OpenTURNS release highlights : the new Gaussian Process API

S.Haddad (Airbus Central R&T)

User Day #18, June 13th 2025, EDF Lab

Contents

Kriging implementation: basic example

```python
import openturns as ot
...
# Call kriging
kriging_algo = ot.KrigingAlgorithm(X_train, Y_train, covarianceModel, basis)
kriging_algo.run()
# Get the result
kriging_result = kriging_algo.getResult()
# Post-processing
func = kriging_result.getMetaModel()
# Conditional variance
cond_var = kriging_result.getConditionalVariance(new_X)
```

First remarks:

- `KrigingAlgorithm` handles the E2E computation,
- Post-processing methods handled by result structures;

Kriging implementation: change some parameters

```
import openturns as ot

# Set optimizer
ot.ResourceMap.SetAsString(
  "GeneralLinearModelAlgorithm-DefaultOptimizationAlgorithm", "Cobyla")
ot.ResourceMap.SetAsScalar(
  "GeneralLinearModelAlgorithm-DefaultOptimizationLowerBound", 0)
ot.ResourceMap.SetAsScalar(
  "GeneralLinearModelAlgorithm-DefaultOptimizationUpperBound", 2)
ot.ResourceMap.SetAsString(
  "KrigingAlgorithm-LinearAlgebra", "LAPACK")
#ot.ResourceMap.SetAsString(
#  "GeneralLinearModelAlgorithm-LinearAlgebra", "LAPACK")

...
# Call kriging
kriging_algo = ot.KrigingAlgorithm(X_train, Y_train, covarianceModel, basis)
kriging_algo.run()
# Get the result
kriging_result = kriging_algo.getResult()
# Post-processing
func = kriging_result.getMetaModel()
# Conditional variance
cond_var = kriging_result.getConditionalVariance(new_X)
```

In a nutshell

`KrigingAlgorithm` is used to fit a Kriging model (aka Gaussian Process Regression), relying on a 2-steps procedure :

- `GeneralLinearModelAlgorithm`: allowing the parametric estimation of a Gaussian Process,
- `KrigingAlgorithm`: conditioning the Gaussian Process;

$\longrightarrow$ `KrigingAlgorithm.run` calibrate a Gaussian Process

- `ResourceMap` keys duplicate,
- Sequential Kriging hard to handle (example for `EGO`);

Our wishes

- Trigger explicitly the parameters fitting,
- Perform the conditioning,
- Enrich the API with missing features (such as "known trend"),
- Build as much post-processing functions as needed;

New API for Gaussian Process Regression

The new API defines the following classes (in the `experimental` submodule):

- `GaussianProcessFitter`: Fitting the Gaussian Process (explicitly),
- `GaussianProcessFitterResult`: result class of a parametric Gaussian Process fitting,
- `GaussianProcessRegression`: conditioning the Gaussian Process,
- `GaussianProcessRegressionResult`: result class of a conditional Gaussian Process fitting,
- `GaussianProcessRandomVector`: generate Gaussian Process realizations,
- `GaussianProcessConditionalCovariance`: Post-processing Gaussian Process;

```
# Call fitter
fitter_algo = otexp.GaussianProcessFitter(X_train, Y_train, covarianceModel, basis)
fitter_algo.run()
fitter_result = fitter_algo.getResult()
# Conditioning part using the fit result
gpr_algo = otexp.GaussianProcessRegression(fitter_result)
gpr_algo.run()
gpr_result = gpr_algo.getResult()
gpr_metamodel = gpr_result.getMetaModel()
```

New feature : known trend

```
# trend function
trend_function = ot.SymbolicFunction("x", "-3.1710410094572903")
# Covariance
scale = [4.51669]
amplitude = [8.648]
covariance_opt = ot.MaternModel(scale, amplitude, 1.5)
# Conditioning part using the data
gpr_algo_noopt = otexp.GaussianProcessRegression(x_train, y_train, covariance_opt, t
gpr_algo_noopt.run()
gpr_result_no_opt = gpr_algo_noopt.getResult()
gpr_nopt_Metamodel = gpr_result_no_opt.getMetaModel()
```

Post-processing : conditional covariance

```
# Call fitter
fitter_algo = otexp.GaussianProcessFitter(X_train, Y_train, covarianceModel, basis)
fitter_algo.run()
fitter_result = fitter_algo.getResult()
# Conditioning part using the fit result
gpr_algo = otexp.GaussianProcessRegression(fitter_result)
gpr_algo.run()
gpr_result = gpr_algo.getResult()
# Conditional covariance
gpcc = otexp.GaussianProcessConditionalCovariance(gpr_result)
cond_var = gpcc.getConditionalVariance(new_X)
```

Kriging vs Gaussian Process

Reach out here to learn more !

openturns.github.io/openturns/master/auto_getting_started/plot_kriging_vs_gpr.html#sphx-glr-auto-getting-started-plot-kriging-vs-gpr-py

OpenTURNS

Home Get it Doc Forum Modules Code Bugs

An Open source initiative for the Treatment of Uncertainties, Risks'N Statistics

OpenTURNS 1.25dev documentation » Contents » Examples » Getting started » Gaussian Process Regression vs KrigingAlgorithm previous | next | index

Gaussian Process Regression vs KrigingAlgorithm

The goal of this example is to highlight the main changes between the old API involving `KrigingAlgorithm` and the new one.

It assumes a basic knowledge of Gaussian Process Regression. For that purpose, we create a Gaussian Process Regression surrogate model for a function which has scalar real inputs and outputs. We select a very simple example.

Introduction

We consider the sine function:

$$y = x \sin(x)$$

for any $x \in [0, 12]$.

We want to create a surrogate of this function. This is why we create a sample of n observations of the function:

$$y_i = x_i \sin(x_i)$$

We are going to consider a Gaussian Process Regression with:

- a constant trend,
- a Matern covariance model.

Table of Contents

Previous topic

Getting started

Next topic

Data analysis

This Page

Show Source

Summary

Feature	OpenTURNS 1.24	New API
Optimisation	TNC	Cobyla
Heteroscedasticity	KrigingAlgorithm.setNoise	Not implemented
Nugget factor est	CovModel	CovModel
Known trend	Not implemented	Implemented
Conditional covariance	KrigingResult	GPCC*

*GPCC: GaussianProcessConditionalCovariance

Integration within OpenTURNS

List of classes supporting the new API:

- `EfficientGlobalOptimization`: rely on `GaussianProcessRegressionResult`,
- `ConditionedGaussianProcess`: rely on `GaussianProcessRegressionResult`

Remark : these classes are now part of the experimental submodule!
In parallel, all examples involving Kriging are progressively moving to the new API !

Outlook

2025-2026 work

- Finalize migration of the examples to the new API,
- Algebra of covariance models,
- Analytical gradient of covariance models,
- Integration into the existing algorithms,
- Cross-validation methods,
- Sequential algorithms,

END

Thank you for your attention!
Any questions?