

---

# otpod Documentation

*Release*

**Antoine Dumas**

**Oct 05, 2016**



## CONTENTS

<b>1</b>	<b>Contents:</b>	<b>3</b>
1.1	Documentation of the API . . . . .	3
1.1.1	Data analysis . . . . .	3
1.1.2	POD computation methods . . . . .	9
1.1.3	Adaptive algorithms . . . . .	29
1.1.4	Tools . . . . .	42
1.2	Examples of the API . . . . .	47
1.2.1	Linear model analysis . . . . .	47
1.2.2	Linear model analysis with censored data . . . . .	56
1.2.3	Linear model POD . . . . .	60
1.2.4	Linear model POD with censored data . . . . .	67
1.2.5	Quantile Regression POD . . . . .	71
1.2.6	Polynomial chaos POD . . . . .	75
1.2.7	Kriging POD . . . . .	83
1.2.8	POD Summary . . . . .	90
1.2.9	Adaptive Signal POD using Kriging . . . . .	106
1.2.10	Adaptive Hit Miss POD . . . . .	117
<b>2</b>	<b>Indices and tables</b>	<b>129</b>
	<b>Index</b>	<b>131</b>



otpod is a module for [OpenTURNS](#).



## CONTENTS:

### 1.1 Documentation of the API

This is the user manual for the Python bindings to the otpod library.

#### 1.1.1 Data analysis

---

<i>UnivariateLinearModelAnalysis</i>	Linear regression analysis with residuals hypothesis tests.
--------------------------------------	---

---

#### UnivariateLinearModelAnalysis

**class UnivariateLinearModelAnalysis** (\*args)

Linear regression analysis with residuals hypothesis tests.

**Available constructors:**

UnivariateLinearModelAnalysis(*inputSample*, *outputSample*)

UnivariateLinearModelAnalysis(*inputSample*, *outputSample*, *noiseThres*, *saturationThres*, *resDistFact*, *boxCox*)

**Parameters** **inputSample** : 2-d sequence of float

Vector of the defect sizes, of dimension 1.

**outputSample** : 2-d sequence of float

Vector of the signals, of dimension 1.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None.

**resDistFact** : `openturns.DistributionFactory`

Distribution hypothesis followed by the residuals. Default is `openturns.NormalFactory`.

**boxCox** : bool or float

Enable or not the Box Cox transformation. If boxCox is a float, the Box Cox transformation is enabled with the given value. Default is False.

## Notes

This method automatically :

- computes the Box Cox parameter if *boxCox* is True,
- computes the transformed signals if *boxCox* is True or a float,
- builds the univariate linear regression model on the data,
- computes the linear regression parameters for censored data if needed,
- computes the residuals,
- runs all hypothesis tests.

## Examples

Generate data :

```
>>> import openturns as ot
>>> import otpod
>>> N = 100
>>> ot.RandomGenerator.SetSeed(0)
>>> defectDist = ot.Uniform(0.1, 0.6)
>>> epsilon = ot.Normal(0, 1.9)
>>> defects = defectDist.getSample(N)
>>> signalsInvBoxCox = defects * 43. + epsilon.getSample(N) + 2.5
>>> invBoxCox = ot.InverseBoxCoxTransform(0.3)
>>> signals = invBoxCox(signalsInvBoxCox)
```

Run analysis with gaussian hypothesis on the residuals :

```
>>> analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, boxCox=True)
>>> print analysis.getIntercept() # get intercept value
[Intercept for uncensored case : 2.51037]
>>> print analysis.getKolmogorovPValue()
[Kolmogorov p-value for uncensored case : 0.835529]
```

Run analysis with noise and saturation threshold :

```
>>> analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, 60., 1700.,
↳boxCox=True)
>>> print analysis.getIntercept() # get intercept value for uncensored and
↳censored case
[Intercept for uncensored case : 4.28758, Intercept for censored case : 3.11243]
>>> print analysis.getKolmogorovPValue()
[Kolmogorov p-value for uncensored case : 0.346827, Kolmogorov p-value for
↳censored case : 0.885006]
```

Run analysis with a Weibull distribution hypothesis on the residuals

```
>>> analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, 60., 1700.,
↳ot.WeibullFactory(), boxCox=True)
>>> print analysis.getIntercept() # get intercept value for uncensored and
↳censored case
[Intercept for uncensored case : 4.28758, Intercept for censored case : 3.11243]
>>> print analysis.getKolmogorovPValue()
[Kolmogorov p-value for uncensored case : 0.476036, Kolmogorov p-value for
↳censored case : 0.71764]
```



## Methods

<code>drawBoxCoxLikelihood([name])</code>	Draw the loglikelihood versus the Box Cox parameter.
<code>drawLinearModel([model, name])</code>	Draw the linear regression prediction versus the true data.
<code>drawResiduals([model, name])</code>	Draw the residuals versus the defect values.
<code>drawResidualsDistribution([model, name])</code>	Draw the residuals histogram with the fitted distribution.
<code>drawResidualsQQplot([model, name])</code>	Draw the residuals QQ plot with the fitted distribution.
<code>getAndersonDarlingPValue()</code>	Accessor to the Anderson Darling test p-value.
<code>getBoxCoxParameter()</code>	Accessor to the Box Cox parameter.
<code>getBreuschPaganPValue()</code>	Accessor to the Breusch Pagan test p-value.
<code>getCramerVonMisesPValue()</code>	Accessor to the Cramer Von Mises test p-value.
<code>getDurbinWatsonPValue()</code>	Accessor to the Durbin Watson test p-value.
<code>getHarrisonMcCabePValue()</code>	Accessor to the Harrison McCabe test p-value.
<code>getInputSample()</code>	Accessor to the input sample.
<code>getIntercept()</code>	Accessor to the intercept of the linear regression model.
<code>getKolmogorovPValue()</code>	Accessor to the Kolmogorov test p-value.
<code>getNoiseThreshold()</code>	Accessor to the noise threshold.
<code>getOutputSample()</code>	Accessor to the output sample.
<code>getR2()</code>	Accessor to the R2 value.
<code>getResiduals()</code>	Accessor to the residuals.
<code>getResidualsDistribution()</code>	Accessor to the residuals distribution.
<code>getSaturationThreshold()</code>	Accessor to the saturation threshold.
<code>getSlope()</code>	Accessor to the slope of the linear regression model.
<code>getStandardError()</code>	Accessor to the standard error of the estimate.
<code>getZeroMeanPValue()</code>	Accessor to the Zero Mean test p-value.
<code>printResults()</code>	Print results of the linear analysis.
<code>saveResults(name)</code>	Save all analysis test results in a file.

### **drawBoxCoxLikelihood** (*name=None*)

Draw the loglikelihood versus the Box Cox parameter.

**Parameters** *name* : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** *fig* : matplotlib.figure

Matplotlib figure object.

*ax* : matplotlib.axes

Matplotlib axes object.

## Notes

This method is available only when the parameter *boxCox* is set to True.

**drawLinearModel** (*model='uncensored', name=None*)

Draw the linear regression prediction versus the true data.

**Parameters** **model** : string

The linear regression model to be used, either *uncensored* or *censored* if censored threshold were given. Default is *uncensored*.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : [matplotlib.figure](#)

Matplotlib figure object.

**ax** : [matplotlib.axes](#)

Matplotlib axes object.

**drawResiduals** (*model='uncensored', name=None*)

Draw the residuals versus the defect values.

**Parameters** **model** : string

The residuals to be used, either *uncensored* or *censored* if censored threshold were given. Default is *uncensored*.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : [matplotlib.figure](#)

Matplotlib figure object.

**ax** : [matplotlib.axes](#)

Matplotlib axes object.

**drawResidualsDistribution** (*model='uncensored', name=None*)

Draw the residuals histogram with the fitted distribution.

**Parameters** **model** : string

The residuals to be used, either *uncensored* or *censored* if censored threshold were given. Default is *uncensored*.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : [matplotlib.figure](#)

Matplotlib figure object.

**ax** : [matplotlib.axes](#)

Matplotlib axes object.

**drawResidualsQQplot** (*model='uncensored', name=None*)

Draw the residuals QQ plot with the fitted distribution.

**Parameters** **model** : string

The residuals to be used, either *uncensored* or *censored* if censored threshold were given. Default is *uncensored*.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**getAndersonDarlingPValue** ()

Accessor to the Anderson Darling test p-value.

**Returns** **pValue** : `openturns.NumericalPoint`

Either the p-value for the uncensored case or for both cases.

**getBoxCoxParameter** ()

Accessor to the Box Cox parameter.

**Returns** **lambdaBoxCox** : float

The Box Cox parameter used to transform the data. If the transformation is not enabled None is returned.

**getBreuschPaganPValue** ()

Accessor to the Breusch Pagan test p-value.

**Returns** **pValue** : `openturns.NumericalPoint`

Either the p-value for the uncensored case or for both cases.

**getCramerVonMisesPValue** ()

Accessor to the Cramer Von Mises test p-value.

**Returns** **pValue** : `openturns.NumericalPoint`

Either the p-value for the uncensored case or for both cases.

**getDurbinWatsonPValue** ()

Accessor to the Durbin Watson test p-value.

**Returns** **pValue** : `openturns.NumericalPoint`

Either the p-value for the uncensored case or for both cases.

**getHarrisonMcCabePValue** ()

Accessor to the Harrison McCabe test p-value.

**Returns** **pValue** : `openturns.NumericalPoint`

Either the p-value for the uncensored case or for both cases.

**getInputSample** ()

Accessor to the input sample.

**Returns defects :** `openturns.NumericalSample`

The input sample which is the defect values.

**getIntercept ()**

Accessor to the intercept of the linear regression model.

**Returns intercept :** `openturns.NumericalPoint`

The intercept parameter for the uncensored and censored (if so) linear regression model.

**getKolmogorovPValue ()**

Accessor to the Kolmogorov test p-value.

**Returns pValue :** `openturns.NumericalPoint`

Either the p-value for the uncensored case or for both cases.

**getNoiseThreshold ()**

Accessor to the noise threshold.

**Returns noiseThres :** float

The noise threshold if it exists, if not it returns *None*.

**getOutputSample ()**

Accessor to the output sample.

**Returns signals :** `openturns.NumericalSample`

The input sample which is the signal values.

**getR2 ()**

Accessor to the R2 value.

**Returns R2 :** `openturns.NumericalPoint`

Either the R2 for the uncensored case or for both cases.

**getResiduals ()**

Accessor to the residuals.

**Returns residuals :** `openturns.NumericalSample`

The residuals computed from the uncensored and censored linear regression model. The first column corresponds with the uncensored case.

**getResidualsDistribution ()**

Accessor to the residuals distribution.

**Returns distribution :** list of `openturns.Distribution`

The fitted distribution on the residuals, computed in the uncensored and censored (if so) case.

**getSaturationThreshold ()**

Accessor to the saturation threshold.

**Returns saturationThres :** float

The saturation threshold if it exists, if not it returns *None*.

**getSlope ()**

Accessor to the slope of the linear regression model.

**Returns slope :** `openturns.NumericalPoint`

The slope parameter for the uncensored and censored (if so) linear regression model.

**getStandardError ()**

Accessor to the standard error of the estimate.

**Returns** `stderr` : `openturns.NumericalPoint`

The standard error of the estimate for the uncensored and censored (if so) linear regression model.

**getZeroMeanPValue ()**

Accessor to the Zero Mean test p-value.

**Returns** `pValue` : `openturns.NumericalPoint`

Either the p-value for the uncensored case or for both cases.

**printResults ()**

Print results of the linear analysis.

**saveResults (name)**

Save all analysis test results in a file.

**Parameters** `name` : string

Name of the file or full path name.

**Notes**

The file can be saved as a csv file. Separations are made with tabulations.

If *name* is the file name, then it is saved in the current working directory.

## 1.1.2 POD computation methods

<i>UnivariateLinearModelPOD</i>	Linear regression based POD.
<i>QuantileRegressionPOD</i>	Quantile regression based POD.
<i>PolynomialChaosPOD</i>	Polynomial chaos based POD.
<i>KrigingPOD</i>	Kriging based POD.

### UnivariateLinearModelPOD

**class** `UnivariateLinearModelPOD (*args)`

Linear regression based POD.

**Available constructors:**

`UnivariateLinearModelPOD(analysis=analysis, detection=detection)`

`UnivariateLinearModelPOD(inputSample, outputSample, detection, noiseThres, saturationThres, resDistFact, boxCox)`

**Parameters** `analysis` : `UnivariateLinearModelAnalysis`

Linear analysis object.

**inputSample** : 2-d sequence of float

Vector of the defect sizes, of dimension 1.

**outputSample** : 2-d sequence of float

Vector of the signals, of dimension 1.

**detection** : float

Detection value of the signal.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**resDistFact** : `openturns.DistributionFactory`

Distribution hypothesis followed by the residuals. Default is None.

**boxCox** : bool or float

Enable or not the Box Cox transformation. If boxCox is a float, the Box Cox transformation is enabled with the given value. Default is False.

## Notes

This class aims at building the POD based on a linear regression model. If a linear analysis has been launched, it can be used as prescribed in the first constructor. It can be noticed that, in this case, with the default parameters of the linear analysis, the POD will corresponds with the linear regression model associated to a Gaussian hypothesis on the residuals.

Otherwise, all parameters can be given as in the second constructor.

Following the given distribution in *resDistFact*, the POD model is built different hypothesis:

- if *resDistFact* = *None*, it corresponds with Berens-Binomial. This is the default case.
- if *resDistFact* = `openturns.NormalFactory`, it corresponds with Berens-Gauss.
- if *resDistFact* = {`openturns.KernelSmoothing`, `openturns.WeibullFactory`, ...}, the confidence interval is built by bootstrap.

If bootstrap is used, a progress bar is shown if the verbosity is enabled. It can be disabled using the method *setVerbose*.

## Methods

<code>computeDetectionSize(*args, **kwargs)</code>	Compute the detection size for a given probability level.
<code>drawBoxCoxLikelihood([name])</code>	Draw the loglikelihood versus the Box Cox parameter.
<code>drawPOD(*args, **kwargs)</code>	Draw the POD curve.
<code>getBoxCoxParameter()</code>	Accessor to the Box Cox parameter.
<code>getPODCLModel([confidenceLevel])</code>	Accessor to the POD model at a given confidence level.
<code>getPODModel()</code>	Accessor to the POD model.
<code>getR2()</code>	Accessor to the R2 value.
<code>getSimulationSize()</code>	Accessor to the simulation size.
<code>getVerbose()</code>	Accessor to the verbosity.
<code>run()</code>	Build the POD models.
<code>setSimulationSize(size)</code>	Accessor to the simulation size.
<code>setVerbose(verbose)</code>	Accessor to the verbosity.

**computeDetectionSize** (\*args, \*\*kwargs)

Compute the detection size for a given probability level.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**Returns** **result** : collection of `openturns.NumericalPointWithDescription`

A `NumericalPointWithDescription` containing the detection size computed at the given probability level and confidence level if provided.

**drawBoxCoxLikelihood** (name=None)

Draw the loglikelihood versus the Box Cox parameter.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

## Notes

This method is available only when the parameter *boxCox* is set to True.

**drawPOD** (\*args, \*\*kwargs)

Draw the POD curve.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**defectMin, defectMax** : float

Define the interval where the curve is plotted. Default : min and max values of the input sample.

**nbPt** : int

The number of points to draw the curves. Default is 100.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** `fig` : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**getBoxCoxParameter** ()

Accessor to the Box Cox parameter.

**Returns** `lambdaBoxCox` : float

The Box Cox parameter used to transform the data. If the transformation is not enabled None is returned.

**getPODCLModel** (*confidenceLevel=0.95*)

Accessor to the POD model at a given confidence level.

**Parameters** `confidenceLevel` : float

The confidence level the POD must be computed. Default is 0.95

**Returns** `PODModelCI` : `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value at the confidence level given as parameter.

**getPODModel** ()

Accessor to the POD model.

**Returns** `PODModel` : `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value.

**getR2** ()

Accessor to the R2 value.

**Returns** `R2` : float

The R2 value.

**getSimulationSize** ()

Accessor to the simulation size.

**Returns** `size` : int

The size of the simulation used to compute the confidence interval.

**getVerbose** ()

Accessor to the verbosity.

**Returns** `verbose` : bool

Enable or disable the verbosity. Default is True.

**run** ()

Build the POD models.

## Notes

This method build the linear model for the uncensored or censored case depending of the input parameters. Then it builds the POD model following the given residuals distribution factory.



**setSimulationSize** (*size*)

Accessor to the simulation size.

**Parameters** **size** : int

The size of the simulation used to compute the confidence interval.

**setVerbose** (*verbose*)

Accessor to the verbosity.

**Parameters** **verbose** : bool

Enable or disable the verbosity.

## QuantileRegressionPOD

class **QuantileRegressionPOD** (*\*args*)

Quantile regression based POD.

**Available constructor:**

QuantileRegressionPOD(*inputSample, outputSample, detection, noiseThres, saturationThres, boxCox*)

**Parameters** **inputSample** : 2-d sequence of float

Vector of the defect sizes, of dimension 1.

**outputSample** : 2-d sequence of float

Vector of the signals, of dimension 1.

**detection** : float

Detection value of the signal.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**boxCox** : bool or float

Enable or not the Box Cox transformation. If boxCox is a float, the Box Cox transformation is enabled with the given value. Default is False.

## Notes

This class aims at building the POD based on a quantile regression model. The return POD model corresponds with an interpolate function built with the defect values computed for the given quantile as parameters. The default is 21 quantile values from 0.05 to 0.98. They can be user-defined using the method *setQuantile*.

The confidence level is computed by bootstrap. The POD model at the given confidence level is also an interpolate function based on the defect quantile value computed at the given confidence level.

The computeDetectionSize method calls the real quantile regression at the given probability level.

A progress bar is shown if the verbosity is enabled. It can be disabled using the method *setVerbose*.

## Methods

<code>computeDetectionSize(*args, **kwargs)</code>	Compute the detection size for a given probability level.
<code>drawBoxCoxLikelihood([name])</code>	Draw the loglikelihood versus the Box Cox parameter.
<code>drawLinearModel(probabilityLevel[, name])</code>	Draw the quantile regression prediction versus the true data.
<code>drawPOD(*args, **kwargs)</code>	Draw the POD curve.
<code>getBoxCoxParameter()</code>	Accessor to the Box Cox parameter.
<code>getPODCLModel([confidenceLevel])</code>	Accessor to the POD model at a given confidence level.
<code>getPODModel()</code>	Accessor to the POD model.
<code>getQuantile()</code>	Accessor to the quantile list for the regression.
<code>getR2(quantile)</code>	Accessor to the pseudo R2 value.
<code>getSimulationSize()</code>	Accessor to the simulation size.
<code>getVerbose()</code>	Accessor to the verbosity.
<code>run()</code>	Build the POD models.
<code>setQuantile(quantile)</code>	Accessor to the quantile list for the regression.
<code>setSimulationSize(size)</code>	Accessor to the simulation size.
<code>setVerbose(verbose)</code>	Accessor to the verbosity.

**computeDetectionSize** (*\*args, \*\*kwargs*)

Compute the detection size for a given probability level.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**Returns** **result** : collection of `openturns.NumericalPointWithDescription`

A `NumericalPointWithDescription` containing the detection size computed at the given probability level and confidence level if provided.

**drawBoxCoxLikelihood** (*name=None*)

Draw the loglikelihood versus the Box Cox parameter.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

## Notes

This method is available only when the parameter *boxCox* is set to True.

**drawLinearModel** (*probabilityLevel, name=None*)

Draw the quantile regression prediction versus the true data.

**Parameters** `probabilityLevel` : float

The probability level for which the quantile regression is performed

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** `fig` : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**drawPOD** (*\*args, \*\*kwargs*)

Draw the POD curve.

**Parameters** `probabilityLevel` : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**defectMin, defectMax** : float

Define the interval where the curve is plotted. Default : min and max values of the input sample.

**nbPt** : int

The number of points to draw the curves. Default is 100.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** `fig` : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**getBoxCoxParameter** ()

Accessor to the Box Cox parameter.

**Returns** `lambdaBoxCox` : float

The Box Cox parameter used to transform the data. If the transformation is not enabled None is returned.

**getPODCLModel** (*confidenceLevel=0.95*)

Accessor to the POD model at a given confidence level.

**Parameters** `confidenceLevel` : float

The confidence level the POD must be computed. Default is 0.95

**Returns** `PODModelCI`: `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value at the confidence level given as parameter.

**getPODModel** ()

Accessor to the POD model.

**Returns** `PODModel`: `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value.

**getQuantile** ()

Accessor to the quantile list for the regression.

**getR2** (*quantile*)

Accessor to the pseudo R2 value.

**Parameters** `quantile`: float

The quantile value for which the regression is performed.

**Returns** `R2`: float

The pseudo R2 value.

**getSimulationSize** ()

Accessor to the simulation size.

**Returns** `size`: int

The size of the simulation used to compute the confidence interval.

**getVerbose** ()

Accessor to the verbosity.

**Returns** `verbose`: bool

Enable or disable the verbosity. Default is True.

**run** ()

Build the POD models.

## Notes

This method build the quantile regression model. First the censored data are filtered if needed. The Box Cox transformation is performed if it is enabled. Then it builds the POD model for given data and computes using bootstrap all the defects quantile needed to build the POD model at the confidence level.

**setQuantile** (*quantile*)

Accessor to the quantile list for the regression.

**Parameters** `quantile`: sequence of float

The quantile value for which the regression is performed and the corresponding defect size is computed.

**setSimulationSize** (*size*)

Accessor to the simulation size.

**Parameters** `size`: int

The size of the simulation used to compute the confidence interval.

**setVerbose** (*verbose*)

Accessor to the verbosity.

**Parameters** **verbose** : bool

Enable or disable the verbosity.

## PolynomialChaosPOD

**class** **PolynomialChaosPOD** (*\*args*)

Polynomial chaos based POD.

**Available constructor:**

**PolynomialChaosPOD**(*inputSample, outputSample, detection, noiseThres, saturationThres, boxCox*)

**Parameters** **inputSample** : 2-d sequence of float

Vector of the input values. The first column must correspond with the defect sizes.

**outputSample** : 2-d sequence of float

Vector of the signals, of dimension 1.

**detection** : float

Detection value of the signal.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**boxCox** : bool or float

Enable or not the Box Cox transformation. If boxCox is a float, the Box Cox transformation is enabled with the given value. Default is False.

**Warning:** The first column of the input sample must corresponds with the defects sample.

## Notes

This class aims at building the POD based on a polynomial chaos model. This method must be used under the assumption that the residuals follows a Normal distribution.

The return POD model corresponds with an interpolate function built with the POD values computed for the given defect sizes. The default values are 20 defect sizes between the minimum and maximum value of the defect sample. The defect sizes can be changed using the method *setDefectSizes*.

The default polynomial chaos model is built with uniform distributions for each parameters. Coefficients are computed using the LAR algorithm combined with the KFold. The AdaptiveStrategy is chosen fixed with a linear enumerate function of maximum degree 3.

For advanced use, all parameters can be defined thanks to dedicated set methods. Moreover, if the user has already built a polynomial chaos result, it can be given as parameter using the method *setPolynomialChaosResult*, then the POD is computed based on this polynomial chaos result.

A progress bar is shown if the verbosity is enabled. It can be disabled using the method *setVerbose*.

## Methods

<code>computeDetectionSize(*args, **kwargs)</code>	Compute the detection size for a given probability level.
<code>drawBoxCoxLikelihood([name])</code>	Draw the loglikelihood versus the Box Cox parameter.
<code>drawPOD(*args, **kwargs)</code>	Draw the POD curve.
<code>drawPolynomialChaosModel([name])</code>	Draw the polynomial chaos prediction versus the true data.
<code>drawValidationGraph(*args, **kwargs)</code>	Draw the validation graph of the metamodel.
<code>getAdaptiveStrategy()</code>	Accessor to the adaptive strategy.
<code>getBoxCoxParameter()</code>	Accessor to the Box Cox parameter.
<code>getDefectSizes()</code>	Accessor to the defect size where POD is computed.
<code>getDegree()</code>	Accessor to the polynomial chaos degree.
<code>getDistribution()</code>	Accessor to the parameters distribution.
<code>getPODCLModel([confidenceLevel])</code>	Accessor to the POD model at a given confidence level.
<code>getPODModel()</code>	Accessor to the POD model.
<code>getPolynomialChaosResult()</code>	Accessor to the polynomial chaos result.
<code>getProjectionStrategy()</code>	Accessor to the projection strategy.
<code>getQ2()</code>	Accessor to the Q2 value.
<code>getR2()</code>	Accessor to the R2 value.
<code>getSamplingSize()</code>	Accessor to the Monte Carlo sampling size.
<code>getSimulationSize()</code>	Accessor to the simulation size.
<code>getVerbose()</code>	Accessor to the verbosity.
<code>run()</code>	Build the POD models.
<code>setAdaptiveStrategy(strategy)</code>	Accessor to the adaptive strategy.
<code>setDefectSizes(size)</code>	Accessor to the defect size where POD is computed.
<code>setDegree(degree)</code>	Accessor to the polynomial chaos degree.
<code>setDistribution(distribution)</code>	Accessor to the parameters distribution.
<code>setPolynomialChaosResult(chaosResult)</code>	Accessor to the polynomial chaos result.
<code>setProjectionStrategy(strategy)</code>	Accessor to the projection strategy.
<code>setSamplingSize(size)</code>	Accessor to the Monte Carlo sampling size.
<code>setSimulationSize(size)</code>	Accessor to the simulation size.
<code>setVerbose(verbose)</code>	Accessor to the verbosity.

**computeDetectionSize** (*\*args, \*\*kwargs*)

Compute the detection size for a given probability level.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**Returns** **result** : collection of `openturns.NumericalPointWithDescription`

A `NumericalPointWithDescription` containing the detection size computed at the given probability level and confidence level if provided.

**drawBoxCoxLikelihood** (*name=None*)

Draw the loglikelihood versus the Box Cox parameter.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

## Notes

This method is available only when the parameter *boxCox* is set to True.

**drawPOD** (\*args, \*\*kwargs)

Draw the POD curve.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**defectMin, defectMax** : float

Define the interval where the curve is plotted. Default : min and max values of the input sample.

**nbPt** : int

The number of points to draw the curves. Default is 100.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**drawPolynomialChaosModel** (name=None)

Draw the polynomial chaos prediction versus the true data.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`



Matplotlib axes object.

## Notes

This method only works if the dimension of the input sample is 1.

**drawValidationGraph** (\*args, \*\*kwargs)

Draw the validation graph of the metamodel.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : matplotlib.figure

Matplotlib figure object.

**ax** : matplotlib.axes

Matplotlib axes object.

**getAdaptiveStrategy** ()

Accessor to the adaptive strategy.

**Returns** **strategy** : openturns.AdaptiveStrategy

The adaptive strategy for the polynomial chaos.

**getBoxCoxParameter** ()

Accessor to the Box Cox parameter.

**Returns** **lambdaBoxCox** : float

The Box Cox parameter used to transform the data. If the transformation is not enabled None is returned.

**getDefectSizes** ()

Accessor to the defect size where POD is computed.

**Returns** **defectSize** : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**getDegree** ()

Accessor to the polynomial chaos degree.

**Returns** **degree** : int

The degree of the polynomial chaos.

**getDistribution** ()

Accessor to the parameters distribution.

**Returns** **distribution** : openturns.ComposedDistribution

The input parameters distribution, default is a Uniform distribution for all parameters.

**getPODCLModel** (confidenceLevel=0.95)

Accessor to the POD model at a given confidence level.

**Parameters** **confidenceLevel** : float

The confidence level the POD must be computed. Default is 0.95

**Returns** `PODModelCI`: `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value at the confidence level given as parameter.

**getPODModel** ()

Accessor to the POD model.

**Returns** `PODModel`: `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value.

**getPolynomialChaosResult** ()

Accessor to the polynomial chaos result.

**Returns** `result`: `openturns.FunctionalChaosResult`

The polynomial chaos result.

**getProjectionStrategy** ()

Accessor to the projection strategy.

**Returns** `strategy`: `openturns.ProjectionStrategy`

The projection strategy for the polynomial chaos.

**getQ2** ()

Accessor to the Q2 value.

**Returns** `Q2`: float

The Q2 value computed analytically.

**getR2** ()

Accessor to the R2 value.

**Returns** `R2`: float

The R2 value.

**getSamplingSize** ()

Accessor to the Monte Carlo sampling size.

**Returns** `size`: int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**getSimulationSize** ()

Accessor to the simulation size.

**Returns** `size`: int

The size of the simulation used to compute the confidence interval.

**getVerbose** ()

Accessor to the verbosity.

**Returns** `verbose`: bool

Enable or disable the verbosity. Default is True.

**run** ()

Build the POD models.

## Notes

This method build the polynomial chaos model. First the censored data are filtered if needed. The Box Cox transformation is performed if it is enabled. Then it builds the POD models, the Monte Carlo simulation is performed for each given defect sizes. The confidence interval is computed by simulating new coefficients of the polynomial chaos, then Monte Carlo simulations are performed.

**setAdaptiveStrategy** (*strategy*)

Accessor to the adaptive strategy.

**Parameters** **strategy** : `openturns.AdaptiveStrategy`

The adaptive strategy for the polynomial chaos.

**setDefectSizes** (*size*)

Accessor to the defect size where POD is computed.

**Parameters** **defectSize** : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**setDegree** (*degree*)

Accessor to the polynomial chaos degree.

**Parameters** **degree** : int

The degree of the polynomial chaos.

**setDistribution** (*distribution*)

Accessor to the parameters distribution.

**Parameters** **distribution** : `openturns.ComposedDistribution`

The input parameters distribution.

**setPolynomialChaosResult** (*chaosResult*)

Accessor to the polynomial chaos result.

**Parameters** **chaosResult** : `openturns.FunctionalChaosResult`

The polynomial chaos result.

**setProjectionStrategy** (*strategy*)

Accessor to the projection strategy.

**Parameters** **strategy** : `openturns.ProjectionStrategy`

The projection strategy for the polynomial chaos.

**setSamplingSize** (*size*)

Accessor to the Monte Carlo sampling size.

**Parameters** **size** : int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**setSimulationSize** (*size*)

Accessor to the simulation size.

**Parameters** **size** : int

The size of the simulation used to compute the confidence interval.

**setVerbose** (*verbose*)

Accessor to the verbosity.

**Parameters** **verbose** : bool

Enable or disable the verbosity.

## KrigingPOD

**class** **KrigingPOD** (\*args)

Kriging based POD.

### Available constructor:

**KrigingPOD**(*inputSample*, *outputSample*, *detection*, *noiseThres*, *saturationThres*, *boxCox*)

**Parameters** **inputSample** : 2-d sequence of float

Vector of the input values. The first column must correspond with the defect sizes.

**outputSample** : 2-d sequence of float

Vector of the signals, of dimension 1.

**detection** : float

Detection value of the signal.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**boxCox** : bool or float

Enable or not the Box Cox transformation. If boxCox is a float, the Box Cox transformation is enabled with the given value. Default is False.

**Warning:** The first column of the input sample must corresponds with the defects sample.

## Notes

This class aims at building the POD based on a kriging model. No assumptions are required for the residuals with this method. The POD are computed by simulating conditional prediction. For each, a Monte Carlo simulation is performed. The accuracy of the Monte Carlo simulation is taken into account using the TCL.

The return POD model corresponds with an interpolate function built with the POD values computed for the given defect sizes. The default values are 20 defect sizes between the minimum and maximum value of the defect sample. The defect sizes can be changed using the method *setDefectSizes*.

The default kriging model is built with a linear basis only for the defect size and constant otherwise. The covariance model is an anisotropic squared exponential model. Parameters are estimated using the TNC algorithm, the initial starting point of the TNC is found thanks to a quasi random search of the best loglikelihood value among 1000 computations.

For advanced use, all parameters can be defined thanks to dedicated set methods. Moreover, if the user has already built a kriging result, it can be given as parameter using the method *setKrigingResult*, then the POD is computed based on this kriging result.

A progress bar is shown if the verbosity is enabled. It can be disabled using the method *setVerbose*.

## Methods

<code>computeDetectionSize(*args, **kwargs)</code>	Compute the detection size for a given probability level.
<code>drawBoxCoxLikelihood([name])</code>	Draw the loglikelihood versus the Box Cox parameter.
<code>drawPOD(*args, **kwargs)</code>	Draw the POD curve.
<code>drawValidationGraph(*args, **kwargs)</code>	Draw the validation graph of the metamodel.
<code>getBasis()</code>	Accessor to the kriging basis.
<code>getBoxCoxParameter()</code>	Accessor to the Box Cox parameter.
<code>getCovarianceModel()</code>	Accessor to the kriging covariance model.
<code>getDefectSizes()</code>	Accessor to the defect size where POD is computed.
<code>getDistribution()</code>	Accessor to the parameters distribution.
<code>getInitialStartSize()</code>	Accessor to the initial random search size.
<code>getKrigingResult()</code>	Accessor to the kriging result.
<code>getPODCLModel([confidenceLevel])</code>	Accessor to the POD model at a given confidence level.
<code>getPODModel()</code>	Accessor to the POD model.
<code>getQ2()</code>	Accessor to the Q2 value.
<code>getSamplingSize()</code>	Accessor to the Monte Carlo sampling size.
<code>getSimulationSize()</code>	Accessor to the simulation size.
<code>getVerbose()</code>	Accessor to the verbosity.
<code>run()</code>	Build the POD models.
<code>setBasis(basis)</code>	Accessor to the kriging basis.
<code>setCovarianceModel(covarianceModel)</code>	Accessor to the kriging covariance model.
<code>setDefectSizes(size)</code>	Accessor to the defect size where POD is computed.
<code>setDistribution(distribution)</code>	Accessor to the parameters distribution.
<code>setInitialStartSize(size)</code>	Accessor to the initial random search size.
<code>setKrigingResult(result)</code>	Accessor to the kriging result.
<code>setSamplingSize(size)</code>	Accessor to the Monte Carlo sampling size.
<code>setSimulationSize(size)</code>	Accessor to the simulation size.
<code>setVerbose(verbose)</code>	Accessor to the verbosity.

**computeDetectionSize** (\*args, \*\*kwargs)

Compute the detection size for a given probability level.

**Parameters** `probabilityLevel` : float

The probability level for which the defect size is computed.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**Returns** `result` : collection of `openturns.NumericalPointWithDescription`

A `NumericalPointWithDescription` containing the detection size computed at the given probability level and confidence level if provided.

**drawBoxCoxLikelihood** (name=None)

Draw the loglikelihood versus the Box Cox parameter.

**Parameters** `name` : string

name of the figure to be saved with `transparent` option sets to True and `bbox_inches='tight'`. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

## Notes

This method is available only when the parameter *boxCox* is set to True.

**drawPOD** (\*args, \*\*kwargs)

Draw the POD curve.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**defectMin, defectMax** : float

Define the interval where the curve is plotted. Default : min and max values of the input sample.

**nbPt** : int

The number of points to draw the curves. Default is 100.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**drawValidationGraph** (\*args, \*\*kwargs)

Draw the validation graph of the metamodel.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**getBasis** ()

Accessor to the kriging basis.

**Returns** `basis` : `openturns.Basis`

The basis used as trend in the kriging model. Default is a linear basis for the defect and constant for the other parameters.

**getBoxCoxParameter** ()

Accessor to the Box Cox parameter.

**Returns** `lambdaBoxCox` : float

The Box Cox parameter used to transform the data. If the transformation is not enabled None is returned.

**getCovarianceModel** ()

Accessor to the kriging covariance model.

**Returns** `covarianceModel` : `openturns.CovarianceModel`

The covariance model in the kriging model. Default is an anisotropic squared exponential covariance model.

**getDefectSizes** ()

Accessor to the defect size where POD is computed.

**Returns** `defectSize` : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**getDistribution** ()

Accessor to the parameters distribution.

**Returns** `distribution` : `openturns.ComposedDistribution`

The input parameters distribution used for the Monte Carlo simulation. Default is a Uniform distribution for all parameters.

**getInitialStartSize** ()

Accessor to the initial random search size.

**Returns** `size` : int

The size of the initial random search to find the best loglikelihood value to start the TNC algorithm to optimize the covariance model parameters. Default is 1000.

**getKrigingResult** ()

Accessor to the kriging result.

**Returns** `result` : `openturns.KrigingResult`

The kriging result.

**getPODCLModel** (*confidenceLevel=0.95*)

Accessor to the POD model at a given confidence level.

**Parameters** `confidenceLevel` : float

The confidence level the POD must be computed. Default is 0.95

**Returns** `PODModelCI` : `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value at the confidence level given as parameter.

**getPODModel** ()

Accessor to the POD model.

**Returns** `PODModel` : `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value.

**getQ2 ()**

Accessor to the Q2 value.

**Returns Q2** : float

The Q2 value computed analytically using Dubrule (1983) technique.

**getSamplingSize ()**

Accessor to the Monte Carlo sampling size.

**Returns size** : int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**getSimulationSize ()**

Accessor to the simulation size.

**Returns size** : int

The size of the simulation used to compute the confidence interval.

**getVerbose ()**

Accessor to the verbosity.

**Returns verbose** : bool

Enable or disable the verbosity. Default is True.

**run ()**

Build the POD models.

## Notes

This method build the kriging model. First the censored data are filtered if needed. The Box Cox transformation is performed if it is enabled. Then it builds the POD models : conditional samples are simulated for each defect size, then the distributions of the probability estimator (for MC simulation) are built. Eventually, a sample of this distribution is used to compute the mean POD and the POD at the confidence level.

**setBasis (basis)**

Accessor to the kriging basis.

**Parameters basis** : `openturns.Basis`

The basis used as trend in the kriging model.

**setCovarianceModel (covarianceModel)**

Accessor to the kriging covariance model.

**Parameters covarianceModel** : `openturns.CovarianceModel`

The covariance model in the kriging model.

**setDefectSizes (size)**

Accessor to the defect size where POD is computed.

**Parameters defectSize** : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**setDistribution (distribution)**

Accessor to the parameters distribution.



**Parameters** `distribution` : `openturns.ComposedDistribution`

The input parameters distribution used for the Monte Carlo simulation.

**setInitialStartSize** (*size*)

Accessor to the initial random search size.

**Parameters** `size` : int

The size of the initial random search to find the best loglikelihood value to start the TNC algorithm to optimize the covariance model parameters.

**setKrigingResult** (*result*)

Accessor to the kriging result.

**Parameters** `result` : `openturns.KrigingResult`

The kriging result.

**setSamplingSize** (*size*)

Accessor to the Monte Carlo sampling size.

**Parameters** `size` : int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**setSimulationSize** (*size*)

Accessor to the simulation size.

**Parameters** `size` : int

The size of the simulation used to compute the confidence interval.

**setVerbose** (*verbose*)

Accessor to the verbosity.

**Parameters** `verbose` : bool

Enable or disable the verbosity.

### 1.1.3 Adaptive algorithms

<code>AdaptiveSignalPOD</code>	Adaptive algorithm for signal data type.
<code>AdaptiveHitMissPOD</code>	Adaptive algorithm for hit miss data type.

#### AdaptiveSignalPOD

**class** `AdaptiveSignalPOD` (*\*args*)

Adaptive algorithm for signal data type.

**Available constructor:**

`AdaptiveSignalPOD(inputDOE, outputDOE, physicalModel, nMorePoints, detection, noiseThres, saturationThres, boxCox)`

**Parameters** `inputDOE` : 2-d sequence of float

Vector of the input values. The first column must correspond with the defect sizes.

**outputDOE** : 2-d sequence of float

Vector of the signals, of dimension 1.

**physicalModel** : `NumericalMathFunction`

True model used to compute the real signal value to be added to the DOE.

**nMorePoints** : positive int

The number of points to add to the DOE, computed by the *physicalModel*.

**detection** : float

Detection value of the signal.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**boxCox** : bool or float

Enable or not the Box Cox transformation. If boxCox is a float, the Box Cox transformation is enabled with the given value. Default is False.

**Warning:** The first column of the input sample must corresponds with the defect sizes.

## Notes

This class aims at building the POD based on a kriging model where the design of experiments is iteratively enriched. The initial design of experiments is given as input parameters. The enrichment criterion is based on the integrated mean squared of the POD. The criterion is computed on several candidate points and the one that minimizes the criterion is added to the current design of experiments. The sample of candidate points is created using a low discrepancy sequence (Sobol') if the input distribution has an independant copula, otherwise a Monte Carlo experiment is used. This is a time consuming technique because it requires to compute the mean and variance of the POD for all candidate points. The stopping criterion is only based on the number of points that must be added to the design of experiments.

No assumptions are required for the residuals with this method. The POD are computed by simulating conditional predictions. For each, a Monte Carlo simulation is performed. The accuracy of the Monte Carlo simulation is taken into account using the TCL.

The return POD model corresponds with an interpolate function built with the POD values computed for the given defect sizes. The default values are 20 defect sizes between the minimum and maximum value of the defect sample. The defect sizes can be changed using the method *setDefectSizes*. It is advised to run a preliminary POD study in order to know the interesting range of defect sizes. This enables reducing the computing time.

The default kriging model is built with a linear basis only for the defect size and constant otherwise. The covariance model is an anisotropic squared exponential model. Parameters are estimated using the TNC algorithm, the initial starting point of the TNC is found thanks to a quasi random search of the best loglikelihood value among 1000 computations.

In the algorithm, when a point is added to the design of experiments, the kriging model is not always optimized. The covariance model scale coefficients are optimized only if the Q2 value is lower than 0.95.

For advanced use, all parameters can be defined thanks to dedicated set methods.

A progress bar is shown if the verbosity is enabled. It can be disabled using the method *setVerbose*.

## Methods

<code>computeDetectionSize(*args, **kwargs)</code>	Compute the detection size for a given probability level.
<code>drawBoxCoxLikelihood([name])</code>	Draw the loglikelihood versus the Box Cox parameter.
<code>drawPOD(*args, **kwargs)</code>	Draw the POD curve.
<code>drawValidationGraph(*args, **kwargs)</code>	Draw the validation graph of the metamodel.
<code>getBasis()</code>	Accessor to the kriging basis.
<code>getBoxCoxParameter()</code>	Accessor to the Box Cox parameter.
<code>getCandidateSize()</code>	Accessor to the number of candidate points.
<code>getCovarianceModel()</code>	Accessor to the kriging covariance model.
<code>getDefectSizes()</code>	Accessor to the defect size where POD is computed.
<code>getDistribution()</code>	Accessor to the parameters distribution.
<code>getGraphActive()</code>	Accessor to the graph verbosity.
<code>getInitialStartSize()</code>	Accessor to the initial random search size.
<code>getInputDOE()</code>	Accessor to the final input values of the DOE.
<code>getKrigingResult()</code>	Accessor to the kriging result.
<code>getOutputDOE()</code>	Accessor to the final output values of the DOE.
<code>getPODCLModel([confidenceLevel])</code>	Accessor to the POD model at a given confidence level.
<code>getPODModel()</code>	Accessor to the POD model.
<code>getQ2()</code>	Accessor to the Q2 value.
<code>getSamplingSize()</code>	Accessor to the Monte Carlo sampling size.
<code>getSimulationSize()</code>	Accessor to the simulation size.
<code>getVerbose()</code>	Accessor to the verbosity.
<code>run()</code>	Launch the algorithm and build the POD models.
<code>setBasis(basis)</code>	Accessor to the kriging basis.
<code>setCandidateSize(size)</code>	Accessor to the number of candidate points.
<code>setCovarianceModel(covarianceModel)</code>	Accessor to the kriging covariance model.
<code>setDefectSizes(size)</code>	Accessor to the defect size where POD is computed.
<code>setDistribution(distribution)</code>	Accessor to the parameters distribution.
<code>setGraphActive(graphVerbose[, ...])</code>	Accessor to the graph verbosity.
<code>setInitialStartSize(size)</code>	Accessor to the initial random search size.
<code>setSamplingSize(size)</code>	Accessor to the Monte Carlo sampling size.
<code>setSimulationSize(size)</code>	Accessor to the simulation size.
<code>setVerbose(verbose)</code>	Accessor to the verbosity.

**computeDetectionSize** (*\*args, \*\*kwargs*)

Compute the detection size for a given probability level.

**Parameters** `probabilityLevel` : float

The probability level for which the defect size is computed.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**Returns** `result` : collection of `openturns.NumericalPointWithDescription`

A `NumericalPointWithDescription` containing the detection size computed at the given probability level and confidence level if provided.

**drawBoxCoxLikelihood** (*name=None*)

Draw the loglikelihood versus the Box Cox parameter.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : [matplotlib.figure](#)

Matplotlib figure object.

**ax** : [matplotlib.axes](#)

Matplotlib axes object.

## Notes

This method is available only when the parameter *boxCox* is set to True.

**drawPOD** (\*args, \*\*kwargs)

Draw the POD curve.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**defectMin, defectMax** : float

Define the interval where the curve is plotted. Default : min and max values of the input sample.

**nbPt** : int

The number of points to draw the curves. Default is 100.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : [matplotlib.figure](#)

Matplotlib figure object.

**ax** : [matplotlib.axes](#)

Matplotlib axes object.

**drawValidationGraph** (\*args, \*\*kwargs)

Draw the validation graph of the metamodel.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : [matplotlib.figure](#)

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**getBasis()**

Accessor to the kriging basis.

**Returns** `basis` : `openturns.Basis`

The basis used as trend in the kriging model. Default is a linear basis for the defect and constant for the other parameters.

**getBoxCoxParameter()**

Accessor to the Box Cox parameter.

**Returns** `lambdaBoxCox` : float

The Box Cox parameter used to transform the data. If the transformation is not enabled None is returned.

**getCandidateSize()**

Accessor to the number of candidate points.

**Returns** `size` : int

The number of candidate points on which the criterion is computed.

**getCovarianceModel()**

Accessor to the kriging covariance model.

**Returns** `covarianceModel` : `openturns.CovarianceModel`

The covariance model in the kriging model. Default is an anisotropic squared exponential covariance model.

**getDefectSizes()**

Accessor to the defect size where POD is computed.

**Returns** `defectSize` : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**getDistribution()**

Accessor to the parameters distribution.

**Returns** `distribution` : `openturns.ComposedDistribution`

The input parameters distribution used for the Monte Carlo simulation. Default is a Uniform distribution for all parameters.

**getGraphActive()**

Accessor to the graph verbosity.

**Returns** `graphVerbose` : bool

Enable or disable the display of the POD graph at each iteration. Default is False.

**getInitialStartSize()**

Accessor to the initial random search size.

**Returns** `size` : int

The size of the initial random search to find the best loglikelihood value to start the TNC algorithm to optimize the covariance model parameters. Default is 1000.

**getInputDOE()**

Accessor to the final input values of the DOE.

**getKrigingResult ()**

Accessor to the kriging result.

**Returns result :** `openturns.KrigingResult`

The kriging result.

**getOutputDOE ()**

Accessor to the final output values of the DOE.

**getPODCLModel (confidenceLevel=0.95)**

Accessor to the POD model at a given confidence level.

**Parameters confidenceLevel :** float

The confidence level the POD must be computed. Default is 0.95

**Returns PODModelCI :** `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value at the confidence level given as parameter.

**getPODModel ()**

Accessor to the POD model.

**Returns PODModel :** `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value.

**getQ2 ()**

Accessor to the Q2 value.

**Returns Q2 :** float

The Q2 value computed analytically using Dubrule (1983) technique.

**getSamplingSize ()**

Accessor to the Monte Carlo sampling size.

**Returns size :** int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**getSimulationSize ()**

Accessor to the simulation size.

**Returns size :** int

The size of the simulation used to compute the confidence interval.

**getVerbose ()**

Accessor to the verbosity.

**Returns verbose :** bool

Enable or disable the verbosity. Default is True.

**run ()**

Launch the algorithm and build the POD models.

## Notes

This method launches the iterative algorithm. First the censored data are filtered if needed. The Box Cox transformation is performed if it is enabled. Then the enrichment of the design of experiments is performed. Once the algorithm stops, it builds the POD models : conditional samples are simulated

for each defect size, then the distributions of the probability estimator (for MC simulation) are built. Eventually, a sample of this distribution is used to compute the mean POD and the POD at the confidence level.

**setBasis** (*basis*)

Accessor to the kriging basis.

**Parameters** **basis** : `openturns.Basis`

The basis used as trend in the kriging model.

**setCandidateSize** (*size*)

Accessor to the number of candidate points.

**Parameters** **size** : int

The number of candidate points on which the criterion is computed

**setCovarianceModel** (*covarianceModel*)

Accessor to the kriging covariance model.

**Parameters** **covarianceModel** : `openturns.CovarianceModel`

The covariance model in the kriging model.

**setDefectSizes** (*size*)

Accessor to the defect size where POD is computed.

**Parameters** **defectSize** : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**setDistribution** (*distribution*)

Accessor to the parameters distribution.

**Parameters** **distribution** : `openturns.ComposedDistribution`

The input parameters distribution used for the Monte Carlo simulation.

**setGraphActive** (*graphVerbose, probabilityLevel=None, confidenceLevel=None, directory=None*)

Accessor to the graph verbosity.

**Parameters** **graphVerbose** : bool

Enable or disable the display of the POD graph at each iteration.

**probabilityLevel** : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**directory** : string

Directory where to save the graphs as png files.

**setInitialStartSize** (*size*)

Accessor to the initial random search size.

**Parameters** **size** : int

The size of the initial random search to find the best loglikelihood value to start the TNC algorithm to optimize the covariance model parameters.

**setSamplingSize** (*size*)

Accessor to the Monte Carlo sampling size.

**Parameters** *size* : int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**setSimulationSize** (*size*)

Accessor to the simulation size.

**Parameters** *size* : int

The size of the simulation used to compute the confidence interval.

**setVerbose** (*verbose*)

Accessor to the verbosity.

**Parameters** *verbose* : bool

Enable or disable the verbosity.

## AdaptiveHitMissPOD

**class AdaptiveHitMissPOD** (*\*args*)

Adaptive algorithm for hit miss data type.

**Available constructor:**

**AdaptiveHitMissPOD**(*inputDOE*, *outputDOE*, *physicalModel*, *nMorePoints*, *detection*, *noiseThres*, *saturationThres*)

**Parameters** *inputDOE* : 2-d sequence of float

Vector of the input values. The first column must correspond with the defect sizes.

**outputDOE** : 2-d sequence of float

Vector of the signals, of dimension 1.

**physicalModel** : `NumericalMathFunction`

True model used to compute the real hit miss value of the signal value to be added to the DOE.

**nMorePoints** : positive int

The number of points to add to the DOE, computed by the *physicalModel*.

**detection** : float

Detection value of the signal if the physical model does not return a hit miss value.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**Warning:** The first column of the input sample must corresponds with the defect sizes.



## Notes

This class aims at building the POD based on a classifier model where the design of experiments is iteratively enriched. The initial design of experiments is given as input parameters. The enrichment criterion is based on the misclassification empirical risk. The criterion is computed on several candidate points. The sample of candidate points is created using a low discrepancy sequence (Sobol') if the input distribution has an independant copula, otherwise a Monte Carlo experiment is used. The stopping criterion is only based on the number of points that must be added to the design of experiments.

The classifier algorithms availables are the SVC and the random forests. The choice of the algorithm can be defined using *setClassifierType*. The default algorithm is the random forests.

The physical model can return either the hit miss value (0 or 1) or the signal value. In this case, the detection value must be given and the physical model is transformed in order to provide a hit miss value.

The POD are computed by a Monte Carlo simulation for several defect values. The accuracy of the Monte Carlo simulation is taken into account using the TCL. The return POD model corresponds with an interpolate function built with the POD values computed for the given defect sizes. The default values are 20 defect sizes between the minimum and maximum value of the defect sample. The defect sizes can be changed using the method *setDefectSizes*.

A progress bar is shown if the verbosity is enabled. It can be disabled using the method *setVerbose*.

## Methods

<i>computeDetectionSize(*args, **kwargs)</i>	Compute the detection size for a given probability level.
<i>drawBoxCoxLikelihood([name])</i>	Draw the loglikelihood versus the Box Cox parameter.
<i>drawPOD(*args, **kwargs)</i>	Draw the POD curve.
<i>getBoxCoxParameter()</i>	Accessor to the Box Cox parameter.
<i>getCandidateSize()</i>	Accessor to the number of candidate points.
<i>getClassifier()</i>	Accessor to the classifier model.
<i>getClassifierParameters()</i>	Accessor to the classifier parameters.
<i>getClassifierType()</i>	Accessor to the classifier type.
<i>getConfusionMatrix()</i>	Accessor to the confusion matrix.
<i>getDefectSizes()</i>	Accessor to the defect size where POD is computed.
<i>getDistribution()</i>	Accessor to the parameters distribution.
<i>getGraphActive()</i>	Accessor to the graph verbosity.
<i>getInputDOE()</i>	Accessor to the final input values of the DOE.
<i>getOutputDOE()</i>	Accessor to the final output values of the DOE.
<i>getPMax()</i>	Accessor to the upper probability bound for the point selections.
<i>getPMin()</i>	Accessor to the lower probability bound for the point selections.
<i>getPODCLModel([confidenceLevel])</i>	Accessor to the POD model at a given confidence level.
<i>getPODModel()</i>	Accessor to the POD model.
<i>getSamplingSize()</i>	Accessor to the Monte Carlo sampling size.
<i>getSimulationSize()</i>	Accessor to the simulation size.
<i>getVerbose()</i>	Accessor to the verbosity.
<i>run()</i>	Launch the algorithm and build the POD models.
<i>setCandidateSize(size)</i>	Accessor to the number of candidate points.
<i>setClassifierParameters(parameters)</i>	Accessor to the classifier parameters.
<i>setClassifierType(classifier)</i>	Accessor to the classifier type.

Continued on next page

Table 1.10 – continued from previous page

<code>setDefectSizes(size)</code>	Accessor to the defect size where POD is computed.
<code>setDistribution(distribution)</code>	Accessor to the parameters distribution.
<code>setGraphActive(graphVerbose[, ...])</code>	Accessor to the graph verbosity.
<code>setPMax(pmax)</code>	Accessor to the upper probability bound for the point selections.
<code>setPMin(pmin)</code>	Accessor to the lower probability bound for the point selections.
<code>setSamplingSize(size)</code>	Accessor to the Monte Carlo sampling size.
<code>setSimulationSize(size)</code>	Accessor to the simulation size.
<code>setVerbose(verbose)</code>	Accessor to the verbosity.

**computeDetectionSize** (\*args, \*\*kwargs)

Compute the detection size for a given probability level.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**Returns** **result** : collection of `openturns.NumericalPointWithDescription`

A `NumericalPointWithDescription` containing the detection size computed at the given probability level and confidence level if provided.

**drawBoxCoxLikelihood** (name=None)

Draw the loglikelihood versus the Box Cox parameter.

**Parameters** **name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

## Notes

This method is available only when the parameter *boxCox* is set to True.

**drawPOD** (\*args, \*\*kwargs)

Draw the POD curve.

**Parameters** **probabilityLevel** : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**defectMin, defectMax** : float

Define the interval where the curve is plotted. Default : min and max values of the input sample.

**nbPt** : int

The number of points to draw the curves. Default is 100.

**name** : string

name of the figure to be saved with *transparent* option sets to True and *bbox\_inches='tight'*. It can be only the file name or the full path name. Default is None.

**Returns** **fig** : `matplotlib.figure`

Matplotlib figure object.

**ax** : `matplotlib.axes`

Matplotlib axes object.

**getBoxCoxParameter** ()

Accessor to the Box Cox parameter.

**Returns** **lambdaBoxCox** : float

The Box Cox parameter used to transform the data. If the transformation is not enabled None is returned.

**getCandidateSize** ()

Accessor to the number of candidate points.

**Returns** **size** : int

The number of candidate points on which the criterion is computed.

**getClassifier** ()

Accessor to the classifier model.

**Returns** **result** : classifier

The classifier model, either random forest or svm.

**getClassifierParameters** ()

Accessor to the classifier parameters.

**getClassifierType** ()

Accessor to the classifier type.

**getConfusionMatrix** ()

Accessor to the confusion matrix.

**getDefectSizes** ()

Accessor to the defect size where POD is computed.

**Returns** **defectSize** : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**getDistribution** ()

Accessor to the parameters distribution.

**Returns** **distribution** : `openturns.ComposedDistribution`

The input parameters distribution, default is a Uniform distribution for all parameters.

**getGraphActive()**

Accessor to the graph verbosity.

**Returns graphVerbose** : bool

Enable or disable the display of the POD graph at each iteration. Default is False.

**getInputDOE()**

Accessor to the final input values of the DOE.

**getOutputDOE()**

Accessor to the final output values of the DOE.

**getPMax()**

Accessor to the upper probability bound for the point selections.

**getPMin()**

Accessor to the lower probability bound for the point selections.

**getPODCLModel** (*confidenceLevel=0.95*)

Accessor to the POD model at a given confidence level.

**Parameters confidenceLevel** : float

The confidence level the POD must be computed. Default is 0.95

**Returns PODModelCI** : `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value at the confidence level given as parameter.

**getPODModel()**

Accessor to the POD model.

**Returns PODModel** : `openturns.NumericalMathFunction`

The function which computes the probability of detection for a given defect value.

**getSamplingSize()**

Accessor to the Monte Carlo sampling size.

**Returns size** : int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**getSimulationSize()**

Accessor to the simulation size.

**Returns size** : int

The size of the simulation used to compute the confidence interval.

**getVerbose()**

Accessor to the verbosity.

**Returns verbose** : bool

Enable or disable the verbosity. Default is True.

**run()**

Launch the algorithm and build the POD models.

## Notes

This method launches the iterative algorithm. Once the algorithm stops, it builds the POD models : Monte Carlo simulation are performed for each defect sizes with the final classifier model. Eventually, the sample is used to compute the mean POD and the POD at the confidence level.

**setCandidateSize** (*size*)

Accessor to the number of candidate points.

**Parameters** **size** : int

The number of candidate points on which the criterion is computed

**setClassifierParameters** (*parameters*)

Accessor to the classifier parameters.

**setClassifierType** (*classifier*)

Accessor to the classifier type.

**setDefectSizes** (*size*)

Accessor to the defect size where POD is computed.

**Parameters** **defectSize** : sequence of float

The defect sizes where the Monte Carlo simulation is performed to compute the POD.

**setDistribution** (*distribution*)

Accessor to the parameters distribution.

**Parameters** **distribution** : `openturns.ComposedDistribution`

The input parameters distribution.

**setGraphActive** (*graphVerbose*, *probabilityLevel=None*, *confidenceLevel=None*, *directory=None*)

Accessor to the graph verbosity.

**Parameters** **graphVerbose** : bool

Enable or disable the display of the POD graph at each iteration.

**probabilityLevel** : float

The probability level for which the defect size is computed. Default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**directory** : string

Directory where to save the graphs as png files.

**setPMax** (*pmax*)

Accessor to the upper probability bound for the point selections.

**setPMin** (*pmin*)

Accessor to the lower probability bound for the point selections.

**setSamplingSize** (*size*)

Accessor to the Monte Carlo sampling size.

**Parameters** **size** : int

The size of the Monte Carlo simulation used to compute the POD for each defect size.

**setSimulationSize** (*size*)

Accessor to the simulation size.

**Parameters** **size** : int

The size of the simulation used to compute the confidence interval.

**setVerbose** (*verbose*)

Accessor to the verbosity.

**Parameters** **verbose** : bool

Enable or disable the verbosity.

## 1.1.4 Tools

---

<i>PODSummary</i>	Run the analysis and compute POD with several methods.
<i>DataHandling</i>	Static methods for data handling.

---

### PODSummary

**class** **PODSummary** (*\*args*)

Run the analysis and compute POD with several methods.

**Available constructor:**

**PODSummary**(*inputSample, outputSample, detection, noiseThres, saturationThres, boxCox*)

**Parameters** **inputSample** : 2-d sequence of float

Vector of the input values. The first column must correspond with the defect sizes.

**outputSample** : 2-d sequence of float

Vector of the signals, of dimension 1.

**detection** : float

Detection value of the signal.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**boxCox** : bool or float

Enable or not the Box Cox transformation. If boxCox is a float, the Box Cox transformation is enabled with the given value. Default is False.

**Warning:** The first column of the input sample must corresponds with the defects sample.

### Notes

This class aims at running the linear analysis and computing the POD with different models:

- Linear regression model with Gaussian residuals hypothesis,
- Linear regression model with no hypothesis on the residuals (binomial),
- Linear regression model with with kernel smoothing on the residuals,
- Quantile regression,
- Polynomial chaos,
- kriging if the dimension of the input sample is greater than 1.

Each method can be deactivated using the method *setMethodActive* and using the key corresponding to the method.

All results can be displayed and saved thanks to the methods *printResults*, *saveResults* and *saveGraphs*. For each method, the probability level and confidence level can be specified in order to compute the defect size to the wanted probability level.

The verbosity is enabled by default but it can be disabled using the method *setVerbose*.

## Methods

<i>drawGraphs</i> ([directory, extension, ...])	draw and save all possible graphs
<i>getKrigingPOD</i> ()	Accessor to the kriging POD object.
<i>getLinearBinomialPOD</i> ()	Accessor to the linear model POD object with no hypothesis on the residuals.
<i>getLinearGaussPOD</i> ()	Accessor to the linear model POD object with Gaussian hypothesis.
<i>getLinearKernelSmoothingPOD</i> ()	Accessor to the linear model POD object with kernel smoothing on the residuals.
<i>getMethodActive</i> ()	Accessor to the dictionary of active methods.
<i>getPolynomialChaosPOD</i> ()	Accessor to the polynomial chaos POD object.
<i>getQuantileRegressionPOD</i> ()	Accessor to the quantile regression POD object.
<i>getSamplingSize</i> ()	Accessor to the Monte Carlo sampling size.
<i>getSimulationSize</i> ()	Accessor to the simulation size.
<i>getVerbose</i> ()	Accessor to the verbosity.
<i>printResults</i> ([probabilityLevel, confidenceLevel])	Print all results in the terminal.
<i>run</i> ()	Run all active methods.
<i>saveResults</i> (name[, probabilityLevel, ...])	Save all analysis test results in a file.
<i>setMethodActive</i> (method, activation)	Accessor to the dictionary of active methods.
<i>setSamplingSize</i> (size)	Accessor to the Monte Carlo sampling size.
<i>setSimulationSize</i> (size)	Accessor to the simulation size.
<i>setVerbose</i> (verbose)	Accessor to the verbosity.

**drawGraphs** (*directory*=None, *extension*='png', *probabilityLevel*=None, *confidenceLevel*=None)  
draw and save all possible graphs

**Parameters** *directory* : string

Directory where to save the graphs. Default is the working directory.

**extension** : string

File extension of the graphs. Default is 'png'.

**probabilityLevel** : float

The probability level for which the defect size is computed. default is None.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is None.

**getKrigingPOD** ()

Accessor to the kriging POD object.

**Returns algorithm** : KrigingPOD

The KrigingPOD object that is used to compute the POD.

**getLinearBinomialPOD** ()

Accessor to the linear model POD object with no hypothesis on the residuals.

**Returns algorithm** : UnivariateLinearModelPOD

The UnivariateLinearModelPOD object that is used to compute the POD.

**getLinearGaussPOD** ()

Accessor to the linear model POD object with Gaussian hypothesis.

**Returns algorithm** : UnivariateLinearModelPOD

The UnivariateLinearModelPOD object that is used to compute the POD.

**getLinearKernelSmoothingPOD** ()

Accessor to the linear model POD object with kernel smoothing on the residuals.

**Returns algorithm** : UnivariateLinearModelPOD

The UnivariateLinearModelPOD object that is used to compute the POD.

**getMethodActive** ()

Accessor to the dictionary of active methods.

**Returns activeDict** : dict

The dictionary containing the bool telling if the methods is activated or not.

**getPolynomialChaosPOD** ()

Accessor to the polynomial chaos POD object.

**Returns algorithm** : PolynomialChaosPOD

The PolynomialChaosPOD object that is used to compute the POD.

**getQuantileRegressionPOD** ()

Accessor to the quantile regression POD object.

**Returns algorithm** : QuantileRegressionPOD

The QuantileRegressionPOD object that is used to compute the POD.

**getSamplingSize** ()

Accessor to the Monte Carlo sampling size.

**Returns size** : int

The size of the Monte Carlo simulation used to compute the POD for each defect size for polynomial chaos and kriging.

**getSimulationSize** ()

Accessor to the simulation size.

**Returns size** : int



The size of the simulation used to compute the confidence interval.

**getVerbose()**

Accessor to the verbosity.

**Returns** `verbose` : bool

Enable or disable the verbosity. Default is True.

**printResults** (*probabilityLevel=0.9, confidenceLevel=0.95*)

Print all results in the terminal.

**Parameters** `probabilityLevel` : float

The probability level for which the defect size is computed. default is 0.9.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is 0.95.

## Notes

The probability level and confidence level can be specified in order to display the defect size for different probability level.

**run()**

Run all active methods.

**saveResults** (*name, probabilityLevel=0.9, confidenceLevel=0.95*)

Save all analysis test results in a file.

**Parameters** `name` : string

Name of the file or full path name.

**probabilityLevel** : float

The probability level for which the defect size is computed. default is 0.9.

**confidenceLevel** : float

The confidence level associated to the given probability level the defect size is computed. Default is 0.95.

## Notes

The probability level and confidence level can be specified in order to display the defect size for different probability level.

The file can be saved as a csv file. Separations are made with tabulations.

If *name* is the file name, then it is saved in the current working directory.

**setMethodActive** (*method, activation*)

Accessor to the dictionary of active methods.

**Parameters** `method` : string

The key of the method to activate or deactivate.

**activation** : bool

Set to True to activate and False to deactivate.

**setSamplingSize** (*size*)

Accessor to the Monte Carlo sampling size.

**Parameters** *size* : int

The size of the Monte Carlo simulation used to compute the POD for each defect size for polynomial chaos and kriging.

**setSimulationSize** (*size*)

Accessor to the simulation size.

**Parameters** *size* : int

The size of the simulation used to compute the confidence interval.

**setVerbose** (*verbose*)

Accessor to the verbosity.

**Parameters** *verbose* : bool

Enable or disable the verbosity.

## DataHandling

**class DataHandling**

Static methods for data handling.

## Methods

---

<i>filterCensoredData</i> (inputSample, signals, ...)	Sort inputSample and signals with respect to the censure thresholds.
---	--

---

**static filterCensoredData** (*inputSample, signals, noiseThres, saturationThres*)

Sort inputSample and signals with respect to the censure thresholds.

**Parameters** *inputSample* : 2-d sequence of float

Vector of the input sample.

**signals** : 2-d sequence of float

Vector of the signals, of dimension 1.

**noiseThres** : float

Value for low censored data. Default is None.

**saturationThres** : float

Value for high censored data. Default is None

**Returns** *inputSampleUnc* : 2-d sequence of float

Vector of the input sample in the uncensored area.

**inputSampleNoise** : 2-d sequence of float

Vector of the input sample in the noisy area.

**inputSampleSat** : 2-d sequence of float

Vector of the input sample in the saturation area.

**signalsUnc** : 2-d sequence of float

Vector of the signals in the uncensored area.

### Notes

The data are sorted in three different vectors whether they belong to the noisy area, the uncensored area or the saturation area.

## 1.2 Examples of the API

ipynb source code

### 1.2.1 Linear model analysis

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
```

#### Generate data

```
N = 100
ot.RandomGenerator.SetSeed(123456)
defectDist = ot.Uniform(0.1, 0.6)
# normal epsilon distribution
epsilon = ot.Normal(0, 1.9)
defects = defectDist.getSample(N)
signalsInvBoxCox = defects * 43. + epsilon.getSample(N) + 2.5
# Inverse Box Cox transformation
invBoxCox = ot.InverseBoxCoxTransform(0.3)
signals = invBoxCox(signalsInvBoxCox)
```

#### Run analysis without Box Cox

```
analysis = otpod.UnivariateLinearModelAnalysis(defects, signals)
```

#### Get some particular results

```
print analysis.getIntercept()
print analysis.getR2()
print analysis.getKolmogorovPValue()
```

```
[Intercept for uncensored case : -604.758]
[R2 for uncensored case : 0.780469]
[Kolmogorov p-value for uncensored case : 0.803087]
```

## Print all results of the linear regression and all tests on the residuals

A warning is printed because some residuals tests failed : the p-value is less than 0.5.

```
analysis.printResults()
```

```
WARNING:root:Some hypothesis tests failed : you may consider to use the Box Cox
↪ transformation.
```

```
-----
Linear model analysis results
-----
Box Cox parameter :                               Not enabled

                                                Uncensored

Intercept coefficient :                          -604.76
Slope coefficient :                               3606.04
Standard error of the estimate :                  291.47

Confidence interval on coefficients
Intercept coefficient :                          [-755.60, -453.91]
Slope coefficient :                               [3222.66, 3989.43]
Level :                                           0.95

Quality of regression
R2 (> 0.8):                                     0.78
-----

Residuals analysis results
-----
Fitted distribution (uncensored) :                Normal(mu = 5.95719e-13, sigma = 289.
↪ 998)

                                                Uncensored

Distribution fitting test
Kolmogorov p-value (> 0.05):                     0.8

Normality test
Anderson Darling p-value (> 0.05):                0.07
Cramer Von Mises p-value (> 0.05):                0.09

Zero residual mean test
p-value (> 0.05):                                1.0

Homoskedasticity test (constant variance)
Breush Pagan p-value (> 0.05):                    0.0
Harrison McCabe p-value (> 0.05):                 0.2

Non autocorrelation test
```

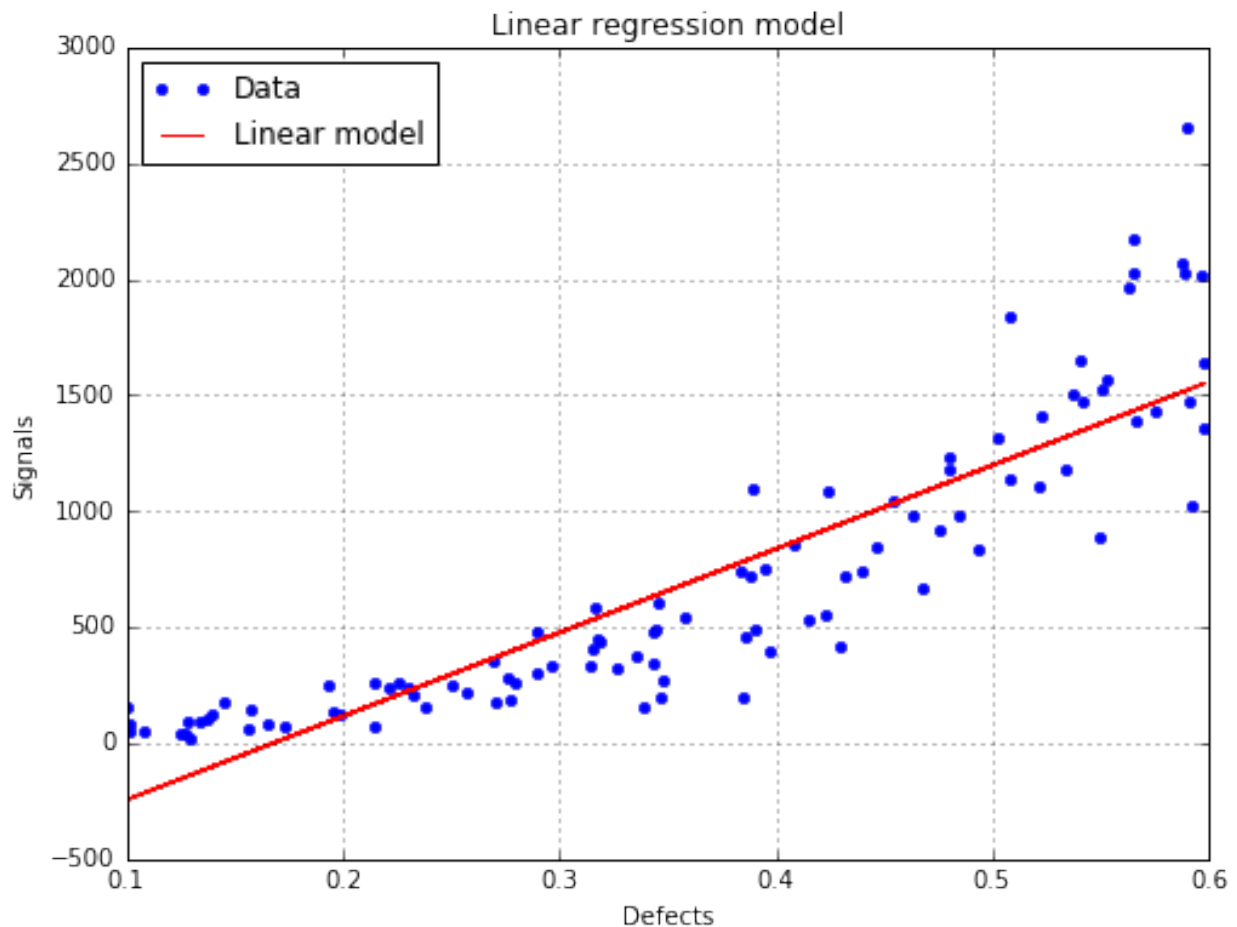
```
Durbin Watson p-value (> 0.05): 0.99
```

## Show graphs

### The linear model is not correct

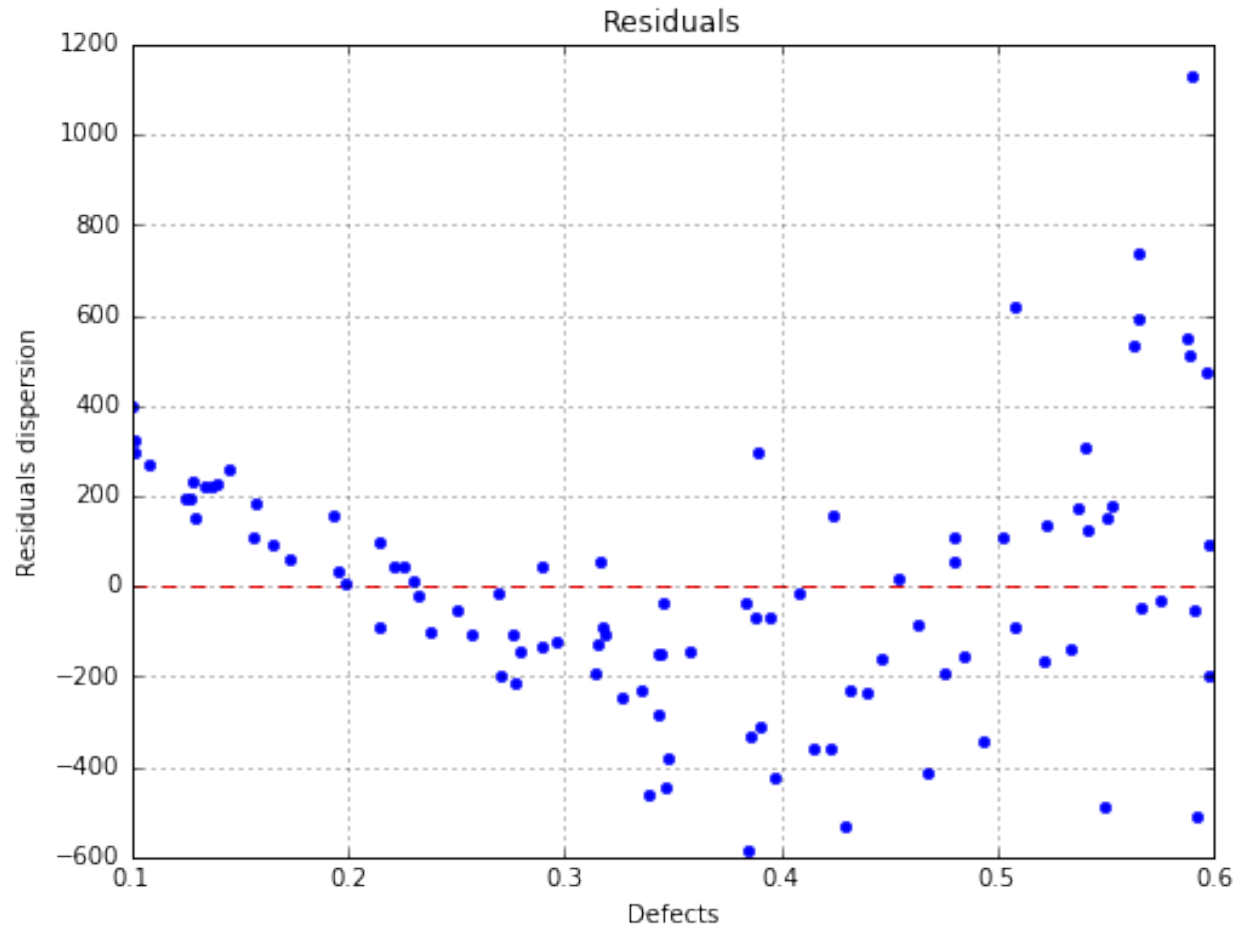
```
fig, ax = analysis.drawLinearModel()
fig.show()
```

```
/home/dumas/anaconda2/lib/python2.7/site-packages/matplotlib/figure.py:397:
↳UserWarning: matplotlib is currently using a non-GUI backend, so cannot show the
↳figure
  "matplotlib is currently using a non-GUI backend, "
```



### The residuals are not homoskedastic

```
fig, ax = analysis.drawResiduals()
fig.show()
```



### Run analysis with Box Cox

```
analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, boxCox=True)
```

### Print results of the linear regression and all tests on the residuals

```
analysis.printResults()
```

```
-----
Linear model analysis results
-----
Box Cox parameter :                                0.22
                                                    Uncensored
Intercept coefficient :                             4.02
Slope coefficient :                                  25.55
Standard error of the estimate :                     1.34

Confidence interval on coefficients
Intercept coefficient :                             [3.33, 4.72]
```

```

Slope coefficient : [23.80, 27.31]
Level : 0.95

Quality of regression
R2 (> 0.8): 0.89
-----

Residuals analysis results
-----

Fitted distribution (uncensored) : Normal(mu = 1.45661e-15, sigma = 1.
↪32901)

                                Uncensored
Distribution fitting test
Kolmogorov p-value (> 0.05): 0.34

Normality test
Anderson Darling p-value (> 0.05): 0.06
Cramer Von Mises p-value (> 0.05): 0.07

Zero residual mean test
p-value (> 0.05): 1.0

Homoskedasticity test (constant variance)
Breush Pagan p-value (> 0.05): 0.65
Harrison McCabe p-value (> 0.05): 0.51

Non autocorrelation test
Durbin Watson p-value (> 0.05): 0.97
-----

```

### Save all results in a csv file

```
analysis.saveResults('results.csv')
```

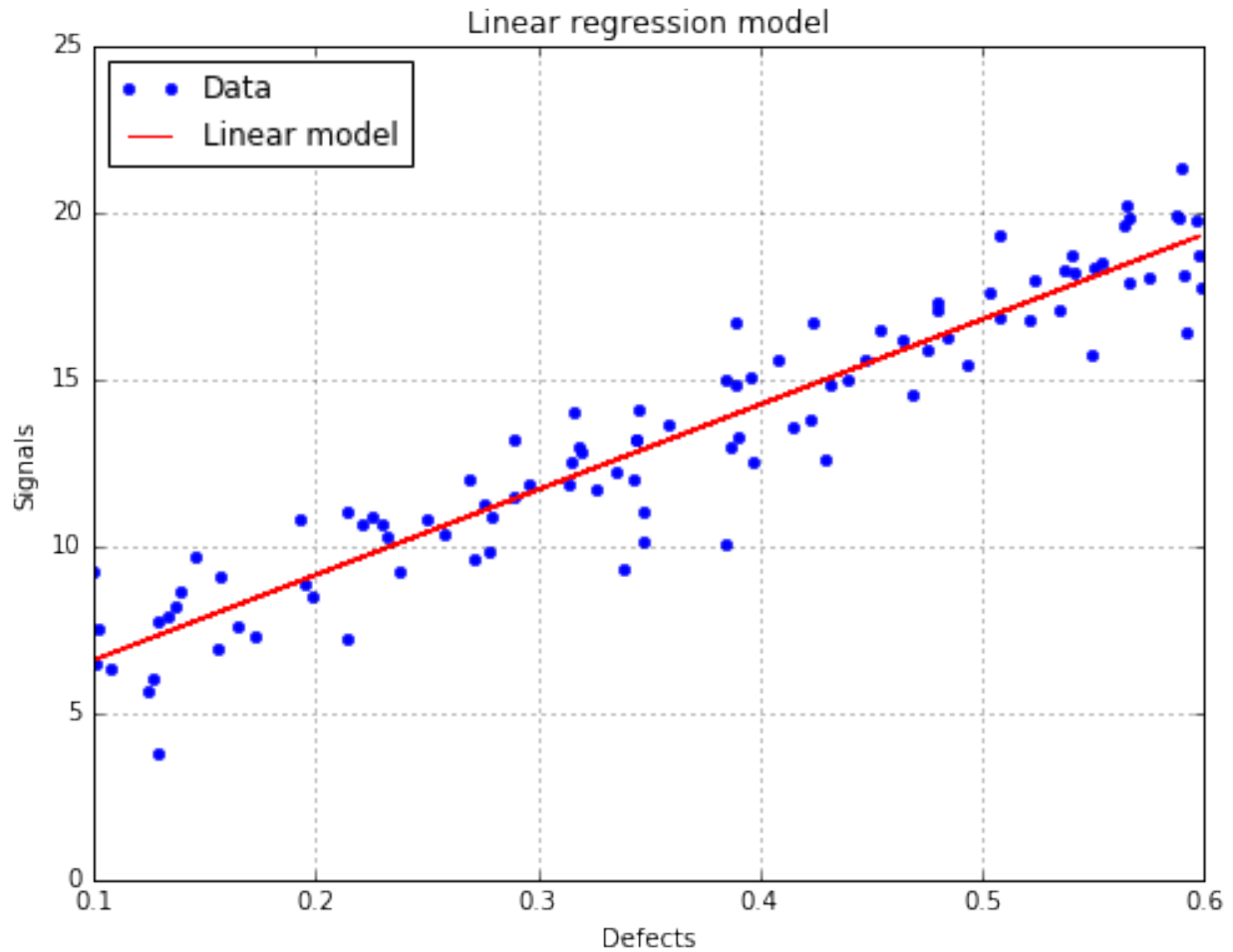
### Show graphs

#### The linear regression model with data

```

fig, ax = analysis.drawLinearModel(name='figure/linearModel.png')
# The figure is saved as png file
fig.show()

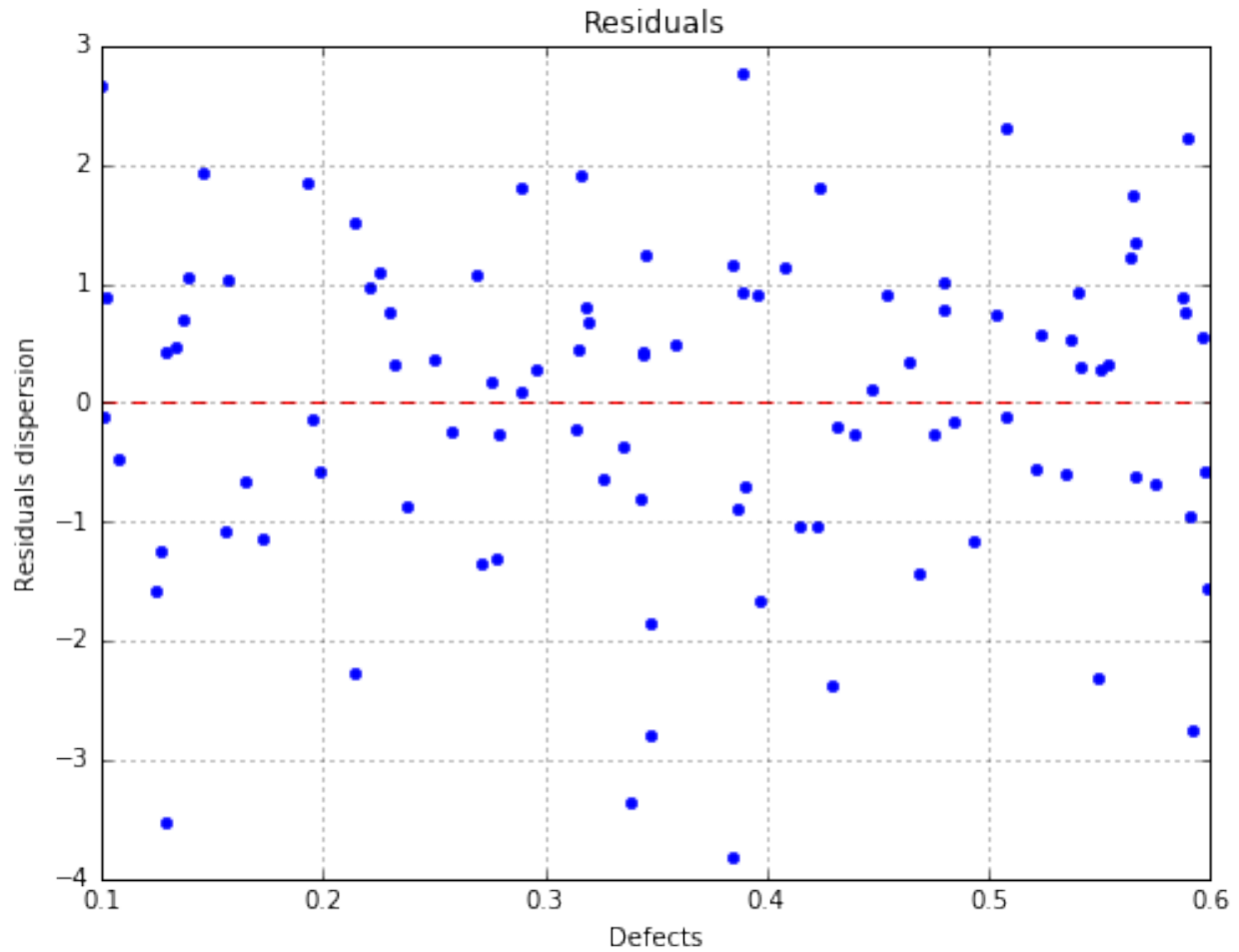
```



### The residuals with respect to the defects

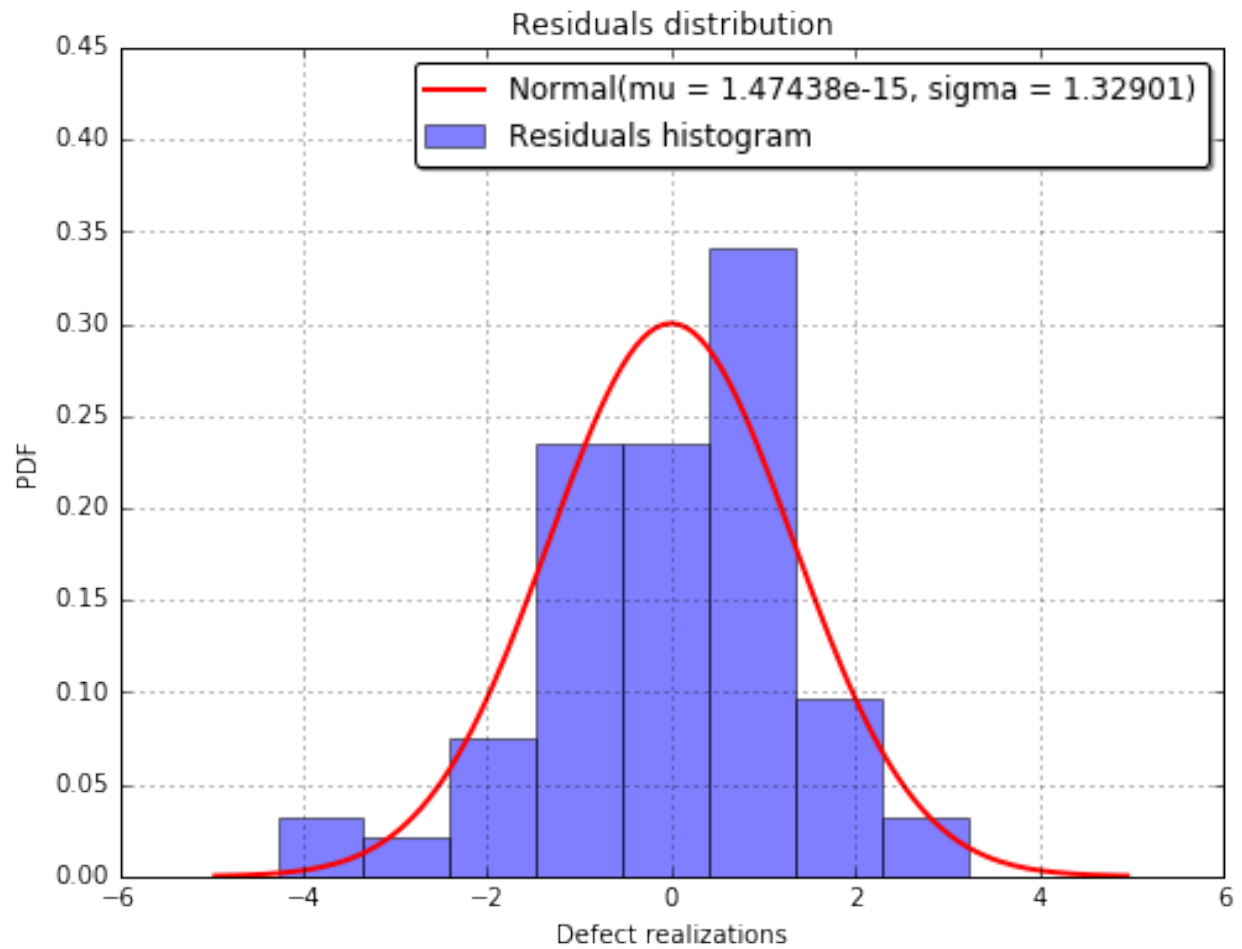
```
fig, ax = analysis.drawResiduals(name='figure/residuals.eps')  
# The figure is saved as eps file  
fig.show()
```





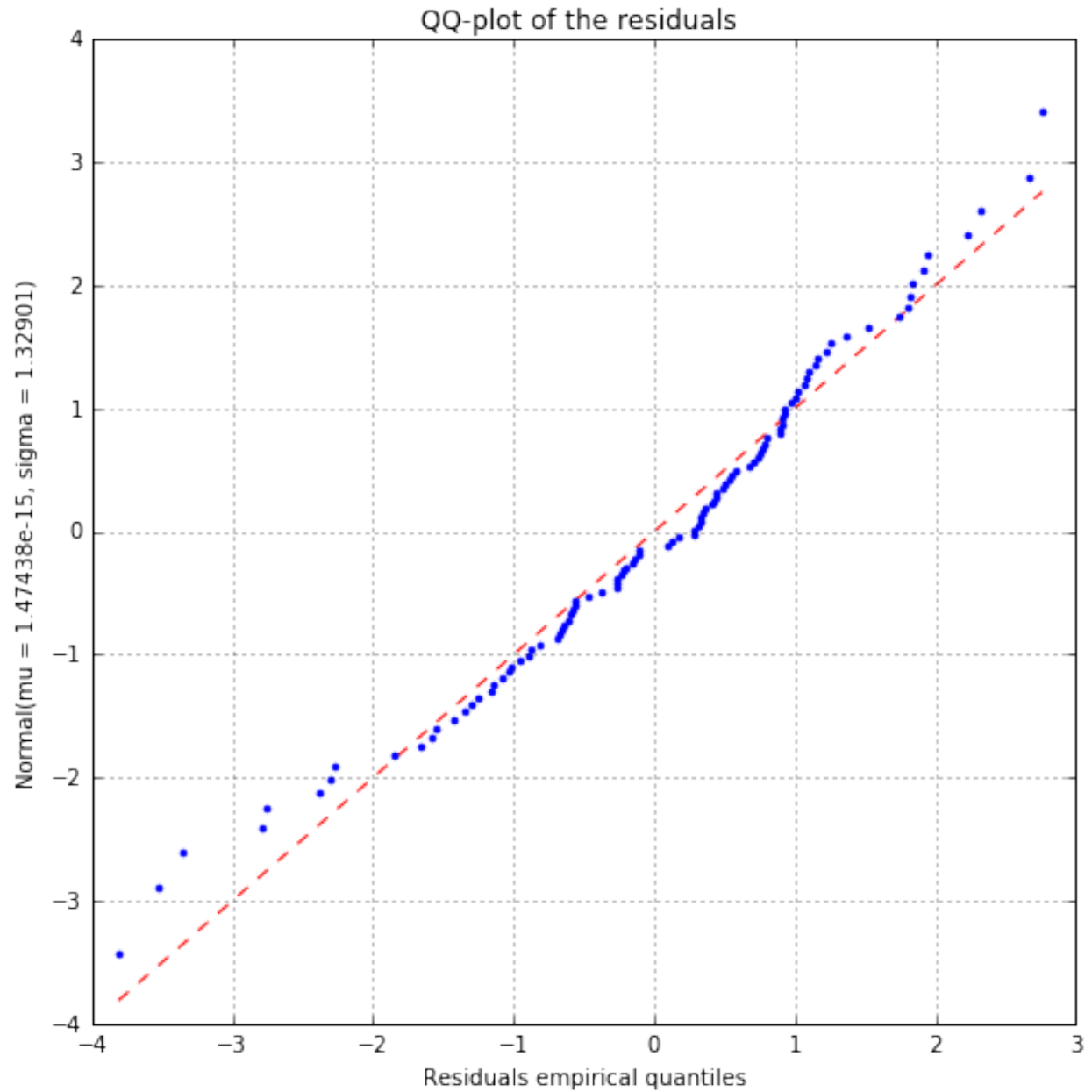
### The fitted residuals distribution with the histogram

```
fig, ax = analysis.drawResidualsDistribution()
ax.set_ylim(ymin=-0.45)
fig.show()
# The figure is saved after the changes
fig.savefig('figure/residualsDistribution.png', bbox_inches='tight')
```



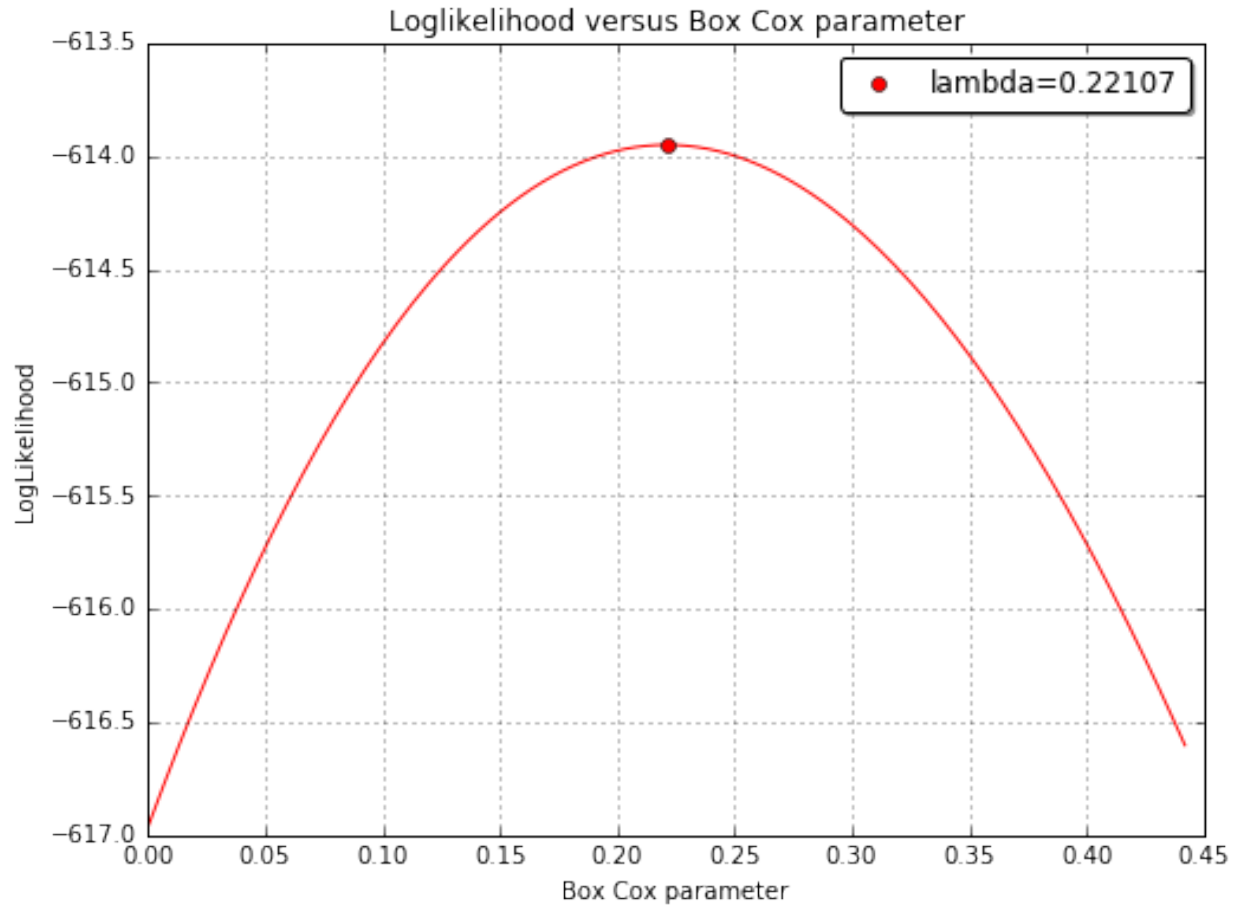
### The residuals QQ plot

```
fig, ax = analysis.drawResidualsQQplot()  
fig.show()
```



### The Box Cox likelihood with respect to the defect

```
fig, ax = analysis.drawBoxCoxLikelihood(name='figure/BoxCoxlikelihood.png')
fig.show()
```



ipy nb source code

## 1.2.2 Linear model analysis with censored data

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
```

### Generate data

```
N = 100
ot.RandomGenerator.SetSeed(123456)
defectDist = ot.Uniform(0.1, 0.6)
# normal epsilon distribution
epsilon = ot.Normal(0, 1.9)
defects = defectDist.getSample(N)
signalsInvBoxCox = defects * 43. + epsilon.getSample(N) + 2.5
# Inverse Box Cox transformation
invBoxCox = ot.InverseBoxCoxTransform(0.3)
signals = invBoxCox(signalsInvBoxCox)
```

## Run analysis with Box Cox

```
noiseThres = 60.
saturationThres = 1700.
analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, noiseThres,
                                                saturationThres, boxCox=True)
```

## Get some particular results

Result values are given for both analysis performed on filtered data (uncensored case) and on censored data.

```
print analysis.getIntercept()
print analysis.getR2()
print analysis.getKolmogorovPValue()
```

```
[Intercept for uncensored case : 4.777, Intercept for censored case : 4.1614]
[R2 for uncensored case : 0.869115, R2 for censored case : 0.860722]
[Kolmogorov p-value for uncensored case : 0.477505, Kolmogorov p-value for censored
↪ case : 0.505919]
```

## Print all results of the linear regression and all tests on the residuals

```
# Results are displayed for both case
analysis.printResults()
```

```
-----
Linear model analysis results
-----
Box Cox parameter :                                0.18

                                Uncensored    Censored

Intercept coefficient :                4.78        4.16
Slope coefficient :                18.15        19.94
Standard error of the estimate :                0.97        1.03

Confidence interval on coefficients
Intercept coefficient :                [4.19, 5.36]
Slope coefficient :                [16.63, 19.67]
Level :                                0.95

Quality of regression
R2 (> 0.8):                                0.87        0.86
-----

Residuals analysis results
-----
Fitted distribution (uncensored) :                Normal(mu = -4.31838e-15, sigma = 0.
↪968046)
Fitted distribution (censored) :                Normal(mu = -0.0237409, sigma = 0.
↪998599)

                                Uncensored    Censored
```

Distribution fitting test		
Kolmogorov p-value (> 0.05):	0.48	0.51
Normality test		
Anderson Darling p-value (> 0.05):	0.06	0.08
Cramer Von Mises p-value (> 0.05):	0.07	0.09
Zero residual mean test		
p-value (> 0.05):	1.0	0.83
Homoskedasticity test (constant variance)		
Breush Pagan p-value (> 0.05):	0.69	0.71
Harrison McCabe p-value (> 0.05):	0.6	0.51
Non autocorrelation test		
Durbin Watson p-value (> 0.05):	0.43	0.48
-----		

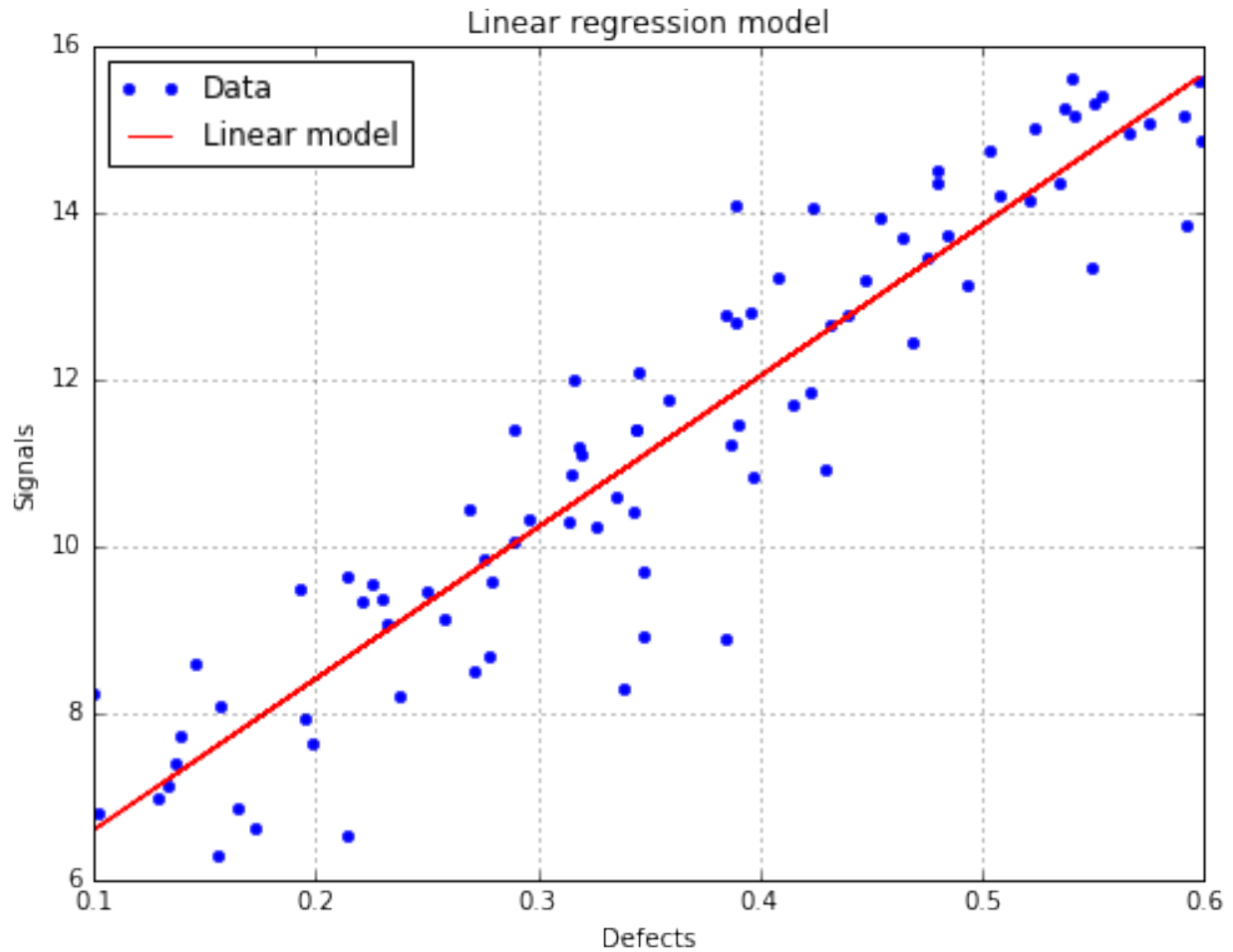
### Save all results in a csv file

```
analysis.saveResults('results.csv')
```

### Show graphs

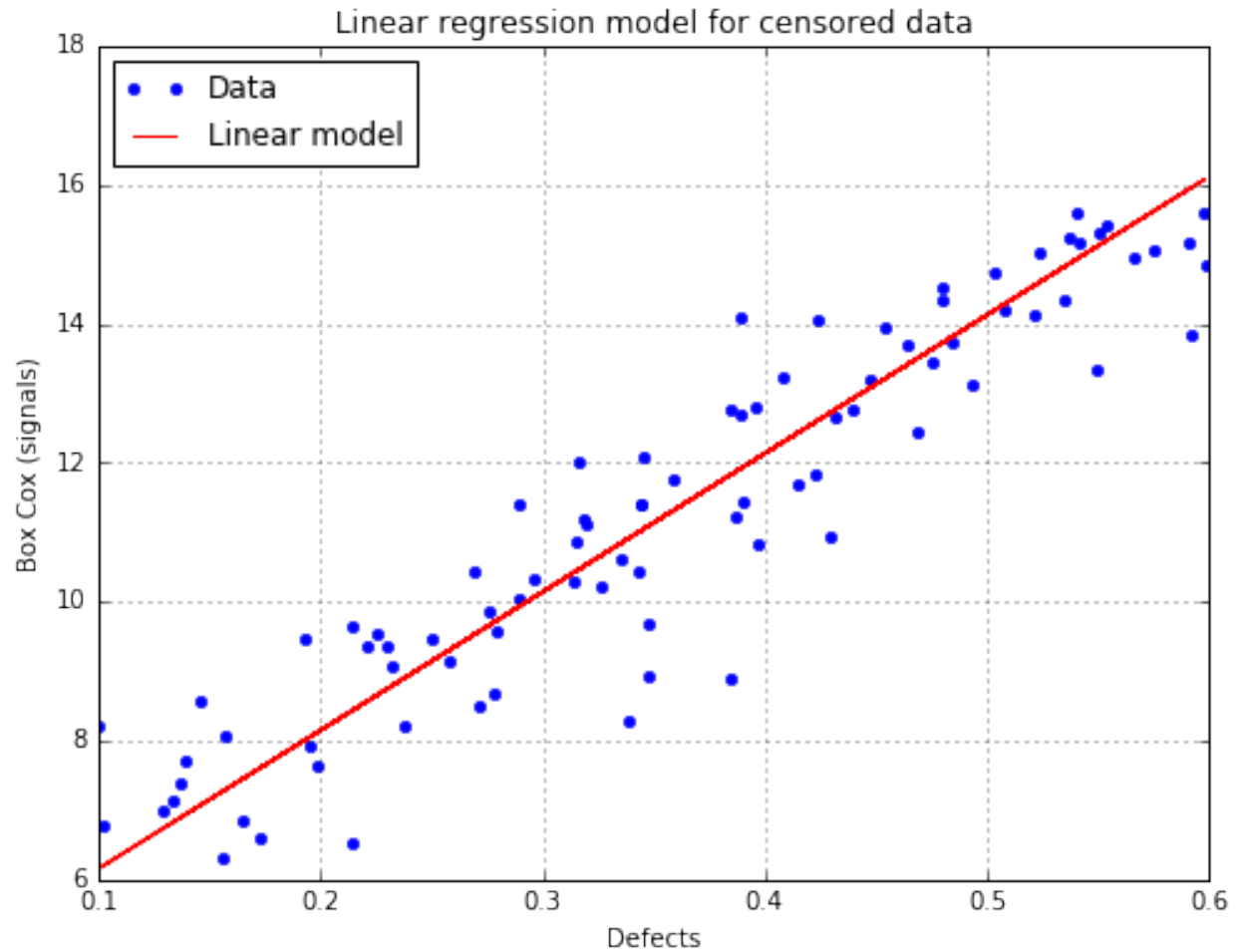
#### The linear regression model with data for the uncensored case (default case)

```
# draw the figure for the uncensored case and save it as png file
fig, ax = analysis.drawLinearModel(name='figure/linearModelUncensored.png')
fig.show()
```



The linear regression model with data for the censored case

```
# draw the figure for the censored case and save it as png file
fig, ax = analysis.drawLinearModel(model='censored', name='figure/linearModelCensored.
→png')
fig.show()
```



[ipynb source code](#)

### 1.2.3 Linear model POD

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
```

#### Generate data

```
N = 100
ot.RandomGenerator.SetSeed(123456)
defectDist = ot.Uniform(0.1, 0.6)
# normal epsilon distribution
epsilon = ot.Normal(0, 1.9)
defects = defectDist.getSample(N)
signalsInvBoxCox = defects * 43. + epsilon.getSample(N) + 2.5
# Inverse Box Cox transformation
```



```
invBoxCox = ot.InverseBoxCoxTransform(0.3)
signals = invBoxCox(signalsInvBoxCox)
```

## Build POD using previous linear analysis

```
# run the analysis with Gaussian hypothesis of the residuals (default case)
analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, boxCox=True)
```

```
# signal detection threshold
detection = 200.
# Use the analysis to build the POD with Gaussian hypothesis
# keyword arguments must be given
PODGauss = otpod.UnivariateLinearModelPOD(analysis=analysis, detection=detection)
PODGauss.run()
```

## Build POD with Gaussian hypothesis

```
# The previous POD is equivalent to the following POD
PODGauss = otpod.UnivariateLinearModelPOD(defects, signals, detection,
                                           resDistFact=ot.NormalFactory(),
                                           boxCox=True)
PODGauss.run()
```

## Get the R2 value of the regression

```
print 'R2 : {:.3f}'.format(PODGauss.getR2())
```

```
R2 : 0.895
```

## Compute detection size

```
# Detection size at probability level 0.9
# and confidence level 0.95
print PODGauss.computeDetectionSize(0.9, 0.95)

# probability level 0.95 with confidence level 0.99
print PODGauss.computeDetectionSize(0.95, 0.99)
```

```
[a90 : 0.303982, a90/95 : 0.317157]
[a95 : 0.323048, a95/99 : 0.343536]
```

## get POD NumericalMathFunction

```
# get the POD model
PODmodel = PODGauss.getPODModel()
# get the POD model at the given confidence level
PODmodelCl95 = PODGauss.getPODCLModel(0.95)
```

```
# compute the probability of detection for a given defect value
print 'POD : {:.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:.3f}'.format(PODmodelC195([0.3])[0])
```

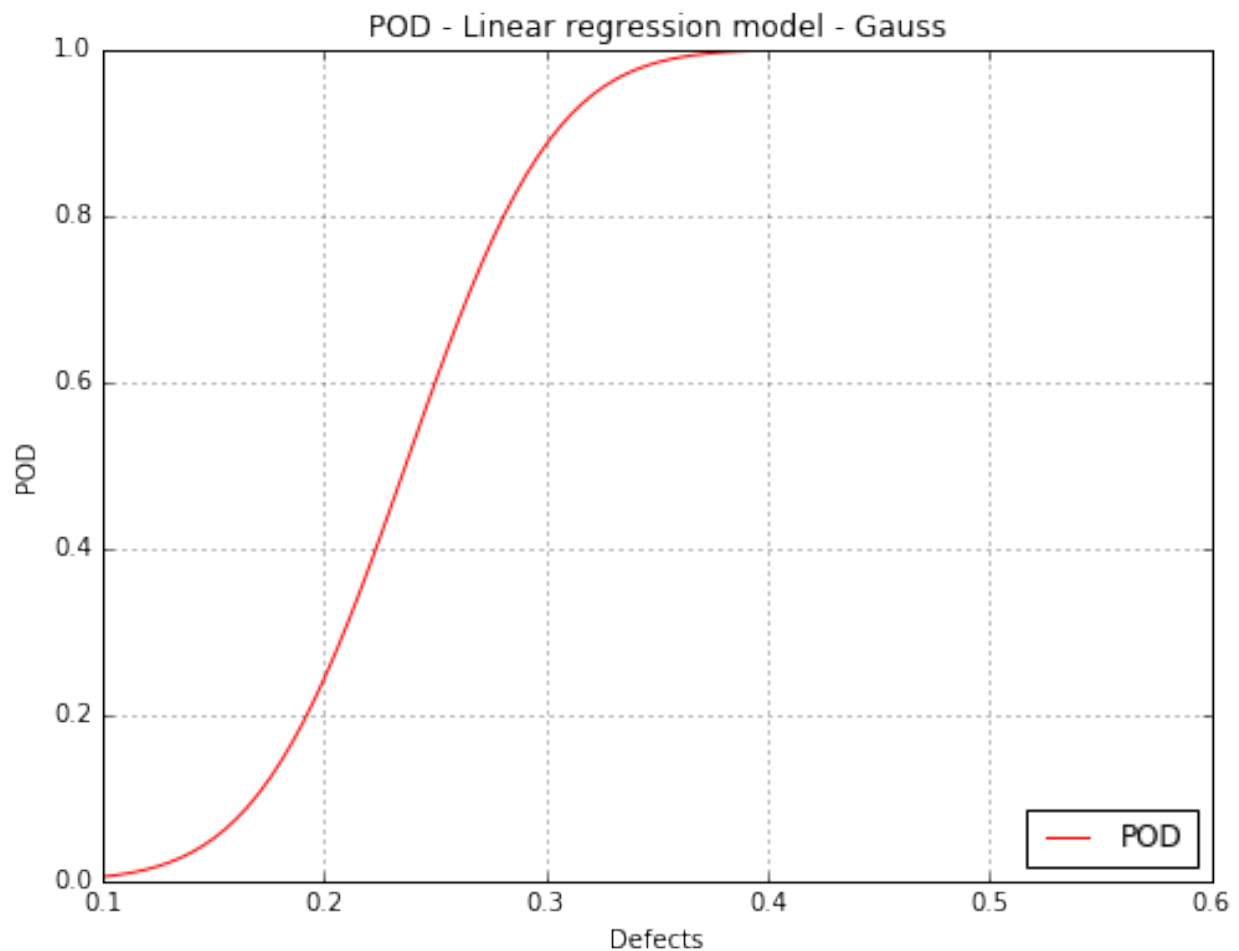
```
POD : 0.886
POD at level 0.95 : 0.834
```

## Show POD graphs

### Only the mean POD

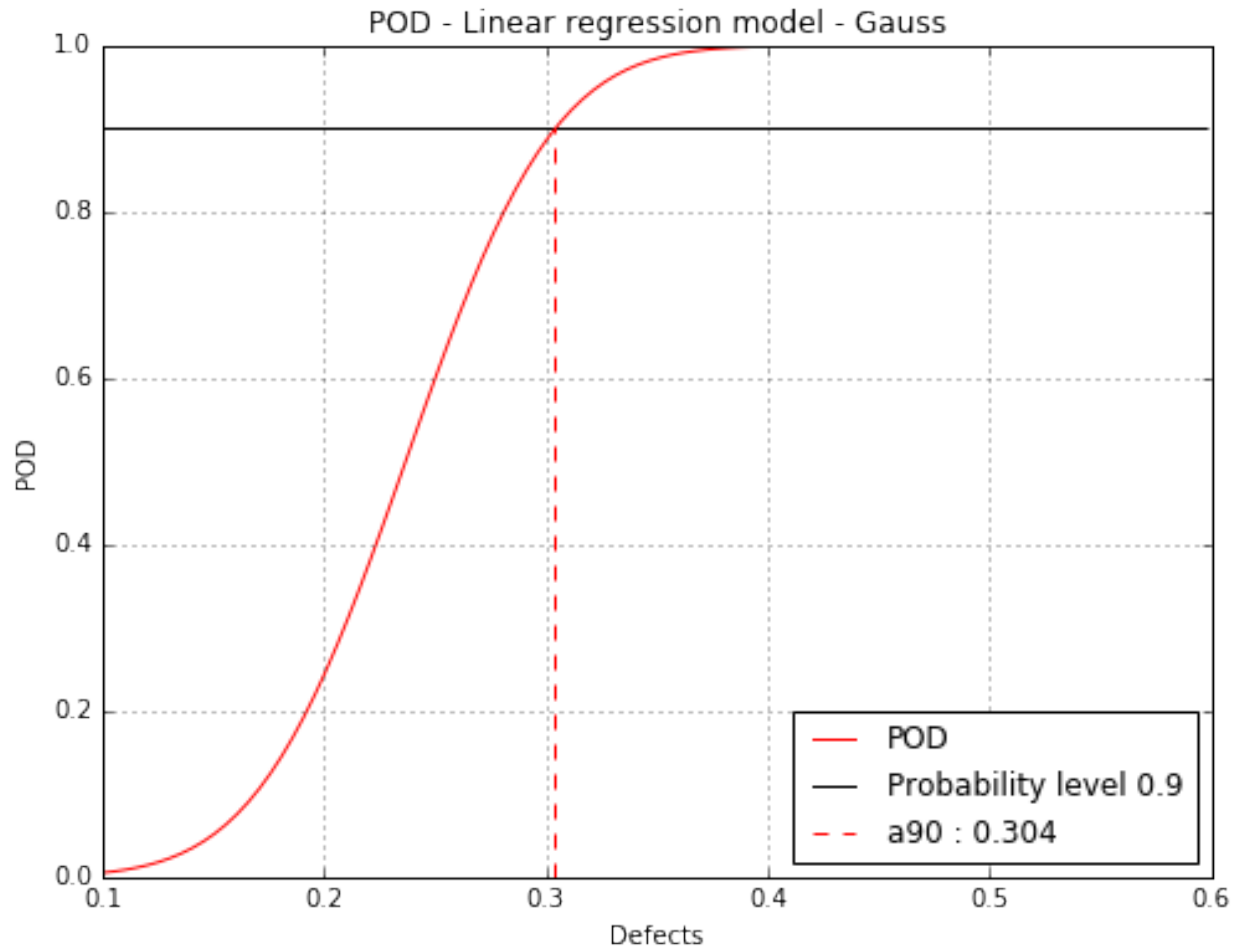
```
fig, ax = PODGauss.drawPOD()
fig.show()
```

```
/home/dumas/anaconda2/lib/python2.7/site-packages/matplotlib/figure.py:397:
↳UserWarning: matplotlib is currently using a non-GUI backend, so cannot show the
↳figure
  "matplotlib is currently using a non-GUI backend, "
```



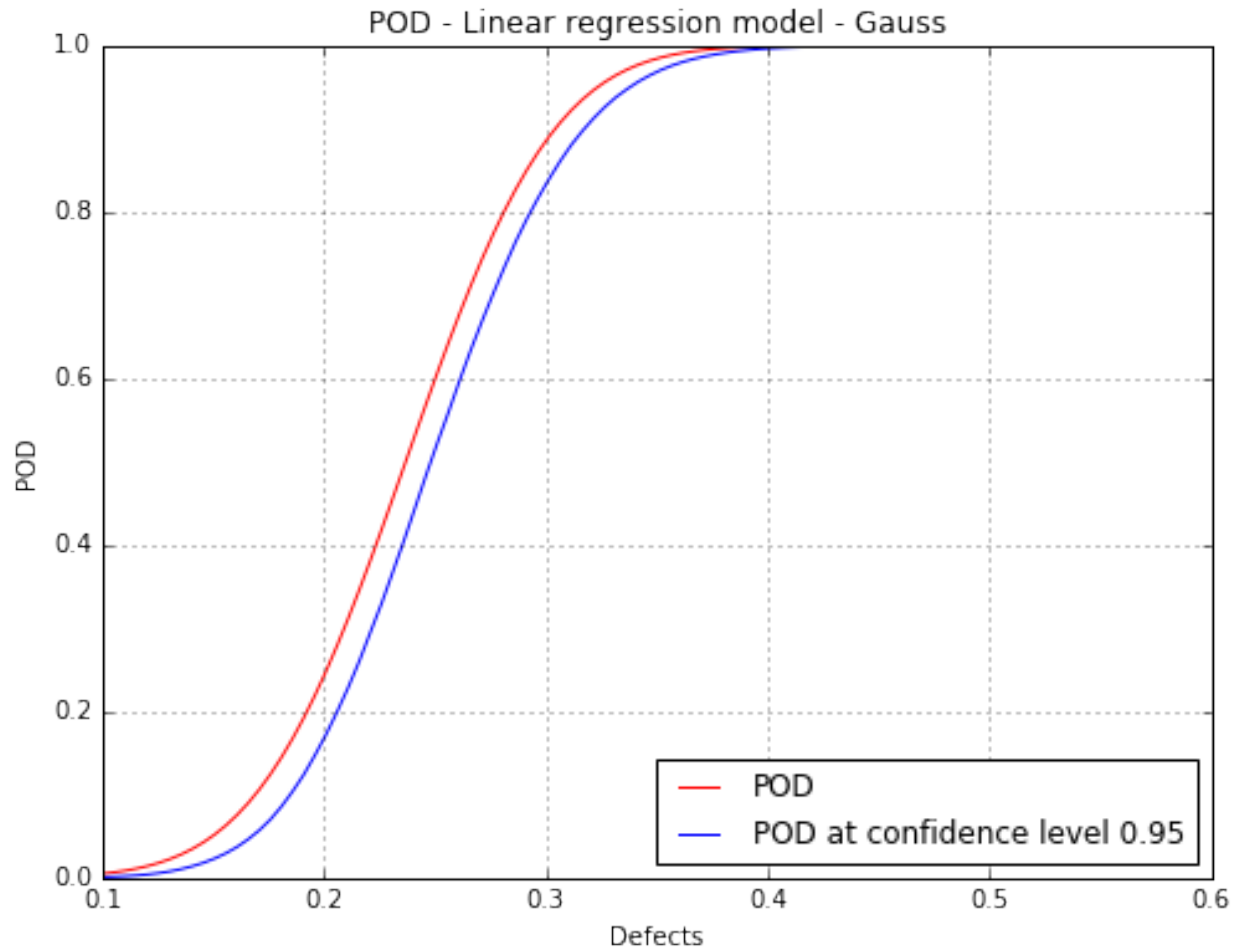
### Mean POD with the detection size for a given probability level

```
fig, ax = PODGauss.drawPOD(probabilityLevel=0.9)
fig.show()
```



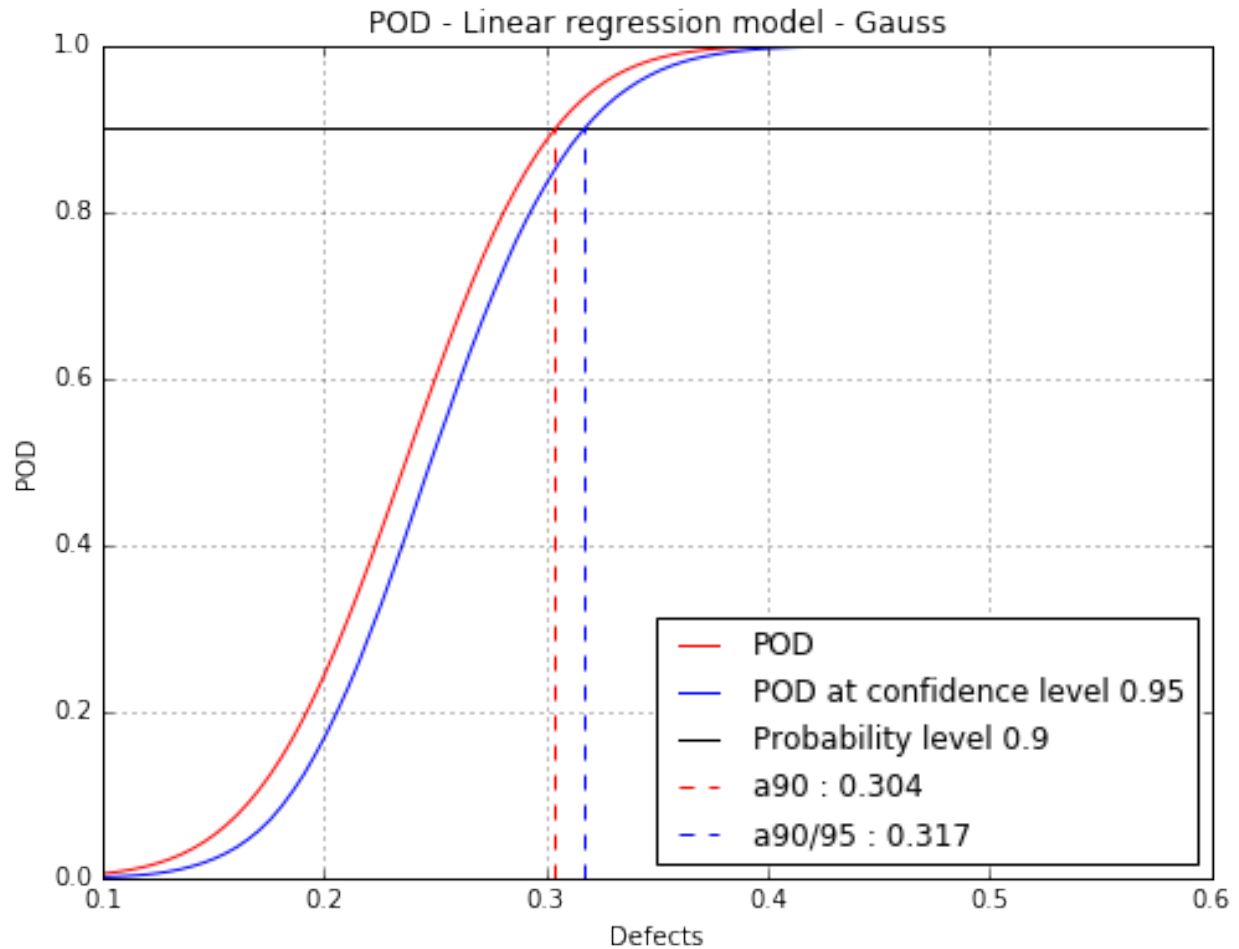
### Mean POD with POD at confidence level

```
fig, ax = PODGauss.drawPOD(confidenceLevel=0.95)
fig.show()
```



**Mean POD and POD at confidence level with the detection size for a given probability level**

```
fig, ax = PODGauss.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95,  
                             name='figure/PODGauss.png')  
# The figure is saved in PODGauss.png  
fig.show()
```



### Build POD with no hypothesis on the residuals

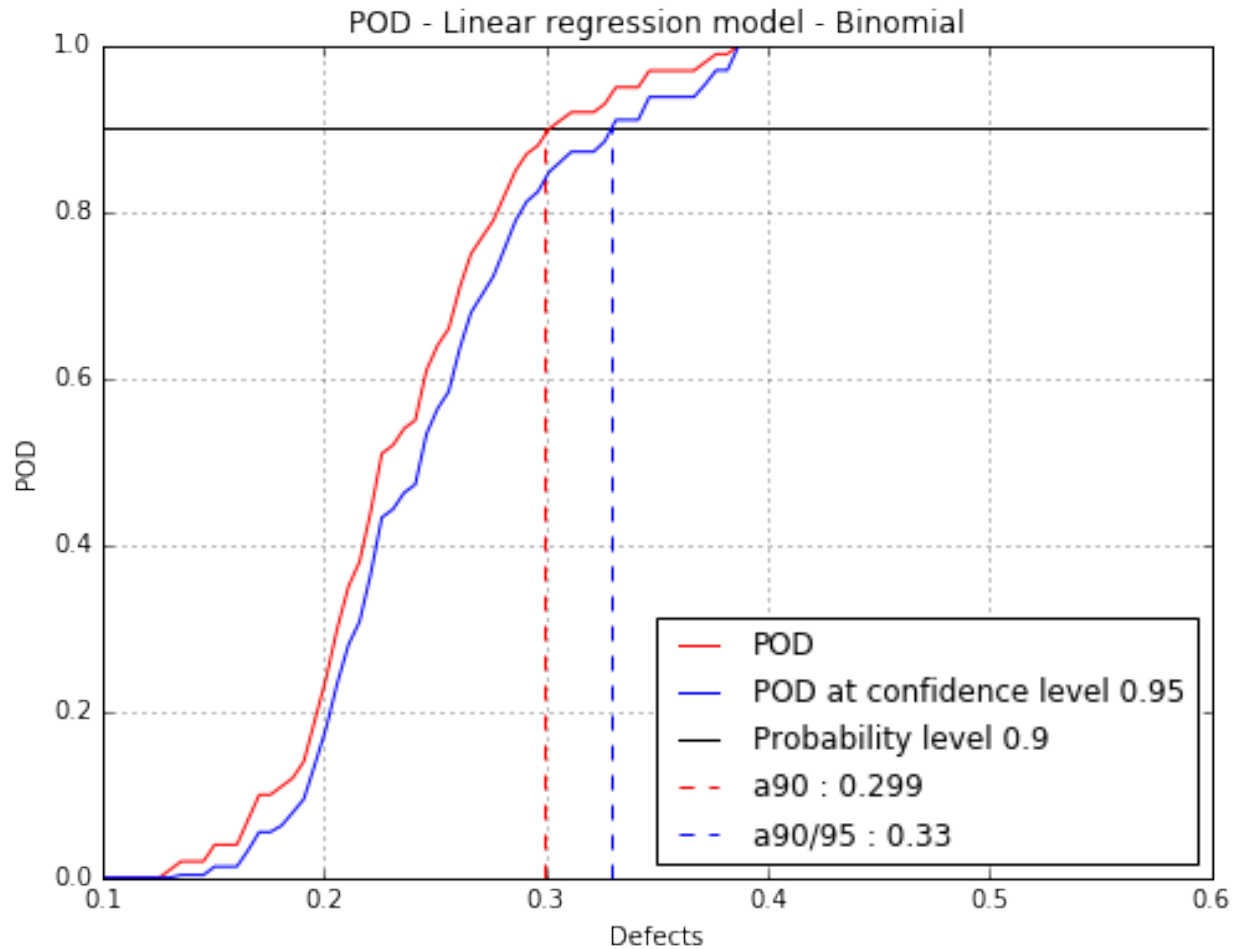
This corresponds with the Berens Binomial method.

```
PODBinomial = otpod.UnivariateLinearModelPOD(defects, signals, detection, boxCox=True)
PODBinomial.run()
```

```
# Detection size at probability level 0.9
# and confidence level 0.95
print PODBinomial.computeDetectionSize(0.9, 0.95)
```

```
[a90 : 0.298739, a90/95 : 0.329606]
```

```
fig, ax = PODBinomial.drawPOD(0.9, 0.95)
fig.show()
```



### Build POD with kernel smoothing on the residuals

The POD at the given confidence level is built using bootstrap. It may take few seconds. A progress bar is displayed in this case. It can be removed using `setVerbose(False)`

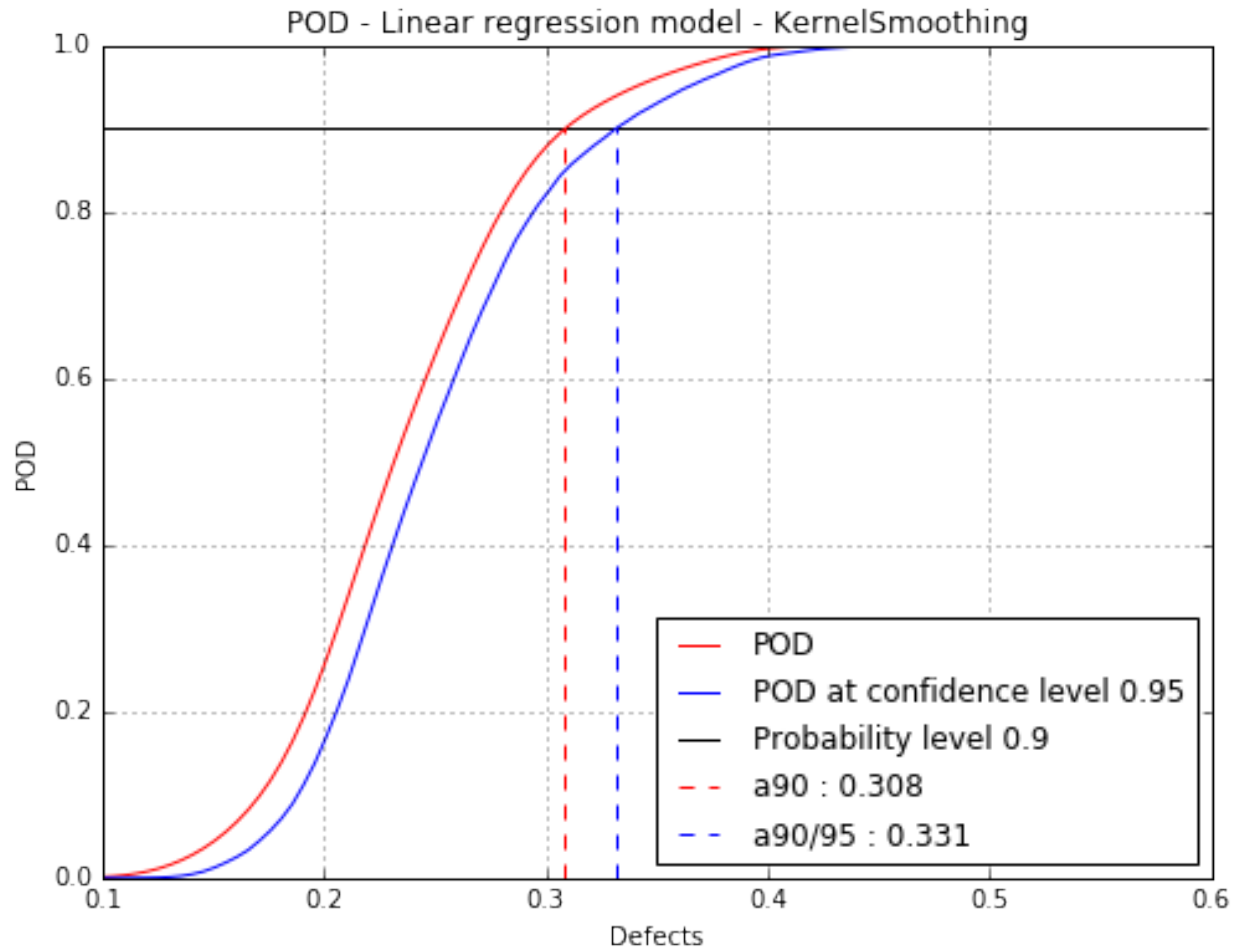
```
PODks = otpod.UnivariateLinearModelPOD(defects, signals, detection,
                                         resDistFact=ot.KernelSmoothing(),
                                         boxCox=True)
PODks.run()
```

```
Computing POD (bootstrap): [=====] 100%
↪ Done
```

```
# Detection size at probability level 0.9
# and confidence level 0.95
print PODks.computeDetectionSize(0.9, 0.95)
```

```
[a90 : 0.308381, a90/95 : 0.331118]
```

```
fig, ax = PODks.drawPOD(0.9, 0.95)
fig.show()
```



ipynb source code

### 1.2.4 Linear model POD with censored data

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
```

#### Generate data

```
N = 100
ot.RandomGenerator.SetSeed(123456)
defectDist = ot.Uniform(0.1, 0.6)
# normal epsilon distribution
epsilon = ot.Normal(0, 1.9)
defects = defectDist.getSample(N)
signalsInvBoxCox = defects * 43. + epsilon.getSample(N) + 2.5
# Inverse Box Cox transformation
```

```
invBoxCox = ot.InverseBoxCoxTransform(0.3)
signals = invBoxCox(signalsInvBoxCox)
```

### Build POD using previous linear analysis

```
noiseThres = 60.
saturationThres = 1700.

# run the analysis with Gaussian hypothesis of the residuals (default case)
analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, noiseThres,
                                                saturationThres, boxCox=True)
```

```
# signal detection threshold
detection = 200.
# Use the analysis to build the POD with Gaussian hypothesis
# keyword arguments must be given
PODGauss = otpod.UnivariateLinearModelPOD(analysis=analysis, detection=detection)
PODGauss.run()
```

### Build POD with Gaussian hypothesis

```
# The previous POD is equivalent to the following POD
PODGauss = otpod.UnivariateLinearModelPOD(defects, signals, detection,
                                           noiseThres, saturationThres,
                                           resDistFact=ot.NormalFactory(),
                                           boxCox=True)

PODGauss.run()
```

### Get the R2 value of the regression

```
print 'R2 : {:.3f}'.format(PODGauss.getR2())
```

```
R2 : 0.861
```

### Compute detection size

```
# Detection size at probability level 0.9
# and confidence level 0.95
print PODGauss.computeDetectionSize(0.9, 0.95)
```

```
[a90 : 0.30373, a90/95 : 0.317848]
```

### get POD NumericalMathFunction

```
# get the POD model
PODmodel = PODGauss.getPODModel()
# get the POD model at the given confidence level
```



```

PODmodelC195 = PODGauss.getPODCLModel(0.95)

# compute the probability of detection for a given defect value
print 'POD : {:.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:.3f}'.format(PODmodelC195([0.3])[0])

```

```

POD : 0.887
POD at level 0.95 : 0.830

```

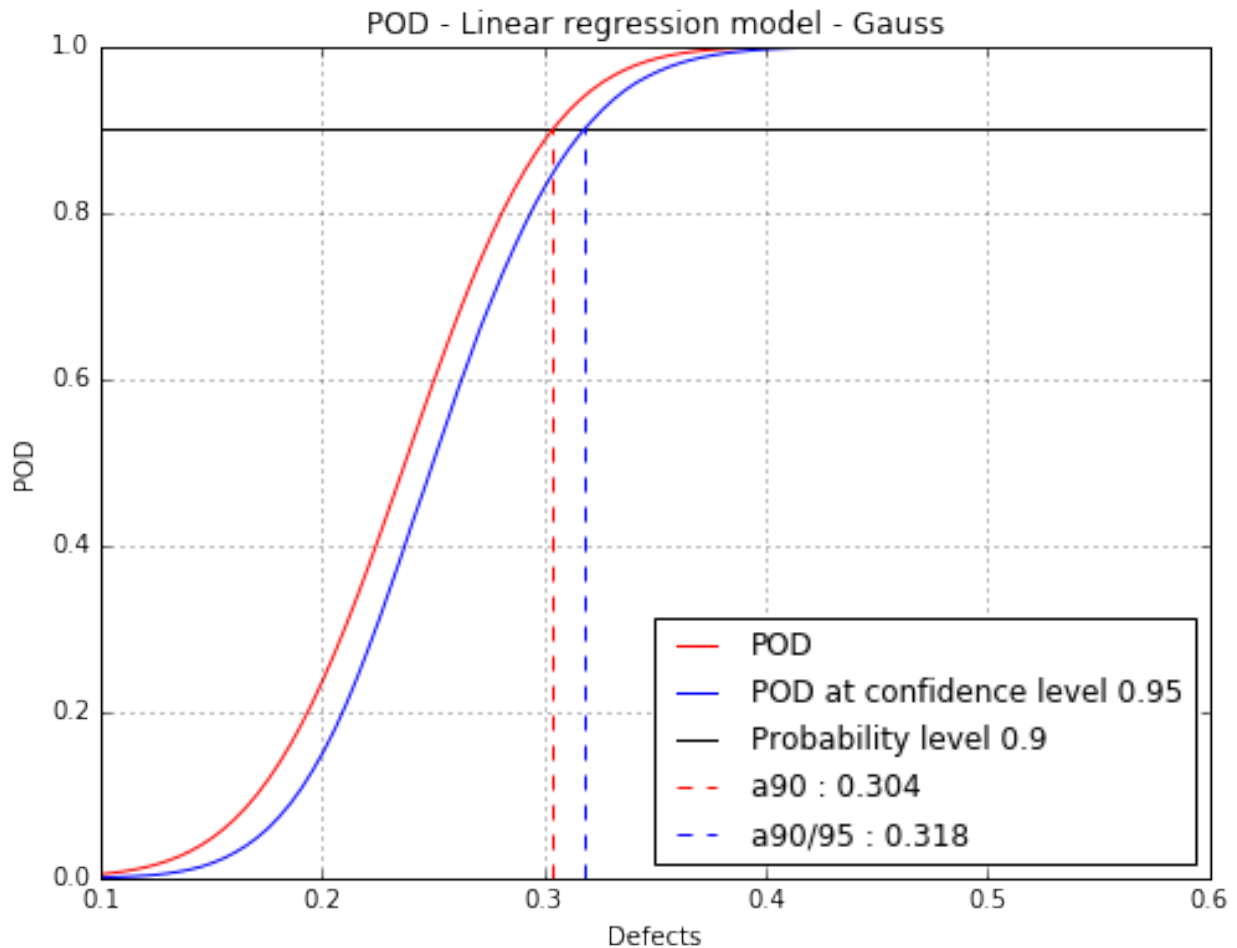
## Show POD graph

### Mean POD and POD at confidence level with the detection size for a given probability level

```

fig, ax = PODGauss.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95,
                             name='figure/PODGaussCensored.png')
# The figure is saved in PODGauss.png
fig.show()

```



## Build POD only with the filtered data

A static method is used to get the defects and signals only in the uncensored area.

```
print otpod.DataHandling.filterCensoredData.__doc__
```

Sort inputSample **and** signals **with** respect to the censure thresholds.

Parameters

```
-----
inputSample : 2-d sequence of float
              Vector of the input sample.
signals : 2-d sequence of float
          Vector of the signals, of dimension 1.
noiseThres : float
            Value for low censored data. Default is None.
saturationThres : float
                Value for high censored data. Default is None
```

Returns

```
-----
inputSampleUnc : 2-d sequence of float
                Vector of the input sample in the uncensored area.
inputSampleNoise : 2-d sequence of float
                  Vector of the input sample in the noisy area.
inputSampleSat : 2-d sequence of float
                Vector of the input sample in the saturation area.
signalsUnc : 2-d sequence of float
            Vector of the signals in the uncensored area.
```

Notes

```
-----
The data are sorted in three different vectors whether they belong to
the noisy area, the uncensored area or the saturation area.
```

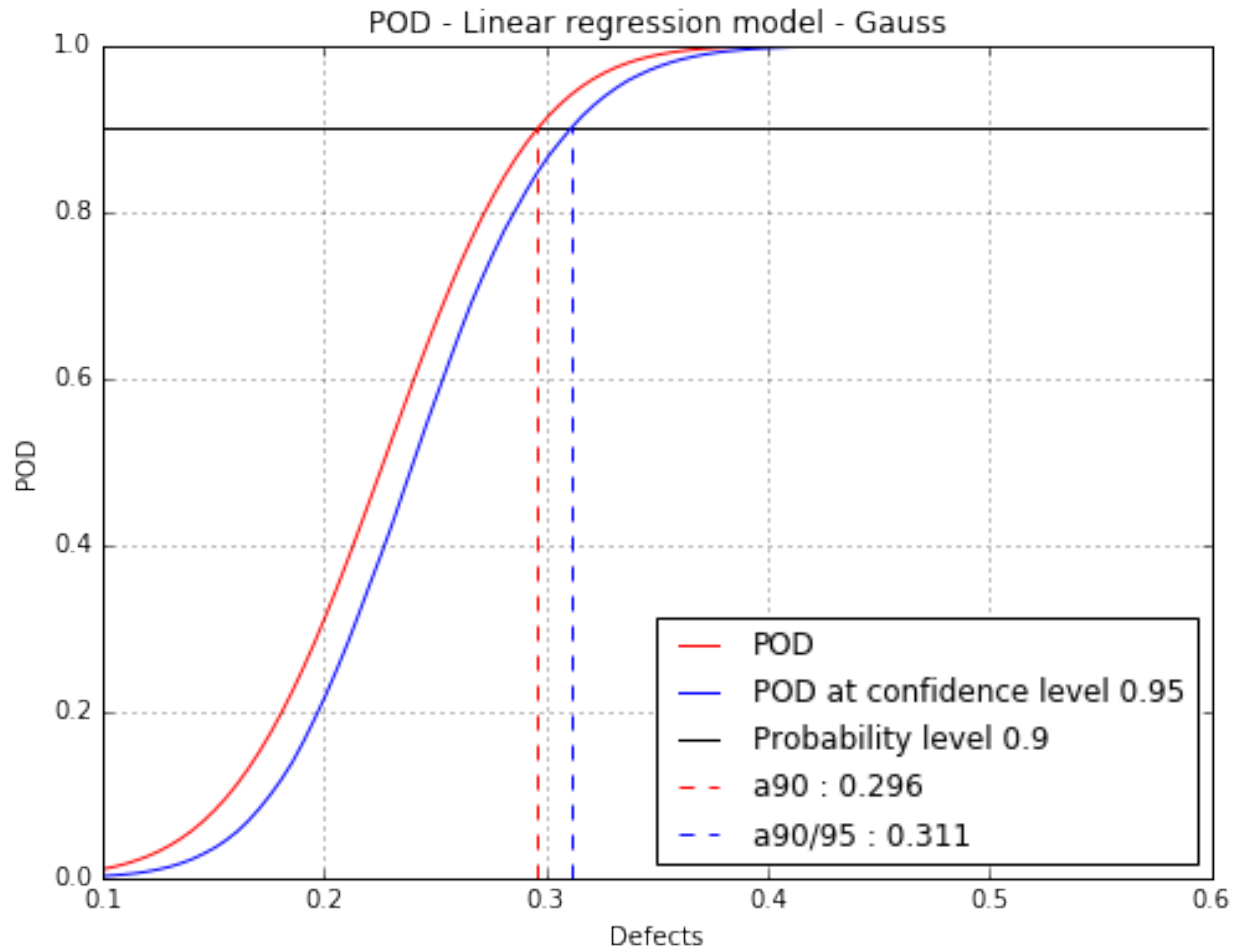
```
result = otpod.DataHandling.filterCensoredData(defects, signals,
                                                noiseThres, saturationThres)
defectsFiltered = result[0]
signalsFiltered = result[3]
```

```
PODfilteredData = otpod.UnivariateLinearModelPOD(defectsFiltered, signalsFiltered,
                                                  detection,
                                                  resDistFact=ot.NormalFactory(),
                                                  boxCox=True)
PODfilteredData.run()
```

```
# Detection size at probability level 0.9
# and confidence level 0.95
print PODfilteredData.computeDetectionSize(0.9, 0.95)
```

```
[a90 : 0.295976, a90/95 : 0.310948]
```

```
fig, ax = PODfilteredData.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95,
                                  name='figure/PODGaussFiltered.png')
# The figure is saved in PODGauss.png
fig.show()
```



ipynb source code

## 1.2.5 Quantile Regression POD

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
from time import time
```

### Generate data

```
N = 100
ot.RandomGenerator.SetSeed(123456)
defectDist = ot.Uniform(0.1, 0.6)
# normal epsilon distribution
epsilon = ot.Normal(0, 1.9)
defects = defectDist.getSample(N)
signalsInvBoxCox = defects * 43. + epsilon.getSample(N) + 2.5
# Inverse Box Cox transformation
```

```
invBoxCox = ot.InverseBoxCoxTransform(0.3)
signals = invBoxCox(signalsInvBoxCox)
```

### Build POD with quantile regression technique

```
# signal detection threshold
detection = 200.
# The POD with censored data actually builds a POD only on filtered data.
# A warning is displayed in this case.
POD = otpod.QuantileRegressionPOD(defects, signals, detection,
                                  noiseThres=60., saturationThres=1700.,
                                  boxCox=True)
```

```
INFO:root:Censored data are not taken into account : the quantile regression model is
↳ only performed on filtered data.
```

### Quantile user-defined

```
# Default quantile values
print 'Default quantile : '
print POD.getQuantile()
# Defining user quantile, they must range between 0 and 1.
POD.setQuantile([0.1, 0.3, 0.5, 0.7, 0.8, 0.85, 0.9, 0.95])
print 'User-defined quantile : '
print POD.getQuantile()
```

```
Default quantile :
[ 0.05   0.0965  0.143   0.1895  0.236   0.2825  0.329   0.3755  0.422
  0.4685  0.515   0.5615  0.608   0.6545  0.701   0.7475  0.794   0.8405
  0.887   0.9335  0.98   ]
User-defined quantile :
[ 0.1   0.3   0.5   0.7   0.8   0.85  0.9   0.95]
```

### Running quantile regression POD

```
# Due to the bootstrap technique used to compute the confidence
# interval, the run takes few minutes.
# A progress bar is displayed (can be removed using setVerbose(False))
t0 = time()
POD = otpod.QuantileRegressionPOD(defects, signals, detection,
                                  boxCox=True)

POD.run()
print 'Computing time : {0.2f} s'.format(time()-t0)
```

```
Computing defect quantile: [=====] 100%
↳ Done
Computing time : 209.77 s
```

The computing time can be reduced by setting the simulation size attribute to another value. However the confidence interval is less accurate.

The number of quantile values can also be reduced to save time.

```
t0 = time()
PODsimulSize100 = otpod.QuantileRegressionPOD(defects, signals, detection,
                                              boxCox=True)
PODsimulSize100.setSimulationSize(100) # default is 1000
PODsimulSize100.run()
print 'Computing time : {:.2f} s'.format(time()-t0)
```

```
Computing defect quantile: [=====] 100%
↪ Done
Computing time : 21.62 s
```

## Compute detection size

```
# Detection size at probability level 0.9
# and confidence level 0.95
print POD.computeDetectionSize(0.9, 0.95)

# probability level 0.95 with confidence level 0.99
print POD.computeDetectionSize(0.95, 0.99)
```

```
[a90 : 0.298115, a90/95 : 0.328775]
[a95 : 0.331931, a95/99 : 0.372112]
```

## get POD NumericalMathFunction

```
# get the POD model
PODmodel = POD.getPODModel()
# get the POD model at the given confidence level
PODmodelC195 = POD.getPODCLModel(0.95)

# compute the probability of detection for a given defect value
print 'POD : {:.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:.3f}'.format(PODmodelC195([0.3])[0])
```

```
POD : 0.899
POD at level 0.95 : 0.832
```

## Compute the pseudo R2 for a given quantile

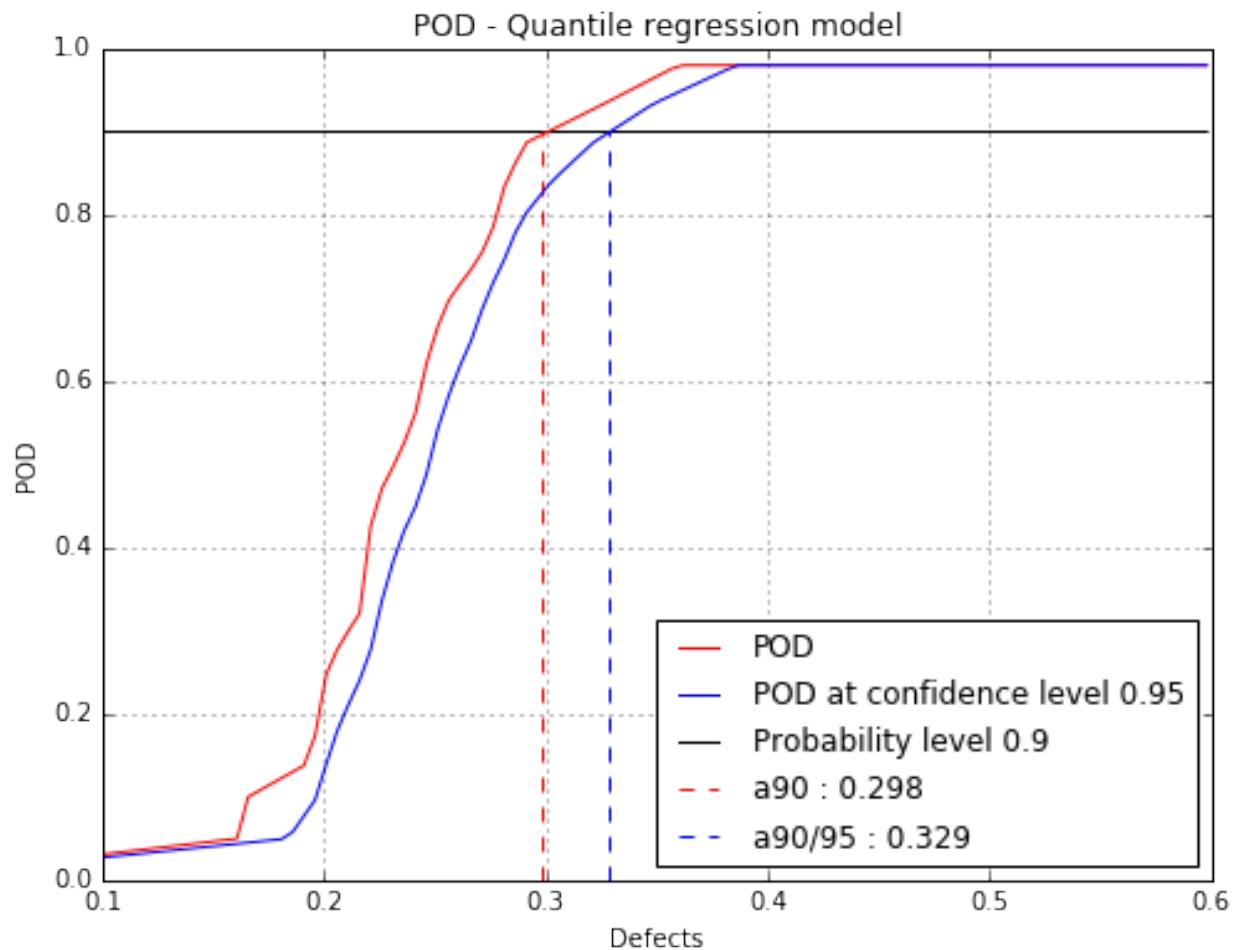
```
print 'Pseudo R2 for quantile 0.9 : {:.3f}'.format(POD.getR2(0.9))
print 'Pseudo R2 for quantile 0.95 : {:.3f}'.format(POD.getR2(0.95))
```

```
Pseudo R2 for quantile 0.9 : 0.675
Pseudo R2 for quantile 0.95 : 0.656
```

## Show POD graphs

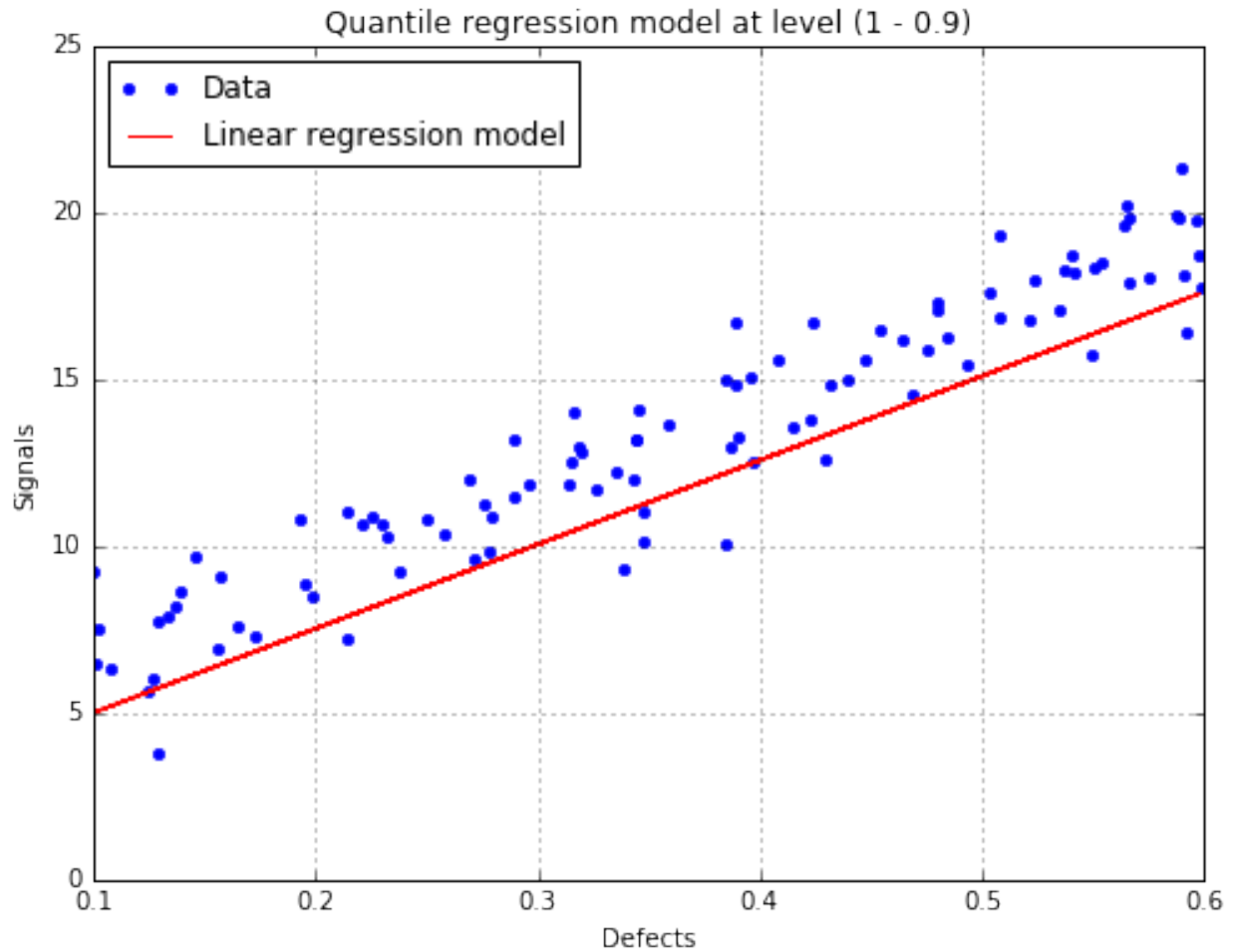
### Mean POD and POD at confidence level with the detection size for a given probability level

```
fig, ax = POD.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95,
                       name='figure/PODQuantReg.png')
# The figure is saved in PODQuantReg.png
fig.show()
```



### Show the linear regression model at the given quantile

```
fig, ax = POD.drawLinearModel(0.9)
fig.show()
```



[ipynb source code](#)

## 1.2.6 Polynomial chaos POD

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
```

### Generate 1D data

```
N = 100
ot.RandomGenerator.SetSeed(123456)
defectDist = ot.Uniform(0.1, 0.6)
# normal epsilon distribution
epsilon = ot.Normal(0, 1.9)
defects = defectDist.getSample(N)
signalsInvBoxCox = defects * 43. + epsilon.getSample(N) + 2.5
# Inverse Box Cox transformation
```

```
invBoxCox = ot.InverseBoxCoxTransform(0.3)
signals = invBoxCox(signalsInvBoxCox)
```

### Build POD with polynomial chaos model

```
# signal detection threshold
detection = 200.
# The POD with censored data actually builds a POD only on filtered data.
# A warning is displayed in this case.
POD = otpod.PolynomialChaosPOD(defects, signals, detection,
                               noiseThres=200., saturationThres=1700.,
                               boxCox=True)
```

### User-defined defect sizes

The user-defined defect sizes must range between the minimum and maximum of the defect values after filtering. An error is raised if it is not the case. The available range is then returned to the user.

```
# Default defect sizes
print 'Default defect sizes : '
print POD.getDefectSizes()

# Wrong range
POD.setDefectSizes([0.12, 0.3, 0.5, 0.57])
```

```
Default defect sizes :
[ 0.19288542  0.21420345  0.23552149  0.25683952  0.27815756  0.29947559
 0.32079363  0.34211166  0.3634297   0.38474773  0.40606577  0.4273838
 0.44870184  0.47001987  0.49133791  0.51265594  0.53397398  0.55529201
 0.57661005  0.59792808]
```

```
-----
ValueError                                Traceback (most recent call last)

<ipython-input-4-ccee3ce344ea> in <module>()
      4
      5 # Wrong range
----> 6 POD.setDefectSizes([0.12, 0.3, 0.5, 0.57])

/home/dumas/projet/ByPASS_pmpr635/otpod/otpod/_polynomial_chaos_pod.pyc in
->setDefectSizes(self, size)
    425         raise ValueError('Defect sizes must range between ' + \
    426                             '{:0.4f}'.format(np.ceil(minMin*10000)/10000) + \
->\
-> 427                             'and {:0.4f}'.format(np.floor(maxMax*10000)/
->10000))
    428         self._defectNumber = self._defectSizes.shape[0]
    429

ValueError: Defect sizes must range between 0.1929 and 0.5979.
```



```
# Good range
POD.setDefectSizes([0.1929, 0.3, 0.4, 0.5, 0.5979])
print 'User-defined defect size : '
print POD.getDefectSizes()
```

```
User-defined defect size :
[ 0.1929  0.3      0.4      0.5      0.5979]
```

## Running the polynomial chaos based POD

The computing time can be reduced by setting the simulation size attribute to another value. However the confidence interval is less accurate.

The sampling size is the number of the samples used to compute the POD with the Monte Carlo simulation for each defect sizes.

A progress is displayed, which can be disabled with the method `setVerbose`.

```
# Computing the confidence interval in the run takes few minutes.
POD = otpod.PolynomialChaosPOD(defects, signals, detection,
                               boxCox=True)
# we can change the sample size of the Monte Carlo simulation
POD.setSamplingSize(2000) # default is 5000
# we can also change the size of the simulation to compute the confidence interval
POD.setSimulationSize(500) # default is 1000
# we can change the degree of the polynomial chaos, default is 3.
POD.setDegree(3)
%time POD.run()
```

```
Start build polynomial chaos model...
Polynomial chaos model completed
R2 : 0.8947
Q2 : 0.8914
Computing POD per defect: [=====] 100.00
↪ % Done
CPU times: user 1min 19s, sys: 576 ms, total: 1min 19s
Wall time: 1min 5s
```

## Compute detection size

```
# Detection size at probability level 0.9
# and confidence level 0.95
print POD.computeDetectionSize(0.9, 0.95)

# probability level 0.95 with confidence level 0.99
print POD.computeDetectionSize(0.95, 0.99)
```

```
[a90 : 0.307344, a90/95 : 0.314406]
[a95 : 0.328888, a95/99 : 0.335715]
```

## get POD NumericalMathFunction

```
# get the POD model
PODmodel = POD.getPODModel()
# get the POD model at the given confidence level
PODmodelC195 = POD.getPODCLModel(0.95)

# compute the probability of detection for a given defect value
print 'POD : {:.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:.3f}'.format(PODmodelC195([0.3])[0])
```

```
POD : 0.871
POD at level 0.95 : 0.841
```

## Compute the R2 and the Q2

Enable to check the quality of the model.

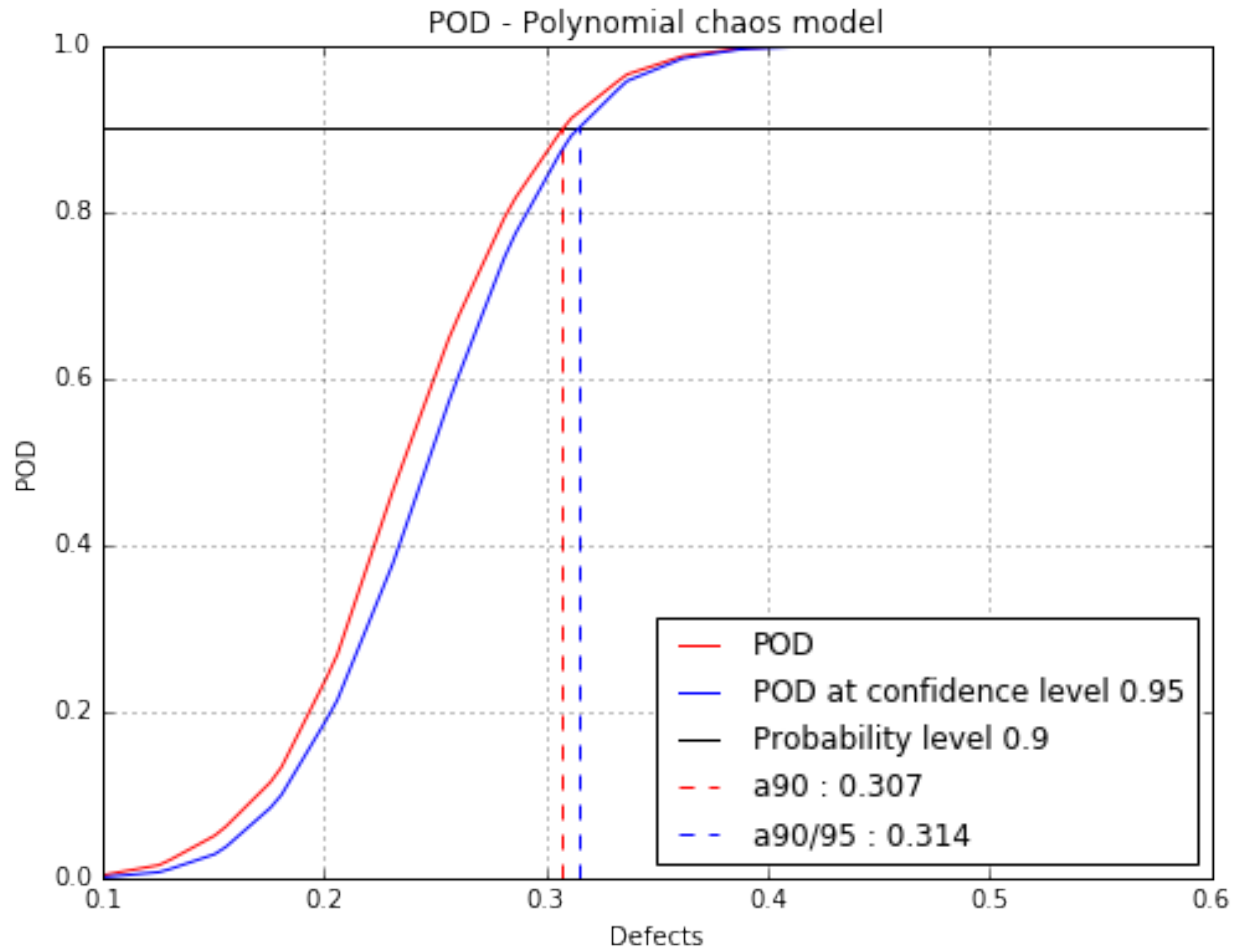
```
print 'R2 : {:.4f}'.format(POD.getR2())
print 'Q2 : {:.4f}'.format(POD.getQ2())
```

```
R2 : 0.8947
Q2 : 0.8914
```

## Show POD graphs

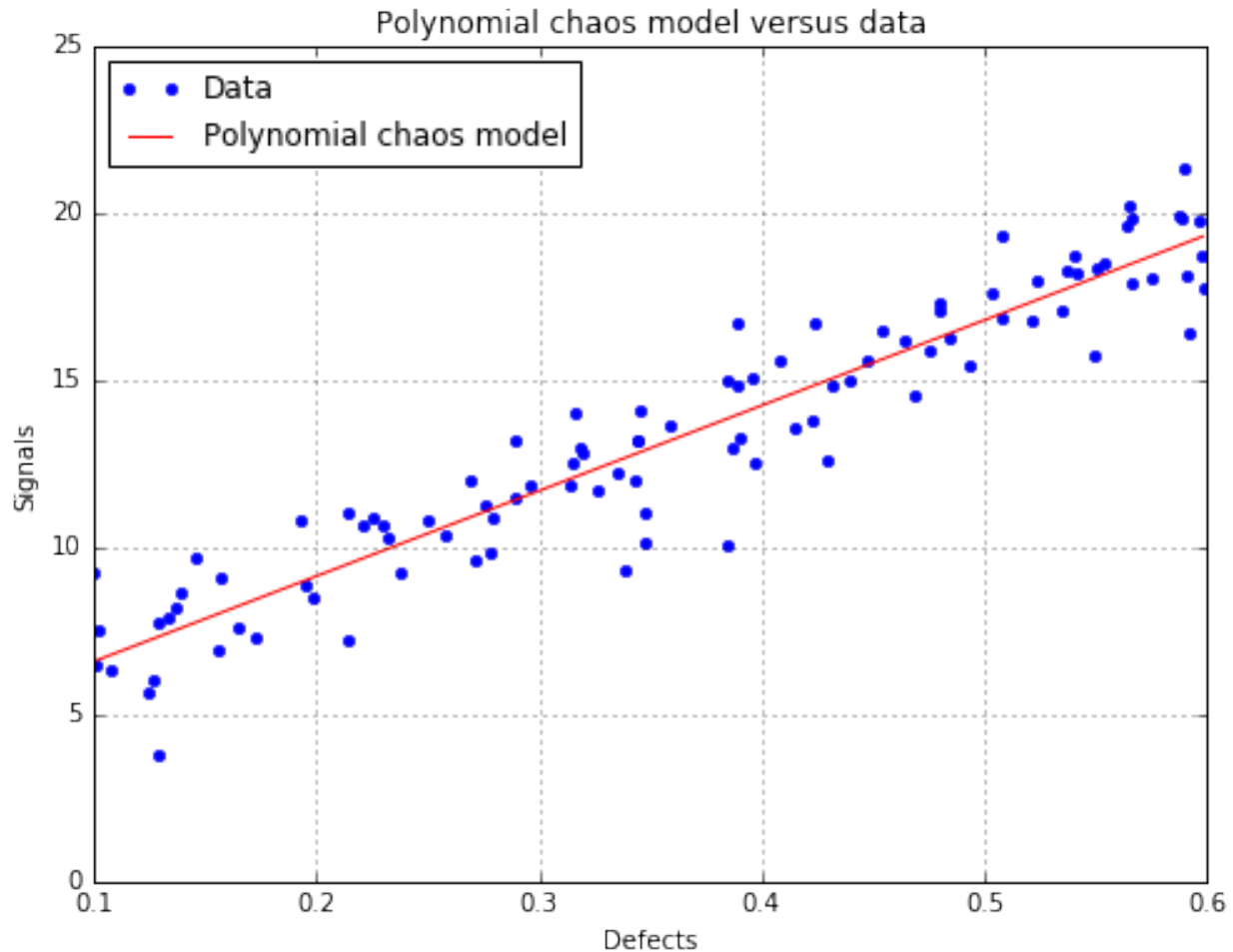
### Mean POD and POD at confidence level with the detection size for a given probability level

```
fig, ax = POD.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95,
                       name='figure/PODPolyChaos.png')
# The figure is saved in PODPolyChaos.png
fig.show()
```



Show the polynomial chaos model (only available if the input dimension is 1)

```
fig, ax = POD.drawPolynomialChaosModel()  
fig.show()
```



### Advanced user mode

The user can defined one or all parameters of the polynomial chaos algorithm : - the distribution of the input parameters  
- the adaptive strategy - the projection strategy

```
# new POD study
PODnew = otpod.PolynomialChaosPOD(defects, signals, detection,
                                   boxCox=True)
```

```
# define the input parameter distribution
distribution = ot.ComposedDistribution([ot.Normal(0.3, 0.1)])
PODnew.setDistribution(distribution)
```

```
# define the adaptive strategy
polyCol = [ot.HermiteFactory()]
enumerateFunction = ot.EnumerateFunction(1)
multivariateBasis = ot.OrthogonalProductPolynomialFactory(polyCol, enumerateFunction)
# degree 1
p = 1
indexMax = enumerateFunction.getStrataCumulatedCardinal(p)
adaptiveStrategy = ot.FixedStrategy(multivariateBasis, indexMax)
```

```
PODnew.setAdaptiveStrategy(adaptiveStrategy)
```

```
# define the projection strategy
projectionStrategy = ot.LeastSquaresStrategy()
PODnew.setProjectionStrategy(projectionStrategy)
```

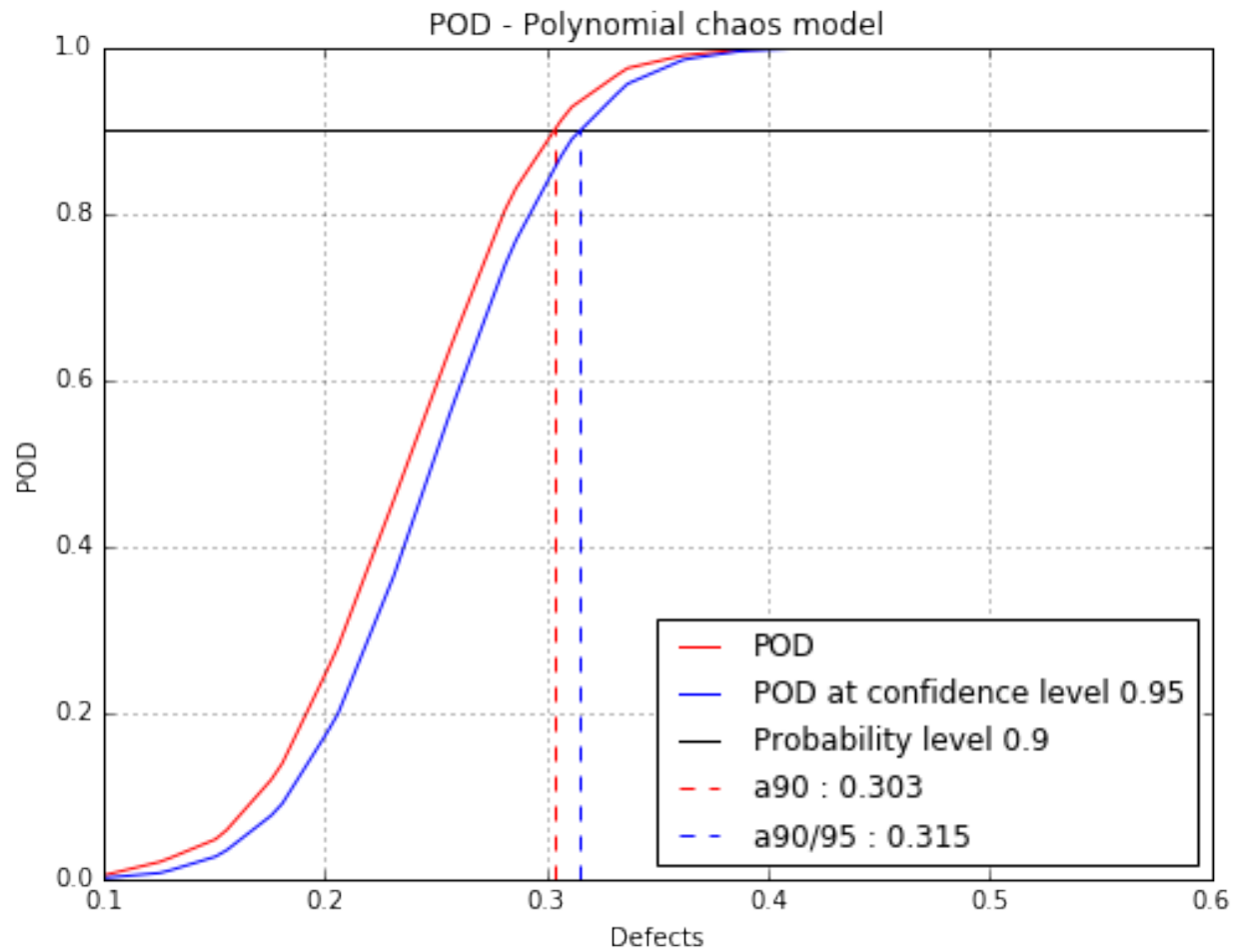
```
PODnew.run()
```

```
Start build polynomial chaos model...
Polynomial chaos model completed
R2 : 0.8947
Q2 : 0.8914
Computing POD per defect: [=====] 100.00
↪ % Done
```

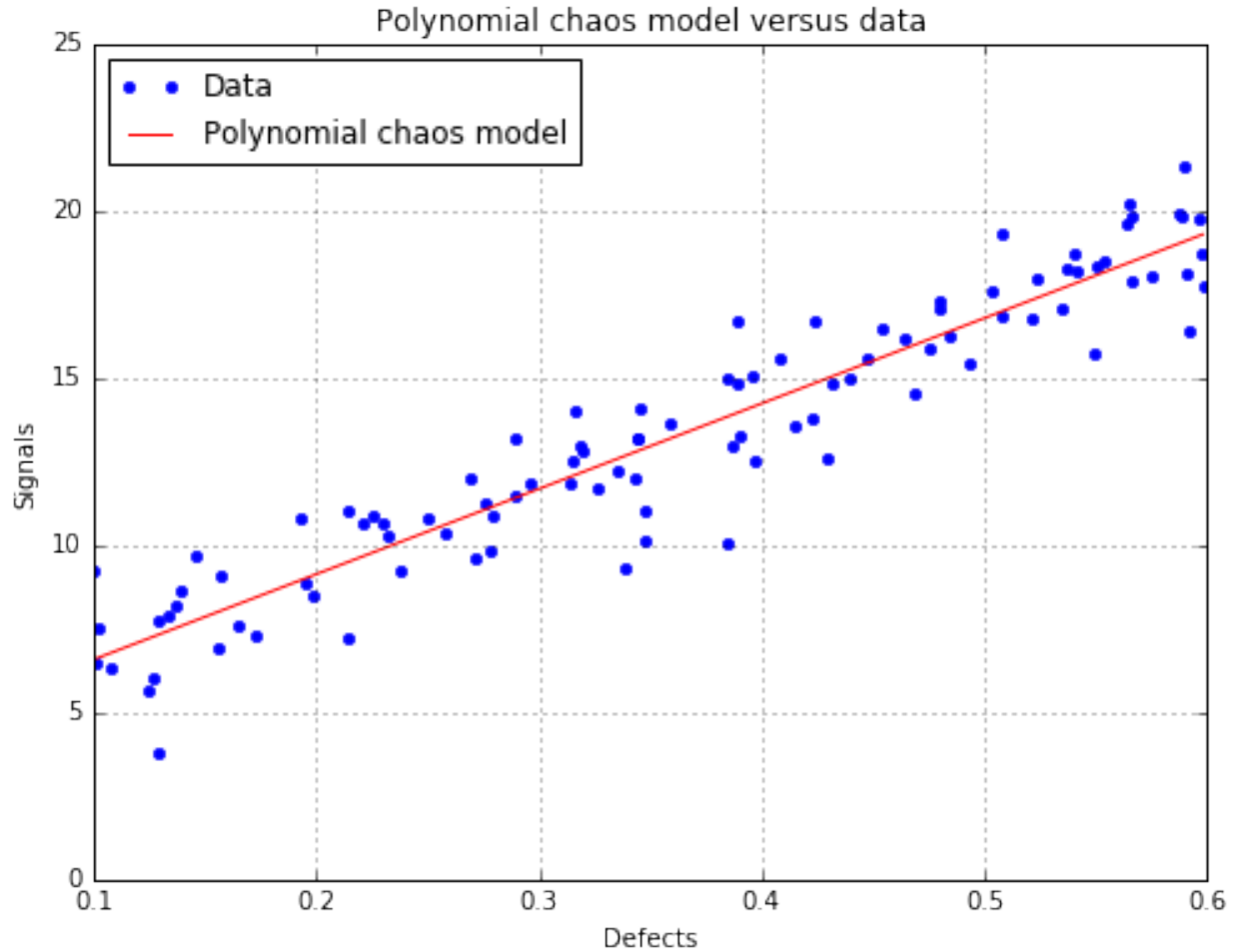
```
print PODnew.computeDetectionSize(0.9, 0.95)
print 'R2 : {:.4f}'.format(POD.getR2())
print 'Q2 : {:.4f}'.format(POD.getQ2())
```

```
[a90 : 0.303335, a90/95 : 0.315078]
R2 : 0.8947
Q2 : 0.8914
```

```
fig, ax = PODnew.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95)
fig.show()
```



```
fig, ax = PODnew.drawPolynomialChaosModel()  
fig.show()
```



ipy nb source code

## 1.2.7 Kriging POD

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
```

### Generate data

```
inputSample = ot.NumericalSample(
    [[4.59626812e+00, 7.46143339e-02, 1.02231538e+00, 8.60042277e+01],
     [4.14315790e+00, 4.20801346e-02, 1.05874908e+00, 2.65757364e+01],
     [4.76735111e+00, 3.72414824e-02, 1.05730385e+00, 5.76058433e+01],
     [4.82811977e+00, 2.49997658e-02, 1.06954641e+00, 2.54461380e+01],
     [4.48961094e+00, 3.74562922e-02, 1.04943946e+00, 6.19483646e+00],
     [5.05605334e+00, 4.87599783e-02, 1.06520409e+00, 3.39024904e+00],
     [5.69679328e+00, 7.74915877e-02, 1.04099514e+00, 6.50990466e+01],
     [5.10193991e+00, 4.35520544e-02, 1.02502536e+00, 5.51492592e+01],
```

```

[4.04791970e+00, 2.38565932e-02, 1.01906882e+00, 2.07875350e+01],
[4.66238956e+00, 5.49901237e-02, 1.02427200e+00, 1.45661275e+01],
[4.86634219e+00, 6.04693570e-02, 1.08199374e+00, 1.05104730e+00],
[4.13519347e+00, 4.45225831e-02, 1.01900124e+00, 5.10117047e+01],
[4.92541940e+00, 7.87692335e-02, 9.91868726e-01, 8.32302238e+01],
[4.70722074e+00, 6.51799251e-02, 1.10608515e+00, 3.30181002e+01],
[4.29040932e+00, 1.75426222e-02, 9.75678838e-01, 2.28186756e+01],
[4.89291400e+00, 2.34997929e-02, 1.07669835e+00, 5.38926138e+01],
[4.44653744e+00, 7.63175936e-02, 1.06979154e+00, 5.19109415e+01],
[3.99977452e+00, 5.80430585e-02, 1.01850716e+00, 7.61988190e+01],
[3.95491570e+00, 1.09302814e-02, 1.03687664e+00, 6.09981789e+01],
[5.16424368e+00, 2.69026464e-02, 1.06673711e+00, 2.88708887e+01],
[5.30491620e+00, 4.53802273e-02, 1.06254792e+00, 3.03856837e+01],
[4.92809155e+00, 1.20616369e-02, 1.00700410e+00, 7.02512744e+00],
[4.68373805e+00, 6.26028935e-02, 1.05152117e+00, 4.81271603e+01],
[5.32381954e+00, 4.33013582e-02, 9.90522007e-01, 6.56015973e+01],
[4.35455857e+00, 1.23814619e-02, 1.01810539e+00, 1.10769534e+01]])

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.
↪609524],
    [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469
↪],
    [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.
↪768369],
    [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.
↪636588],
    [ 40.643948]])

```

## Build POD with Kriging model

```

# signal detection threshold
detection = 38.
# The POD with censored data actually builds a POD only on filtered data.
# A warning is displayed in this case.
POD = otpod.KrigingPOD(inputSample, signals, detection,
                        noiseThres=35., saturationThres=45.)

```

```

INFO:root:Censored data are not taken into account : the kriging model is only built
↪on filtered data.

```

## User-defined defect sizes

The user-defined defect sizes must range between the minimum and maximum of the defect values after filtering. An error is raised if it is not the case. The available range is then returned to the user.

```

# Default defect sizes
print 'Default defect sizes : '
print POD.getDefectSizes()

# Wrong range
POD.setDefectSizes([3.2, 3.6, 4.5, 5.5])

```



```

Default defect sizes :
[ 3.9549157  4.0152854  4.07565509  4.13602479  4.19639448  4.25676418
  4.31713387  4.37750357  4.43787326  4.49824296  4.55861265  4.61898235
  4.67935204  4.73972174  4.80009143  4.86046113  4.92083082  4.98120052
  5.04157021  5.10193991]

```

```

-----
ValueError                                Traceback (most recent call last)

<ipython-input-4-af50a2a6fa25> in <module>()
      4
      5 # Wrong range
----> 6 POD.setDefectSizes([3.2, 3.6, 4.5, 5.5])

/home/dumas/projet/ByPASS_pmpr635/otpod/otpod/_kriging_tools.pyc in
-> setDefectSizes(self, size)
    230         raise ValueError('Defect sizes must range between ' + \
    231                          '{:0.4f}'.format(np.ceil(minMin*10000)/10000) +
-> \
--> 232                          'and {:0.4f}'.format(np.floor(maxMax*10000)/
-> 10000))
    233         self._defectNumber = self._defectSizes.shape[0]
    234

ValueError: Defect sizes must range between 3.9550 and 5.1019.

```

```

# Good range
POD.setDefectSizes([4., 4.3, 4.6, 4.9, 5.1])
print 'User-defined defect size : '
print POD.getDefectSizes()

```

## Running the Kriging based POD

The computing time can be reduced by setting the simulation size attribute to another value. However the confidence interval is less accurate.

The sampling size is the number of the samples used to compute the POD with the Monte Carlo simulation for each defect sizes.

A progress is displayed, which can be disabled with the method `setVerbose`.

```

POD = otpod.KrigingPOD(inputSample, signals, detection)
# we can change the number of initial random search for the best starting point
# of the TNC algorithm which optimizes the covariance model parameters
POD.setInitialStartSize(500) # default is 1000
# we can change the sample size of the Monte Carlo simulation
POD.setSamplingSize(2000) # default is 5000
# we can also change the size of the simulation to compute the confidence interval
POD.setSimulationSize(500) # default is 1000
%time POD.run()

```

```
Start optimizing covariance model parameters...
Kriging optimizer completed
Q2 : 1.0000
Computing POD per defect: [=====] 100.00
↪ % Done
CPU times: user 31.9 s, sys: 1.83 s, total: 33.7 s
Wall time: 20.2 s
```

### Compute detection size

```
# Detection size at probability level 0.9
# and confidence level 0.95
print POD.computeDetectionSize(0.9, 0.95)

# probability level 0.95 with confidence level 0.99
print POD.computeDetectionSize(0.95, 0.99)
```

```
[a90 : 4.62384, a90/95 : 4.63254]
[a95 : 4.66756, a95/99 : 4.6761]
```

### get POD NumericalMathFunction

```
# get the POD model
PODmodel = POD.getPODModel()
# get the POD model at the given confidence level
PODmodelCl95 = POD.getPODCLModel(0.95)

# compute the probability of detection for a given defect value
print 'POD : {:.3f}'.format(PODmodel([4.2])[0])
print 'POD at level 0.95 : {:.3f}'.format(PODmodelCl95([4.2])[0])
```

```
POD : 0.148
POD at level 0.95 : 0.136
```

### Compute the Q2

Enable to check the quality of the model.

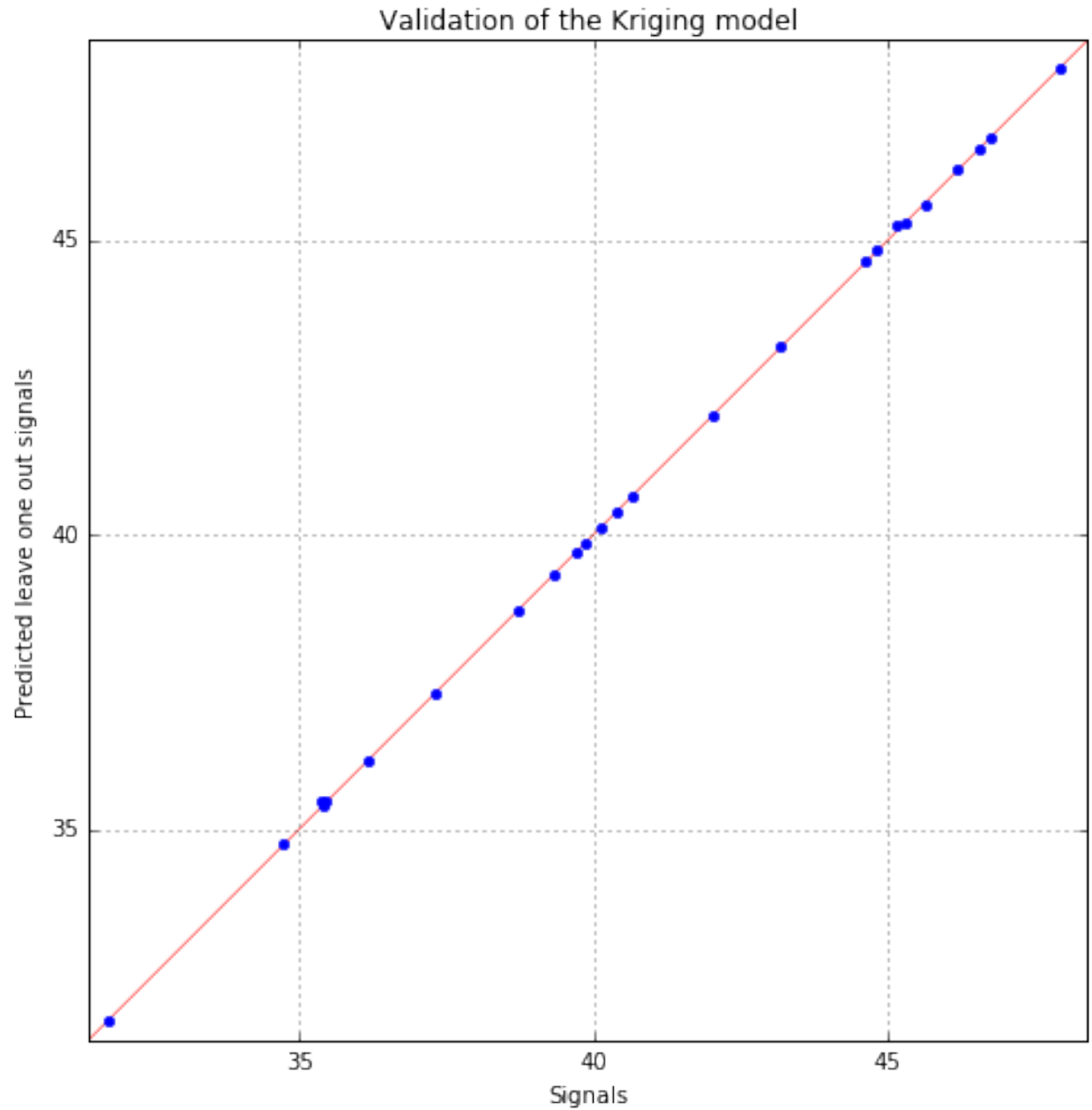
```
print 'Q2 : {:.4f}'.format(POD.getQ2())
```

```
Q2 : 1.0000
```

### Draw the validation graph

The predictions are the one computed by leave one out.

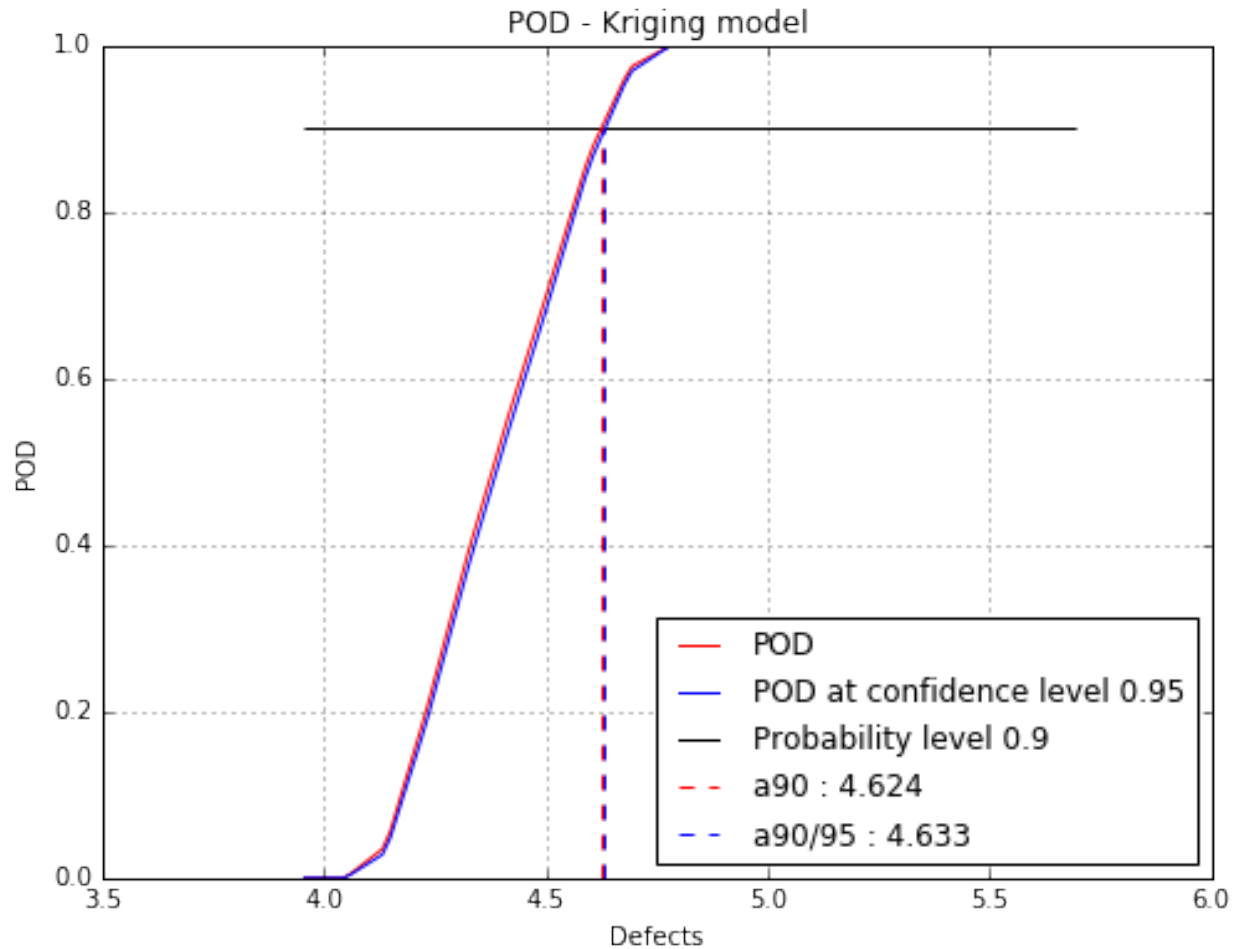
```
fig, ax = POD.drawValidationGraph()
fig.show()
```



### Show POD graphs

Mean POD and POD at confidence level with the detection size for a given probability level

```
fig, ax = POD.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95,
                       name='figure/PODKriging.png')
# The figure is saved in PODPolyChaos.png
fig.show()
```



### Advanced user mode

The user can defined one or both parameters of the kriging algorithm : - the basis - the covariance model

The user can also defined the input parameter distribution it is known.

The user can set the KrigingResult object if it built from other data.

```
# new POD study
PODnew = otpod.KrigingPOD(inputSample, signals, detection)
```

```
# set the basis constant
basis = ot.ConstantBasisFactory(4).build()
PODnew.setBasis(basis)
```

```
# set the covariance Model as an absolute exponential model
covColl = ot.CovarianceModelCollection(4)
for i in xrange(4):
    covColl[i] = ot.AbsoluteExponential(1, 1.)
covarianceModel = ot.ProductCovarianceModel(covColl)

PODnew.setCovarianceModel(covarianceModel)
```

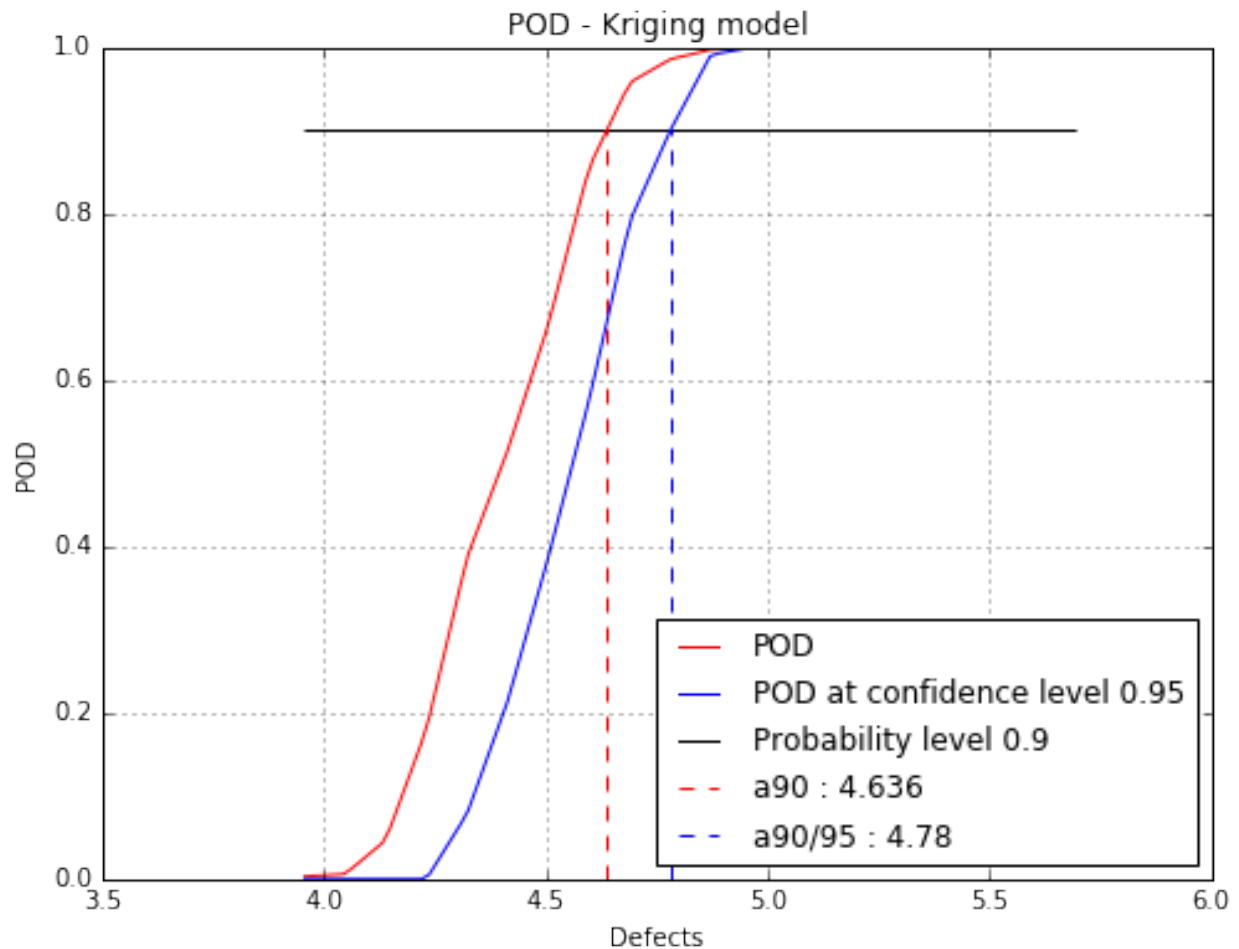
```
PODnew.run()
```

```
Start optimizing covariance model parameters...
Kriging optimizer completed
Q2 : 0.9668
Computing POD per defect: [=====] 100.00
↪ % Done
```

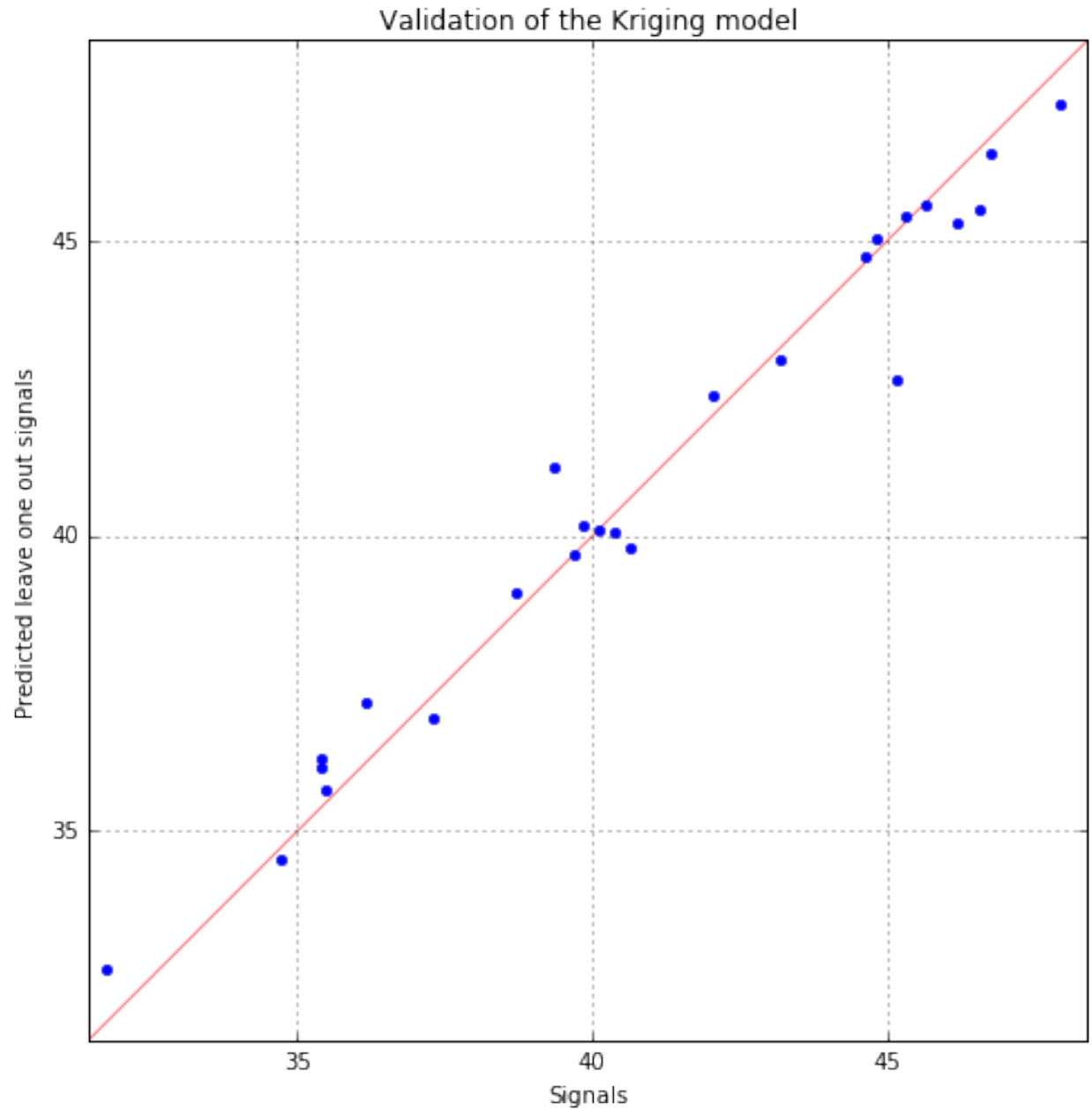
```
print PODnew.computeDetectionSize(0.9, 0.95)
print 'Q2 : {:.4f}'.format(POD.getQ2())
```

```
[a90 : 4.63637, a90/95 : 4.77956]
Q2 : 1.0000
```

```
fig, ax = PODnew.drawPOD(probabilityLevel=0.9, confidenceLevel=0.95)
fig.show()
```



```
fig, ax = PODnew.drawValidationGraph()
fig.show()
```



[ipynb source code](#)

## 1.2.8 POD Summary

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
```

## Generate data

```
inputSample = ot.NumericalSample(
    [[4.59626812e+00, 7.46143339e-02, 1.02231538e+00, 8.60042277e+01],
    [4.14315790e+00, 4.20801346e-02, 1.05874908e+00, 2.65757364e+01],
    [4.76735111e+00, 3.72414824e-02, 1.05730385e+00, 5.76058433e+01],
    [4.82811977e+00, 2.49997658e-02, 1.06954641e+00, 2.54461380e+01],
    [4.48961094e+00, 3.74562922e-02, 1.04943946e+00, 6.19483646e+00],
    [5.05605334e+00, 4.87599783e-02, 1.06520409e+00, 3.39024904e+00],
    [5.69679328e+00, 7.74915877e-02, 1.04099514e+00, 6.50990466e+01],
    [5.10193991e+00, 4.35520544e-02, 1.02502536e+00, 5.51492592e+01],
    [4.04791970e+00, 2.38565932e-02, 1.01906882e+00, 2.07875350e+01],
    [4.66238956e+00, 5.49901237e-02, 1.02427200e+00, 1.45661275e+01],
    [4.86634219e+00, 6.04693570e-02, 1.08199374e+00, 1.05104730e+00],
    [4.13519347e+00, 4.45225831e-02, 1.01900124e+00, 5.10117047e+01],
    [4.92541940e+00, 7.87692335e-02, 9.91868726e-01, 8.32302238e+01],
    [4.70722074e+00, 6.51799251e-02, 1.10608515e+00, 3.30181002e+01],
    [4.29040932e+00, 1.75426222e-02, 9.75678838e-01, 2.28186756e+01],
    [4.89291400e+00, 2.34997929e-02, 1.07669835e+00, 5.38926138e+01],
    [4.44653744e+00, 7.63175936e-02, 1.06979154e+00, 5.19109415e+01],
    [3.99977452e+00, 5.80430585e-02, 1.01850716e+00, 7.61988190e+01],
    [3.95491570e+00, 1.09302814e-02, 1.03687664e+00, 6.09981789e+01],
    [5.16424368e+00, 2.69026464e-02, 1.06673711e+00, 2.88708887e+01],
    [5.30491620e+00, 4.53802273e-02, 1.06254792e+00, 3.03856837e+01],
    [4.92809155e+00, 1.20616369e-02, 1.00700410e+00, 7.02512744e+00],
    [4.68373805e+00, 6.26028935e-02, 1.05152117e+00, 4.81271603e+01],
    [5.32381954e+00, 4.33013582e-02, 9.90522007e-01, 6.56015973e+01],
    [4.35455857e+00, 1.23814619e-02, 1.01810539e+00, 1.10769534e+01]])

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.
    ↪609524],
    [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469
    ↪],
    [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.
    ↪768369],
    [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.
    ↪636588],
    [ 40.643948]])
```

## Compute POD with several methods

The object POD summary enables the user to compute the POD with all available techniques. techniques can be activated or not thanks to the method *setMethodActive*. Then results can be printed or saved in a file to be compared. Moreover all graphs from the studies can be saved in a given directory.

The techniques are all activated by default : - Univariate linear model with Gaussian residuals, - Univariate linear model with no hypothesis on the residuals (Binomial), - Univariate linear model with kernel smoothing on the residuals, - Quantile regression, - Polynomial chaos, - Kriging (if input dimension > 1)

```
# signal detection threshold
detection = 38.
# The POD summary take
POD = otpod.PODSummary(inputSample, signals, detection)
# The main parameters can modified :
# The number of simulation to compute the confidence level
POD.setSimulationSize(500)
```

```
# The number of Monte Carlo simulation to compute the POD for polynomial chaos and
↳kriging
POD.setSamplingSize(2000)
# Deactivate the quantile regression technique
POD.setMethodActive('QuantileRegression', False)
# Finally run
POD.run()
```

```
Start univariate linear model analysis...

Start univariate linear model POD with Gaussian residuals...

Start univariate linear model POD with no hypothesis on the residuals...

Start univariate linear model POD with kernel smoothing on the residuals...
Computing POD (bootstrap): [=====] 100.00
↳ % Done

Start polynomial chaos POD...
Start build polynomial chaos model...
Polynomial chaos model completed
R2 : 0.9999
Q2 : 0.9987
Computing POD per defect: [=====] 100.00
↳ % Done

Start kriging POD...
Start optimizing covariance model parameters...
Kriging optimizer completed
Q2 : 0.9999
Computing POD per defect: [=====] 100.00
↳ % Done
```

## Access to the dictionary of the active methods

```
POD.getMethodActive()
```

```
{'Kriging': True,
 'LinearBinomial': True,
 'LinearGauss': True,
 'LinearKernelSmoothing': True,
 'PolynomialChaos': True,
 'QuantileRegression': False}
```

## Show results

It is shown the linear analysis results as well as the validation results of each model with the detection size computed for a given probability level and confidence level. These both values can be changed as parameters of the *printResults* method. The default values are probability level = 0.9 and confidence level = 0.95.

A warning is printed when the detection size with a technique returns an error. In this case, the return value is -1.

```
POD.printResults()
```



-----			
Linear model analysis results			
-----			
Box Cox parameter :	Not enabled		
	Uncensored		
Intercept coefficient :	0.02		
Slope coefficient :	8.71		
Standard error of the estimate :	2.29		
Confidence interval on coefficients			
Intercept coefficient :	[-10.03, 10.07]		
Slope coefficient :	[6.58, 10.85]		
Level :	0.95		
Quality of regression			
R2 (> 0.8):	0.76		
-----			
-----			
Residuals analysis results			
-----			
Fitted distribution (uncensored) :	Normal(mu = -7.10543e-15, sigma = 2.441)		
	Uncensored		
Distribution fitting test			
Kolmogorov p-value (> 0.05):	0.99		
Normality test			
Anderson Darling p-value (> 0.05):	0.76		
Cramer Von Mises p-value (> 0.05):	0.83		
Zero residual mean test			
p-value (> 0.05):	1.0		
Homoskedasticity test (constant variance)			
Breush Pagan p-value (> 0.05):	0.09		
Harrison McCabe p-value (> 0.05):	0.21		
Non autocorrelation test			
Durbin Watson p-value (> 0.05):	0.34		
-----			
-----			
Model validation results			
-----			
	Uncensored		Censored
	R2	Q2	R2
Linear Regression (> 0.8):	0.76		
Polynomial Chaos (> 0.8):	1.0	1.0	
Kriging (> 0.8):		1.0	
-----			

POD results		
	a90	a90/95
Linear Regression		
Gaussian residuals :	4.7	4.85
No residuals hypothesis :	4.71	4.88
Kernel smoothing on residuals :	4.75	4.82
Polynomial chaos :	4.63	4.65
Kriging :	4.62	4.63

Results can be displayed for another probability and confidence level.

```
POD.printResults(0.8, 0.9)
```

Linear model analysis results	
Box Cox parameter :	Not enabled
	Uncensored
Intercept coefficient :	0.02
Slope coefficient :	8.71
Standard error of the estimate :	2.29
Confidence interval on coefficients	
Intercept coefficient :	[-10.03, 10.07]
Slope coefficient :	[6.58, 10.85]
Level :	0.95
Quality of regression	
R2 (> 0.8):	0.76
Residuals analysis results	
Fitted distribution (uncensored) :	Normal(mu = -7.10543e-15, sigma = 2.2441)
	Uncensored
Distribution fitting test	
Kolmogorov p-value (> 0.05):	0.99
Normality test	
Anderson Darling p-value (> 0.05):	0.76
Cramer Von Mises p-value (> 0.05):	0.83
Zero residual mean test	
p-value (> 0.05):	1.0
Homoskedasticity test (constant variance)	
Breush Pagan p-value (> 0.05):	0.09
Harrison McCabe p-value (> 0.05):	0.21

Non autocorrelation test			
Durbin Watson p-value (> 0.05):		0.34	
-----			
-----			
Model validation results			
-----			
		Uncensored	Censored
	R2	Q2	R2
Linear Regression (> 0.8):	0.76		
Polynomial Chaos (> 0.8):	1.0	1.0	
Kriging (> 0.8):		1.0	
-----			
-----			
POD results			
-----			
		a80	a80/90
Linear Regression			
Gaussian residuals :		4.58	4.67
No residuals hypothesis :		4.64	4.68
Kernel smoothing on residuals :		4.61	4.69
Polynomial chaos :		4.55	4.57
Kriging :		4.55	4.56
-----			

## Save results

The results can be saved in a text or csv file. As for the print method, the probability level and confidence level can be specified as parameters.

```
POD.saveResults('results.csv', probabilityLevel=0.9, confidenceLevel=0.95)
```

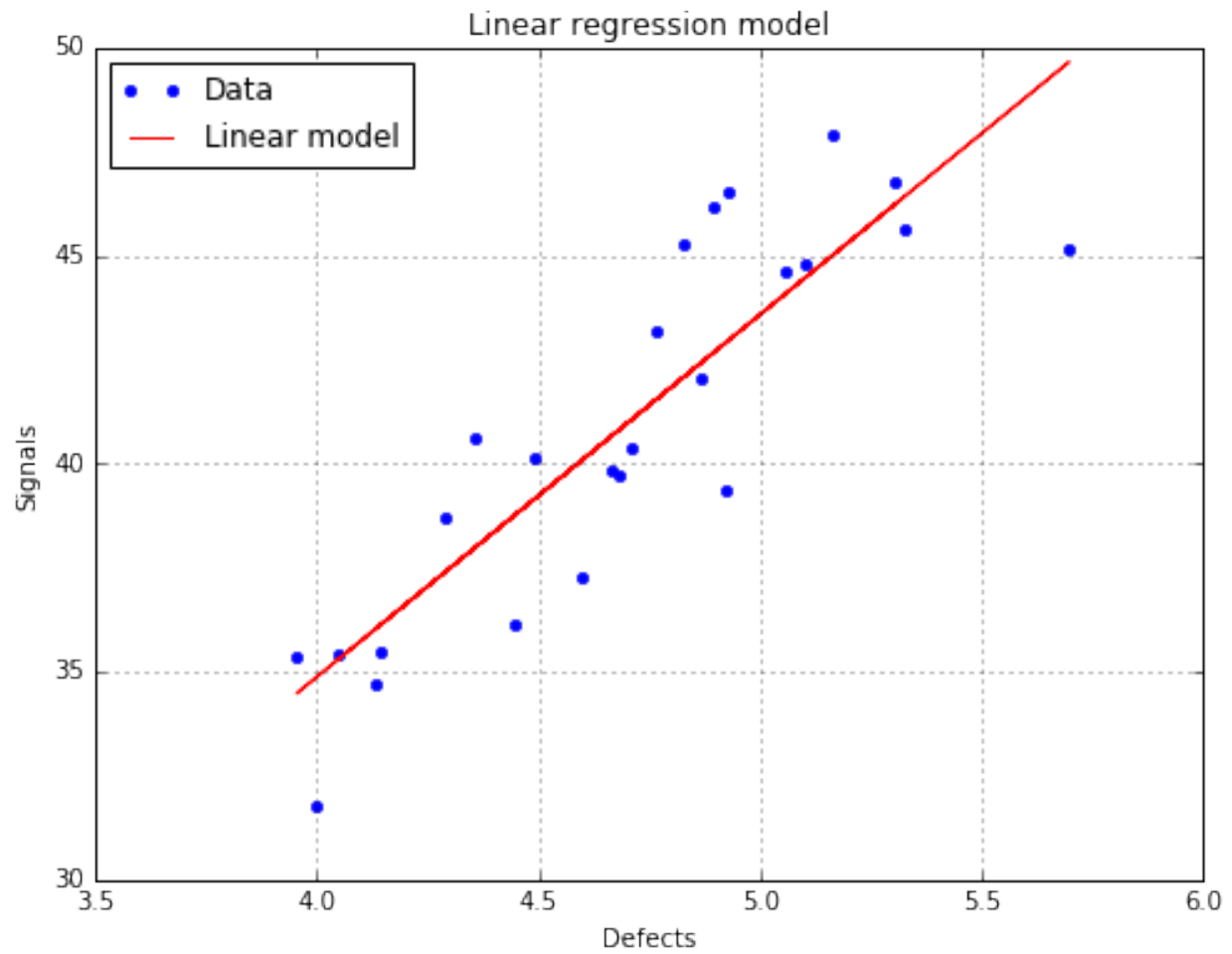
## Draw and save graphs

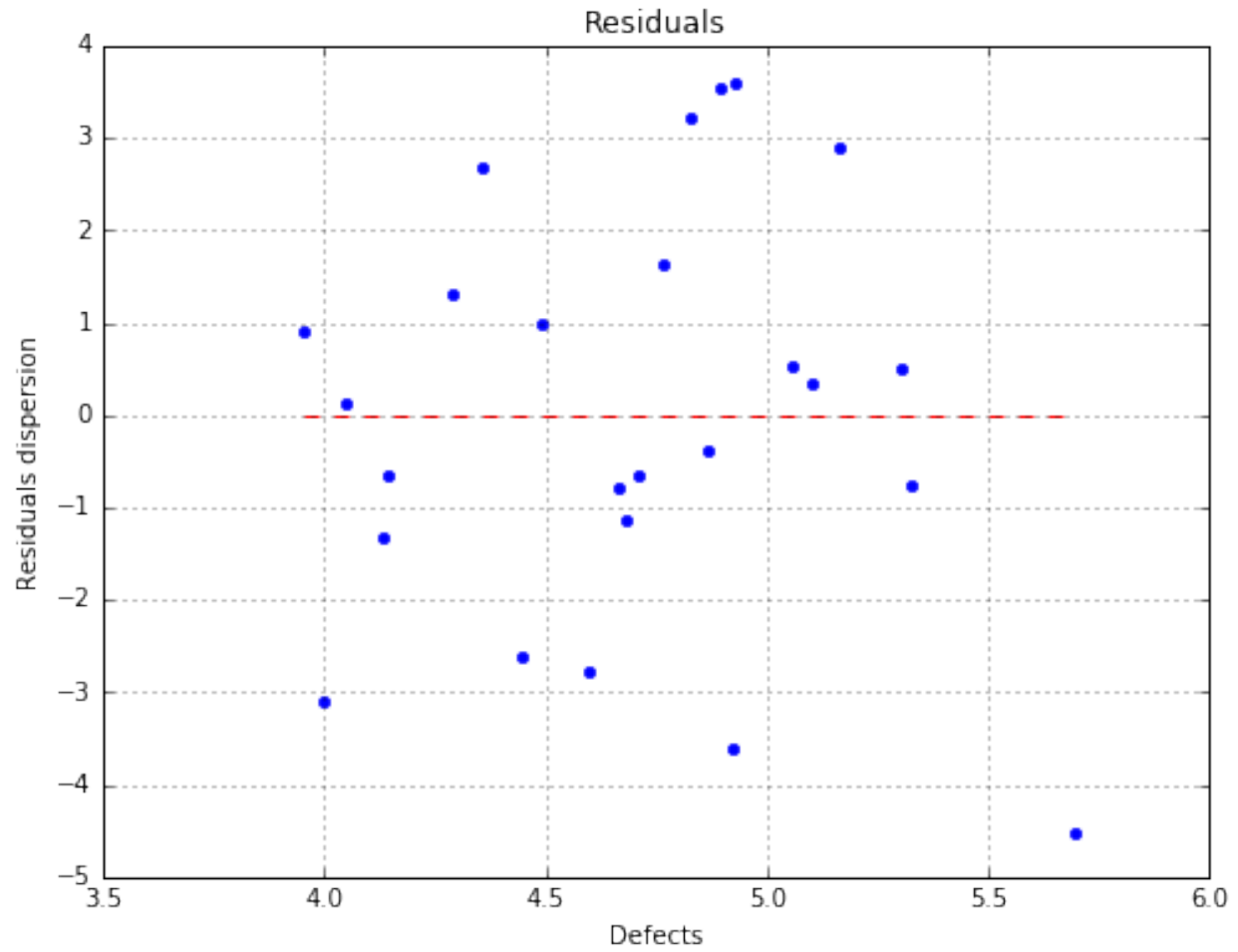
All available graphs can be saved using the method *saveGraphs*. A specific directory and the extension of the files can be given as parameters. As before the probability level and confidence level can also be chosen by the user.

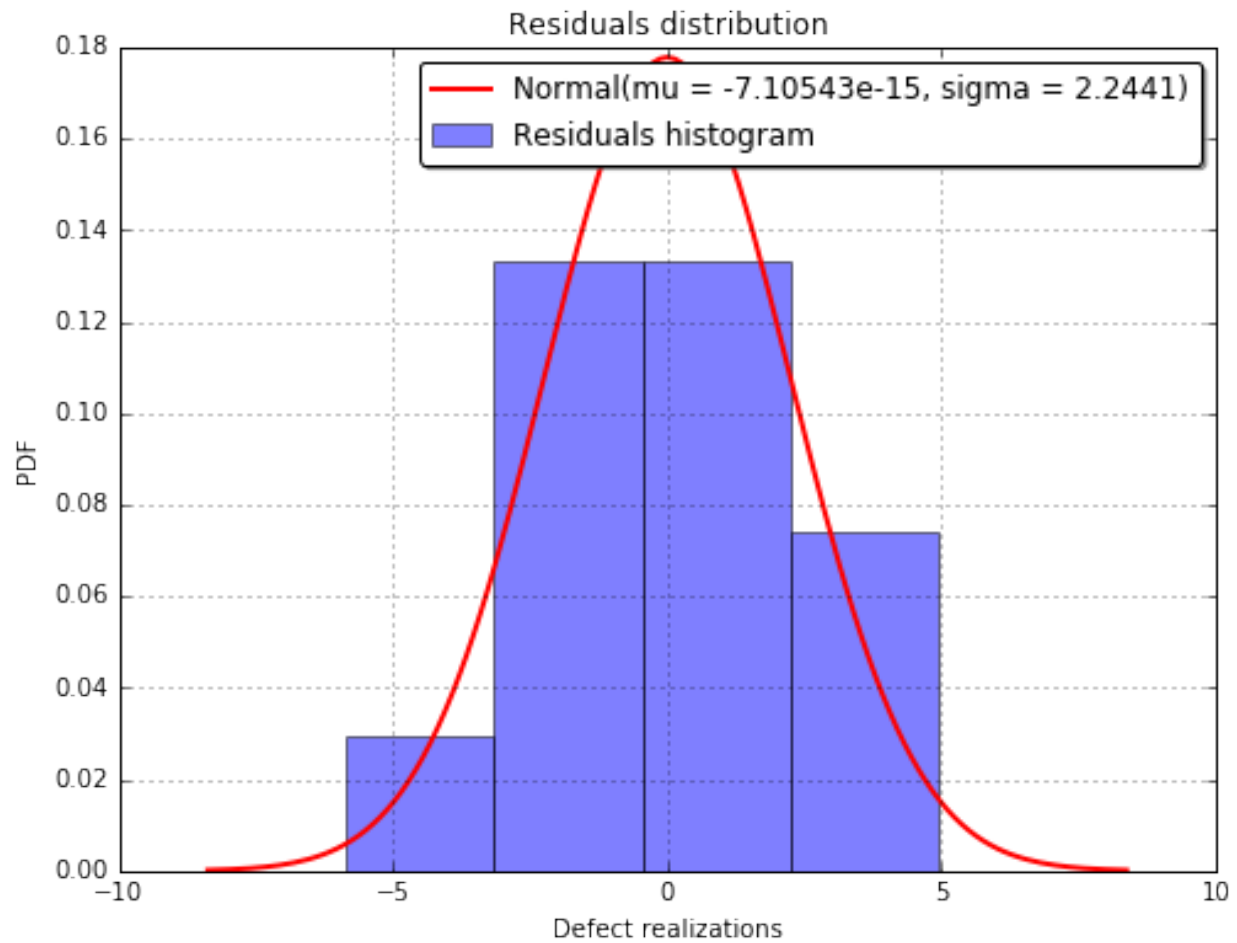
The warning is also printed here for the polynomial chaos because the detection size at the given probability level cannot be computed. A solution is to set *probabilityLevel = None*.

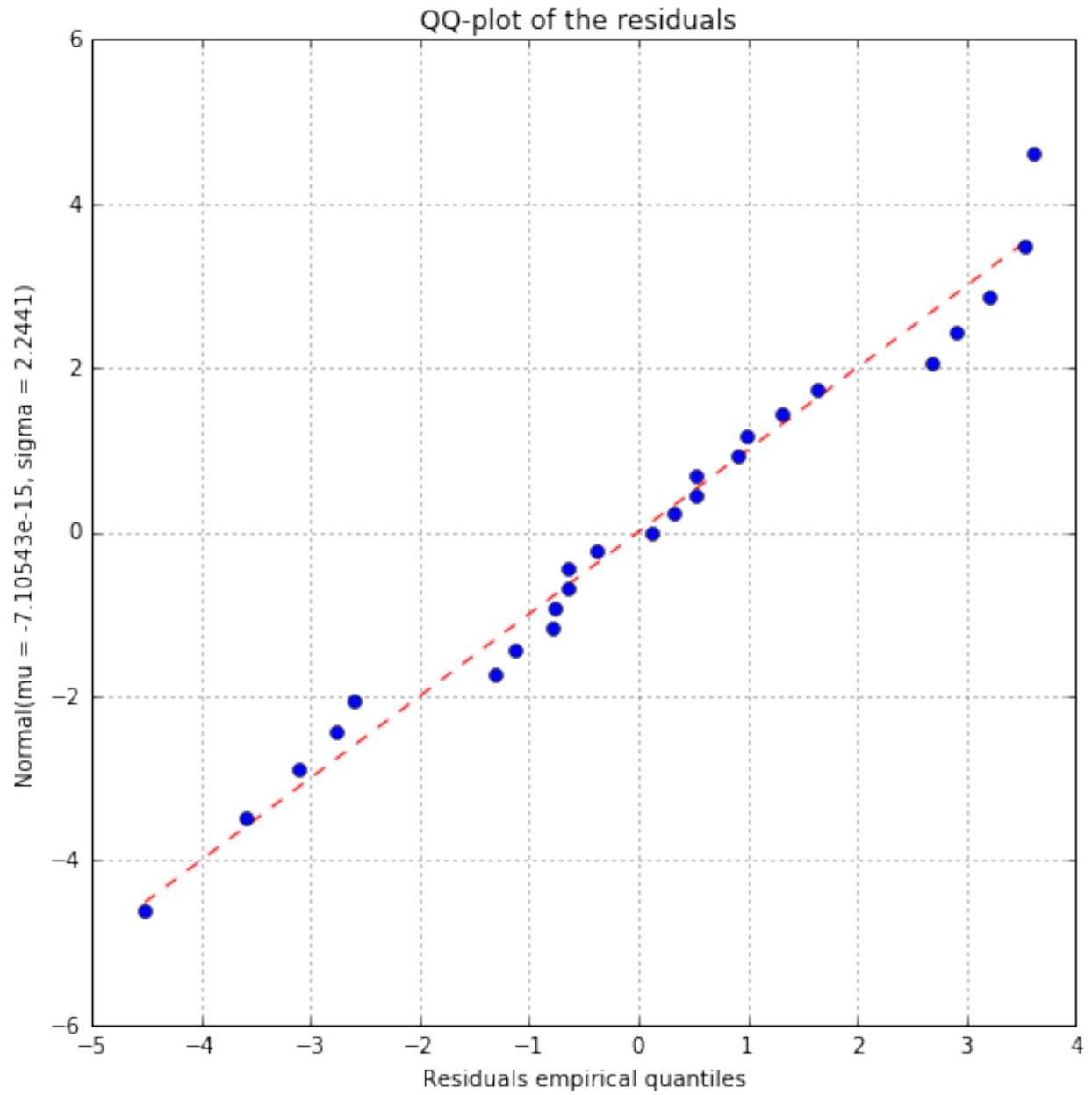
```
# return a list a figure
fig = POD.drawGraphs('./figure/', 'png', probabilityLevel=0.9, confidenceLevel=0.95)

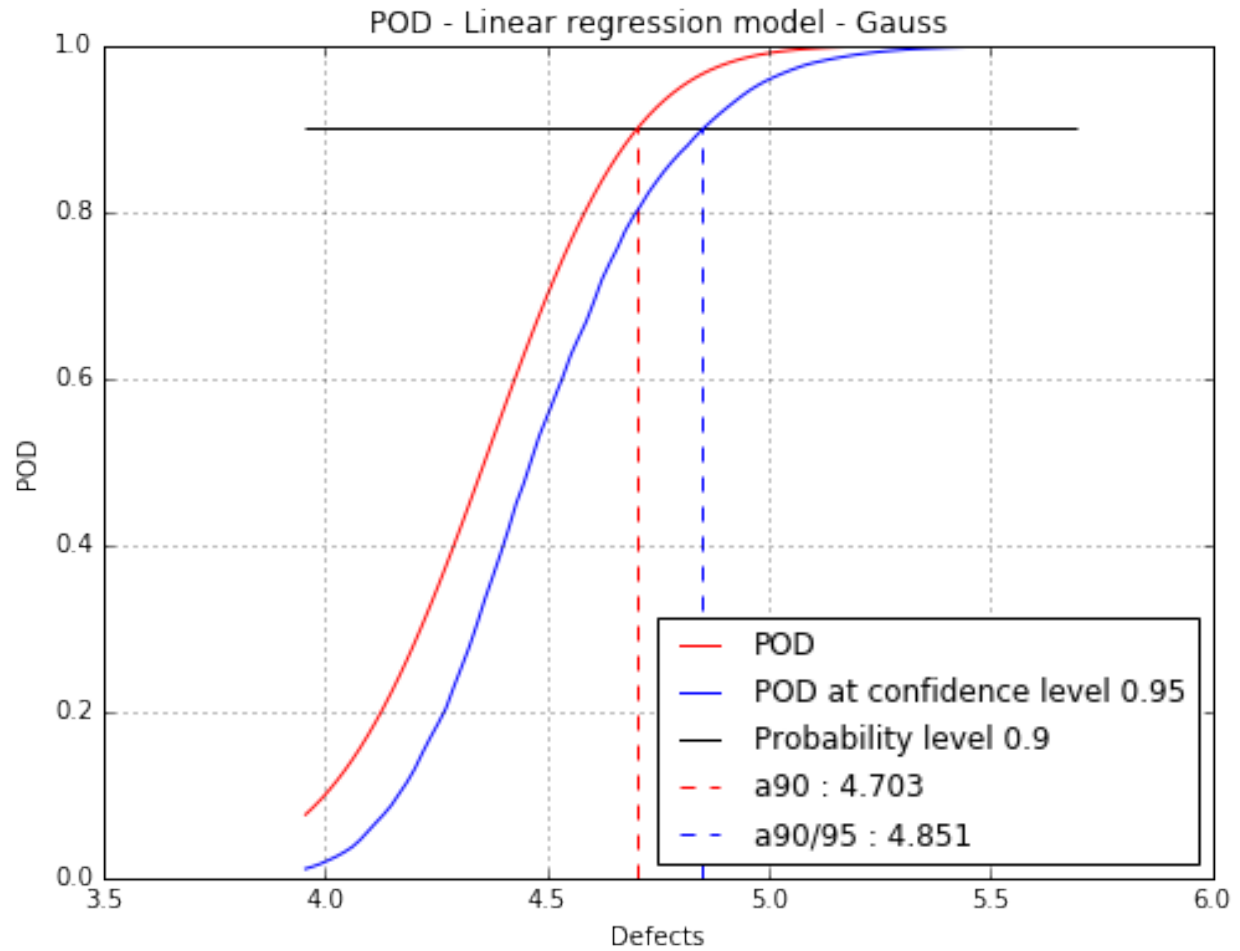
for i in xrange(len(fig)):
    fig[i].show()
```



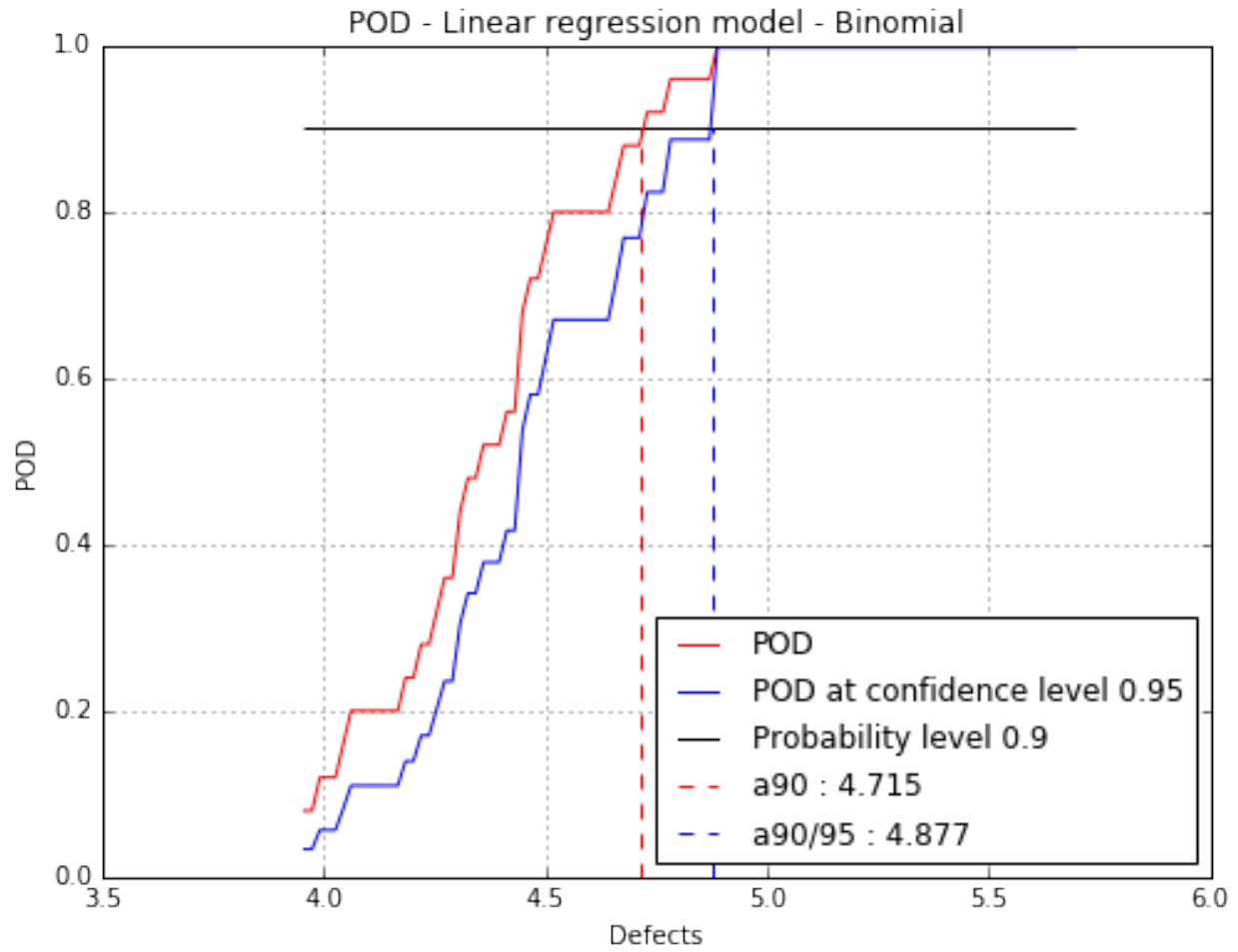


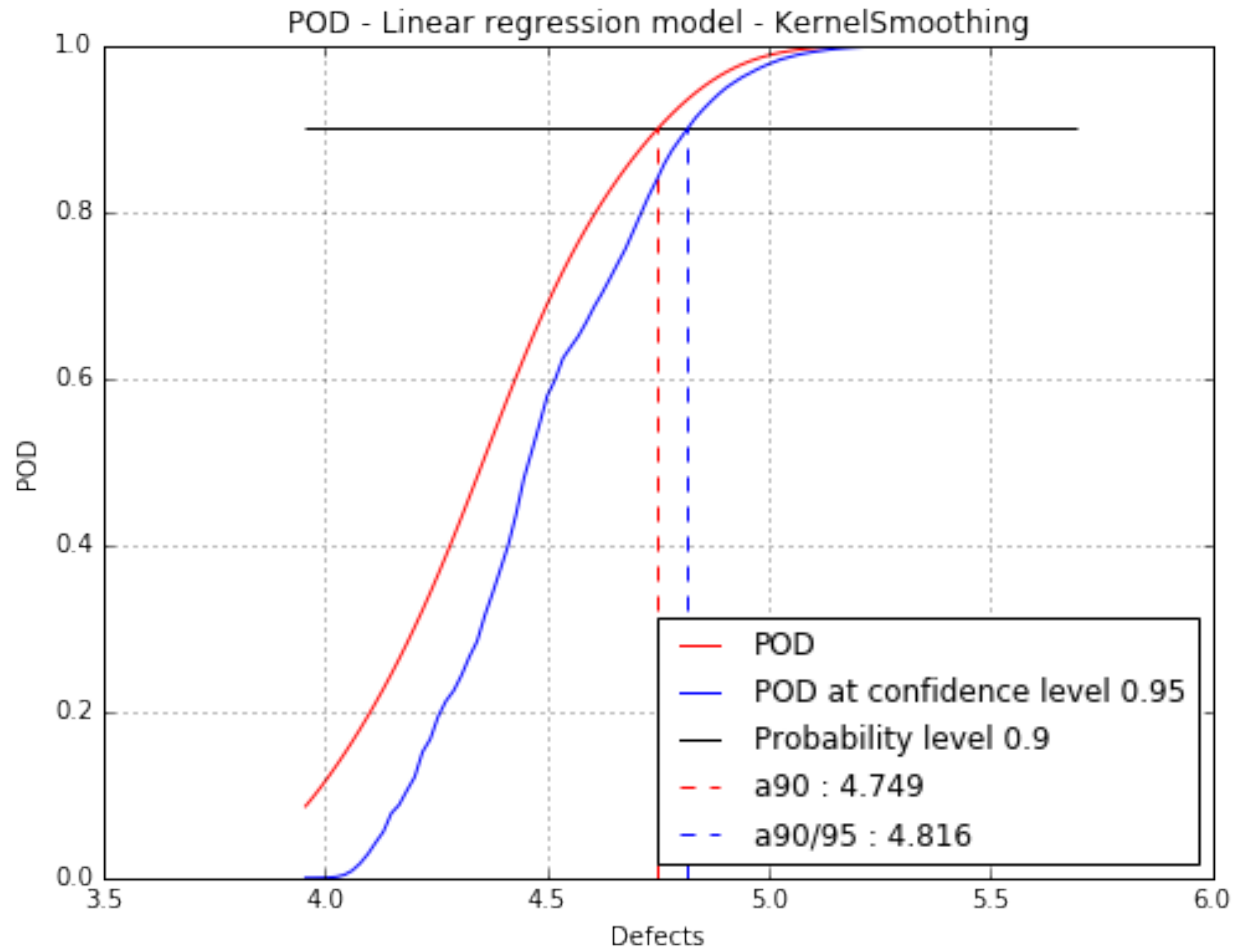


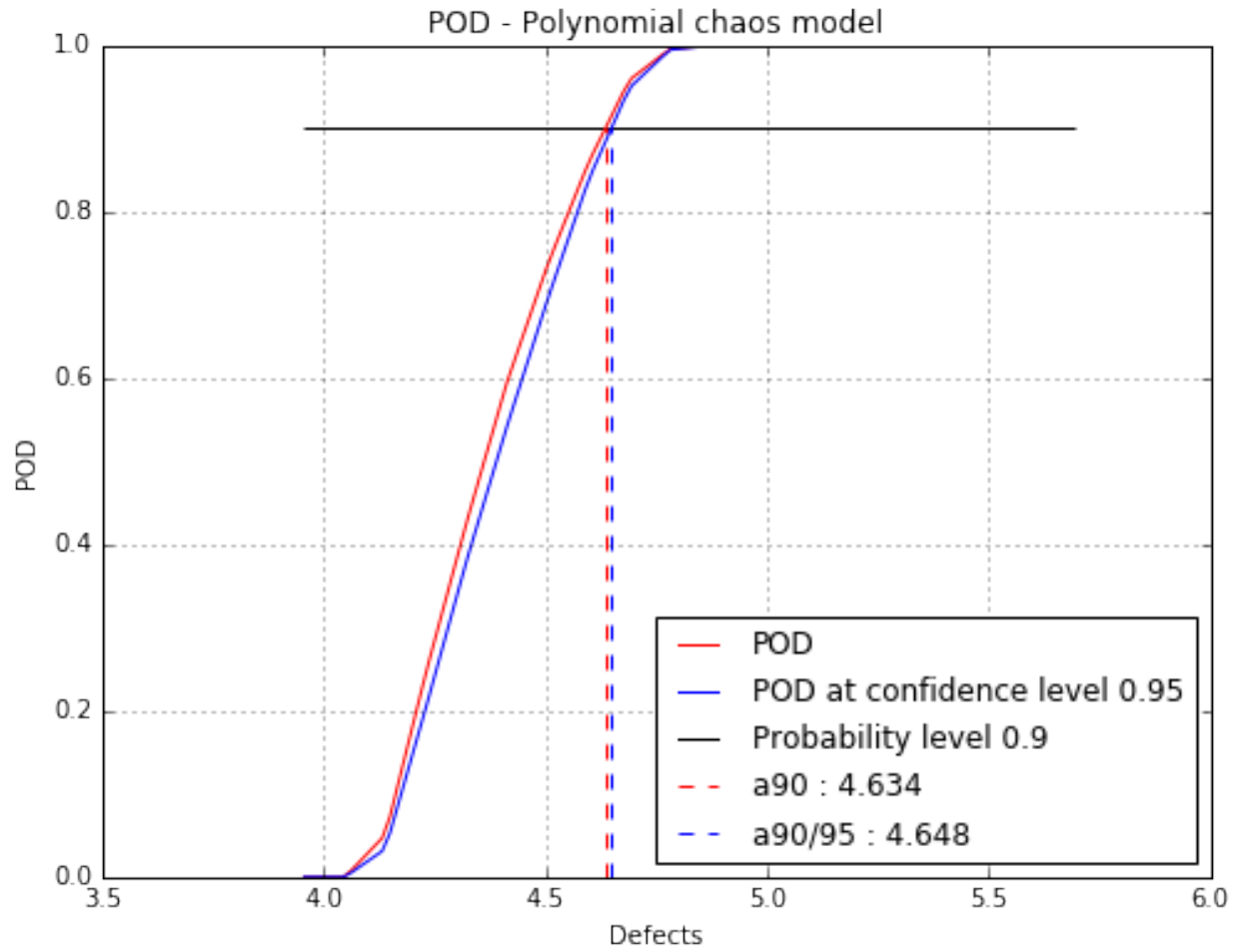


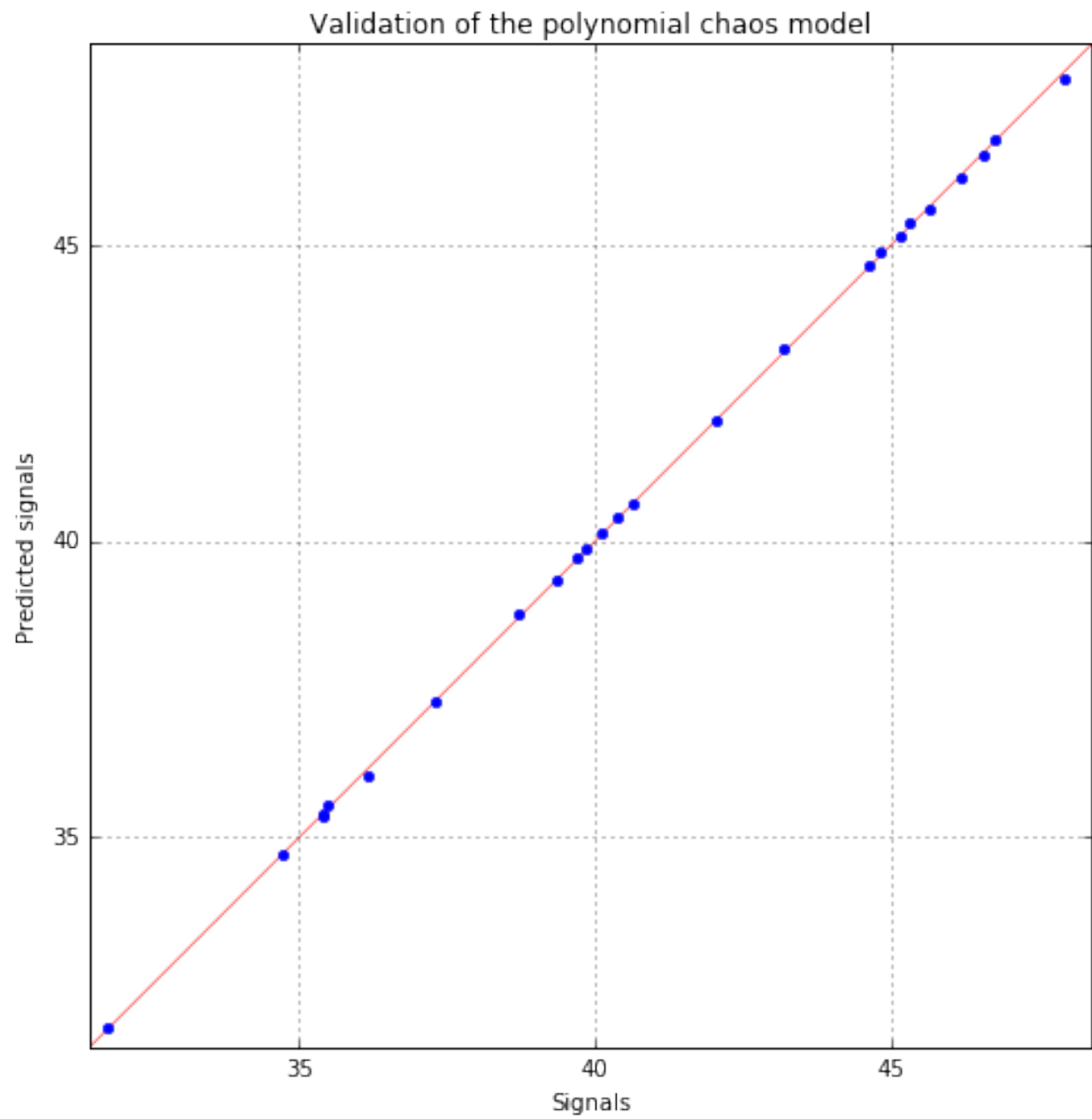


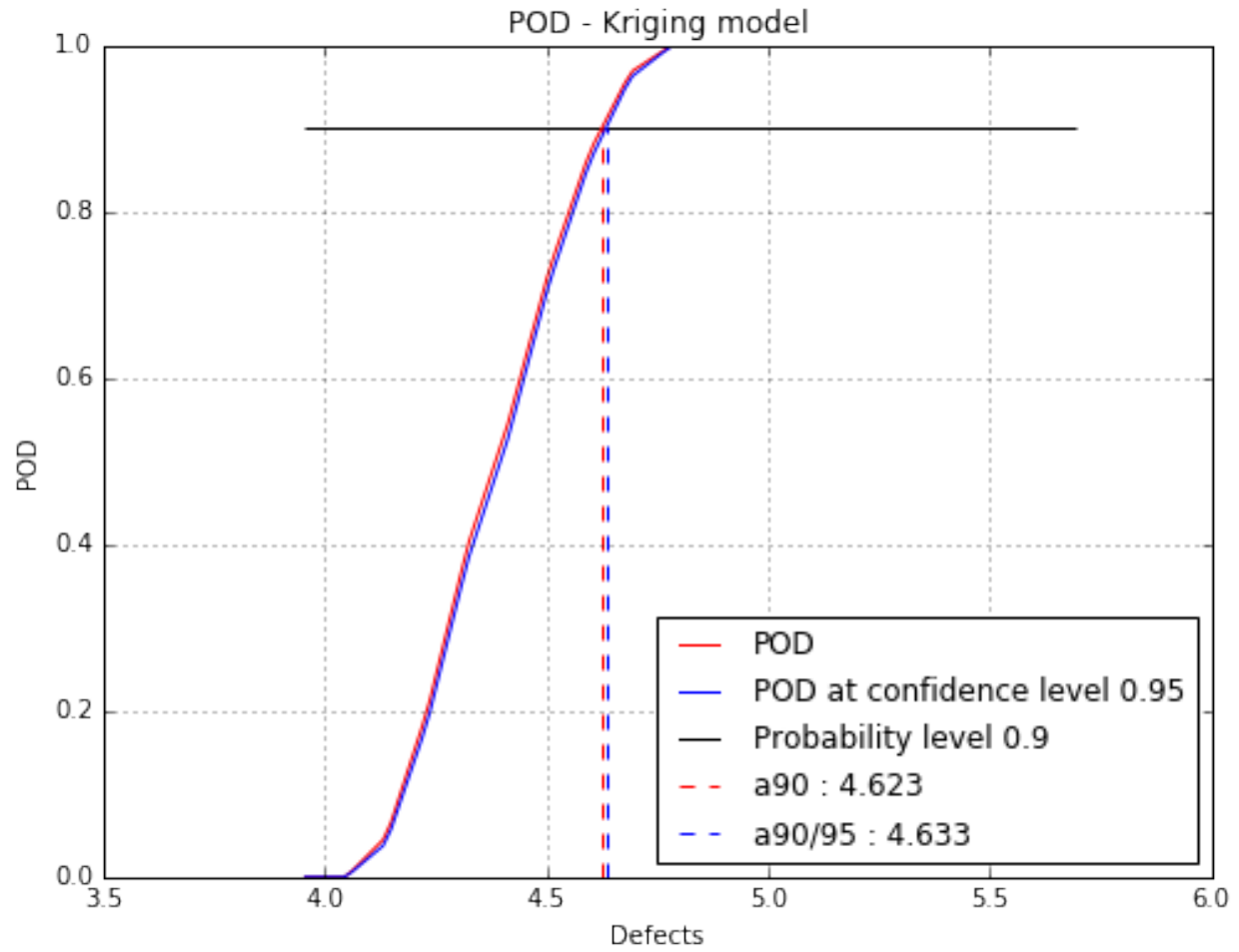


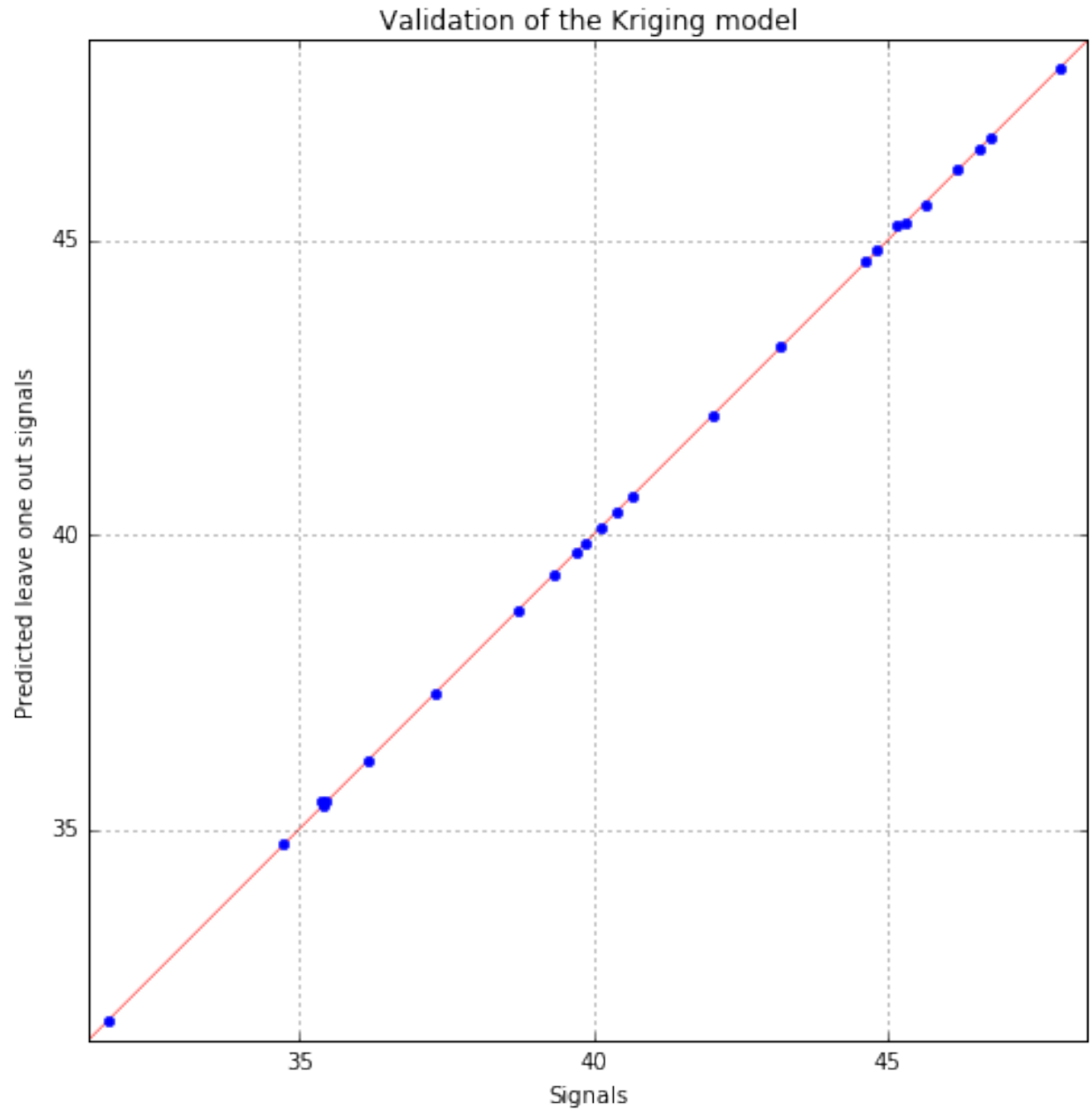












ipynb source code

### 1.2.9 Adaptive Signal POD using Kriging

```
# import relevant module
import openturns as ot
import otpod
# enable display figure in notebook
%matplotlib inline
import numpy as np
```

## Generate data

```

inputSample = ot.NumericalSample(
    [[4.59626812e+00, 7.46143339e-02, 1.02231538e+00, 8.60042277e+01],
    [4.14315790e+00, 4.20801346e-02, 1.05874908e+00, 2.65757364e+01],
    [4.76735111e+00, 3.72414824e-02, 1.05730385e+00, 5.76058433e+01],
    [4.82811977e+00, 2.49997658e-02, 1.06954641e+00, 2.54461380e+01],
    [4.48961094e+00, 3.74562922e-02, 1.04943946e+00, 6.19483646e+00],
    [5.05605334e+00, 4.87599783e-02, 1.06520409e+00, 3.39024904e+00],
    [5.69679328e+00, 7.74915877e-02, 1.04099514e+00, 6.50990466e+01],
    [5.10193991e+00, 4.35520544e-02, 1.02502536e+00, 5.51492592e+01],
    [4.04791970e+00, 2.38565932e-02, 1.01906882e+00, 2.07875350e+01],
    [4.66238956e+00, 5.49901237e-02, 1.02427200e+00, 1.45661275e+01],
    [4.86634219e+00, 6.04693570e-02, 1.08199374e+00, 1.05104730e+00],
    [4.13519347e+00, 4.45225831e-02, 1.01900124e+00, 5.10117047e+01],
    [4.92541940e+00, 7.87692335e-02, 9.91868726e-01, 8.32302238e+01],
    [4.70722074e+00, 6.51799251e-02, 1.10608515e+00, 3.30181002e+01],
    [4.29040932e+00, 1.75426222e-02, 9.75678838e-01, 2.28186756e+01],
    [4.89291400e+00, 2.34997929e-02, 1.07669835e+00, 5.38926138e+01],
    [4.44653744e+00, 7.63175936e-02, 1.06979154e+00, 5.19109415e+01],
    [3.99977452e+00, 5.80430585e-02, 1.01850716e+00, 7.61988190e+01],
    [3.95491570e+00, 1.09302814e-02, 1.03687664e+00, 6.09981789e+01],
    [5.16424368e+00, 2.69026464e-02, 1.06673711e+00, 2.88708887e+01],
    [5.30491620e+00, 4.53802273e-02, 1.06254792e+00, 3.03856837e+01],
    [4.92809155e+00, 1.20616369e-02, 1.00700410e+00, 7.02512744e+00],
    [4.68373805e+00, 6.26028935e-02, 1.05152117e+00, 4.81271603e+01],
    [5.32381954e+00, 4.33013582e-02, 9.90522007e-01, 6.56015973e+01],
    [4.35455857e+00, 1.23814619e-02, 1.01810539e+00, 1.10769534e+01]])

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.
    ↪609524],
    [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469
    ↪],
    [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.
    ↪768369],
    [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.
    ↪636588],
    [ 40.643948]])

# detection threshold
detection = 38

# Select point as initial DOE
inputDOE = inputSample[:7]
outputDOE = signals[:7]

# simulate the true physical model
basis = ot.ConstantBasisFactory(4).build()
covModel = ot.SquaredExponential(4)
krigingModel = ot.KrigingAlgorithm(inputSample, signals, basis, covModel)
TNC = ot.TNC()
TNC.setBoundConstraints(ot.Interval([0.001], [100]))
krigingModel.setOptimizer(TNC)
krigingModel.run()
physicalModel = krigingModel.getResult().getMetaModel()

```

## Create the Adaptive Signal POD with Kriging model

This method aims at improving the quality of the Kriging model where the accuracy of the computed POD is the lowest.

As this method is time consuming, it is more efficient to reduce the area of the defect size only in the most interesting part. To do that, an initial POD study can be run.

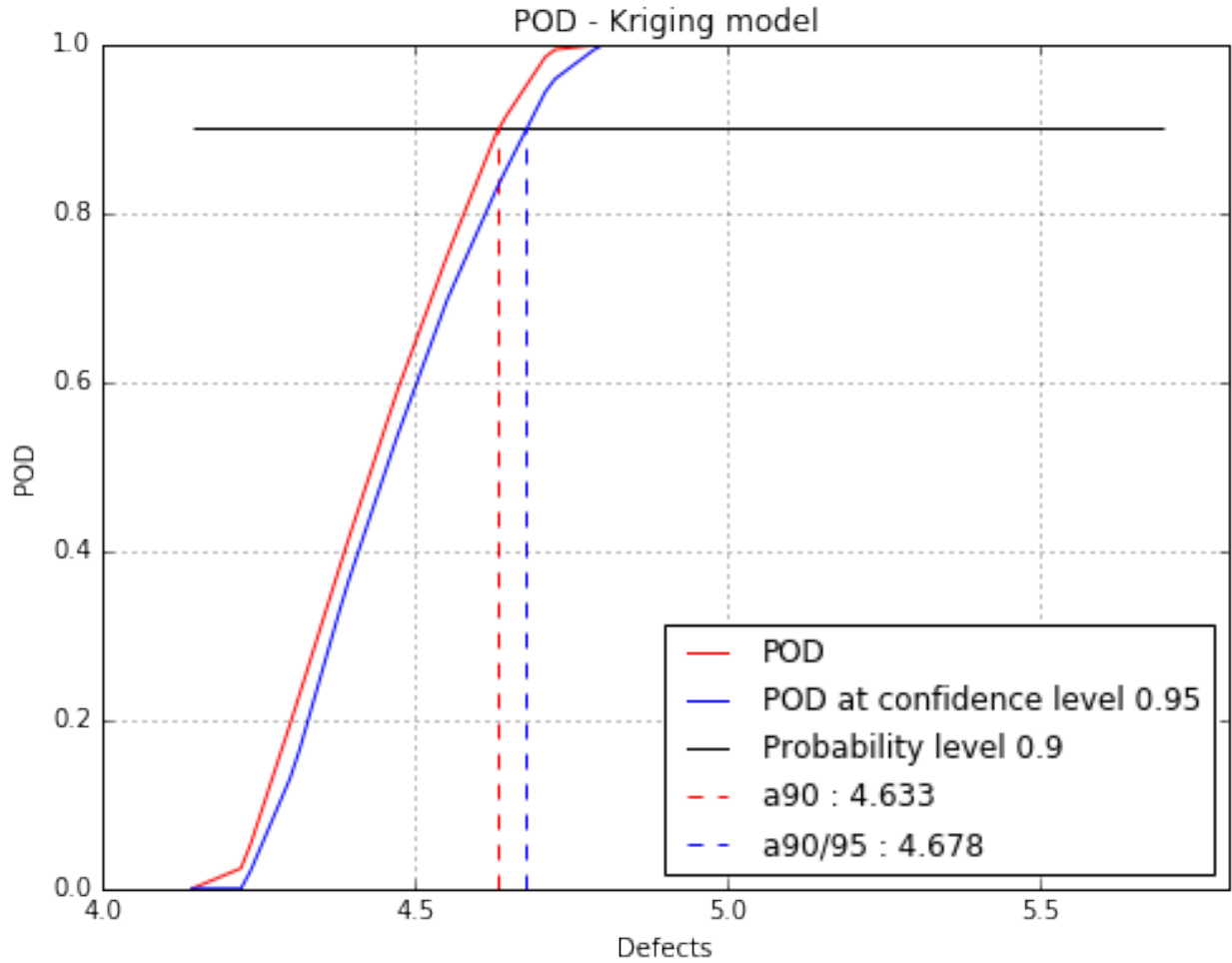
## Run an initial POD study with the kriging technique

```
initialPOD = otpod.KrigingPOD(inputDOE, outputDOE, detection)
%time initialPOD.run()
```

```
Start optimizing covariance model parameters...
Kriging optimizer completed
kriging validation Q2 (>0.9): 0.9425
Computing POD per defect: [=====] 100.00
↪ % Done
CPU times: user 1min 5s, sys: 7.32 s, total: 1min 12s
Wall time: 40.4 s
```

```
fig, ax = initialPOD.drawPOD(0.9, 0.95)
fig.show()
```





Based on this study, the interesting part for the defects ranges from 4.2 to 4.8. The adaptive signal algorithm will be then reduced to this area.

### Run the adaptive algorithm

Computing the criterion is costly so the sampling and simulation size are reduced.

```
# set the number of iterations
nIteration = 5

# Creating the adaptivePOD object
adaptivePOD = otpod.AdaptiveSignalPOD(inputDOE, outputDOE, physicalModel, nIteration,
↳detection)

# Change the range for the defect sizes between 4.2 and 4.8
adaptivePOD.setDefectSizes([4.2, 4.35, 4.5, 4.6, 4.7, 4.8])

# We can change also the number of candidate points for which the criterion is
↳computed
adaptivePOD.setCandidateSize(1000)
# we can change the sample size of the Monte Carlo simulation
adaptivePOD.setSamplingSize(500) # default is 5000
# we can also change the size of the simulation to compute the confidence interval
```

```

adaptivePOD.setSimulationSize(100) # default is 1000

# The current iteration POD graph can be displayed with multiple options :
## with or without the confidence level curve
## and with or without the intersection value at the given probability level
## if a directory is given, all graphs are saved as AdaptiveSignalPOD_i.png
adaptivePOD.setGraphActive(graphVerbose=True, probabilityLevel=0.9, confidenceLevel=0.
↪ 95,
                           directory='figure/')

%time adaptivePOD.run()

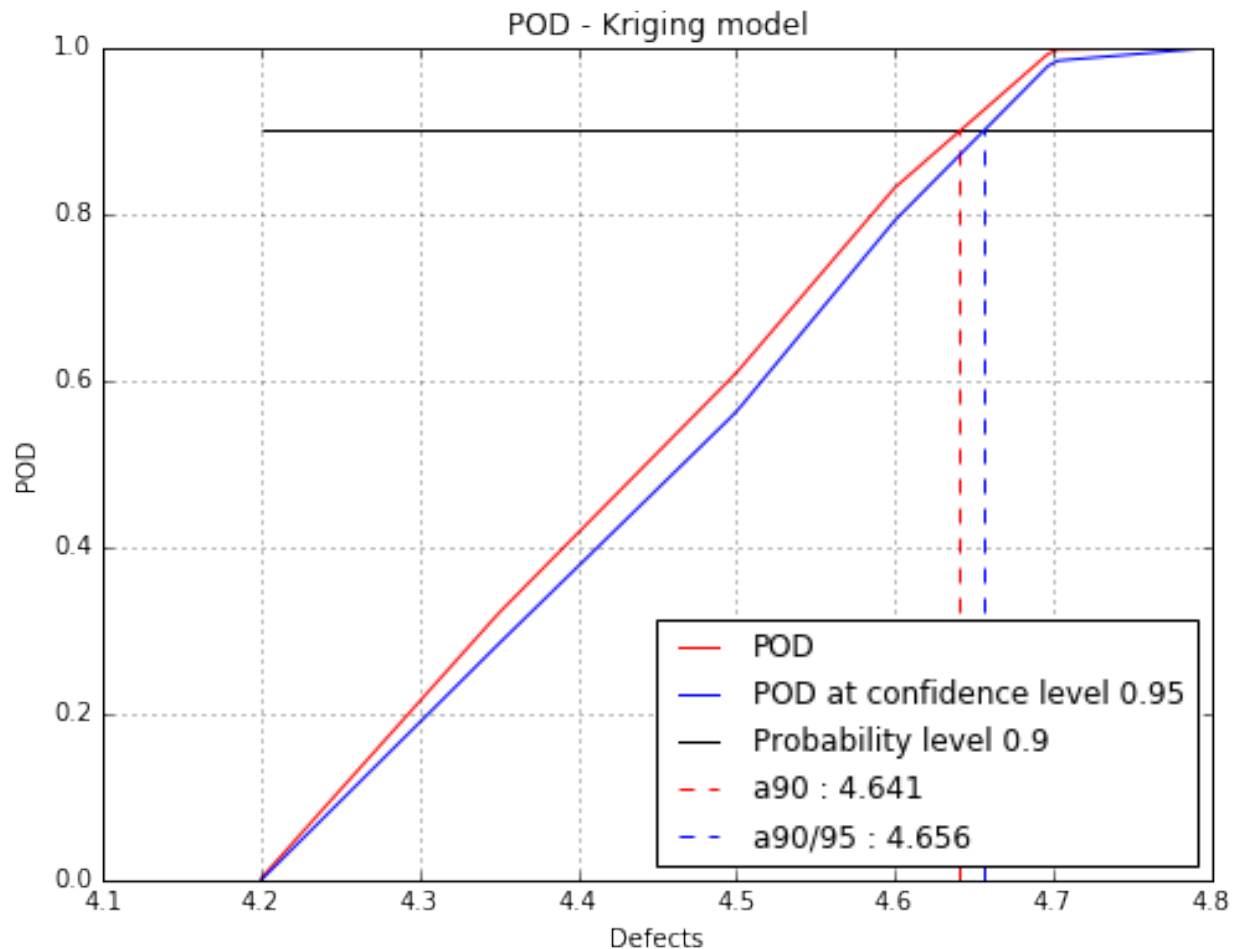
```

```

Build kriging model
Optimization of the covariance model parameters...
Kriging validation Q2 (>0.9): 0.9425

Iteration : 1/5
Computing criterion: [=====] 100.00% Done
Criterion value : 0.0190
Added point : [4.54805,0.0374051,1.03053,57.4443]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9898

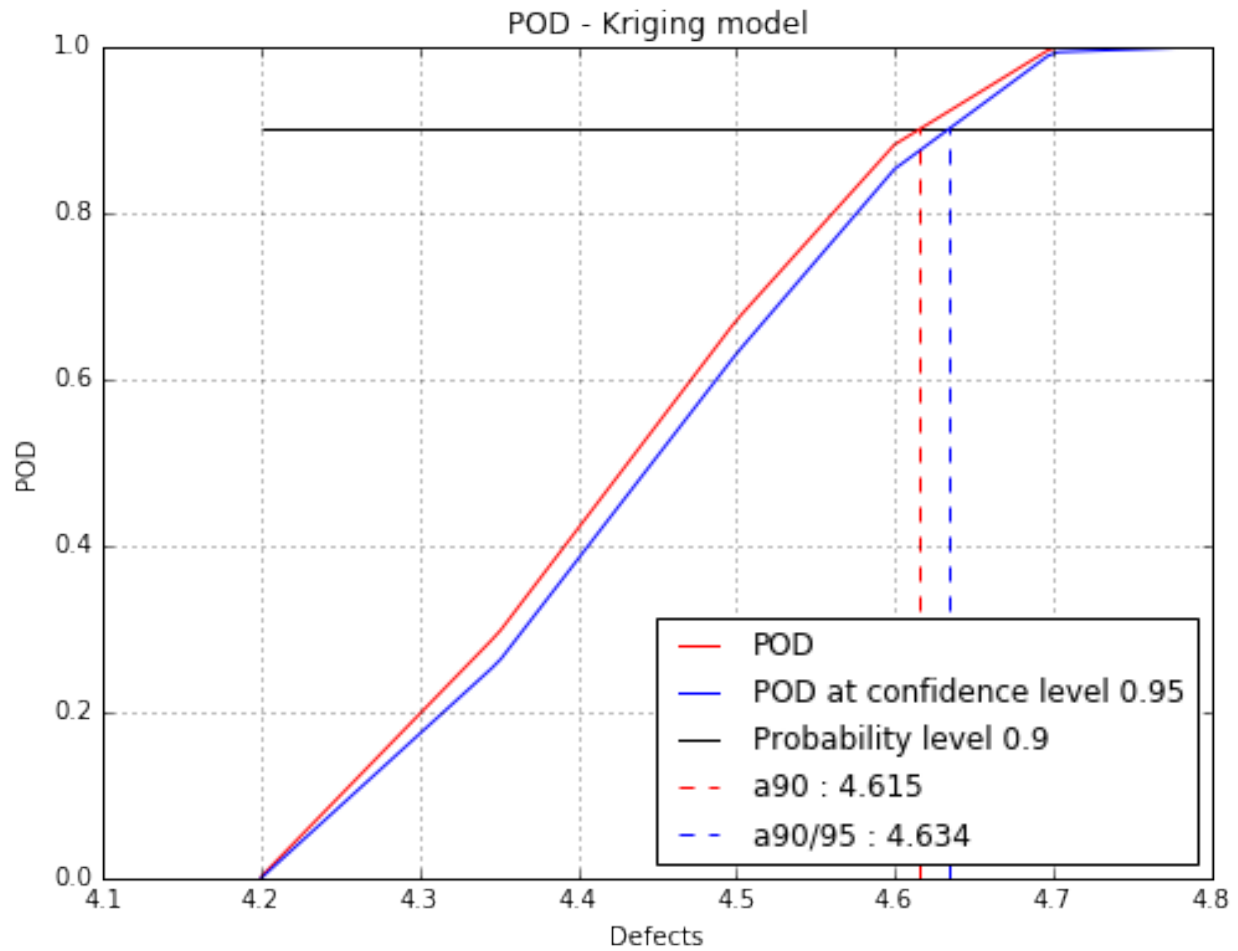
```



```

Iteration : 2/5
Computing criterion: [=====] 100.00% Done
Criterion value : 0.0158
Added point : [4.3998,0.0651376,1.06941,29.9332]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9954

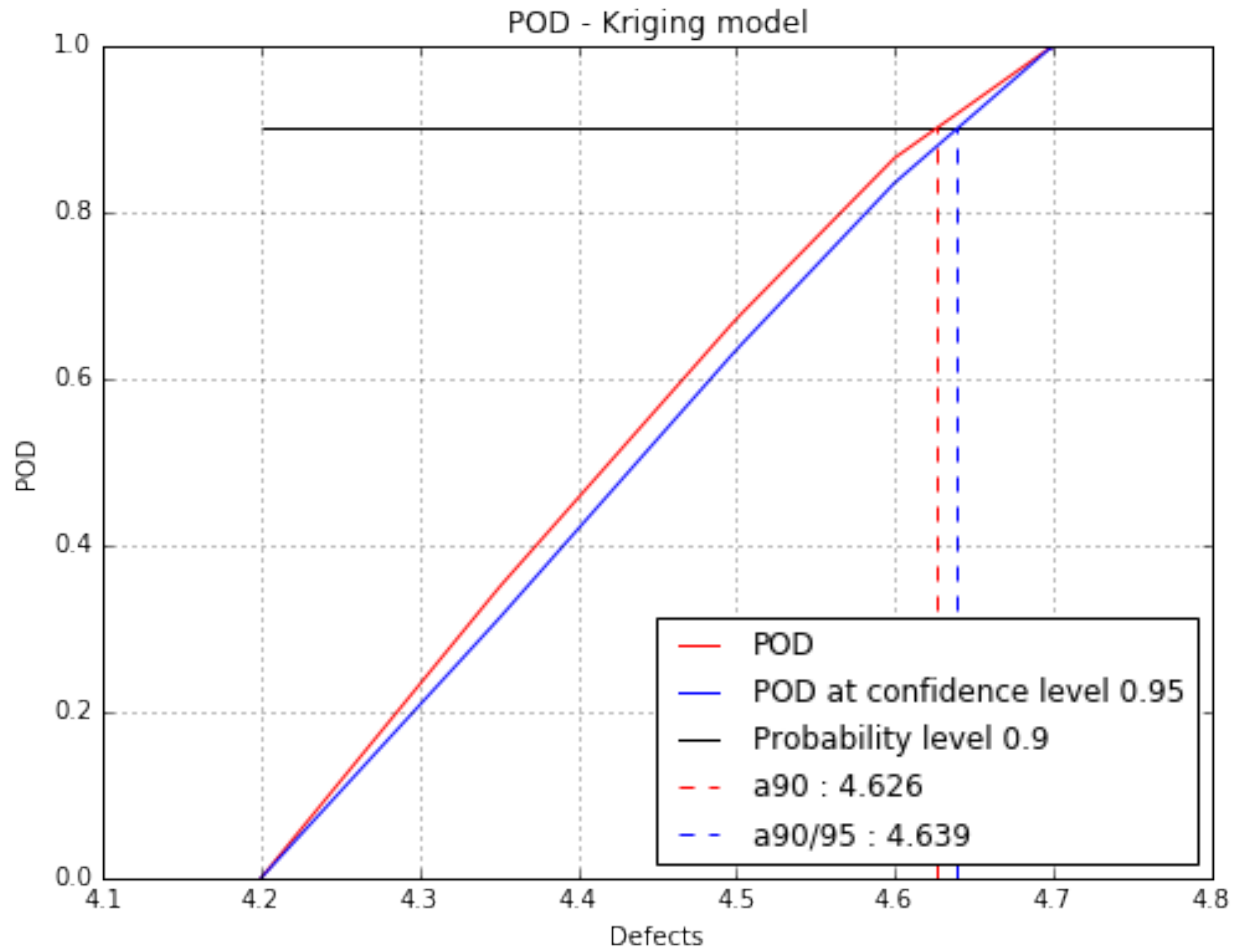
```



```

Iteration : 3/5
Computing criterion: [=====] 100.00% Done
Criterion value : 0.0157
Added point : [4.3582,0.0294083,1.0403,79.0659]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9925

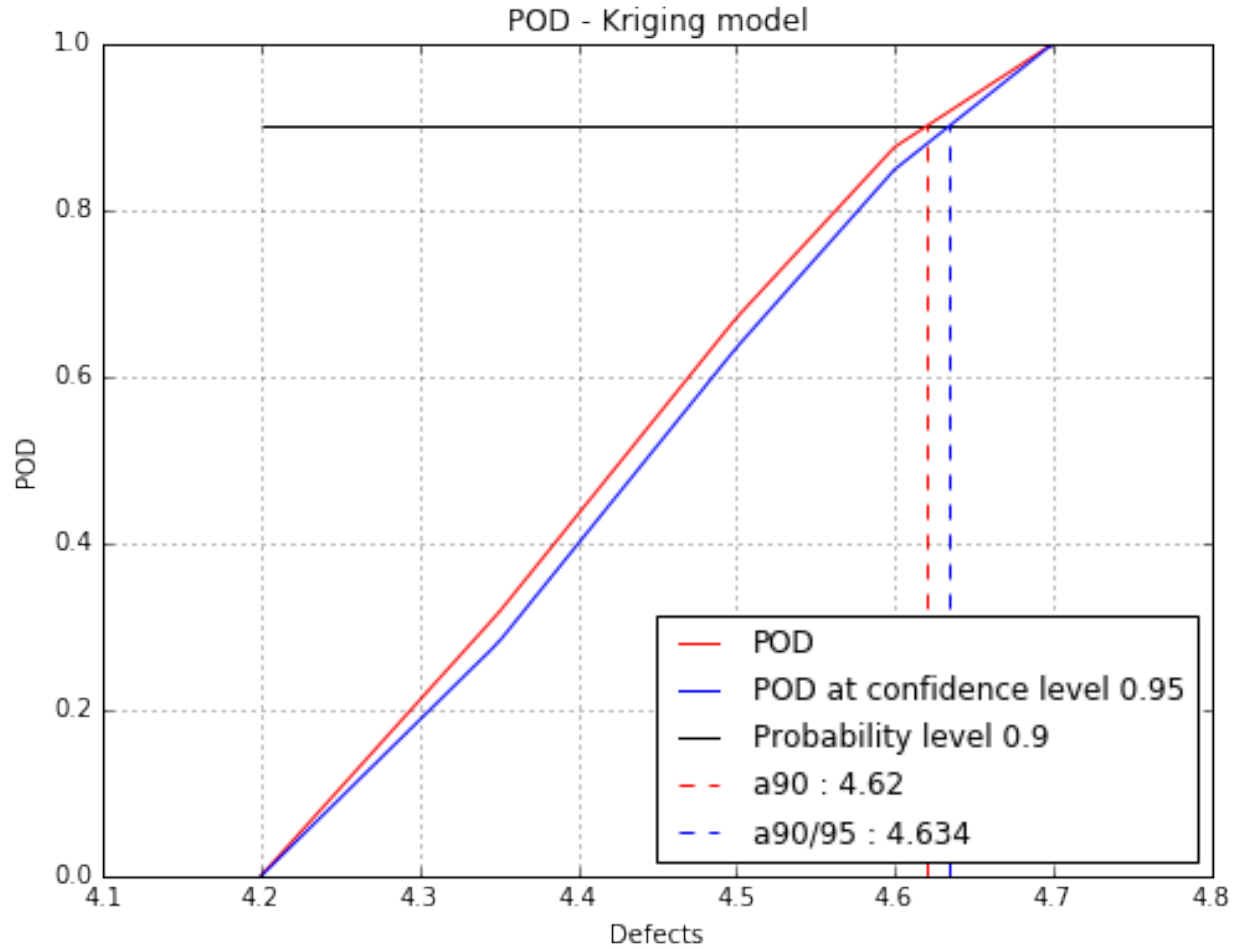
```



```

Iteration : 4/5
Computing criterion: [=====] 100.00% Done
Criterion value : 0.0143
Added point : [4.32422,0.0452993,1.06567,49.5379]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9886

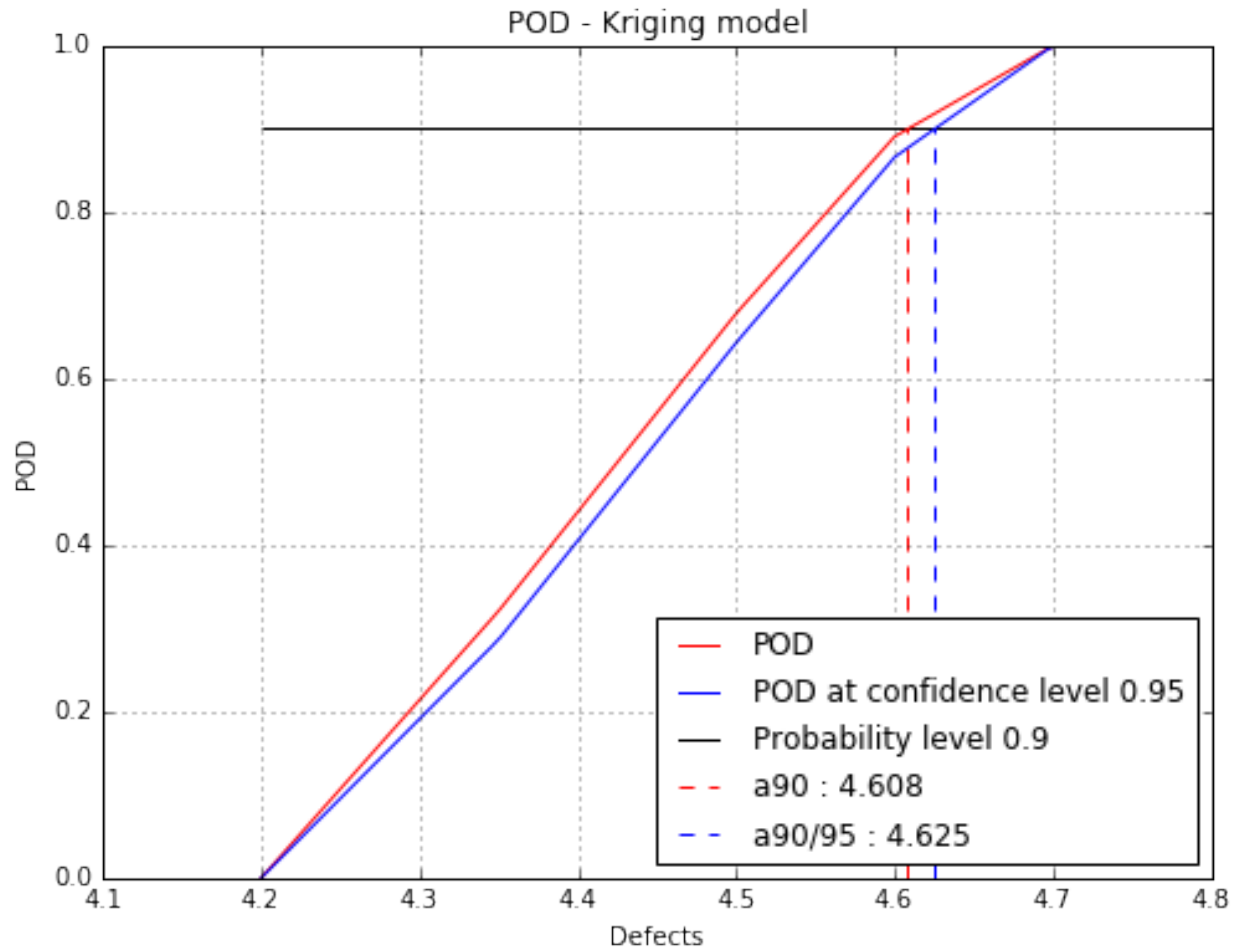
```



```

Iteration : 5/5
Computing criterion: [=====] 100.00% Done
Criterion value : 0.0139
Added point : [4.66465,0.0723142,1.02347,40.5827]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9882

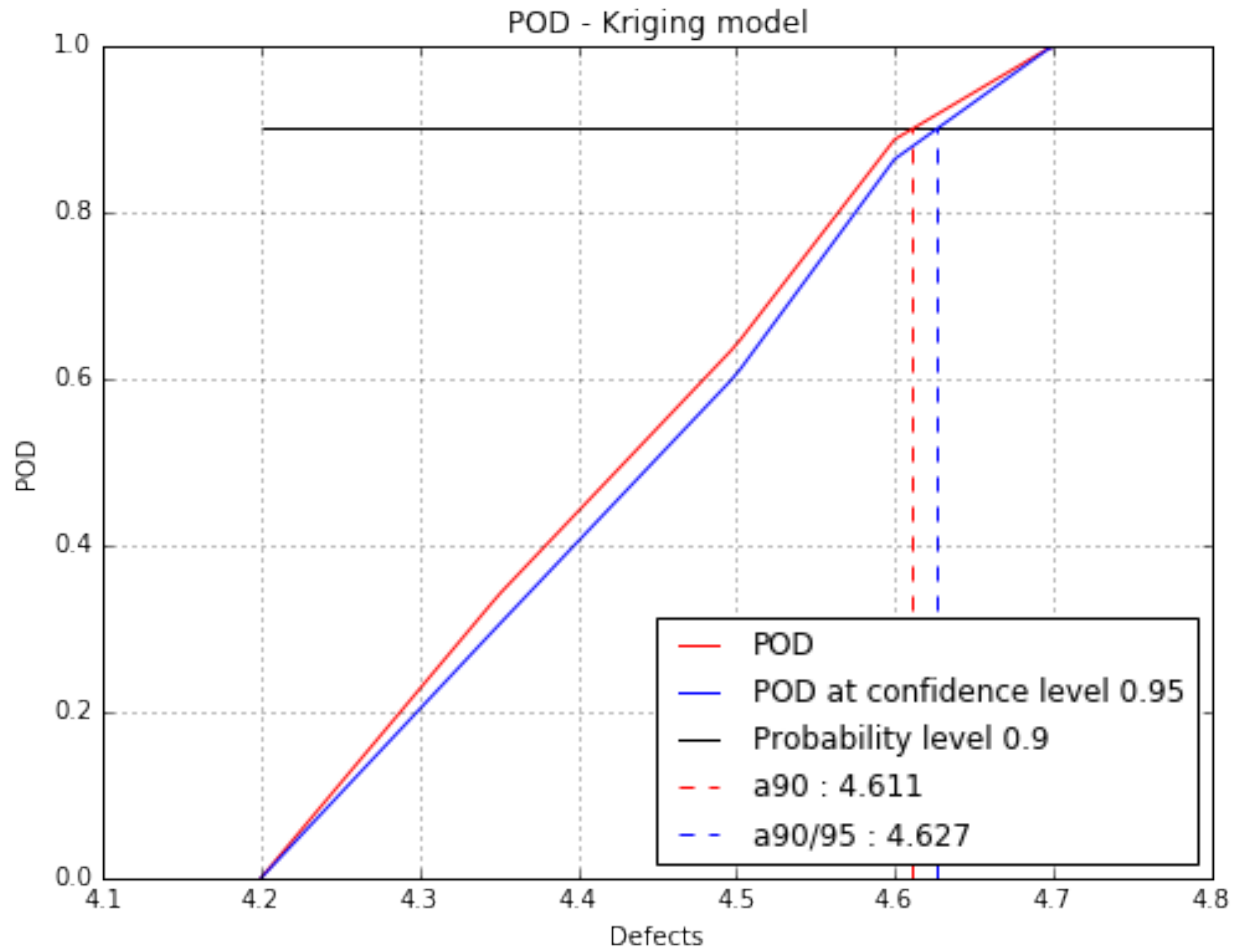
```



```
Start computing the POD with the last updated kriging model
Computing POD per defect: [=====] 100.00
→% Done
CPU times: user 1h 56min 23s, sys: 17min 58s, total: 2h 14min 21s
Wall time: 45min 14s
```

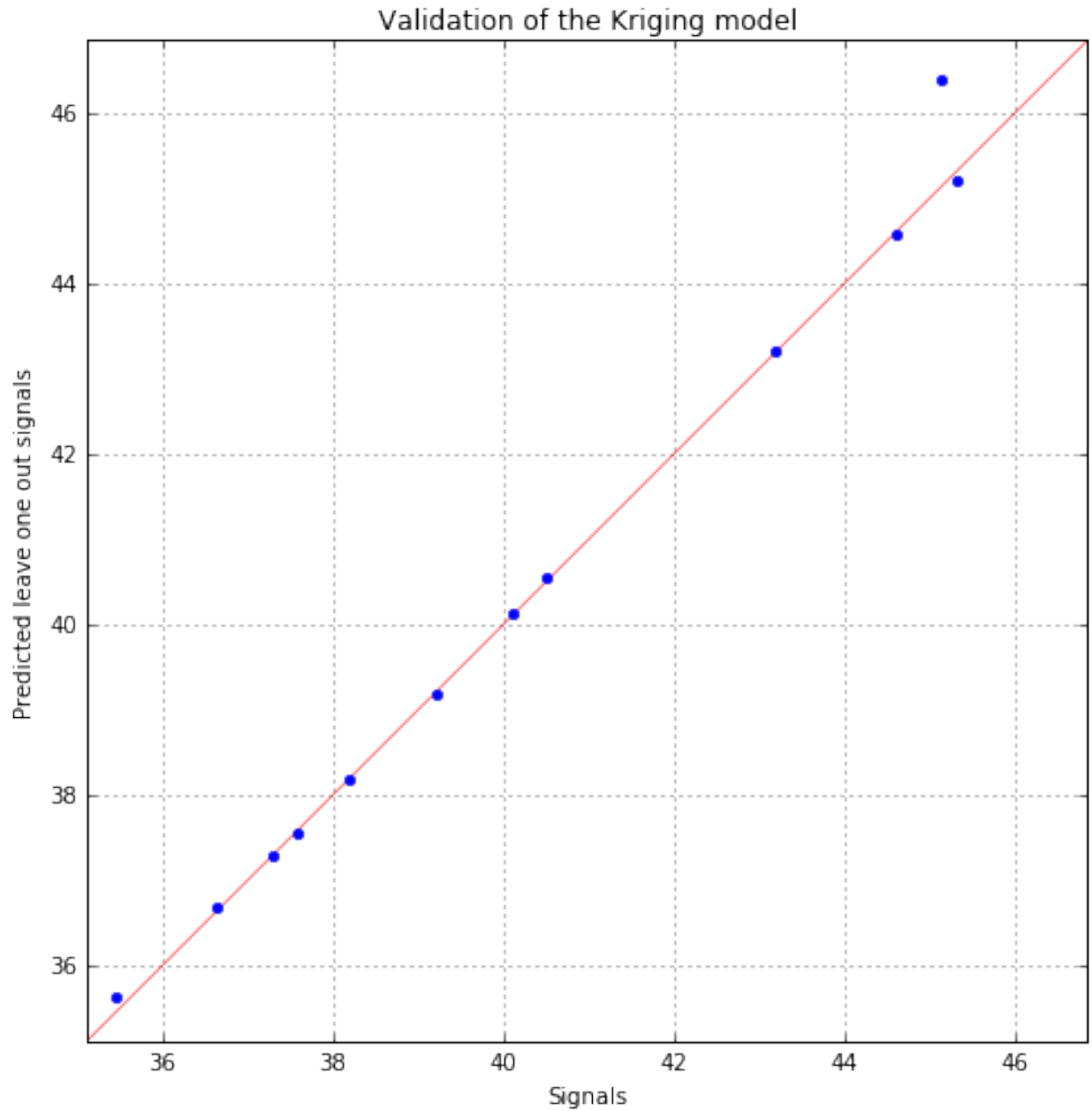
### Display the POD result based on the adative kriging model

```
fig, ax = adaptivePOD.drawPOD(0.9, 0.95)
fig.show()
```



### Display the validation graph

```
fid, ax = adaptivePOD.drawValidationGraph()
fig.show()
```



### Quality improvement of the POD computation

From the adaptive algorithm, the kriging result and the final DOE are available. As the number of simulations were reduced, we can compute again the POD with more accuracy than before if needed.

```
# get the kriging result and the final DOE from the adaptive algorithm
krigingRes = adaptivePOD.getKrigingResult()
inputfinal = adaptivePOD.getInputDOE()
outputfinal = adaptivePOD.getOutputDOE()
defectSizes = adaptivePOD.getDefectSizes()

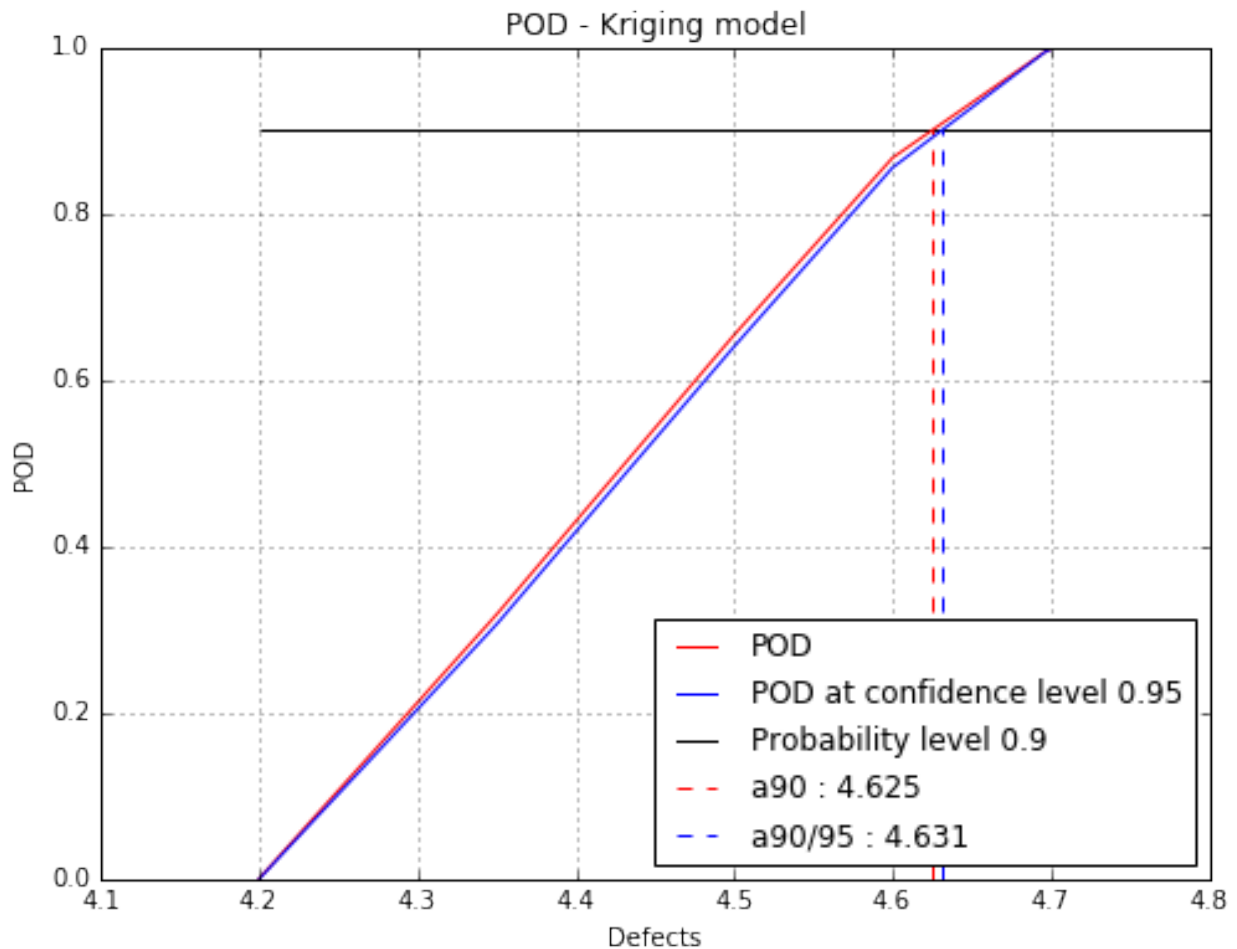
# A new POD study is launch with the DOE values
```



```
finalPOD = otpod.KrigingPOD(inputfinal, outputfinal, detection)
finalPOD.setDefectSizes(defectSizes)
# The kriging model is already known so it is given to this study
finalPOD.setKrigingResult(krigingRes)
finalPOD.run()
```

```
kriging validation Q2 (>0.9): 0.9882
Computing POD per defect: [=====] 100.00
→ % Done
```

```
fig, ax = finalPOD.drawPOD(0.9, 0.95)
fig.show()
```



[ipynb source code](#)

## 1.2.10 Adaptive Hit Miss POD

```
%matplotlib inline
import numpy as np
import openturns as ot
import otpod
```

```
import warnings
warnings.filterwarnings("ignore")
```

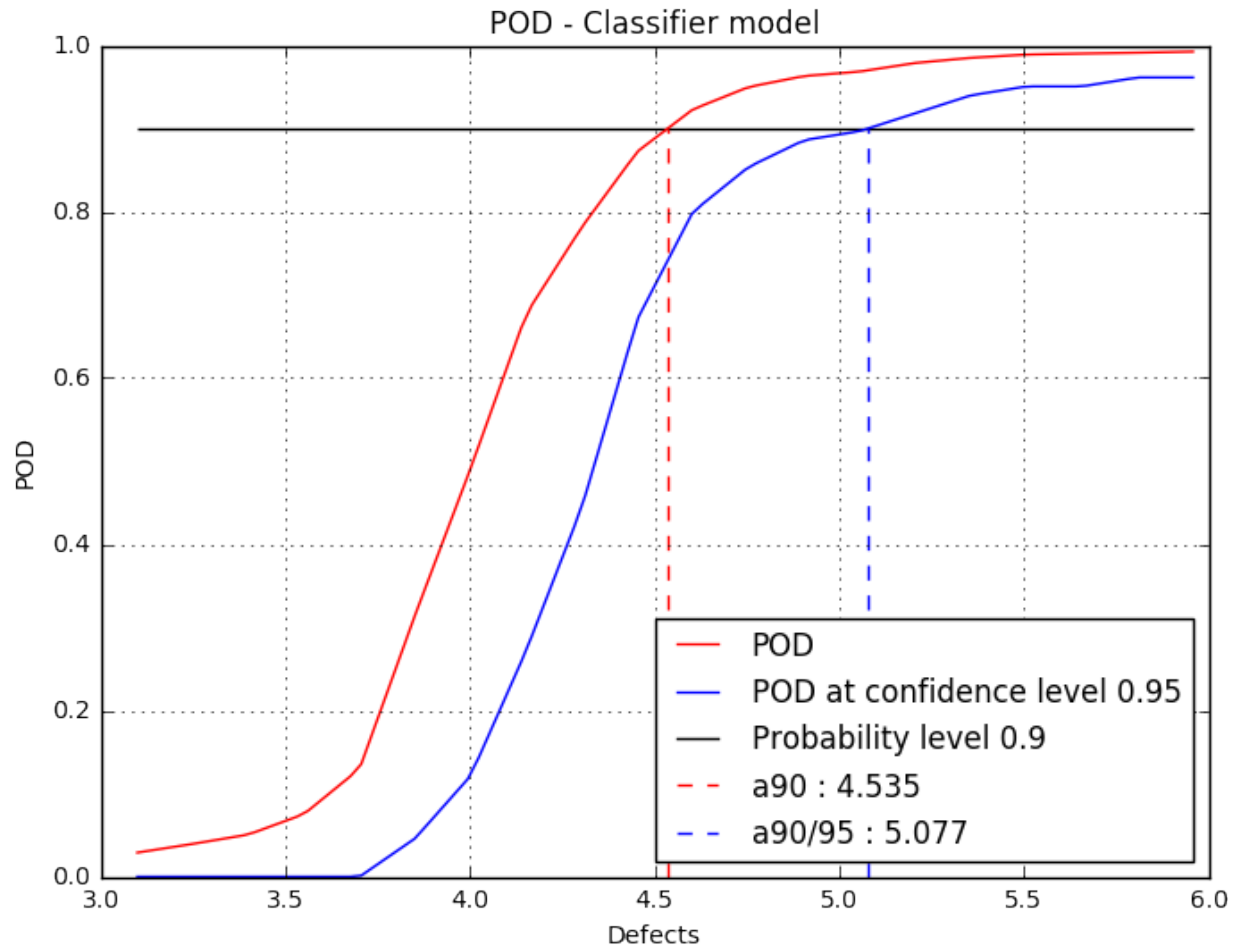
```
# The Hit/Miss function is build by executing "Make_HM.py"
# The function is called "MyHM"
execfile("Make_HM.py")
```

```
-----
The function 'MyHM' has been loaded
MyHM inputs dimension : 4
MyHM output dimension :
1 if signal > 33
0 if signal < 33
```

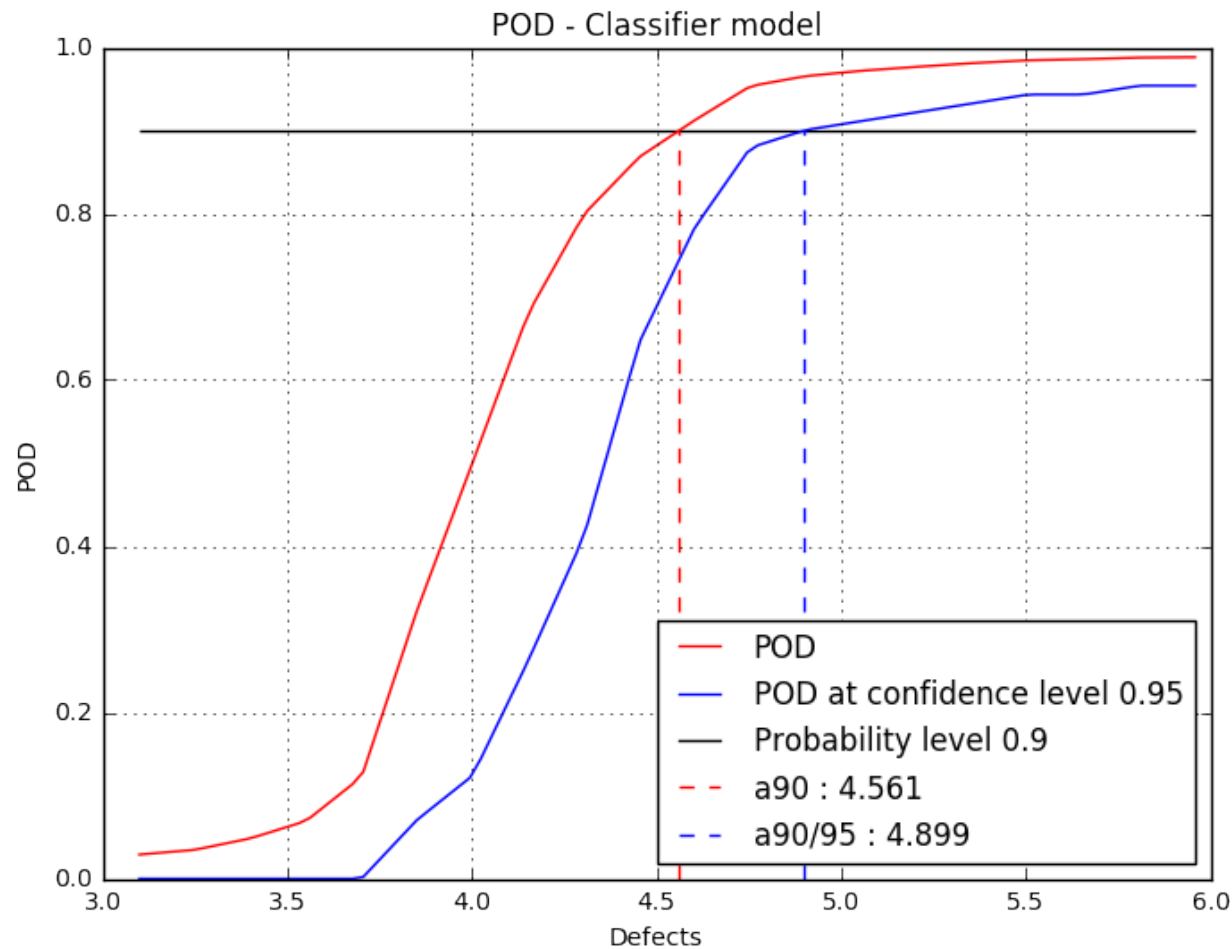
```
n_ini = 100
inputDOE = np.array([np.random.uniform(x_min[i],
                                     x_max[i], n_ini)
                    for i in range(d)]).T
outputDOE = MyHM(inputDOE)
```

```
n_more = 30
# Add n_more points with the adaptive algorithm
# 5 points are added at each iteration
hitmiss_algo = res_algo = otpod.AdaptiveHitMissPOD(inputDOE, outputDOE, MyHM, n_more)
hitmiss_algo.setClassifierType("rf")
# Computation of the POD at each iteration activated and display the POD graph
hitmiss_algo.setGraphActive(True, 0.9, 0.95, 'figure/')
hitmiss_algo.run()
```

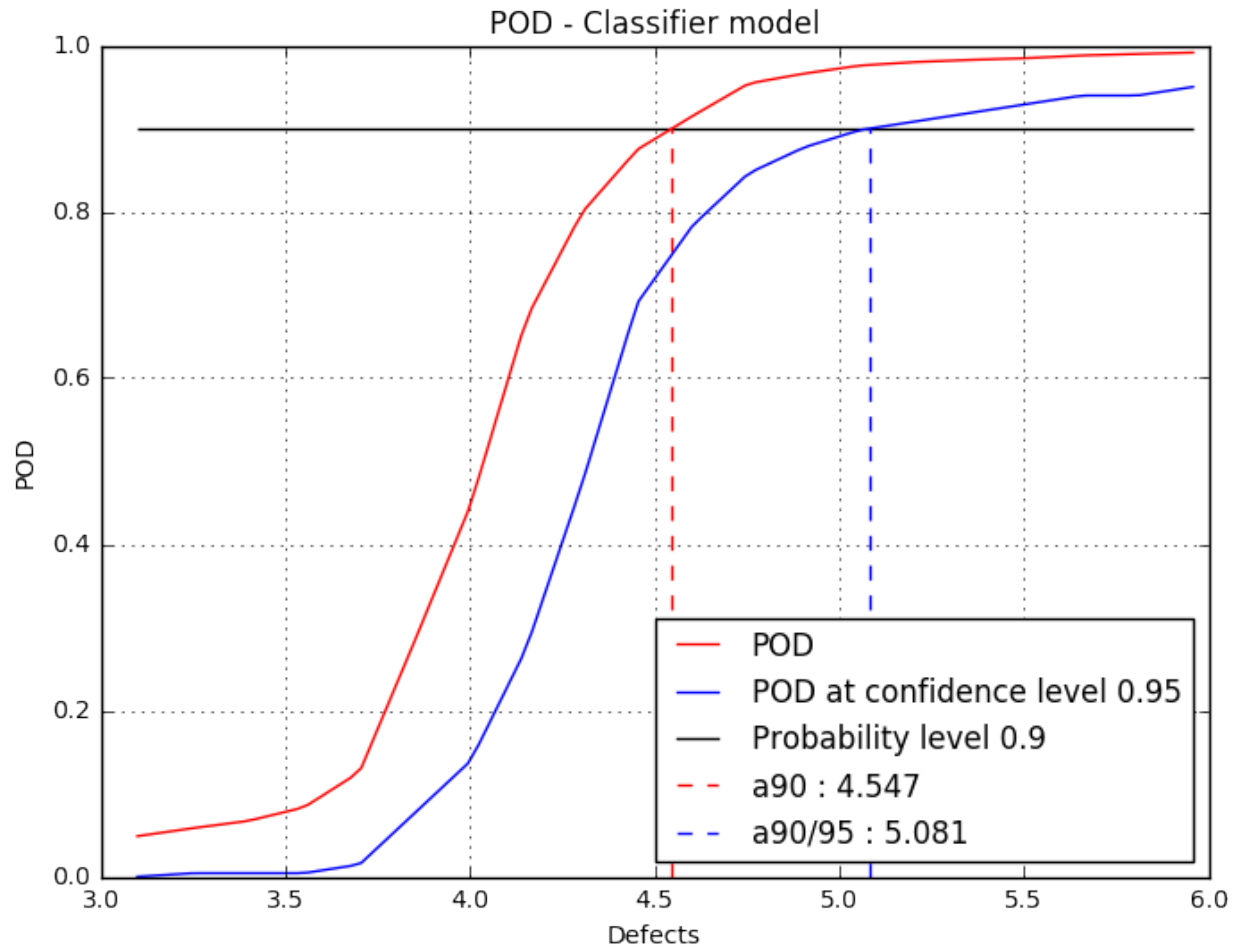
```
Building the classifier
Start the improvement loop
Adding points: [=====] 16.67%
```



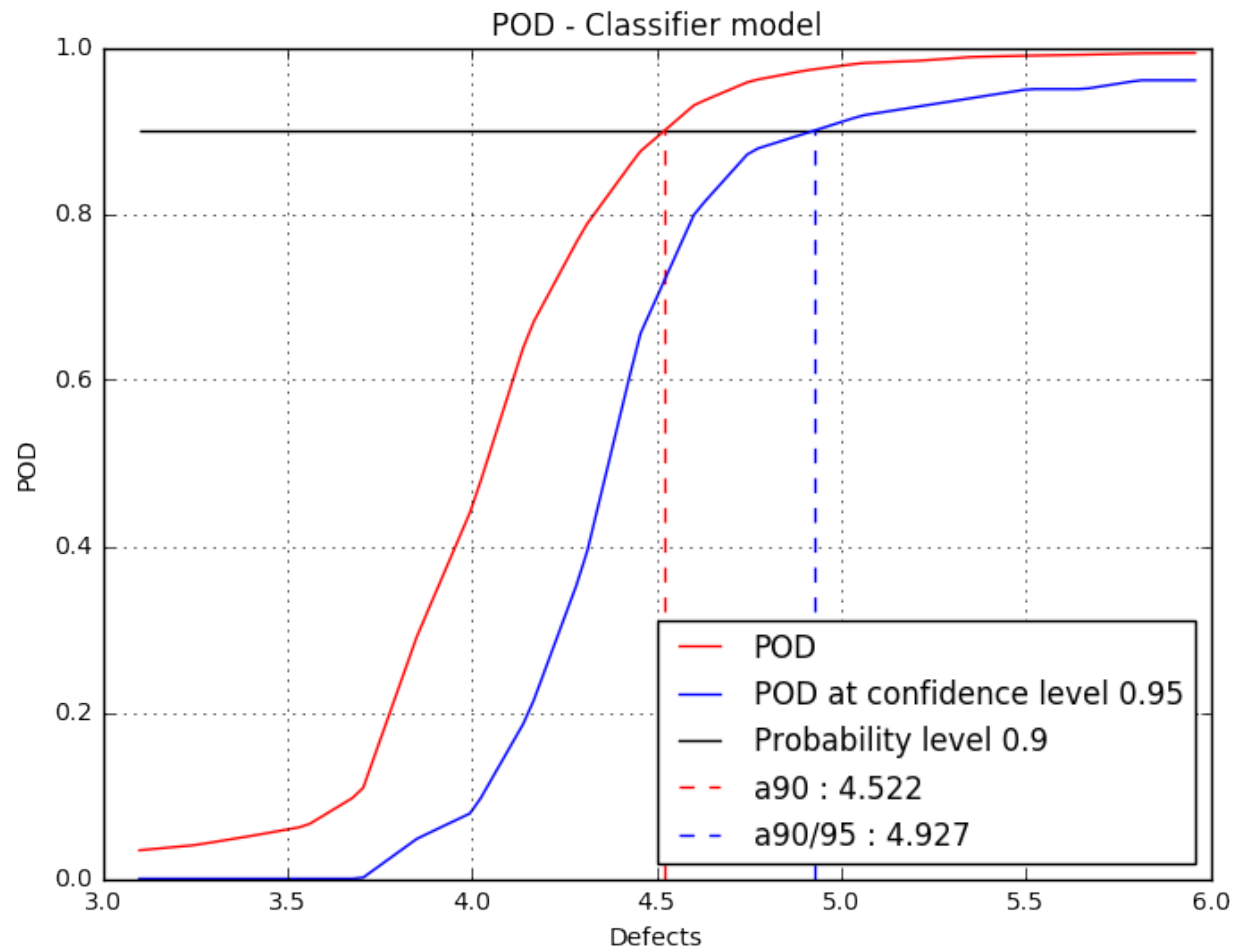
Adding points: [=====] 33.33%



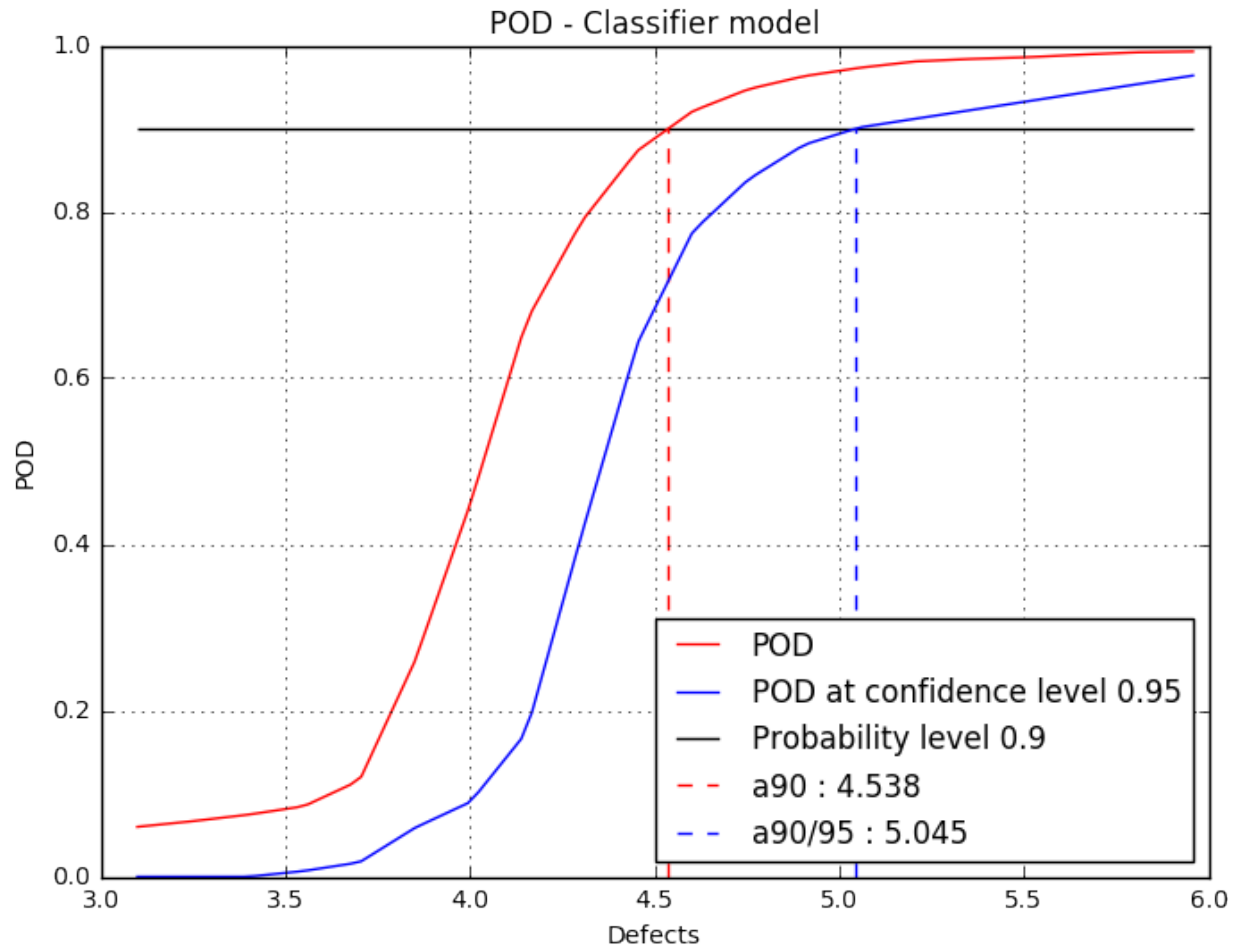
Adding points: [=====] 50.00%



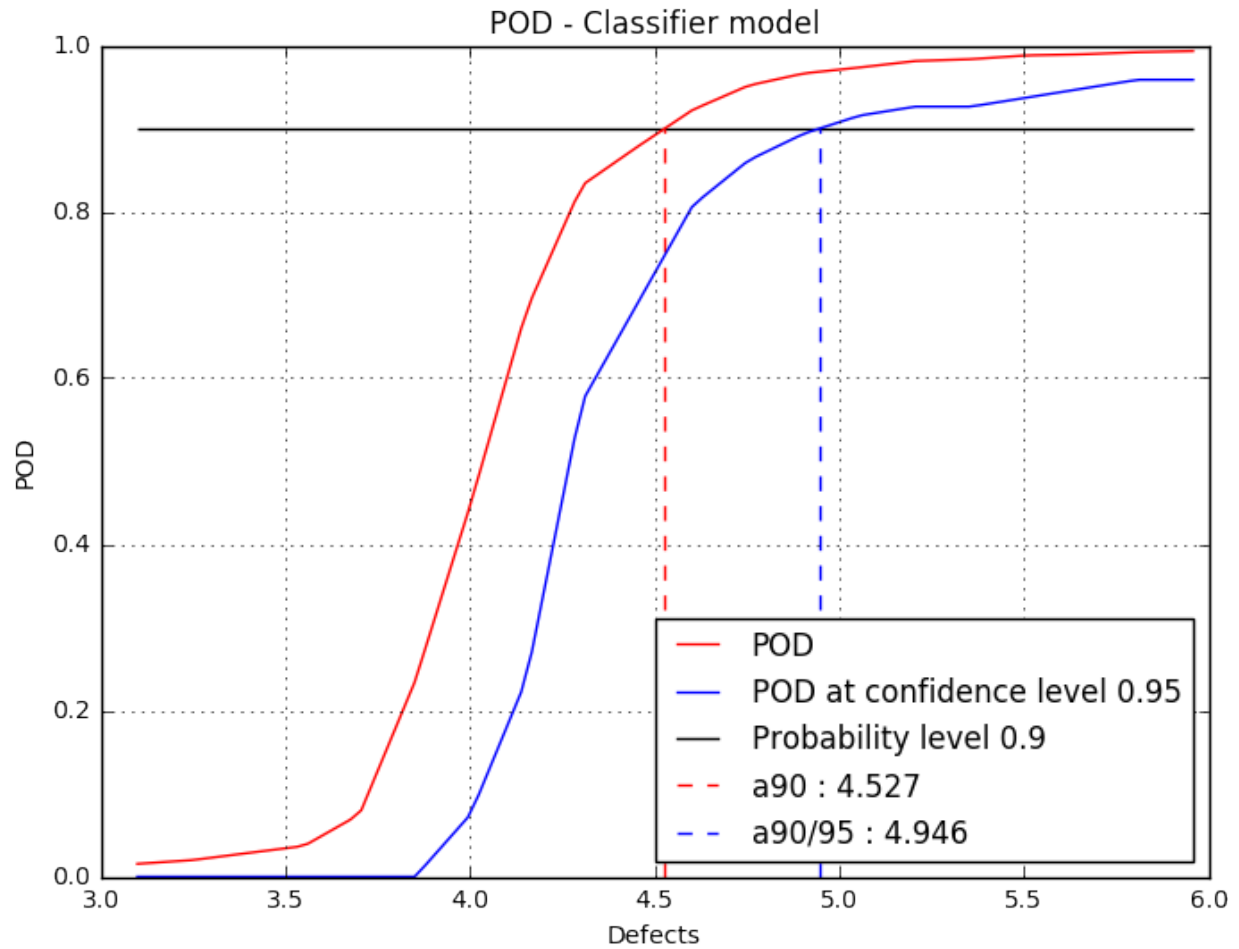
Adding points: [=====] 66.67%



Adding points: [=====] 83.33%

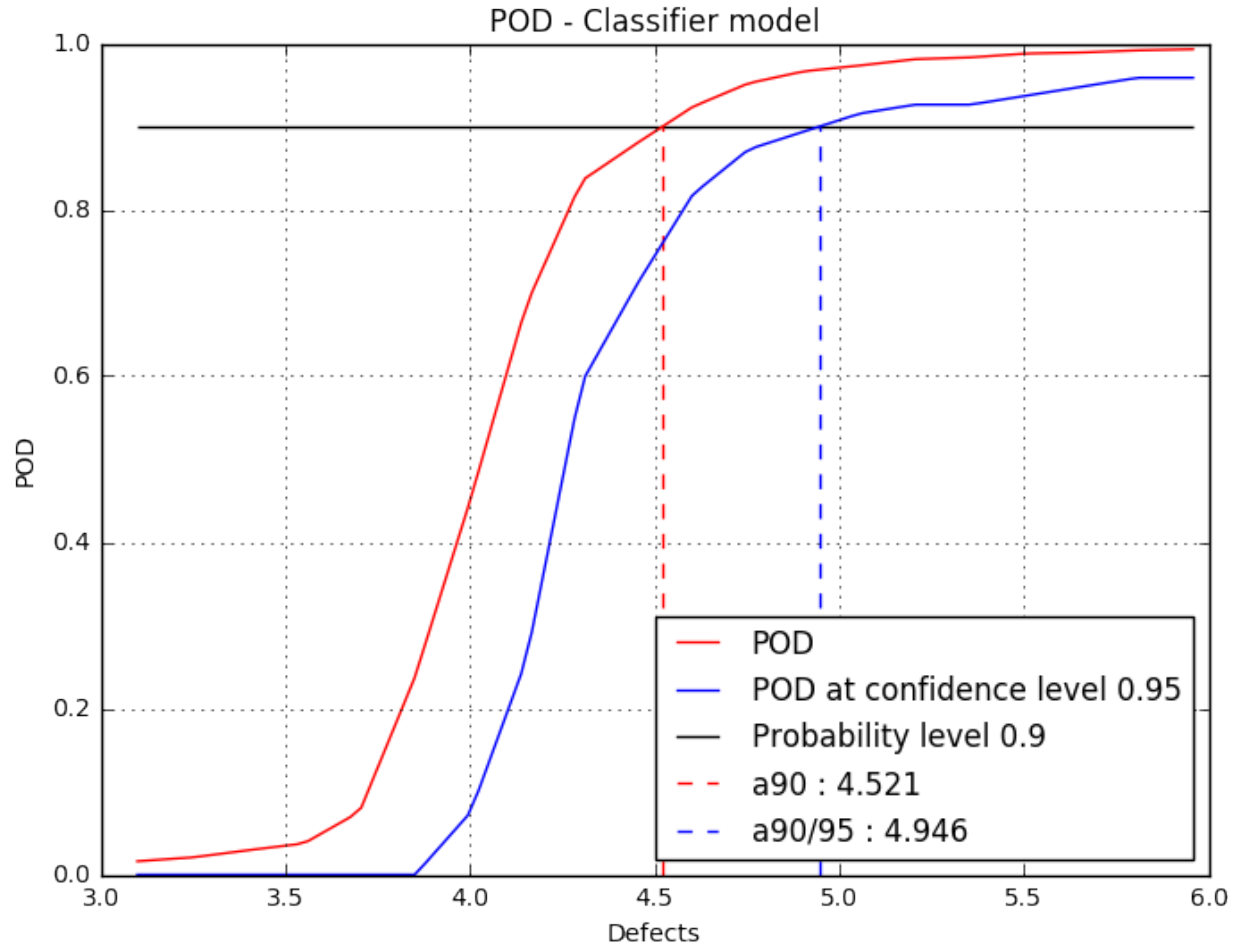


Adding points: [=====] 100.00% Done



```
fig, ax = hitmiss_algo.drawPOD(0.9, confidenceLevel=0.95)
fig.show()
```





### Signal case

Case where the physical model is a function providing the a signal value. In this case, the detection threshold must be given. The hit miss function is built inside the AdaptiveHitMissPOD class and is then used in the algorithm.

```
inputSample = ot.NumericalSample(
    [[4.59626812e+00, 7.46143339e-02, 1.02231538e+00, 8.60042277e+01],
     [4.14315790e+00, 4.20801346e-02, 1.05874908e+00, 2.65757364e+01],
     [4.76735111e+00, 3.72414824e-02, 1.05730385e+00, 5.76058433e+01],
     [4.82811977e+00, 2.49997658e-02, 1.06954641e+00, 2.54461380e+01],
     [4.48961094e+00, 3.74562922e-02, 1.04943946e+00, 6.19483646e+00],
     [5.05605334e+00, 4.87599783e-02, 1.06520409e+00, 3.39024904e+00],
     [5.69679328e+00, 7.74915877e-02, 1.04099514e+00, 6.50990466e+01],
     [5.10193991e+00, 4.35520544e-02, 1.02502536e+00, 5.51492592e+01],
     [4.04791970e+00, 2.38565932e-02, 1.01906882e+00, 2.07875350e+01],
     [4.66238956e+00, 5.49901237e-02, 1.02427200e+00, 1.45661275e+01],
     [4.86634219e+00, 6.04693570e-02, 1.08199374e+00, 1.05104730e+00],
     [4.13519347e+00, 4.45225831e-02, 1.01900124e+00, 5.10117047e+01],
     [4.92541940e+00, 7.87692335e-02, 9.91868726e-01, 8.32302238e+01],
     [4.70722074e+00, 6.51799251e-02, 1.10608515e+00, 3.30181002e+01],
     [4.29040932e+00, 1.75426222e-02, 9.75678838e-01, 2.28186756e+01],
     [4.89291400e+00, 2.34997929e-02, 1.07669835e+00, 5.38926138e+01],
     [4.44653744e+00, 7.63175936e-02, 1.06979154e+00, 5.19109415e+01],
```

```

[3.99977452e+00, 5.80430585e-02, 1.01850716e+00, 7.61988190e+01],
[3.95491570e+00, 1.09302814e-02, 1.03687664e+00, 6.09981789e+01],
[5.16424368e+00, 2.69026464e-02, 1.06673711e+00, 2.88708887e+01],
[5.30491620e+00, 4.53802273e-02, 1.06254792e+00, 3.03856837e+01],
[4.92809155e+00, 1.20616369e-02, 1.00700410e+00, 7.02512744e+00],
[4.68373805e+00, 6.26028935e-02, 1.05152117e+00, 4.81271603e+01],
[5.32381954e+00, 4.33013582e-02, 9.90522007e-01, 6.56015973e+01],
[4.35455857e+00, 1.23814619e-02, 1.01810539e+00, 1.10769534e+01]])

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.
↪609524],
    [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469
↪],
    [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.
↪768369],
    [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.
↪636588],
    [ 40.643948]])

# detection threshold
detection = 38

# Select point as initial DOE
inputDOE = inputSample[:]
outputDOE = signals[:]

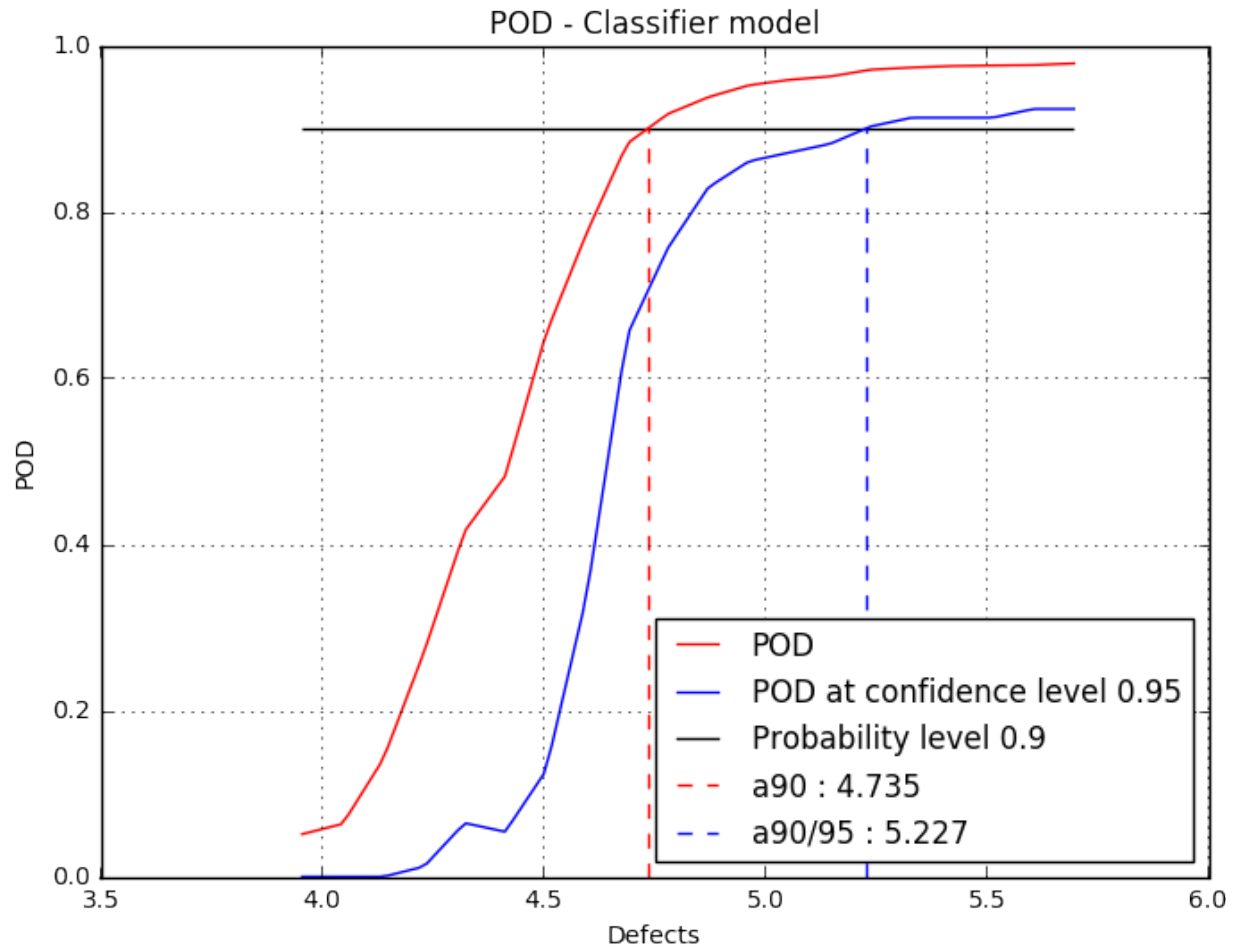
# simulate the true physical model
basis = ot.ConstantBasisFactory(4).build()
covModel = ot.SquaredExponential(4)
krigingModel = ot.KrigingAlgorithm(inputSample, signals, basis, covModel)
TNC = ot.TNC()
TNC.setBoundConstraints(ot.Interval([0.001], [100]))
krigingModel.setOptimizer(TNC)
krigingModel.run()
physicalModel = krigingModel.getResult().getMetaModel()

adaptivePOD = otpod.AdaptiveHitMissPOD(inputDOE, outputDOE, physicalModel, 100,
↪detection)
adaptivePOD.run()

Building the classifier
Start the improvement loop
Adding points: [=====] 100.00% Done

fig, ax = adaptivePOD.drawPOD(0.9, confidenceLevel=0.95)
fig.show()

```





## INDICES AND TABLES

- `genindex`
- `search`



**A**

AdaptiveHitMissPOD (class in otpod), 36

AdaptiveSignalPOD (class in otpod), 29

**C**

computeDetectionSize() (AdaptiveHitMissPOD method), 38

computeDetectionSize() (AdaptiveSignalPOD method), 31

computeDetectionSize() (KrigingPOD method), 25

computeDetectionSize() (PolynomialChaosPOD method), 19

computeDetectionSize() (QuantileRegressionPOD method), 15

computeDetectionSize() (UnivariateLinearModelPOD method), 11

**D**

DataHandling (class in otpod), 46

drawBoxCoxLikelihood() (AdaptiveHitMissPOD method), 38

drawBoxCoxLikelihood() (AdaptiveSignalPOD method), 31

drawBoxCoxLikelihood() (KrigingPOD method), 25

drawBoxCoxLikelihood() (PolynomialChaosPOD method), 19

drawBoxCoxLikelihood() (QuantileRegressionPOD method), 15

drawBoxCoxLikelihood() (UnivariateLinearModelAnalysis method), 5

drawBoxCoxLikelihood() (UnivariateLinearModelPOD method), 11

drawGraphs() (PODSummary method), 43

drawLinearModel() (QuantileRegressionPOD method), 15

drawLinearModel() (UnivariateLinearModelAnalysis method), 5

drawPOD() (AdaptiveHitMissPOD method), 38

drawPOD() (AdaptiveSignalPOD method), 32

drawPOD() (KrigingPOD method), 26

drawPOD() (PolynomialChaosPOD method), 20

drawPOD() (QuantileRegressionPOD method), 16

drawPOD() (UnivariateLinearModelPOD method), 11

drawPolynomialChaosModel() (PolynomialChaosPOD method), 20

drawResiduals() (UnivariateLinearModelAnalysis method), 6

drawResidualsDistribution() (UnivariateLinearModelAnalysis method), 6

drawResidualsQQplot() (UnivariateLinearModelAnalysis method), 6

drawValidationGraph() (AdaptiveSignalPOD method), 32

drawValidationGraph() (KrigingPOD method), 26

drawValidationGraph() (PolynomialChaosPOD method), 21

**F**

filterCensoredData() (DataHandling static method), 46

**G**

getAdaptiveStrategy() (PolynomialChaosPOD method), 21

getAndersonDarlingPValue() (UnivariateLinearModelAnalysis method), 7

getBasis() (AdaptiveSignalPOD method), 33

getBasis() (KrigingPOD method), 26

getBoxCoxParameter() (AdaptiveHitMissPOD method), 39

getBoxCoxParameter() (AdaptiveSignalPOD method), 33

getBoxCoxParameter() (KrigingPOD method), 27

getBoxCoxParameter() (PolynomialChaosPOD method), 21

getBoxCoxParameter() (QuantileRegressionPOD method), 16

getBoxCoxParameter() (UnivariateLinearModelAnalysis method), 7

getBoxCoxParameter() (UnivariateLinearModelPOD method), 12

getBreuschPaganPValue() (UnivariateLinearModelAnalysis method), 7

getCandidateSize() (AdaptiveHitMissPOD method), 39

getCandidateSize() (AdaptiveSignalPOD method), 33

getClassifier() (AdaptiveHitMissPOD method), 39

getClassifierParameters() (AdaptiveHitMissPOD method), 39

getClassifierType() (AdaptiveHitMissPOD method), 39  
 getConfusionMatrix() (AdaptiveHitMissPOD method), 39  
 getCovarianceModel() (AdaptiveSignalPOD method), 33  
 getCovarianceModel() (KrigingPOD method), 27  
 getCramerVonMisesPValue() (UnivariateLinearModelAnalysis method), 7  
 getDefectSizes() (AdaptiveHitMissPOD method), 39  
 getDefectSizes() (AdaptiveSignalPOD method), 33  
 getDefectSizes() (KrigingPOD method), 27  
 getDefectSizes() (PolynomialChaosPOD method), 21  
 getDegree() (PolynomialChaosPOD method), 21  
 getDistribution() (AdaptiveHitMissPOD method), 39  
 getDistribution() (AdaptiveSignalPOD method), 33  
 getDistribution() (KrigingPOD method), 27  
 getDistribution() (PolynomialChaosPOD method), 21  
 getDurbinWatsonPValue() (UnivariateLinearModelAnalysis method), 7  
 getGraphActive() (AdaptiveHitMissPOD method), 39  
 getGraphActive() (AdaptiveSignalPOD method), 33  
 getHarrisonMcCabePValue() (UnivariateLinearModelAnalysis method), 7  
 getInitialStartSize() (AdaptiveSignalPOD method), 33  
 getInitialStartSize() (KrigingPOD method), 27  
 getInputDOE() (AdaptiveHitMissPOD method), 40  
 getInputDOE() (AdaptiveSignalPOD method), 33  
 getInputSample() (UnivariateLinearModelAnalysis method), 7  
 getIntercept() (UnivariateLinearModelAnalysis method), 8  
 getKolmogorovPValue() (UnivariateLinearModelAnalysis method), 8  
 getKrigingPOD() (PODSummary method), 44  
 getKrigingResult() (AdaptiveSignalPOD method), 33  
 getKrigingResult() (KrigingPOD method), 27  
 getLinearBinomialPOD() (PODSummary method), 44  
 getLinearGaussPOD() (PODSummary method), 44  
 getLinearKernelSmoothingPOD() (PODSummary method), 44  
 getMethodActive() (PODSummary method), 44  
 getNoiseThreshold() (UnivariateLinearModelAnalysis method), 8  
 getOutputDOE() (AdaptiveHitMissPOD method), 40  
 getOutputDOE() (AdaptiveSignalPOD method), 34  
 getOutputSample() (UnivariateLinearModelAnalysis method), 8  
 getPMax() (AdaptiveHitMissPOD method), 40  
 getPMin() (AdaptiveHitMissPOD method), 40  
 getPODCLModel() (AdaptiveHitMissPOD method), 40  
 getPODCLModel() (AdaptiveSignalPOD method), 34  
 getPODCLModel() (KrigingPOD method), 27  
 getPODCLModel() (PolynomialChaosPOD method), 21  
 getPODCLModel() (QuantileRegressionPOD method), 16  
 getPODCLModel() (UnivariateLinearModelPOD method), 12  
 getPODModel() (AdaptiveHitMissPOD method), 40  
 getPODModel() (AdaptiveSignalPOD method), 34  
 getPODModel() (KrigingPOD method), 27  
 getPODModel() (PolynomialChaosPOD method), 22  
 getPODModel() (QuantileRegressionPOD method), 17  
 getPODModel() (UnivariateLinearModelPOD method), 12  
 getPolynomialChaosPOD() (PODSummary method), 44  
 getPolynomialChaosResult() (PolynomialChaosPOD method), 22  
 getProjectionStrategy() (PolynomialChaosPOD method), 22  
 getQ2() (AdaptiveSignalPOD method), 34  
 getQ2() (KrigingPOD method), 28  
 getQ2() (PolynomialChaosPOD method), 22  
 getQuantile() (QuantileRegressionPOD method), 17  
 getQuantileRegressionPOD() (PODSummary method), 44  
 getR2() (PolynomialChaosPOD method), 22  
 getR2() (QuantileRegressionPOD method), 17  
 getR2() (UnivariateLinearModelAnalysis method), 8  
 getR2() (UnivariateLinearModelPOD method), 12  
 getResiduals() (UnivariateLinearModelAnalysis method), 8  
 getResidualsDistribution() (UnivariateLinearModelAnalysis method), 8  
 getSamplingSize() (AdaptiveHitMissPOD method), 40  
 getSamplingSize() (AdaptiveSignalPOD method), 34  
 getSamplingSize() (KrigingPOD method), 28  
 getSamplingSize() (PODSummary method), 44  
 getSamplingSize() (PolynomialChaosPOD method), 22  
 getSaturationThreshold() (UnivariateLinearModelAnalysis method), 8  
 getSimulationSize() (AdaptiveHitMissPOD method), 40  
 getSimulationSize() (AdaptiveSignalPOD method), 34  
 getSimulationSize() (KrigingPOD method), 28  
 getSimulationSize() (PODSummary method), 44  
 getSimulationSize() (PolynomialChaosPOD method), 22  
 getSimulationSize() (QuantileRegressionPOD method), 17  
 getSimulationSize() (UnivariateLinearModelPOD method), 12  
 getSlope() (UnivariateLinearModelAnalysis method), 8  
 getStandardError() (UnivariateLinearModelAnalysis method), 8  
 getVerbose() (AdaptiveHitMissPOD method), 40  
 getVerbose() (AdaptiveSignalPOD method), 34  
 getVerbose() (KrigingPOD method), 28  
 getVerbose() (PODSummary method), 45  
 getVerbose() (PolynomialChaosPOD method), 22  
 getVerbose() (QuantileRegressionPOD method), 17  
 getVerbose() (UnivariateLinearModelPOD method), 12



getZeroMeanPValue() (UnivariateLinearModelAnalysis method), 9

## K

KrigingPOD (class in otpod), 24

## P

PODSummary (class in otpod), 42

PolynomialChaosPOD (class in otpod), 18

printResults() (PODSummary method), 45

printResults() (UnivariateLinearModelAnalysis method), 9

## Q

QuantileRegressionPOD (class in otpod), 13

## R

run() (AdaptiveHitMissPOD method), 40

run() (AdaptiveSignalPOD method), 34

run() (KrigingPOD method), 28

run() (PODSummary method), 45

run() (PolynomialChaosPOD method), 22

run() (QuantileRegressionPOD method), 17

run() (UnivariateLinearModelPOD method), 12

## S

saveResults() (PODSummary method), 45

saveResults() (UnivariateLinearModelAnalysis method), 9

setAdaptiveStrategy() (PolynomialChaosPOD method), 23

setBasis() (AdaptiveSignalPOD method), 35

setBasis() (KrigingPOD method), 28

setCandidateSize() (AdaptiveHitMissPOD method), 41

setCandidateSize() (AdaptiveSignalPOD method), 35

setClassifierParameters() (AdaptiveHitMissPOD method), 41

setClassifierType() (AdaptiveHitMissPOD method), 41

setCovarianceModel() (AdaptiveSignalPOD method), 35

setCovarianceModel() (KrigingPOD method), 28

setDefectSizes() (AdaptiveHitMissPOD method), 41

setDefectSizes() (AdaptiveSignalPOD method), 35

setDefectSizes() (KrigingPOD method), 28

setDefectSizes() (PolynomialChaosPOD method), 23

setDegree() (PolynomialChaosPOD method), 23

setDistribution() (AdaptiveHitMissPOD method), 41

setDistribution() (AdaptiveSignalPOD method), 35

setDistribution() (KrigingPOD method), 28

setDistribution() (PolynomialChaosPOD method), 23

setGraphActive() (AdaptiveHitMissPOD method), 41

setGraphActive() (AdaptiveSignalPOD method), 35

setInitialStartSize() (AdaptiveSignalPOD method), 35

setInitialStartSize() (KrigingPOD method), 29

setKrigingResult() (KrigingPOD method), 29

setMethodActive() (PODSummary method), 45

setPMax() (AdaptiveHitMissPOD method), 41

setPMin() (AdaptiveHitMissPOD method), 41

setPolynomialChaosResult() (PolynomialChaosPOD method), 23

setProjectionStrategy() (PolynomialChaosPOD method), 23

setQuantile() (QuantileRegressionPOD method), 17

setSamplingSize() (AdaptiveHitMissPOD method), 41

setSamplingSize() (AdaptiveSignalPOD method), 35

setSamplingSize() (KrigingPOD method), 29

setSamplingSize() (PODSummary method), 46

setSamplingSize() (PolynomialChaosPOD method), 23

setSimulationSize() (AdaptiveHitMissPOD method), 41

setSimulationSize() (AdaptiveSignalPOD method), 36

setSimulationSize() (KrigingPOD method), 29

setSimulationSize() (PODSummary method), 46

setSimulationSize() (PolynomialChaosPOD method), 23

setSimulationSize() (QuantileRegressionPOD method), 17

setSimulationSize() (UnivariateLinearModelPOD method), 12

setVerbose() (AdaptiveHitMissPOD method), 42

setVerbose() (AdaptiveSignalPOD method), 36

setVerbose() (KrigingPOD method), 29

setVerbose() (PODSummary method), 46

setVerbose() (PolynomialChaosPOD method), 23

setVerbose() (QuantileRegressionPOD method), 17

setVerbose() (UnivariateLinearModelPOD method), 13

## U

UnivariateLinearModelAnalysis (class in otpod), 3

UnivariateLinearModelPOD (class in otpod), 9