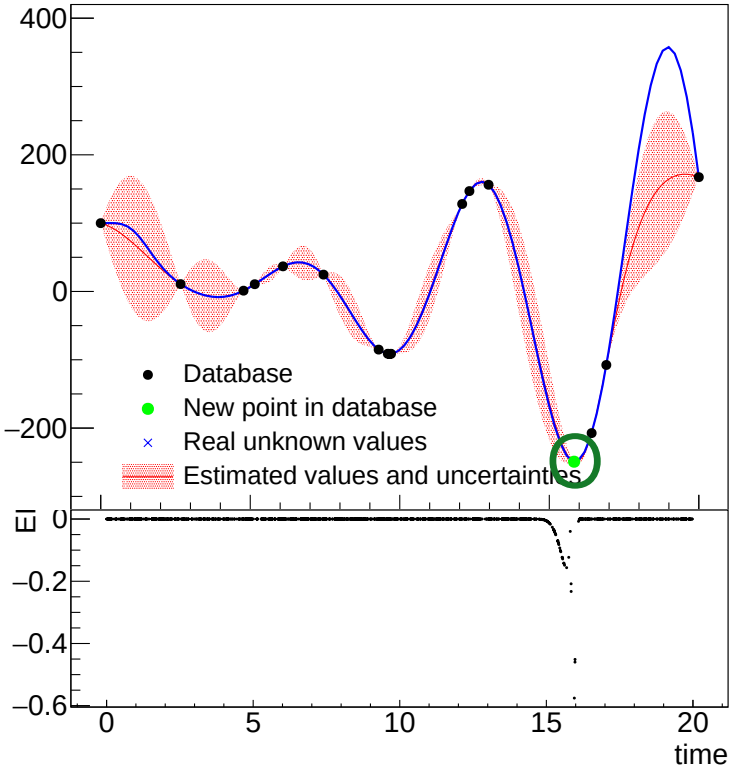
Efficient Global Optimisation (EGO)

From a small database (here 8 points)

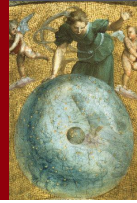
- Construct a kriging model
- Compute the Expected Improvement with the kriging model
  - → using genetic algorithm to get the minimum t^*
- Compute the real new value with the code at t^*
- Reconstruct the kriging on the database + t^*
- Continue this process iteratively. . .

Ongoing work to parallelise this process

Typically used for very time/cpu consuming code. Investigating different approaches to estimate the new points.



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 (in France)

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