

# OpenTURNS cheat sheet



This *OpenTURNS* v1.17 cheat sheet provides a quick overview of all the programming interface. For full documentation, please read the doc. A beginner may be interested in the Quick start guides.

This cheat sheet follows the steps of the ABC method.

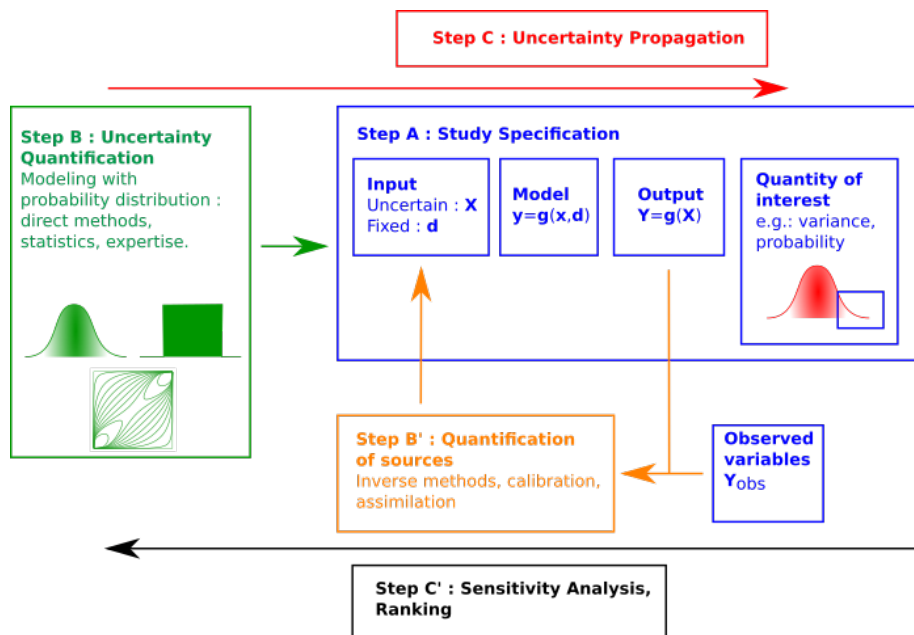


Figure 1: The ABC method

## Step A : define the study

Purpose	Class / Method
Import <i>OpenTURNS</i>	<code>import openturns as ot</code>
Vector	<code>ot.Point(dimension)</code>
Sample	<code>ot.Sample(size, dimension)</code>
Symbolic function	<code>ot.SymbolicFunction(["x0", "x1"], ["1 + x0 + x1"])</code>
Python function	<code>ot.PythonFunction(number_of_inputs, number_of_outputs, g)</code>
Manage history and cache	<code>ot.MemoizeFunction(myfunction)</code>

Purpose	Class / Method
Normal	<code>ot.Normal(mu, sigma)</code>
Uniform	<code>ot.Uniform(a, b)</code>
Multivariate distribution with independent copula	<code>ot.ComposedDistribution((X0, X1, X2))</code>
Input random vector	<code>ot.RandomVector(inputDistribution)</code>
Output random vector	<code>ot.CompositeRandomVector(g, inputRandomVector)</code>
Generate observations	<code>randomVector.getSample(sample_size)</code>
Set the seed	<code>ot.RandomGenerator.SetSeed(1976)</code>
Get sample size	<code>sample.getSize()</code>
Get sample dimension	<code>sample.getDimension()</code>
Compute sample mean	<code>outputSample.computeMean()</code>
Compute sample standard deviation	<code>outputSample.computeStandardDeviation()</code>

## Step B : quantification of the sources of uncertainties

Purpose	Class / Method
Fit a Normal	<code>ot.NormalFactory().build(sample)</code>
Fit a Beta	<code>ot.BetaFactory().build(sample)</code>
Fit an histogram	<code>ot.HistogramFactory().build(sample)</code>
Fit a kernel density estimator	<code>ot.KernelSmoothing().build(sample)</code>
Draw QQ-plot	<code>ot.VisualTest.DrawQQplot(sample, distribution)</code>
Kolmogorov-Smirnov test (known parameters)	<code>ot.FittingTest.Kolmogorov(sample, distribution)</code>
Kolmogorov-Smirnov test (unknown parameters)	<code>ot.FittingTest.Lilliefors(sample, factory)</code>
BIC criteria	<code>ot.FittingTest.BIC(sample, distribution)</code>

## Step C : push forward the uncertainties

Purpose	Class / Method
Taylor expansion	<code>ot.TaylorExpansionMoments(output_random_vector)</code>
Estimate mean	<code>ot.ExpectationSimulationAlgorithm(output_random_vector)</code>
Estimate $P(Y > s)$	<code>sample.computeEmpiricalCDF(s, True)</code>
Create the event ( $Y > s$ )	<code>ot.ThresholdEvent(output_random_vector, ot.Greater(), s)</code>
Create a Monte-Carlo experiment	<code>ot.MonteCarloExperiment()</code>

Purpose	Class / Method
Estimate a probability	<code>ot.ProbabilitySimulationAlgorithm(myEvent, experiment)</code>

### Step C' : sensitivity analysis

Purpose	Class / Method
Perform linear regression	<code>ot.LinearLeastSquares(input_sample, output_sample)</code>
Standardized regression coefficients	<code>ot.CorrelationAnalysis_SignedSRC(input_sample, output_sample)</code>
Draw indices	<code>ot.SobolIndicesAlgorithm.DrawCorrelationCoefficients(src_index, input_names, "SRC coefficients")</code>
Estimate Sobol' indices given budget	<code>ot.SobolIndicesExperiment(distribution, sample_size)</code>
Estimate Sobol' indices	<code>ot.SaltelliSensitivityAlgorithm()</code>

### Step B' : calibration

Purpose	Class / Method
Create the parametric model	<code>ot.ParametricFunction(g, calibrated_indices, theta_prior)</code>
Linear least squares	<code>ot.LinearLeastSquaresCalibration(parametric_g, input_sample, output_sample, theta_prior, "SVD")</code>
Non linear least squares	<code>ot.NonLinearLeastSquaresCalibration(parametric_g, input_sample, output_sample, theta_prior)</code>
Linear gaussian	<code>ot.GaussianLinearCalibration(parametric_g, input_sample, output_sample, theta_prior, theta_sigma, output_covariance)</code>
Non linear gaussian	<code>ot.GaussianNonLinearCalibration(parametric_g, input_sample, output_sample, theta_prior, theta_sigma, output_covariance)</code>

Purpose	Class / Method
Bayesian calibration	<code>ot.RandomWalkMetropolisHastings(prior, conditional, model, input_sample, output_sample, initialState, proposal)</code>

## Metamodel

Purpose	Class / Method
Squared exponential	<code>ot.SquaredExponential([1.0] * dimension, [1.0])</code>
Matern 5/2 covariance	<code>ot.MaternModel([1.0] * dimension, 2.5)</code>
Kriging	<code>ot.KrigingAlgorithm(input_sample, output_sample, covarianceModel, basis)</code>
Sample from kriging	<code>ot.KrigingRandomVector(result, input_sample)</code>
Conditioned gaussian process	<code>ot.ConditionedGaussianProcess(kriging_result, mesh)</code>
Multivariate basis	<code>ot.OrthogonalProductPolynomialFactory(distribution_collection)</code>
Polynomial chaos (given data)	<code>ot.FunctionalChaosAlgorithm(input_sample, output_sample)</code>
Polynomial chaos (given distribution)	<code>ot.FunctionalChaosAlgorithm(input_sample, output_sample, distribution, adaptive_strategy, projection_strategy)</code>
Sobol' indices from chaos	<code>ot.FunctionalChaosSobolIndices(functional_chaos_result)</code>
Sample from chaos	<code>ot.FunctionalChaosRandomVector(functional_chaos_result)</code>
Validation	<code>ot.MetaModelValidation(input_test, output_test, metamodel)</code>

## Design of experiments

Purpose	Class / Method
Monte-Carlo	<code>ot.MonteCarloExperiment(distribution, sample_size)</code>
Latin Hypercube Sampling	<code>ot.LHSExperiment(distribution, sample_size)</code>
Optimized LHS	<code>ot.SimulatedAnnealingLHS(lhs_experiment, criteria, temperature_profile)</code>
Sobol' sequence	<code>ot.SobolSequence(dimension)</code>

Purpose	Class / Method
Low discrepancy sequence	<code>ot.LowDiscrepancyExperiment(ld_sequence, distribution, samplesize, False)</code>
Import viewer	<code>import openturns.viewer as otv</code>
Plot DOE	<code>otv.PlotDesign(sample, bounds)</code>

## Graphics

Purpose	Class / Method
Import viewer	<code>import openturns.viewer as otv</code>
Graph	<code>ot.Graph(title, x_title, y_title, show_axes_bool)</code>
Curve	<code>ot.Curve(sample_x, sample_y)</code>
Cloud	<code>ot.Cloud(sample_x, sample_y)</code>
Set colors	<code>graph.setColors(ot.Drawable().BuildDefaultPalette(number_of_colors))</code>
Customize size	<code>view = otv.View(graph, figure_kw={"figsize": (4.0, 3.0)})</code>
Save figure	<code>view.getFigure().savefig("filename.pdf", bbox_inches="tight")</code>

## More resources

Resource	Link
Forum	<a href="https://openturns.discourse.group">https://openturns.discourse.group</a>
Chat	<a href="https://gitter.im/openturns/community">https://gitter.im/openturns/community</a>
Modules	<a href="https://github.com/openturns/openturns/wiki/Modules">https://github.com/openturns/openturns/wiki/Modules</a>
Install	<a href="http://openturns.github.io/openturns/master/install.html">http://openturns.github.io/openturns/master/install.html</a>
Bugs	<a href="https://github.com/openturns/openturns/issues">https://github.com/openturns/openturns/issues</a>
Events	<a href="https://github.com/openturns/openturns/wiki/OpenTURNS-events">https://github.com/openturns/openturns/wiki/OpenTURNS-events</a>
Bibliography	<a href="https://github.com/openturns/openturns/wiki/Bibliography">https://github.com/openturns/openturns/wiki/Bibliography</a>
Bib resources	Bibtex file
Presentations	<a href="https://github.com/openturns/presentation">https://github.com/openturns/presentation</a>