DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

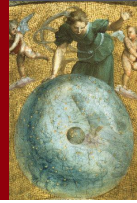
The Uranie platform

A broad introduction

J-B. Blanchard, on behalf of the Uranie team
jean-baptiste.blanchard@cea.fr

ETSN 2018 | 2018/04/25

Purpose of this presentation



Uncertainty at CEA/DEN are of interest for many purposes:

- LEONAR tool for severe accidents in french nuclear reactor (CEA-EDF)
- Sensitivity Analysis for Cathare code (Areva TA)
- Multi-criteria optimisation (CEA CESTA/CELIA), Astrid...
- European project NURESIM/NURISP

Uranie platform has been developped to deal with

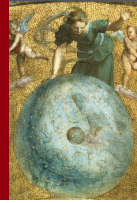
- Uncertainty propagation
- Optimisation (Single/Multi objectives)
- Sensitivity analysis
- Surrogate modeling generation
- Code calibration
- ...

What I've been asked by Renaud

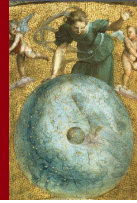
Introduce the platform so that if people are interested they can get it and start playing with it.

- Technical description of the platform (hopefully not to long)
  $\Rightarrow$ to keep as a reference if anyone wants to give a shot (on top of the documentation)
- Introducing a "toy-physical" use-case used throughout the rest
- Go through main steps of a possible analysis.

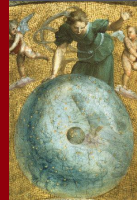
Outline



Outline

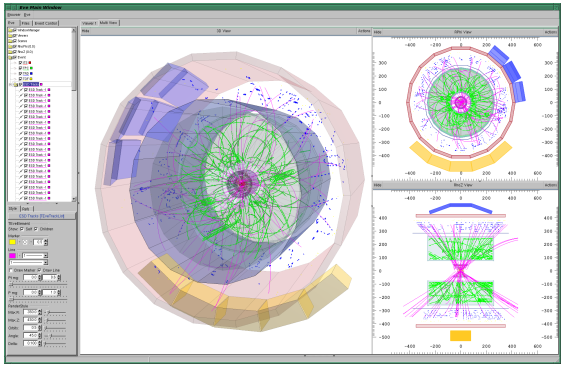


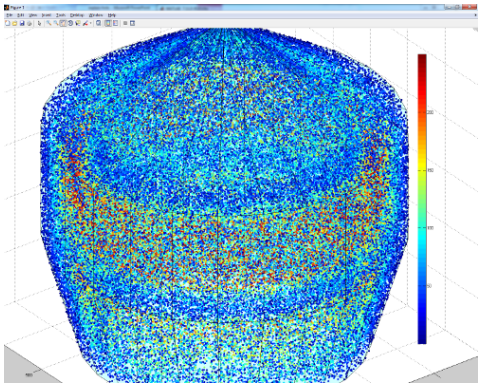
The ROOT platform

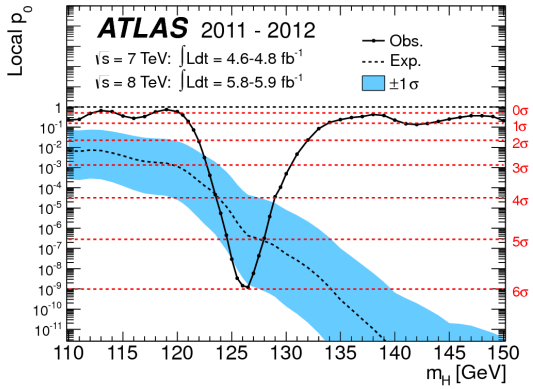


Developed at CERN to help analyse the huge amount of data delivered by the successive particle accelerators

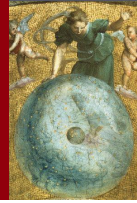
- Written in C++ (3/4 releases a year)
- Multi platform (Unix/Windows/Mac OSX)
- Started and maintained over more than 20 years
- It brings:
  - a C++ interpreter, but also Python and Ruby interface
  - a hierarchical object-oriented database (machine independent and highly compressed)
  - advanced visualisation tool (graphics are very important in HEP)
  - statistical analysis tools (*RooStats*, *RooFit*...)
  - and many more (3D object modelling, distributed computing interface...)
- LGPL







Coding conventions



ROOT uses some unusual coding/naming conventions

- Class names start with capital T: `TH1F`, `TF1`, `TVector` ✔
- Names of non-class data types end with _t: `Int_t`
- Separate words with in names are capitalised: `TH1::GetTitleOffset()` ✔
- Class method names start with a capital letter: `TH1F::Fill()` ✗
- Class data member names start with an f: `TH1::fXaxis` ✗
- Global variable names start with a g: `gPad`
- Constant names start with a k: `TH1::kNoStats` ✔

All what will be discussed here, is dealing with ROOT-5 versions, unless otherwise specified

C++ interpreter

First contact you'll have with ROOT

- Allows to write C++ line-by-line
- Native prints variable-content when ";" is omitted
- Provides syntax highlighting and simple auto-completion

```
bash-4.3$ root -l
root [0] int a=3;
root [1] int b=3*a;
root [2] b
(int)9
root [3]
```

Pros:

- Very simple to use
- Allows you to grope for what you really want
- Provide an integrated compiler if one is not familiar with C++ compilation

Cons:

- Is slower than compiled C++
- Allows strange code (forbidden by proper C++)
- Has its own way to handle memory (`delete` are risky)
- All variables are global → Restart CInt regularly

Good practice: you can play with it to test, but most of your work should be done through scripts

Drastic changes once moving to ROOT6

- It becomes C++11 compliant
- CInt becomes Clang which compiles (on the fly) the provided code
  - ➙ Much less differences with proper C++
  - ➙ Induce changes for some of our constructor

A simple example

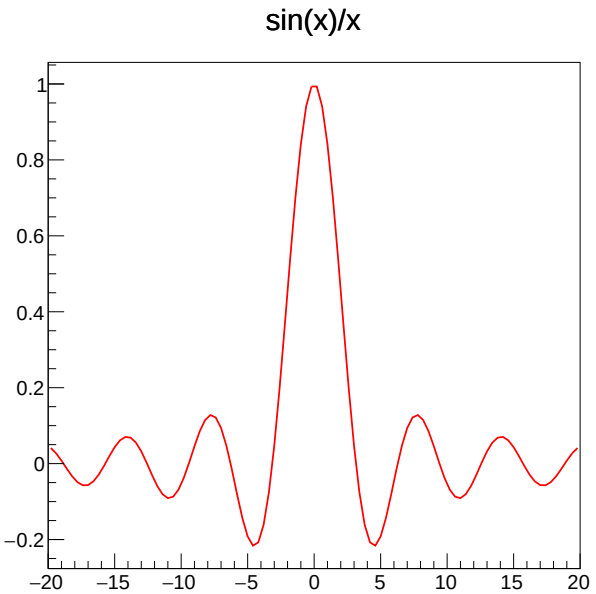
Interpreted scripts

```
{
     TCanvas *Can  = new TCanvas("Can","Can",2);
     TF1 *f = new TF1("f","sin(x)/x", -20, 20);
     f->Draw();
     Can->SaveAs("SinCar.pdf");
}
```

```
bash-4.3$ root -l ROOTSEUn.C
root [0]
Processing ROOTSEUn.C...
Info in <TCanvas::Print>: pdf file SinCar.pdf has been created
root [1]
```

```
void ROOTSENamed()
{
     TCanvas *Can  = new TCanvas("Can","Can",2);
     TF1 *f = new TF1("f","sin(x)/x", -20, 20);
     f->Draw();
     Can->SaveAs("SinCar.pdf");
}
```

```
bash-4.3$ root -l ROOTSENamed.C
root [0]
Processing ROOTSENamed.C...
Info in <TCanvas::Print>: pdf file SinCar.pdf has been created
root [1] .q
bash-4.3$ root -l
root [0] .L ROOTSENamed.C
root [1] ROOTSENamed()
Info in <TCanvas::Print>: pdf file SinCar.pdf has been created
root [2]
```



A simple example

Compiled scripts

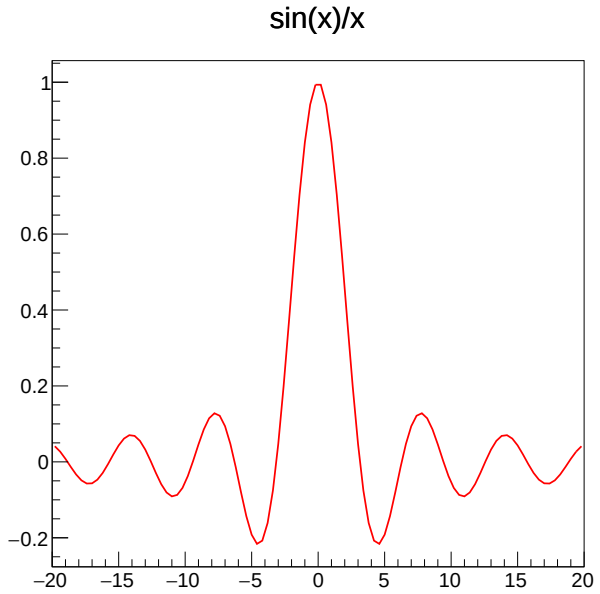
```
#include "TF1.h"
#include "TCanvas.h"

int main()
{
     TCanvas *Can  = new TCanvas("Can","Can",2);
     TF1 *f = new TF1("f","sin(x)/x", -20, 20);
     f->Draw();
     Can->SaveAs("SinCar.pdf");
     return 0;
}
```

```
bash-4.3$ g++ -o test ROOTSEComp.C `root-config --cflags --evelibs`
bash-4.3$ ./test
Info in <TCanvas::Print>: pdf file SinCar.pdf has been created
bash-4.3$
```

Python Equivalent

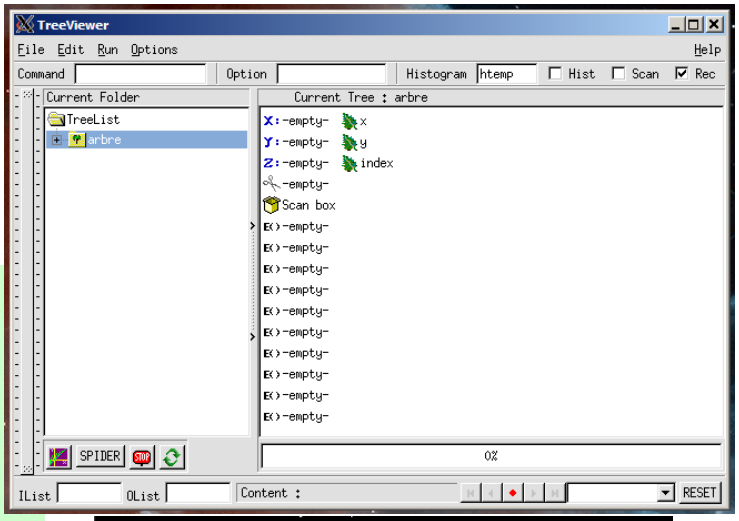
```
import ROOT

Can  =  ROOT.TCanvas("Can","Can",2);
f = ROOT.TF1("f","sin(x)/x", -20, 20);
f.Draw();
Can.SaveAs("SinCar.pdf");
```

```
bash-4.3$ python -i ROOTSE.py
Info in <TCanvas::Print>: pdf file SinCar.pdf has been created
>>>
```



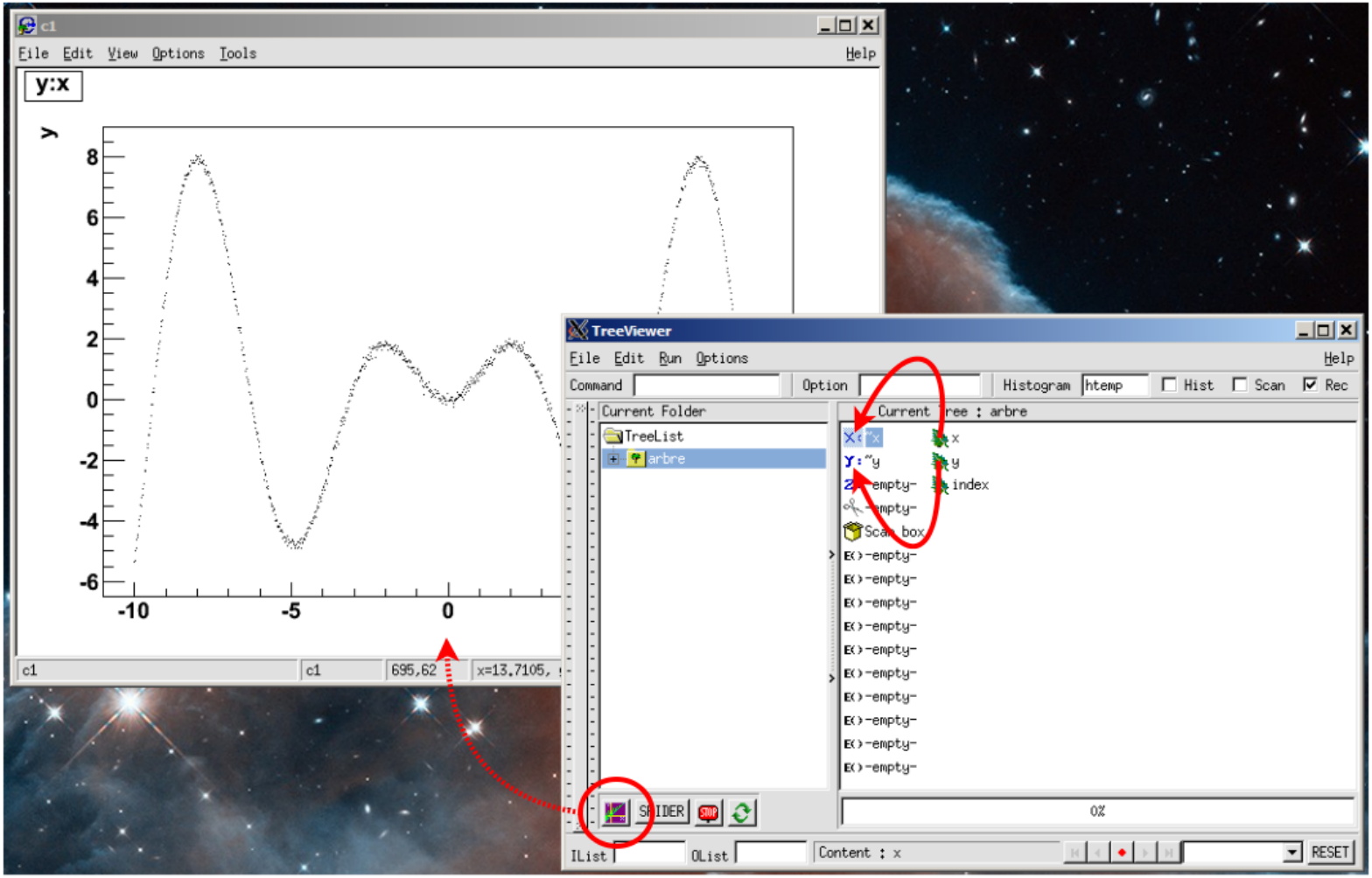
TTree or how to store data

```
{
 //Define constant
 Int_t nbSteps = 1000;
 Double_t xMin = -10.0, xMax = 10.0;

 //Create the function
 TF1 *myFunction = new TF1("function", "x*sin(x)",xMin,xMax);
 // Create a tree in which results will be stored
 TNtupleD *myTree = new TNtupleD("tree", "Example", "x:y:index");

 //Fill the tree
 for(Int_t i = 0; i < nbSteps; i++)
 {
    Double_t v1 = xMin + i * (xMax-xMin)/nbSteps;
    Double_t v2 = myFunction->Eval(v1) + gRandom->Gaus(0.0, 0.1);
    myTree->Fill(v1,v2,(Double_t)(i+1));
 }

 // Showing content
 myTree->StartViewer();
 myTree->Scan("*");
}
```

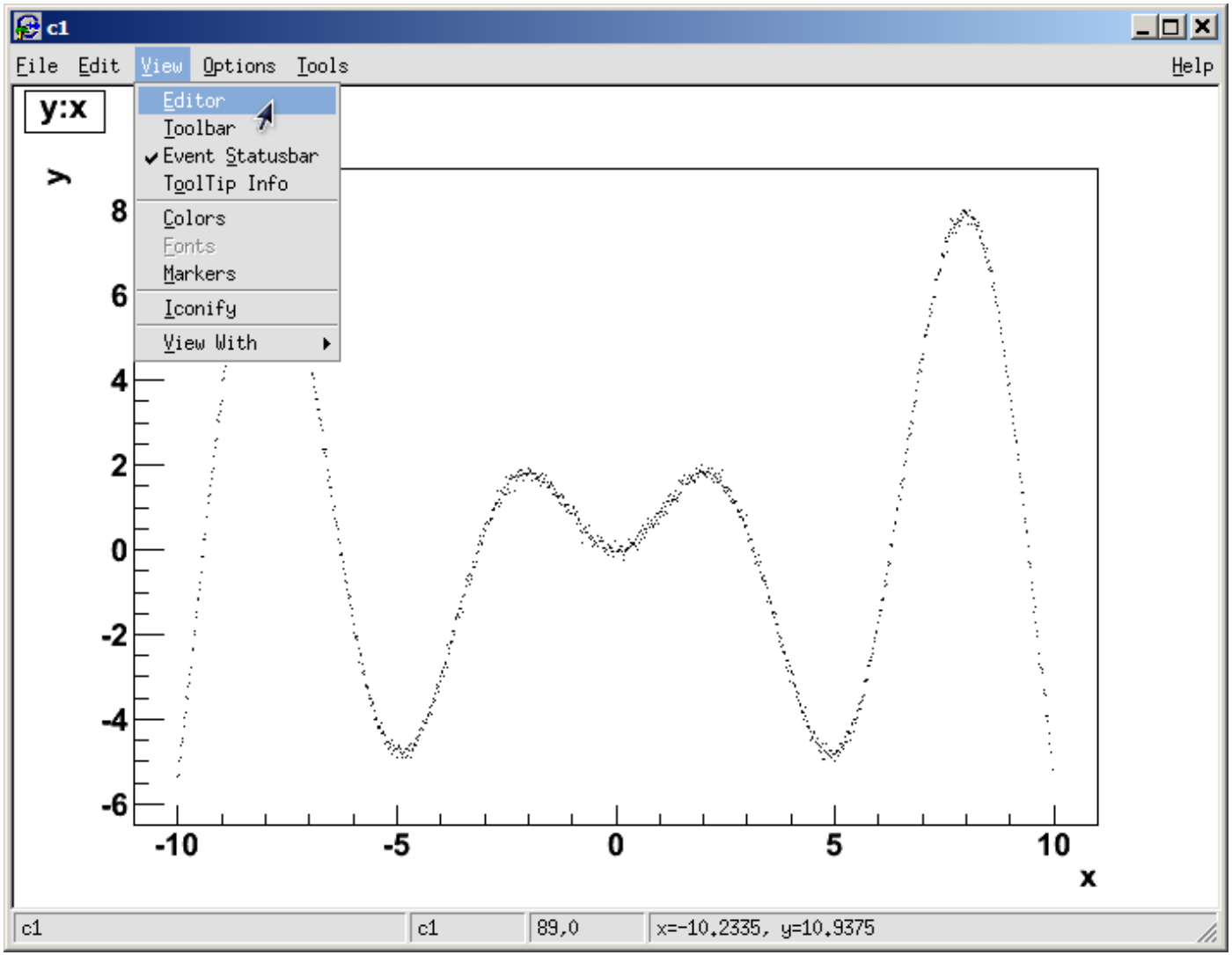



```
root [0]
Processing myscript.C...
*************************************************
*    Row   *         x *         y *     index *
*************************************************
*        0 *       -10 * -5.340317 *         1 *
*        1 *     -9.98 * -5.304253 *         2 *
*        2 *     -9.96 * -5.001739 *         3 *
*        3 *     -9.94 * -4.900722 *         4 *
*        4 *     -9.92 * -4.631825 *         5 *
*        5 *      -9.9 * -4.535277 *         6 *
*        6 *     -9.88 * -4.433945 *         7 *
*        7 *     -9.86 * -4.164562 *         8 *
*        8 *     -9.84 * -3.968597 *         9 *
*        9 *     -9.82 * -3.821905 *        10 *
*       10 *      -9.8 * -3.452376 *        11 *
*       11 *     -9.78 * -3.499976 *        12 *
*       12 *     -9.76 * -3.215727 *        13 *
*       13 *     -9.74 * -3.164002 *        14 *
*       14 *     -9.72 * -2.934123 *        15 *
*       15 *      -9.7 * -2.774907 *        16 *
*       16 *     -9.68 * -2.367075 *        17 *
*       17 *     -9.66 * -2.324952 *        18 *
*       18 *     -9.64 * -2.000788 *        19 *
*       19 *     -9.62 * -1.904343 *        20 *
*       20 *      -9.6 * -1.467446 *        21 *
*       21 *     -9.58 * -1.604544 *        22 *
*       22 *     -9.56 * -1.172253 *        23 *
*       23 *     -9.54 * -1.142207 *        24 *
*       24 *     -9.52 * -0.918621 *        25 *
Type <CR> to continue or q to quit ==> q
*************************************************
root [1]
```

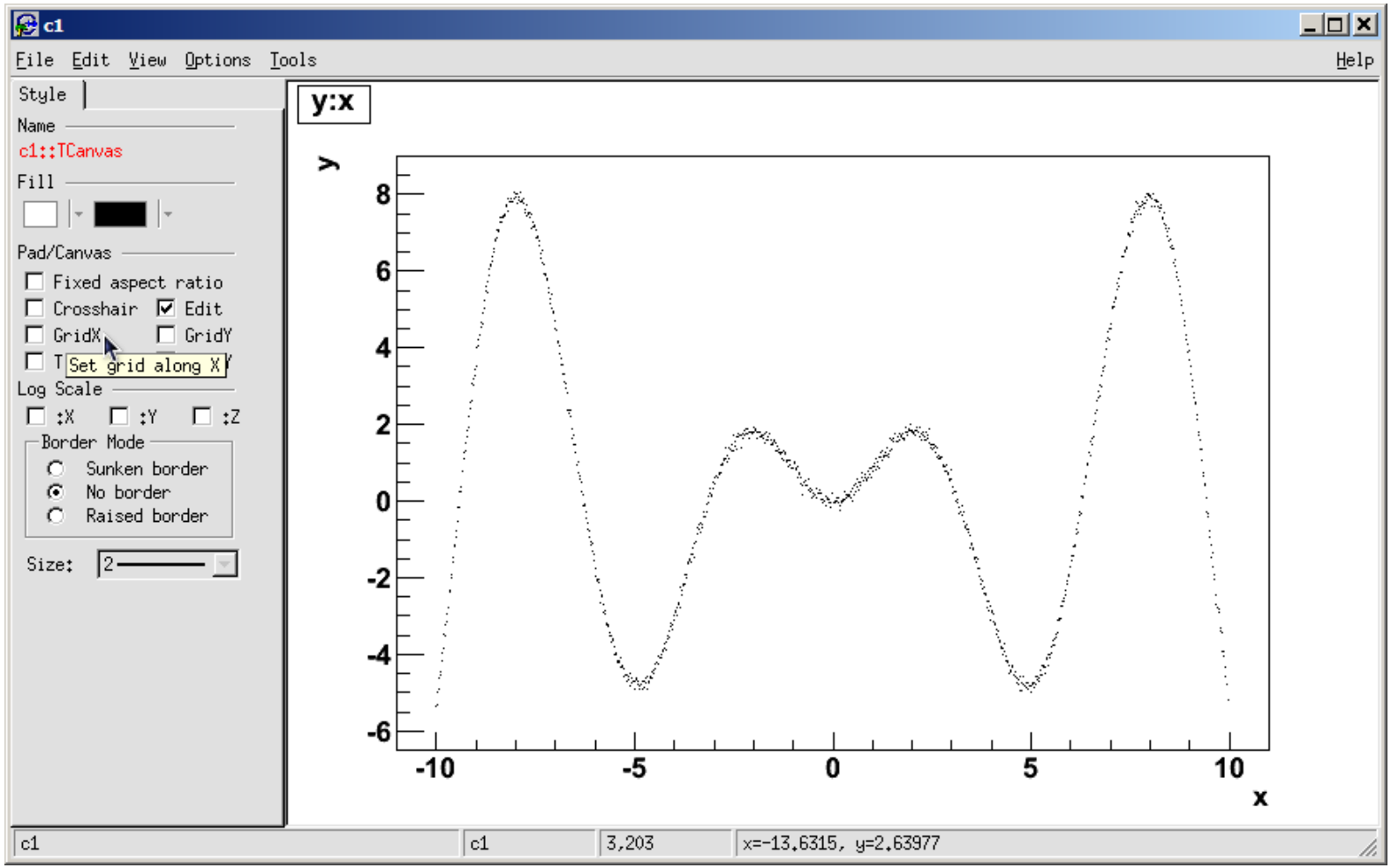
Producing plots and playing with style



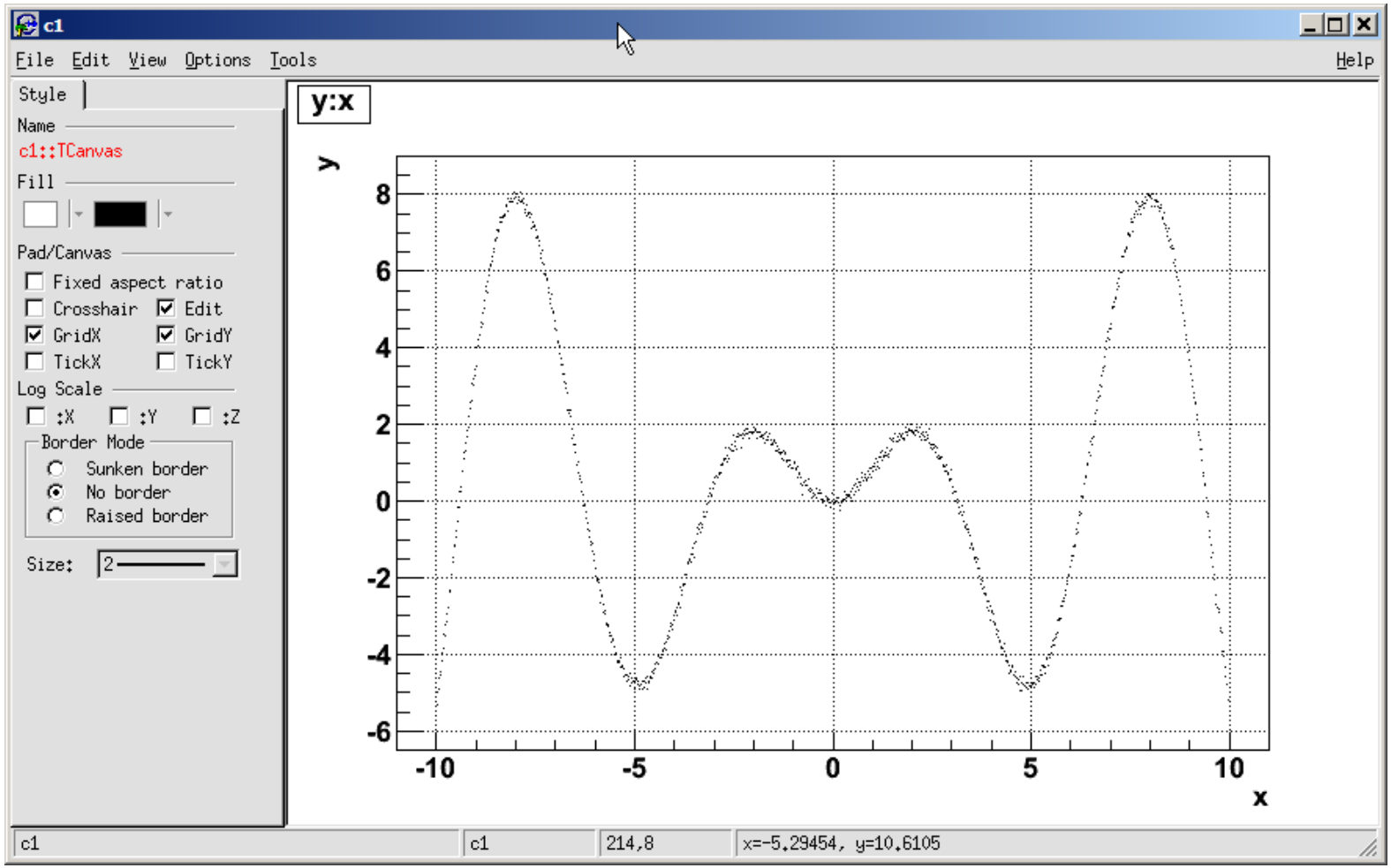
Producing plots and playing with style



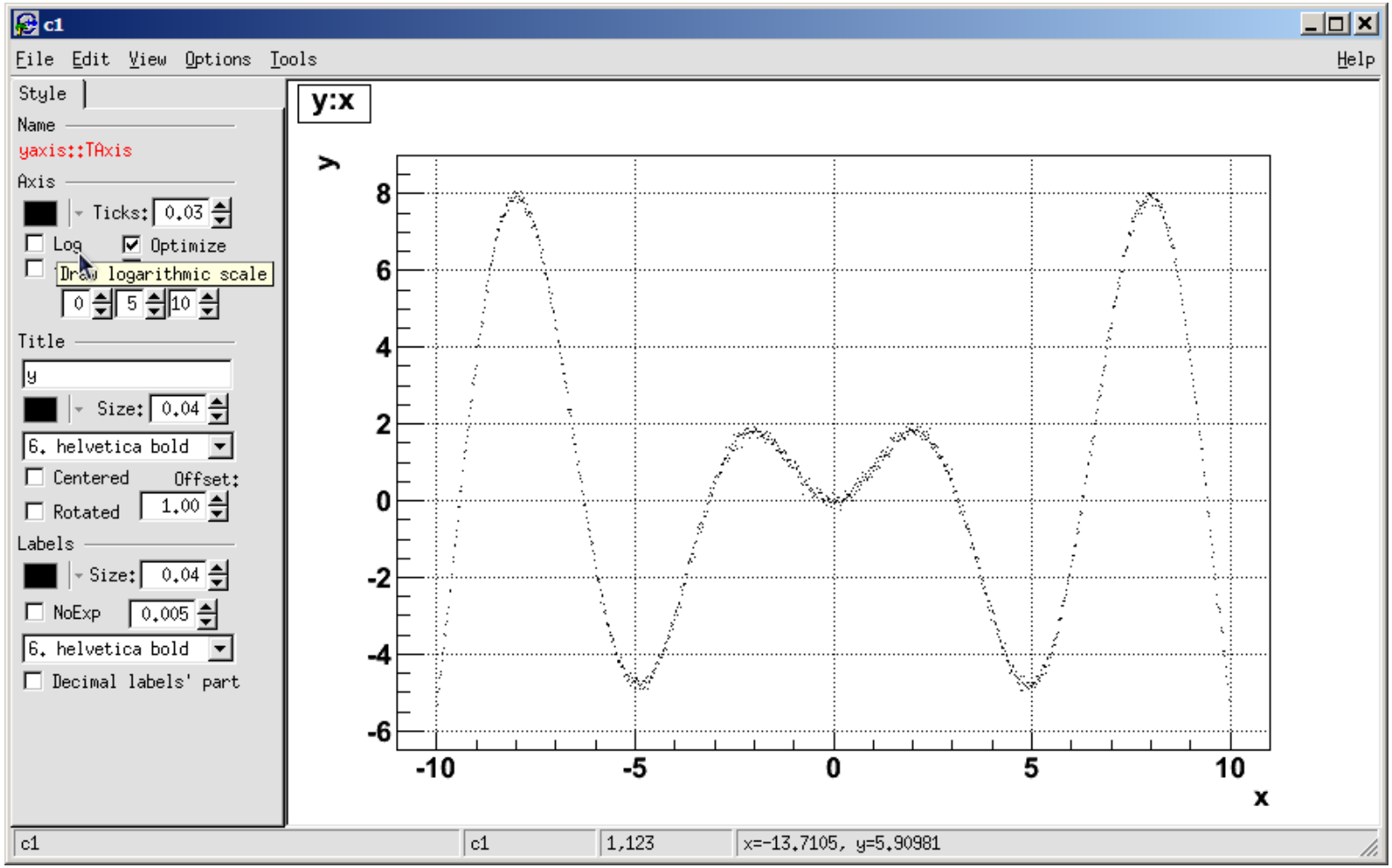
Producing plots and playing with style



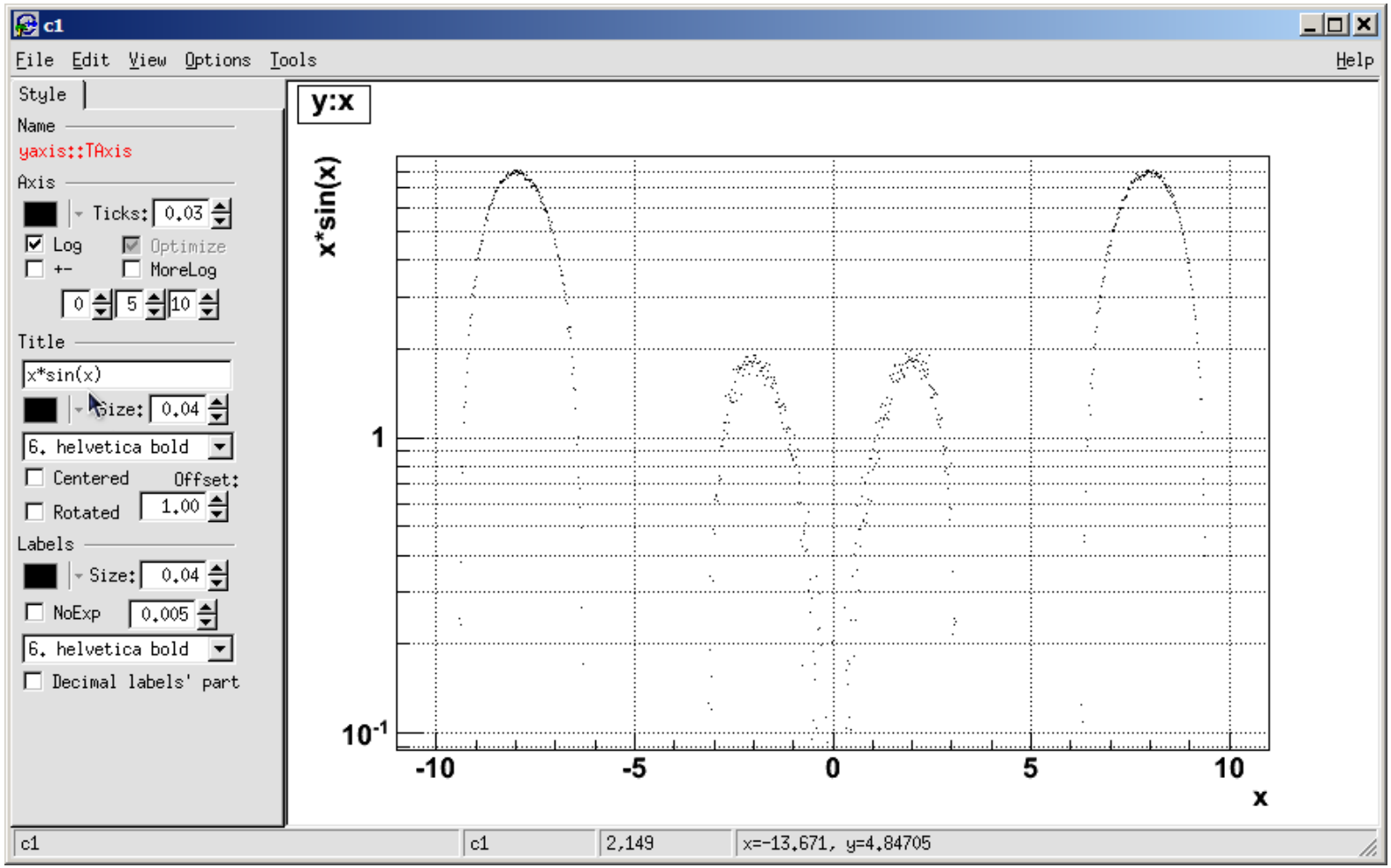
Producing plots and playing with style



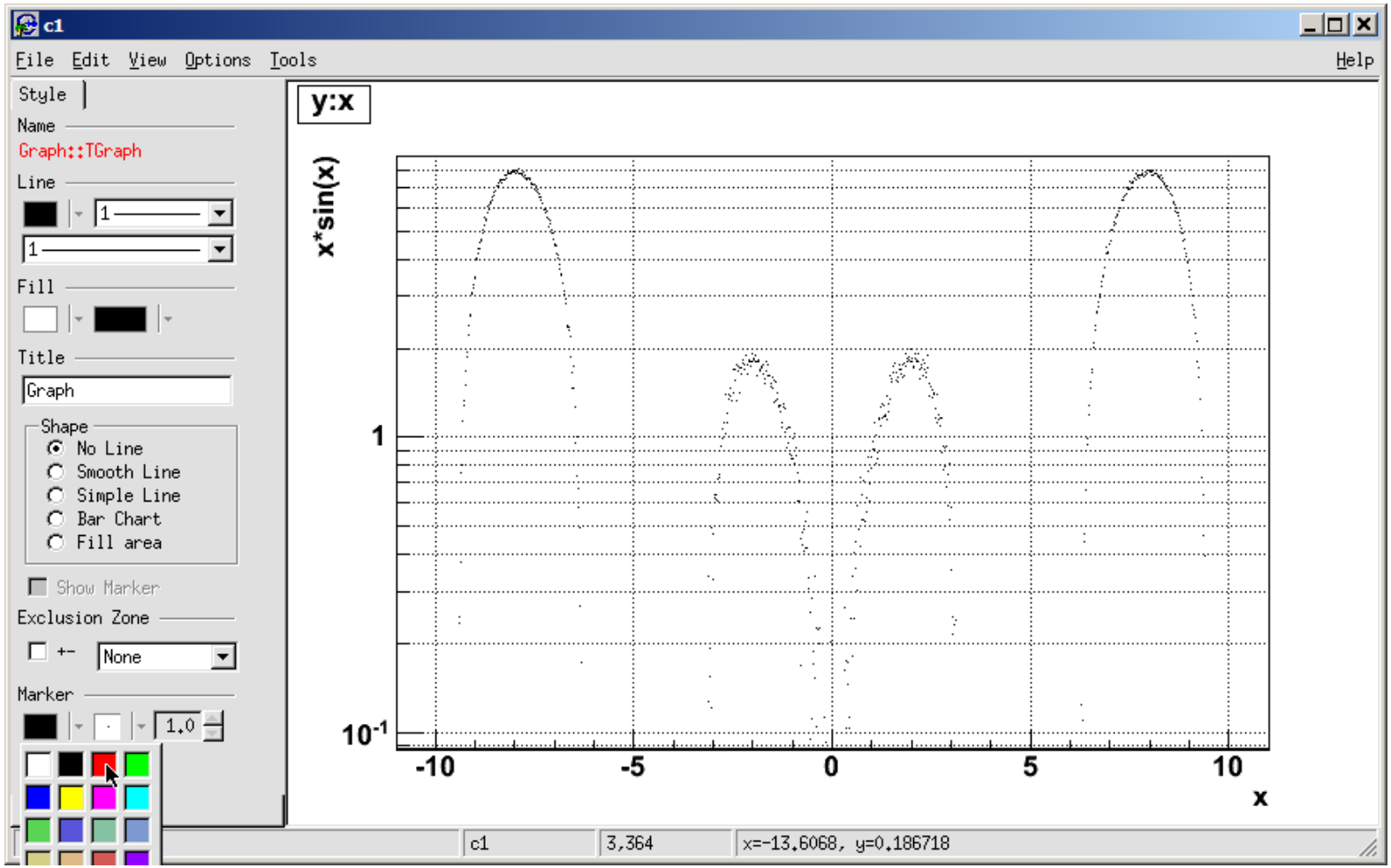
Producing plots and playing with style



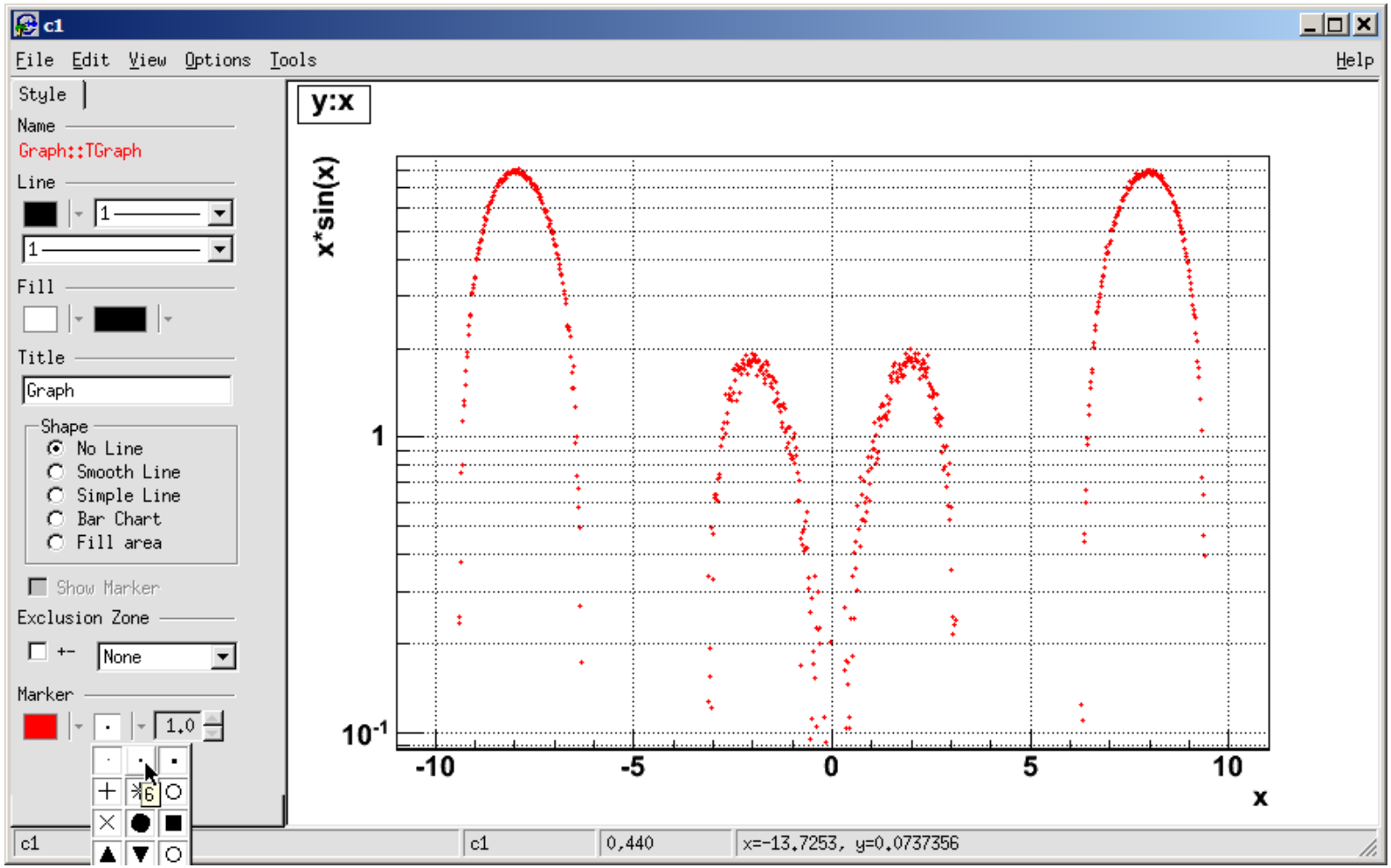
Producing plots and playing with style



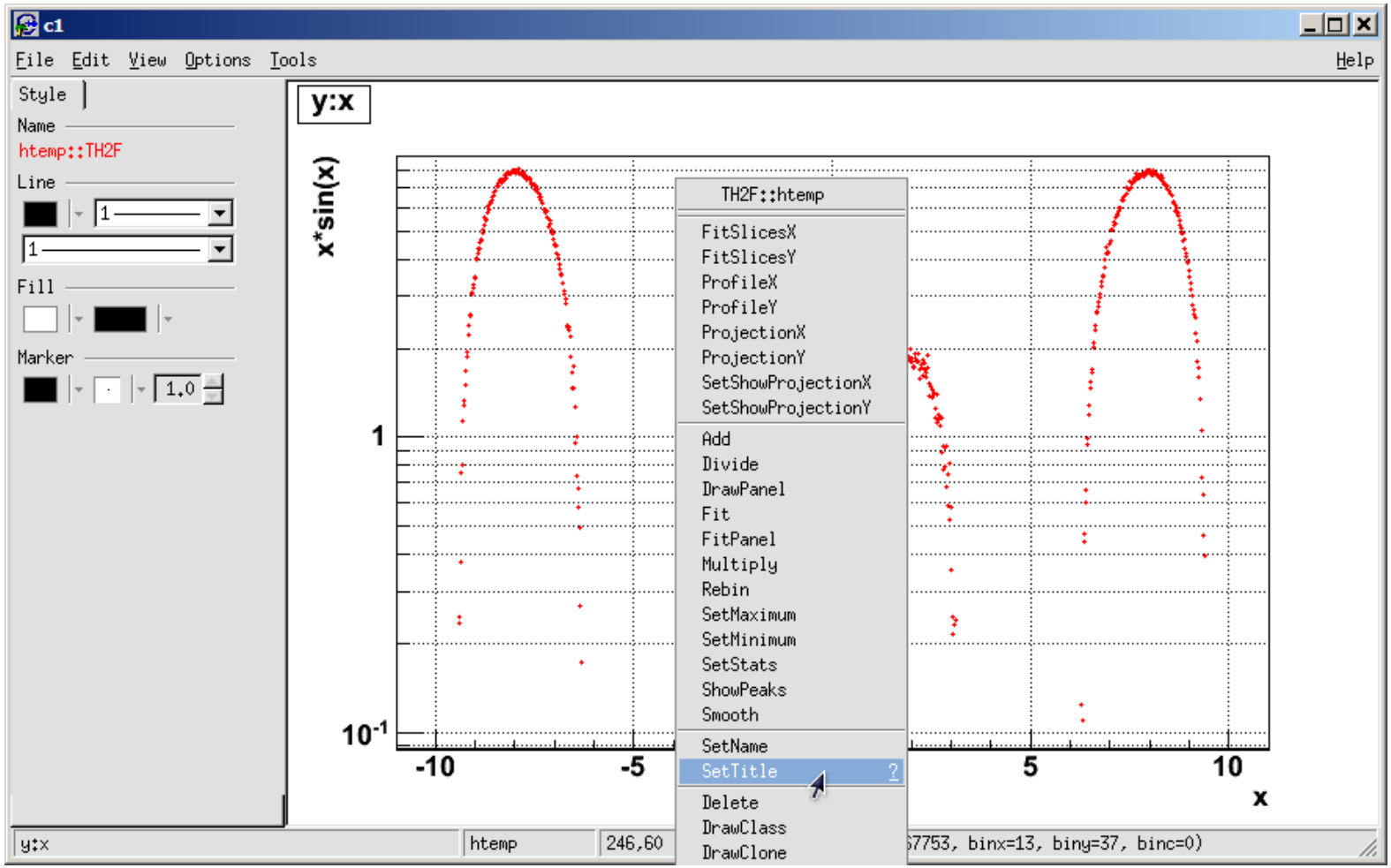
Producing plots and playing with style



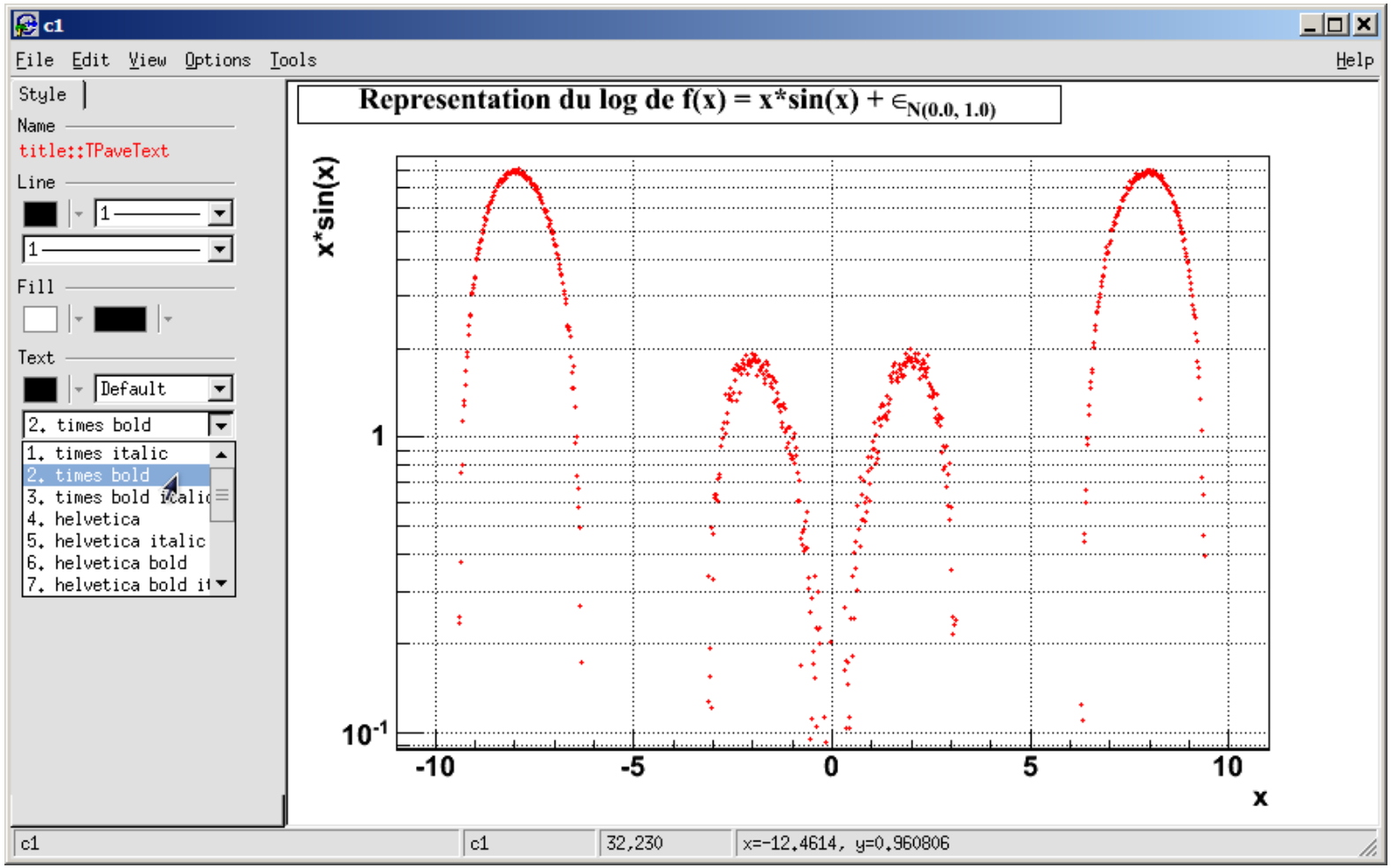
Producing plots and playing with style



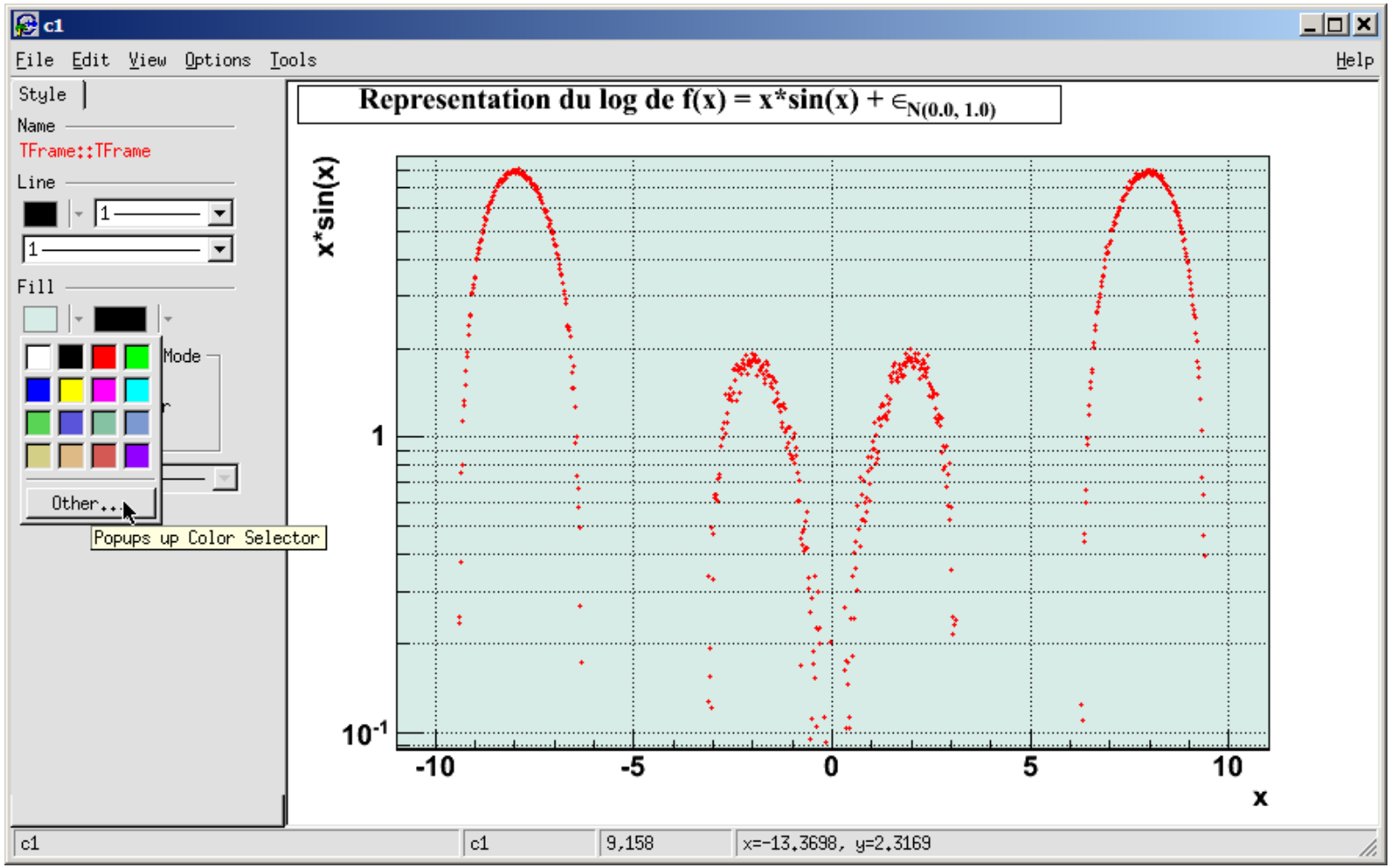
Producing plots and playing with style



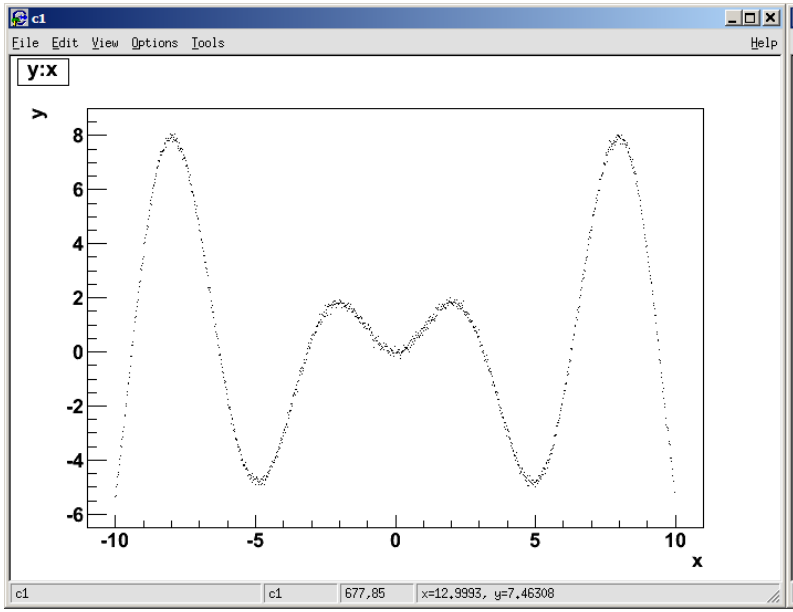
Producing plots and playing with style

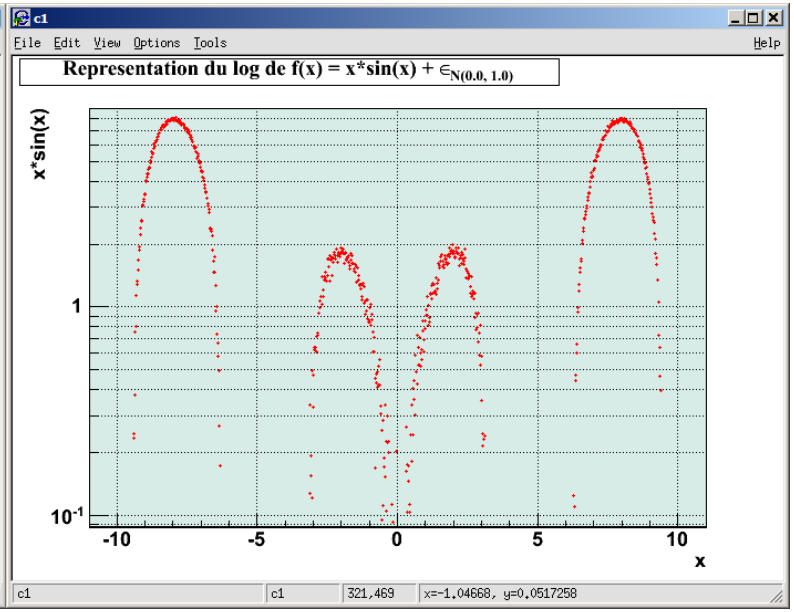


Producing plots and playing with style

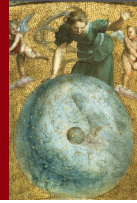


Producing plots and playing with style



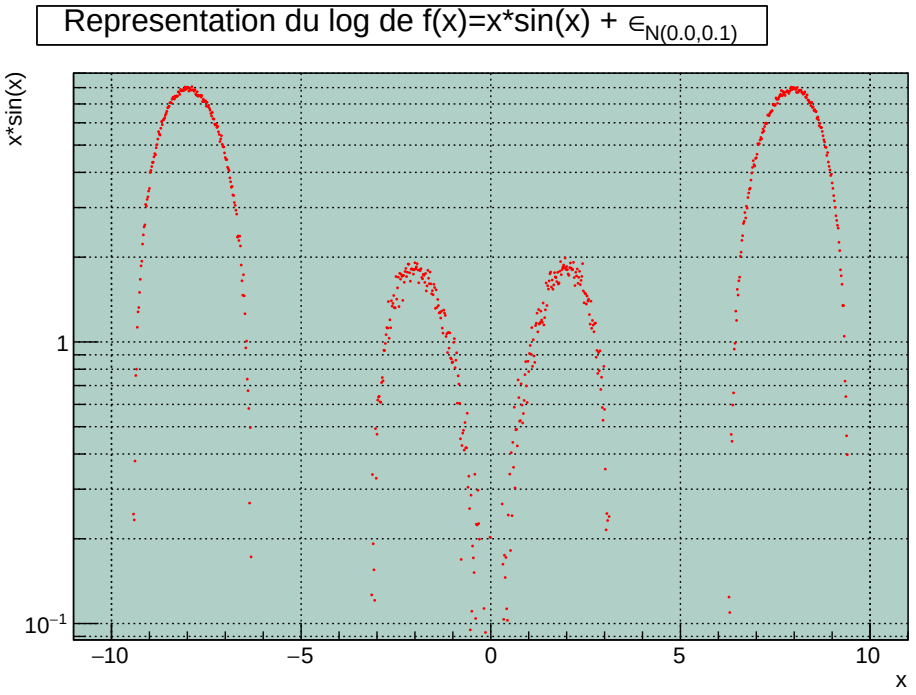


Code lines to do the same

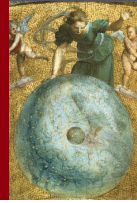


Everything can be done with code instructions, in a script.

```
// Drawing
gStyle->SetTitleX(0.1); // Title Start
gStyle->SetTitleBorderSize(1); // Border around title
// Create a canvas
TCanvas *Can = new TCanvas("can","can",1);
myTree->Draw("y:x"); // Draw the graph

// Change Canvas property
Can->SetLogy(); // Put log scale on y
Can->SetGrid(); // Add the grid
// Put the weird background color
Can->GetFrame()->SetFillColor(29);

// Get the graph object
TGraph *gr = (TGraph*)Can->GetPrimitive("Graph");
// Change marker color and size
gr->SetMarkerColor(2);
gr->SetMarkerStyle(6);

// Get the histogram object
TH2F *h = (TH2F*)Can->GetPrimitive("htemp");
// Change title and Y axis title
h->SetTitle("Representation du log de f(x)=x*sin(x)
+ #epsilon_{N(0.0,0.1)}");
h->GetYaxis()->SetTitle("x*sin(x)");
```



Many sources for documentation



Online

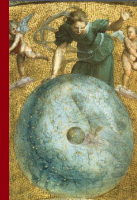
- Reference guide: https://root.cern.ch/root/html534/ClassIndex.html
  - ➙ Details all the methods (inherited or not) of a given class
- User-guide: https://root.cern.ch/root/html534/guides/users-guide/ROOTUsersGuideA4.pdf
  - ➙ Description of what can be done from installation to high level usage. Nicely 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

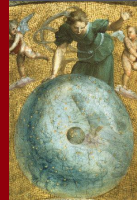
- User guide and manual: They are provided in markdown, ready to be compiled
  - ➙ $ROOTSYS/documentation/users-guide and $ROOTSYS/documentation/primer
- Tutorials: plenty of examples to be run
  - ➙ $ROOTSYS/tutorials
- Macros: place to store your own macros that you might call from anywhere
  - ➙ $ROOTSYS/macros

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

Outline



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

General overview: version 3.12 (1/2)


Fabrice Gaudier


Jean-Marc Martinez


Gilles Arnaud


Guillaume Damblin


J-B. Blanchard

General description:

- ROOT version: 5.34.36
- 11 modules / 246 classes
  $\sim$ 134 000 lines of code
- Compilation using CMAKE
- OpenSource since 2013/05

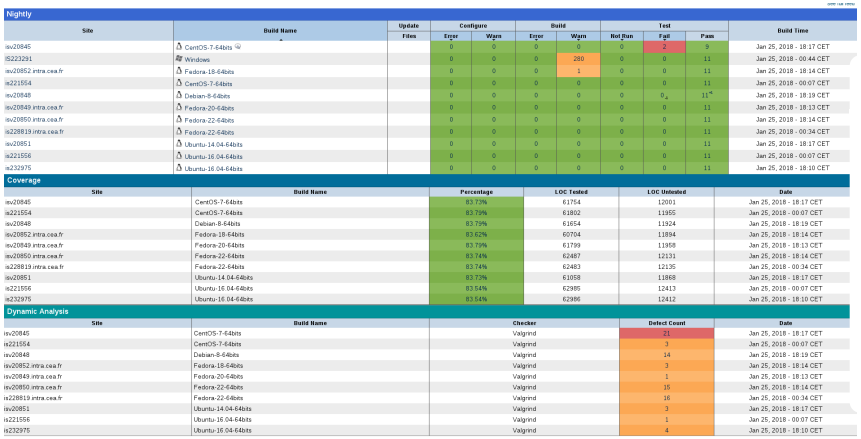
Regularly tested:

- 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

Unit Testing Report

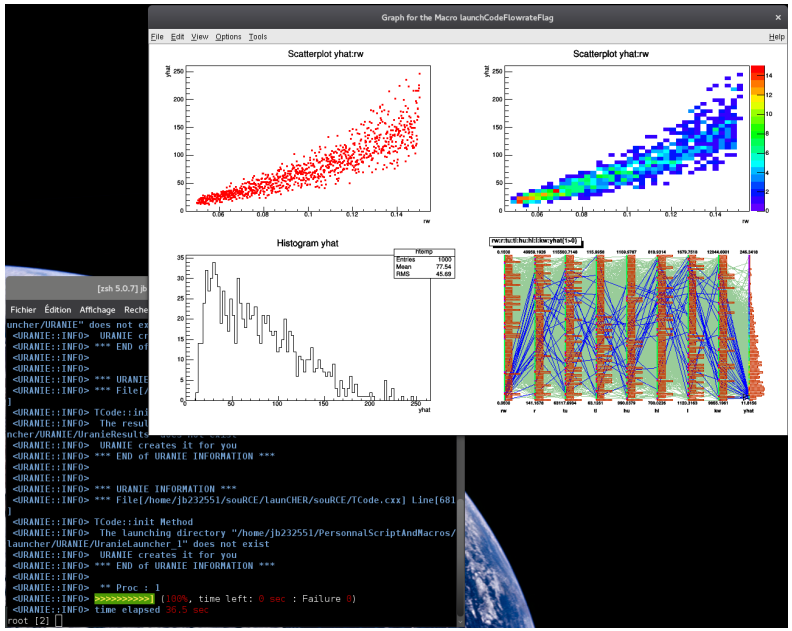
	DataServer	Launcher	Relauncher	Sampler	Sensitivity	Optimizer	reOptimizer	Modeler	UncertModeler	reLiability	XMLProblem
Status	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED
Duration											
Num. test	328	112	39	176	115	139	46	429	53	2	13
Total Failures	0	0	0	0	0	0	0	0	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

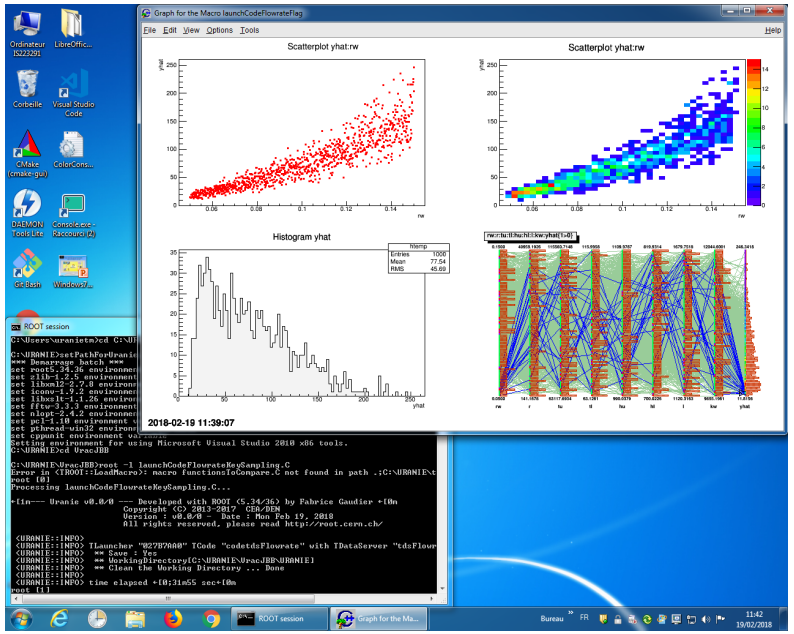
General overview: version 3.12 (2/2)



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

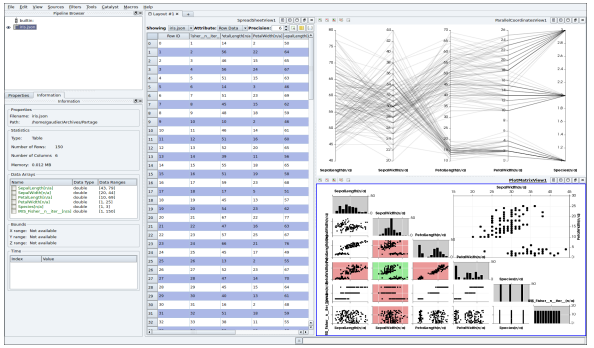




Communication with other platforms

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

Documentation

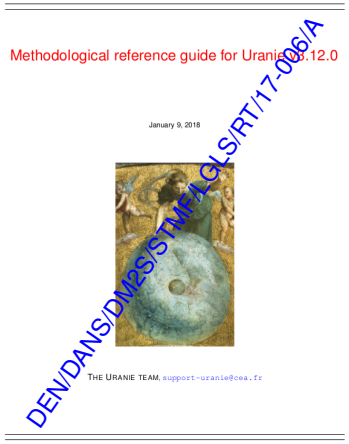
Three different levels

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)
  - ➙ Standalone code and use-cases macros are tested.
  - ➙ Plots are reproduced and compared to reference.

Developer's guide using Doxygen (HTML only)

- describing methods from comments in the code



Test UseCases Macros of UserManual Report

Date : 2018-01-10_09:48:51

OS : Ubuntu 16.04.3 LTS with (is221556)

ROOT version : 5.34/36

Global success rate : 849/ 849/ 849

Id	Macro Name	Status	Success	Failures
1	ChangeCopule.C	OK	4/4	0/4
2	ChangeField.C	OK	2/2	0/2
3	ChangeStochaDistribution.C	OK	16/16	0/16
4	launchCodeFlowrateFlagFailure.C	OK	13/13	0/13
5	launchCodeFlowrateFlagOATMinMax.C	OK	12/12	0/12
6	launchCodeFlowrateFlagSamplingKey.C	OK	11/11	0/11
7	launchCodeFlowrateFlagSampling.C	OK	11/11	0/11
8	launchCodeFlowrateKeyDataBase.C	OK	11/11	0/11
9	launchCodeFlowrateKeyFailure.C	OK	13/13	0/13
10	launchCodeFlowrateKeyFlagSampling.C	OK	11/11	0/11
11	launchCodeFlowrateKeyOATMinMax.C	OK	12/12	0/12
12	launchCodeFlowrateKeyRecreateSamplingOutputDataServer.C	OK	11/11	0/11
13	launchCodeFlowrateKeyRecreateSampling.C	OK	11/11	0/11
14	launchCodeFlowrateKeySamplingKey.C	OK	11/11	0/11

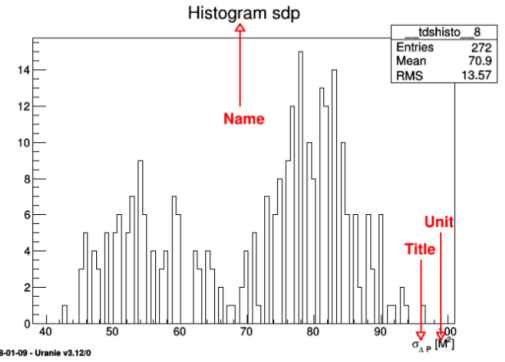
- Name + title: constructor defined from the name and the title of the variable

```
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()** pr
the unit. In this case, by default, the field *key* is identical to the field *name*. We will use the ability given by
to write LaTeX expressions in graphics to improve graphics rendering without weighing down the manipula
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



Documentation

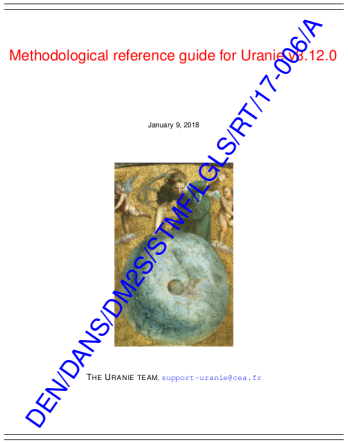
Three different levels

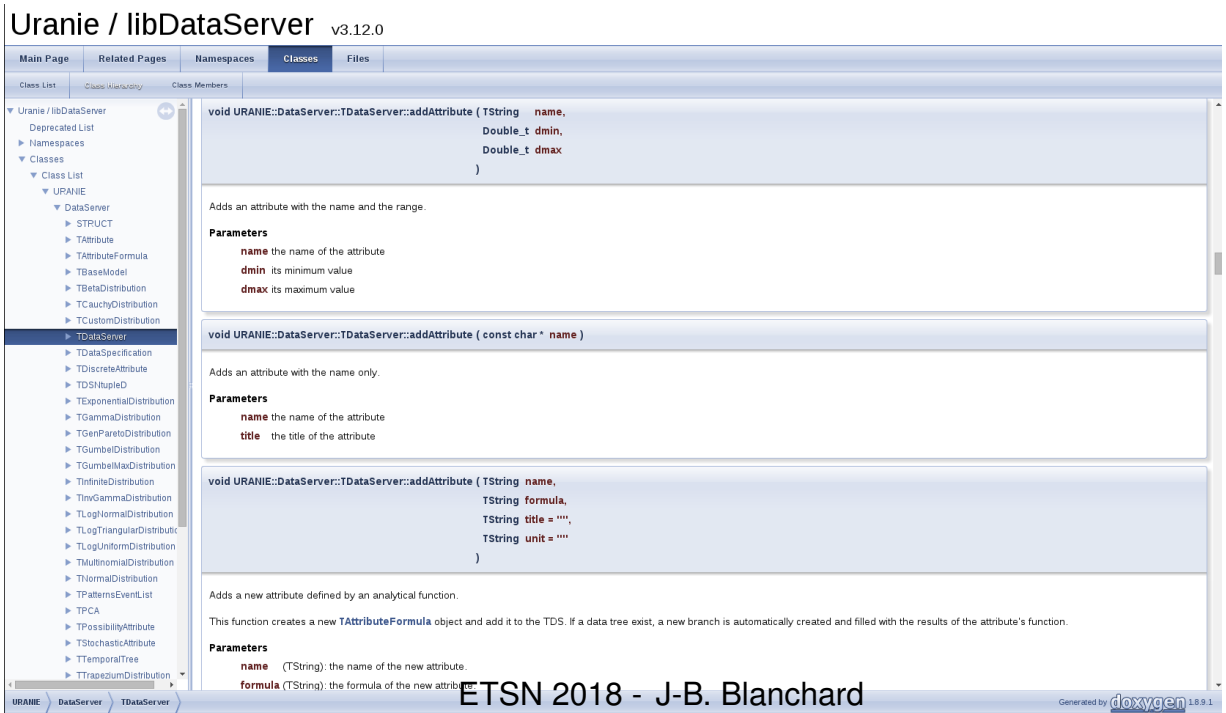
2 using DocBook, generating both PDF and HTML formats.

- 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)
  - → Standalone code and use-cases macros are tested.
  - → Plots are reproduced and compared to reference.

Developer's guide using Doxygen (HTML only)

- describing methods from comments in the code





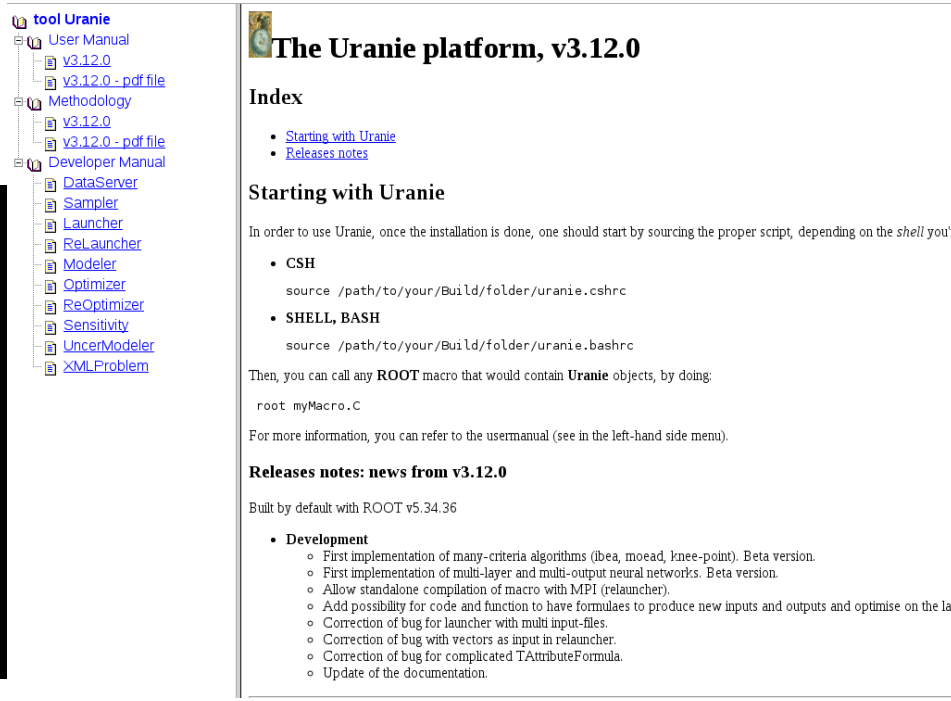
Accessing this material

Once Uranie is installed from the archive

As for ROOT, $URANIESYS is defined and used:

- to point out toward documentation
  - ➙ $URANIESYS/share/uranie/index.html
- to point out toward the use-case macros
  - ➙ $URANIESYS/share/uranie/macros

```
bash-4.3$ source /home/uranie/tools/fedora22/root5.34.36/URANIE3.12.0/uranie.bashrc
**************************
**
**  Environment variables for Uranie v3.12.0
**
**  URANIESYS : /home/uranie/tools/fedora22/root5.34.36/URANIE3.12.0
**   with
**      ROOT   : v5.34/36
**      FFTW libs    : /home/uranie/tools/fedora22/FFTW/lib
**      OPT++ libs   : /home/uranie/tools/fedora22/OPT++/lib
**      JSONCPP libs   : /usr/lib64
**      MPI libs   : /usr/lib64/openmpi/lib
**      NLOPT lib    : /home/uranie/tools/fedora22/NLOPT/lib
**      PCL lib      : /home/uranie/tools/fedora22/PCL/lib
**************************

Documentation can be seen with a web browser at:
file:///home/uranie/tools/fedora22/root5.34.36/URANIE3.12.0/share/uranie/index.html

Use-case macros can be found here:
/home/uranie/tools/fedora22/root5.34.36/URANIE3.12.0/share/uranie/macros
bash-4.3$
```



These paths are reminded at every setup when Uranie is installed from archive (from v3.12)

The module point of view

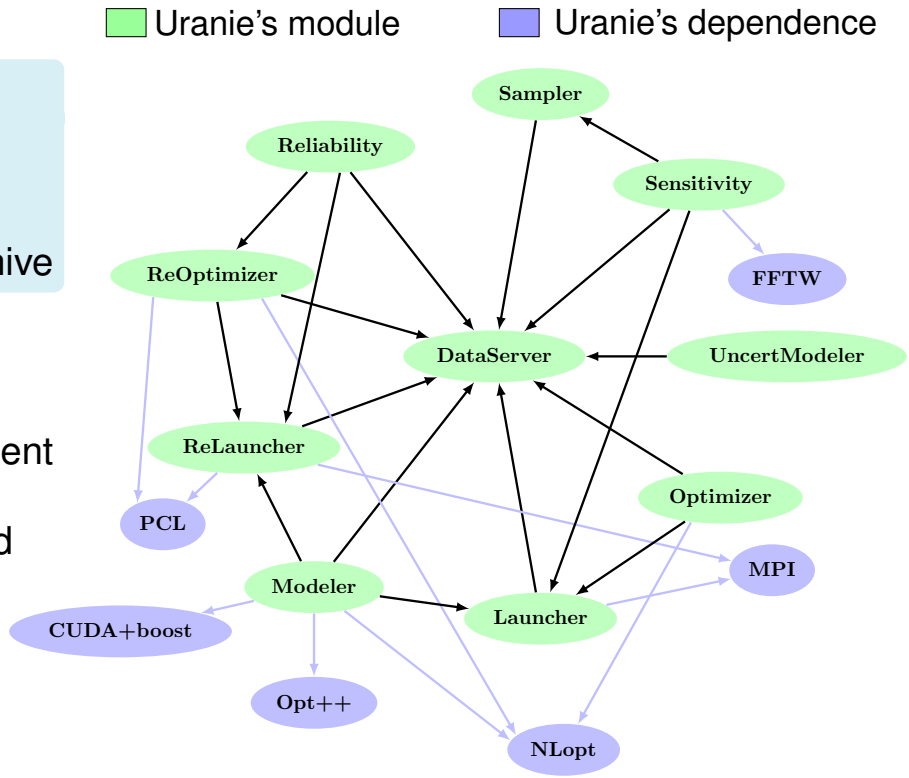
Few dependencies:

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

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

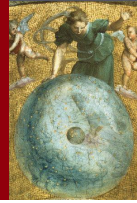
Organised in modules:

- 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

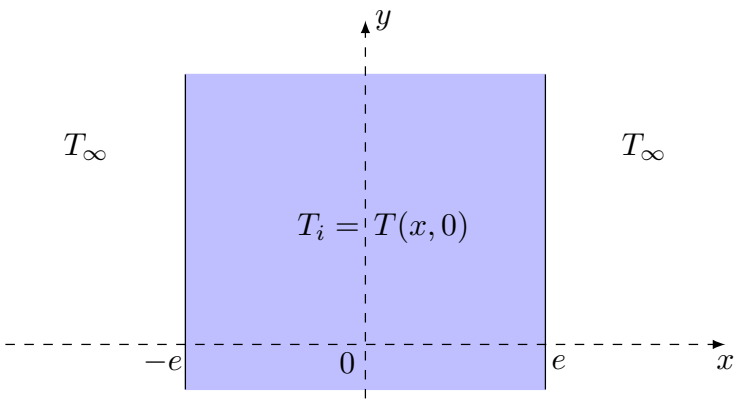


The next following slides will discuss the content of the main dedicated modules

Outline



Use-case: the thermal exchange model



Experimental setup and goals:

- Planar sheet of width $2e$ (infinite in y).
  - ➡ Initial temperature is T_i (homogeneous)
- t=0, sheet is exposed to warmer fluid (T_∞)

Aim:

- Find out the thermal gauge $\theta(x,t) = \dfrac{T(x,t) - T_i}{T_\infty - T_i}$
- Get the thermal exchange coefficient h [W.m^{-2}.K^{-1}] through the Biot number B_i.

Theoretically, under certain hypothesis

$$\theta(x_{ds}, t_{ds}) = 2\sum_{n=1}^{\infty} \beta_n \cos(\omega_n x_{ds}) \exp(-\frac{1}{4}\omega_n^2 t_{ds})$$

using dimensionless parameters

$$x_{ds} = x/e \text{ and } t_{ds} = \frac{t}{t_D} = t \times \frac{4\lambda}{e^2 \rho C_\rho}$$

$$\beta_n = \frac{\gamma_n \sin(\omega_n)}{\omega_n(\gamma_n + B_i)} \text{ and } \gamma_n = \omega_n^2 + B_i^2$$

and ω_n are solutions of the following equation

$$\omega_n \tan(\omega_n) = B_i$$

	Value	Uncertainty
Thickness [m] : e	10×10^{-3}	5×10^{-5}
Thermal conductivity [W.(m.K)$^{-1}$] : λ	0.25	1.5×10^{-3}
Massive thermal capacity [J.(kg.K)$^{-1}$] : C_ρ	1300	15.6
Volumic mass [kg.m^{-3}] : ρ	2200	4.4

Table: Summary of PTFE properties along with their uncertainty.

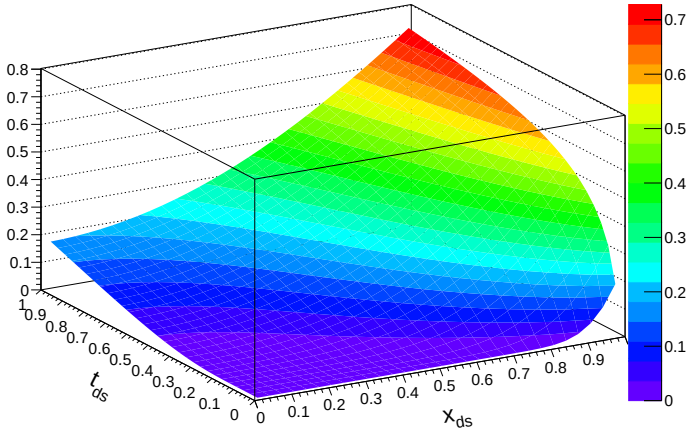
Uncertainties are taken mimicking Iron tabulated values.

Create a code to perform this estimation

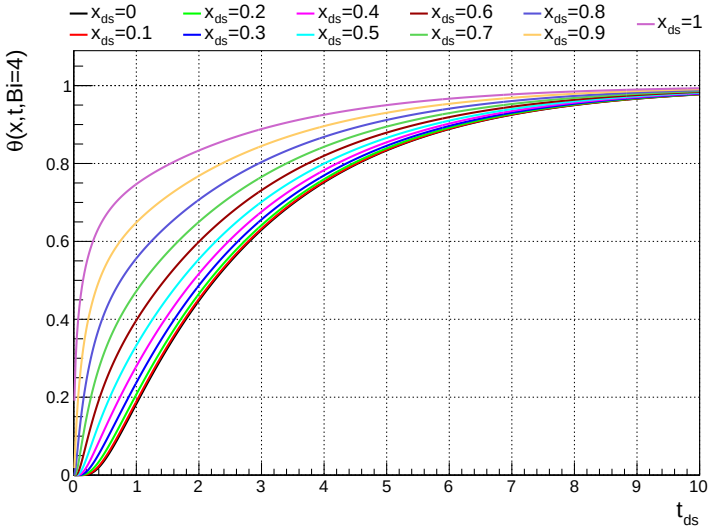
Using previous formula

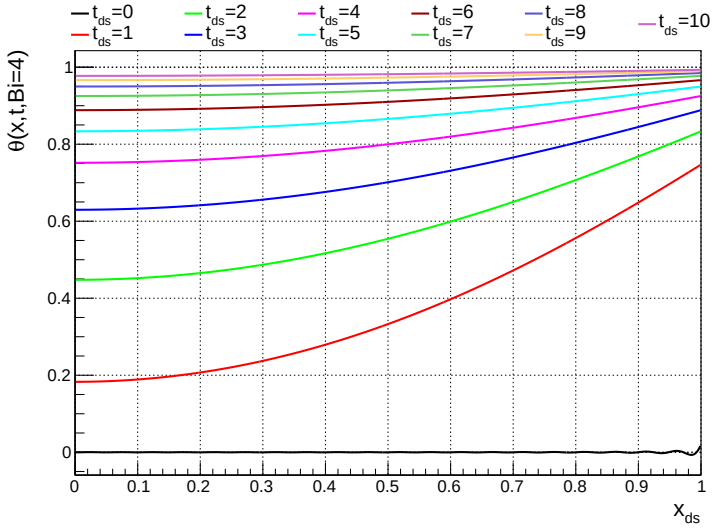
$$\theta(x_{ds}, t_{ds}) = 2 \sum_{n=1}^{\infty} \beta_n \cos(\omega_n x_{ds}) \exp(-\frac{1}{4}\omega_n^2 t_{ds})$$

- Choose a threshold on the infinite serie (here $M = 40$)
- Implement this as a C++ function with different levels
  - Compute θ for a single (x,t) configuration
  - Compute θ for vectors of x and t.

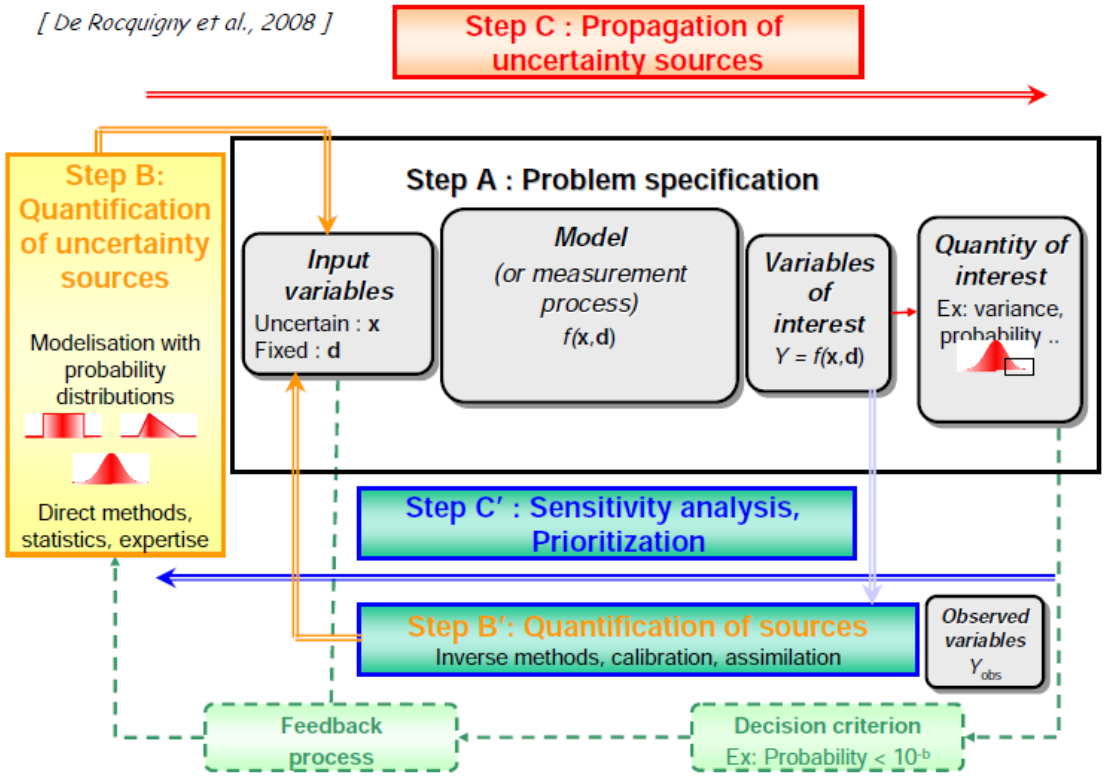


This code is rather fast, but we'll consider it as a large time/cpu consuming code





Workflow: breakdown into steps

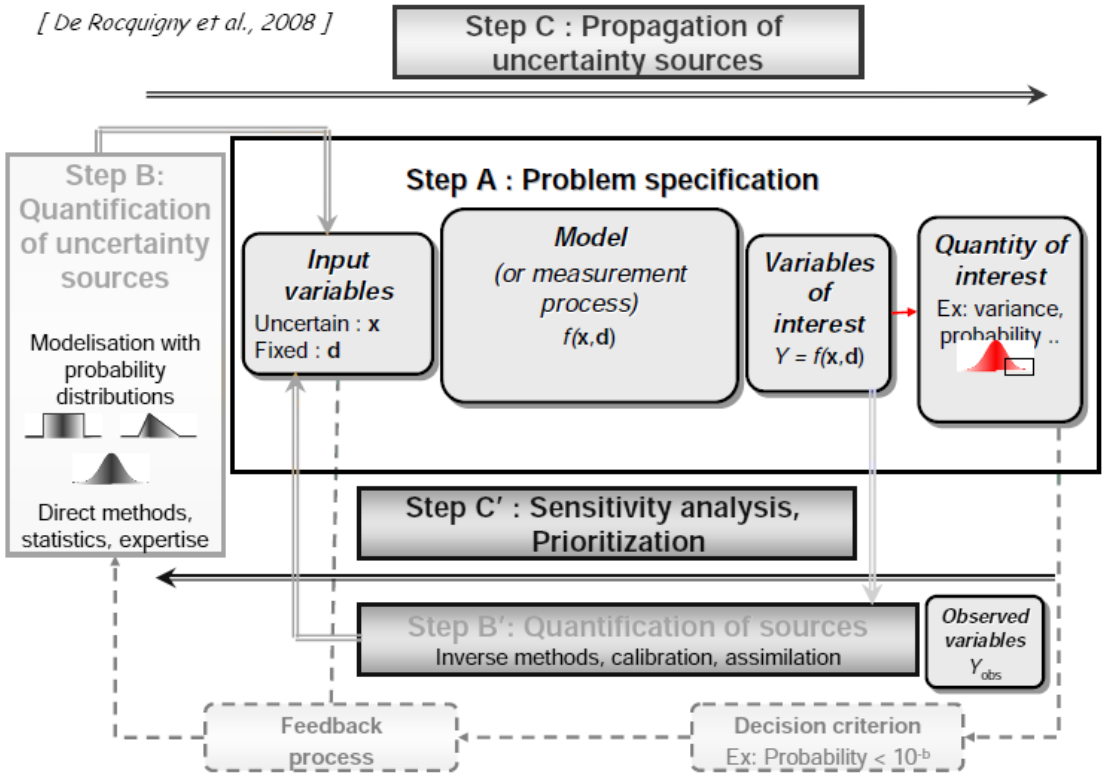


Main steps:

- 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

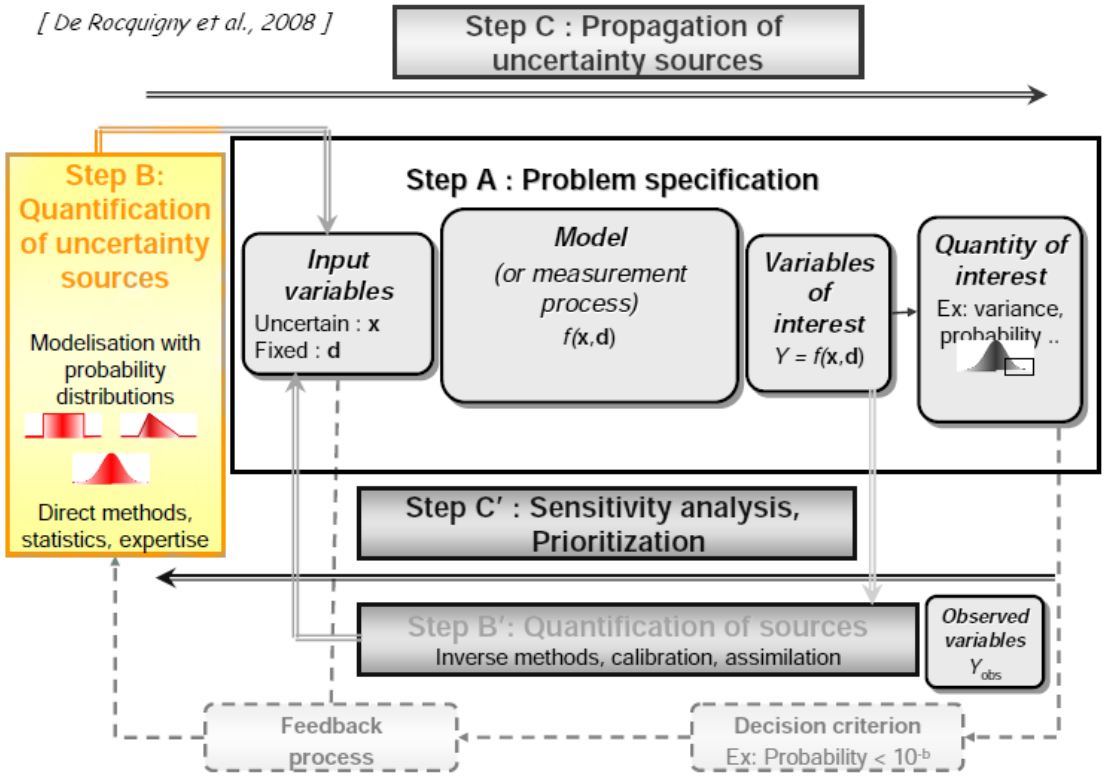


Main steps:

- 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

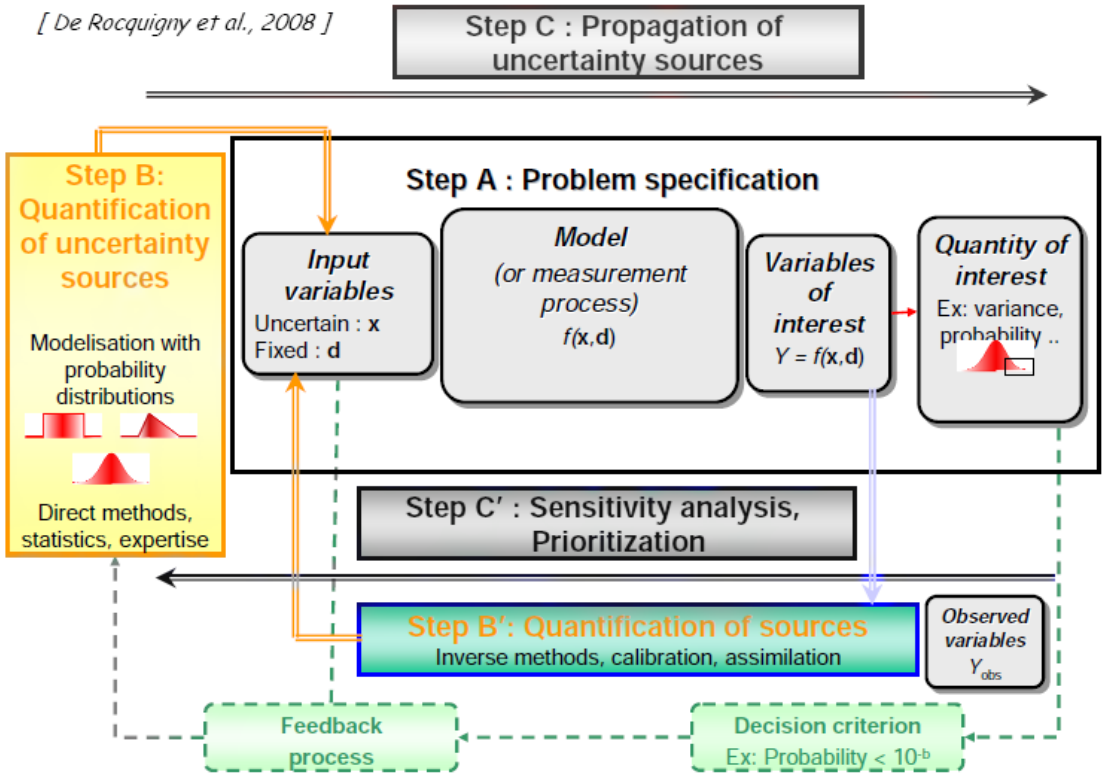


Main steps:

- 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

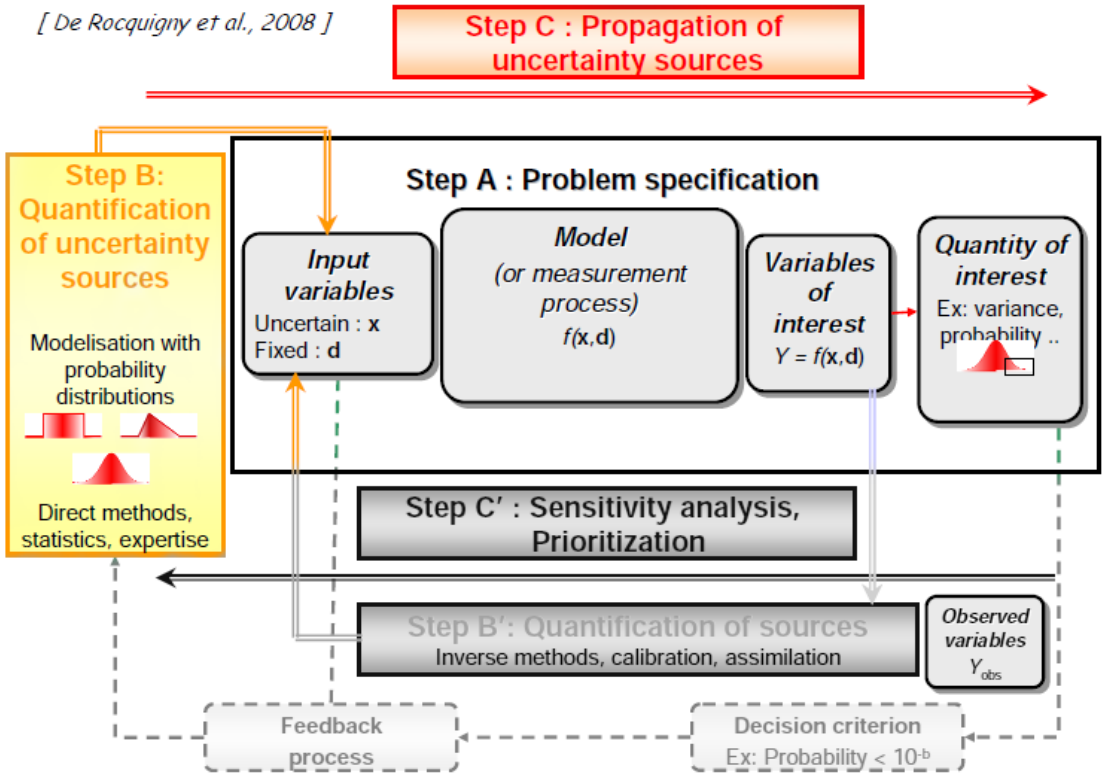


Main steps:

- 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

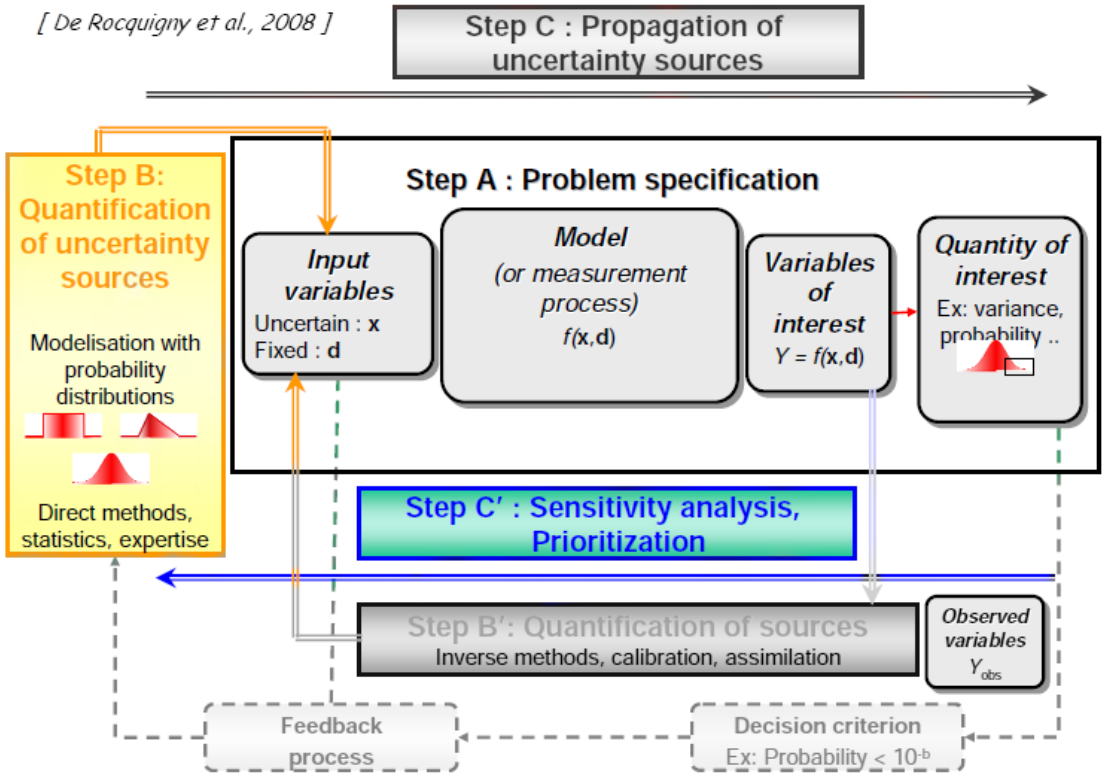


Main steps:

- 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

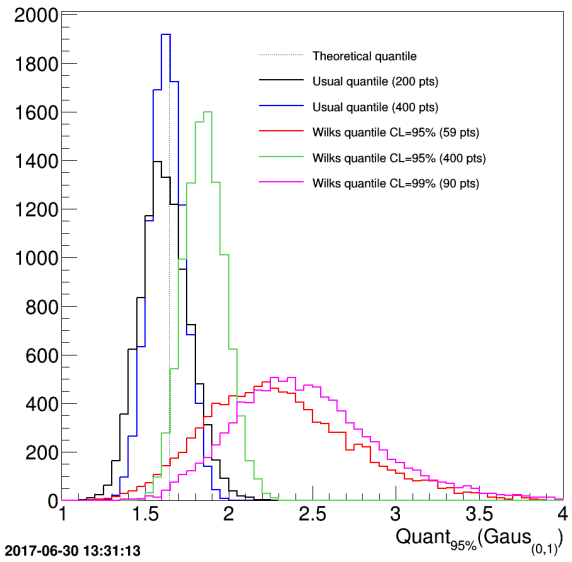
These steps are usually model dependent, it might be useful to iterate to help converging to proper conclusions

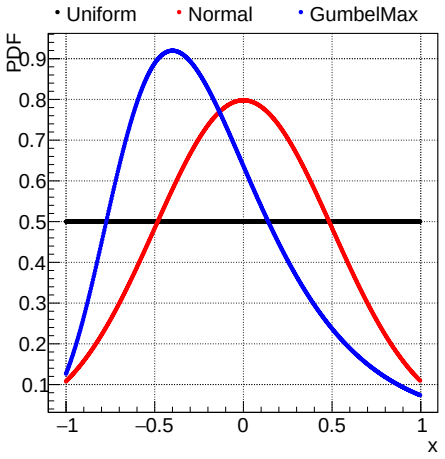
Outline

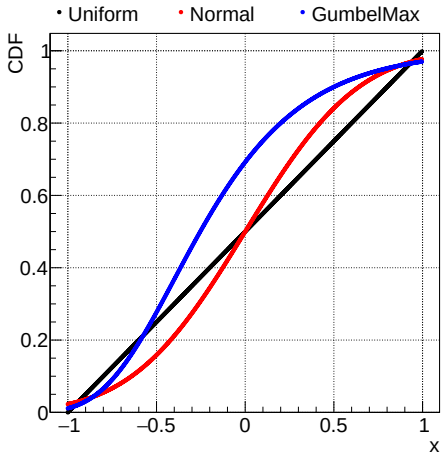
Dataserver module: create and handling variables

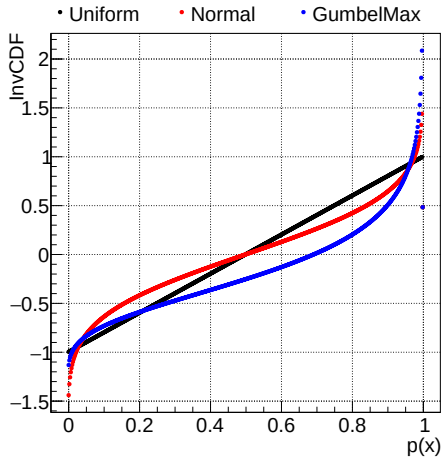
With the DataServer module, one can:

- create new variables using pre-defined statistical laws
- 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.

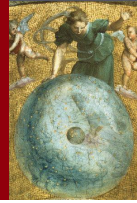








Dataserver module: import/export/represent data

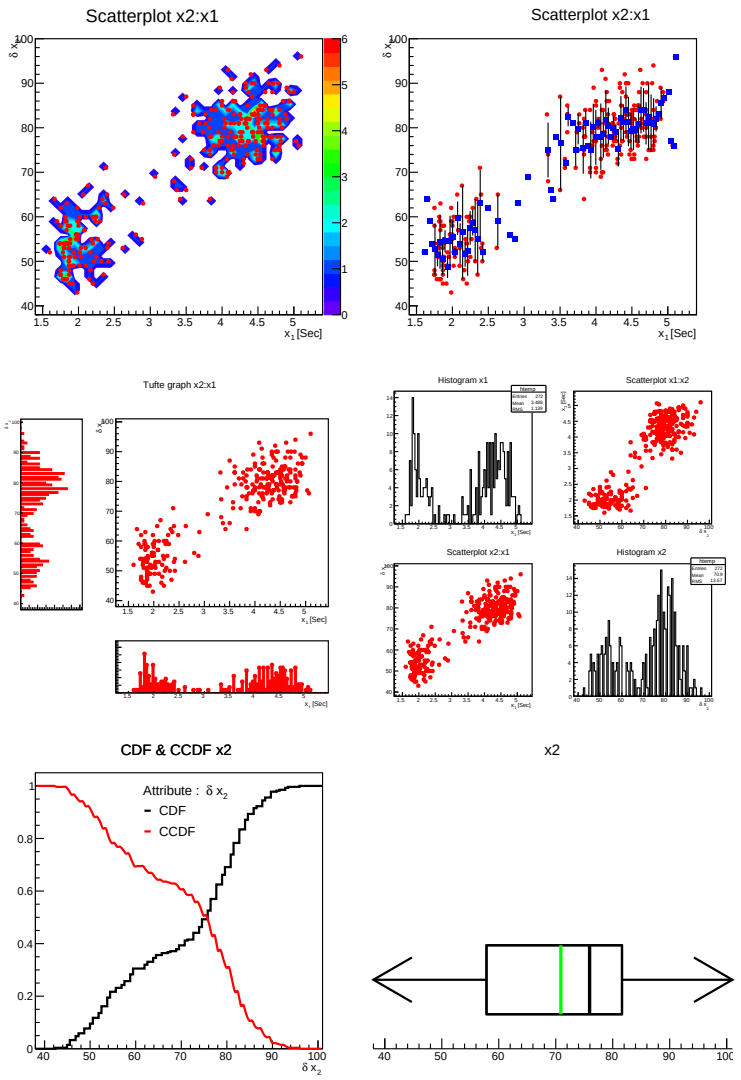


```
//Loading namespaces to get rid of complicated names
using namespace URANIE::DataServer;

//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");
```

```
#TITLE: geyser data
#COLUMN_NAMES: x1| x2
#COLUMN_TITLES: x_{1}| #delta x_{2}
#COLUMN_UNITS: Sec|

3.600 79.000
1.800 54.000
3.333 74.000
2.283 62.000
4.533 85.000
2.883 55.000
4.700 88.000
3.600 85.000
1.950 51.000
4.350 85.000
1.833 54.000
3.917 84.000
4.200 78.000
1.750 47.000
4.700 83.000
2.167 52.000
1.750 62.000
4.800 84.000
1.600 52.000
4.250 79.000
1.800 51.000
1.750 47.000
3.450 78.000
3.067 69.000
4.533 74.000
3.600 83.000
1.967 55.000
4.083 76.000
3.850 78.000
4.433 79.000
```

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

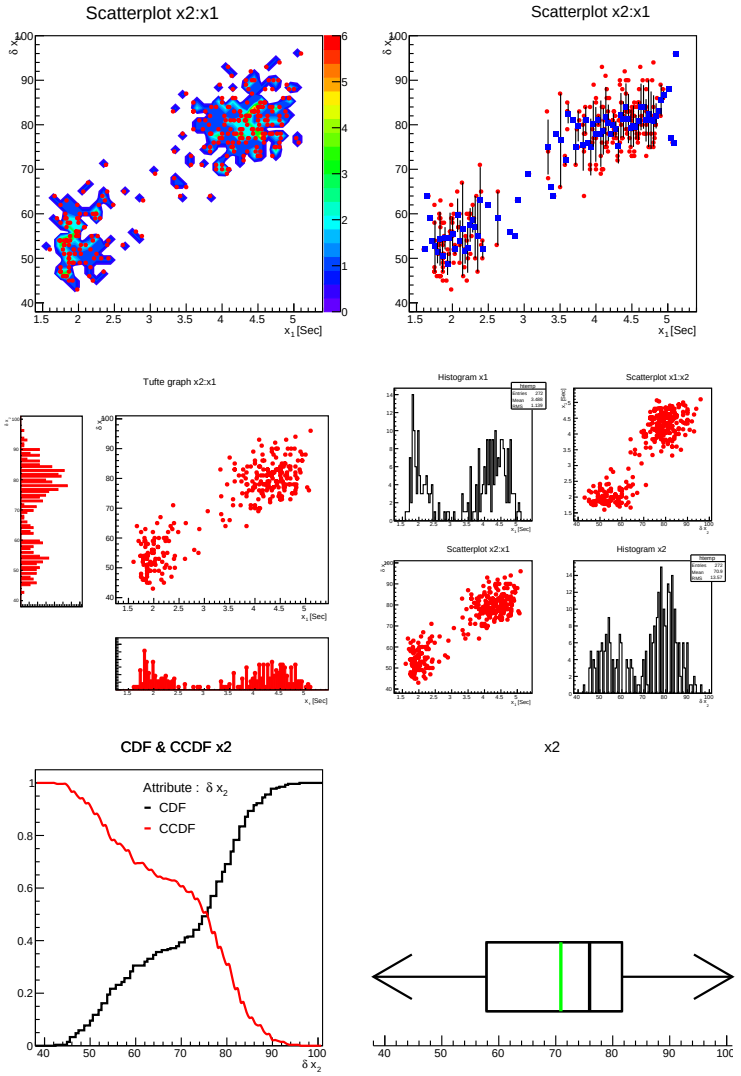
- interactively: `(%) root File.C`
- compiled:

```
(%) 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");
```



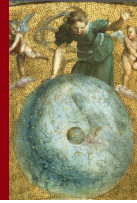
Can be used either

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

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

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

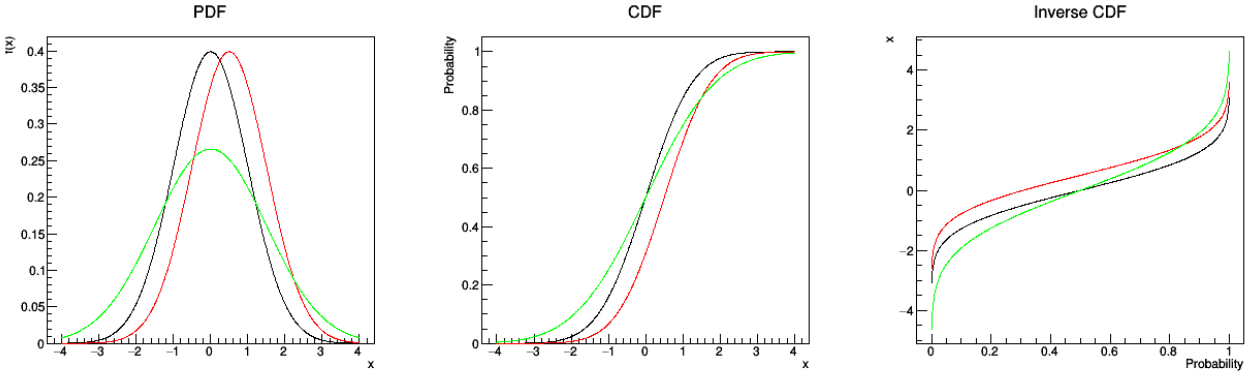
Defining stochastic variables I



A large number of stochastic variable can be implemented

Law	Class in Uranie	Adjustable parameters
Uniform	TUniformDistribution	Min, Max
Log-uniform	TLogUniformDistribution	Min, Max
Triangular	TTriangularDistribution	Min, Max, Mode
Log-triangular	TLogTriangularDistribution	Min, Max, Mode
Normal (gaussian)	TNormalDistribution	Mean (μ), Sigma (σ)
Log-normal	TLogNormalDistribution	Mean (μ), Sigma (σ)
Trapezium	TTrapeziumDistribution	Min, Max, Low, Up
Uniform by parts	TUniformByPartsDistribution	Min, Max, Median
Exponential	TExponentialDistribution	Rate (λ), Min
Cauchy	TCauchyDistribution	Scale (γ), Median
GumbelMax	TGumbelMaxDistribution	Mode (μ), Scale (β)
Weibull	TWeibullDistribution	Scale (λ) , Shape (k) , Min
Beta	TBetaDistribution	alpha (α) , beta (β) , Min, Max
GenPareto	TGenParetoDistribution	Location (μ), Scale (σ), Shape (ξ)
Gamma	TGammaDistribution	Shape (α) , Scale (β) , Location (ξ)
Inverse gamma	TInvGammaDistribution	Shape (α) , Scale (β) , Location (ξ)

Defining stochastic variables II

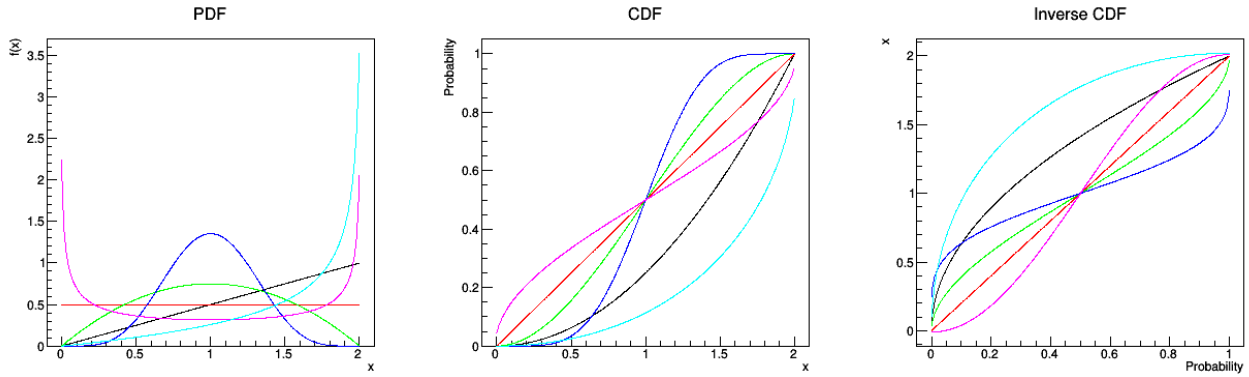


TNormal

μ=0.0; σ=1.0;
μ=0.5; σ=1.0;
μ=0; σ=1.5;

$$f(x) = e^{\frac{-(x-\mu)^2}{2\sigma^2}} \times \frac{1}{\sqrt{2\pi\sigma^2}}$$

2018-04-05 - Uranie v2016.2/10



TBeta

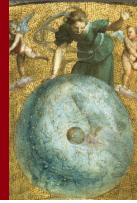
α=2.0; β=1.0; x_{min}=0.0; x_{max}=2.0;
α=1.0; β=1.0; x_{min}=0.0; x_{max}=2.0;
α=2.0; β=2.0; x_{min}=0.0; x_{max}=2.0;
α=6.0; β=6.0; x_{min}=0.0; x_{max}=2.0;
α=0.5; β=0.5; x_{min}=-0.01; x_{max}=2.01;
α=2.0; β=0.5; x_{min}=0.0; x_{max}=2.02;

$$f(x) = \frac{Y^{\alpha-1} \times (1-Y)^{\beta-1}}{B(\alpha,\beta)} \text{ for } x \in [x_{min}, x_{max}]$$

where $Y = \frac{(x-x_{min})}{(x_{max}-x_{min})}$ and $B(\alpha,\beta)$ is beta function

2018-04-05 - Uranie v2016.2/10

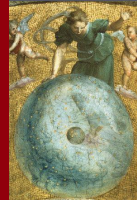
Operations on variables



```
TDataServer * tds = new TDataServer("Name", "Titre");
tds->fileDataRead("simpleNorm.dat");

//Create new variables
tds->addAttribute("double_y", "y * 2");
tds->addAttribute("y_even", "((y%2==0)? 1 : 0)");

tds->computeRank("y");  // Compute the rank of y's value

//Compute a global normalisation of y :
tds->normalize("y","GZO",TDataServer::kZeroOne); // From 0 to 1
tds->normalize("y","GCent",TDataServer::kCentered); // 0-Centered
tds->normalize("y","MOO",TDataServer::kMinusOneOne); // From -1 to 1

tds->Scan("*"); // Dump results on screen

// Compute the correlation matrix and dump it as well
TMatrixD mat = tds->computeCorrelationMatrix("y:x:y_even");
mat.Print();
```

```
#NAME: cho
#COLUMN_NAMES: x|y
#COLUMN_TYPES: D|D

0 0
1 1
2 4
3 9
4 16
5 25
```

Operations on variables

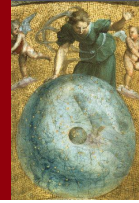


```
TDataServer * tds = new TDataServer("Name", "Titre");
tds->fileDataRead("simpleNorm.dat");

//Create new variables
tds->addAttribute("double_y", "y * 2");
tds->addAttribute("y_even", "((y%2==0)? 1 : 0)");

tds->computeRank("y");  // Compute the rank of y's value

//Compute a global normalisation of y :
tds->normalize("y","GZO",TDataServer::kZeroOne); // From 0 to 1
tds->normalize("y","GCent",TDataServer::kCentered); // 0-Centered
tds->normalize("y","MOO",TDataServer::kMinusOneOne); // From -1 to 1

tds->Scan("*"); // Dump results on screen

// Compu
TMatrixD
mat.Prin
```

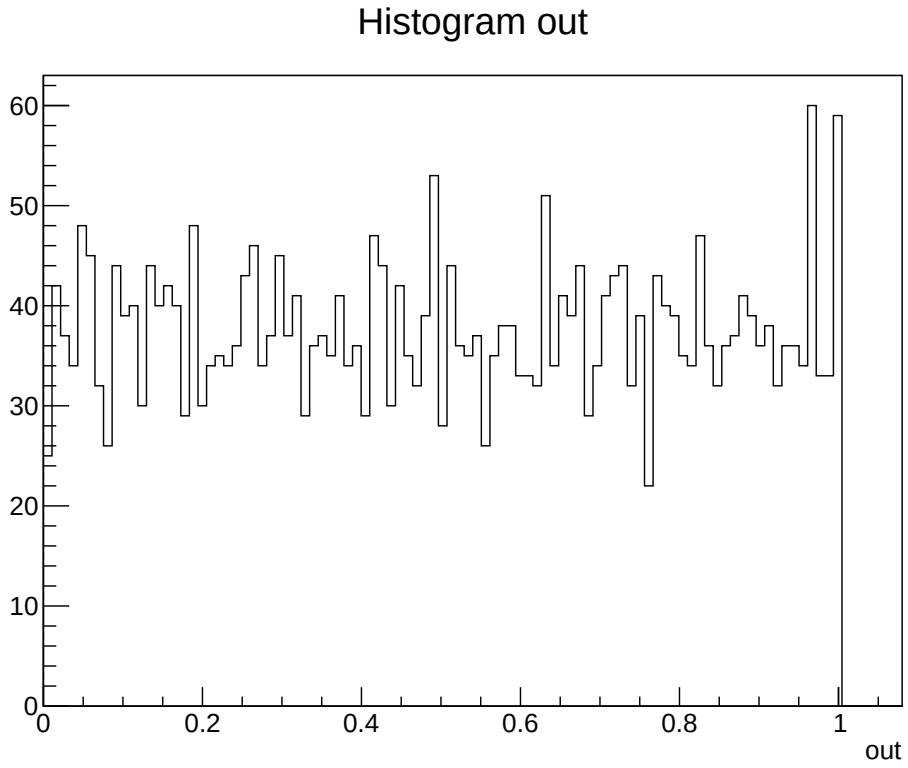
```
#NAME: cho
#COLUMN_NAMES: x|y
#COLUMN_TYPES: D|D

0 0
1 1
2 4
3 9
4 16
5 25
```

```
bash-4.3$ root -q -l operationSimple.C
root [0]
Processing operationSimple.C...

--- Uranie v2016.2/10 --- Developed with ROOT (5.34/36) by Fabrice Gaudier
                    Copyright (C) 2013-2017  CEA/DEN
                    Version : v2016.2/10 -  Date : Tue Feb 09, 2016
                    All rights reserved, please read http://root.cern.ch/

*****************************************************************************************************************
*    Row   * Name__n__ *         x *         y *  double_y *    y_even *      Rk_y *      yGZO *    yGCent *      yMOO *
*****************************************************************************************************************
*        0 *         1 *         0 *         0 *         0 *         1 *         1 *         0 * -9.166667 *        -1 *
*        1 *         2 *         1 *         1 *         2 *         0 *         2 *      0.04 * -8.166667 *     -0.92 *
*        2 *         3 *         2 *         4 *         8 *         1 *         3 *      0.16 * -5.166667 *     -0.68 *
*        3 *         4 *         3 *         9 *        18 *         0 *         4 *      0.36 * -0.166667 *     -0.28 *
*        4 *         5 *         4 *        16 *        32 *         1 *         5 *      0.64 *  6.833333 *      0.28 *
*        5 *         6 *         5 *        25 *        50 *         0 *         6 *         1 * 15.833333 *         1 *
*****************************************************************************************************************

3x3 matrix is as follows

     |      0    |      1    |      2    |
-------------------------------------------
   0 |          1      0.9599      -0.281
   1 |     0.9599           1     -0.2928
   2 |     -0.281     -0.2928           1
```

Importance of the visualisation



Before getting in into complicated analysis/methods it is always a good idea to visualise data to check basic hypothesis and expectation.

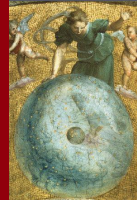


Simple Plot

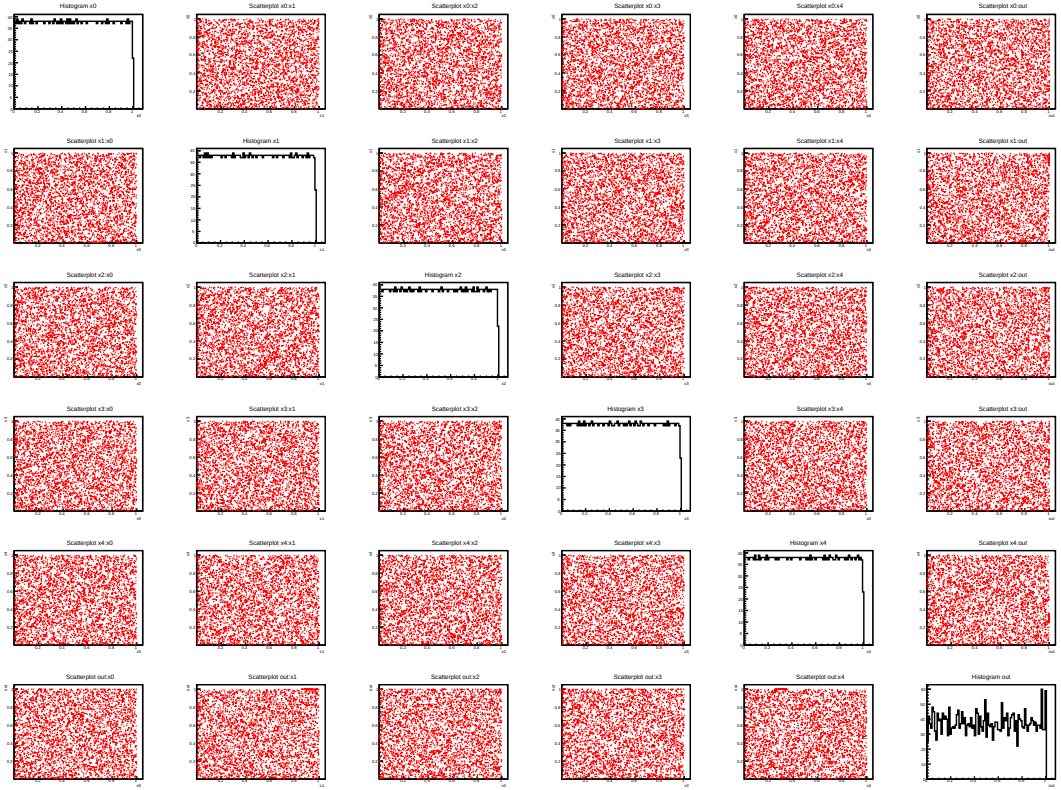
Simple illustration

- 5 inputs ($\mathcal{U}(0,1)$)
- 1 output (between 0 and 1)
- No trend in 2×2 correlation matrix
- No trend in the parallel plot..
- ... but when focusing on specific part

Importance of the visualisation



Before getting in into complicated analysis/methods it is always a good idea to visualise data to check basic hypothesis and expectation.

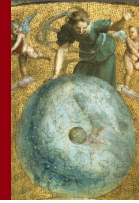


drawPair

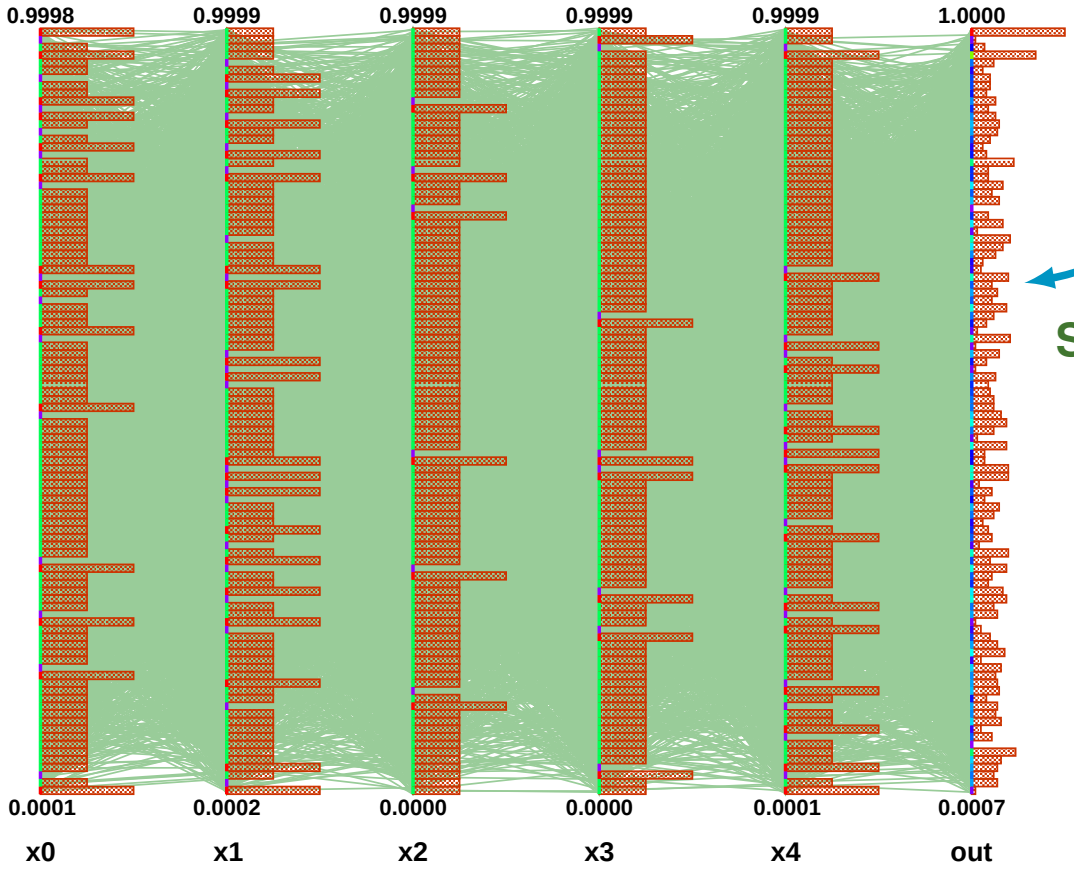
Simple illustration

- 5 inputs ($\mathcal{U}(0,1)$)
- 1 output (between 0 and 1)
- No trend in 2×2 correlation matrix
- No trend in the parallel plot..
- ... but when focusing on specific part

Importance of the visualisation



Before getting in into complicated analysis/methods it is always a good idea to visualise data to check basic hypothesis and expectation.



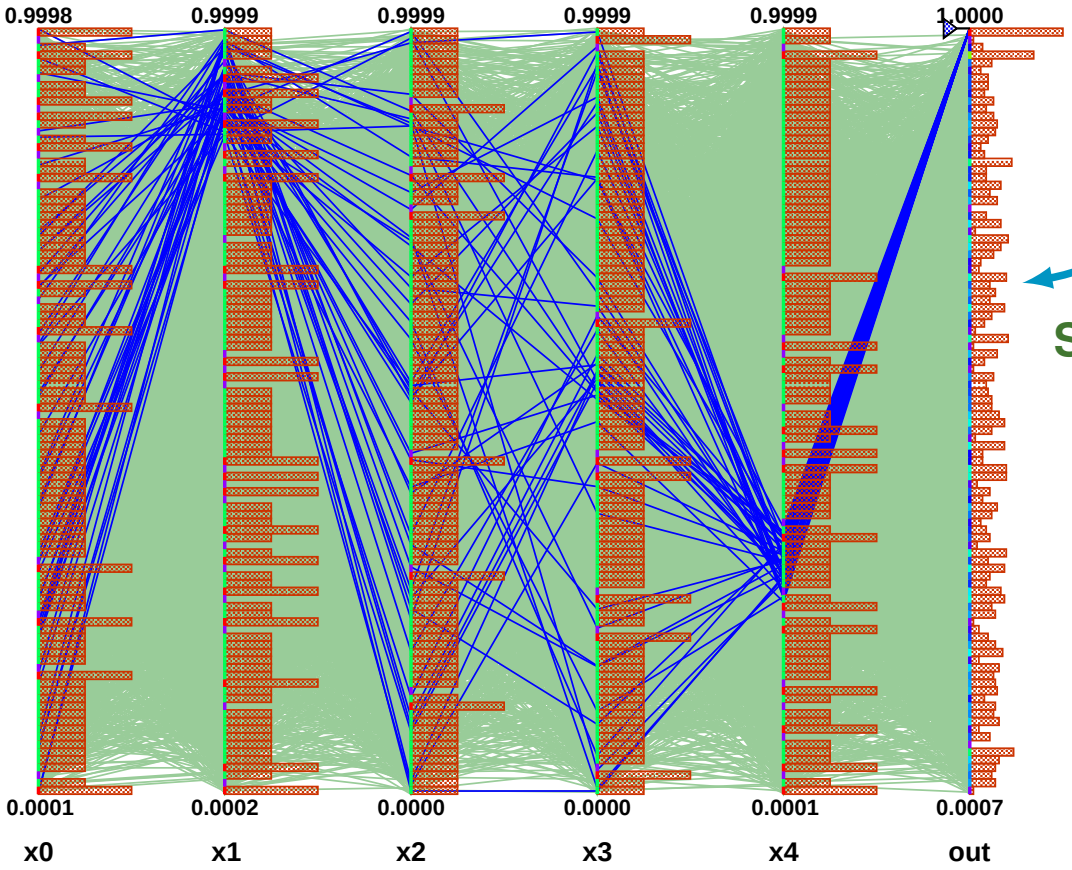
Parallel plot or Cobweb

Simple illustration

- 5 inputs ($\mathcal{U}(0,1)$)
- 1 output (between 0 and 1)
- No trend in 2×2 correlation matrix
- No trend in the parallel plot..
- ... but when focusing on specific part

Importance of the visualisation

Before getting in into complicated analysis/methods it is always a good idea to visualise data to check basic hypothesis and expectation.



Parallel plot or Cobweb

Simple illustration

- 5 inputs ($\mathcal{U}(0,1)$)
- 1 output (between 0 and 1)
- No trend in 2×2 correlation matrix
- No trend in the parallel plot..
- ... but when focusing on specific part

Outline

General discussion: introducing the concepts

Uranie's approach

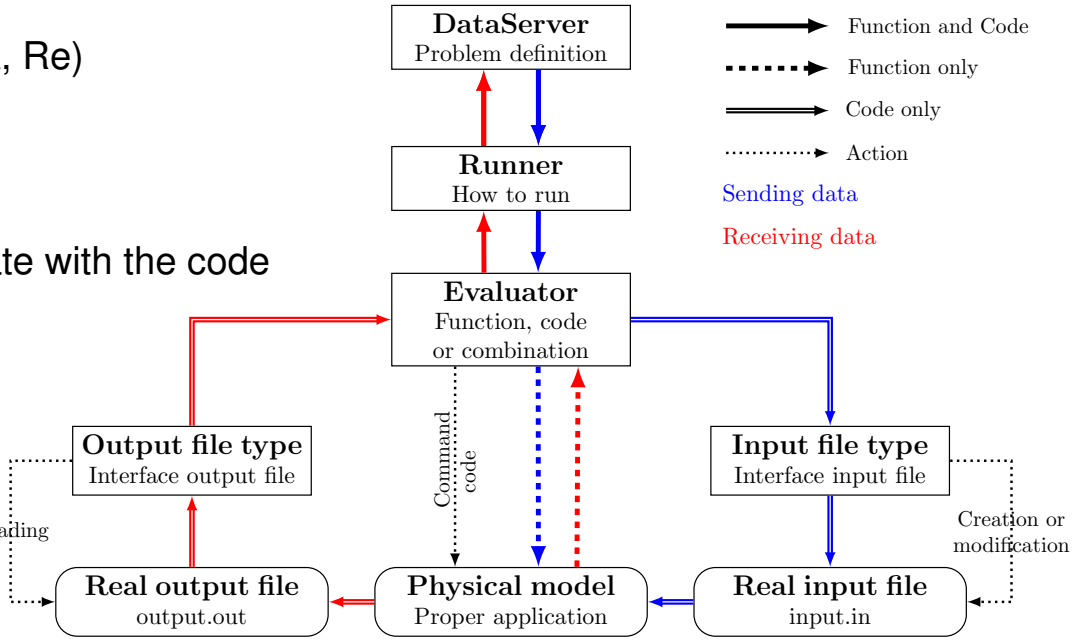
- Non-intrusive: code is a black box that cannot be modified but for some allowed parameters
- Two implementations (historical) from two different perspective, allowing redundancy
  - Launcher: (La)
  - Relauncher: (Re)

Nature of Evaluators

- C++ function interpreted through CInt (La, Re)
- C++ compiled function (Re)
- python function (Re)
- external code (La, Re)
  - → Need input / output files to communicate with the code

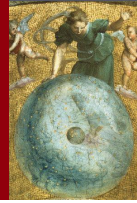
Ways of submitting jobs

- Sequentially (La, Re)
- Forking the code (La)
- Shared-memory distribution `pthread` (Re)
- Split-memory distribution `mpirun` (Re)
- Distributed on certain clusters (La)



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

Starting with a function



Type of functions

- C++ interpreted function
  - Very simple to implement
  - Inputs and outputs are only double-value
  - Interpreted so slow if it contains loop
- Python function (Re)
  - Can deal with double, vectors and strings
  - Can only be launch from python interface
  - Interpreted so slow if it contains loop
- C++ compiled function (Re)
  - Compiled ⇒ logic and speed of C++ conserved
  - Can deal with double, vectors and strings
  - To do this, a complicated structure should be used (no example provided, contact us)

Working with it

1. Evaluator object is constructed
2. Inputs are provided in the correct order
3. Outputs are provided in the correct order
4. Evaluator provided to the Runner

```
void MyFunction(double *in, double *out)
{
    double par1 = in[0], par2=in[1]; // ...
    // Many interesting operations

    // ...

    // as many as requested outputs
    out[0] = ...;
    out[1] = ...;
}
```

It can be defined

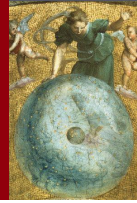
- in the same file
- in another file

```
// Loading function from another file
gROOT->LoadMacro("functionfile.C");
```

Caution:

- Cannot be used with shared-memory distribution

Starting with a function



Type of functions

- C++ interpreted function
  - Very simple to implement
  - Inputs and outputs are only double-value
  - Interpreted so slow if it contains loop
- Python function (Re)
  - Can deal with double, vectors and strings
  - Can only be launch from python interface
  - Interpreted so slow if it contains loop
- C++ compiled function (Re)
  - Compiled ⇒ logic and speed of C++ conserved
  - Can deal with double, vectors and strings
  - To do this, a complicated structure should be used (no example provided, contact us)

Working with it

1. Evaluator object is constructed
2. Inputs are provided in the correct order
3. Outputs are provided in the correct order
4. Evaluator provided to the Runner

```python
def PythonFunction(arg1, arg2, arg3) :

    # Random operation on the argument
    vect = [ i* arg1 for i in range(3)]
    doub = 2*arg2
    stri = ",".join(arg3) # arg3 is a list

    # Always return in a list
    return [vect, stri, doub]
```

It can be defined

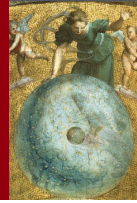
- in the same file
- in another file

```python
# Loading function from functionfile.py
from functionfile import PythonFunction
```

Caution:

- Unless otherwise specified, all argument are assumed to be double
- Cannot be used with shared-memory distribution

Starting with a function



Type of functions

- C++ interpreted function
  - Very simple to implement
  - Inputs and outputs are only double-value
  - Interpreted so slow if it contains loop
- Python function (Re)
  - Can deal with double, vectors and strings
  - Can only be launch from python interface
  - Interpreted so slow if it contains loop
- C++ compiled function (Re)
  - Compiled ⇒ logic and speed of C++ conserved
  - Can deal with double, vectors and strings
  - To do this, a complicated structure should be used (no example provided, contact us)

Working with it

1. Evaluator object is constructed
2. Inputs are provided in the correct order
3. Outputs are provided in the correct order
4. Evaluator provided to the Runner

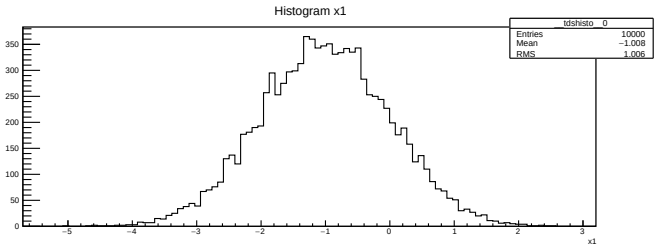
```
int MyCompicatedFunction(
    std::vector<UEntry*> *in,
    std::vector<UEntry*> *out
    )
{

    // Not discussed here :D

    return getOutDimension(out); // Not
       discussed as well
}
```

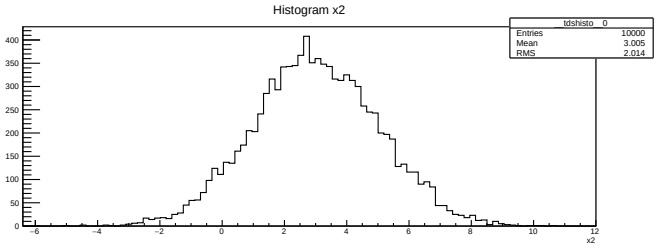
Caution:

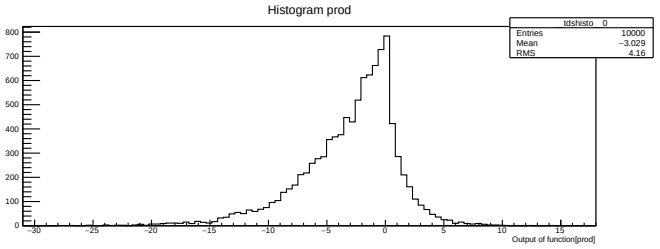
- Unless otherwise specified, all argument are assumed to be double
- Can be used with shared-memory distribution (depends on implementation)

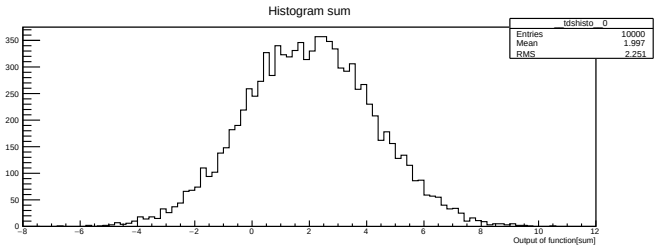
Simple example of an interpreted C++ function

```
void dummyFunction(double *in, double *out)
{

  out[0] = in[0] * in[1];
  out[1] = in[0] + in[1];

}

int ExampleFunction()
{
  //Define problem
  TDataServer *tds = new TDataServer("tds","pouet");
  tds->addAttribute( new TNormalDistribution("x1",-1,1) );
  tds->addAttribute( new TNormalDistribution("x2",3,2) );

  // Construct DOE
  TSampling *ts = new TSampling(tds, "srs", 10000);
  ts->generateSample();

  // Launch Function
  TLauncherFunction *tlf = new TLauncherFunction(tds,
     dummyFunction, "x1:x2", "prod:sum");
  tlf->run();

  // Draw results
  TCanvas *Can = new TCanvas("Can","Can",10,32,600,900);
  Can->Divide(1,4);
  Can->cd(1); tds->Draw("x1");
  Can->cd(2); tds->Draw("x2");
  Can->cd(3); tds->Draw("prod");
  Can->cd(4); tds->Draw("sum");
  Can->SaveAs("exampleFunctionProdAndSum.pdf");
}
```









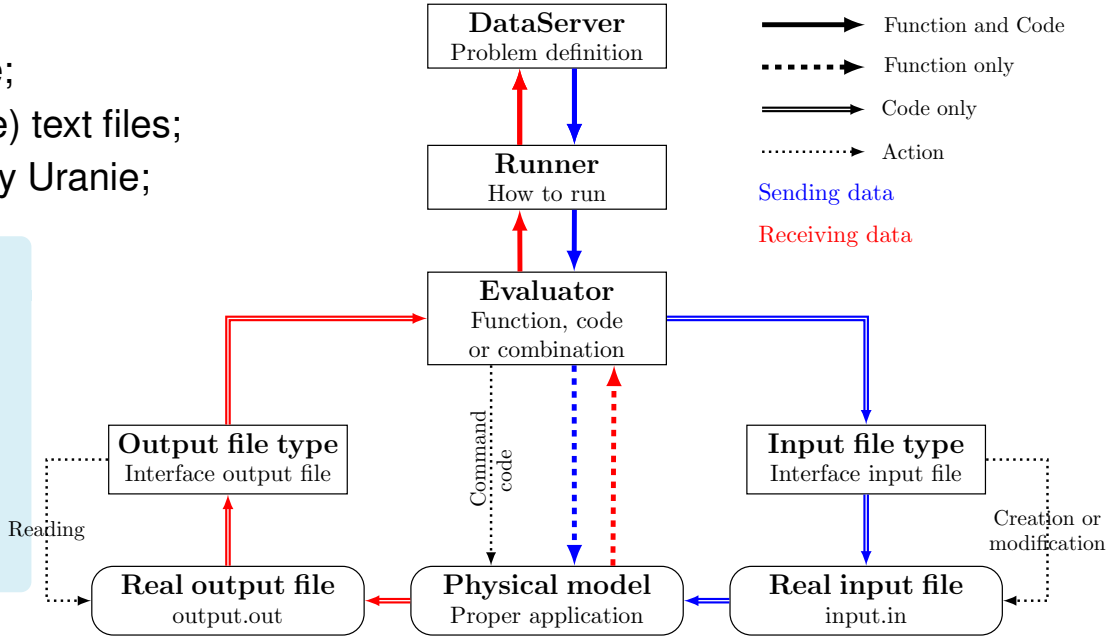
The needed steps for code

To be run by Uranie, a code must:

- be able to be called on a command line;
- receive its entry inputs via one (or more) text files;
- write its outputs in a text file readable by Uranie;

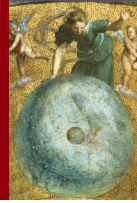
The procedure consists in:

1. creating the input/output file interfaces and connect their variables;
2. executing the code on newly created or modified input files;
3. Reading the output file obtained and storing the values in the TDS.



The main difference with a function is the good handling of input / output files.

Available format for input files



Uranie is capable of dealing with several kinds of input file

row: Numerical values of all the variables of entry are written on only one line (separated by spaces).

column: Numerical values of every variable are written on its own line (a line by variable).

key: Variables associated with a keyword followed by a field where to write its numerical value ("Key = Value"). Key should be unique.

flag: Variables associated with marker in reference input file. The marker will be replaced by corresponding numerical value every time it will appear.

XML: The entry file of reference is a corresponding XML file. Every variable is associated with a XML tag and the numerical value is written in the attribute or the corresponding field (La).

Special care for vectors and strings

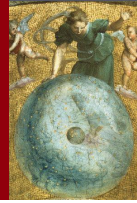
For many files (input/output file can be code) one might need to define

- boundaries: to state where does the string/vector starts and ends
- delimiter: to state how to separate two consecutive values in a vector

Methods exist for that, for all input/output files

[0.325625 , 0.6546941684 , 0.035654685] "Chocolat"

Available format for input files



Uranie is capable of dealing with several kinds of input file

row: Numerical values of all the variables of entry are written on only one line (separated by spaces).

column: Numerical values of every variable are written on its own line (a line by variable).

key: Variables associated with a keyword followed by a field where to write its numerical value ("Key = Value"). Key should be unique.

flag: Variables associated with marker in reference input file. The marker will be replaced by corresponding numerical value every time it will appear.

XML: The entry file of reference is a corresponding XML file. Every variable is associated with a XML tag and the numerical value is written in the attribute or the corresponding field (La).

Special care for vectors and strings

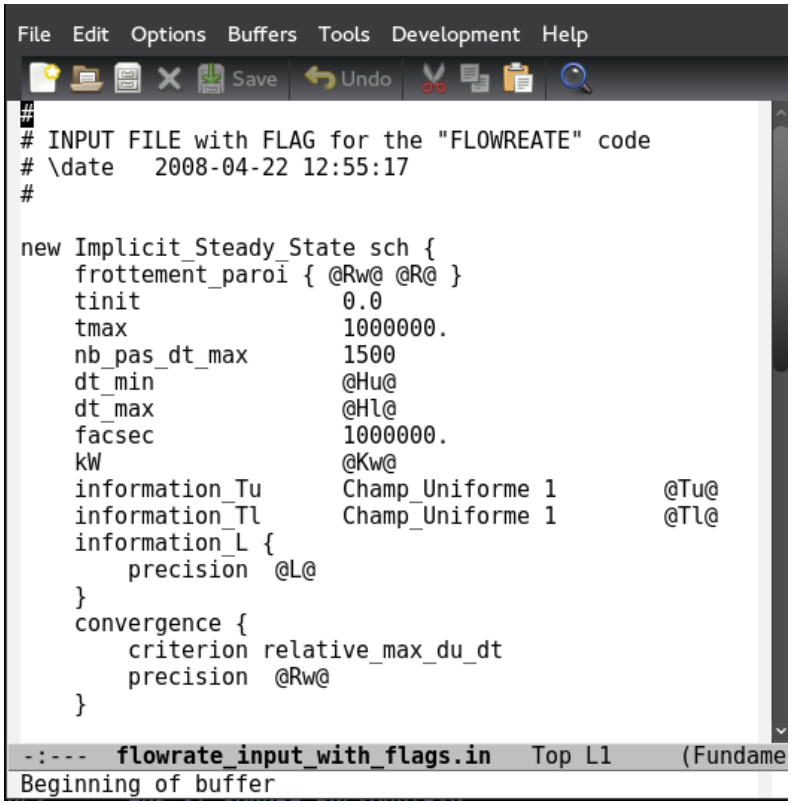
For many files (input/output file can be code) one might need to define

- boundaries: to state where does the string/vector starts and ends
- delimiter: to state how to separate two consecutive values in a vector

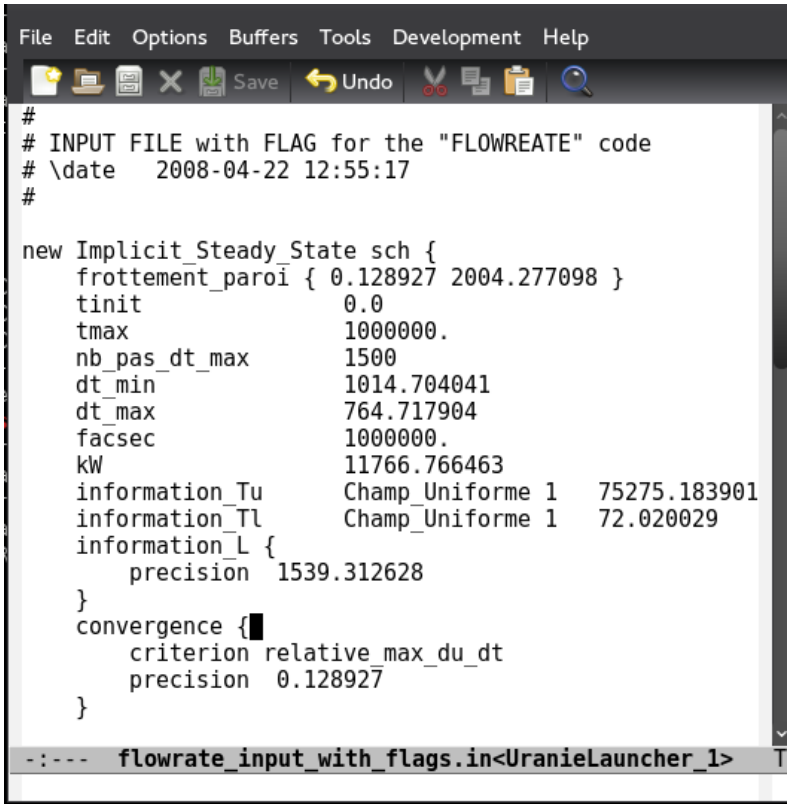
Methods exist for that, for all input/output files

[0.325625 ; 0.6546941684 ; 0.035654685] "Chocolat"

Example of flag format



File containing flags

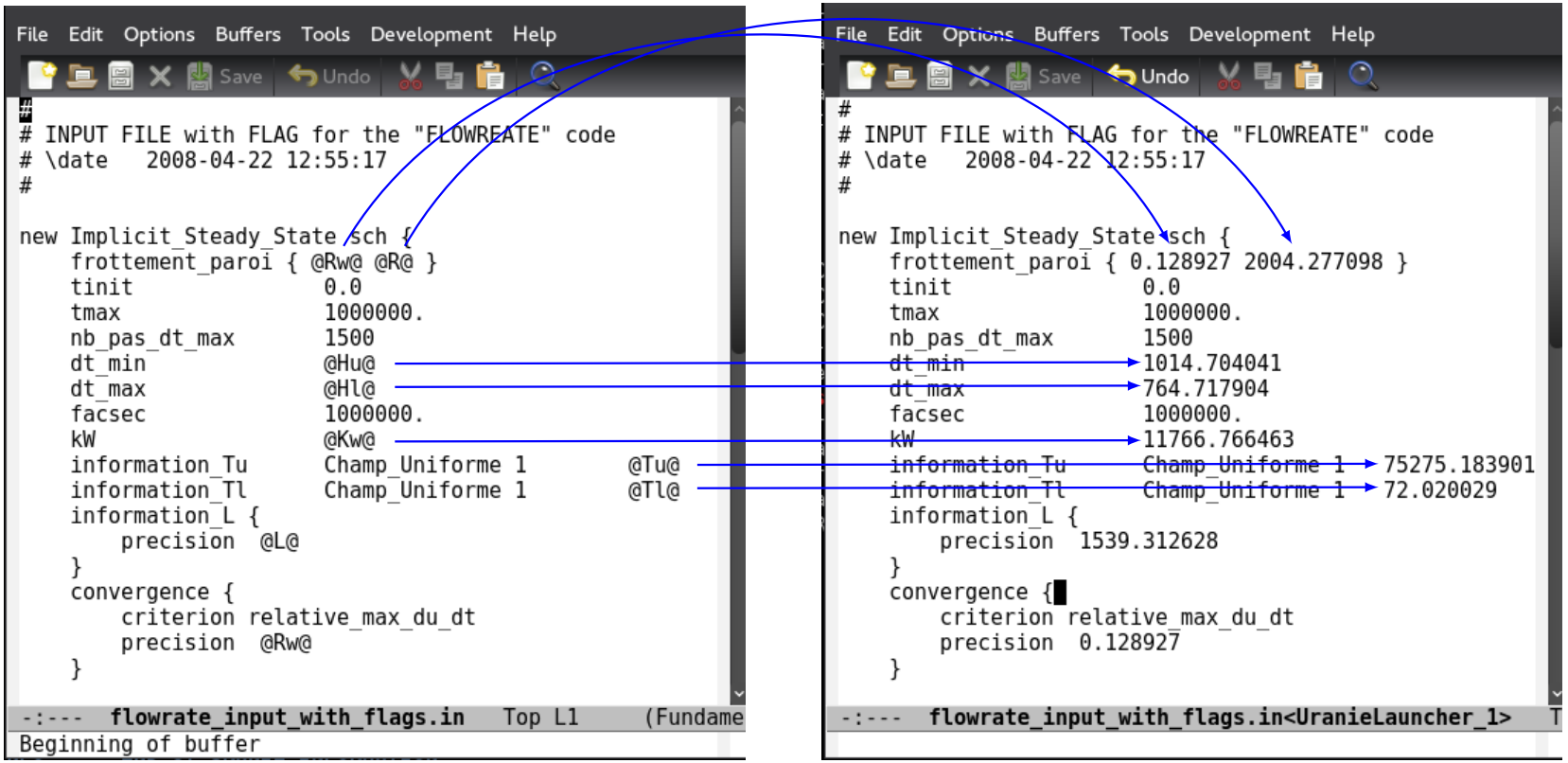


Modified file

Advantage

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

Example of flag format



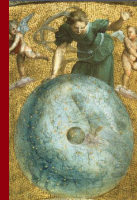
File containing flags

Modified file

Advantage

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

Available format for output files



Uranie is capable of dealing with several kinds of output file

row: Numerical values of all the variables of entry are written on only one line (separated by spaces).

column: Numerical values of every variable are written on its own line (a line by variable).

key: Variables associated with a keyword followed by a field where to write its numerical value ("Key = Value"). Key should be unique.

XML: The entry file of reference is a corresponding XML file. Every variable is associated with a XML tag and the numerical value is written in the attribute or the corresponding field (La).

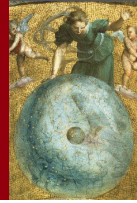
It is also possible to add scripts or codes on top of the one under study to reshape the output file

Composition

Composition of functions/codes can be done as well

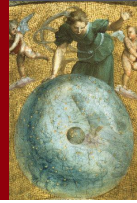
$\Rightarrow$ A proper chain can be defined, output of the Nth assessor can become an input of the N+1th one.

Simple example with a code



```
{
  //Define problem
  TDataServer *tds = new TDataServer("tds","pouet");
  TNormalDistribution *x1=new TNormalDistribution("x1",-1,1);
  TNormalDistribution *x2=new TNormalDistribution("x2",3,2);
  tds->addAttribute( x1 ); tds->addAttribute( x2 );

  // Construct large DOE
  TSampling *ts = new TSampling(tds, "srs", 1000);
  ts->generateSample();

  // Specify where to write input values
  char inF[20]= "input.in";
  x1->setFileKey(inF, "x1", "%f", TAttributeFileKey::kNewKey);
  x2->setFileKey(inF, "x2", "%f", TAttributeFileKey::kNewKey);

  // Create the output file interface
  TOutputFileRow *fout = new TOutputFileRow("toto.out");
  fout->addAttribute("prod"); fout->addAttribute("sum");

  // Define Code
  TCode *code = new TCode(tds, "Dummy");
  code->addOutputFile( fout );

  // Launch Code
  TLauncher *tlf = new TLauncher(tds, code);
  //tlf->setVarDraw("sum");
  tlf->run("nointermed");

  // Draw results
  TCanvas *Can = new TCanvas("Can","Can",10,32,600,900);
  Can->Divide(1,4);
  Can->cd(1); tds->Draw("x1");
  Can->cd(2); tds->Draw("x2");
  Can->cd(3); tds->Draw("prod");
  Can->cd(4); tds->Draw("sum");
  Can->SaveAs("exampleCodeProdAndSum.pdf");
}
```

Simple example with a code



```
{
  //Define problem
  TDataServer *tds = new TDataServer("tds","pouet");
  TNormalDistribution *x1=new TNormalDistribution("x1",-1,1);
  TNormalDistribution *x2=new TNormalDistribution("x2",3,2);
  tds->addAttribute( x1 ); tds->addAttribute( x2 );

  // Construct large DOE
  TSampling *ts = new TSampling(tds, "srs", 1000);
  ts->generateSample();

  // Specify where to write input values
  char inF[20]= "input.in";
  x1->setFileKey(inF, "x1", "%f", TAttributeFileKey::kNewKey);
  x2->setFileKey(inF, "x2", "%f", TAttributeFileKey::kNewKey);

  // Create the output file interface
  TOutputFileRow *fout = new TOutputFileRow("toto.out");
  fout->addAttribute("prod"); fout->addAttribute("sum");

  // Define Code
  TCode *code = new TCode(tds, "Dummy");
  code->addOutputFile( fout );

  // Launch Code
  TLauncher *tlf = new TLauncher(tds, code);
  //tlf->setVarDraw("sum");
  tlf->run("nointermed");

  // Draw results
  TCanvas *Can = new TCanvas("Can","Can",10,32,600,900);
  Can->Divide(1,4);
  Can->cd(1); tds->Draw("x1");
  Can->cd(2); tds->Draw("x2");
  Can->cd(3); tds->Draw("prod");
  Can->cd(4); tds->Draw("sum");
  Can->SaveAs("exampleCodeProdAndSum.pdf");
}
```

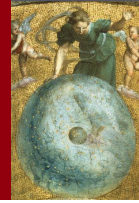
Dummy.C

```
#include <iostream>
#include <fstream>
#include <sstream>

int main(void)
{
  double x1, x2, out1, out2;

  //  Getting the inputs
  std::string key, equal, ponct;
  std::fstream  afile;
  afile.open("input.in", std::ios::in );
  afile >> key >> equal >> x1 >> ponct;
  afile >> key >> equal >> x2 >> ponct;
  afile.close();

  out1 = x1 * x2;
  out2 = x1 + x2;

   // Output
  std::ofstream out;
  out.open("toto.out");
  out << out1 << " " << out2 << "\n";
  out.close();
}
```

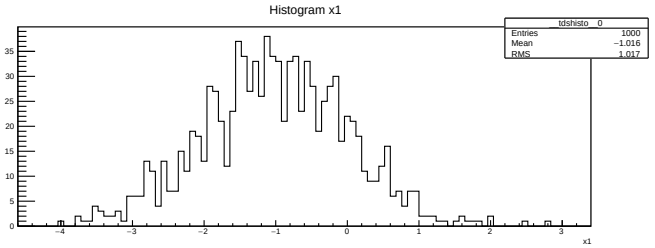
input.in

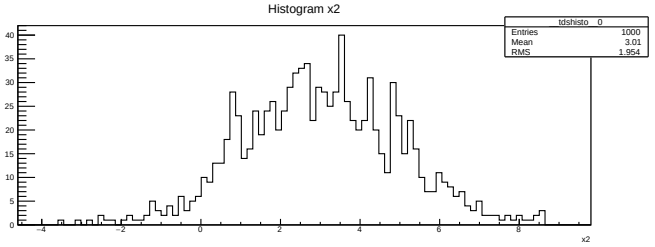
```
x1 = 0.161040 ;
x2 = 4.591294 ;
```

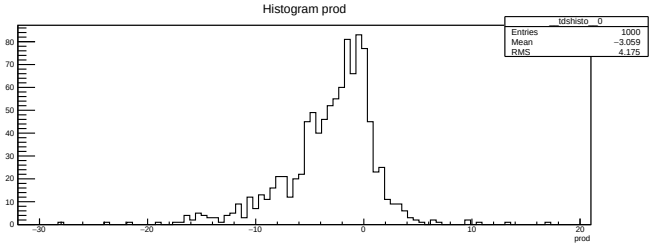
Simple example with a code

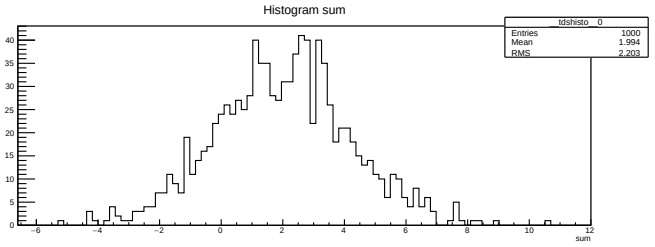


```
{
  //Define problem
  TDataServer *tds = new TDataServer("tds","pouet");
  TNormalDistribution *x1=new TNormalDistribution("x1",-1,1);
  TNormalDistribution *x2=new TNormalDistribution("x2",3,2);
  tds->addAttribute( x1 ); tds->addAttribute( x2 );

  // Construct large DOE
  TSampling *ts = new TSampling(tds, "srs", 1000);
  ts->generateSample();

  // Specify where to write input values
  char inF[20]= "input.in";
  x1->setFileKey(inF, "x1", "%f", TAttributeFileKey::kNewKey);
  x2->setFileKey(inF, "x2", "%f", TAttributeFileKey::kNewKey);

  // Create the output file interface
  TOutputFileRow *fout = new TOutputFileRow("toto.out");
  fout->addAttribute("prod"); fout->addAttribute("sum");

  // Define Code
  TCode *code = new TCode(tds, "Dummy");
  code->addOutputFile( fout );

  // Launch Code
  TLauncher *tlf = new TLauncher(tds, code);
  //tlf->setVarDraw("sum");
  tlf->run("nointermed");

  // Draw results
  TCanvas *Can = new TCanvas("Can","Can",10,32,600,900);
  Can->Divide(1,4);
  Can->cd(1); tds->Draw("x1");
  Can->cd(2); tds->Draw("x2");
  Can->cd(3); tds->Draw("prod");
  Can->cd(4); tds->Draw("sum");
  Can->SaveAs("exampleCodeProdAndSum.pdf");
}
```





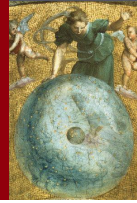




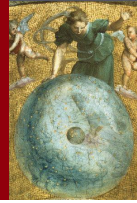
DE LA RECHERCHE À L'INDUSTRIE

cea

Outline



Main principle of surrogate-models (1/2)



A surrogate model is a - more or less - complicated function that reproduce / mimic as best as possible the behaviour of a complicated code

Surrogate-models need a training basis, $\mathcal{L}$, defined as:

$$\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i \in [1, n_S]}, \quad \text{where} \quad \mathbf{x}_i = (x_i^1 \ \dots \ x_i^{n_X})$$

- $\mathcal{C}$ is the code;
- $\mathbf{x}_i$ is the ith realisation of random variable inputs vector $\mathbf{X}$;
- y_i is the ith realisation of the output random variable Y ($y_i = \mathcal{C}(\mathbf{x}_i)$) whose expectation on $\mathcal{L}$ is $\overline{y}$.

The estimation of the output of interest is written $\hat{y} = M(\mathbf{x})$ where M is the surrogate model.

Quality criteria

Using the training basis only:

- Mean Square Error : $MSE = \frac{1}{n_S} \sum_{i=1}^{n_S} (y_i - \hat{y}_i)^2$
- $R^2 = 1 - \sum_{i=1}^{n_S} \frac{(y_i - \hat{y}_i)^2}{(y_i - \overline{y})^2}$

The closer to 1, R^2 and/or Q^2 are the better the model M is.

Given a test basis $\mathcal{P}$ of size n_P:

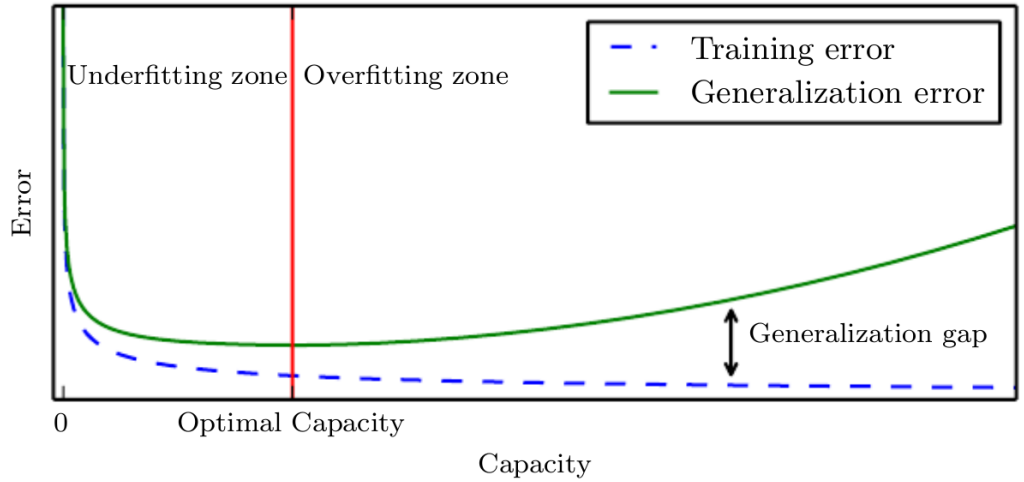
$$Q^2 = 1 - \sum_{i=1}^{n_P} \frac{(y_i - \hat{y}(\mathbf{x_i}))^2}{(y_i - \overline{y})^2}, \mathbf{x_i} \in \mathcal{P}$$

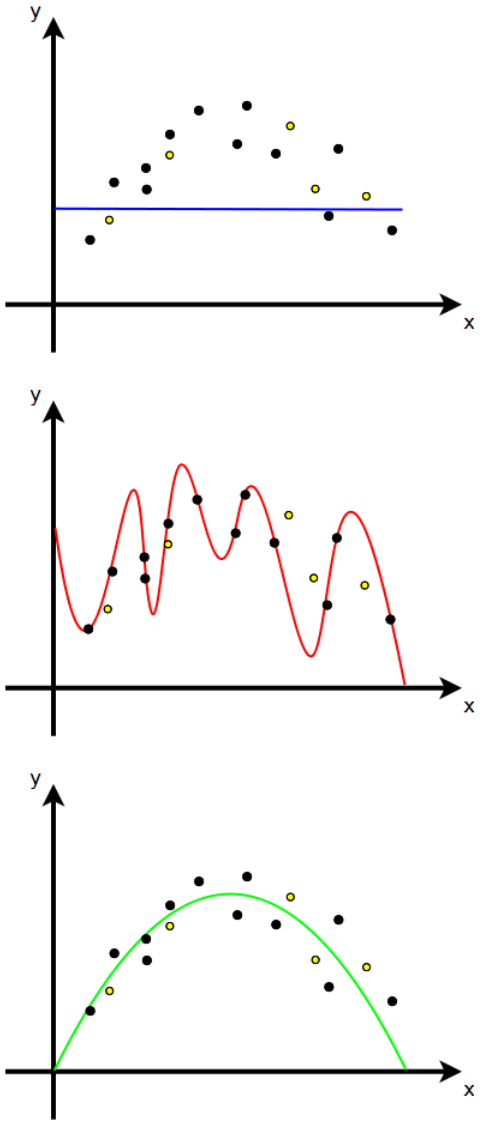
Main principle of surrogate-models (2/2)

Advance techniques with the training basis

- Regularisation can be used to avoid over/under fitting problems:
  - ➙ Split $\mathcal{L}$ into two parts: training (large) and control (small)
  - ➙ Compute MSE for the training part (**training error**) and the control one (**Generalisation error**)
  - ➙ stop optimisation once generalisation error is growing.
- *Leave-one-out* method (LOO):
  - ➙ estimate $\hat{y'}_i = M'(\mathbf{x_i})$ where M' is the model whose training has been made on $\mathcal{L}$ without the ith point.
  - ➙ repeat this n_S times and compute the statistical measurements:

$$MSE_{Loo} = \frac{1}{n_S}\sum_{i=1}^{n_S}(y_i - \hat{y'}_i)^2 \quad \text{and} \quad Q^2_{Loo} = 1 - \sum_{i=1}^{n_S}\frac{(y_i - \hat{y'}_i)^2}{(y_i - \bar{y})^2}$$





Main principle of surrogate-models (2/2)

To avoid under-fitting requires:

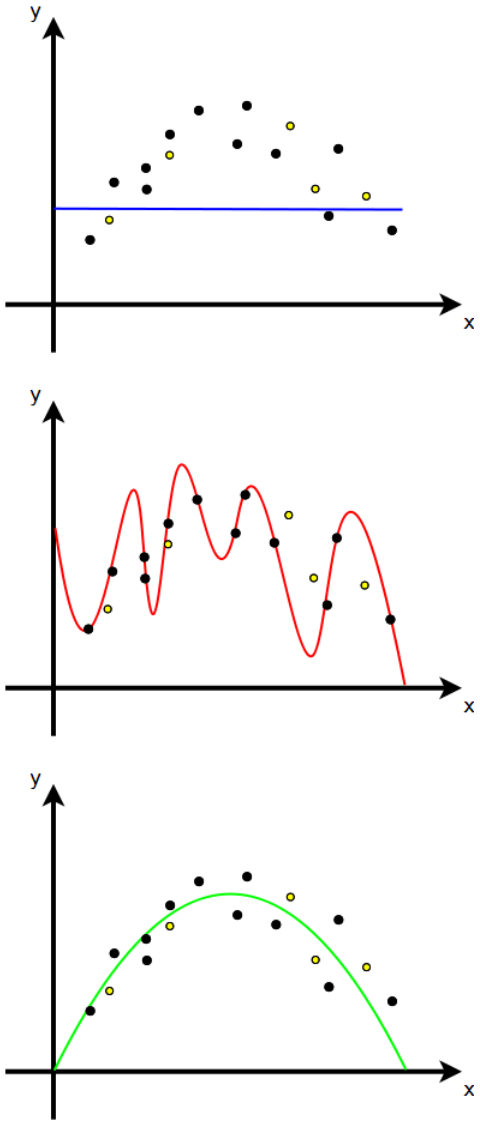
- selecting a family of functions adapted to the data to be modelled;
- choosing an optimisation algorithm capable of minimising the chosen criterion;
- giving enough degrees of freedom to the model to fit the data.

To avoid over-fitting, you must:

- ensure that the examples used to build the model are representative of the domain of validity in question;
- have a strategy to control the degrees of freedom of the model.

Some examples of control strategies role:

- control the number of model parameters;
- control the range of variation of the parameters values;
- monitor the progress of the optimisation;
- etc. . .

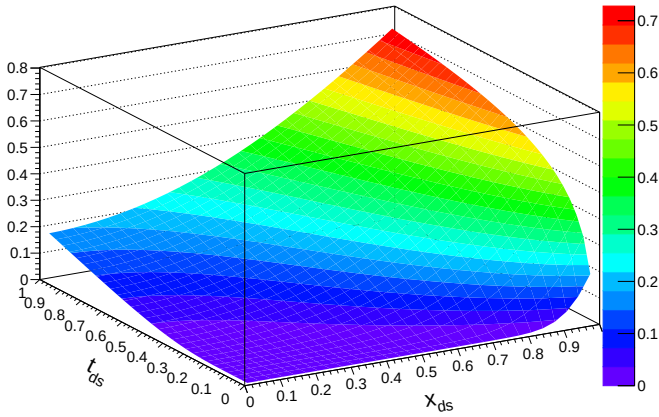


Training on a known database for our case...

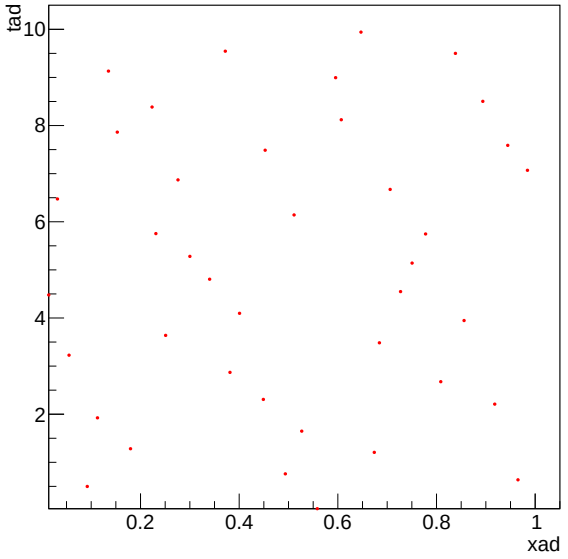
Simple case:

Reproduce the behaviour of the simple case in dimensionless space

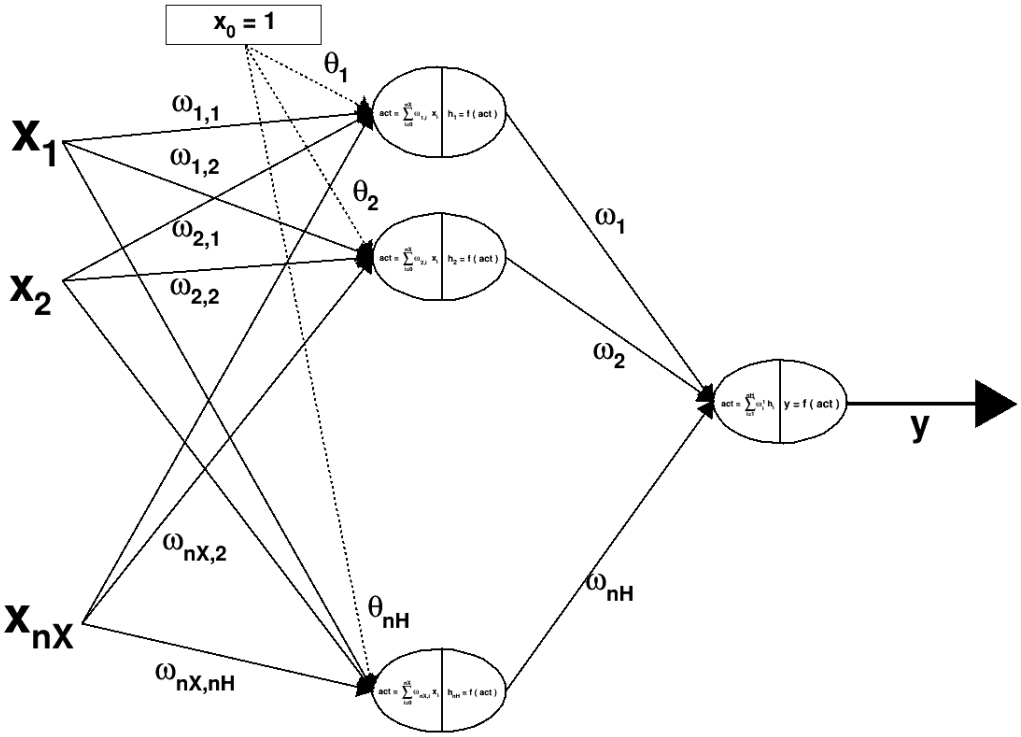
- Using a 40 locations DOE as inputs in (x_{ds},t_{ds})
- Generate various models from it
  - Linear regression
  - k-nearest neighbour
  - Neural network
  - Kriging
  - Chaos polynomial expansion
  - ...
- Run the model on a test basis of 2000 locations
  - ➙ Plot the estimated values $\hat{\theta}$ as a function of the real one (θ)



tad:xad

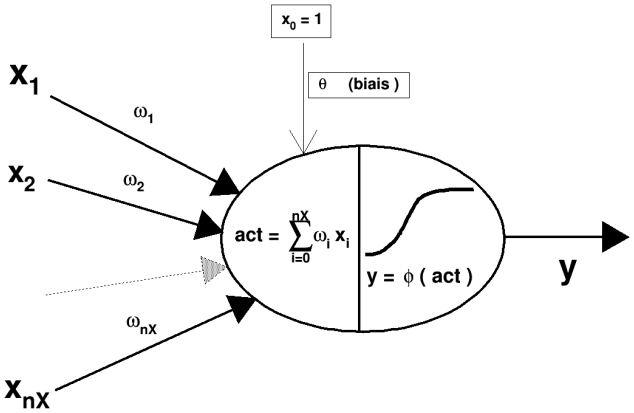


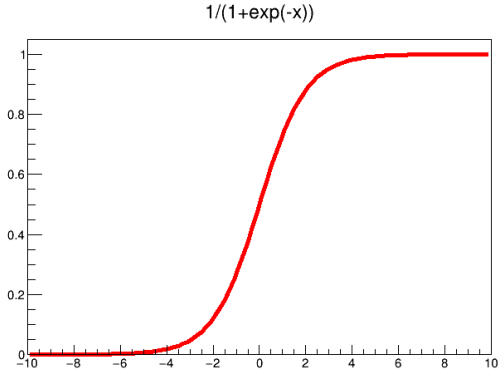
The neural network concept (1/2)

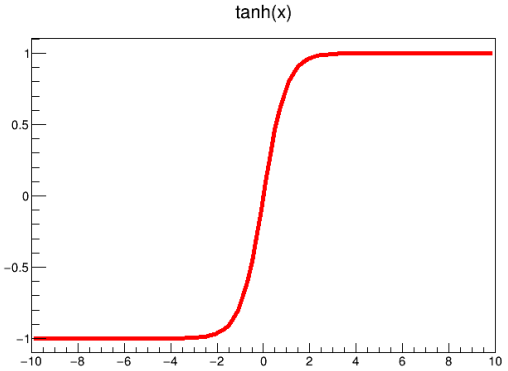


- A neural network is the association, in a graph more or less complex, of elementary objects called formal neurons.
- The Artificial Neural Networks (ANN) in Uranie are Multi Layer Perceptron (MLP) with one (or more) hidden layer and one (or more) output variables. A MLP is a network composed of successive layers where the neurons of one layer does not have any connections with each other.

The neural network concept (2/2)







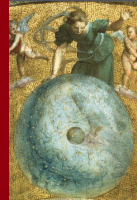
A formal neuron is a model that is characterised by:

- an internal state y $\in$ S;
- $x_1, \ldots, x_{n_X}$ input signals
- a weight vector $[\omega_0, \ldots, \omega_{n_X}]$
- an activation function ϕ
  - The activation function performs a transformation of a linear combination of the input signals:

$$y = \phi\left(\sum_{i=1}^{n_X} \omega_i X_i\right) \text{ with } \phi(x) = \frac{1}{1+e^{-x}} \text{ or } tanh(x)$$

  - This linear combination is determined by a weight vector $[\omega_0, \ldots, \omega_{n_X}]$ associated with each neuron and whose values are estimated in the learning phase.

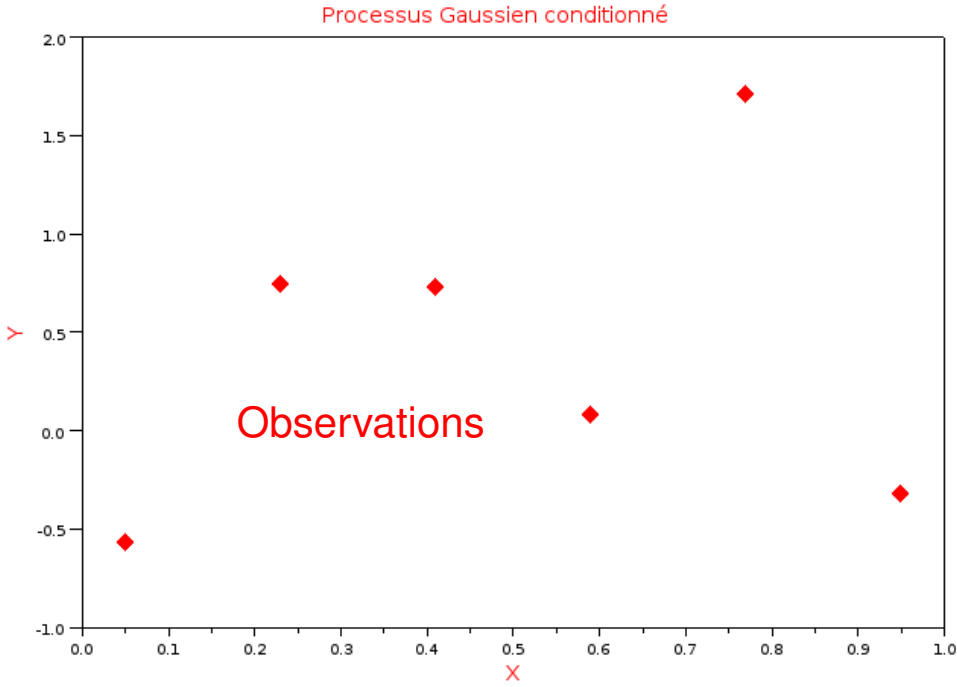
Gaussian process in action



- Observations $y_{obs} = (y_{obs,i})_{1 \leq i \leq n}$ for $(\mathbf{x}_i)_{1 \leq i \leq n}$
- Choose covariance function $k(\mathbf{x}_i, \mathbf{x}_j; \theta)$
- Estimate θ by Maximum Likelihood
- Compute covariance matrix
  $\mathbf{K} = (k(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i,j \leq n}$
- Law Y conditionally to the observations $Y|y_{obs}$
  $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}, \mathbf{x}_j))_{1 \leq i,j \leq n}$
  $y(\mathbf{x}) = \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{y}_{obs}$
  $\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{k}(\mathbf{x})$
  For these, the mean is considered equal to 0
- Conditional expectation $y(\mathbf{x}) \Rightarrow$ estimation by the model
- Conditional variance $\sigma^2(\mathbf{x}) \Rightarrow$ confidence interval

$\emptyset$ uncertainty on observations $\Rightarrow$ interpolation mode

Validation can be done with Leave-One Out method (simple as its needs only removing a line/column in matrices)

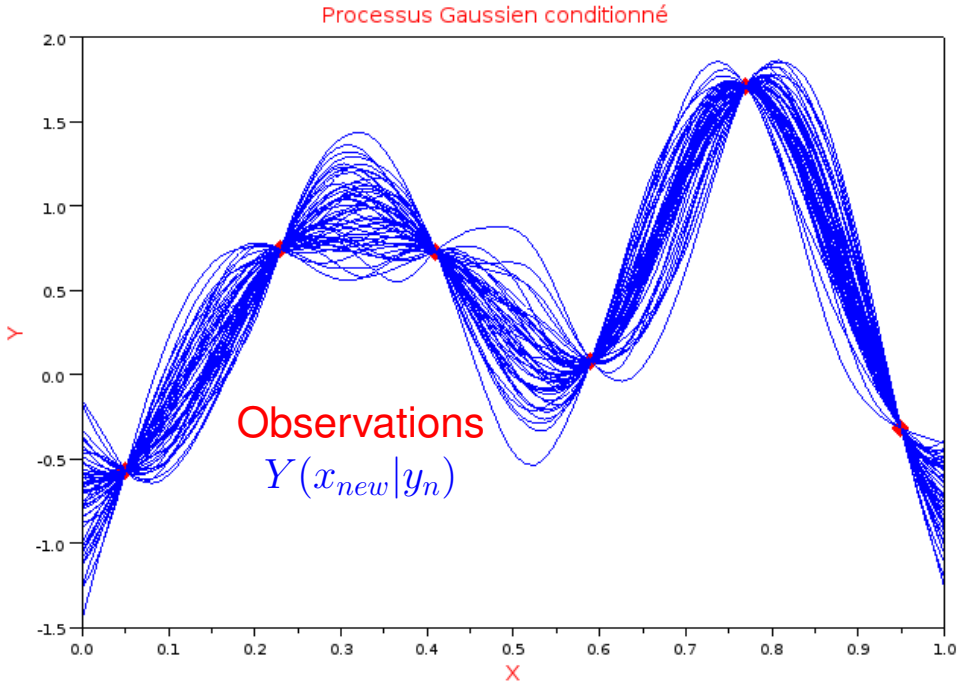


Gaussian process in action

- Observations $y_{obs} = (y_{obs,i})_{1 \leq i \leq n}$ for $(\mathbf{x}_i)_{1 \leq i \leq n}$
- Choose covariance function $k(\mathbf{x}_i, \mathbf{x}_j; \theta)$
- Estimate θ by Maximum Likelihood
- Compute covariance matrix
  $\mathbf{K} = (k(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i,j \leq n}$
- Law Y conditionally to the observations $Y|y_{obs}$
  $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}, \mathbf{x}_j))_{1 \leq i,j \leq n}$
  $y(\mathbf{x}) = \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{y}_{obs}$
  $\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{k}(\mathbf{x})$
  For these, the mean is considered equal to 0
- Conditional expectation $y(\mathbf{x}) \Rightarrow$ estimation by the model
- Conditional variance $\sigma^2(\mathbf{x}) \Rightarrow$ confidence interval

$\emptyset$ uncertainty on observations $\Rightarrow$ interpolation mode

Validation can be done with Leave-One Out method (simple as its needs only removing a line/column in matrices)

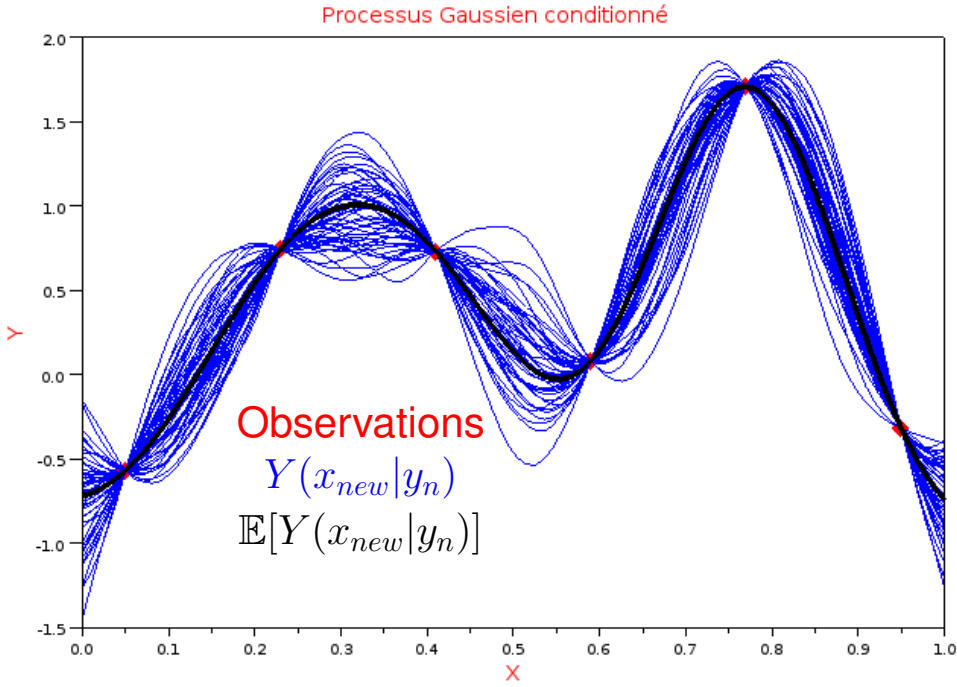


Gaussian process in action

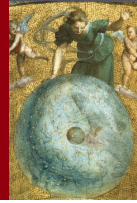
- Observations $y_{obs} = (y_{obs,i})_{1 \leq i \leq n}$ for $(\mathbf{x}_i)_{1 \leq i \leq n}$
- Choose covariance function $k(\mathbf{x}_i, \mathbf{x}_j; \theta)$
- Estimate θ by Maximum Likelihood
- Compute covariance matrix $\mathbf{K} = (k(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i,j \leq n}$
- Law Y conditionally to the observations $Y|y_{obs}$
  $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}, \mathbf{x}_j))_{1 \leq i,j \leq n}$
  $y(\mathbf{x}) = \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{y}_{obs}$
  $\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{k}(\mathbf{x})$
  For these, the mean is considered equal to 0
- Conditional expectation $y(\mathbf{x}) \Rightarrow$ estimation by the model
- Conditional variance $\sigma^2(\mathbf{x}) \Rightarrow$ confidence interval

$\emptyset$ uncertainty on observations $\Rightarrow$ interpolation mode

Validation can be done with Leave-One Out method (simple as its needs only removing a line/column in matrices)



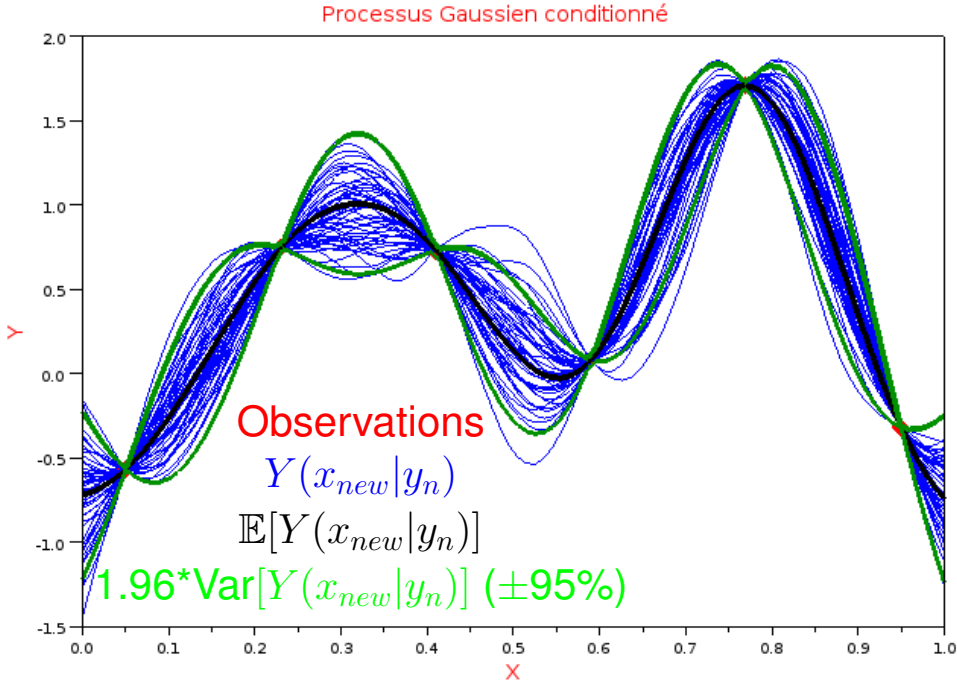
Gaussian process in action



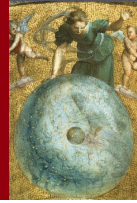
- Observations $y_{obs} = (y_{obs,i})_{1 \leq i \leq n}$ for $(\mathbf{x}_i)_{1 \leq i \leq n}$
- Choose covariance function $k(\mathbf{x}_i, \mathbf{x}_j; \theta)$
- Estimate θ by Maximum Likelihood
- Compute covariance matrix
  $\mathbf{K} = (k(\mathbf{x}_i, \mathbf{x}_j))_{1 \leq i,j \leq n}$
- Law Y conditionally to the observations $Y|y_{obs}$
  $\mathbf{k}(\mathbf{x}) = (k(\mathbf{x}, \mathbf{x}_j))_{1 \leq i,j \leq n}$
  $y(\mathbf{x}) = \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{y}_{obs}$
  $\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^t(\mathbf{x})\mathbf{K}^{-1}\mathbf{k}(\mathbf{x})$
  For these, the mean is considered equal to 0
- Conditional expectation $y(\mathbf{x}) \Rightarrow$ estimation by the model
- Conditional variance $\sigma^2(\mathbf{x}) \Rightarrow$ confidence interval

$\emptyset$ uncertainty on observations $\Rightarrow$ interpolation mode

Validation can be done with Leave-One Out method (simple as its needs only removing a line/column in matrices)



Chaos polynomial expansion principle I



Chaos polynomial expansion is functional approach that allows to build a surrogate model on a base of orthogonal polynomial well suited for sensitivity analysis.

Introduced by Wiener (1938) popularised by Ghanem (1999)

Every random variable whose mean and variance is finite can be written

$$X(\xi) = \sum_{\alpha} x_\alpha \Psi_\alpha(\xi)$$

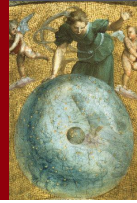
- ξ_i independant gaussian random variables and x_α are the coefficients
- $\Psi_\alpha(\xi) = \prod_i H_{\alpha_i}(\xi_\mathbf{i})$ obtained by Hermite polynomial tensorisation whose degree α_i

Usage in Uranie:

This summation needs to be truncated twice :

- on the degree (p)
- on the number of variable ξ_i
  $\Rightarrow$ This number is fixed in Uranie: every ξ_i associated to an uncertainty to be modelled.

Chaos polynomial expansion principle II



- The group of random variable $X(\xi)$ with finite mean and variance is an Hilbert space with a scalar product

$$< X, Y >= \mathbb{E}(XY)$$

- ξ_i are independant $\Rightarrow$ the Ψ_α construction from Hermite tensorisation are orthogonal

$$< \Psi_\alpha, \Psi_{\alpha'} >= \mathbf{1}_{\alpha=\alpha'} ||\Psi_\alpha||^2$$

- Ψ_α form an orthogonal basis

Polynomial chaos property

With ϕ_0 = 1 and $||\phi_\alpha||$ = 1

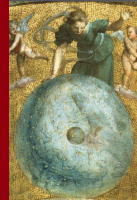
- Expansion coefficients are obtained by projection $x_\alpha =< X, \Psi_\alpha >$
- $\mathbb{E} = x_0$
- $\sigma^2(X) = \sum_{|\alpha|>0} x_\alpha^2$

Can link the value of the coefficient to mean, variance so the functionnal decomposition of variance (Sobol indices)

Generalised Chaos polynomial expansion

- Generalised to other polynomials tensorisation depending on considered densities
- in Uranie : Hermite (gaussian), Legendre (Uniform), Laguerre (Exponential), Jacobi (Beta)

Chaos polynomial expansion principle III



Using a system depending on two random variables, ξ_U (uniform) and ξ_N (gaussian).

$$Y(\xi_U, \xi_N) = \sum_{|\alpha|_1 \leq p} \beta_\alpha \Psi_\alpha(\xi_U, \xi_N), \quad \text{where } \alpha \in \mathbb{N}^2$$

$$\begin{aligned} Y(\xi_U, \xi_N) = \beta_{0,0} \qquad & (|\alpha|_1 = 0) \\ + \beta_{1,0}\,\mathcal{L}_1(\xi_U) + \beta_{0,1}\,\mathcal{H}_1(\xi_N) \quad & (|\alpha|_1 = 1) \\ + \beta_{1,1}\,\mathcal{L}_1(\xi_U)\mathcal{H}_1(\xi_N) + \beta_{2,0}\,\mathcal{L}_2(\xi_U) + \beta_{0,2}\,\mathcal{H}_2(\xi_N) \quad & (|\alpha|_1 = 2) \\ + \beta_{2,1}\,\mathcal{L}_2(\xi_U)\mathcal{H}_1(\xi_N) + \beta_{1,2}\,\mathcal{L}_1(\xi_U)\mathcal{H}_2(\xi_N) + \beta_{3,0}\,\mathcal{L}_3(\xi_U) + \beta_{0,3}\,\mathcal{H}_3(\xi_N) \quad & (|\alpha|_1 = 3) \\ + \ldots & \end{aligned}$$

Properties

- The chosen degree (p) sets the truncation of the expansion
- Resulting number of coefficients

$$N_{\text{coeff}} = \frac{(n_X + p)!}{n_X!\, p!}$$

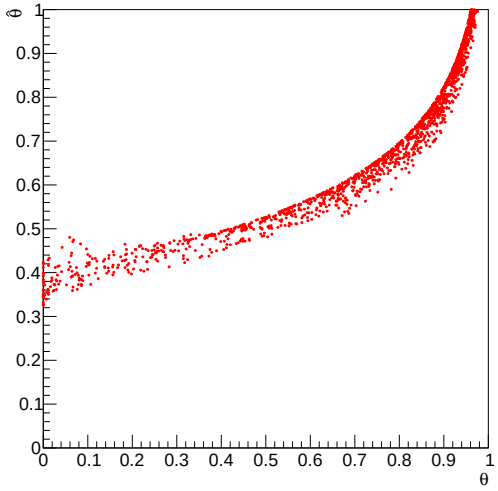
Coefficient measurement

Can be done by regression or integration

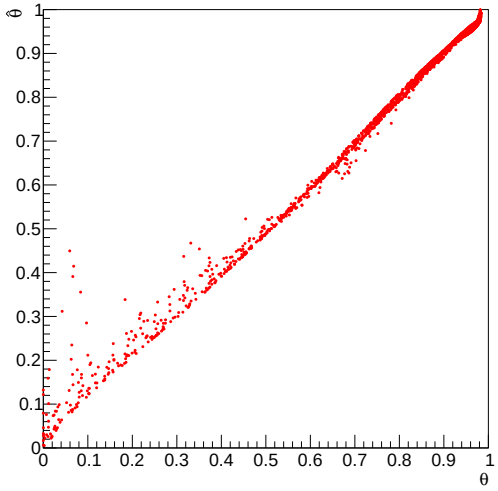
- Monte-Carlo or QMC
- Gauss quadrature, tensorised 1D formula, Smolyak construction (sparse grid)

Summary of different models

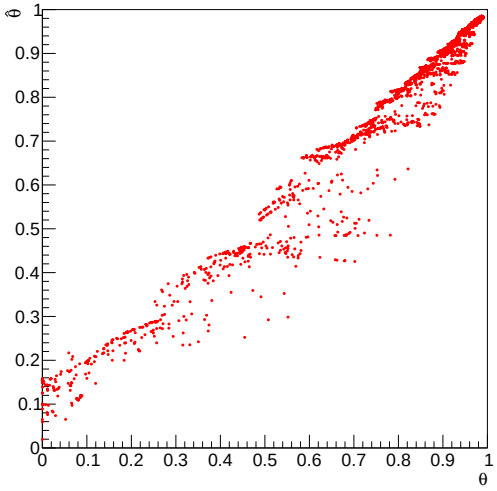
Linear Regression

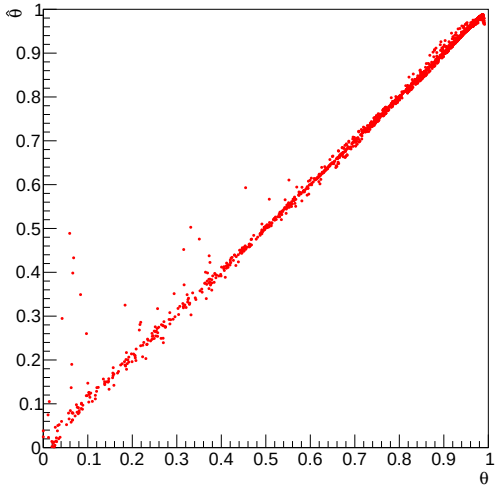


CP expansion

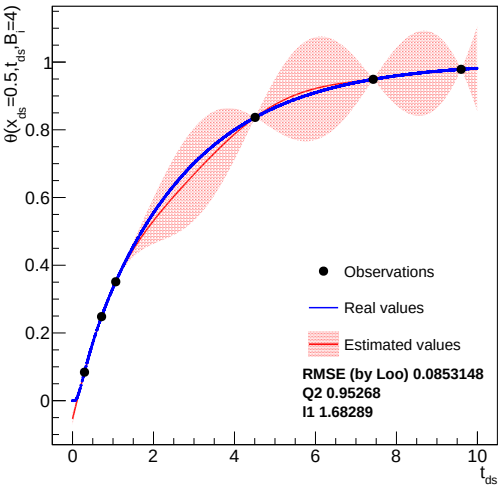


kNN

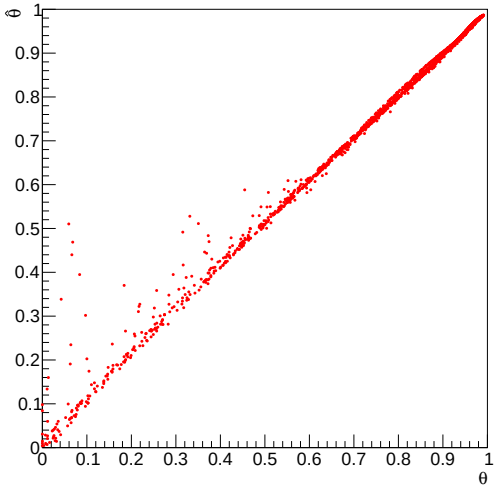




Kriging 2D

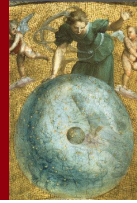


Kriging 1D

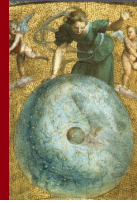


Neural network

Outline



Global picture



This module contains methods to generate design-of-experiments (DOE), from the information provided on the variables of the study.

- Some methods are deterministic:
  - variable's law might not be known: default value and interval of variation are (often) sufficient.
  - variable's observations are not independent, but its distribution brings other advantages.
  - a set of data is perfectly reproducible.

- Some methods are stochastic:
  - the generated values are realisations of random variables of known laws.
  - variable's observations are independent. Two variable's observations are not a priori correlated.
  - a doe cannot be reproduced, unless the "seed" of the random generator is known.

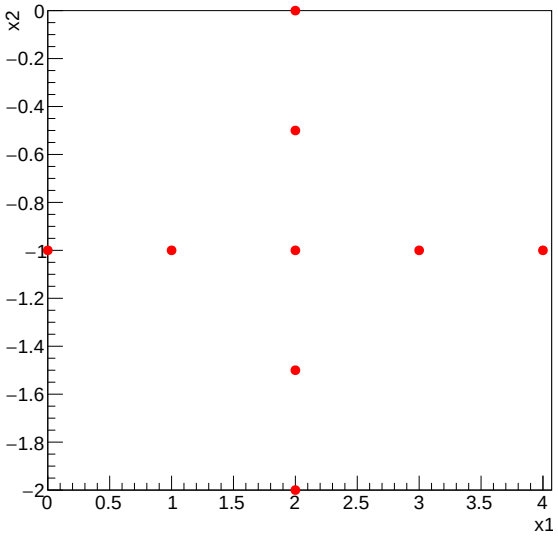
Discussing some deterministic approaches

What DOE can be produced ?

Depending on the analysis purpose

- check impact of certain parameters
- get the best estimate of integral for instance
- get the best coverage of the input parameter space (lowest discrepance)
- ...

Regular OAT



Petras, level=8

Halton Sequence

Discussing some deterministic approaches

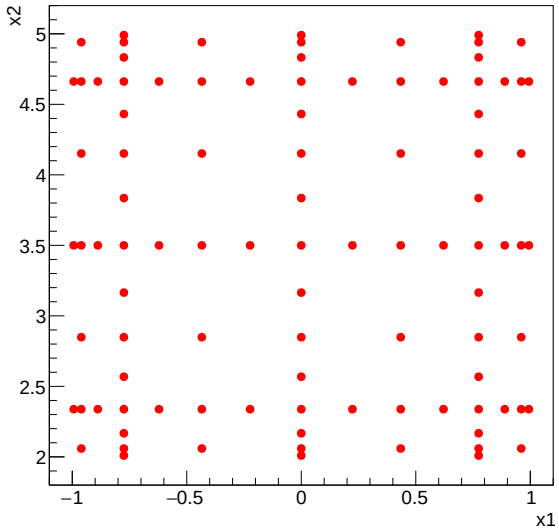
What DOE can be produced ?

Depending on the analysis purpose

- check impact of certain parameters
- get the best estimate of integral for instance
- get the best coverage of the input parameter space (lowest discrepance)
- ...

Regular OAT

Petras, level=8



Halton Sequence

Discussing some deterministic approaches

What DOE can be produced ?

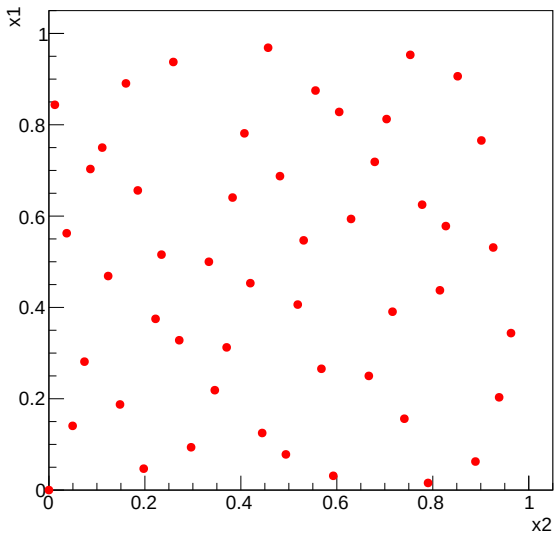
Depending on the analysis purpose

- check impact of certain parameters
- get the best estimate of integral for instance
- get the best coverage of the input parameter space (lowest discrepance)
- ...

Regular OAT

Petras, level=8

Halton Sequence



Random sampling: simple or stratified

Different possible construction

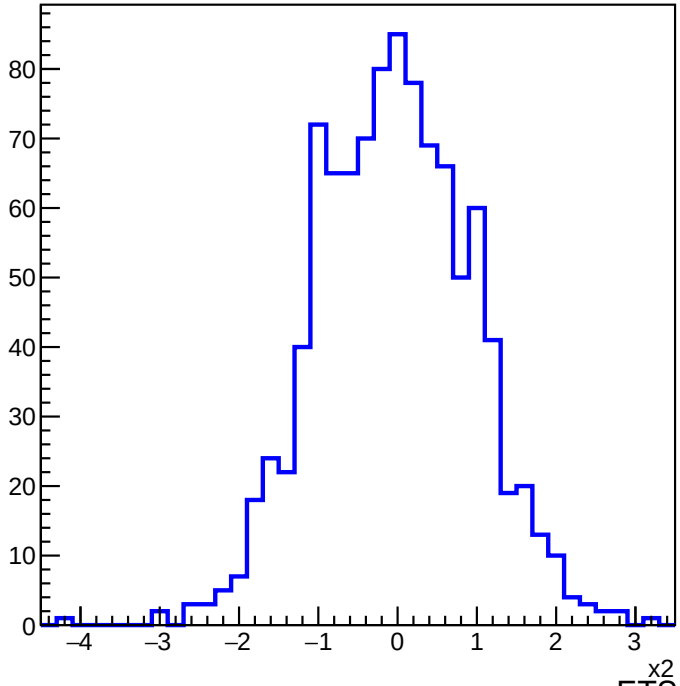
SRS: Generating the samples for each parameter following its own probability density function.

- ➙ Obtained parameter variance rather high ⇔ precision of the estimation is poor
- ➙ Many repetitions needed in order to reach a satisfactory precision.

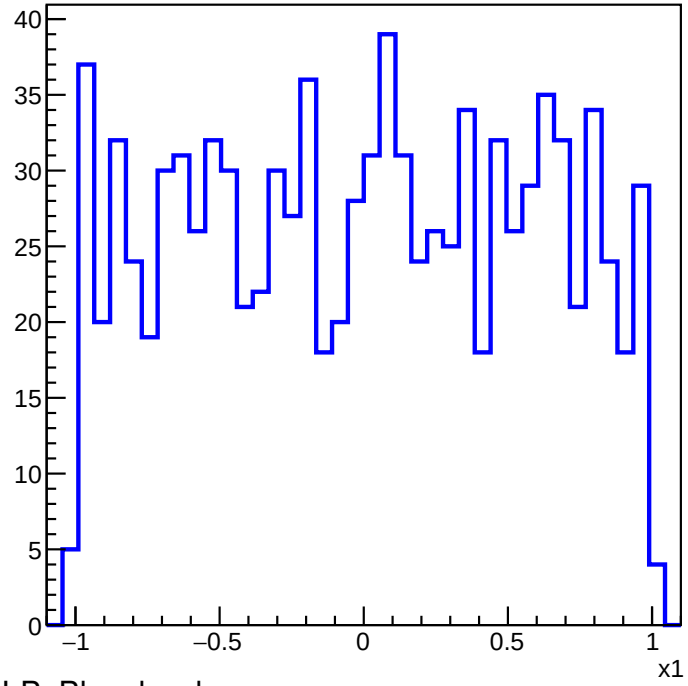
LHS: Split each parameter interval into equal-probabilities. One value drawn for every segment.

- ➙ Ensure that each variable's domain of variation totally covered in a homogeneous way
- ➙ Cannot add points once generated.

SRS case, N(0,1)



SRS case, U(0,1)



Random sampling: simple or stratified

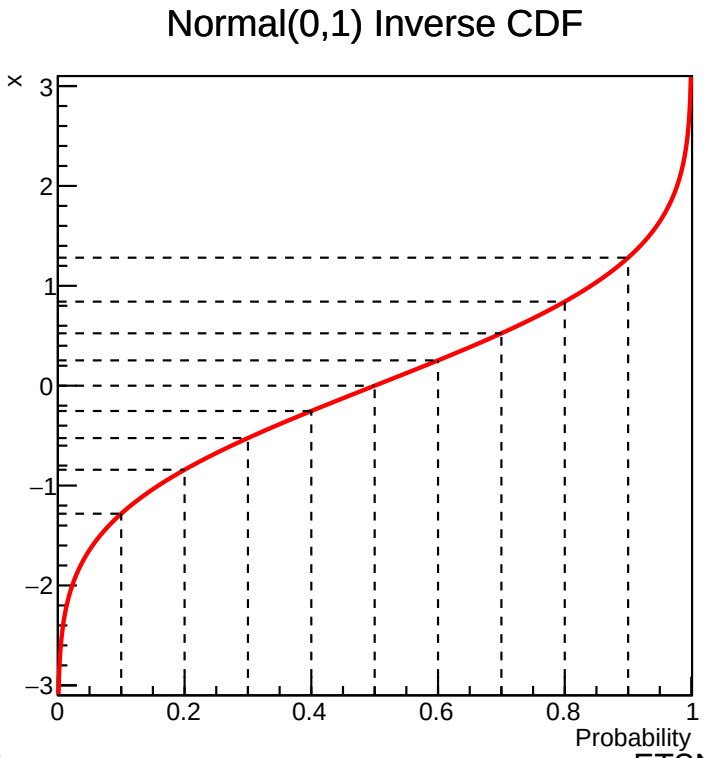
Different possible construction

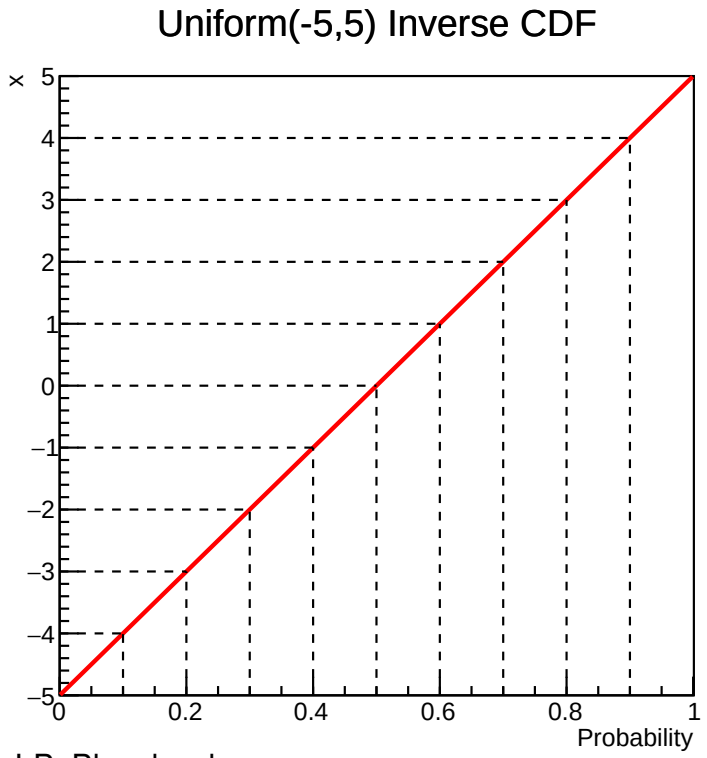
SRS: Generating the samples for each parameter following its own probability density function.

- ➡ Obtained parameter variance rather high ⇔ precision of the estimation is poor
- ➡ Many repetitions needed in order to reach a satisfactory precision.

LHS: Split each parameter interval into equal-probabilities. One value drawn for every segment.

- ➡ Ensure that each variable's domain of variation totally covered in a homogeneous way
- ➡ Cannot add points once generated.





Random sampling: simple or stratified

Different possible construction

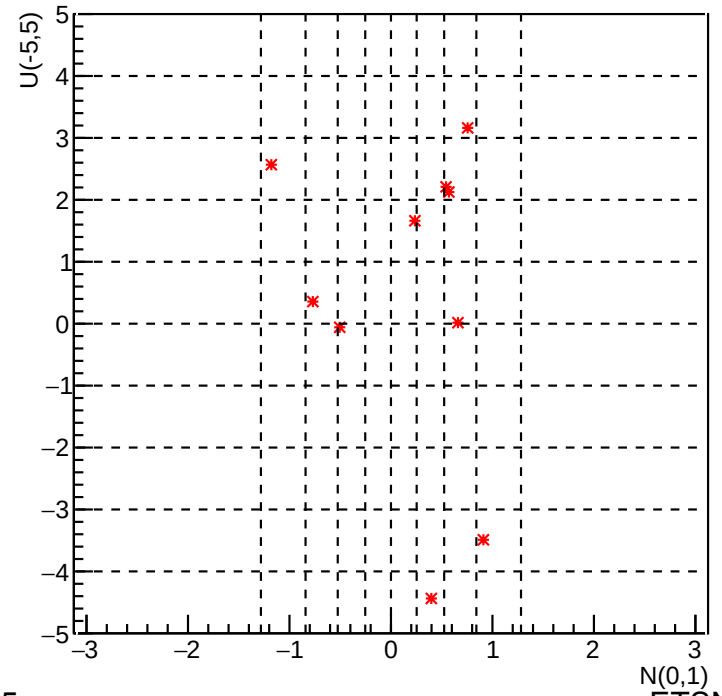
SRS: Generating the samples for each parameter following its own probability density function.

- ➡ Obtained parameter variance rather high ⇔ precision of the estimation is poor
- ➡ Many repetitions needed in order to reach a satisfactory precision.

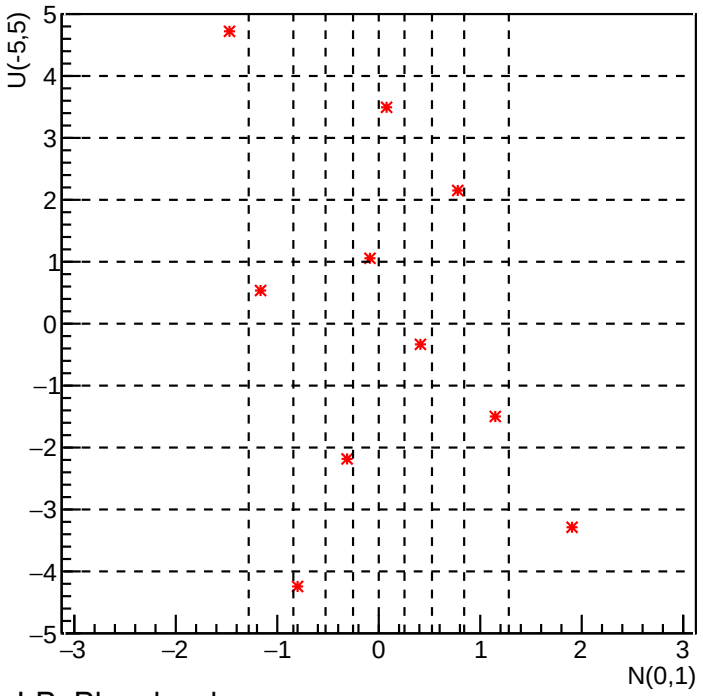
LHS: Split each parameter interval into equal-probabilities. One value drawn for every segment.

- ➡ Ensure that each variable's domain of variation totally covered in a homogeneous way
- ➡ Cannot add points once generated.

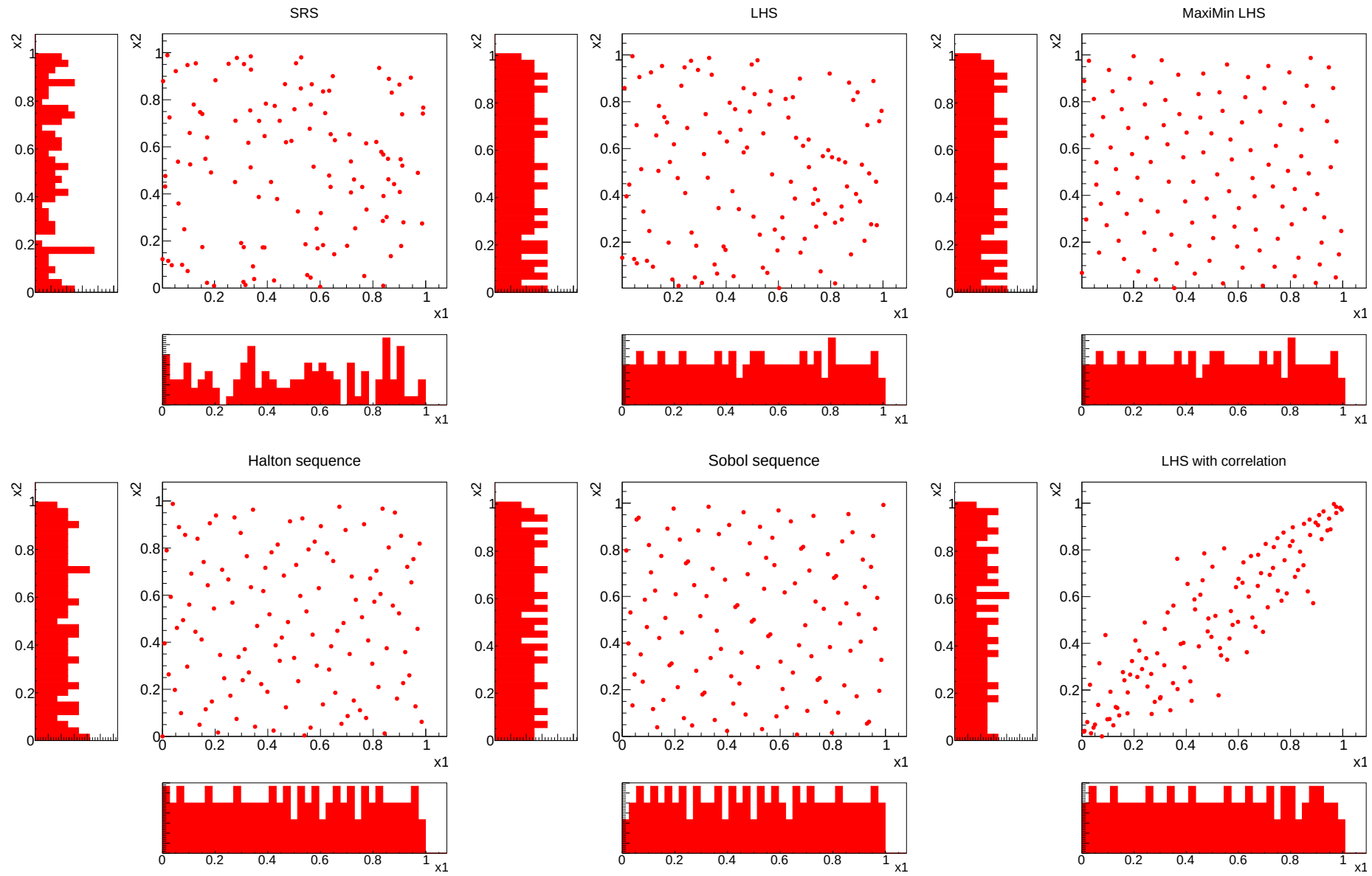
SRS drawing



LHS drawing



Simple case comparison

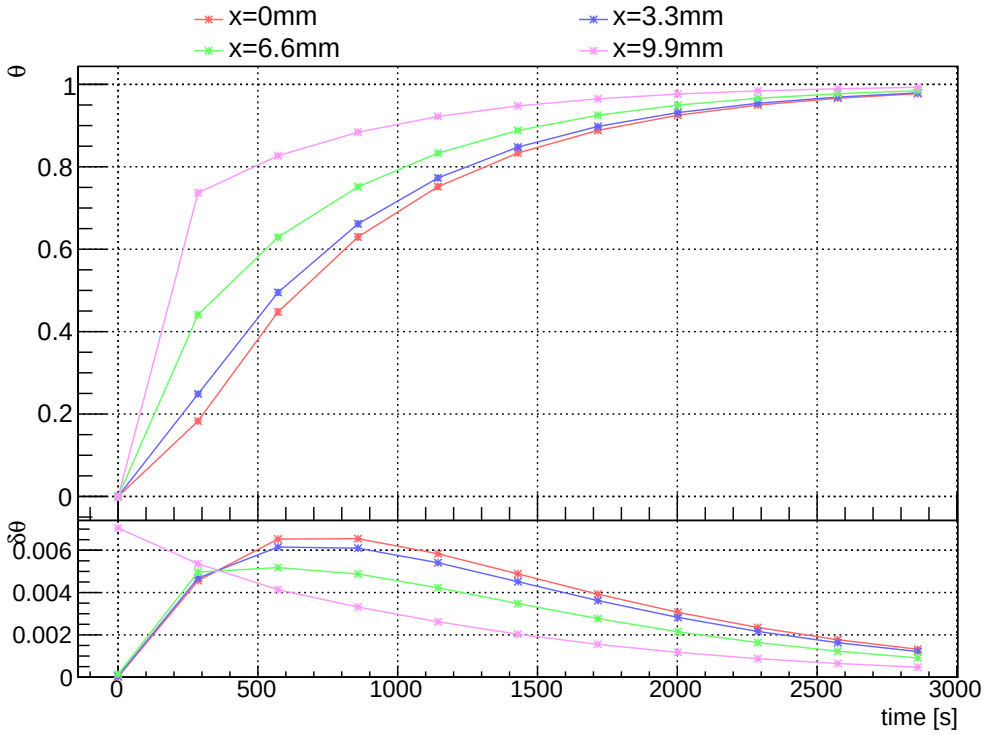


Application to our use-case

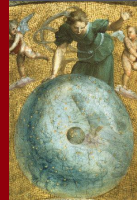
Using a LHS grid, requesting 100 points defining the unknown parameter laws to be gaussian

	Value	Uncertainty
Thickness [m] : e	10×10^{-3}	5×10^{-5}
Thermal conductivity [W.(m.K)$^{-1}$] : λ	0.25	1.5×10^{-3}
Massive thermal capacity [J.(kg.K)$^{-1}$] : C_ρ	1300	15.6
Volumic mass [kg.m^{-3}] : ρ	2200	4.4

Table: Summary of PTFE properties along with their uncertainty.



To go further



The drawing used to get locations is important as, in most cases, the number of allowed estimation is small

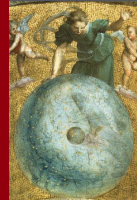
Thorough investigation of the analysis must be done:

- the quantity of interest: quantile measurement, mean / variance of a distribution
- the possibility to increase the dataset if needed
- the input parameter space dimensions.
- ...

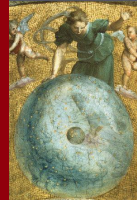
More possibilities

- Create a sub-sample of points representative of the complete provided dataset.
- Can emphasise some low-probability region through importance sampling
- Adaptive designs-of-experiments construction using surrogate-models
- Can put correlation through copulas of different kinds (parametric functions)

Outline



What's a sensitivity analysis



The goals:

- Estimate the fraction of variability of the output Y generated by every input X_i
  - ➙ Using either quantitative or hierarchical results
- Get an idea of which input can be considered useless (if so) and reduce the dimension
- Get an idea on which variable is dominant (if so) and work to reduce the uncertainty on it

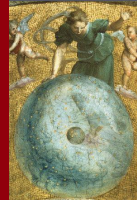
General kind of analysis:

- local ones: variation around only one of the input (mainly deterministic method)
- global ones: more global variation,

Available in Uranie

- finite differences: $\frac{\delta Y}{\delta X_i}(x_0)$ (not discussed here, see use-case macro)
- Screening method like **Morris**
- Regression methods (not discussed here, see use-case macro)
  - on values (Person)
  - on ranks (Spearman)
- Sobol indexes
  - "Sobol/Saltelli" methods (first and total order)
  - Fourier-based : FAST/RBD (first order)

Screening method



Screening methods:

- allow a "rough" first SA, for a smaller computational cost than Monte Carlo methods.
- can be applied to problems of reasonably high dimensions (even above 100 dimensions)
- may be followed by more refined techniques to focus the SA on the more important inputs

The Morris method

- Introduced by M. Morris in 1991
- Consists of varying one input at a time (OAT), but at different starting points
- Uses an average measure of the "elementary effect" (EE) of each input, by observing the effect on the target variable
- Is quantitative, but does not have a direct interpretation in terms of output variance S_{T_i})
- Provides information to classify input factors in three sets:
  - ➙ factors that have negligible effect on Y
  - ➙ factors that have linear effects without interactions
  - ➙ factors that have non-linear effects and/or interactions

Description of the Morris method (1/2)

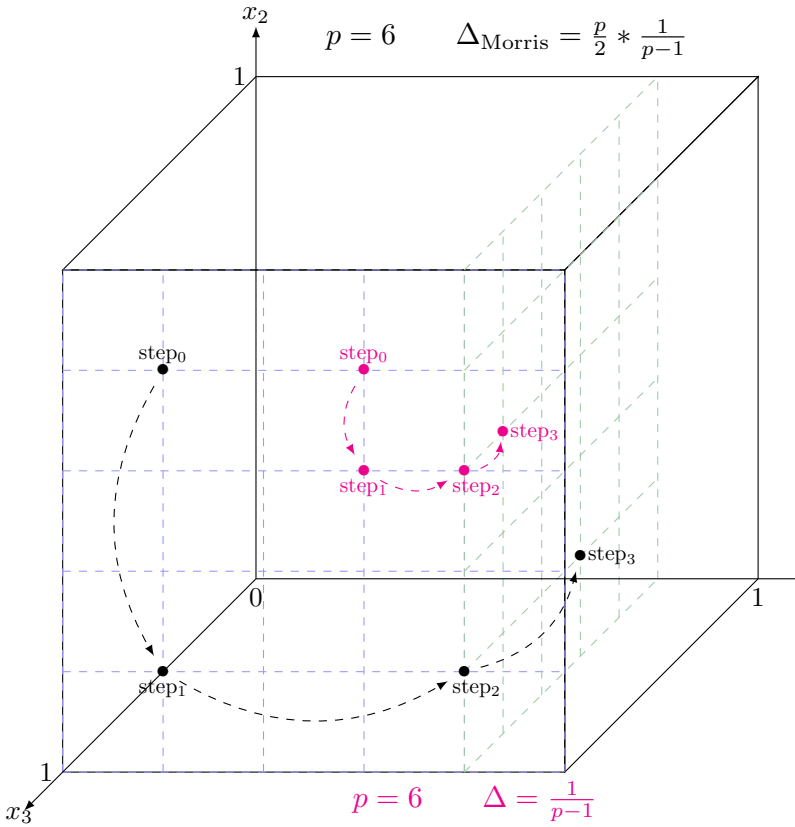
1. Every inputs' range-of-evolution is transformed into [0,1]
2. A p-**level** grid is created in the $[0,1]^{n_X}$ hyper-cube. The allowed value being then $[0, \frac{1}{p-1}, \frac{2}{p-1}, \ldots, 1]$
3. A trajectory t is drawn at random as :
   - a starting point in the hyper-cube (assessment 0 of the code)
   - a direction to a new point (assessment 1 of the code)
   - repeating previous step for all other directions (assessment 2 to n_X of the code)
4. Elementary Effect are computed for every move:

$$EE_i^t = \frac{f(x_1^t, .., x_i^t + \Delta, .., x_{n_X}^t) - f(x_1^t, .., x_i^t, .., x_{n_X}^t)}{\Delta}$$

   Δ is larger than local methods (*e.g.* finite differences)
5. Statistics over n_t EE is computed ($\forall i \in [1, n_X]$)

$$\mu_i = \frac{1}{n_t}\sum_{j=1}^{n_t} EE_i^j \quad \text{and} \quad \mu_i^* = \frac{1}{n_t}\sum_{j=1}^{n_t} |EE_i^j|$$

$$\sigma_i^2 = \frac{1}{n_t - 1}\sum_{j=1}^{n_t} (EE_i^j - \mu_i)^2$$

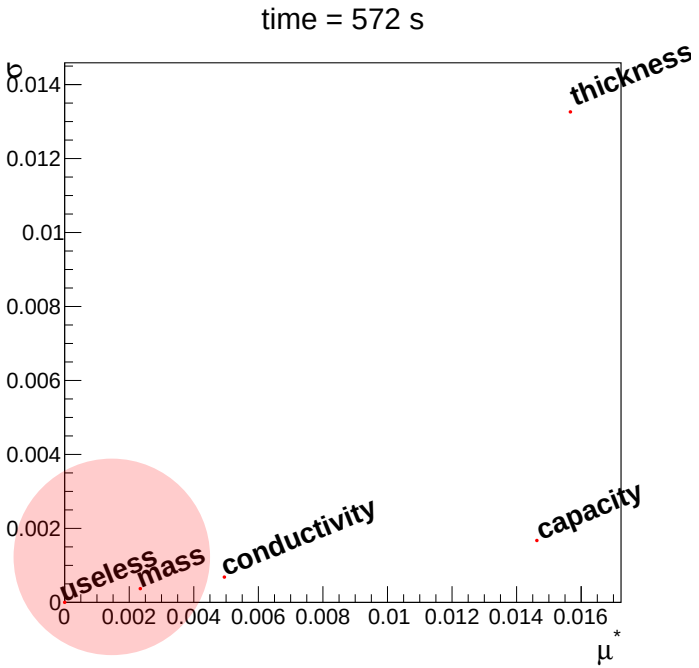


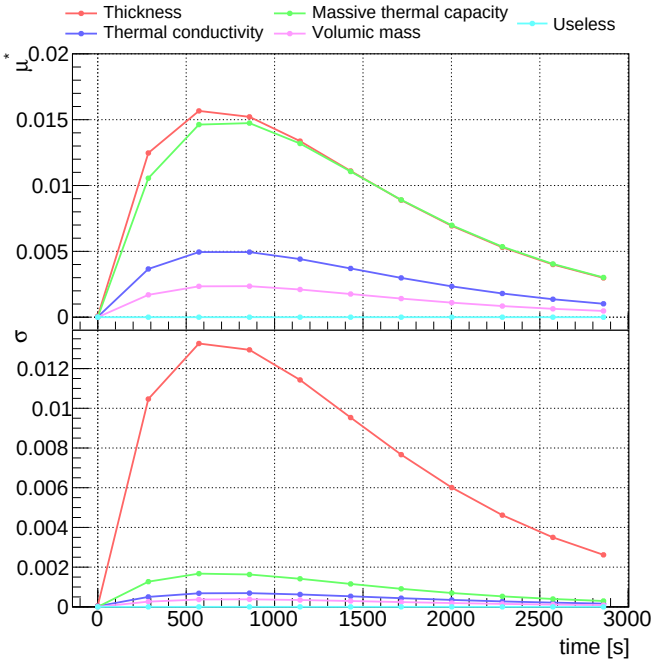
Description of the Morris method (2/2)

Total cost is $n_t(n_X + 1)$ assessment of the code (n_t being at least 4-10)

Interpret these results in σ_i vs μ_i^* (or μ_i) plane

- factors that have negligible effect : both μ^* and σ are small.
- factors that have linear effect, without interactions with other inputs: μ^* is large (all variations have an impact) but σ is small (the impact is the same independently of the starting point).
- factors that have non-linear effects and/or interaction with other inputs: both μ^* and σ are large.





Local modification of the model

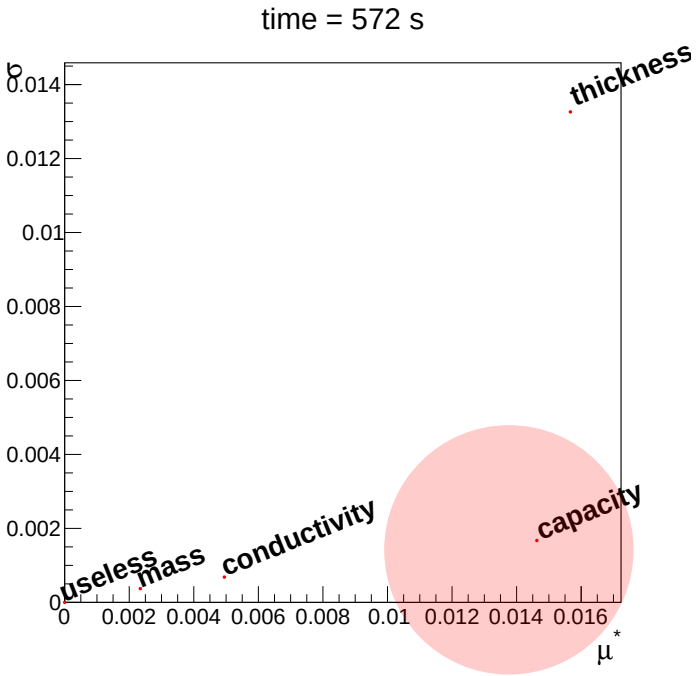
To show the effect of non-used variable, a new input called “useless” has been introduced.

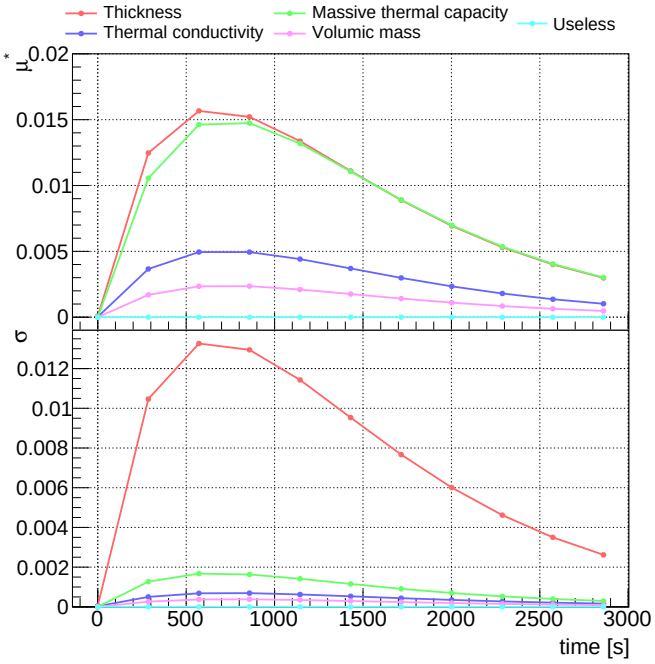
Description of the Morris method (2/2)

Total cost is $n_t(n_X + 1)$ assessment of the code (n_t being at least 4-10)

Interpret these results in σ_i vs μ_i^* (or μ_i) plane

- factors that have negligible effect : both μ^* and σ are small.
- factors that have linear effect, without interactions with other inputs: μ^* is large (all variations have an impact) but σ is small (the impact is the same independently of the starting point).
- factors that have non-linear effects and/or interaction with other inputs: both μ^* and σ are large.





Local modification of the model

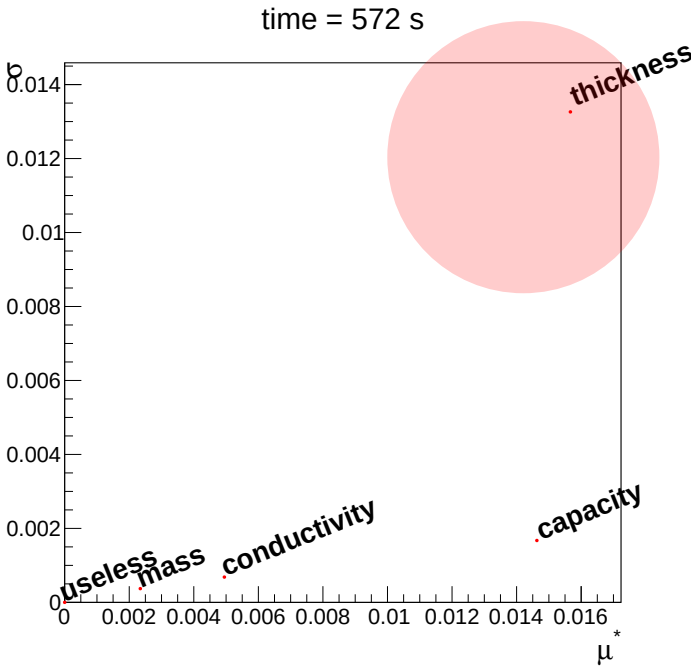
To show the effect of non-used variable, a new input called “useless” has been introduced.

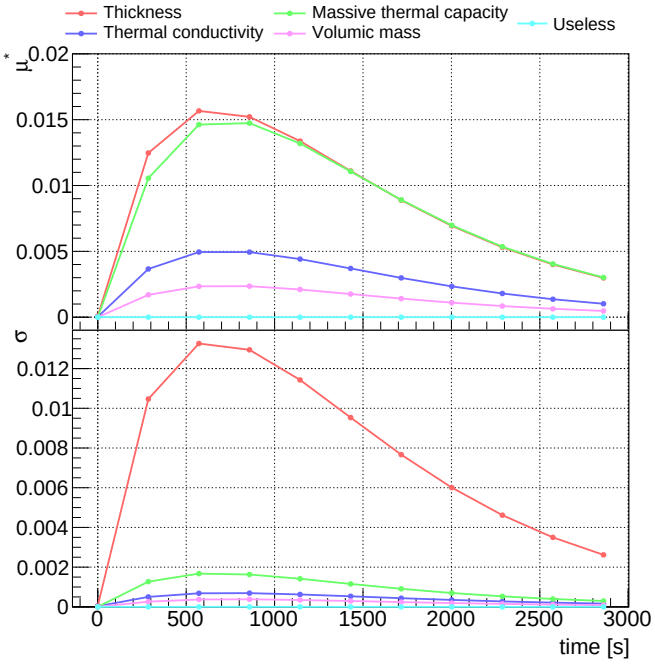
Description of the Morris method (2/2)

Total cost is $n_t(n_X + 1)$ assessment of the code (n_t being at least 4-10)

Interpret these results in σ_i vs μ_i^* (or μ_i) plane

- factors that have negligible effect : both μ^* and σ are small.
- factors that have linear effect, without interactions with other inputs: μ^* is large (all variations have an impact) but σ is small (the impact is the same independently of the starting point).
- factors that have non-linear effects and/or interaction with other inputs: both μ^* and σ are large.

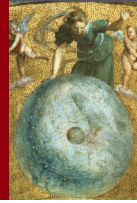




Local modification of the model

To show the effect of non-used variable, a new input called “useless” has been introduced.

HDMR/Sobol/ANOVA decomposition (and other names)



Considering the function f defined by $y = f(x_1, x_2, \ldots, x_{n_X})$, f can be integrated in Ω with a finite expectation then there is a unique decomposition

$$y = f_0 + \sum_{i=1}^{n_X} f_i(x_i) + \sum_{1 \leq i < j \leq n_X} f_{i,j}(x_i, x_j) + \ldots + f_{x_1,x_2,\ldots,x_{n_X}}(x_1, x_2, \ldots, x_{n_X})$$

The decomposition properties

- f_0 is a constant: the expectation of y — $f_0 = \mathbb{E}[f(\mathbf{x})] = \int_\Omega f(\mathbf{x}) d\mathbf{x}$
- the expectation of all terms is null — $\int_\Omega f_l d\mathbf{x} = 0 \ \forall \ l$
- all the summands are orthogonal: — $\int_\Omega f_k f_l d\mathbf{x} = 0 \ \forall \ k \neq l$
- each term can be defined as

$$f_i(x_i) = \mathbb{E}[f(\mathbf{x}|x_i)] - f_0$$

$$f_{i,j}(x_i, x_j) = \mathbb{E}[f(\mathbf{x}|x_i, x_j)] - f_i(x_i) - f_j(x_j) - f_0$$

an so on

Variance decomposition

$$y = f_0 + \sum_{i=1}^{n_X} f_i(x_i) + \sum_{1 \leq i < j \leq n_X} f_{i,j}(x_i, x_j) + \ldots + f_{x_1,x_2,\ldots,x_{n_X}}(x_1, x_2, \ldots, x_{n_X})$$

When the inputs are independent, we can partition $\mathrm{Var}[f(\mathbf{x})]$ into

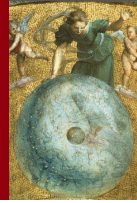
$$\mathrm{Var}[f(\mathbf{x})] = \sum_{i=1}^{n_X} V_i + \sum_{1 \leq i < j \leq n_X} V_{i,j} + \ldots + V_{x_1,x_2,\ldots,x_{n_X}}$$

In this new form, $V_i = \mathrm{Var}[f_i(x_i)] = \int f_i^2(x_i) dx_i$... so if we normalise

$$1 = \sum_{i=1}^{n_X} S_i + \sum_{1 \leq i < j \leq n_X} S_{i,j} + \ldots + S_{x_1,x_2,\ldots,x_{n_X}}$$

Leading to $2^{n_X} - 1$ coefficients

Estimate and interpret the coefficients



First order coefficients: $S_i = \dfrac{\mathrm{Var}[\mathbb{E}[y|x_i]]}{\mathrm{Var}[Y]}$

Cope for the impact of x_i on y **without considering the interaction between inputs**

- $\sum S_i \leq 1$
- $\sum S_i = 1$ for a purely additive model
- $1 - \sum S_i$ quantifies the presence of interactions

4-dimensional problem $\Leftrightarrow 1 = S_1 + S_2 + S_3 + S_4 + S_{1,2} + S_{1,3} + S_{1,4} + S_{2,3} + S_{2,4} + S_{3,4} + S_{1,2,3} + S_{1,2,4} + S_{1,3,4} + S_{2,3,4} + S_{1,2,3,4}$

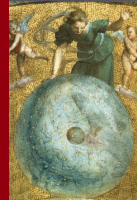
Total order coefficients: taking all effect into account

$$\text{Example for entry } 1 : S_{T_1} = S_1 + S_{1,2} + S_{1,3} + S_{1,4} + S_{1,2,3} + S_{1,2,4} + S_{1,3,4} + S_{1,2,3,4}$$

$$S_{T_i} = 1 - \frac{\mathrm{Var}[\mathbb{E}[y|x_{\sim i}]]}{\mathrm{Var}[Y]} \quad \text{where} \quad \sim i \text{ are all sets not including i}$$

- $S_{T_i} - S_i$ is a measure of interactions with any other inputs
- $\sum S_{T_i} \geq 1$. If this is an equality, the model is perfectly additive.

The Saltelli methods



This method starts with two matrices M and N of size n_S by n_X.

$$M = \begin{pmatrix} m_{1,1} & \dots & m_{1,i} & \dots & m_{1,n_X} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ m_{j,1} & \dots & m_{j,i} & \dots & m_{j,n_X} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ m_{n_S,1} & \dots & m_{n_S,i} & \dots & m_{n_S,n_X} \end{pmatrix} \quad - \text{n}_\text{S} \rightarrow \quad \begin{pmatrix} f(m_{1,1},\dots,m_{1,i},\dots,m_{1,n_X}) \\ \vdots \\ f(m_{j,1},\dots,m_{j,i},\dots,m_{j,n_X}) \\ \vdots \\ f(m_{n_S,1},\dots,m_{n_S,i},\dots,m_{n_S,n_X}) \end{pmatrix}$$

$$N = \begin{pmatrix} n_{1,1} & \dots & n_{1,i} & \dots & n_{1,n_X} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ n_{j,1} & \dots & n_{j,i} & \dots & n_{j,n_X} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ n_{n_S,1} & \dots & n_{n_S,i} & \dots & n_{n_S,n_X} \end{pmatrix} - \text{M's i}^{\text{th}} \text{ input in N} \rightarrow \quad N_i = \begin{pmatrix} n_{1,1} & \dots & n_{1,i-1} & m_{1,i} & n_{1,i+1} & \dots & n_{1,n_X} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ n_{j,1} & \dots & n_{j,i-1} & m_{j,i} & n_{j,i+1} & \dots & n_{j,n_X} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ n_{n_S,1} & \dots & n_{n_S,i-1} & m_{n_S,i} & n_{n_S,i+1} & \dots & n_{n_S,n_X} \end{pmatrix} - \text{n}_\text{S} \times \text{n}_\text{X} \rightarrow \begin{pmatrix} f(n_{1,1},\dots,n_{1,i-1},m_{1,i},n_{1,i+1},\dots,n_{1,n_X}) \\ \vdots \\ f(n_{j,1},\dots,n_{j,i-1},m_{j,i},n_{j,i+1},\dots,n_{j,n_X}) \\ \vdots \\ f(n_{n_S,1},\dots,n_{n_S,i-1},m_{n_S,i},n_{n_S,i+1},\dots,n_{n_S,n_X}) \end{pmatrix}$$

$$\downarrow \text{n}_\text{S} \text{ (for } \text{S}_{\text{T}_\text{i}}\text{)}$$

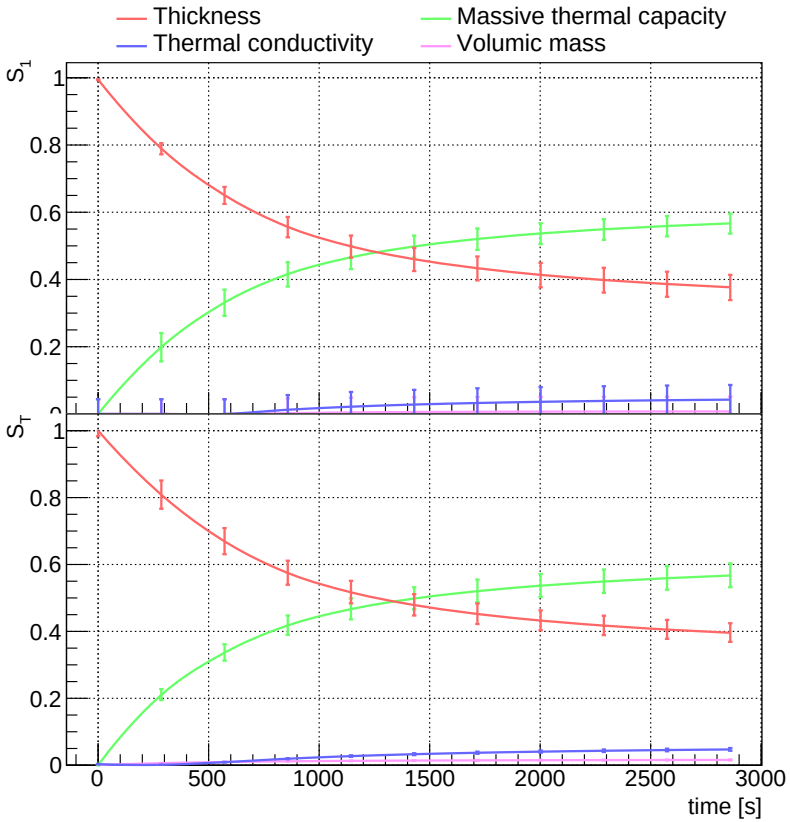
$$\begin{pmatrix} f(n_{1,1},\dots,n_{1,i},\dots,n_{1,n_X}) \\ \vdots \\ f(n_{j,1},\dots,n_{j,i},\dots,n_{j,n_X}) \\ \vdots \\ f(n_{n_S,1},\dots,n_{n_S,i},\dots,n_{n_S,n_X}) \end{pmatrix}$$

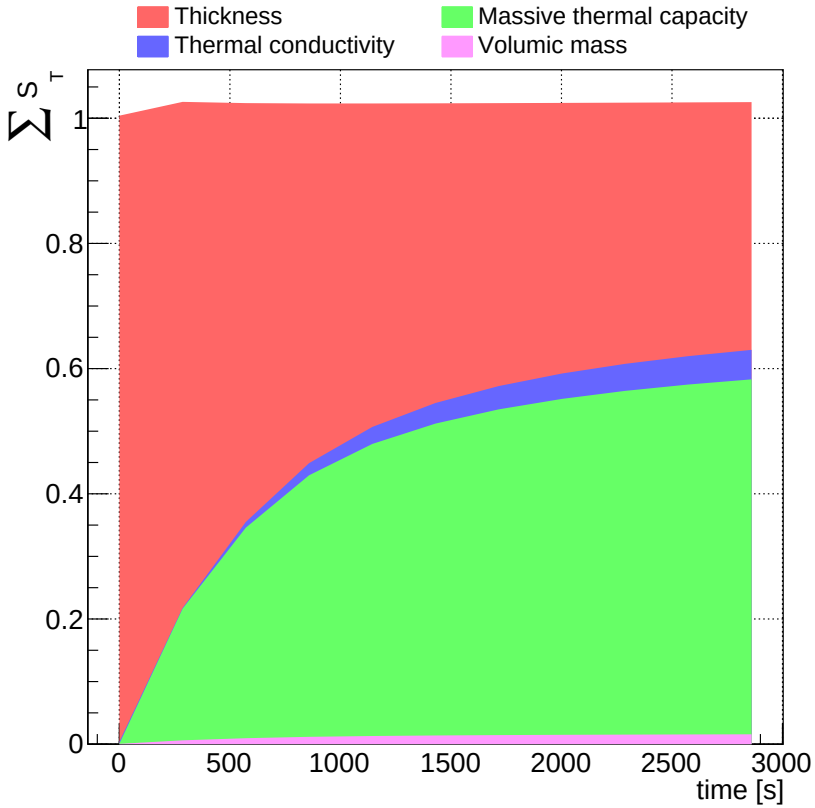
Estimation of the indexes

- $S_i = \dfrac{\text{Var}[\mathbb{E}(y|x_i)]}{\text{Var}[y]}$. Comparing M's and N_i's results
  - ➡ Cost is $n_S(n_X + 1)$
- $S_{T_i} = 1 - \dfrac{\text{Var}[\mathbb{E}[y|x_{\sim i}]]}{\text{Var}[Y]}$. Comparing N's and N_i's results
  - ➡ Cost is n_S

➡ The total cost of this method is $n_S(n_X + 2)$

Application to our use-case





For a specific slice of x

Ten time steps are computed to show the impact of every inputs through time.

- Same hierarchy as for the Morris methods
- Same values for first and total order $\Leftrightarrow$ no interactions between inputs.

The FAST method

Principle

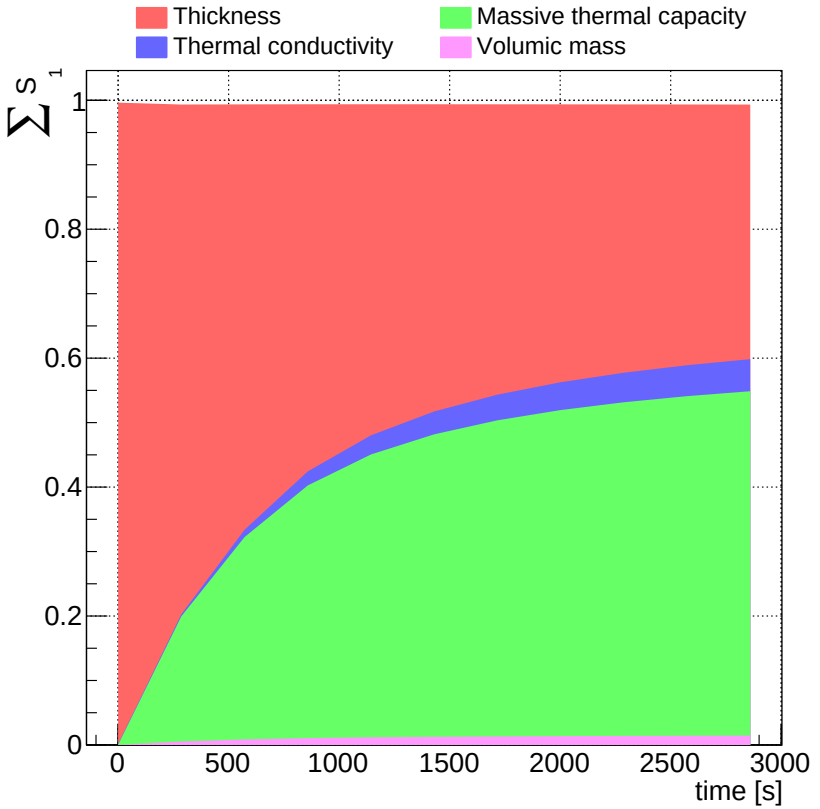
- DOE done with optimised-space-filling curve
- Inputs have different frequencies $(\omega_1, .., \omega_{n_X})$ free of interference up to a given order M=6

$$x_i(s_j) = G_i(\sin(\omega_i s_j)) \; \forall i = 1, .., n_X$$

$$\text{where } s_j \in [-\pi, \pi] \; \forall j = 1, .., n_S$$

- Evaluate the model for each points
- Fourier spectrum is calculated on y at frequencies $(\omega_i, 2\omega_i, .., M\omega_i)$
- Sensitivity index S_i is

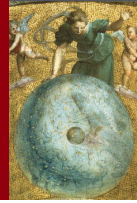
$$S_i = \frac{\sum_{k=1}^{M}(A_{k\omega_i}^2 + B_{k\omega_i}^2)}{\sum_{i=1}^{n_X}\sum_{k_i=1}^{M}(A_{k_i\omega_i}^2 + B_{k_i\omega_i}^2)}$$



Previous conclusions hold with these results

Outline

Reminder on optimisation



To achieve an optimisation, one must have:

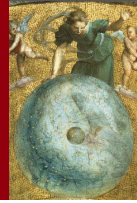
- one or more **criteria** that we will seek to minimise (or maximise)
- **parameters** whose values influence the criteria
- possibly, some **constraints** on the values of these parameters
- an optimisation **algorithm**, to decide the new value of the parameters to improve the criteria.

The optimisation is a complex problem, and there is no "universal" algorithm. Each study has its own peculiarities and it often takes a bit of trial and error before you find an interesting solution.

Various optimisation algorithms can be divided into two categories:

- **local methods**: allow mono-criterion optimisation, with or without constraints. Generally computationally efficient, but not parallelisable and tend to be trapped in local optima.
- **global methods**: allow multi-criteria optimisation with or without constraints. Suitable for problems with many local optima, computationally expensive but easily parallelisable

Single-objective optimisation problem

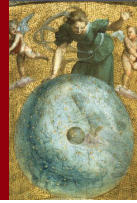


Two main packages to deal with optimisation

- **Minuit**: ROOT's package for SO problem, without constraint. It provides two algorithms
  - Simplex: does not use first derivatives, insensitive to local optima, but no guarantee of convergence.
  - Migrad: fairly sophisticated gradient descent algorithm, able to escape from some local optima.

- **NLopt**: Package for nonlinear optimisation. Provides algorithms for SO problem, with or without constraint. The available algorithms through Uranie are:
  - Cobyla (*Constrained Optimisation BY Linear Approximation)*
  - *Bobyqa (Bounded Optimisation BY Quadratic Approximation)*
  - *Praxis (PRincipal AXIS method)*
  - *MMA (Method of Moving Asymptotes)*
  - *SLSQP (Sequential Least-Squares Quadratic Programming)*
  - *LBFGS (Limited memory Broyden-Fletcher-Goldfarb-Shanno algorithm)*
  - *Newtown*
  - *VariableMetric*
  - *NelderMead*
  - *Subplexe*

Example of SO problem: code calibration



Performing a code calibration $\Leftrightarrow$ finding optimal code's parameters minimising a "distance" between reference values and computations.

Distances implemented in Uranie:

- the root mean square deviation,

$$\text{obj} = \frac{\alpha}{n_S} \sum_{i=1}^{n_S} (y_i^\star - \hat{y}_i)^2$$

- the weighted root mean square deviation.

$$\text{obj} = \alpha \sum_{i=1}^{n_S} \frac{(y_i^\star - \hat{y}_i)^2}{\sigma_i^2}$$

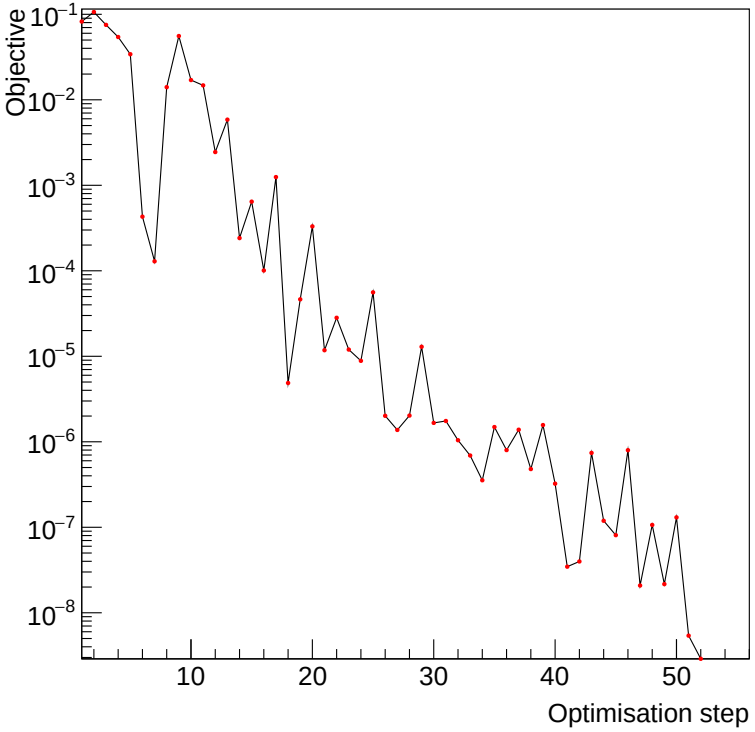
To calibrate a code in Uranie, one needs:

- Reference values of the output of the code over a set of data in a file
  - ➟ 30 θ computations is provided with "unknown" e and h value (e_{truth}=0.01, h_{truth}=100)
- The command to run the code over the same reference data.
  - ➟ Create a small new program that embed the code to run it over the 30 measurements
- To call dedicated method so that a distance is computed from reference

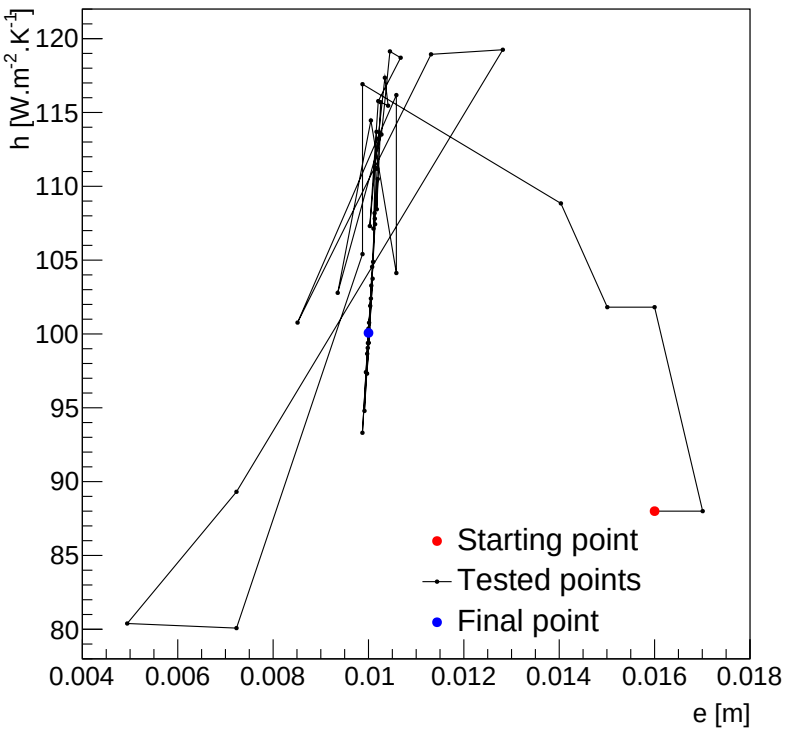
```
addObjective(ojbname, tdsref, yrefname, outfileTested, yhatTested);
```

Application to our use-case

Optimisation criterion



Trajectory in parameter space



Observations

- The sensitivity with respect to e seems larger (expected)
- Convergence toward the nominal values

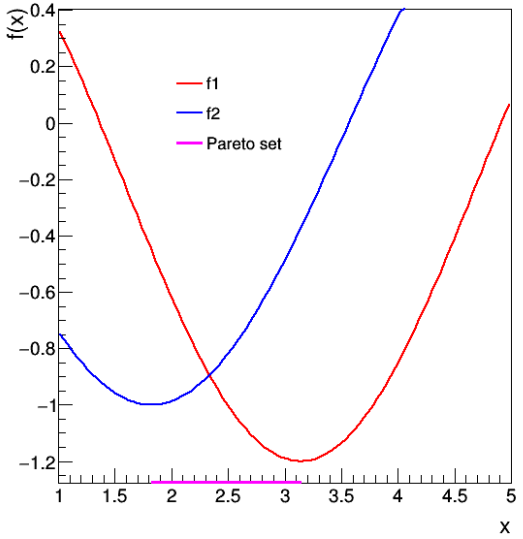
Multi-objective optimisation vocabulary

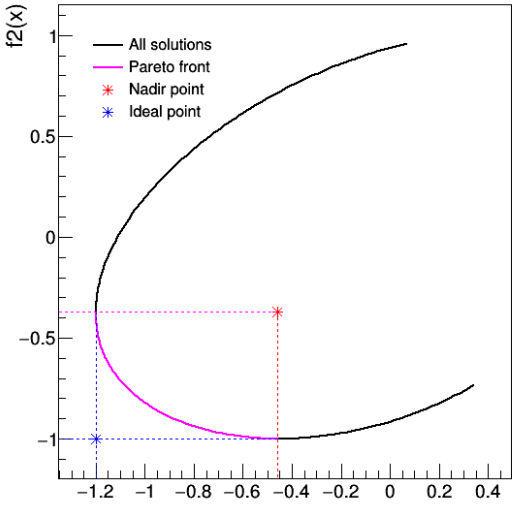
- When several criteria must be taken into consideration, the solution kept is always a **compromise**.
- Multi-criteria optimisation consists in finding a set of "**acceptable**" solutions according to criteria and constraints posed.
- In the diagram at top opposite, we search the values of x which optimize the criteria f_1 and f_2 .
- Let x_1 and x_2 the minimum of f_1 and f_2 , respectively. For all $x_i < x_2$, if $x_2 > x_j > x_i$ then

$$f_1(x_j) < f_1(x_i) \text{ and } f_2(x_j) < f_2(x_i)$$

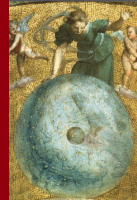
  Also true for $x_i > x_1$ if $x_1 < x_j < x_i$: x_j **dominates** x_i
- However for $x_2 < x_i < x_1$ no value of x_j does improve both criteria simultaneously (previous equation).
- Compromise solutions are to be found in the area $x_2 < x < x_1$ called the Pareto zone ($\mathcal{P}$).
- The group of corresponding solutions in the space of criteria (listed below opposite) is called the Pareto front. They are said to be **non-dominated**: if $x_a, x_b \in \mathcal{P}$ then

$$f_1(x_a) < f_1(x_b) \text{ and } f_2(x_a) < f_2(x_b) \text{ is impossible}$$

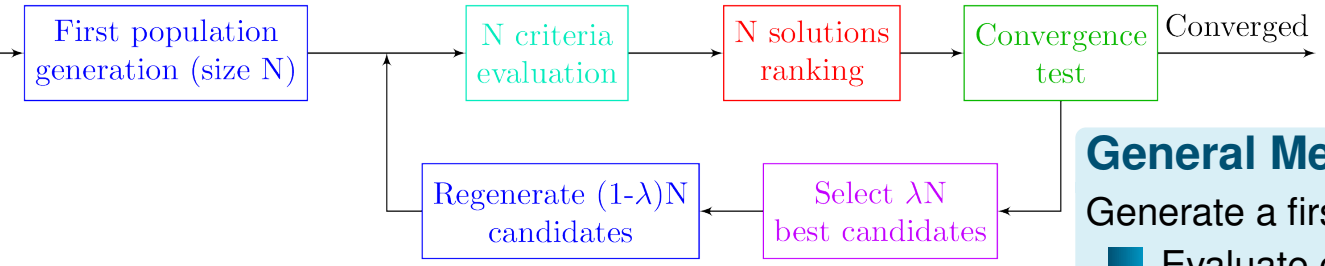




Example of multi-objective optimisation



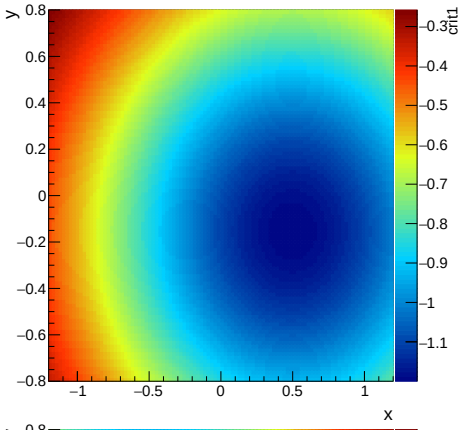
With two antagonist criteria (crit1, crit2) depending both on x and y

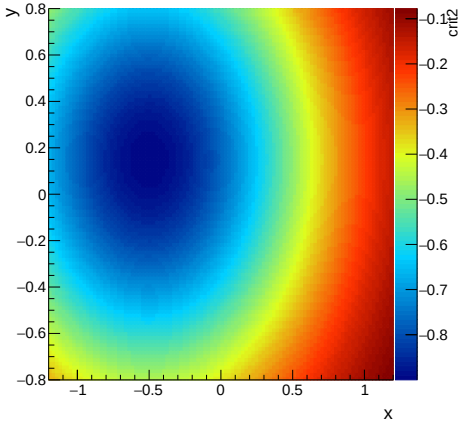


General Methodology

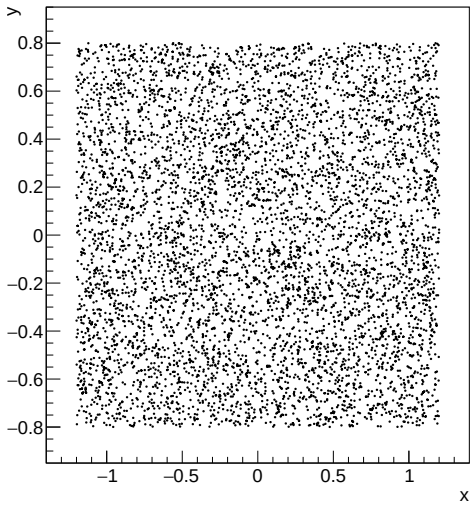
Generate a first family of N people and then

- Evaluate criteria for all people
- Rank them according to criteria
- Test convergence.
  - Converged: stop
- Create new people from the best λN
- Start all over

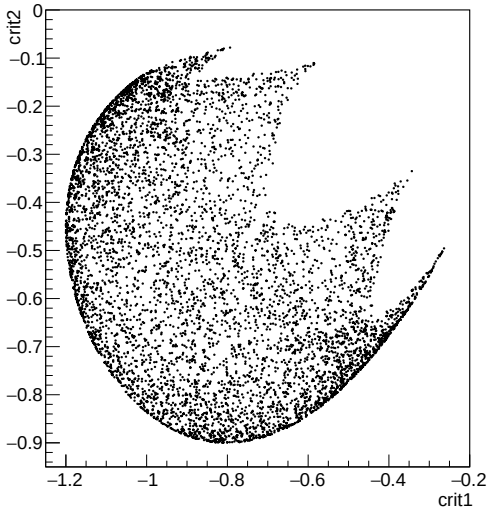




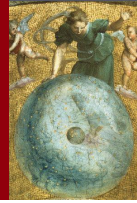
• First family Parameters space



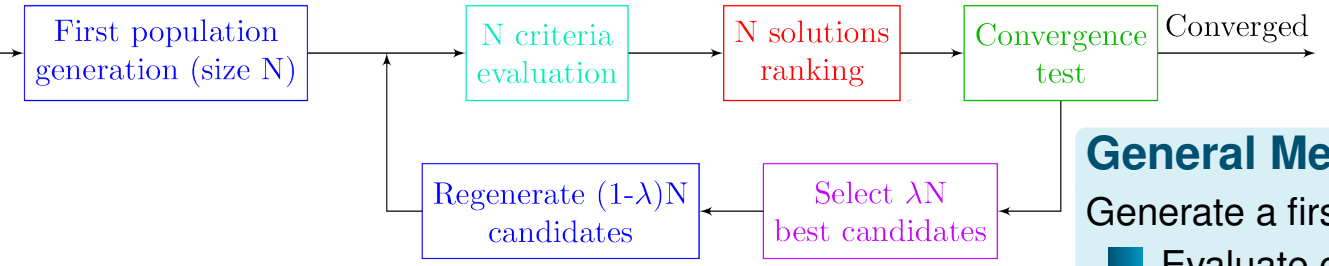
• First family Objectives space



Example of multi-objective optimisation



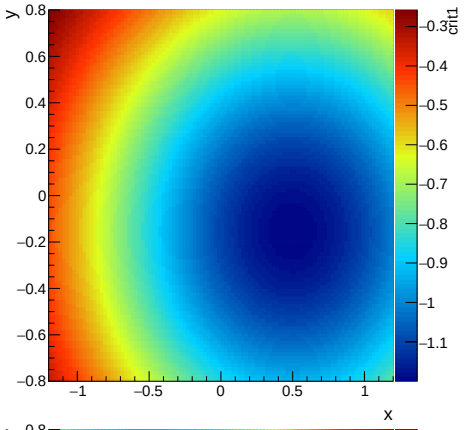
With two antagonist criteria (crit1, crit2) depending both on x and y

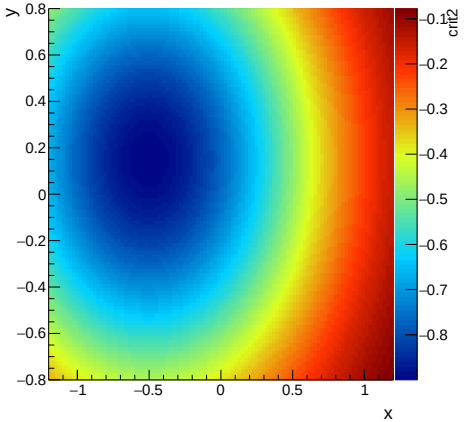


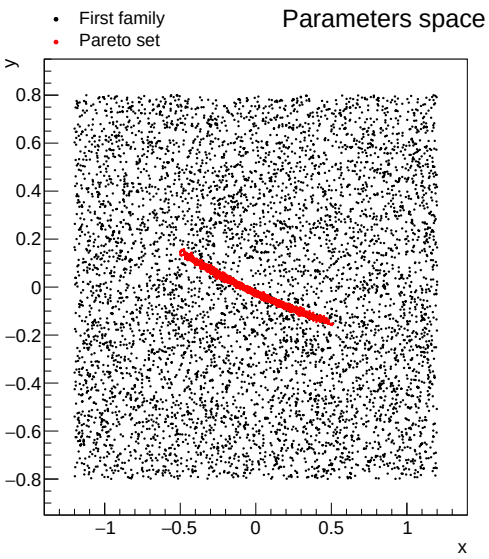
General Methodology

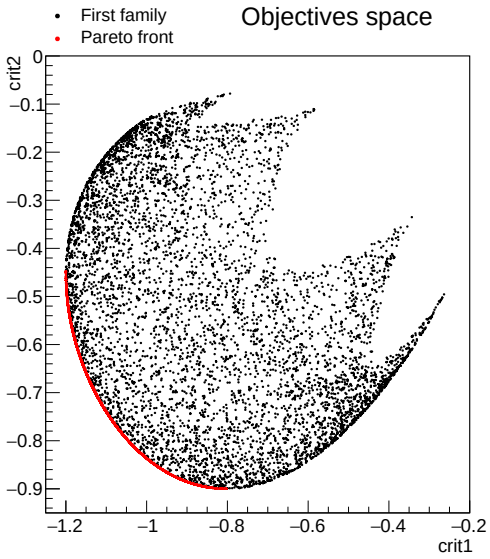
Generate a first family of N people and then

- Evaluate criteria for all people
- Rank them according to criteria
- Test convergence.
  - Converged: stop
- Create new people from the best λN
- Start all over









Outline

Combining techniques

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

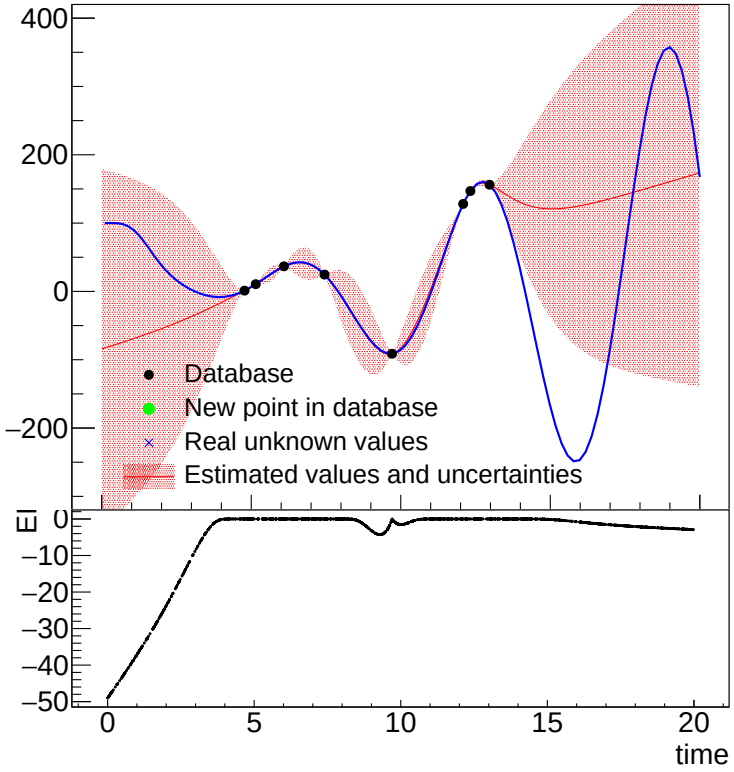
Efficient Global Optimisation (EGO)

From a small database (here 8 points)

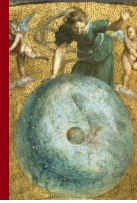
- Construct a kriging model
- Compute the Expected Improvement with the kriging model
  - → using genetic algorithm to get the minimum t^*
- Compute the real new value with the code at t^*
- Reconstruct the kriging on the database + t^*
- Continue this process iteratively...

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.

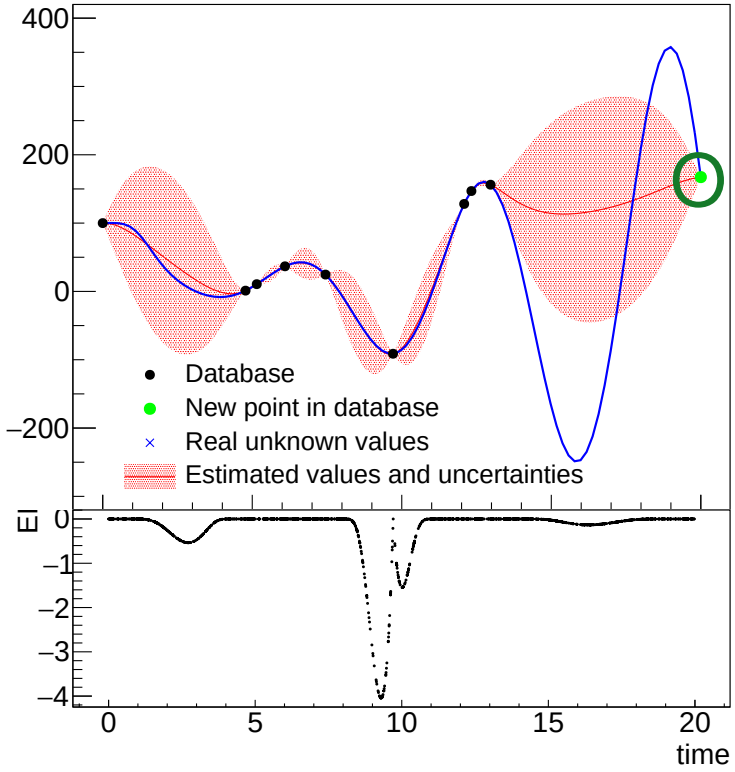
Efficient Global Optimisation (EGO)

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.

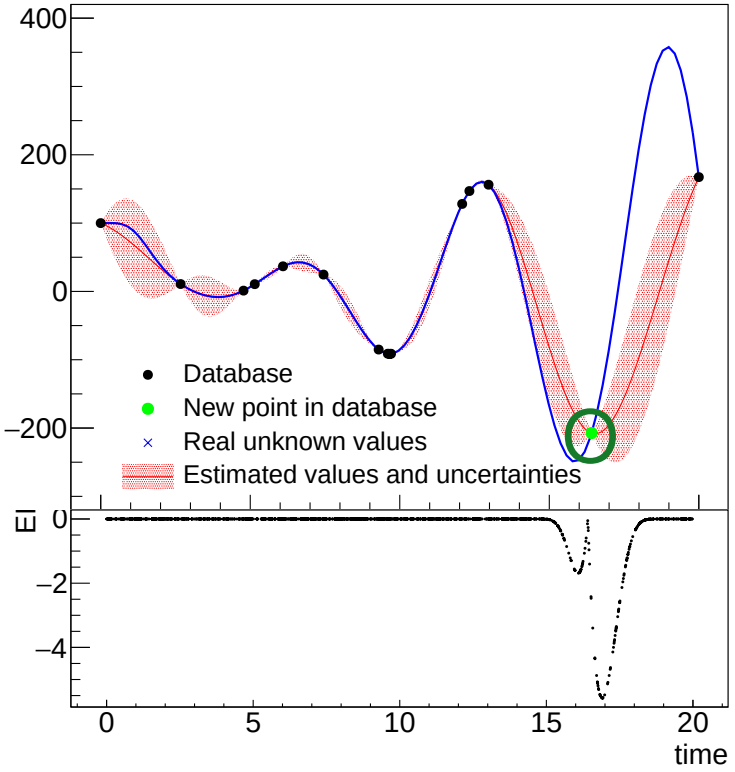
Efficient Global Optimisation (EGO)

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.

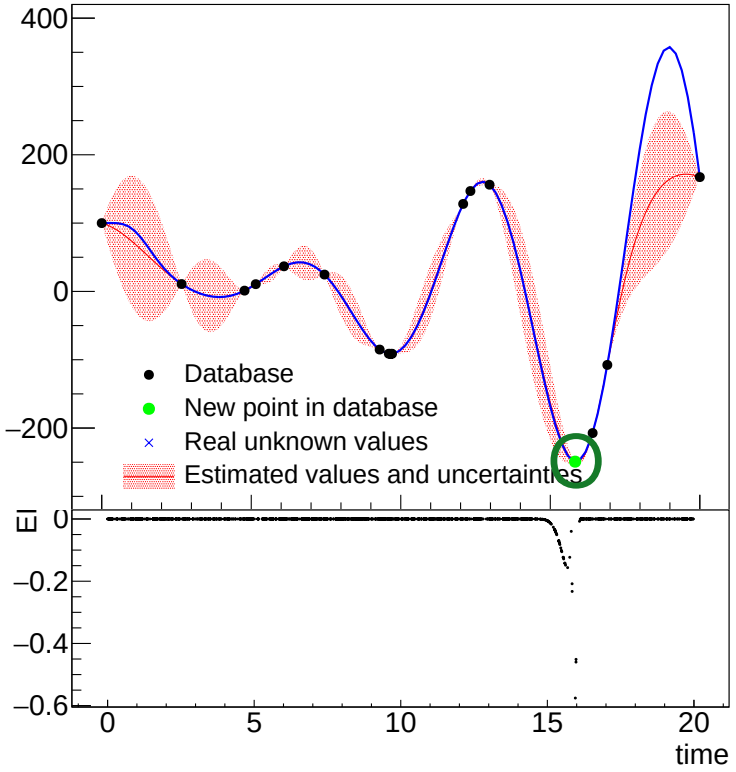
Efficient Global Optimisation (EGO)

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...

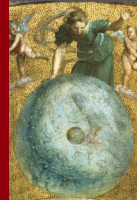
Ongoing work to parallelise this process

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



Outline

ROOT v6 & Jupyter Notebook



Moving to ROOT v6 brings :

- the opportunity to be C++11 compliant
- the clang compiler in LLVM
- the new C++11 interpreter **cling**

Possibility to use Jupyter Notebook

- The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualisations and explanatory text.
- It allows also different kernels
  - ➙ IPython is the reference Jupyter kernel, providing a powerful environment for interactive computing in Python
  - ➙ ROOT (Cling) now offers a Jupyter kernel (54th entry in the list with other languages like : Bash, Matlab, Scilab, PHP, Perl, Erlang, Go, Javascript, ..)

Caveat: not yet available for Windows.

ROOT v6 & Jupyter Notebook

Moving to

- the oppor
- the clang
- the new C

Possibility

- The Jupy
  contain li
- It allows a
  → IPyth
  computin
  → ROO
  Bash, Ma

that

e

ke :

jupyter modelerANNDirichlet.C.nbconvert (autosaved)

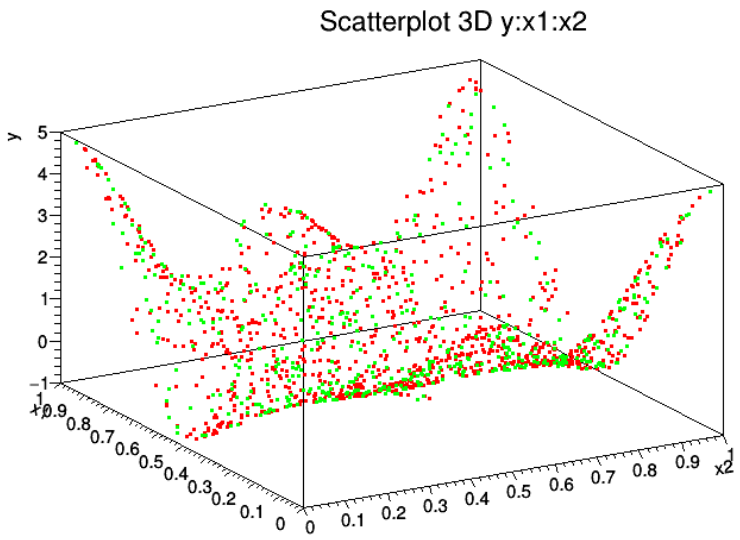
Logout

File Edit View Insert Cell Kernel Help Not Trusted No Kernel

Run Markdown

In [7]:

```
TCanvas c1(Form("canDirichlet%02d", gDimDirichlet), Form("ANN for the Dirichlet kernel %dD", gDimDirichlet), 65,594,700,5
tdsValidate->draw(Form("%s:%s", gsY.Data(), gsX.Data()));
gPad->SetGrid(1,1);

tdsLearning->getTuple()->SetMarkerColor(kGreen);
tdsLearning->draw(Form("%s:%s", gsY.Data(), gsX.Data()),"","same");
c1.Draw();
```



Create the ann modeler

In [8]:

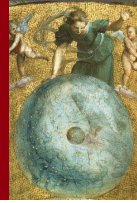
```
TANNModeler* tann = new TANNModeler(tdsLearning, Form("%s,%d,%s", gsX.Data(), nH, gsY.Data()));
tann->setFcnTol (1e-6);
tann->setNormalization  (TANNModeler::kMinusOneOne);
tann->train(5, 4);
cout << " **  EQM  :  Learn[" << tann->getEQM("learn") << "]  test[" << tann->getEQM("test") << "]" << endl;
```

```
** TANNModeler::train niter[5] ninit[4]
** init the ANN
** Input  (1) Name[x1] Min[0.000277464] Max[0.999091]
** Input  (2) Name[x2] Min[0.00131526] Max[0.998016]
** Output     Name[y] Min[-0.821961] Max[4.82086]
```

DE LA RECHERCHE À L'INDUSTRIE

cea

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
- Any justified request from the community

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)

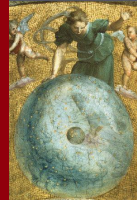
More information can be found in our recent paper (submitted to CPC):
http://arxiv.org/abs/1803.10656

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

Backup outline



Backup

- Use-case, theoretical explanations
- Basic statistic of a sample
- PDF and CDF
- Matern correlation function
- Example of CP expansion

- QMC: Van der Corput
- QMC: the dimension problem
- Example of interaction
- About the Morris method
- Vizir genetic in a nutshell
- The expected improvement definition

More details on the model I

Studying the evolution of the temperature within the sheet in fact consists in solving the heat equation which can be written as follows:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \tag{1}$$

In this equation α [m^2.s^{-1}] is the thermal diffusivity which is defined by

$$\alpha = \frac{\lambda}{\rho C_\rho} \tag{2}$$

where λ is the thermal conductivity [W.m^{-1}.K^{-1}], C_ρ is the massive thermal capacity [J.kg^{-1}.K^{-1}] and ρ is the volumic mass [kg.m^{-3}]. There are three conditions used to resolve the heat equation, the first one being the initial temperature

$$T(x, t = 0) = T_i \tag{3}$$

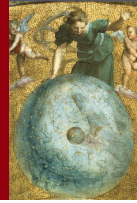
the second one relies on the flow being null at the centre of the sheet

$$\left.\frac{\partial T}{\partial x}\right|_{x=0} = 0 \tag{4}$$

while the last one relies on the thermal flow equilibrium at the surface of the sheet

$$-\lambda \left.\frac{\partial T}{\partial x}\right|_{x=e} = h(T(x = e, t) - T_\infty) \tag{5}$$

More details on the model II



Usually, the thermal coupling between a fluid and a solid structure is characterised by the thermal exchange coefficient h [W.m^{-2}.K^{-1}]. This coefficient allows to free oneself from a complete description of the fluid, when one is only interested in the thermal evolution of the structure (and *vice-versa*). Its value depends on the dimension of the complete system, on the physical properties of both the fluid and the structure, on the liquid flow, on the temperature difference... The thermal exchange coefficient is characterised by the Nusselt number (N_u), from the fluid point of view, and by the Biot number (B_i), from the structure point of view. In the rest of this paper, the latter will be discussed and used thanks to the relation

$$B_i = \frac{he}{\lambda} \tag{6}$$

In the specific case where the thermal exchange coefficient, h and the fluid temperature T_∞ can be considered constant, Eqn 1 has an analytic solution for all initial conditions (all the more so for the one stated in Eqn 3), when it respects the flow conditions defined in Eqns 4 and 5. The resulting analytic form is usually express in terms of thermal gauge θ, which is defined as

$$\theta(x,t) = \frac{T(x,t) - T_i}{T_\infty - T_i} \tag{7}$$

More details on the model III

The complete form is the following infinite serie

$$\theta(x_{ds}, t_{ds}) = 2\sum_{n=1}^{\infty} \beta_n \cos(\omega_n x_{ds}) \exp(-\frac{1}{4}\omega_n^2 t_{ds}) \tag{8}$$

where the original parameters have been changed to dimensionless ones

$$x_{ds} = x/e \tag{9}$$

$$t_{ds} = \frac{t}{t_D} = t \times \frac{4\alpha}{e^2} = t \times \frac{4\lambda}{e^2 \rho C_\rho} \tag{10}$$

Given this, the elements in the serie (Eqn 8) can be written

$$\beta_n = \frac{\gamma_n \sin(\omega_n)}{\omega_n(\gamma_n + B_i)} \tag{11}$$

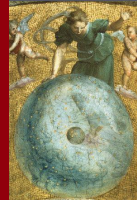
where

$$\gamma_n = \omega_n^2 + B_i^2 \tag{12}$$

and ω_n are solutions of the following equation

$$\omega_n \tan(\omega_n) = B_i \tag{13}$$

Uni-variate case: “Location” parameters



The effect of the "location" parameter is to translate the graph relative to the standard distribution

- **Mean** μ:

$$\mu = \frac{1}{n_S} \sum_{i=1}^{n_S} x_i$$

- **Mode M**: Value where the probability is the greatest value
- α**-Quantile** q_{alpha} **with** $\alpha \in [0, 1]$: defined as

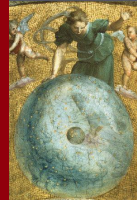
$$\mathbb{P}[X \leq q_\alpha] = \alpha$$

- **Median** $q_{0.5}$: it is the 0.5-quantile defined as

$$\mathbb{P}[X \leq q_{0.5}] = 0.5 = \mathbb{P}[X \geq q_{0.5}]$$

- **Quartiles**: $q_{0.25}$, $q_{0.5}$, $q_{0.75}$
- **Extreme values** : Min and Max

Uni-variate case: “Dispersion” parameters



The effect of a "dispersion" parameter is to stretch/shrink the standard distribution

- **Variance** $\mathrm{Var}(X)$: measure of spread in the data about the mean $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}(X))^2]$, and can be estimated by:

$$\mathrm{Var}(X) = \frac{1}{n_S - 1} \sum_{i=1}^{n_S} (x_i - \mu)^2$$

- **Standard Deviation** σ: to have an information in the same unit as the variable

$$\sigma = \sqrt{\mathrm{Var}(X)}$$

- **Coefficient of Variation** δ: σ does not indicate the degree (%) of dispersion around the mean value μ, a non-dimensional term can be introduced:

$$\delta = \frac{\sigma}{\mu}$$

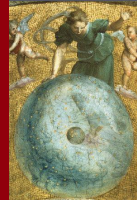
- **Range** R:

$$R = Max - Min$$

- **Interquartile interval H**:

$$H = q_{0.75} - q_{0.25}$$

Uni-variate case: “Shape” parameters



Any parameter of a PDF that affect the shape of a distribution rather than simply shifting it or stretching/shrinking it.

- **Moment order p**: $\mu_p = \mathbb{E}[(X - \mathbb{E}(X))^p]$

$$\mu_p = \frac{1}{n_S}\sum_{i=1}^{n_S}(x_i - \mu)^p$$

- **Skewness**: γ_1 is a measure of the asymmetry of the PDF

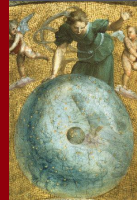
$$\gamma_1 = \mathbb{E}[\left(\frac{X-\mu}{\sigma}\right)^3] = \frac{\mu_3}{\sigma^3} = \frac{\mathbb{E}(X^3) - 3\mu\sigma^2 - \mu^3}{\sigma^3}$$

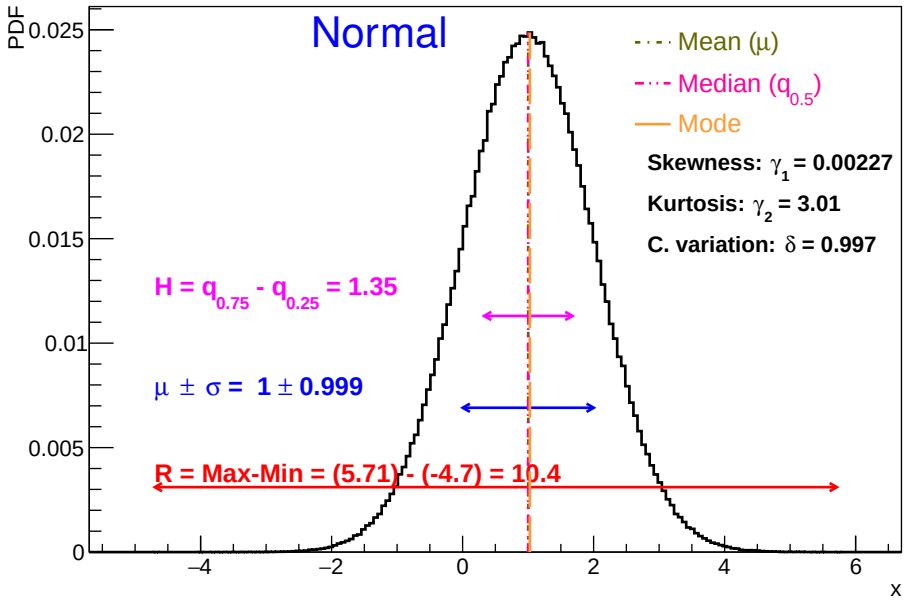
- **Kurtosis**: γ_2 is a measure of the "peakedness" of the PDF

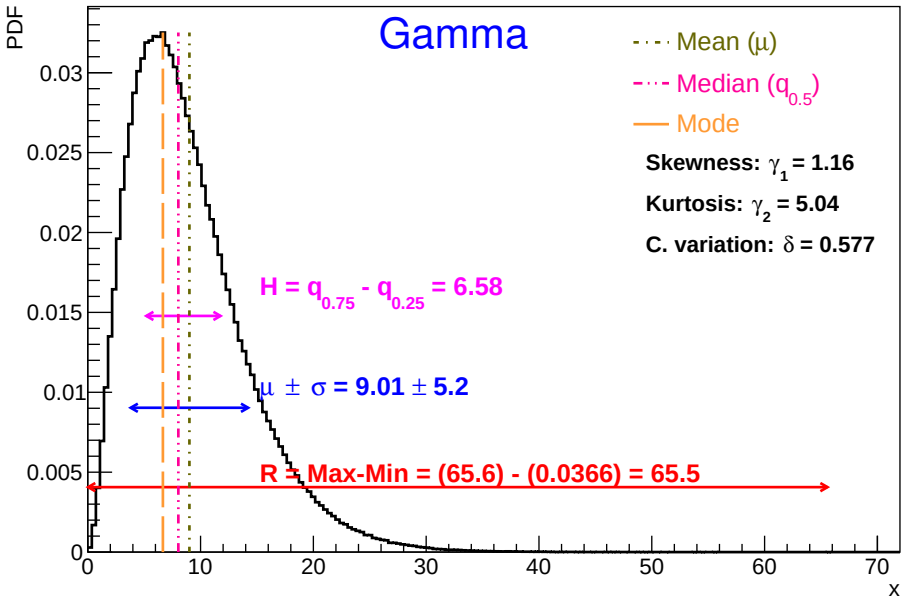
$$\gamma_2 = \frac{\mu_4}{\sigma^4};$$

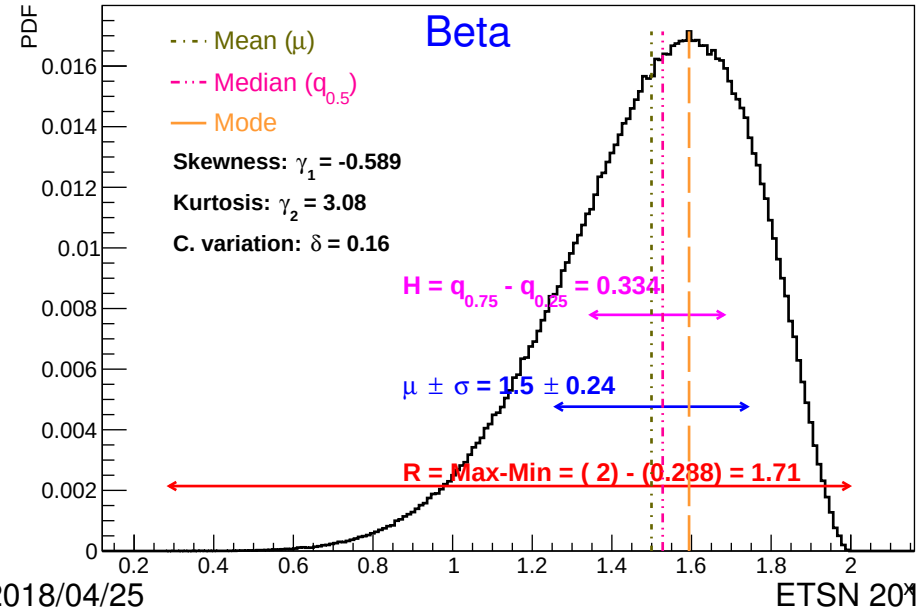
  - Normalised γ_2: sometimes -3.0 is added to it as γ_2=3.0 for $\mathcal{N}(\mu, \sigma)$

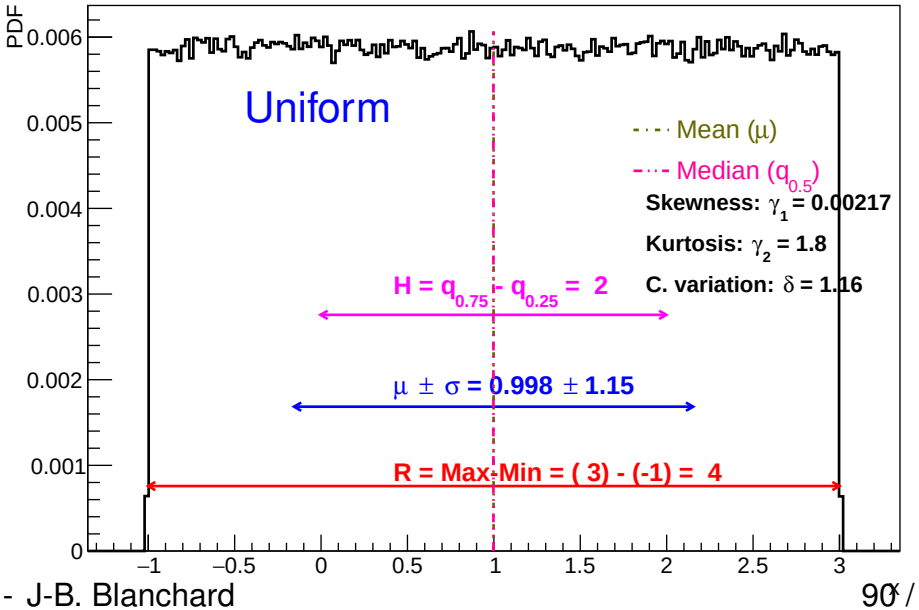
Uni-variate case: illustration of some parameters











Distribution principle recap

For every random variable $X : \Omega \to \mathbb{R}$

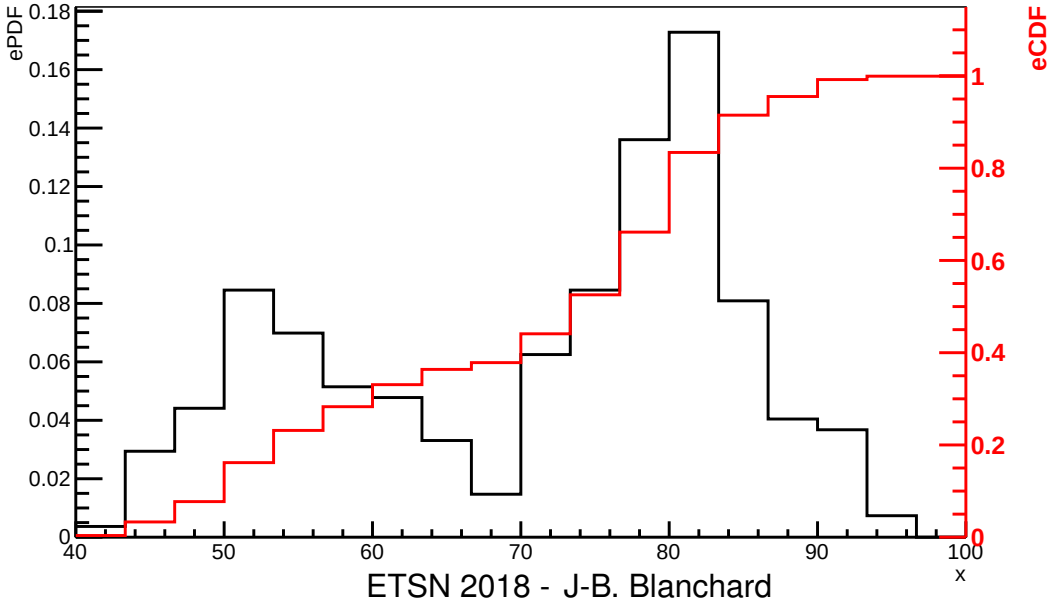
- **PDF** (Probability Density Function): if the random variable X has a density f_X, where f_X is a non-negative Lebesgue-integrable function, then

$$P\{a \leq X \leq b\} = \int_a^b f_X(s)ds$$

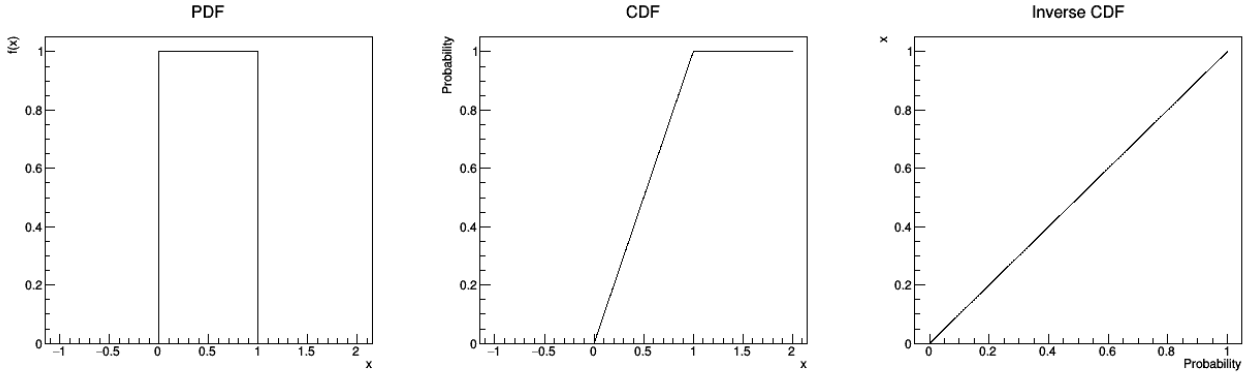
- **CDF** (Cumulative Distribution Function): the function $F_X : \mathbb{R} \to [0, 1]$, given by

$$F_X(a) = \int_{-\infty}^{a} f_X(s)ds, \ a \in \mathbb{R}$$

  - One might find **CCDF** for Complementary CDF, simply defined as $CCDF(a) = 1 - CDF(a)$



Example of distribution for the main laws I

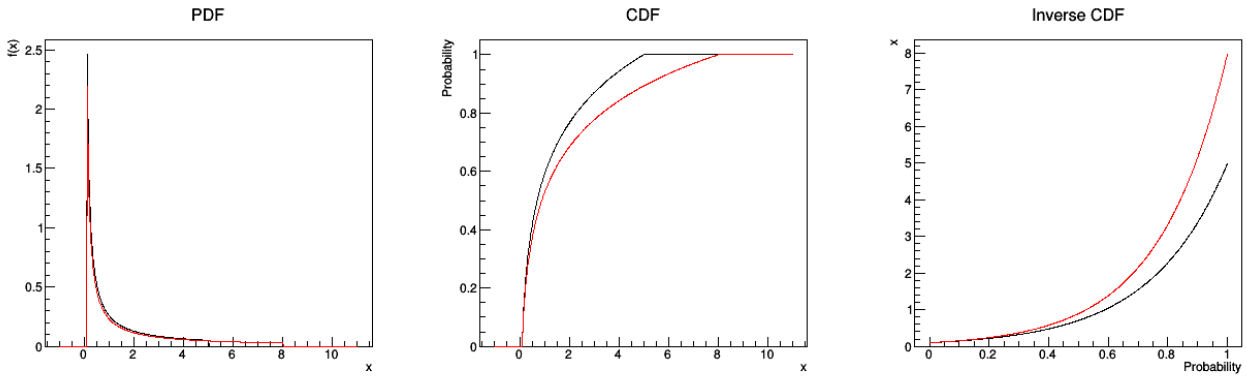


TUniform

x_{min}=0.0; x_{max}=1.0;

$$f(x) = \frac{1}{(x_{max}-x_{min})} \text{ for } x \in [x_{min},x_{max}]$$

2018-04-05 - Uranie v2016.2/10



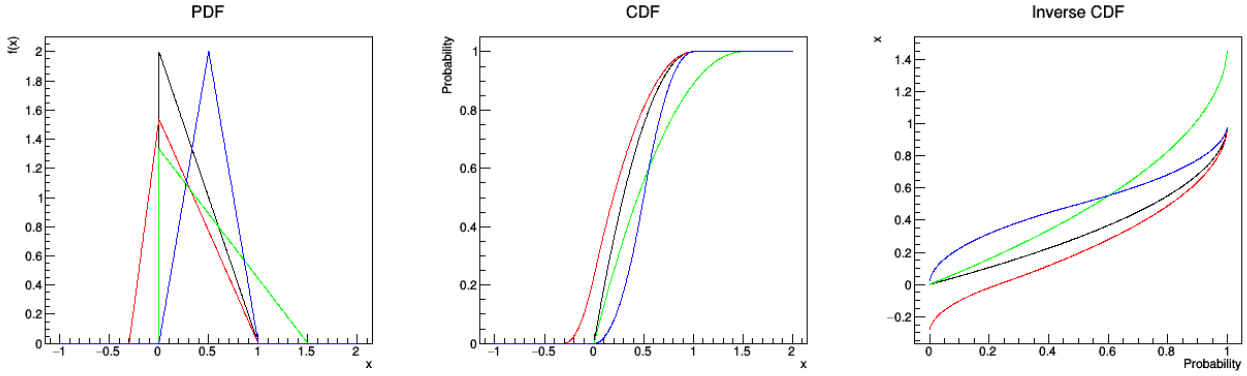
TLogUniform

x_{min}=0.1; x_{max}=5.0;
x_{min}=0.1; x_{max}=8.0;

$$f(x) = \frac{1}{(x \times \ln(x_{max}/x_{min}))} \text{ for } x \in [x_{min},x_{max}]$$

2018-04-05 - Uranie v2016.2/10

Example of distribution for the main laws II

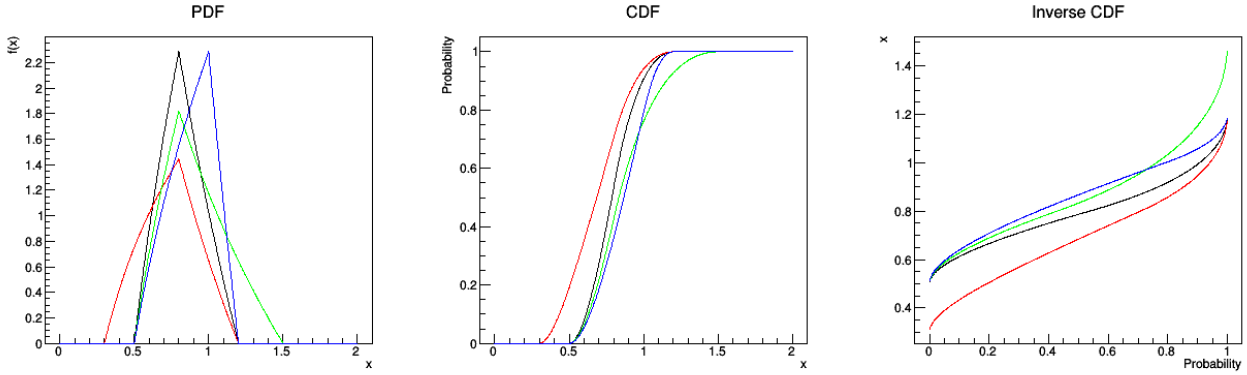


TTriangular

$$f(x) = \begin{cases} \dfrac{2 \times (x - x_{min})}{(x_{max} - x_{min}) \times (x_{mode} - x_{min})} & \text{for } x \in [x_{min}, x_{mode}] \\ \dfrac{2 \times (x_{max} - x)}{(x_{max} - x_{min}) \times (x_{max} - x_{mode})} & \text{for } x \in [x_{mode}, x_{max}] \end{cases}$$

x_{min}=0.0; x_{max}=1.0; x_{mod}=0.0;
x_{min}=-0.3; x_{max}=1.0; x_{mod}=0.0;
x_{min}=0.0; x_{max}=1.5; x_{mod}=0.0;
x_{min}=0.0; x_{max}=1.0; x_{mod}=0.5;

2018-04-05 - Uranie v2016.2/10



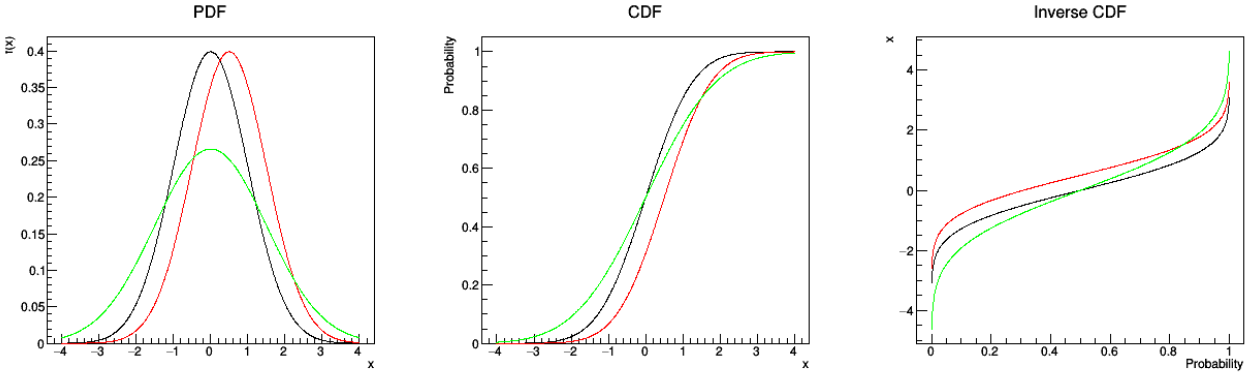
TLogTriangular

$$f(x) = \begin{cases} \dfrac{2 \times \ln(x/x_{min})}{x \times \ln(x_{max}/x_{min}) \times \ln(x_{mode}/x_{min})} & \text{for } x \in [x_{min}, x_{mode}] \\ \dfrac{2 \times \ln(x_{max}/x)}{x \times \ln(x_{max}/x_{min}) \times \ln(x_{max}/x_{mode})} & \text{for } x \in [x_{mode}, x_{max}] \end{cases}$$

x_{low}=0.5; x_{ip}=1.2; x_{mod}=0.8;
x_{low}=0.3; x_{ip}=1.2; x_{mod}=0.8;
x_{low}=0.5; x_{ip}=1.5; x_{mod}=0.8;
x_{low}=0.5; x_{ip}=1.2; x_{mod}=1.0;

2018-04-05 - Uranie v2016.2/10

Example of distribution for the main laws III

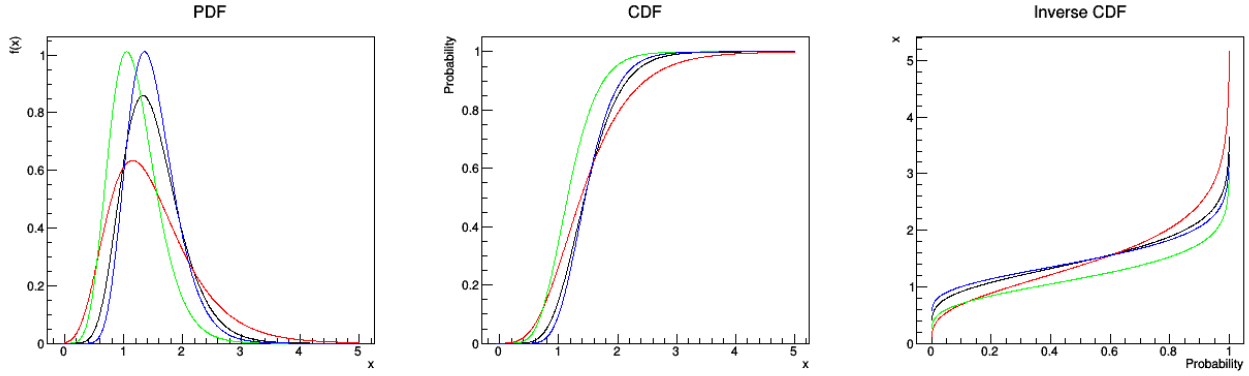


TNormal

μ=0.0; σ=1.0;
μ=0.5; σ=1.0;
μ=0; σ=1.5;

$$f(x) = e^{\frac{-(x-\mu)^2}{2\sigma^2}} \times \frac{1}{\sqrt{2\pi\sigma^2}}$$

2018-04-05 - Uranie v2016.2/10



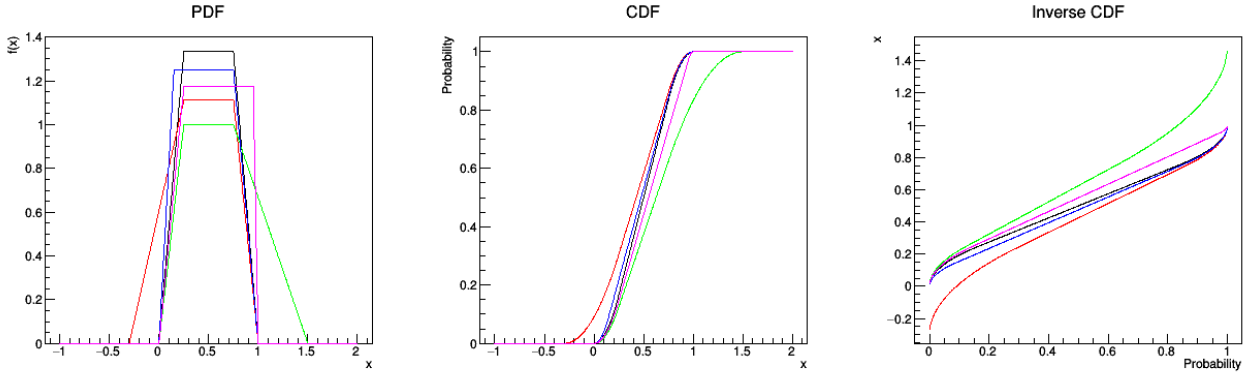
TLogNormal

μ=1.5; σ=1.5; x^{min}=-0.5;
μ=1.5; σ=1.8; x^{min}=-0.5;
μ=1.2; σ=1.5; x^{min}=-0.5;
μ=1.5; σ=1.5; x^{min}=-0.2;

$$f(x) = \frac{1}{((x-x_{min})\sigma\sqrt{2\pi})} \times e^{\frac{-(\ln(x-x_{min})-\mu)^2}{2\sigma^2}} \quad \text{for } x > x_{min}$$

2018-04-05 - Uranie v2016.2/10

Example of distribution for the main laws IV



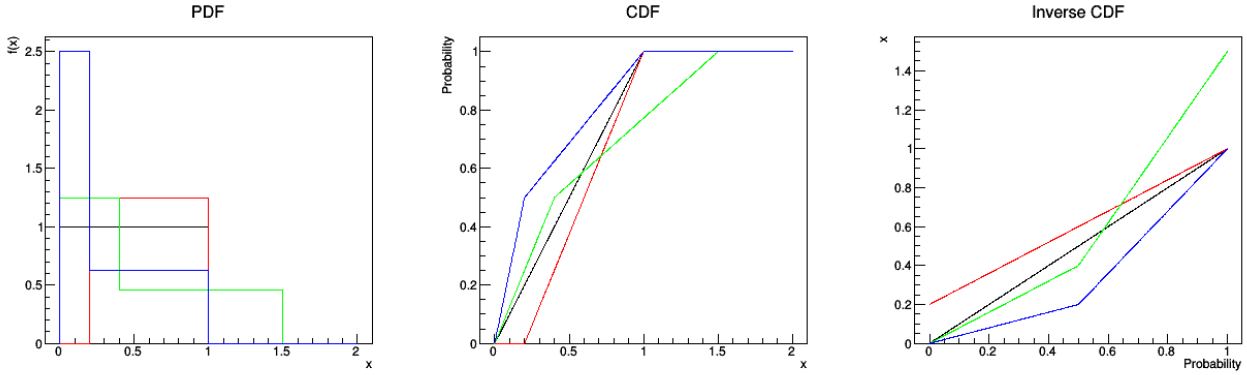
TTrapezium

$$f(x)=\frac{2}{(x_{up}-x_{low})+(x_{max}-x_{min})} \times Y,\ Y=1\ x\in[x_{low},x_{up}]$$

$$Y=\frac{(x-x_{min})}{(x_{low}-x_{min})}\ (x\in[x_{min},x_{low}])\ \text{and}\ \frac{(x_{max}-x)}{(x_{max}-x_{up})}\ (x\in[x_{up},x_{max}])$$

x_{min}=0.0; x_{max}=1.0; x_{low}=0.25; x_{up}=0.75;
x_{min}=-0.3; x_{max}=1.0; x_{low}=0.25; x_{up}=0.75;
x_{min}=0.0; x_{max}=1.5; x_{low}=0.25; x_{up}=0.75;
x_{min}=0.0; x_{max}=1.0; x_{low}=0.15; x_{up}=0.75;
x_{min}=0.0; x_{max}=1.0; x_{low}=0.25; x_{up}=0.95;

2018-04-05 - Uranie v2016.2/10



TUniformByParts

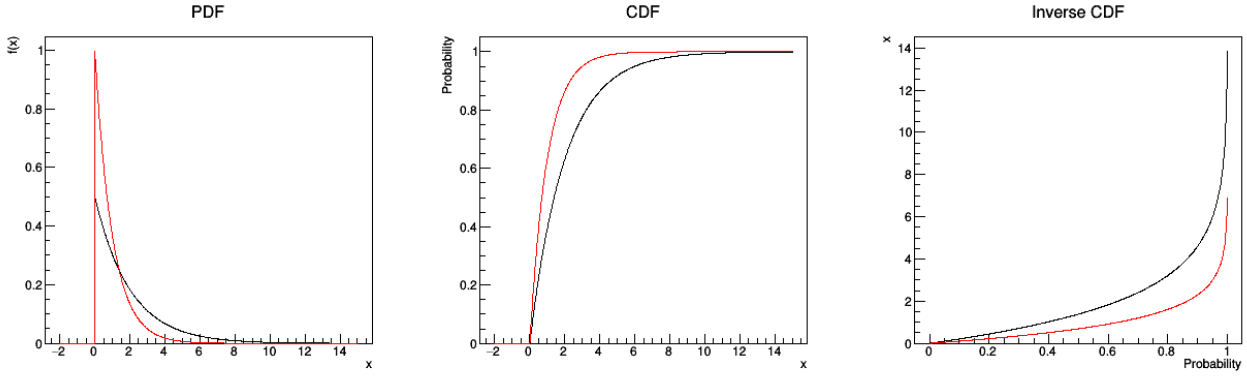
$$f(x)=\frac{0.5}{(x_{median}-x_{min})}\ \text{for}\ x\in[x_{min},x_{median}]$$

$$\frac{0.5}{(x_{max}-x_{median})}\ \text{for}\ x\in[x_{median},x_{max}]$$

x_{min}=0.0; x_{max}=1.0; x_{median}=0.5;
x_{min}=0.2; x_{max}=1.0; x_{median}=0.6;
x_{min}=0.0; x_{max}=1.5; x_{median}=0.4;
x_{min}=0.0; x_{max}=1.0; x_{median}=0.2;

2018-04-05 - Uranie v2016.2/10

Example of distribution for the main laws V

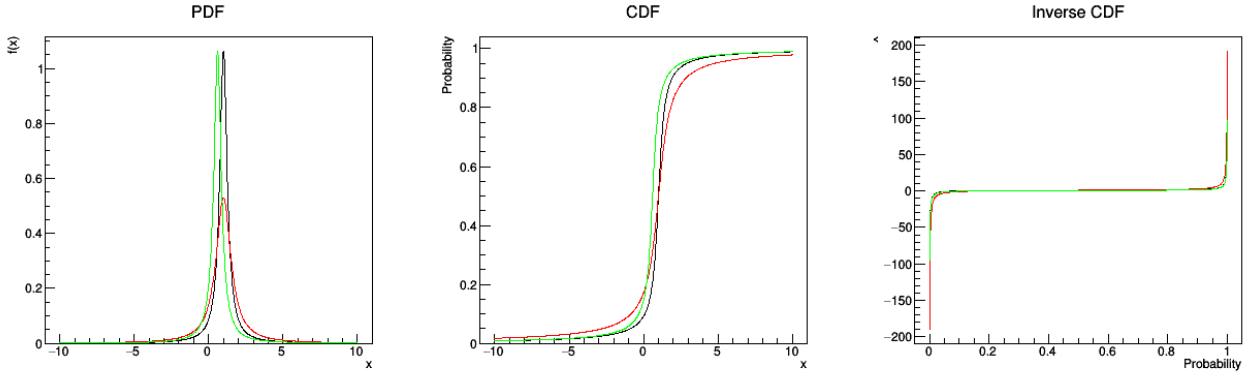


TExponential

λ=0.5;
λ=1.0;

$$f(x) = \lambda \times e^{-\lambda \times (x - x_{min})}, \text{ for } x \geq x_{min}$$

2018-04-05 - Uranie v2016.2/10



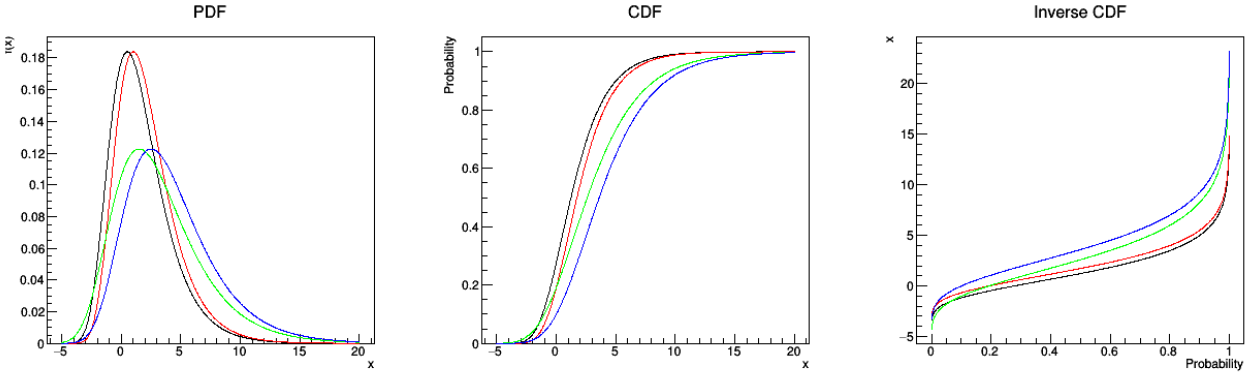
TCauchy

γ=0.3; x_0=1.0;
γ=0.6; x_0=1.0;
γ=0.3; x_0=0.6;

$$f(x) = \frac{\gamma}{\pi \times (\gamma^2 + (x - x_0)^2)}$$

2018-04-05 - Uranie v2016.2/10

Example of distribution for the main laws VI

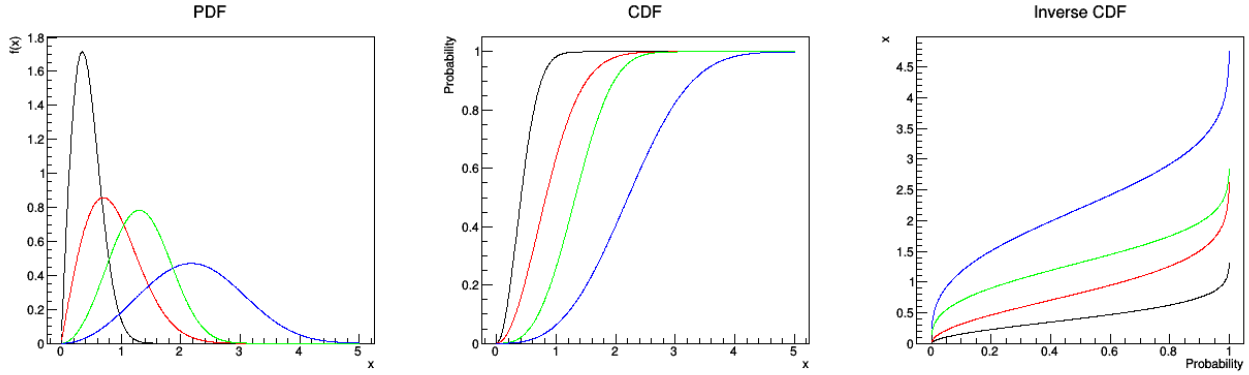


TGumbelMax

μ=0.5; β=2.0;
μ=1.0; β=2.0;
μ=1.5; β=3.0;
μ=2.5; β=3.0;

$$f(x) = z \times \frac{e^{-z}}{\beta}, \text{ where } z = e^{\frac{-(x-\mu)}{\beta}}$$

2018-04-05 - Uranie v2016.2/10



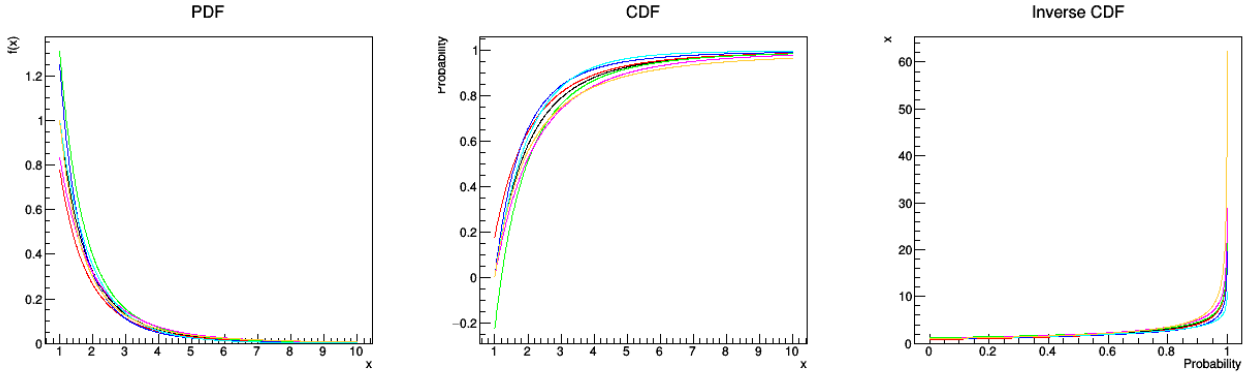
TWeibull

λ=0.5; k=2.0; x_{min}=-0.01;
λ=1.0; k=2.0; x_{min}=-0.01;
λ=1.5; k=3.0; x_{min}=-0.01;
λ=2.5; k=3.0; x_{min}=-0.01;

$$f(x) = \frac{k}{\lambda} \times \left(\frac{x-x_{min}}{\lambda}\right)^{k-1} \times e^{-\left(\frac{x-x_{min}}{\lambda}\right)^{k}} \text{ for } x > x_{min}$$

2018-04-05 - Uranie v2016.2/10

Example of distribution for the main laws VII

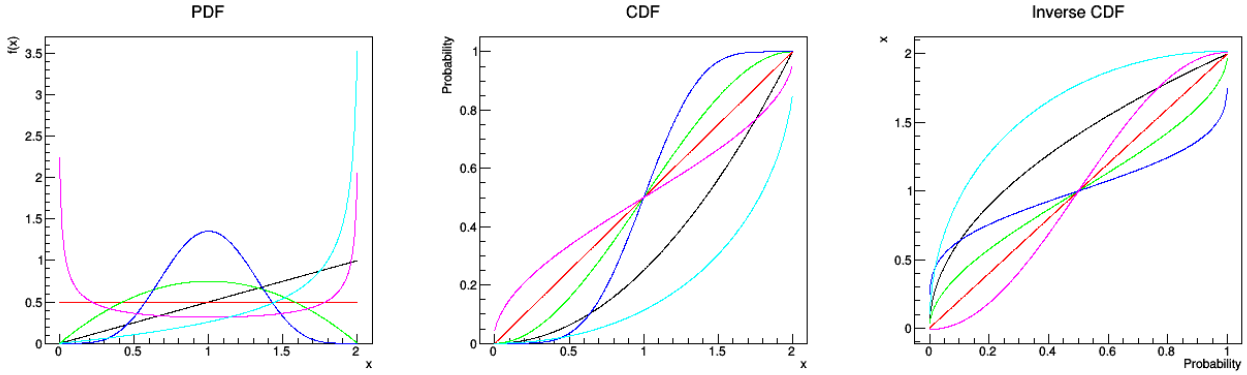


TGenPareto

$$f(x) = \frac{1}{\sigma} \times \left(1 + \xi\left(\frac{x-\mu}{\sigma}\right)\right)^{-(1/\xi+1)}$$

μ=1.0; σ=1.0; ξ=0.3;
μ=0.8; σ=1.0; ξ=0.3;
μ=1.2; σ=1.0; ξ=0.3;
μ=1.0; σ=0.8; ξ=0.3;
μ=1.0; σ=1.2; ξ=0.3;
μ=1.0; σ=1.0; ξ=0.1;
μ=1.0; σ=1.0; ξ=0.5;

2018-04-05 - Uranie v2016.2/10



TBeta

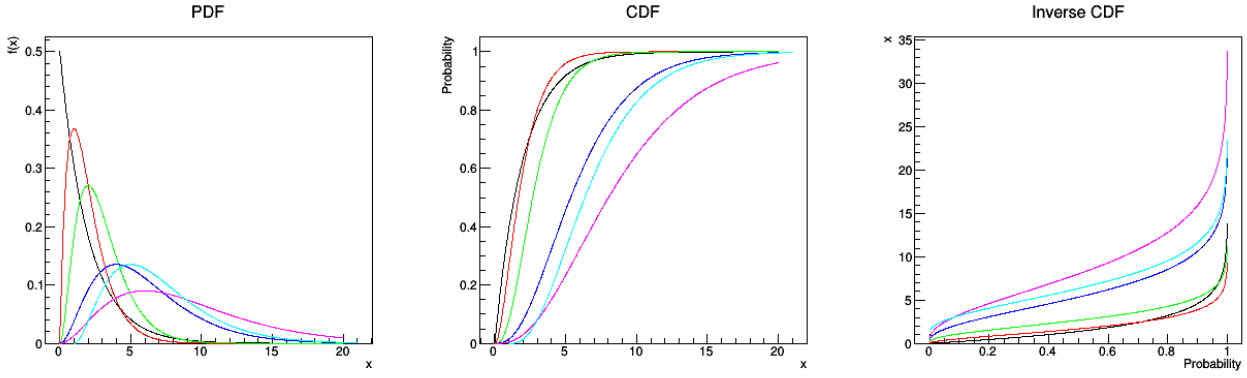
$$f(x) = \frac{Y^{\alpha-1} \times (1-Y)^{\beta-1}}{B(\alpha,\beta)} \text{ for } x \in [x_{min}, x_{max}]$$

where $Y = \frac{(x-x_{min})}{(x_{max}-x_{min})}$ and $B(\alpha,\beta)$ is beta function

α=2.0; β=1.0; x_{min}=0.0; x_{max}=2.0;
α=1.0; β=1.0; x_{min}=0.0; x_{max}=2.0;
α=2.0; β=2.0; x_{min}=0.0; x_{max}=2.0;
α=6.0; β=6.0; x_{min}=0.0; x_{max}=2.0;
α=0.5; β=0.5; x_{min}=-0.01; x_{max}=2.01;
α=2.0; β=0.5; x_{min}=0.0; x_{max}=2.02;

2018-04-05 - Uranie v2016.2/10

Example of distribution for the main laws VIII

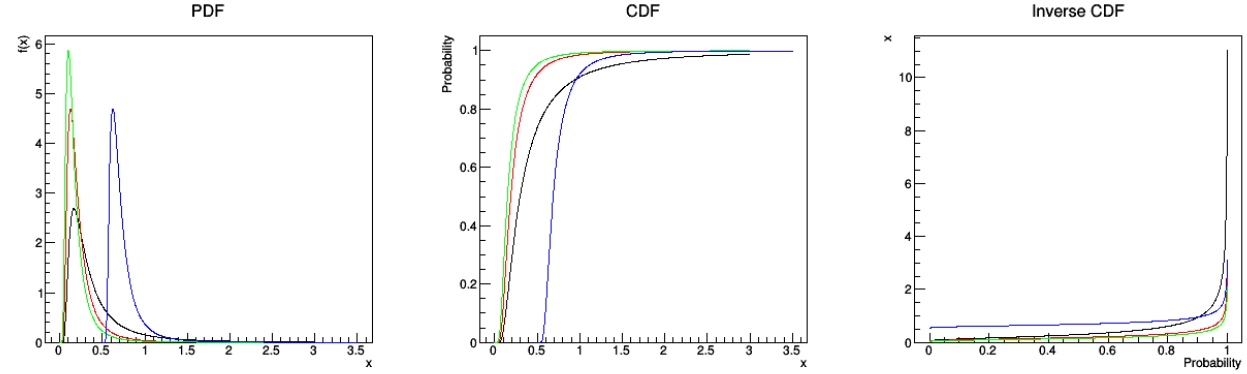


TGamma

$$f(x) = \frac{(x-\xi)^{\alpha-1} e^{-(x-\xi)/\beta}}{\Gamma(\alpha)\beta^{\alpha}}$$

α=1.0; β=2.0; ξ=0.0;
α=2.0; β=1.0; ξ=0.0;
α=3.0; β=1.0; ξ=0.0;
α=3.0; β=2.0; ξ=0.0;
α=3.0; β=3.0; ξ=0.0;
α=3.0; β=2.0; ξ=1.0;

2018-04-05 - Uranie v2016.2/10



TInvGamma

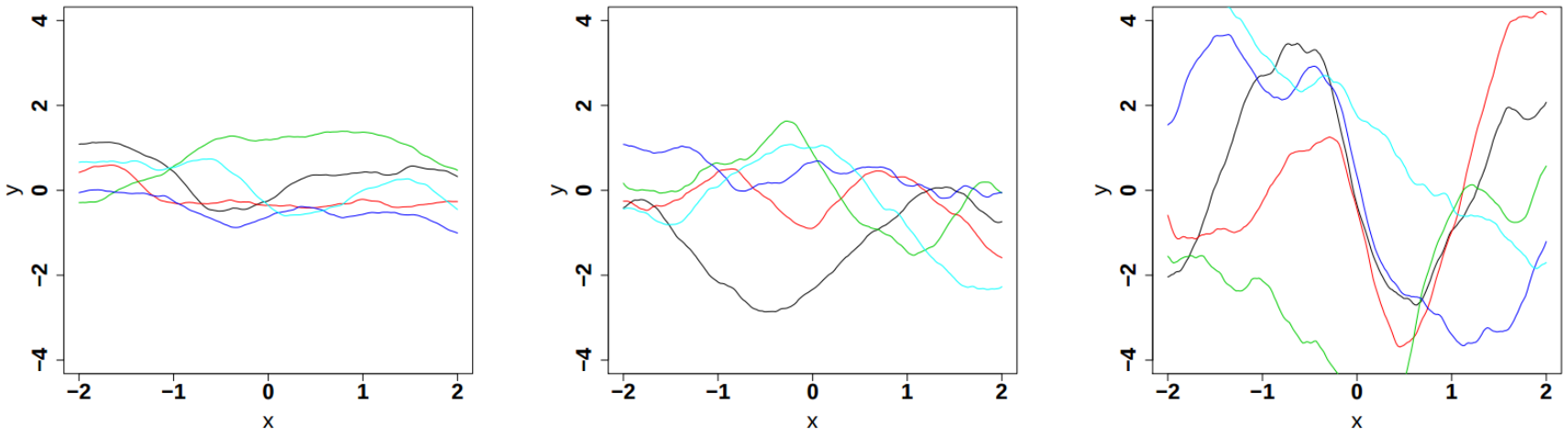
$$f(x) = \frac{\beta^{\alpha}(x-\xi)^{-\alpha-1} e^{-\beta/(x-\xi)}}{\Gamma(\alpha)}$$

α=2.0; β=0.5; ξ=0.0;
α=3.0; β=0.5; ξ=0.0;
α=3.0; β=0.4; ξ=0.0;
α=3.0; β=0.5; ξ=0.5;

2018-04-05 - Uranie v2016.2/10

Parameters for matern function I

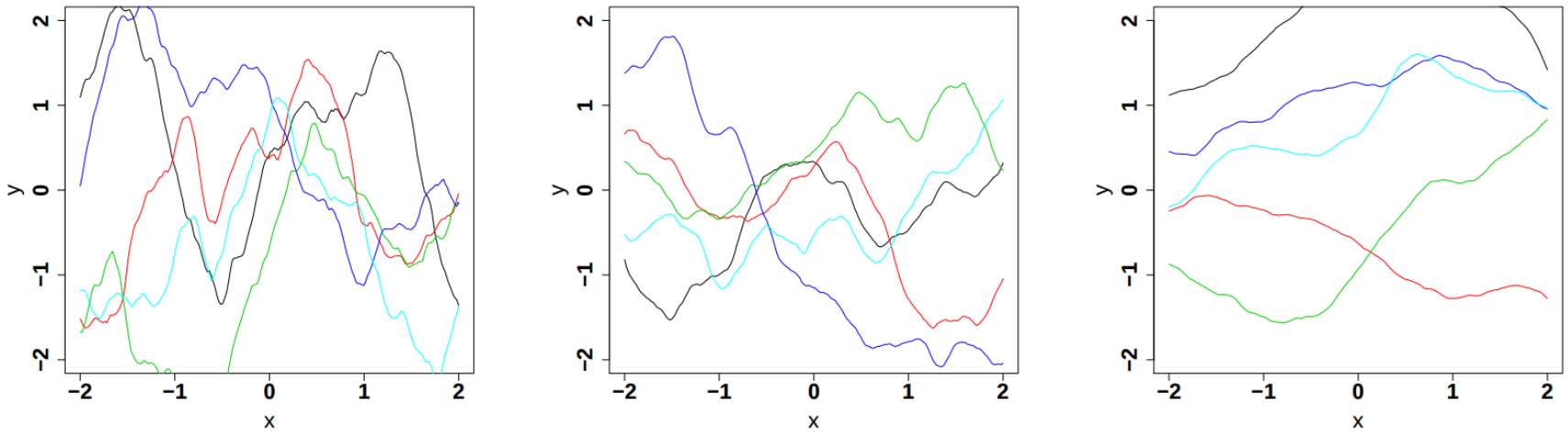
$$c(\delta x) = \frac{\sigma^2}{\Gamma(\nu)2^{\nu-1}} \left(2\sqrt{\nu}\frac{\delta x}{l}\right)^{\nu} K_\nu \left(2\sqrt{\nu}\frac{\delta x}{l}\right).$$



Variation of the variance σ

Parameters for matern function II

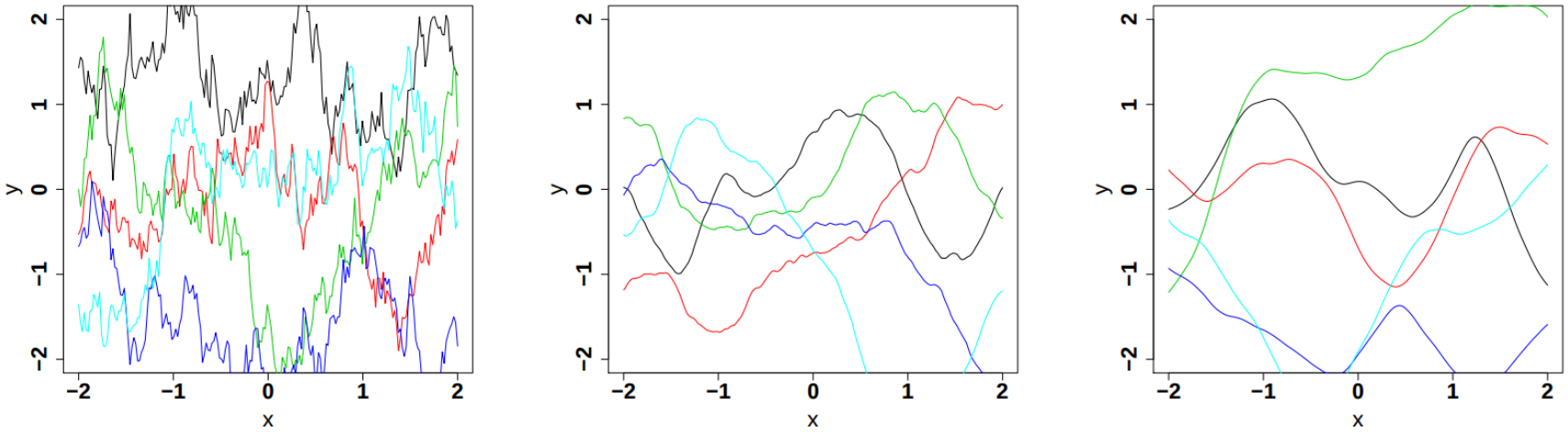
$$c(\delta x) = \frac{\sigma^2}{\Gamma(\nu)2^{\nu-1}} \left(2\sqrt{\nu}\frac{\delta x}{l}\right)^{\nu} K_{\nu}\left(2\sqrt{\nu}\frac{\delta x}{l}\right).$$



Variation of the correlation length l

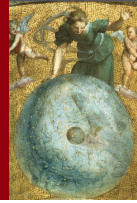
Parameters for matern function III

$$c(\delta x) = \frac{\sigma^2}{\Gamma(\nu)2^{\nu-1}} \left(2\sqrt{\nu}\frac{\delta x}{l}\right)^{\nu} K_\nu \left(2\sqrt{\nu}\frac{\delta x}{l}\right).$$



Variation of the smoothness ν

Chaos polynomial expansion example I



The interpretation of the polynomial coefficients as Sobol's coefficients is strongly relying on the hypothesis that the probability laws have been properly defined and that their decomposition is done on their natural polynomial basis :

	Legendre	Hermite	Laguerre	Jacobi
Uniform	X			
Normal		X		
Exponential			X	
Beta				X

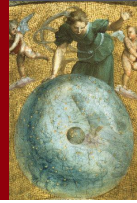
From there, with $p = 2$ one obtains 6 coefficients: $\beta_{0,0}$, $\beta_{1,0}$, $\beta_{0,1}$, $\beta_{2,0}$, $\beta_{0,2}$, $\beta_{1,1}$. These coefficients are characterising the surrogate model and can be used, **when the inputs are independent**, to estimate the corresponding Sobol's coefficients

$$S_1^U = \frac{\beta_{1,0}^2 + \beta_{2,0}^2}{\mathrm{Var}(Y)} \text{ and } S_1^N = \frac{\beta_{0,1}^2 + \beta_{0,2}^2}{\mathrm{Var}(Y)},$$

$$S_T^U = \frac{\beta_{1,0}^2 + \beta_{2,0}^2 + \beta_{1,1}^2}{\mathrm{Var}(Y)} \text{ and } S_T^N = \frac{\beta_{0,1}^2 + \beta_{0,2}^2 + \beta_{1,1}^2}{\mathrm{Var}(Y)}.$$

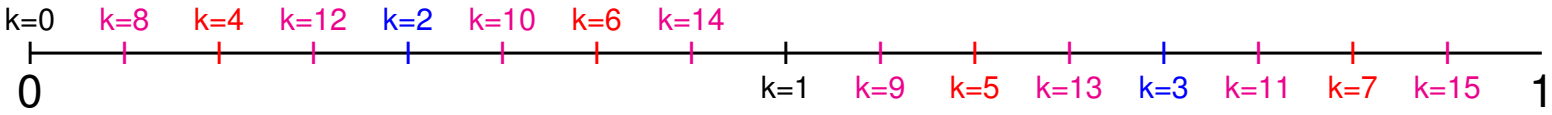
$$\mathrm{Var}(Y) = \sum_{|\alpha|_1 \leq 2} \beta_\alpha^2$$

Logic behind the Van der Corput implementation



k	k in base 2	$\phi(k)$ in base 2	x_k
0	0	0.0	0
1	1	0.1	1/2
2	10	0.01	1/4
3	11	0.11	3/4
4	100	0.001	1/8
5	101	0.101	5/8
k	$a_{m-1}\dots a_1 a_0$	$0.a_0 a_1 \dots a_{m-1}$	$\sum_{l=0}^{m-1} a_l 2^{-(l+1)}$

Representing them on a normalised line for 1, 2, 3, 4 bites in 2-base

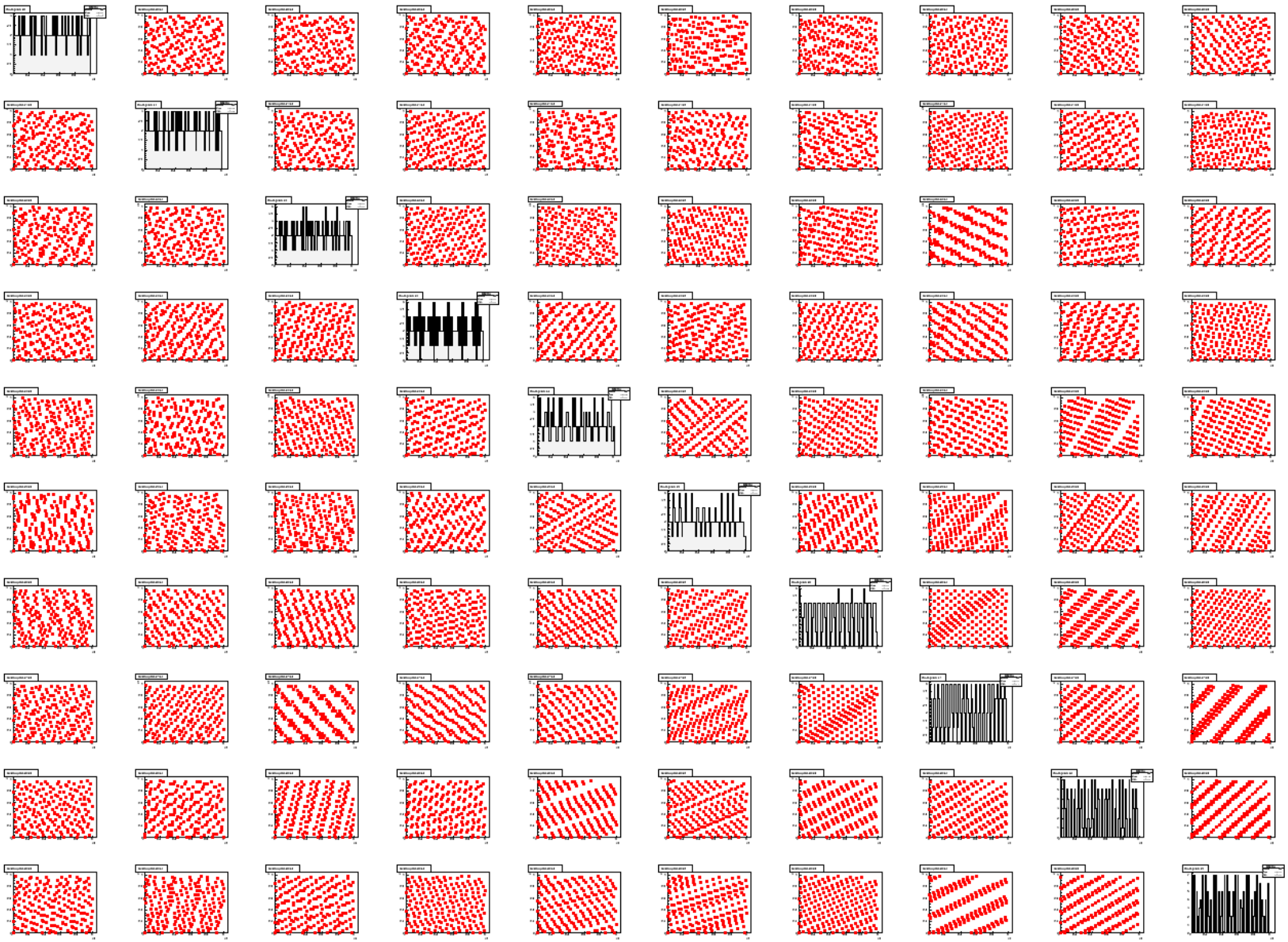


With more than one dimension, the idea is the same using other prime number as basis (2, 3, 5, . . .)

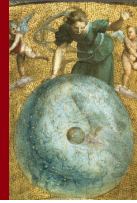
DE LA RECHERCHE À L'INDUSTRIE

cea

QMC: the dimension problem



Toy model for interaction I



Considering the function $Y = g(\mathbf{X}) = X_1 X_2$, where $X_a = \mathcal{N}(\mu_a, \sigma_a)$ are independent random variable, for $a = 1, 2$

$$\mathbb{E}(Y) = \int_{-\infty}^{-\infty} \int_{-\infty}^{-\infty} x_1 x_2 f_{X_1,X_2}(x_1, x_2) dx_1 dx_2$$

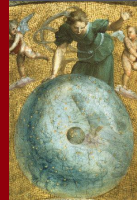
The random variables are independent so

$$f_{X_1,X_2}(x_1, x_2) = f_{X_1}(x_1) f_{X_2}(x_2)$$

One can then develop the expectation formula

$$\begin{aligned}
\mathbb{E}(Y) &= \int_{-\infty}^{-\infty} \int_{-\infty}^{-\infty} x_1 x_2 f_{X_1,X_2}(x_1, x_2) dx_1 dx_2 \\
&= \left(\int_{-\infty}^{-\infty} x_1 f_{X_1}(x_1) dx_1 \right) \left(\int_{-\infty}^{-\infty} x_2 f_{X_2}(x_2) dx_2 \right) \\
&= \mathbb{E}(X_1)\mathbb{E}(X_2) \\
&= \mu_1 \mu_2
\end{aligned}$$

Toy model for interaction II



Considering the variance one can start from this:

$$V(Y) = \mathbb{E}(Y^2) - \mathbb{E}(Y)^2$$

Developing the expectation

$$\begin{aligned}
\mathbb{E}(Y^2) &= \int_{-\infty}^{-\infty}\int_{-\infty}^{-\infty}(x_1x_2)^2 f_{X_1,X_2}(x_1,x_2)dx_1dx_2 \\
&= \left(\int_{-\infty}^{-\infty} x_1^2 f_{X_1}(x_1)dx_1\right)\left(\int_{-\infty}^{-\infty} x_2^2 f_{X_2}(x_2)dx_2\right) \\
&= \mathbb{E}(X_1^2)\mathbb{E}(X_2^2)
\end{aligned}$$

Also for $a = 1, 2$

$$V(X_a) = \mathbb{E}(X_a^2) - \mathbb{E}(X_a)^2$$

$$\mathbb{E}(X_a^2) = V(X_a) + \mathbb{E}(X_a)^2$$

$$\begin{aligned}
\mathbb{E}(Y^2) &= (V(X_1) + \mathbb{E}(X_1)^2)(V(X_2) + \mathbb{E}(X_2)^2) \\
&= (\sigma_1^2 + \mu_1^2)(\sigma_2^2 + \mu_2^2)
\end{aligned}$$

Putting these results together

$$\begin{aligned}
V(Y) &= (\sigma_1^2 + \mu_1^2)(\sigma_2^2 + \mu_2^2) - (\mu_1\mu_2)^2 \\
&= \mu_1^2\sigma_2^2 + \mu_2^2\sigma_1^2 + \sigma_1^2\sigma_2^2
\end{aligned}$$

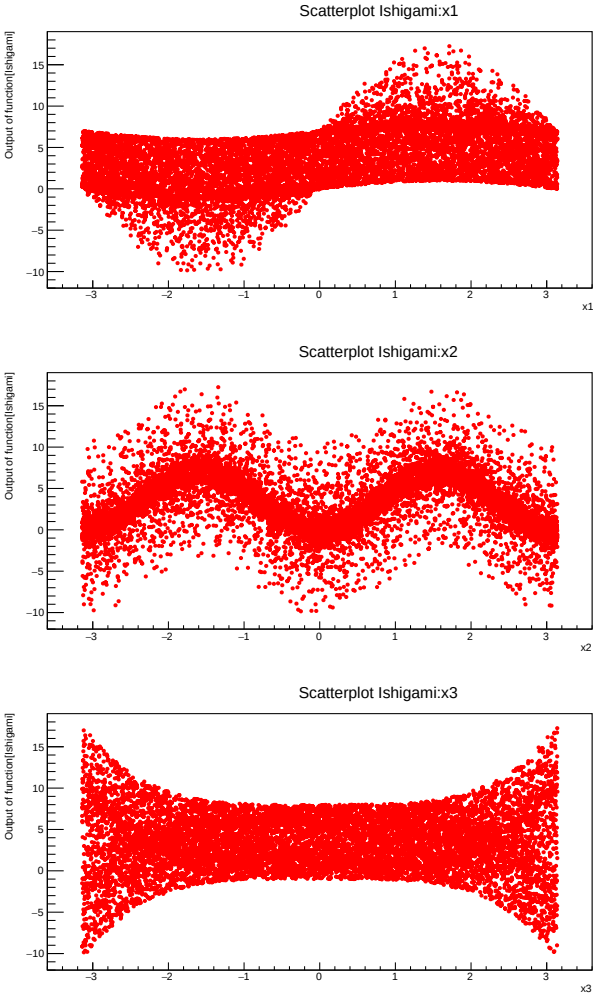
The variance $V(Y)$ cannot be explained neither by X_1 alone, or by X_2 alone, their interaction has to be taken into account

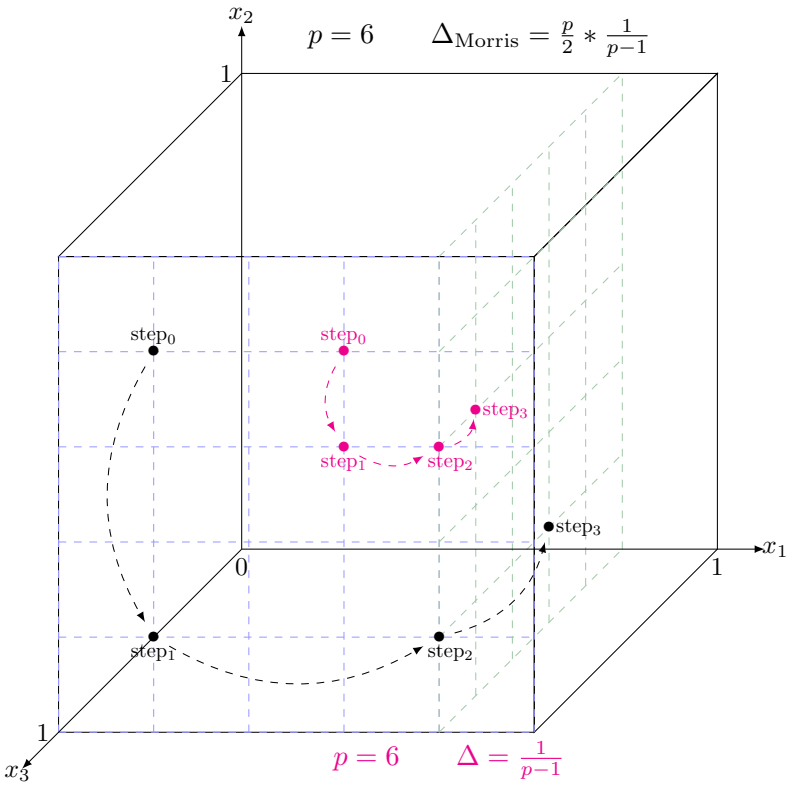
Warning! Interaction = impact of the coupling of factors on a considered output. Independence = evolution of the values taken by two factors.

Caution about the Morris method

The default Δ is chosen after optimisation to be $\Delta_{\text{Morris}} = \frac{p}{2}\frac{1}{p-1}$

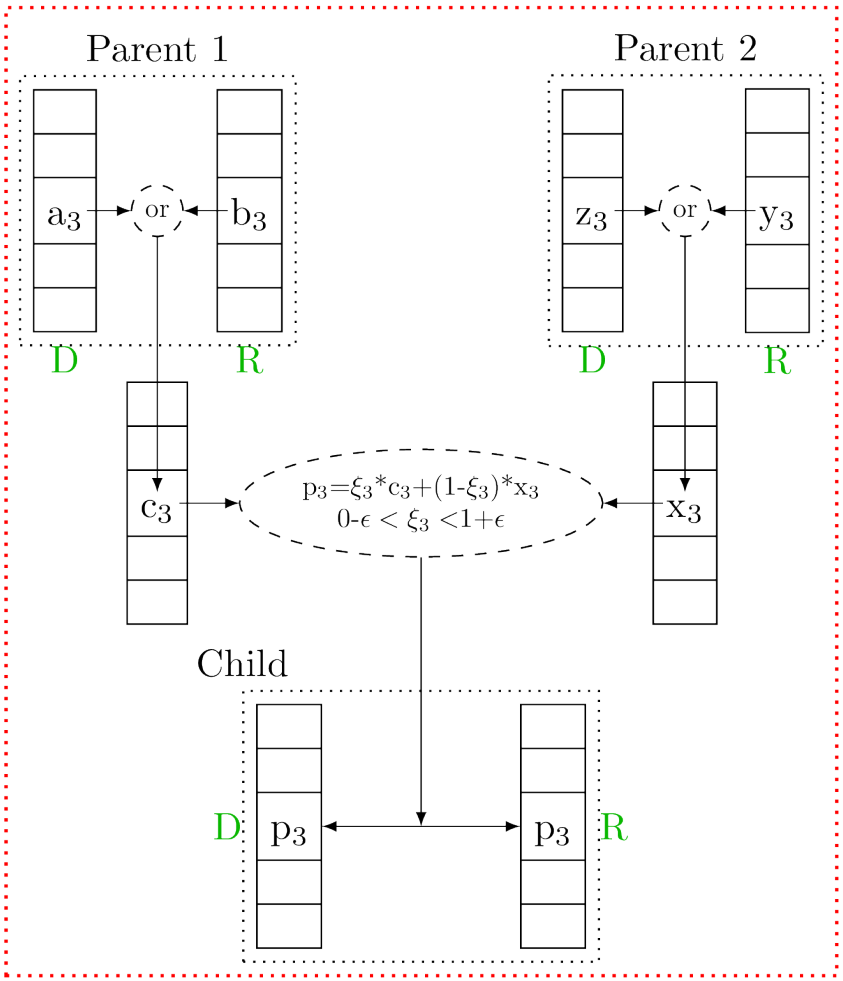
$\Rightarrow$ Works fine as long as the output is not periodic w.r.t one input.



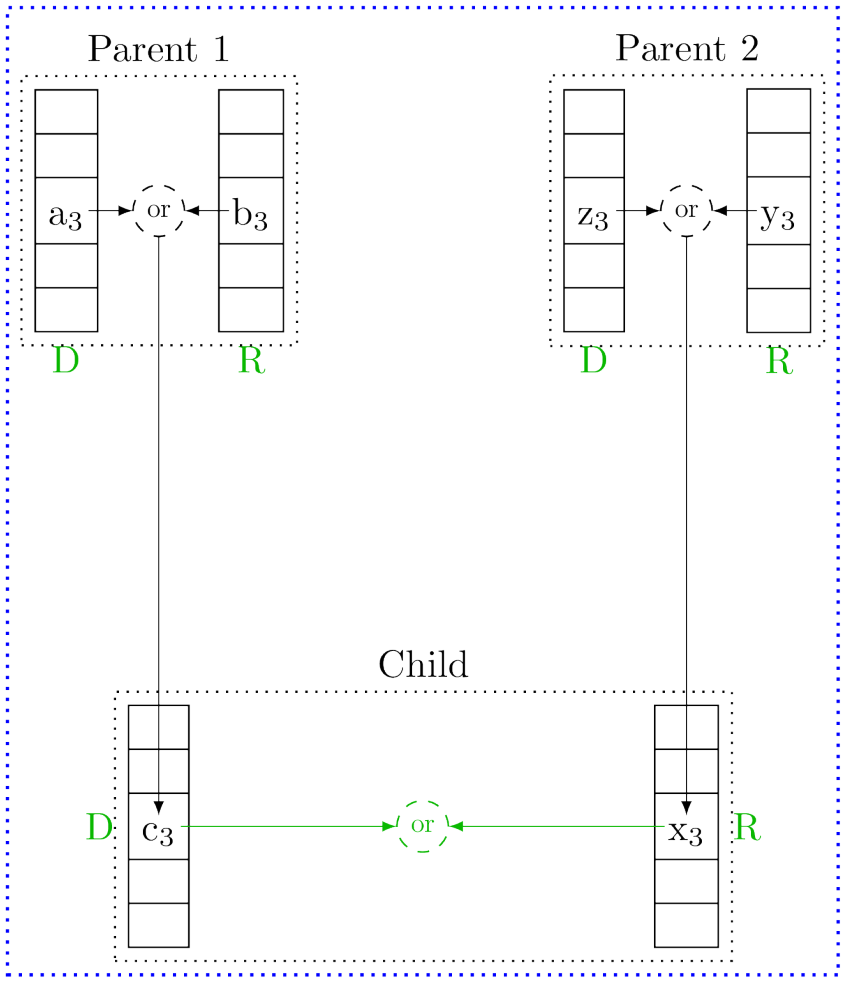


Vary Δ as fifth argument of TMorris constructor to check the impact

Genetic evolution in a drawing



Homozygote (proba Ho)



Heterozygote (proba (1-Ho))

+ possible mutation
(proba M)

The expected improvement definition

$$E[I(\mathbf{x})] = (f_{min} - \hat{y}(\mathbf{x}))\Phi\left(\frac{f_{min} - \hat{y}(\mathbf{x})}{\hat{\sigma}(\mathbf{x})}\right) + \hat{\sigma}(\mathbf{x})\phi\left(\frac{f_{min} - \hat{y}(\mathbf{x})}{\hat{\sigma}(\mathbf{x})}\right)$$

$\phi(.)$ and $\Phi(.)$ are respectively the standard normal density and its cumulative distribution.