otpod Documentation

Release

Antoine Dumas

April 28, 2016

otpod is a module for OpenTURNS.

CHAPTER

ONE

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

`UnivariateLinearModelAnalysis`	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 : 0.885
```

Run analysis with a Weibull distribution hypothesis on the residuals

```
>>> analysis = otpod.UnivariateLinearModelAnalysis(defects, signals, 60., 1700., ot.WeibullFacto
>>> 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 : 0.717
```

Methods

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

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 model

UnivariateLinearModelPOD	Linear regression based POD.
QuantileRegressionPOD	Quantile regression based POD.
PolynomialChaosPOD	Polynomial chaos based POD.
KrigingPOD	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

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

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 **PODModelCl** : `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

computeDetectionSize(*args, **kwargs)	Compute the detection size for a given probability level.
drawBoxCoxLikelihood([name])	Draw the loglikelihood versus the Box Cox parameter.
drawLinearModel(probabilityLevel[, name])	Draw the quantile regression prediction versus the true data.
drawPOD(*args, **kwargs)	Draw the POD curve.
getPODCLModel([confidenceLevel])	Accessor to the POD model at a given confidence level.
getPODModel()	Accessor to the POD model.
getQuantile()	Accessor to the quantile list for the regression.
getR2(quantile)	Accessor to the pseudo R2 value.
getSimulationSize()	Accessor to the simulation size.

Continued on next page

Table 1.5 – continued from previous page

getVerbose()	Accessor to the verbosity.
run()	Build the POD models.
setQuantile(quantile)	Accessor to the quantile list for the regression.
setSimulationSize(size)	Accessor to the simulation size.
setVerbose(verbose)	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.

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 **PODModelCl** : `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 5.

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

`computeDetectionSize`(*args, **kwargs)	Compute the detection size for a given probability level.
`drawBoxCoxLikelihood`([name])	Draw the loglikelihood versus the Box Cox parameter.
`drawPOD`(*args, **kwargs)	Draw the POD curve.
`drawPolynomialChaosModel`([name])	Draw the polynomial chaos prediction versus the true data.
`drawValidationGraph`(*args, **kwargs)	Draw the validation graph of the metamodel.
`getAdaptiveStrategy`()	Accessor to the adaptive strategy.
`getDefectSizes`()	Accessor to the defect size where POD is computed.
`getDistribution`()	Accessor to the parameters distribution.
`getPODCLModel`([confidenceLevel])	Accessor to the POD model at a given confidence level.
`getPODModel`()	Accessor to the POD model.
`getPolynomialChaosResult`()	Accessor to the polynomial chaos result.
`getProjectionStrategy`()	Accessor to the projection strategy.
`getQ2`()	Accessor to the Q2 value.
`getR2`()	Accessor to the R2 value.

Continued on next page

Table 1.6 – continued from previous page

getSamplingSize()	Accessor to the Monte Carlo sampling size.
getSimulationSize()	Accessor to the simulation size.
getVerbose()	Accessor to the verbosity.
run()	Build the POD models.
setAdaptiveStrategy(strategy)	Accessor to the adaptive strategy.
setDefectSizes(size)	Accessor to the defect size where POD is computed.
setDistribution(distribution)	Accessor to the parameters distribution.
setPolynomialChaosResult(chaosResult)	Accessor to the polynomial chaos result.
setProjectionStrategy(strategy)	Accessor to the projection strategy.
setSamplingSize(size)	Accessor to the Monte Carlo sampling size.
setSimulationSize(size)	Accessor to the simulation size.
setVerbose(verbose)	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.

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.

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 PODModelCl : `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.

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

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

DataHandling	Static methods for data handling.

DataHandling

class DataHandling

Static methods for data handling.

Methods

filterCensoredData(inputSample, signals, ...)	Sort inputSample and signals with respect to the censore thresholds.

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

Sort inputSample and signals with respect to the censore 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 : please consider to use the Box Cox transformation.
```

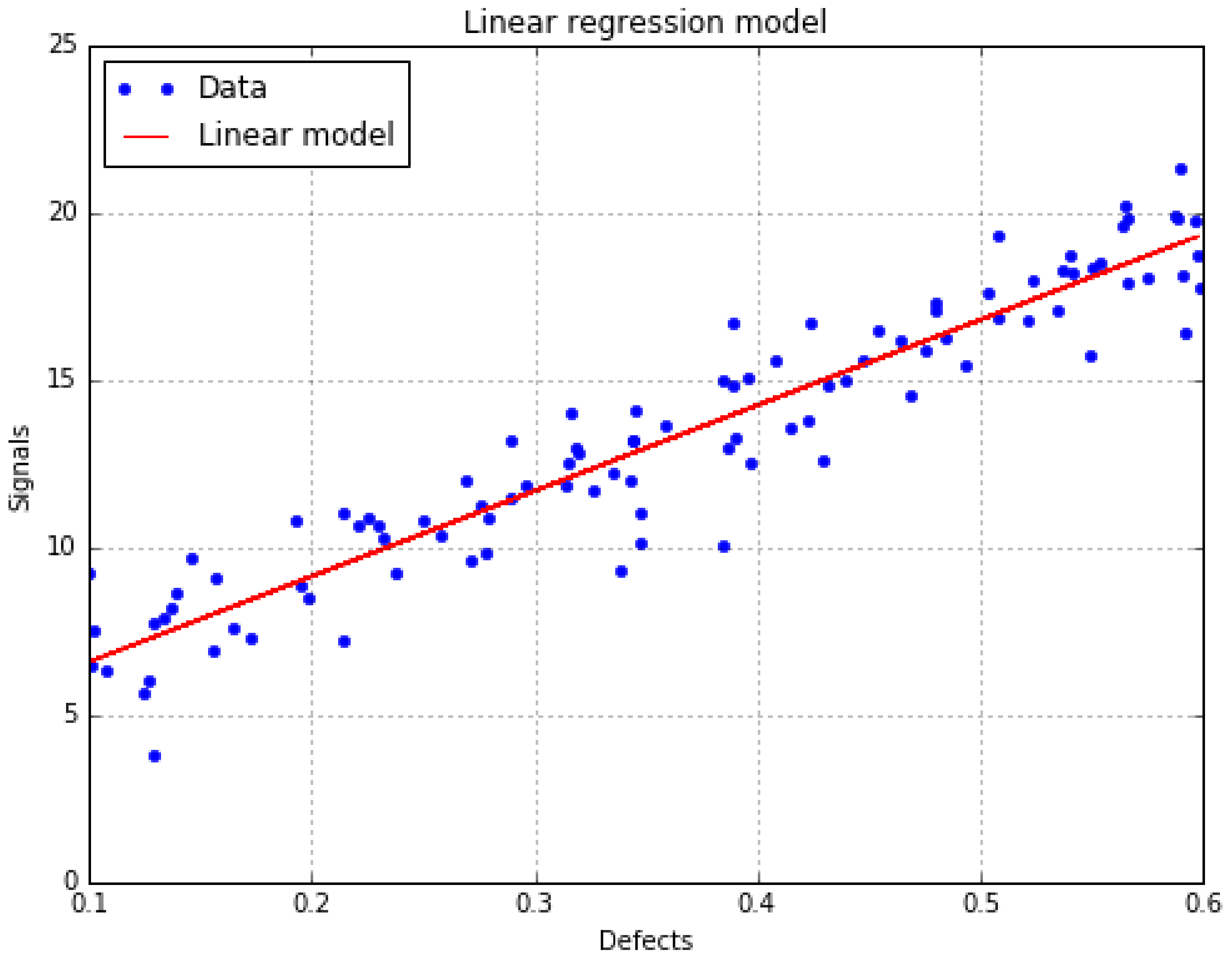
```
--------------------------------------------------------------------------------
         Linear model analysis results
--------------------------------------------------------------------------------
Box Cox parameter :                                 Not enabled

                                                     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

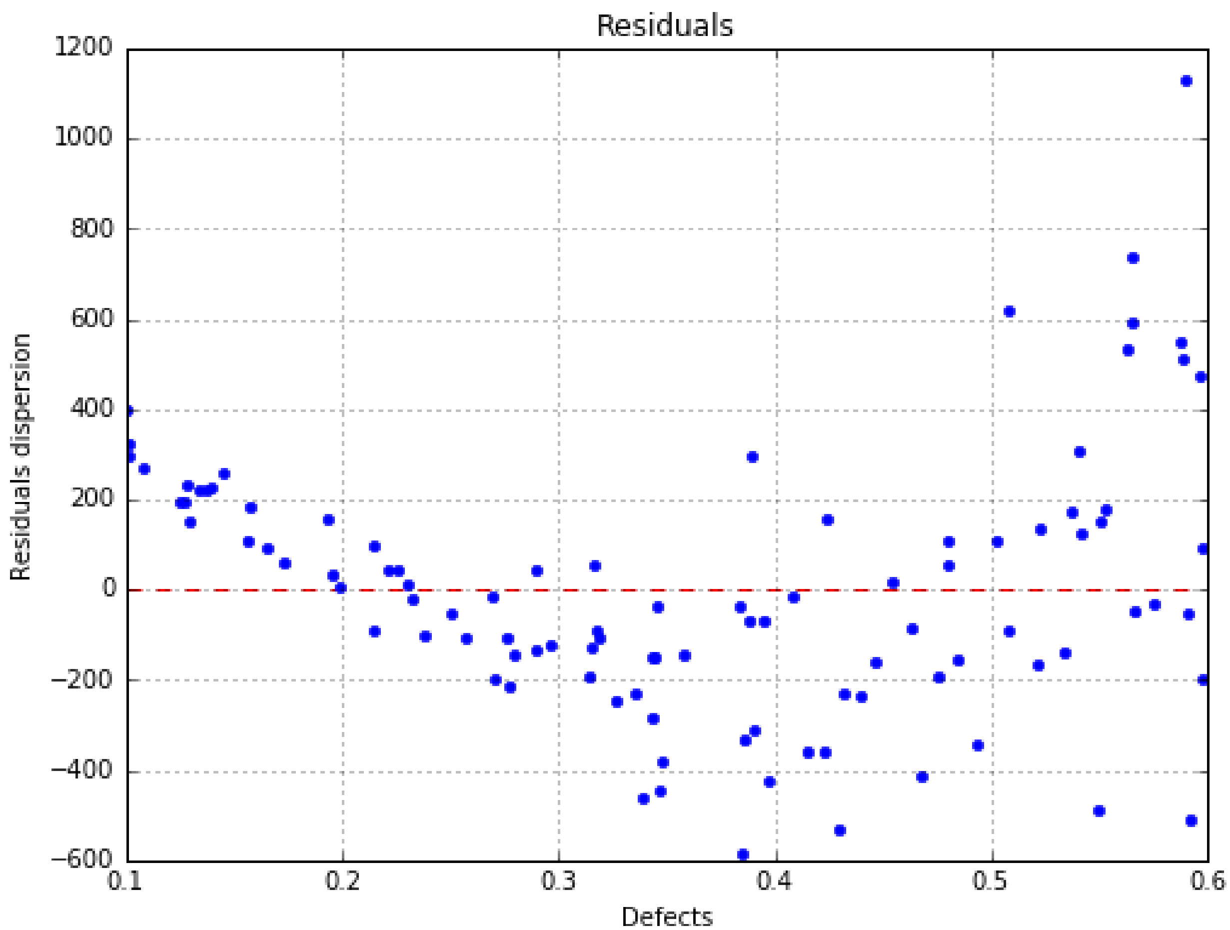
The linear model is not correct

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



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 = 1.47438e-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
--------------------------------------------------------------------------------
```

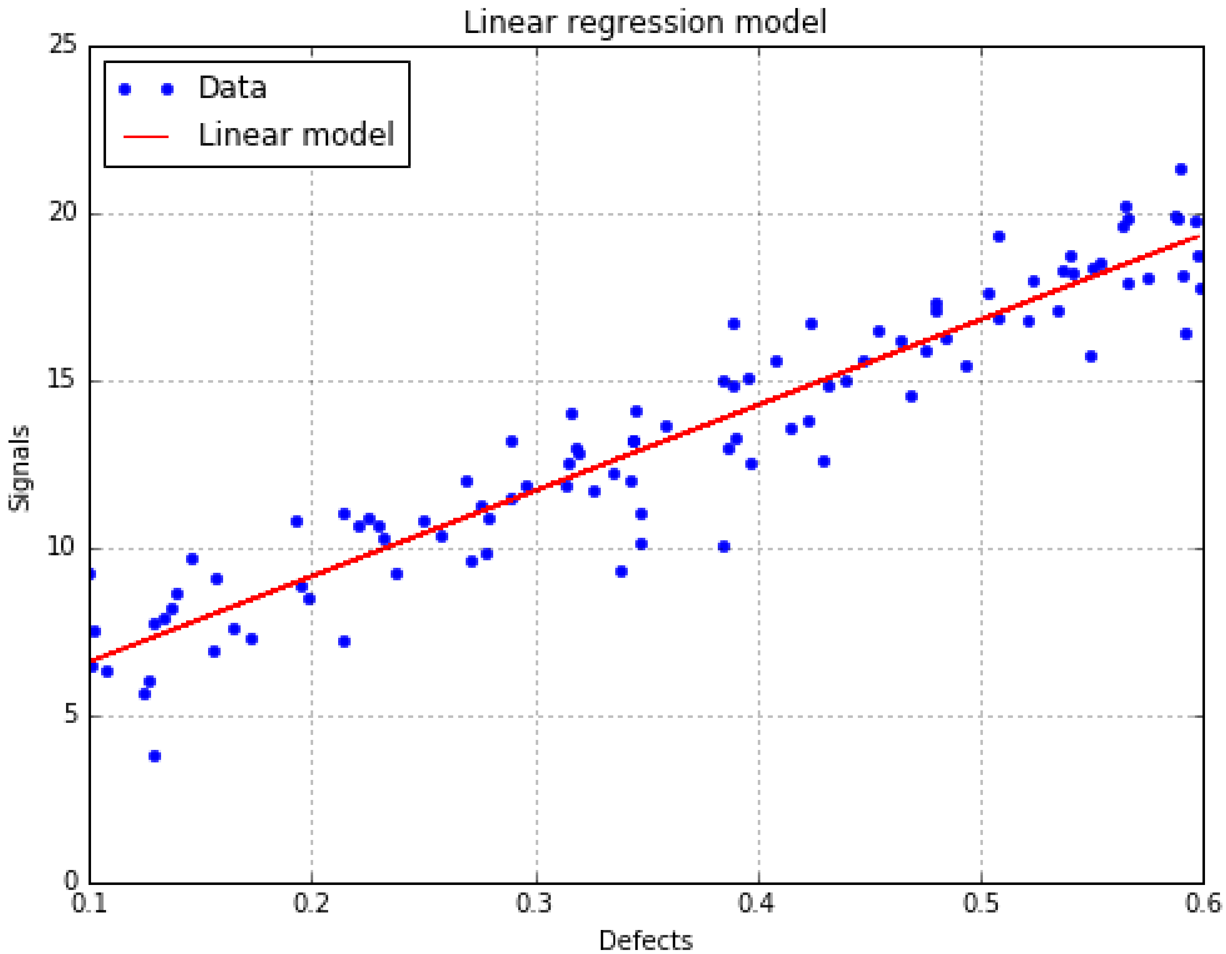
Save all results in a csv file

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

Show graphs

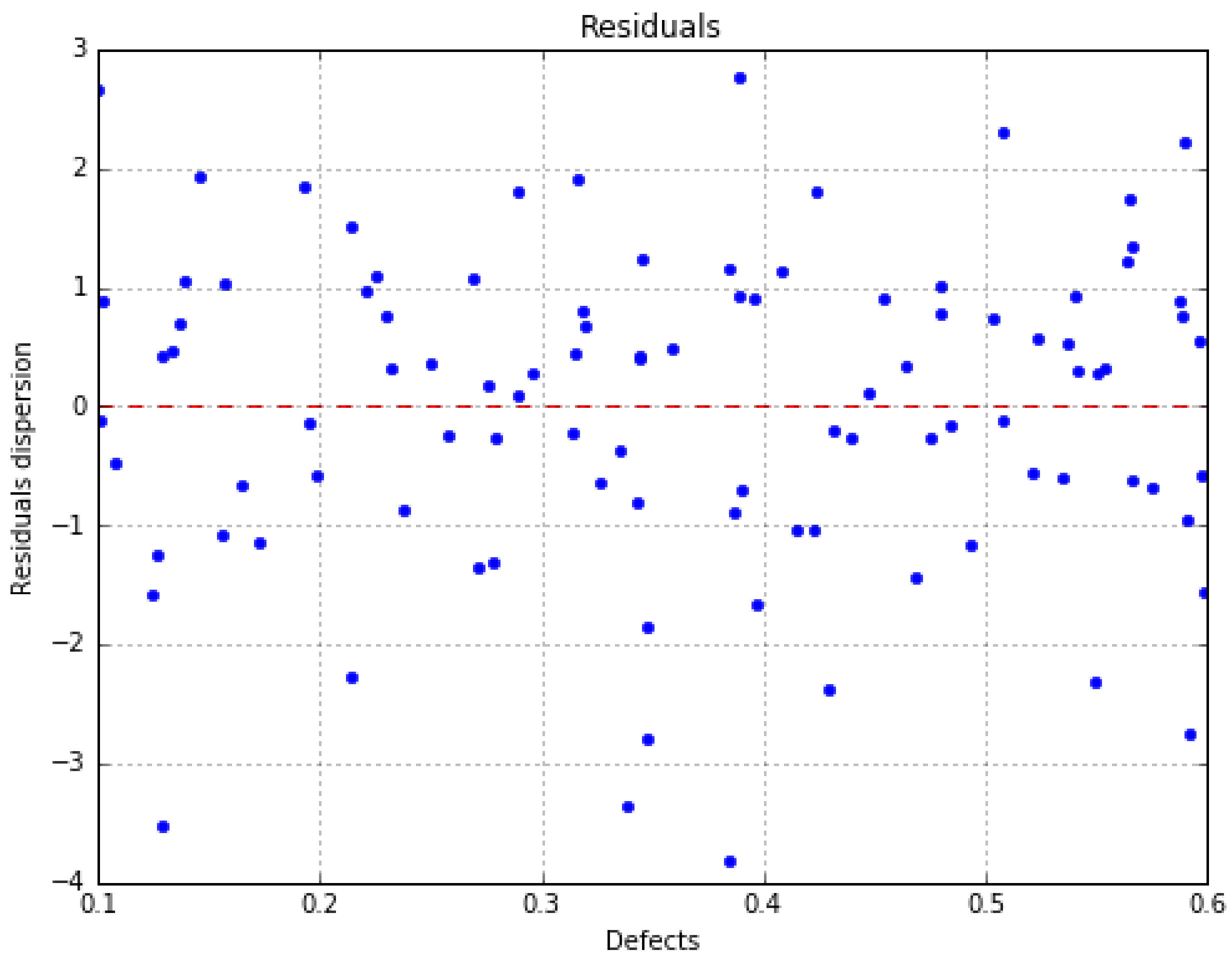
The linear regression model with data

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



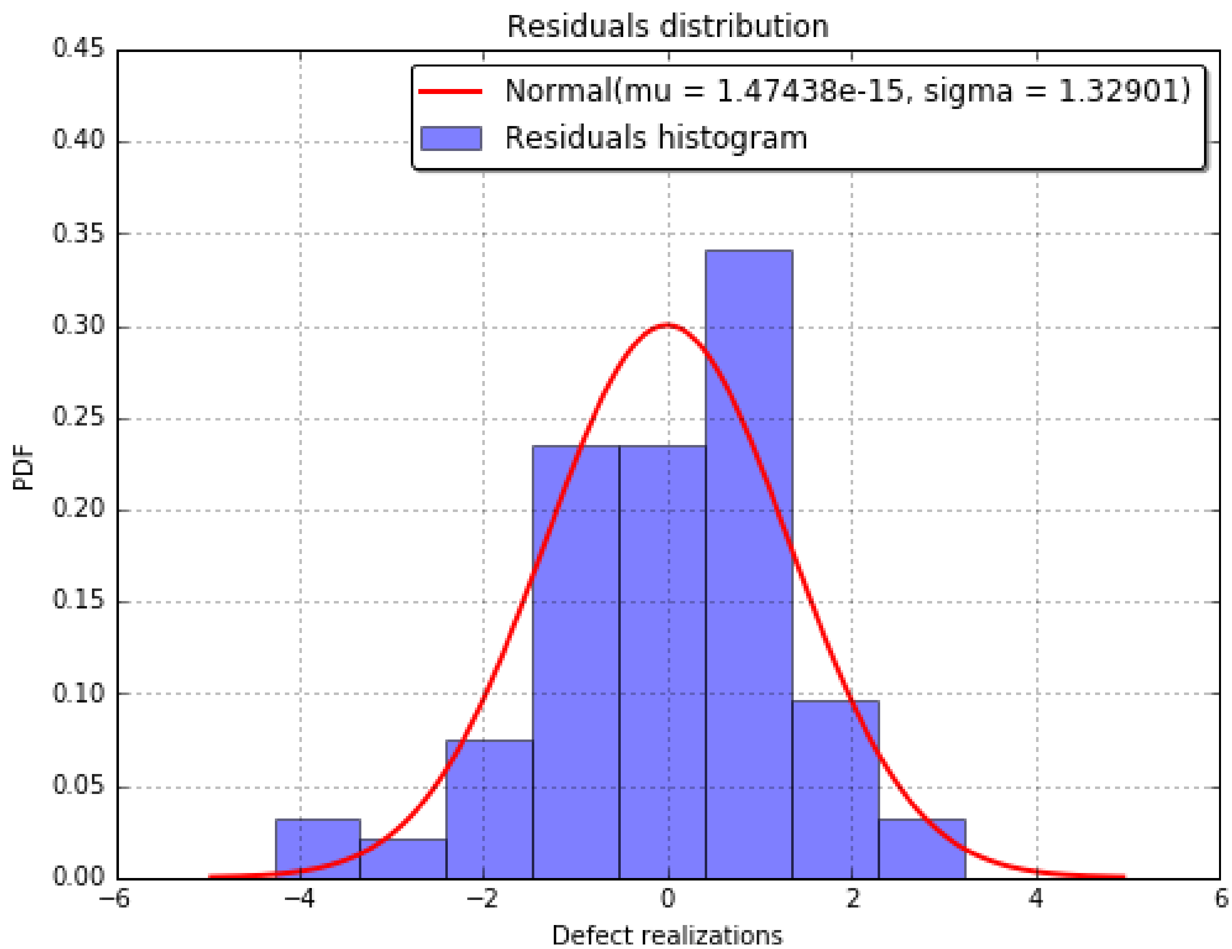
The residuals with respect to the defects

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



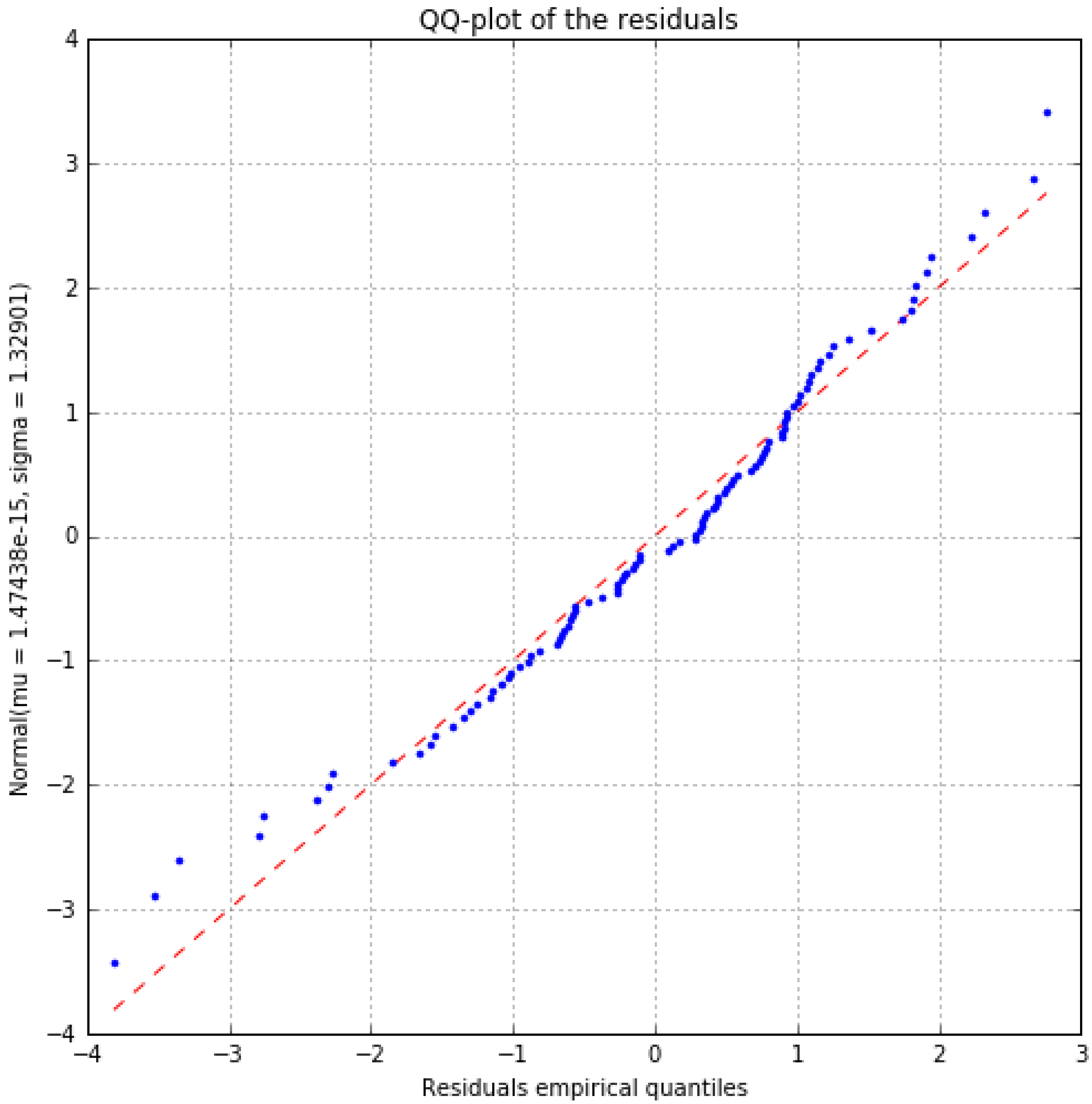
The fitted residuals distribution with the histogram

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



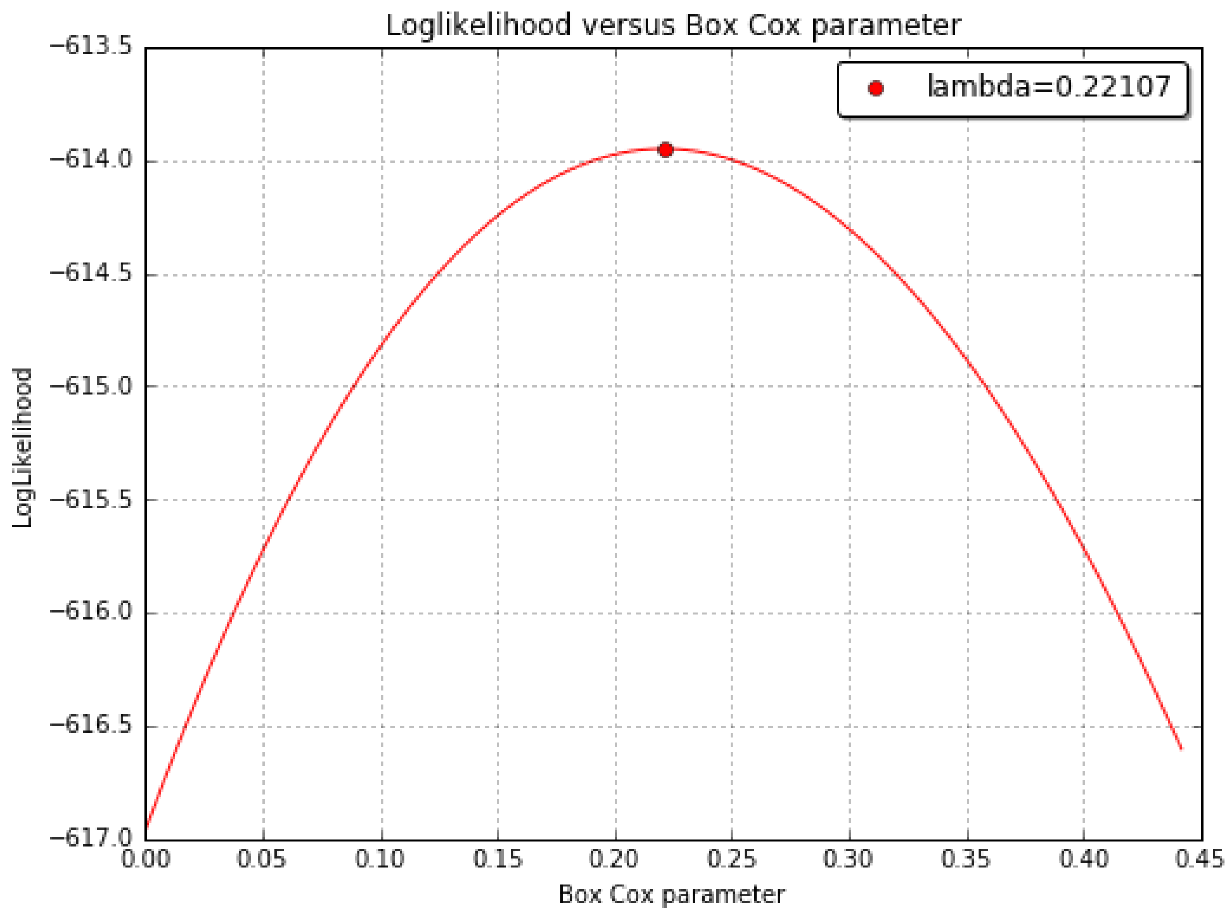
The residuals QQ plot

```
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()
```



ipynb 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
--------------------------------------------------------------------------------
```

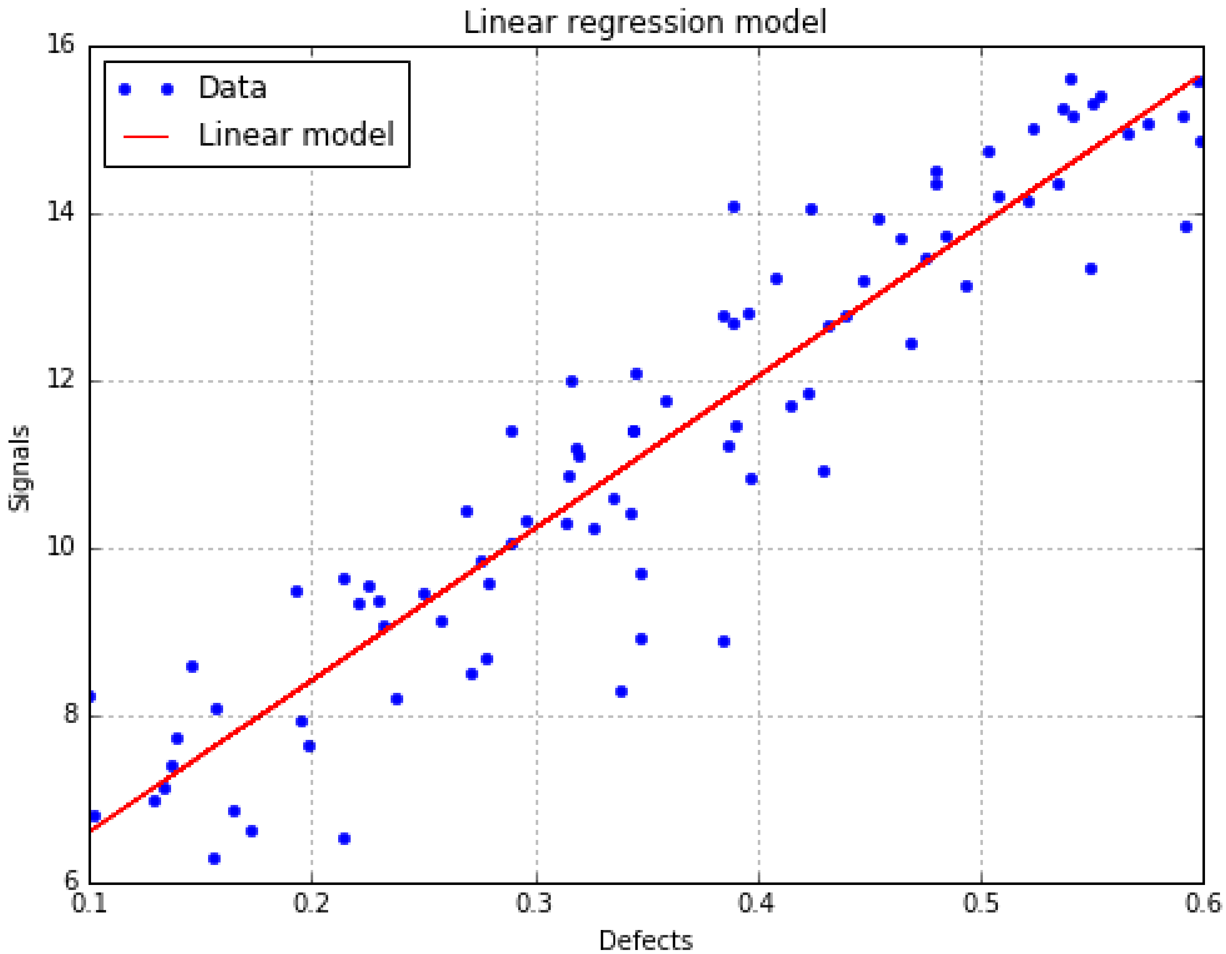
Save all results in a csv file

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

Show graphs

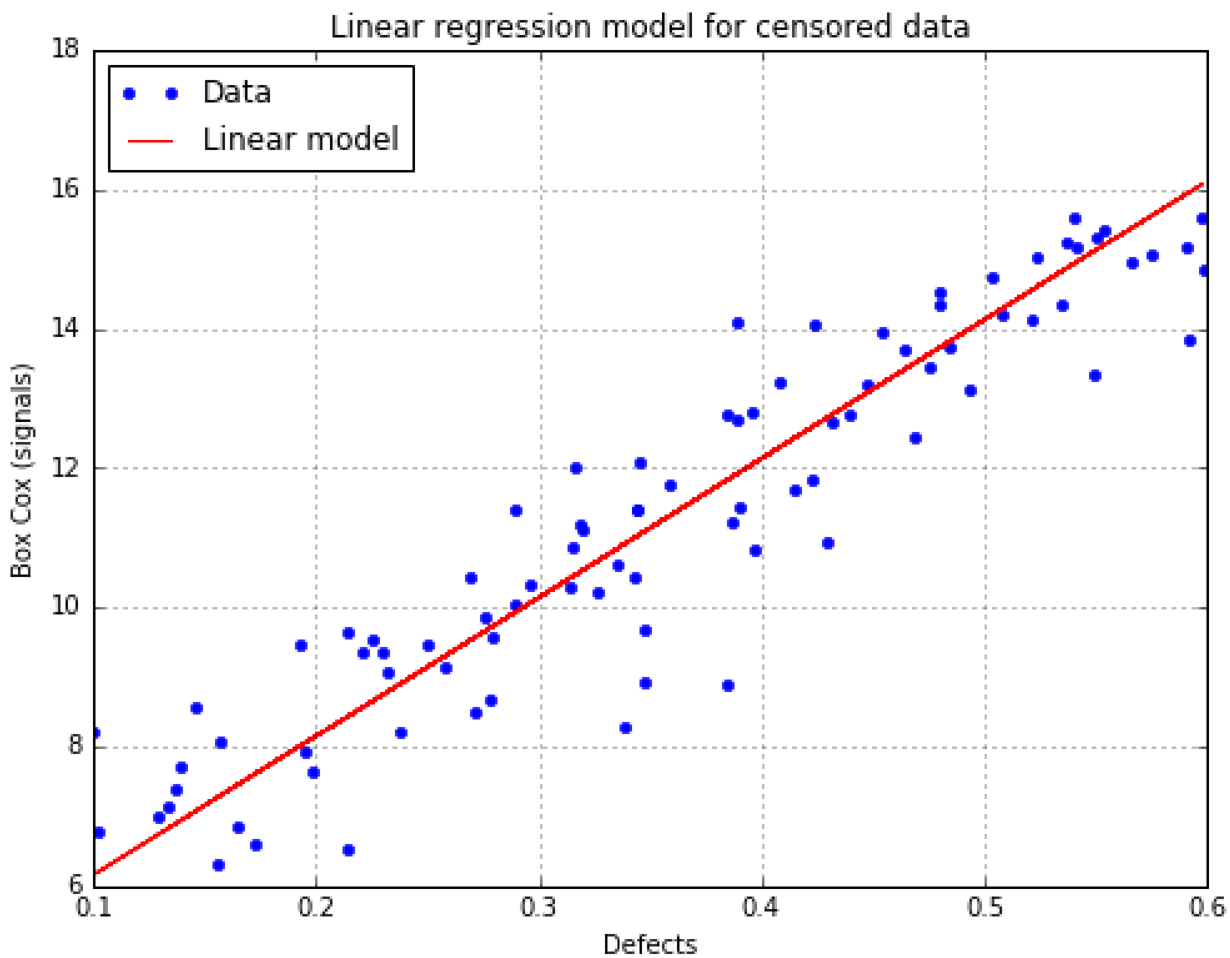
The linear regression model with data for the uncensored case (default case)

```
# 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 : {:0.3f}'.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

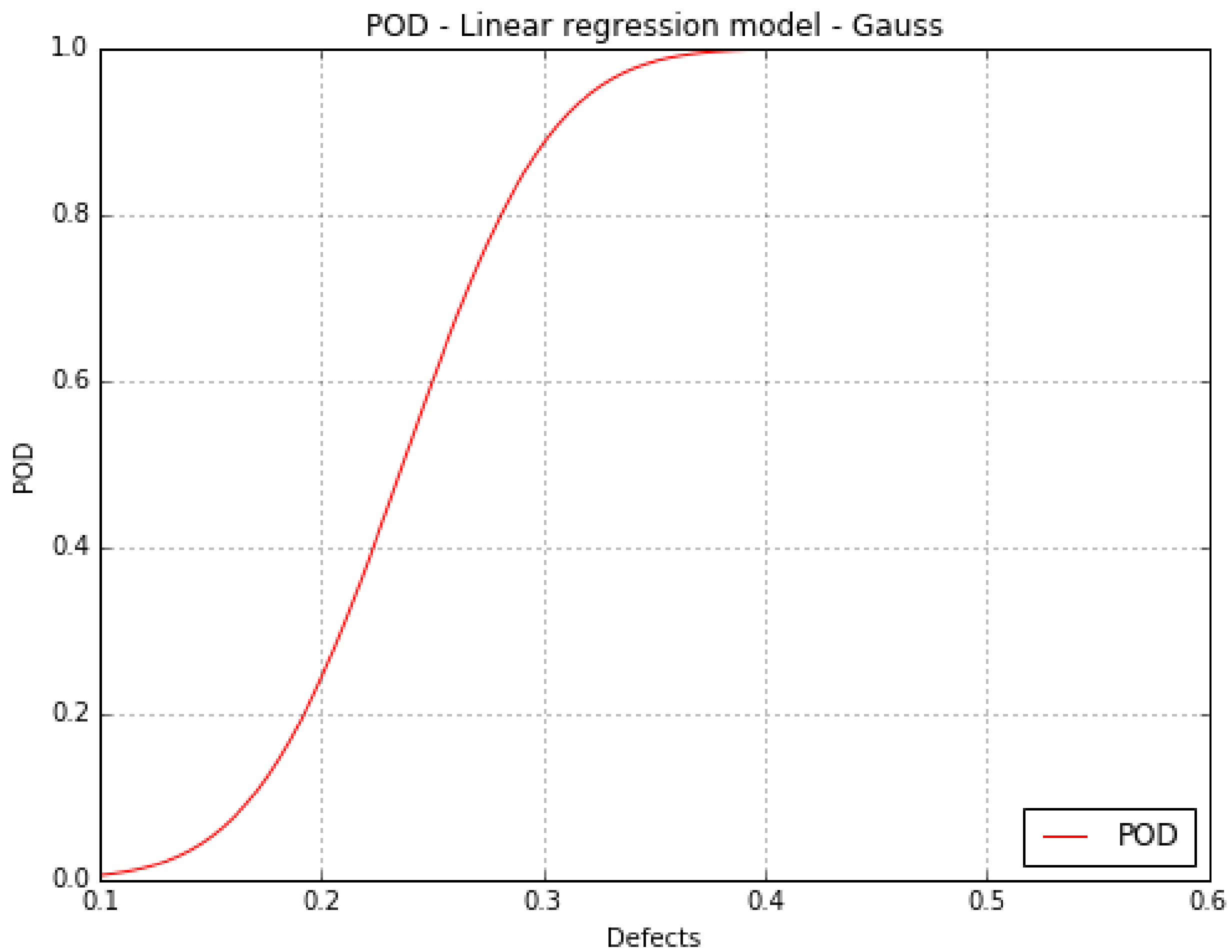
```
# get the POD model
PODmodel = PODGauss.getPODModel()
# get the POD model at the given confidence level
PODmodelCl95 = PODGauss.getPODCLModel(0.95)

# compute the probability of detection for a given defect value
print 'POD : {:0.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:0.3f}'.format(PODmodelCl95([0.3])[0])
```

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

Show POD graphs

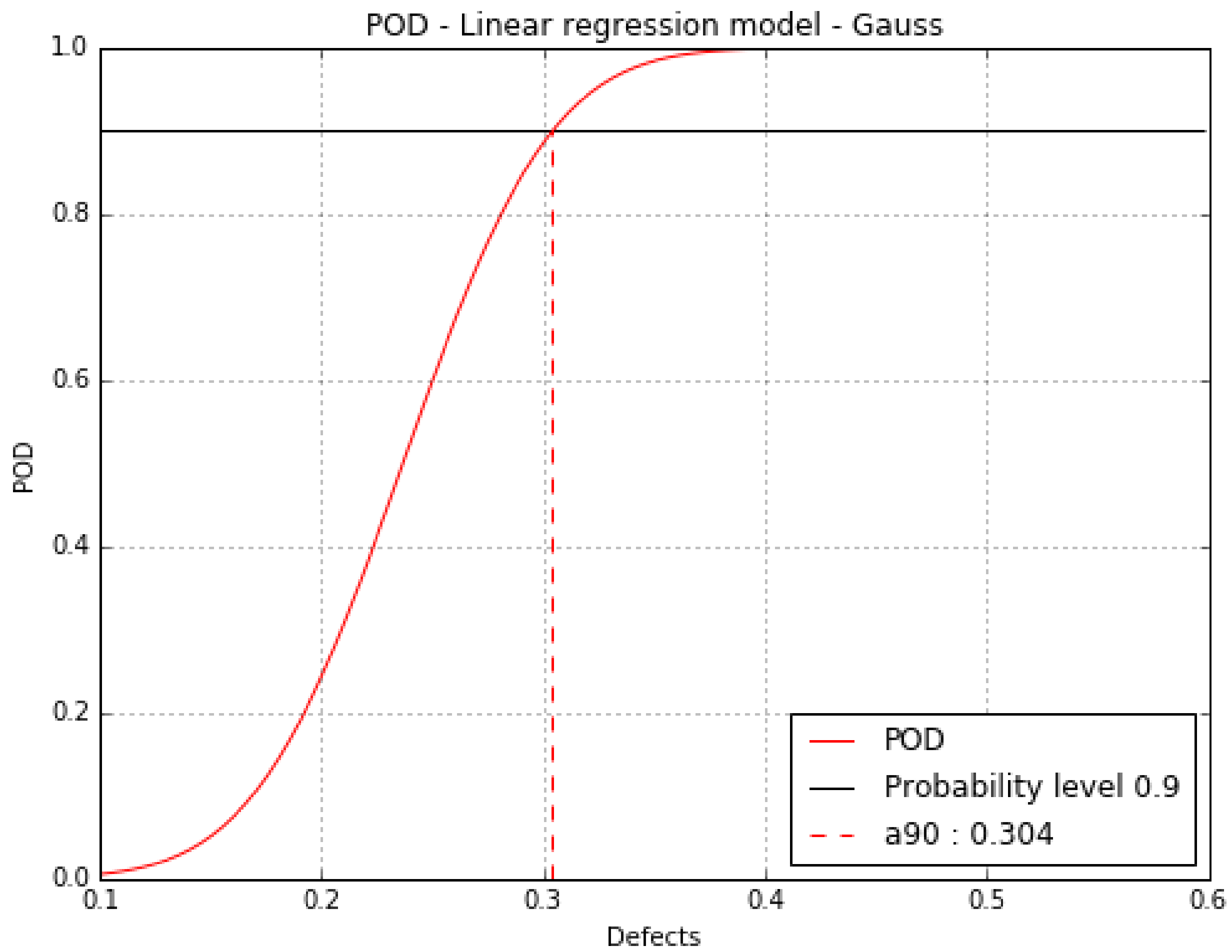
Only the mean POD

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



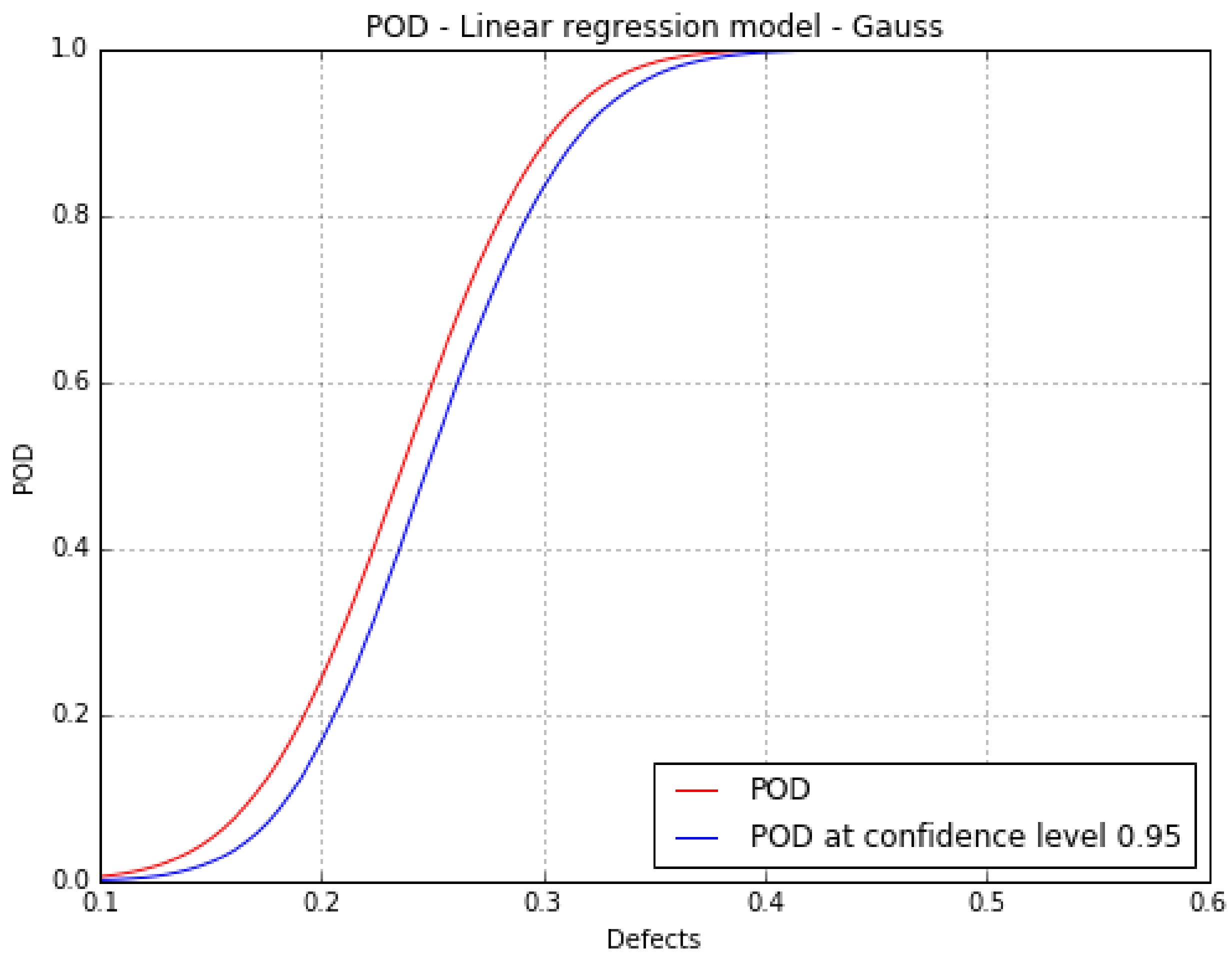
Mean POD with the detection size for a given probability level

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



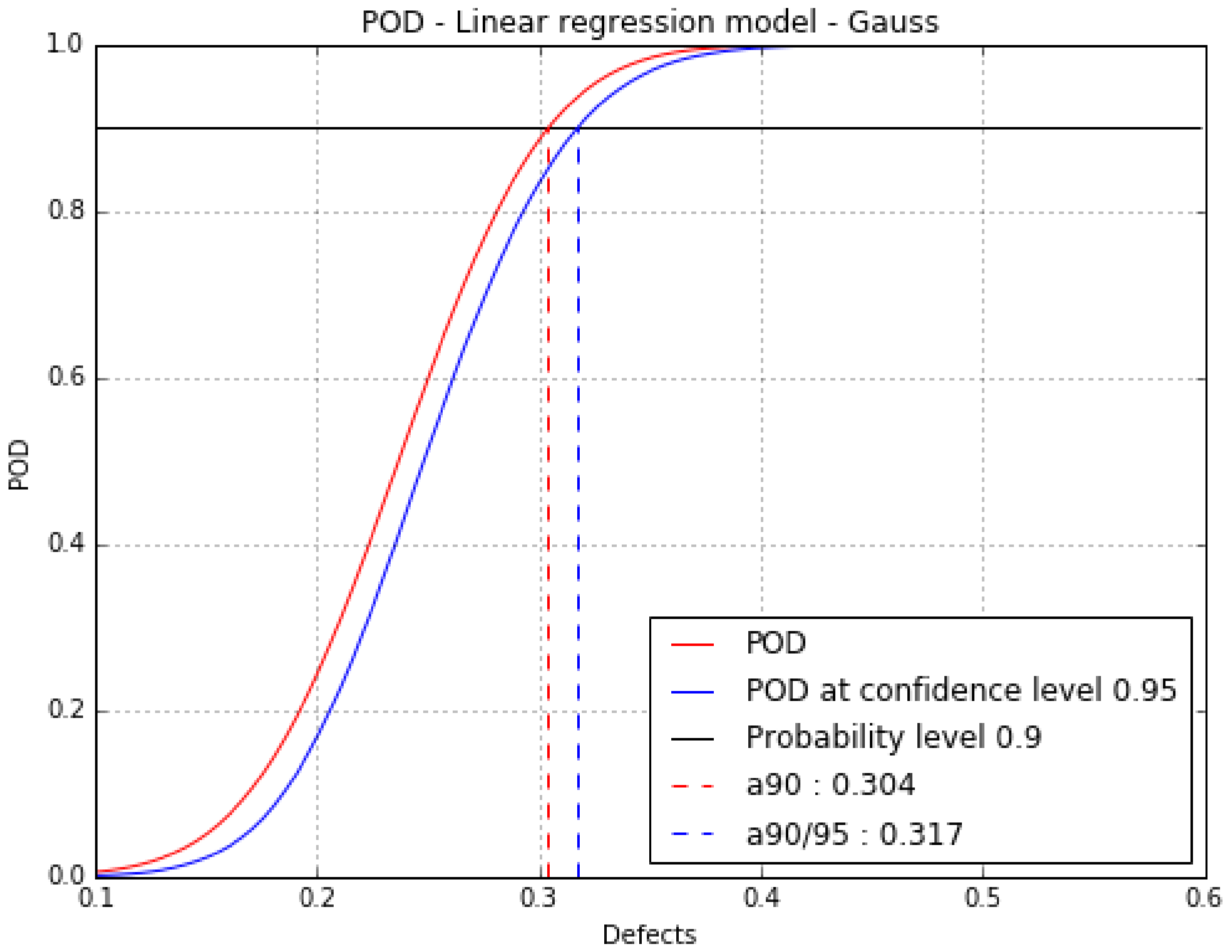
Mean POD with POD at confidence level

```
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

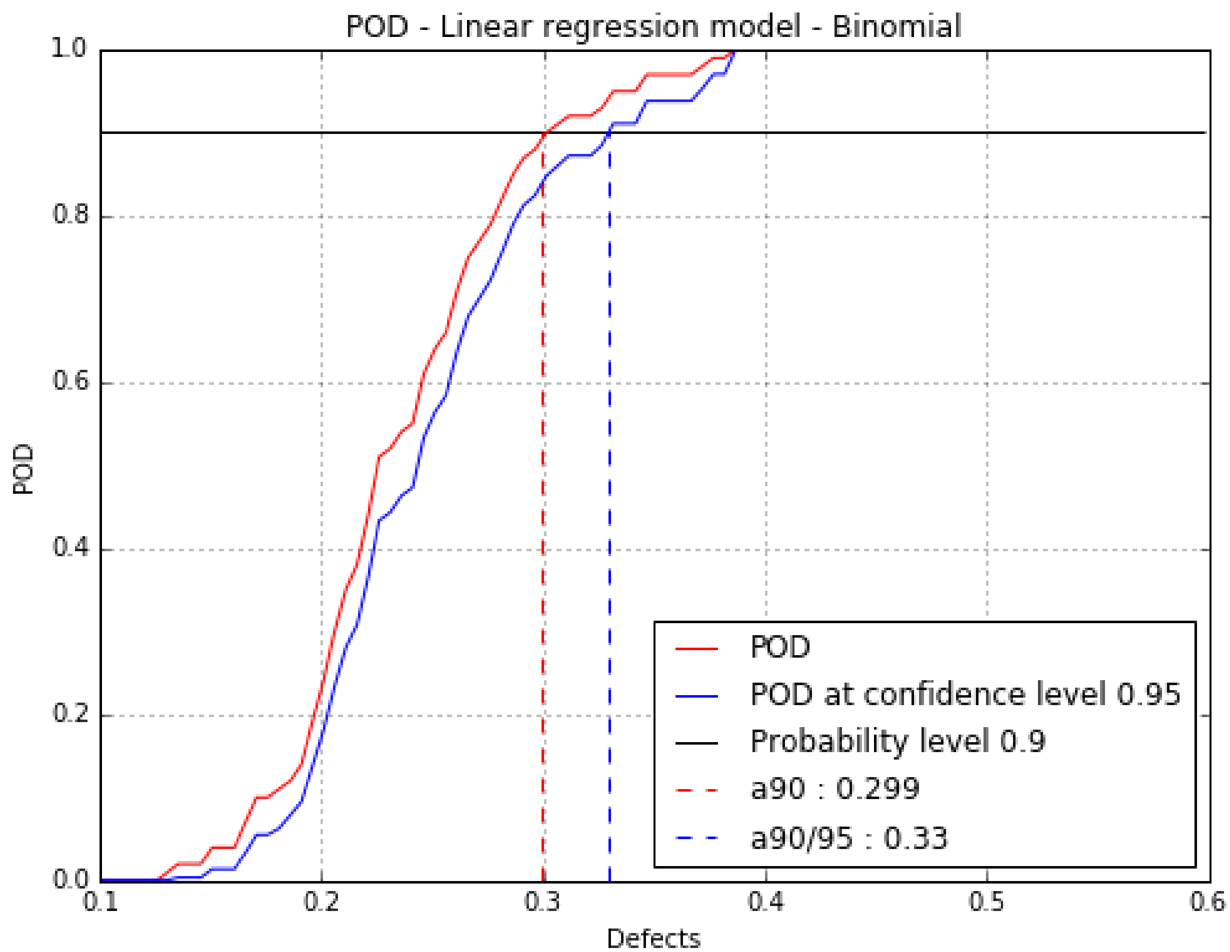
This corresponds with the Berens Binomial method.

```
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 if displayed is in this case. It can be remove using setVerbose(False)

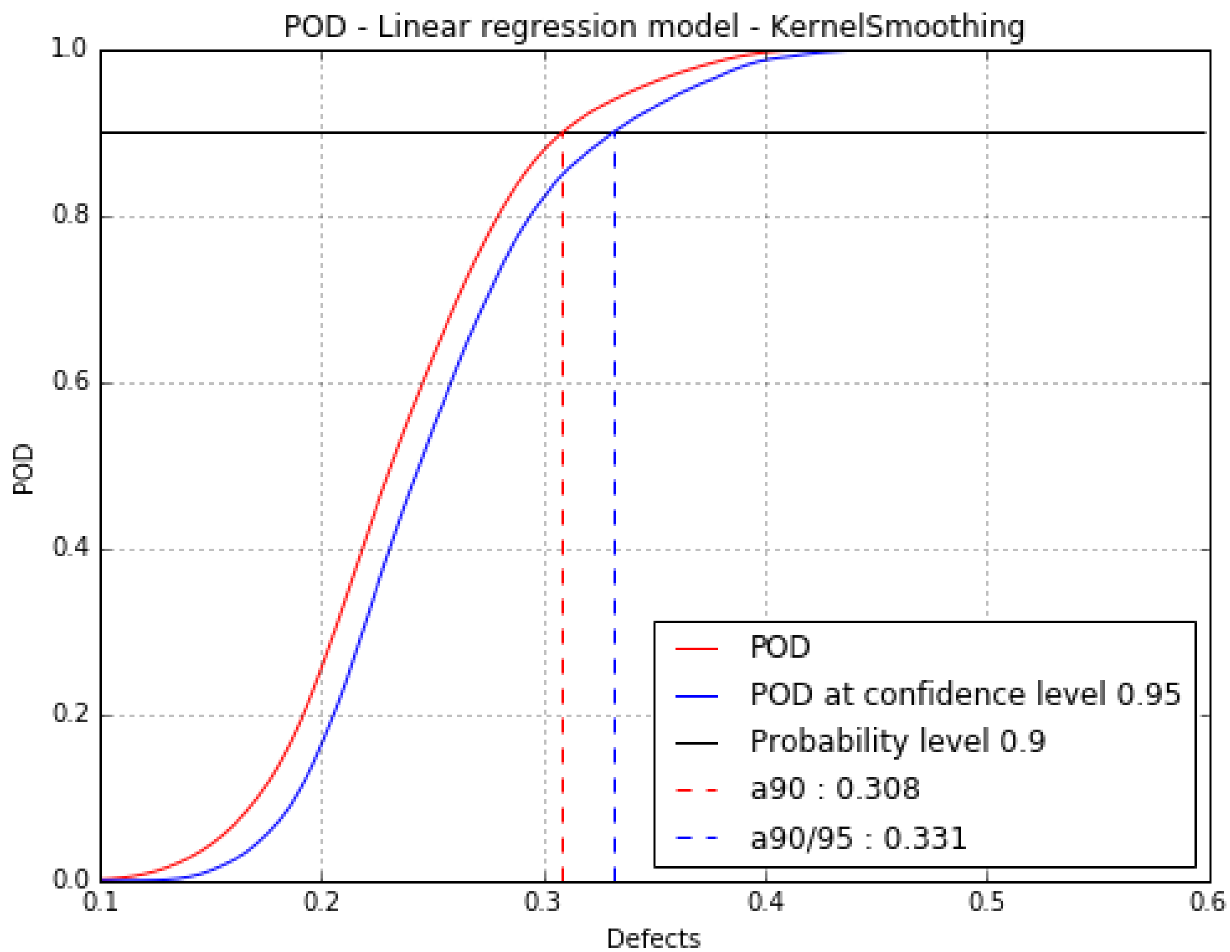
```
PODks = otpod.UnivariateLinearModelPOD(defects, signals, detection,
                                        resDistFact=ot.KernelSmoothing(),
                                        boxCox=True)
PODks.run()
```

```
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 : {:0.3f}'.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

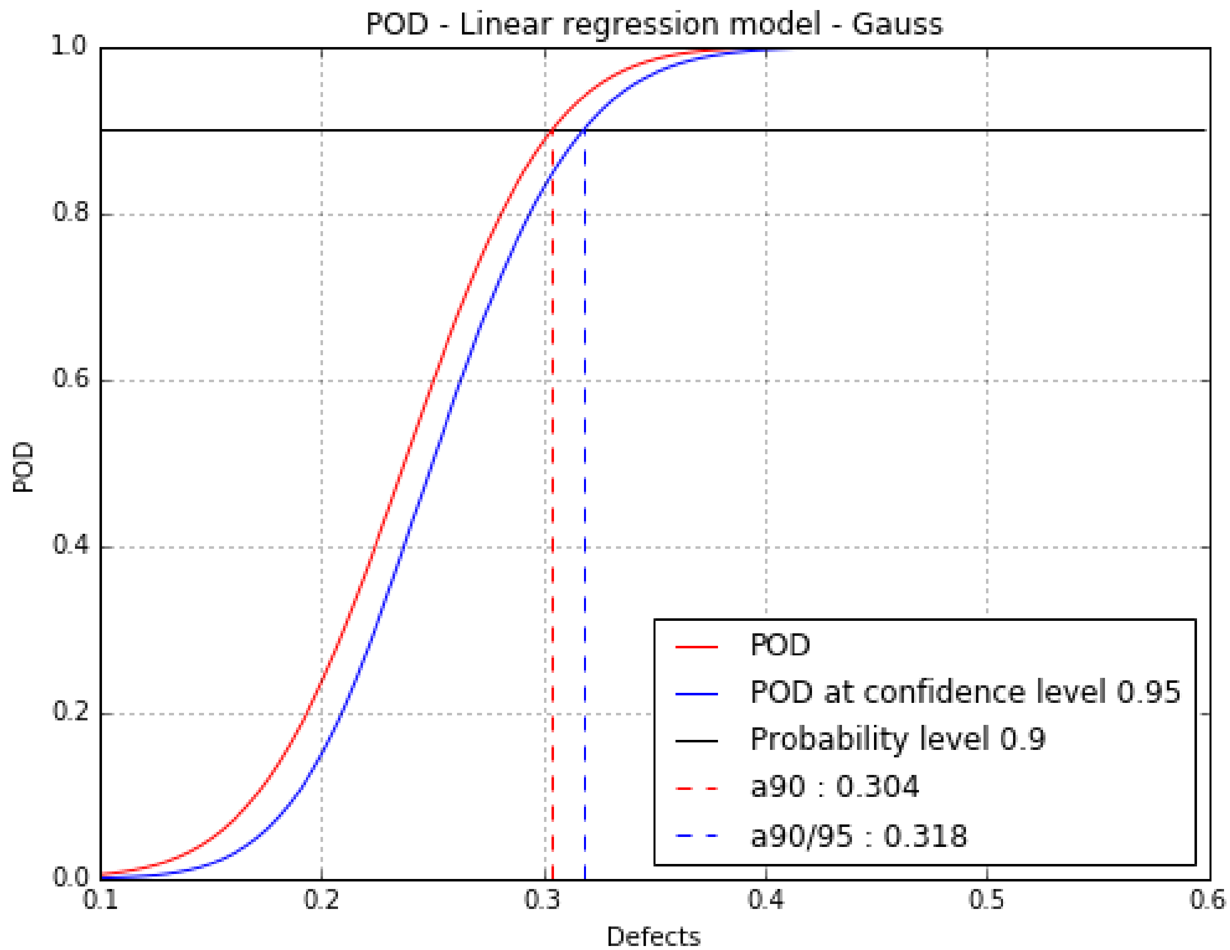
```
# get the POD model
PODmodel = PODGauss.getPODModel()
# get the POD model at the given confidence level
PODmodelCl95 = PODGauss.getPODCLModel(0.95)

# compute the probability of detection for a given defect value
print 'POD : {:0.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:0.3f}'.format(PODmodelCl95([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 defects and signals with respect to the censore threholds.

Parameters
----------
defects : 2-d sequence of float
    Vector of the defect sizes.
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
-------
defectsUnc : 2-d sequence of float
    Vector of the defect sizes in the uncensored area.
defectsNoise : 2-d sequence of float
    Vector of the defect sizes in the noisy area.
defectsSat : 2-d sequence of float
    Vector of the defect sizes 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.
```

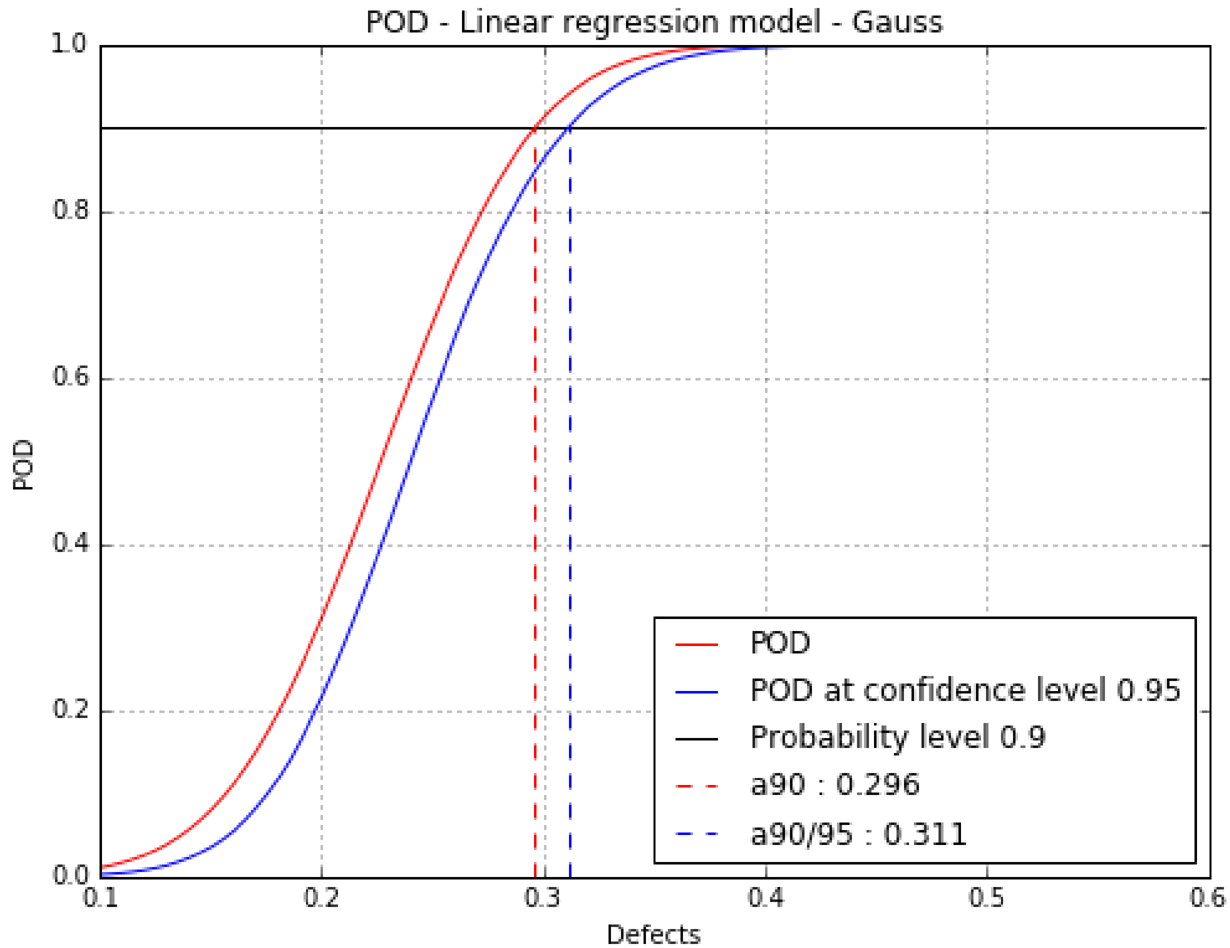
```
result = otpod.DataHandling.filterCensoredData(defects, signals,
                                               noiseThres, saturationThres)
defectsFiltered = result[0]
signalsFiltered = result[3]
```

```
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 Qantile 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 diplayed 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
```

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 : 160.07 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 : {:0.2f} s'.format(time()-t0)
```

```
Computing defect quantile: [==================================================] 100% Done
Computing time : 16.44 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
PODmodelCl95 = POD.getPODCLModel(0.95)

# compute the probability of detection for a given defect value
print 'POD : {:0.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:0.3f}'.format(PODmodelCl95([0.3])[0])
```

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

Compute the pseudo R2 for a given quantile

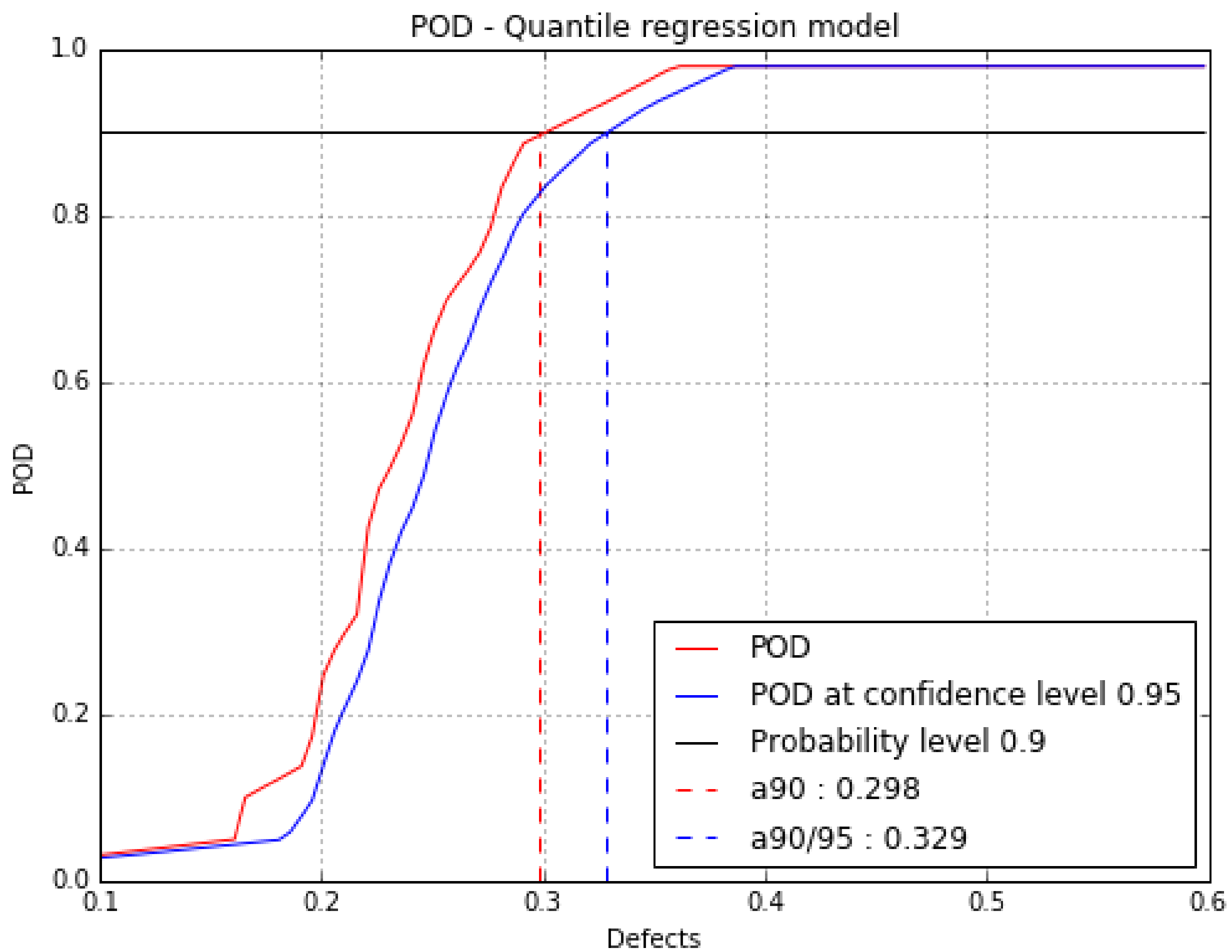
```
print 'Pseudo R2 for quantile 0.9 : {:0.3f}'.format(POD.getR2(0.9))
print 'Pseudo R2 for quantile 0.95 : {:0.3f}'.format(POD.getR2(0.95))
```

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

Show POD graphs

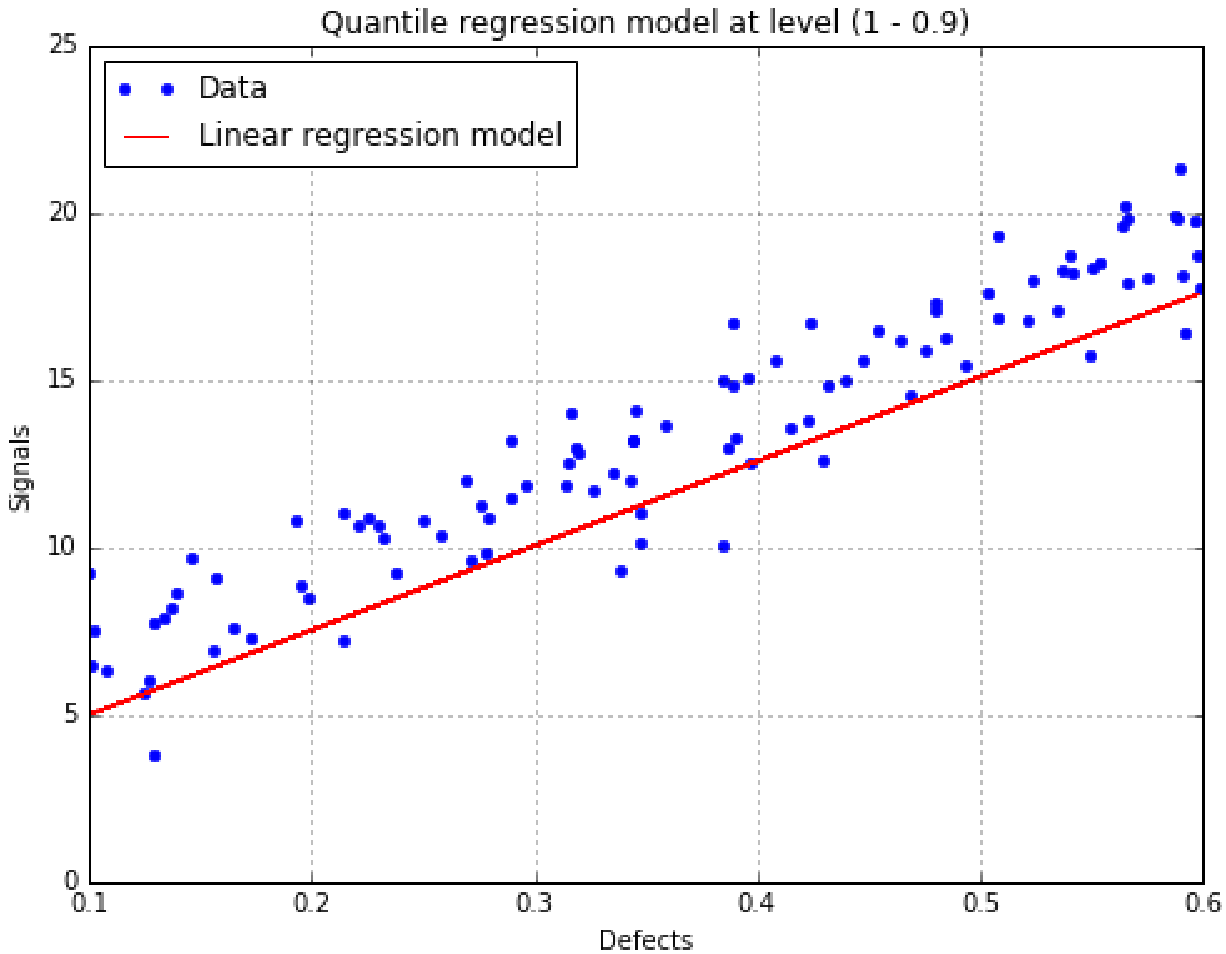
Mean POD and POD at confidence level with the detection size for a given probability level

```
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
from time import time
```

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 diplayed in this case.
POD = otpod.PolynomialChaosPOD(defects, signals, detection,
                               noiseThres=200., saturationThres=1700.,
                               boxCox=True)
```

```
INFO:root:Censored data are not taken into account : the polynomial chaos model is only built on filt
```

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-24-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)
    409             raise ValueError('Defect sizes must range between ' + \
    410                              '{:0.4f} '.format(np.ceil(minMin*10000)/10000) + \
--> 411                              'and {:0.4f}.'.format(np.floor(maxMax*10000)/10000))
    412         self._defectNumber = self._defectSizes.shape[0]
    413


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

```
Start build polynomial chaos model...
Polynomial chaos model completed
Computing POD per defect: [==================================================] 100% Done
Computing time : 67.72 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 : 0.300985, a90/95 : 0.309081]
[a95 : 0.324088, a95/99 : 0.334221]
```

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 : {:0.3f}'.format(PODmodel([0.3])[0])
print 'POD at level 0.95 : {:0.3f}'.format(PODmodelCl95([0.3])[0])
```

```
POD : 0.897
POD at level 0.95 : 0.861
```

Compute the R2 and the Q2

Enable to check the quality of the model.

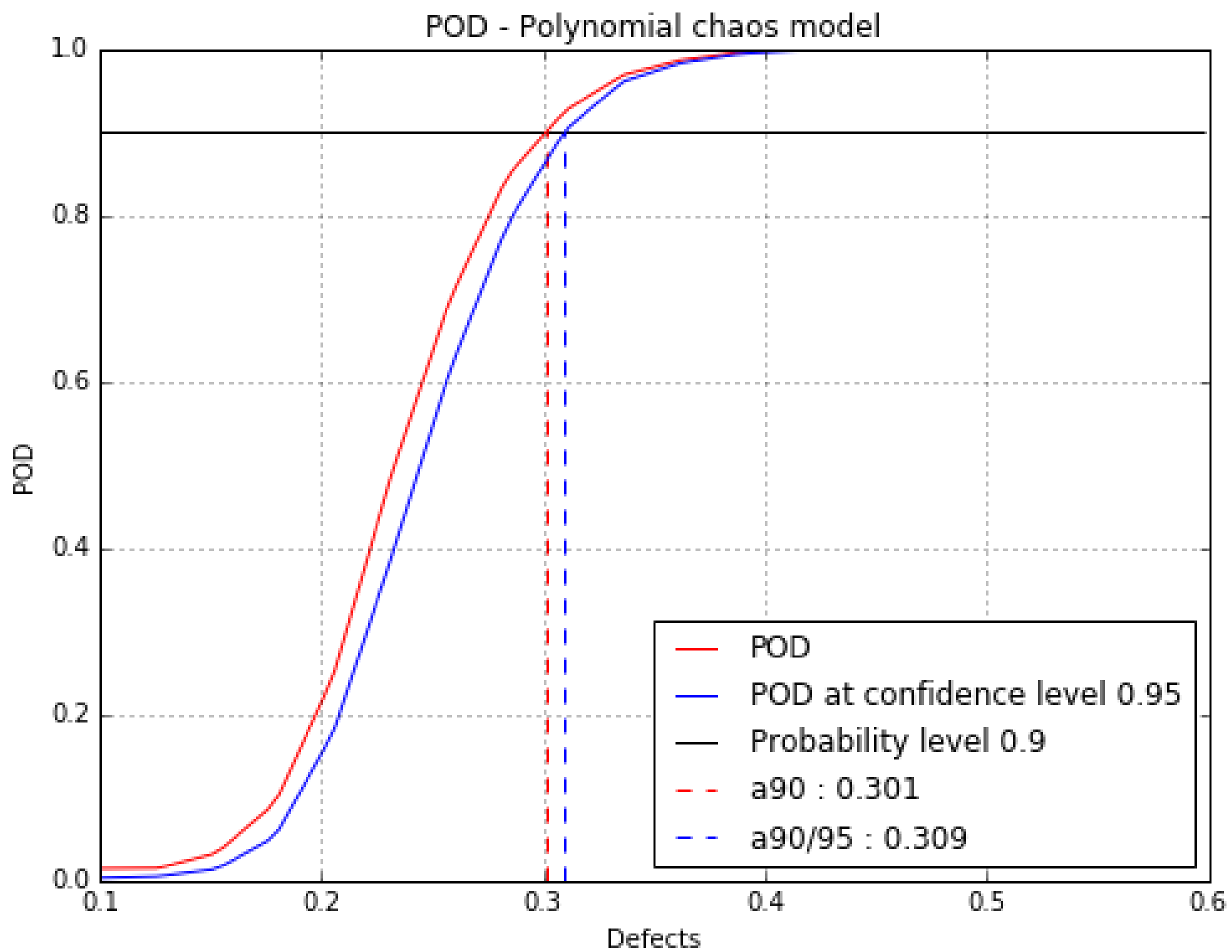
```
print 'R2 : {:0.4f}'.format(POD.getR2())
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

```
R2 : 0.8975
Q2 : 0.8922
```

Show POD graphs

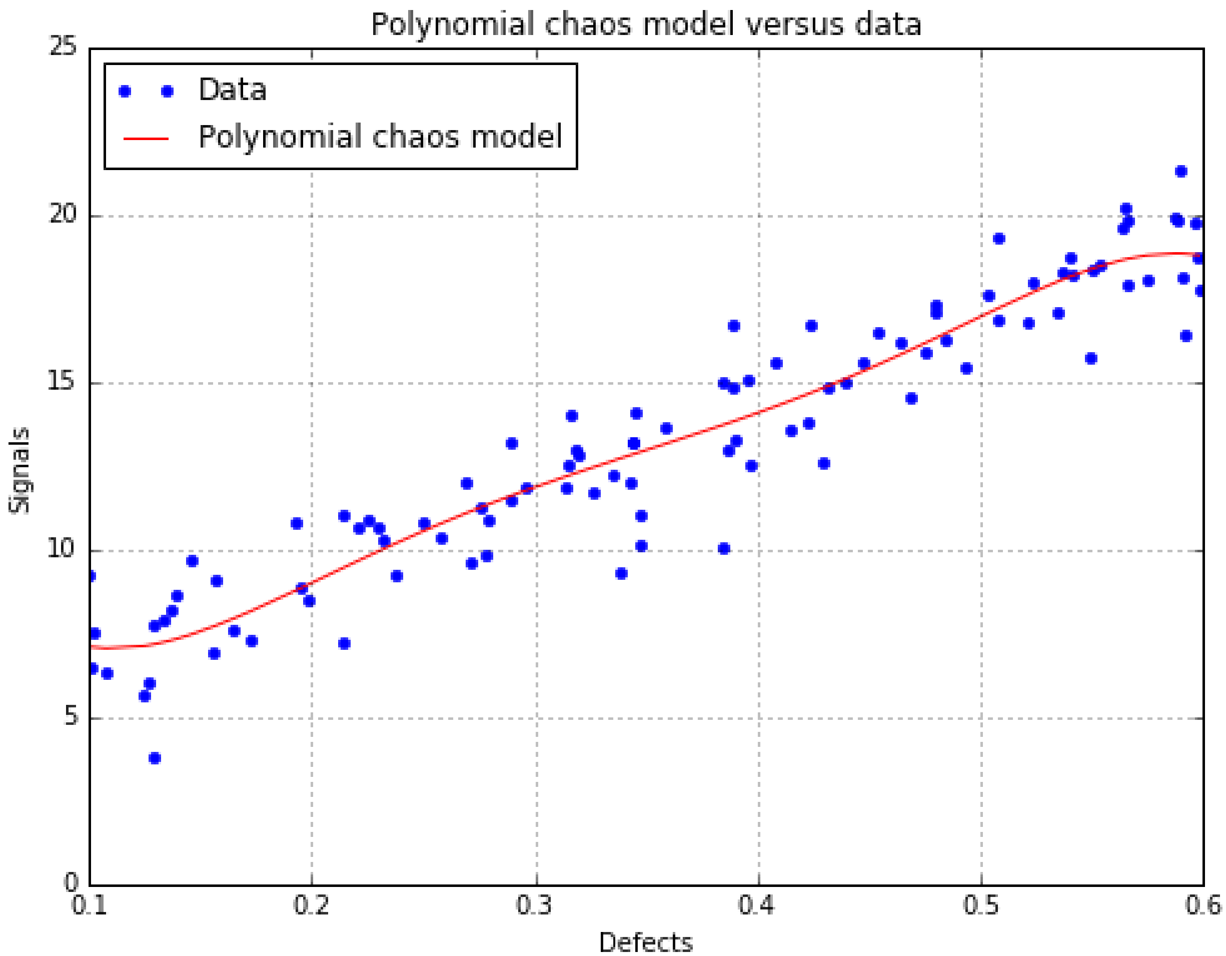
Mean POD and POD at confidence level with the detection size for a given probability level

```
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)
```

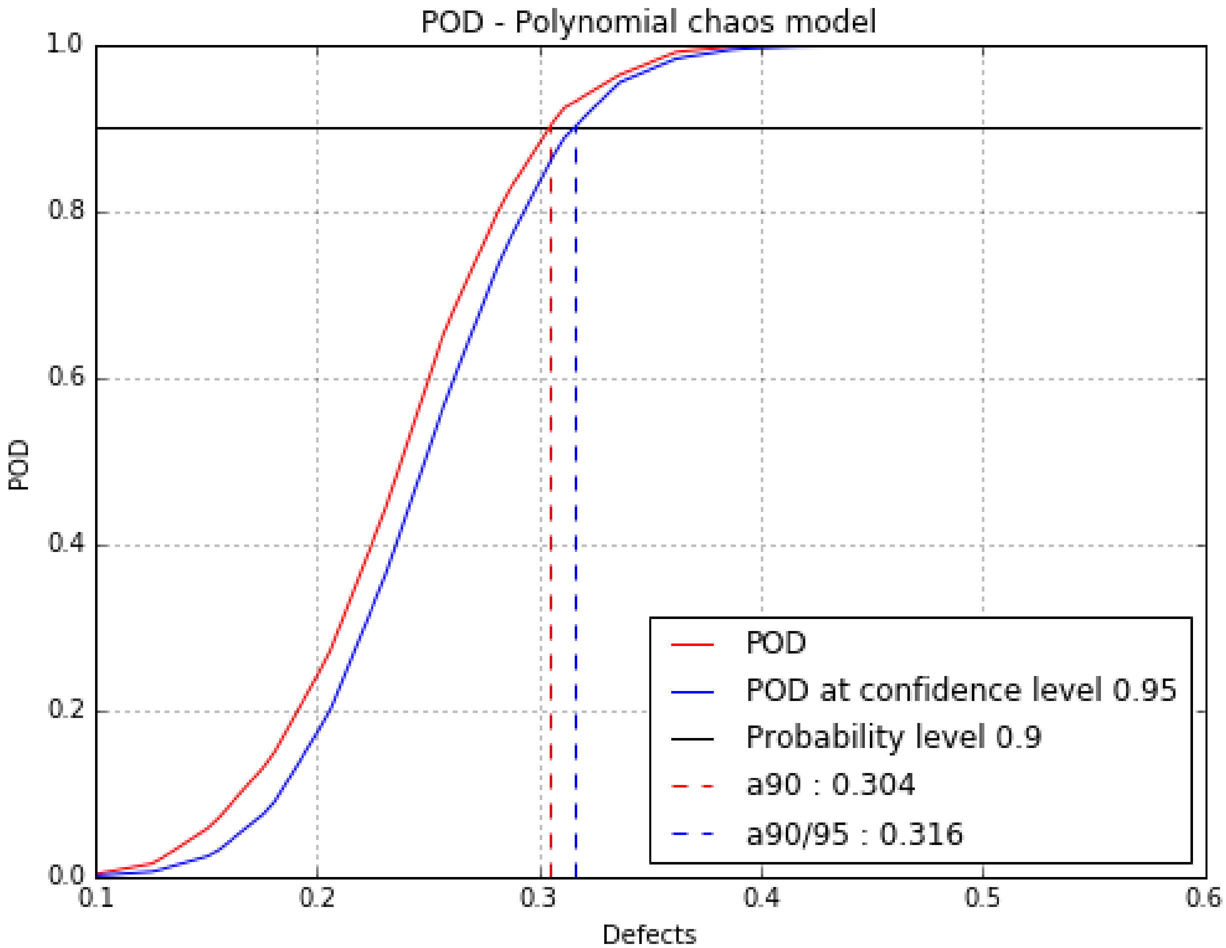
```
PODnew.run()
```

```
Start build polynomial chaos model...
Polynomial chaos model completed
Computing POD per defect: [==================================================] 100% Done
```

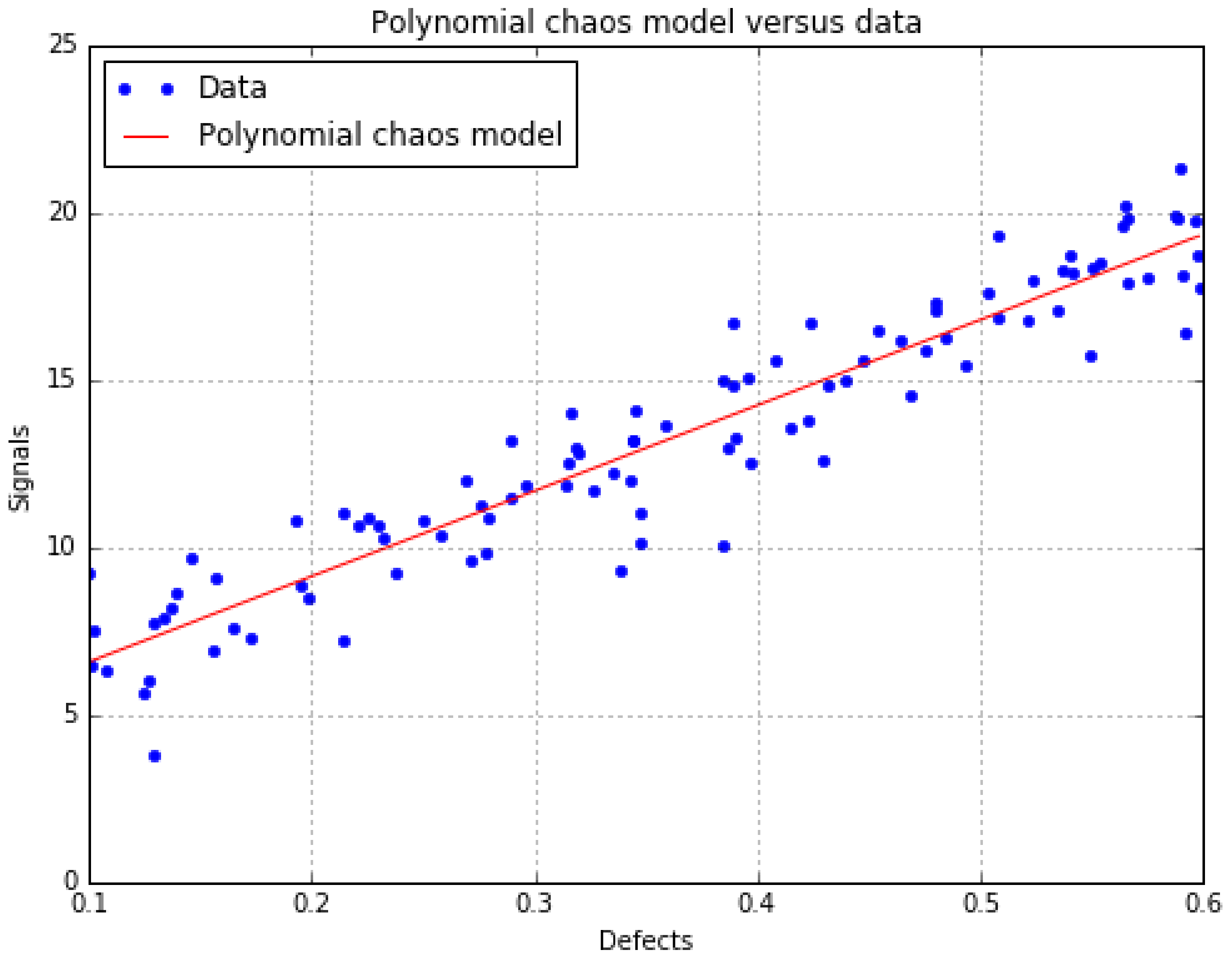
```
print PODnew.computeDetectionSize(0.9, 0.95)
print 'R2 : {:0.4f}'.format(POD.getR2())
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

```
[a90 : 0.304485, a90/95 : 0.315531]
R2 : 0.8975
Q2 : 0.8922
```

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




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



ipynb source code

1.2.7 Kriging POD

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

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 diplayed 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_pod.py in setDefectSizes(self, size)
    376             raise ValueError('Defect sizes must range between ' + \
    377                              '{:0.4f} '.format(np.ceil(minMin*10000)/10000) + \
--> 378                              'and {:0.4f}.'.format(np.floor(maxMax*10000)/10000))
    379         self._defectNumber = self._defectSizes.shape[0]
    380


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()
```

```
User-defined defect size :
[ 4.   4.3  4.6  4.9  5.1]
```

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.
t0 = time()
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 10000
# we can also change the size of the simulation to compute the confidence interval
POD.setSimulationSize(500) # default is 1000
POD.run()
print 'Computing time : {:0.2f} s'.format(time()-t0)
```

```
Start optimizing covariance model parameters...
Kriging optimizer completed
Computing POD per defect: [==================================================] 100% Done
Computing time : 406.09 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 : 4.64491, a90/95 : 4.65305]
[a95 : 4.6813, a95/99 : 4.69299]
```

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 : {:0.3f}'.format(PODmodel([4.2])[0])
print 'POD at level 0.95 : {:0.3f}'.format(PODmodelCl95([4.2])[0])
```

```
POD : 0.154
POD at level 0.95 : 0.141
```

Compute the Q2

Enable to check the quality of the model.

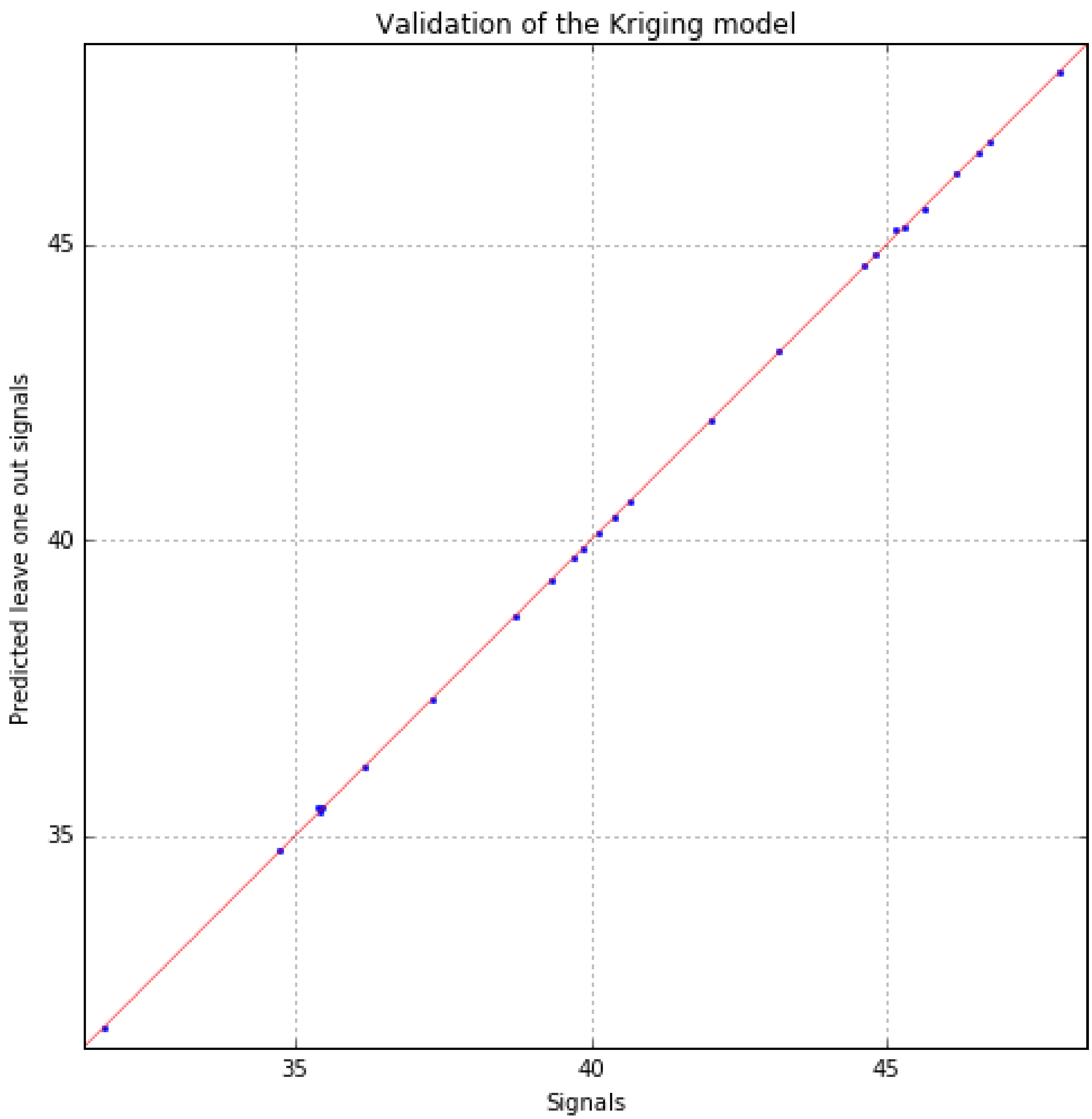
```
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

```
Q2 : 1.0000
```

Draw the validation graph

The predictions are the one computed by leave one out.

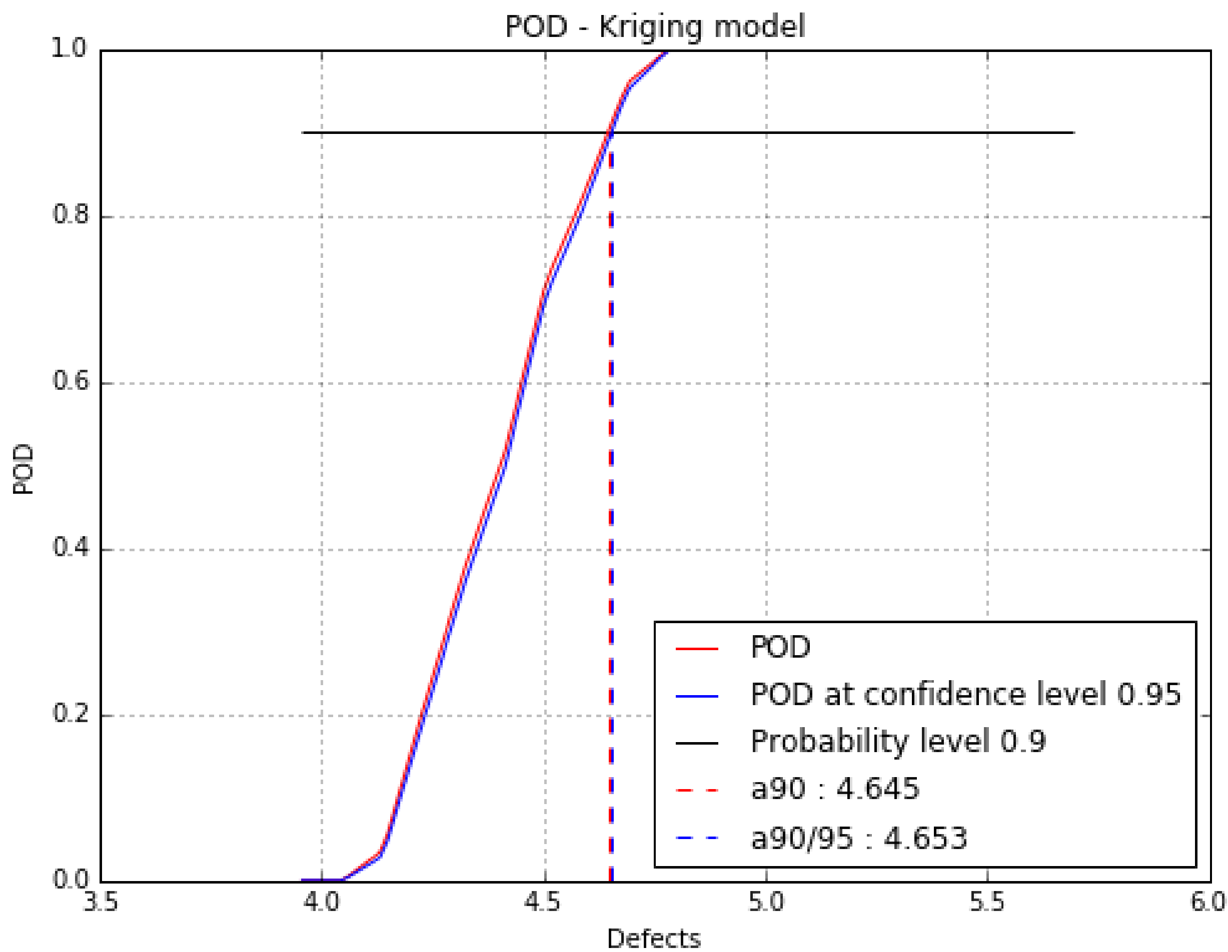
```
fig, ax = POD.drawValidationGraph()
fig.show()
```



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)
```

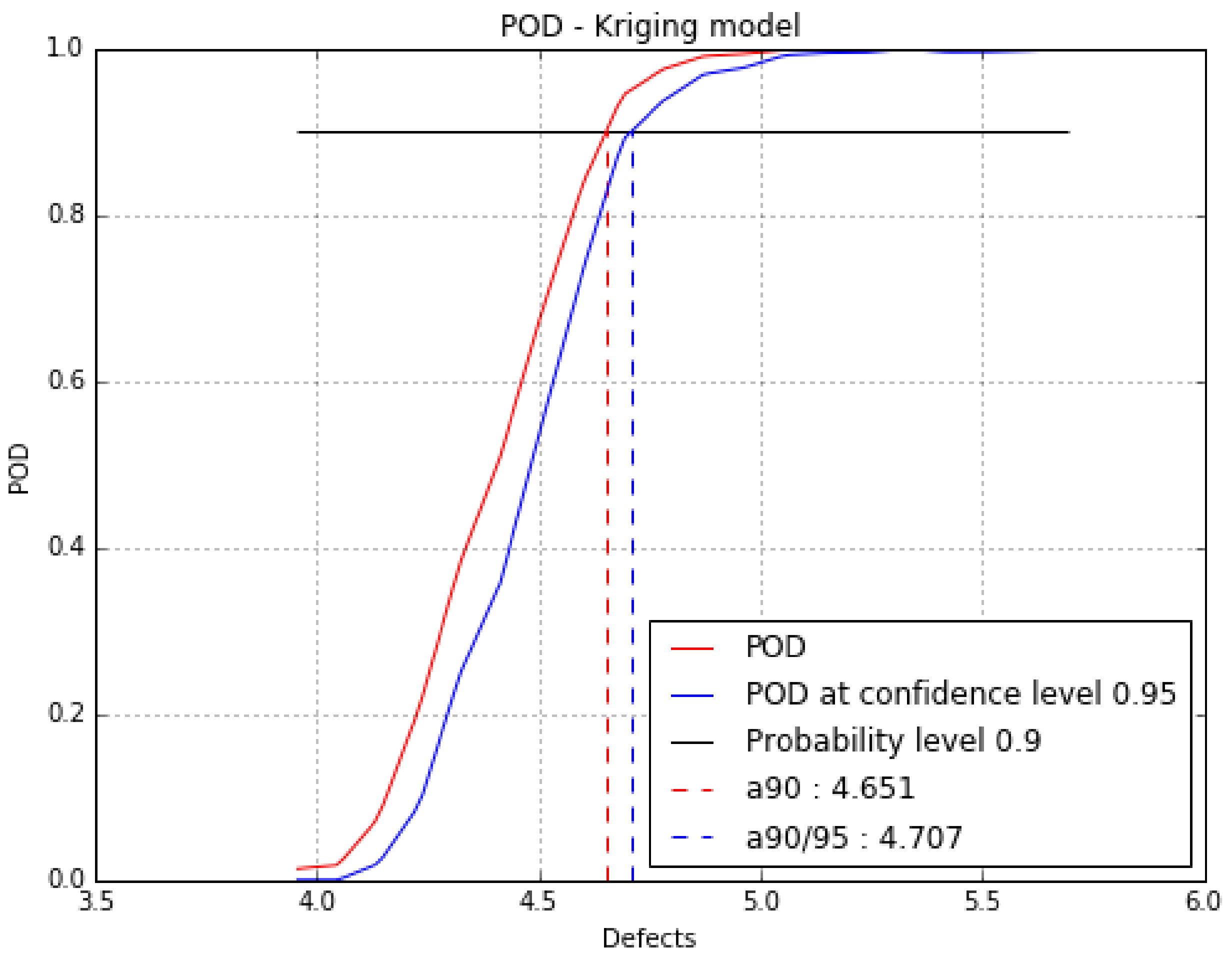
```
PODnew.setSamplingSize(500)
PODnew.run()
```

```
Start optimizing covariance model parameters...
Kriging optimizer completed
Computing POD per defect: [==================================================] 100% Done
```

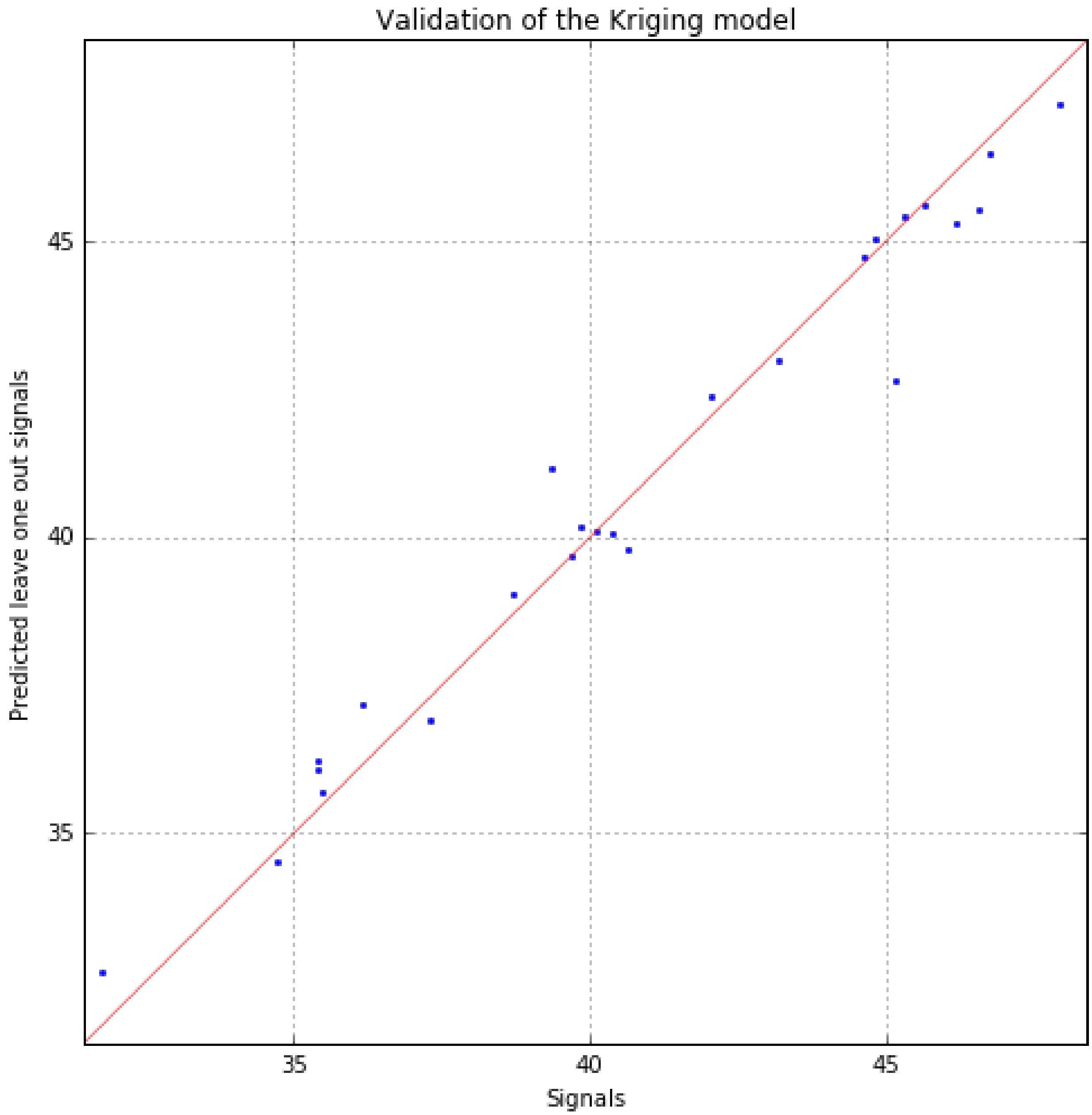
```
print PODnew.computeDetectionSize(0.9, 0.95)
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

```
[a90 : 4.65125, a90/95 : 4.70682]
Q2 : 1.0000
```

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




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


Validation of the Kriging model
Predicted leave one out signals
Signals
35
40
45
35
40
45

CHAPTER TWO

INDICES AND TABLES

C

D

F

G

K

P

Q

R

S

U