otpod Documentation

Release

Antoine Dumas

April 22, 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.

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.

Methods

computeDetectionSize(*args, **kwargs)	Compute the detection size for a given probability level.
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.
run()	Build the POD models.
setSimulationSize(size)	Accessor to the simulation size

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 list of NumericalPointWithDescription containing the detection size computing for each case.

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 inputSample.

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.

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.

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.

1.1.3 Tools

DataHandling	Static methods for data handling.

DataHandling

class **DataHandling**

Static methods for data handling.

Methods

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

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

Sort defects and signals with respect to the censore thresholds.

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.

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()
```

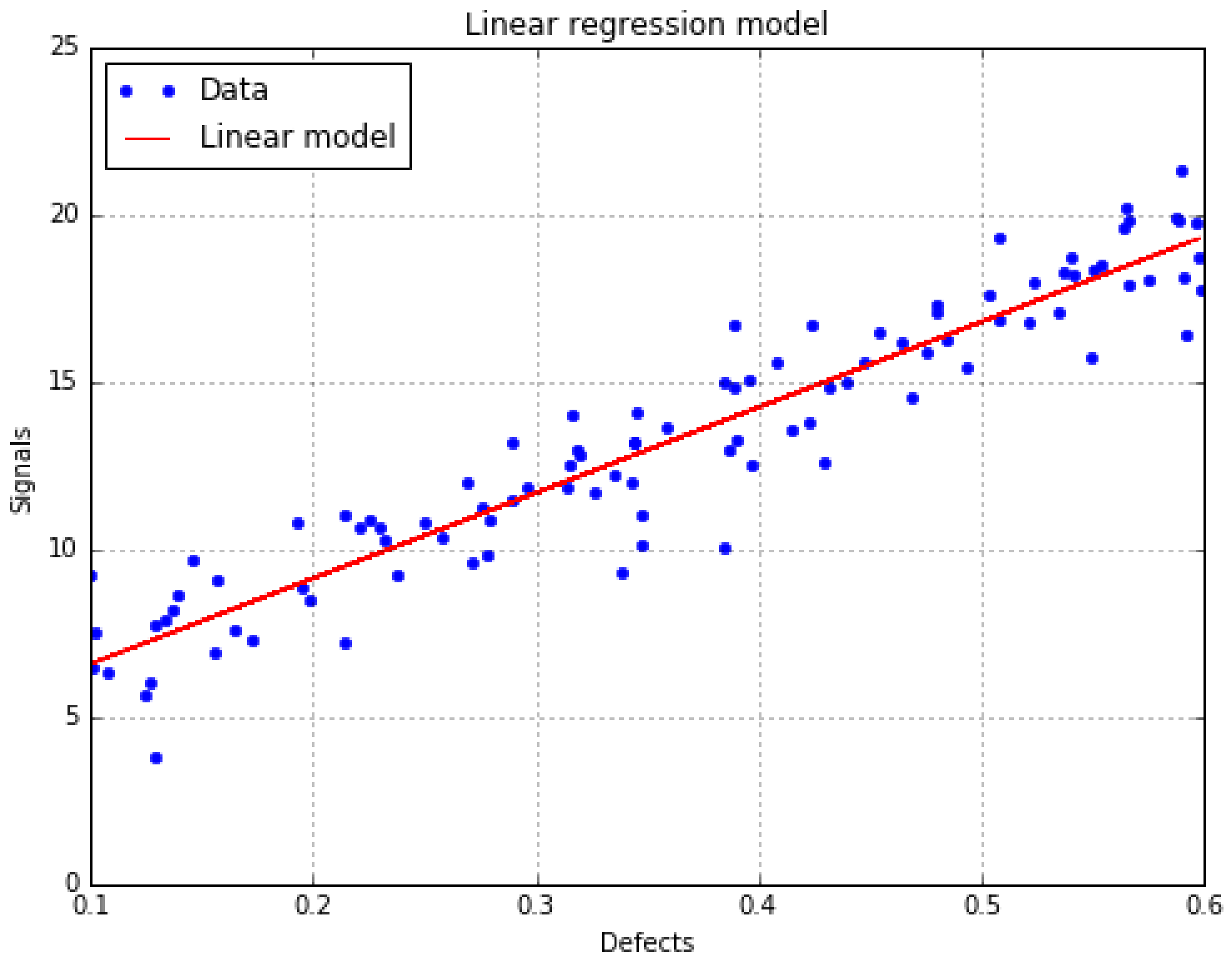
```
WARNING:root:Some hypothesis tests failed : please consider to use the Box Cox transformation.
```

```
--------------------------------------------------------------------------------
         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

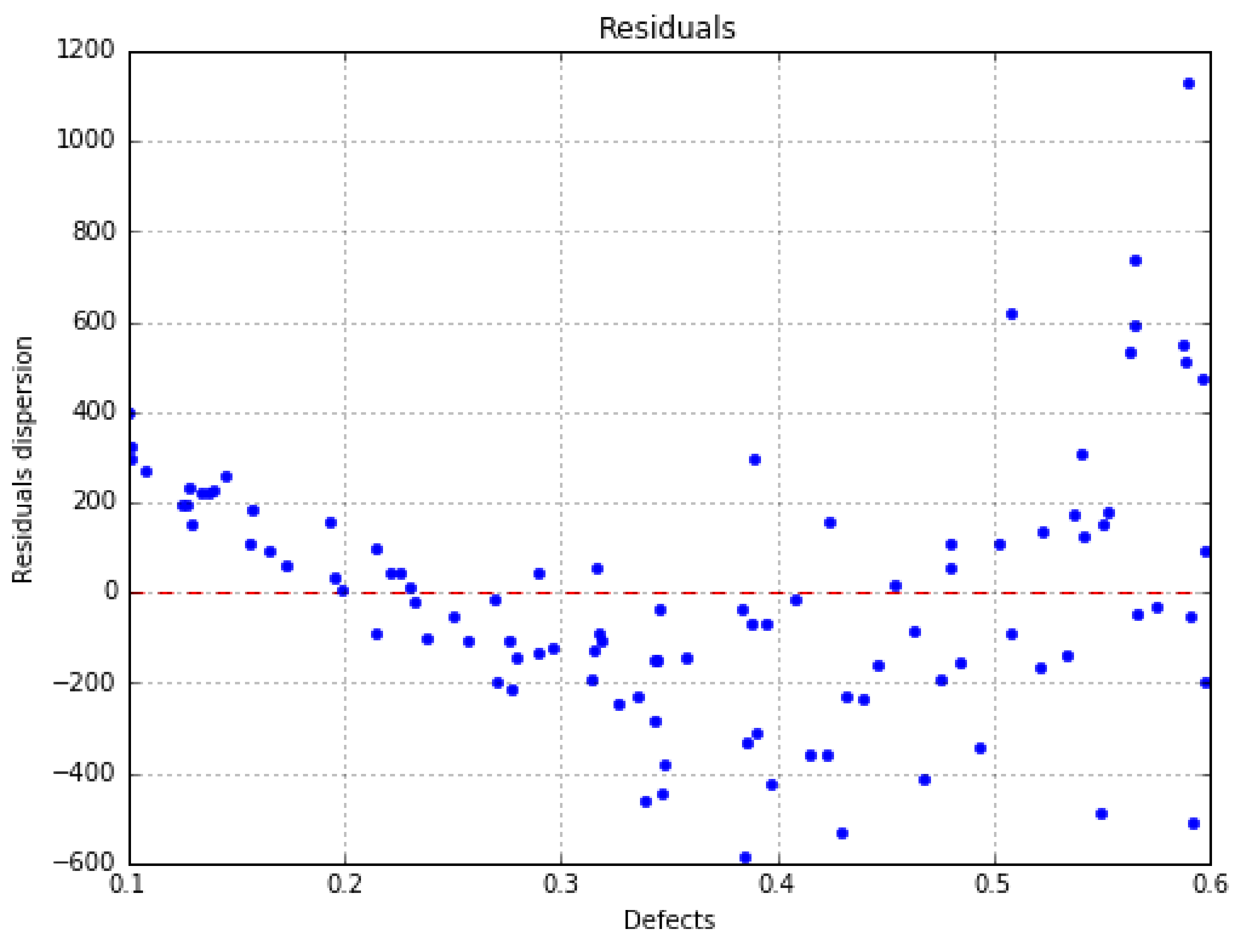
The linear model is not correct

```
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
--------------------------------------------------------------------------------
```

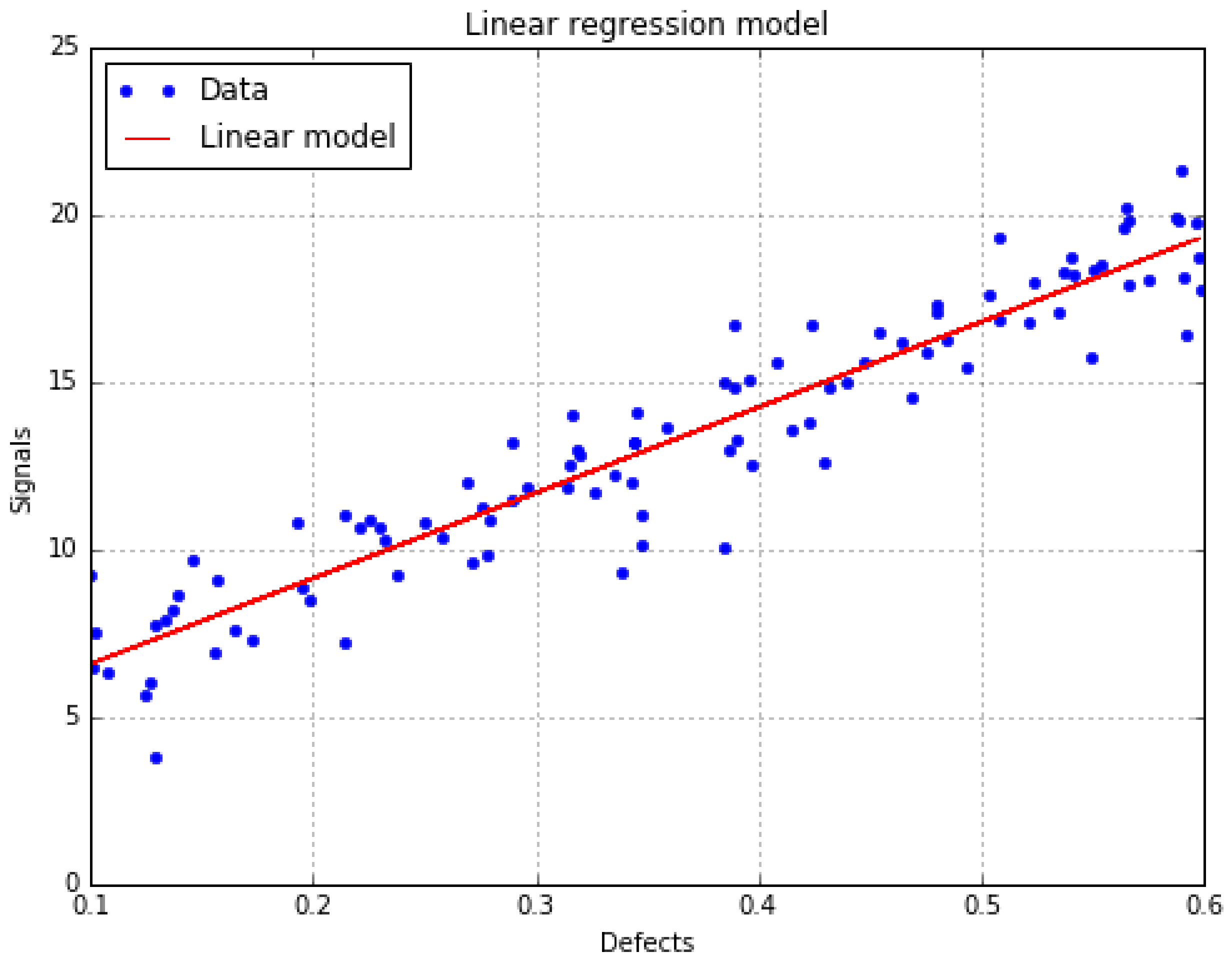
Save all results in a csv file

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

Show graphs

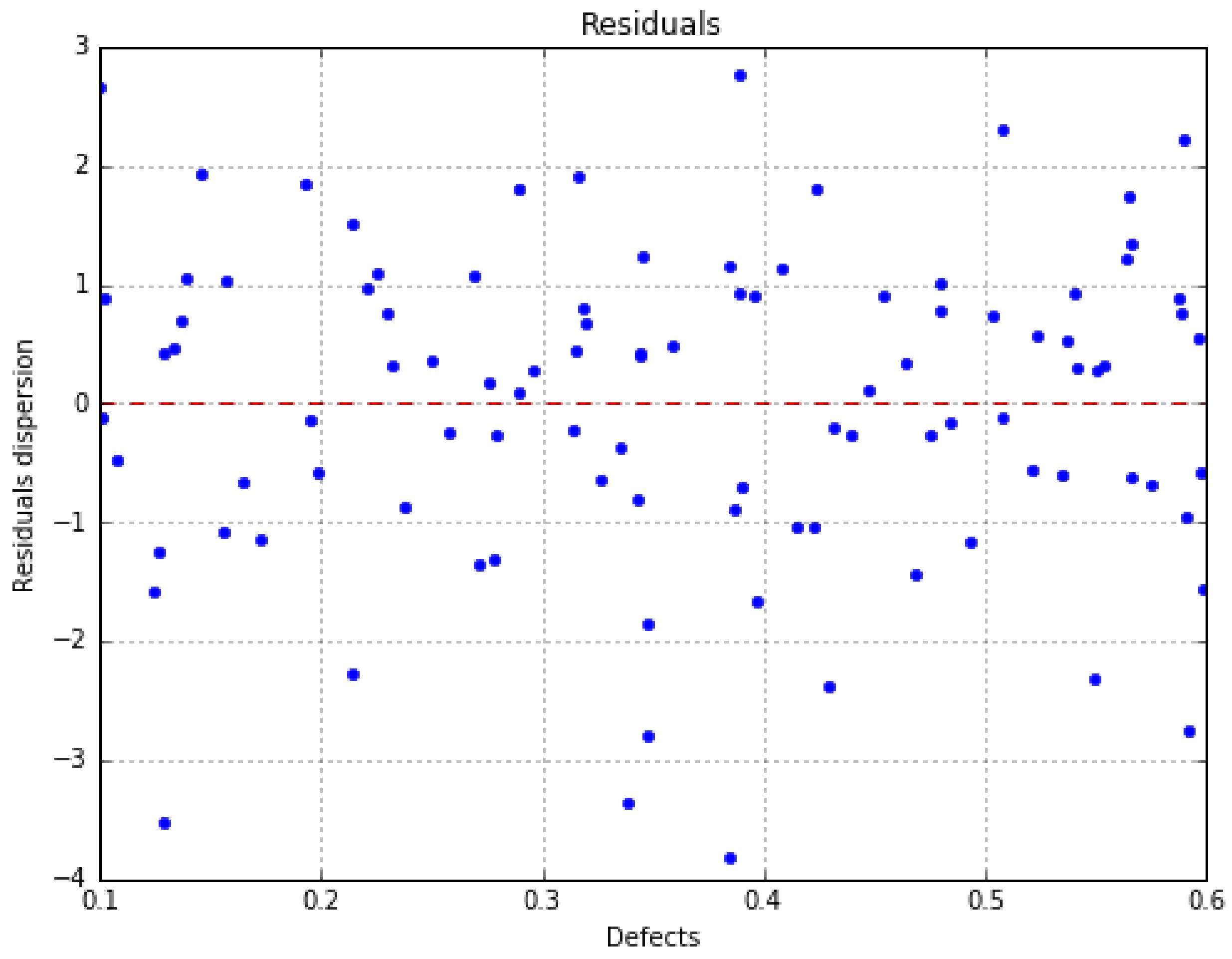
The linear regression model with data

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



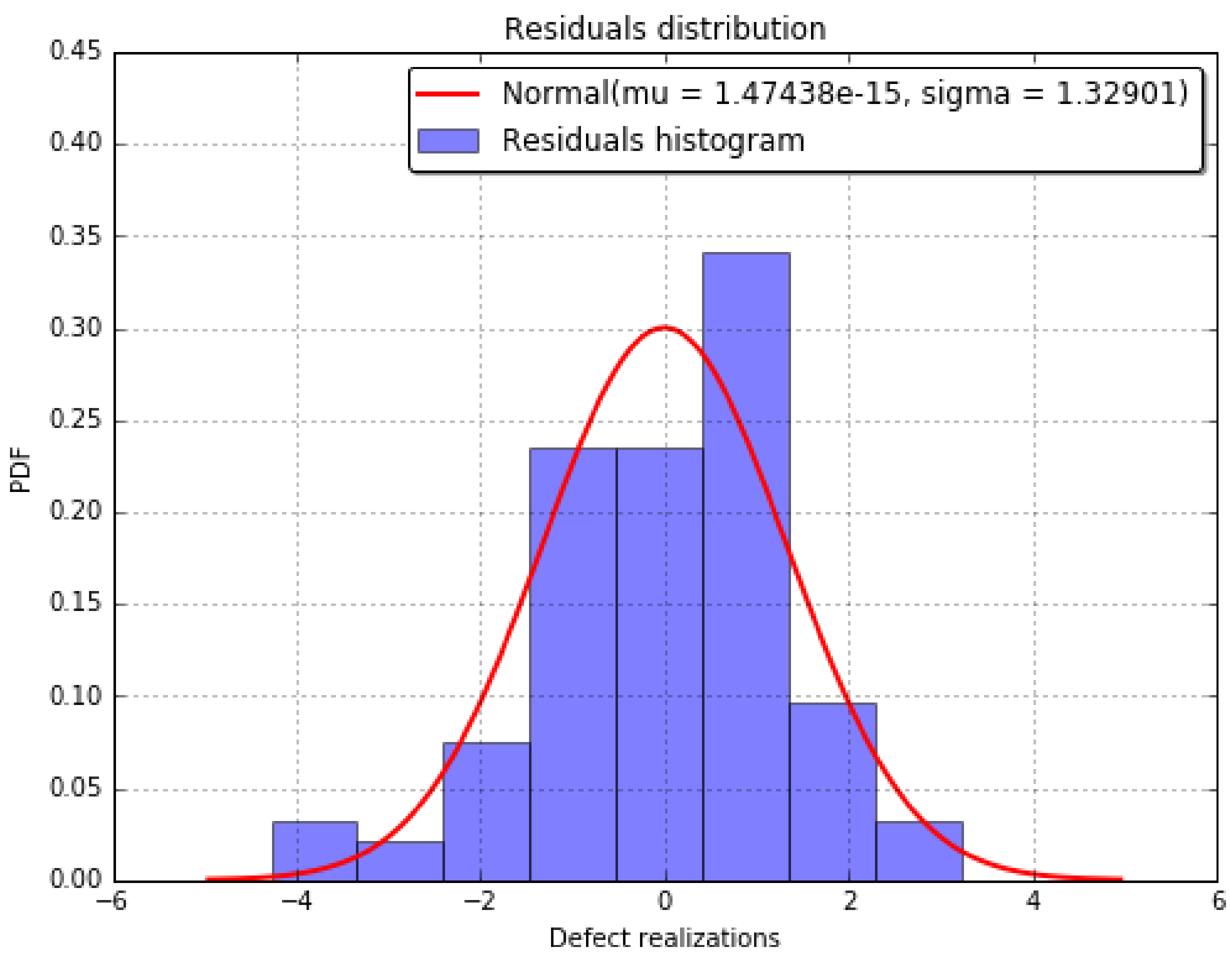
The residuals with respect to the defects

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



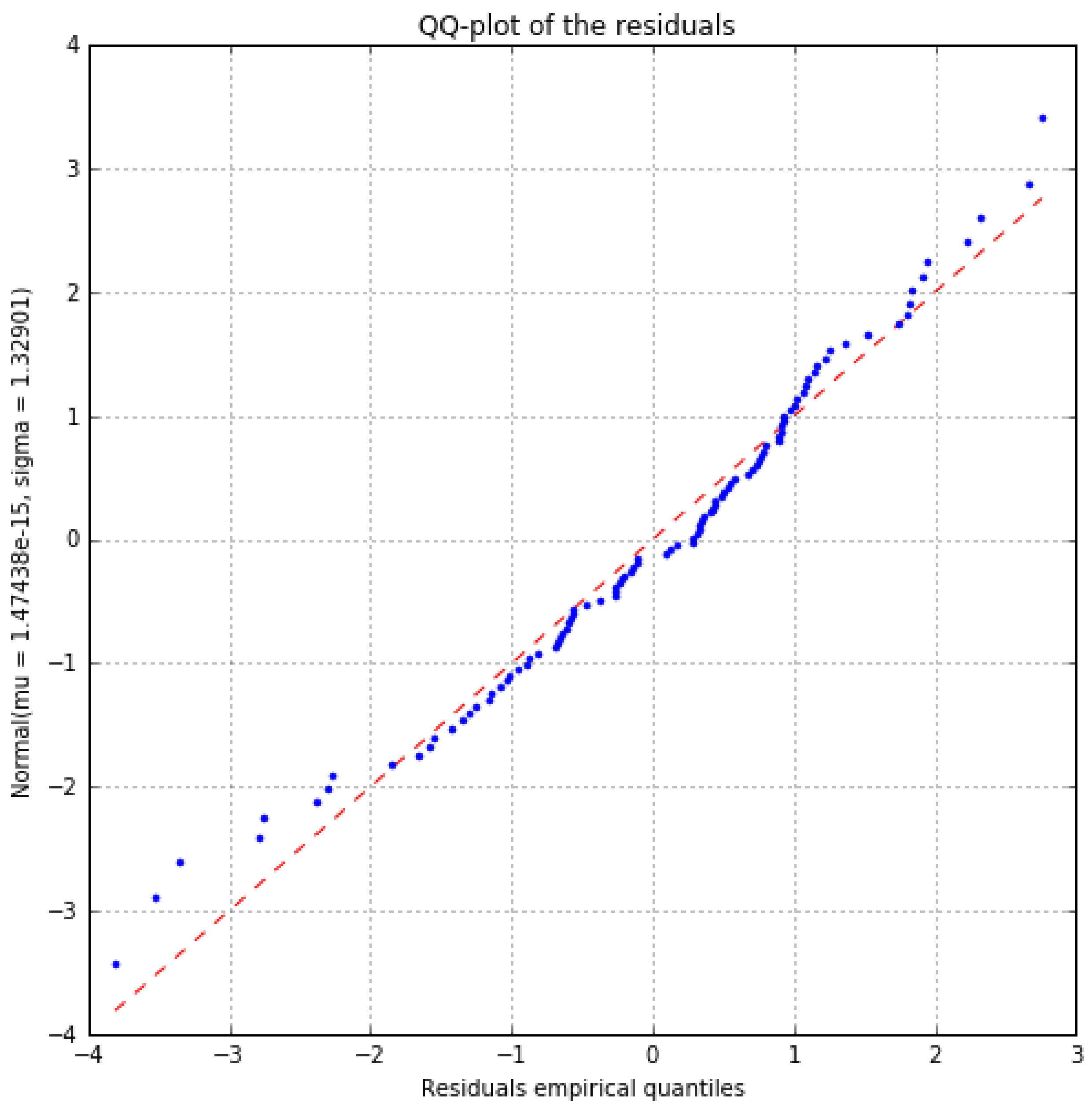
The fitted residuals distribution with the histogram

```python
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')
```



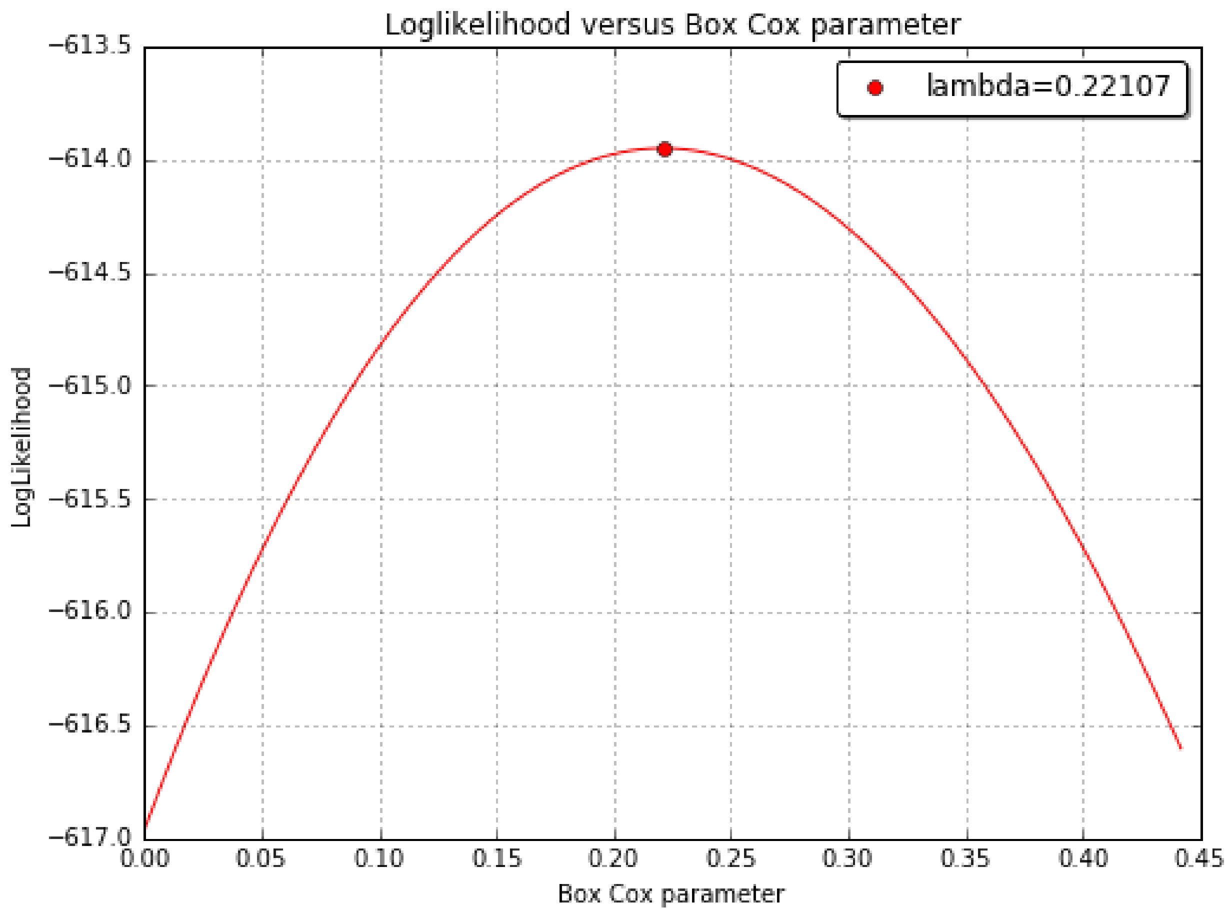
The residuals QQ plot

```
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
--------------------------------------------------------------------------------
```

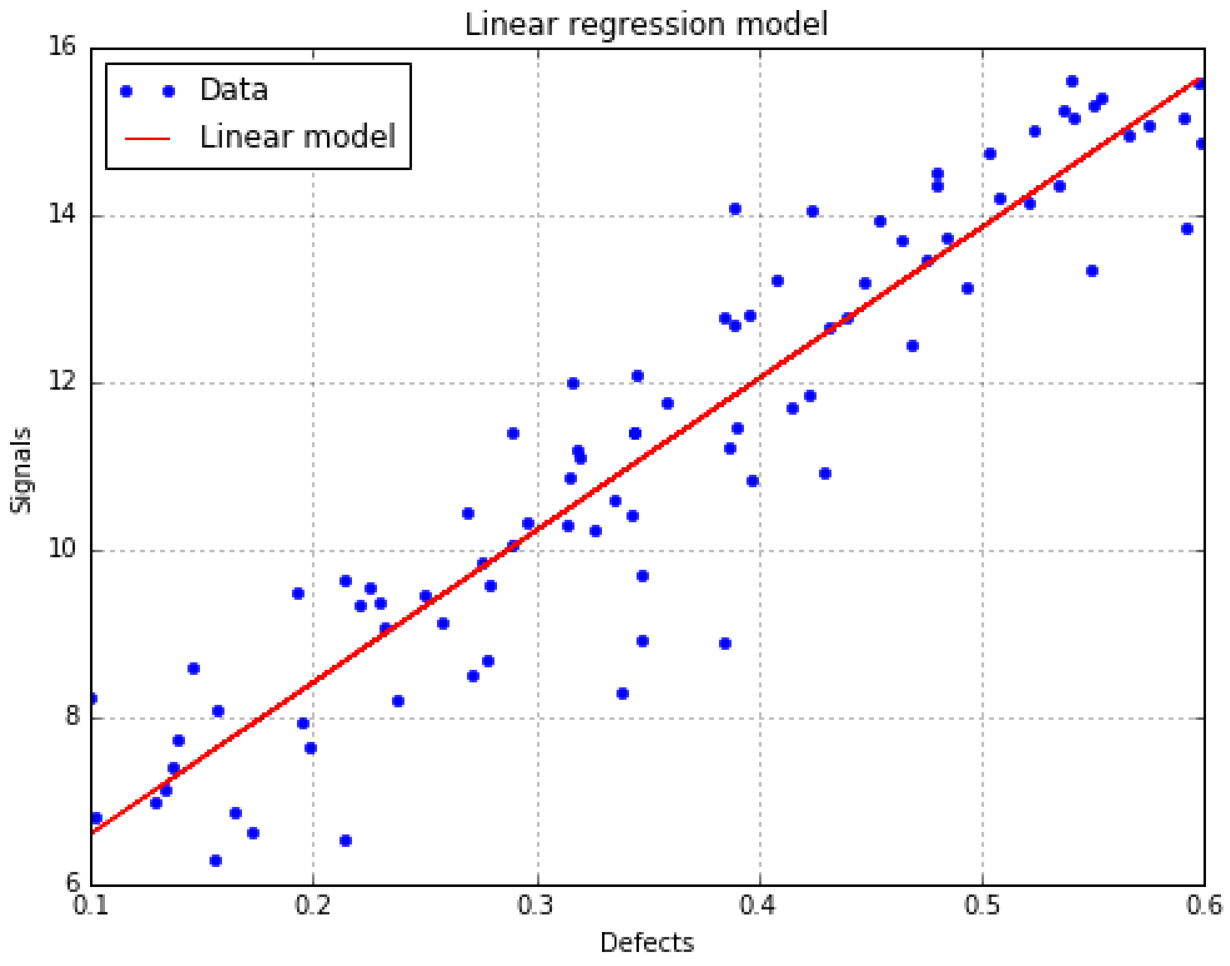
Save all results in a csv file

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

Show graphs

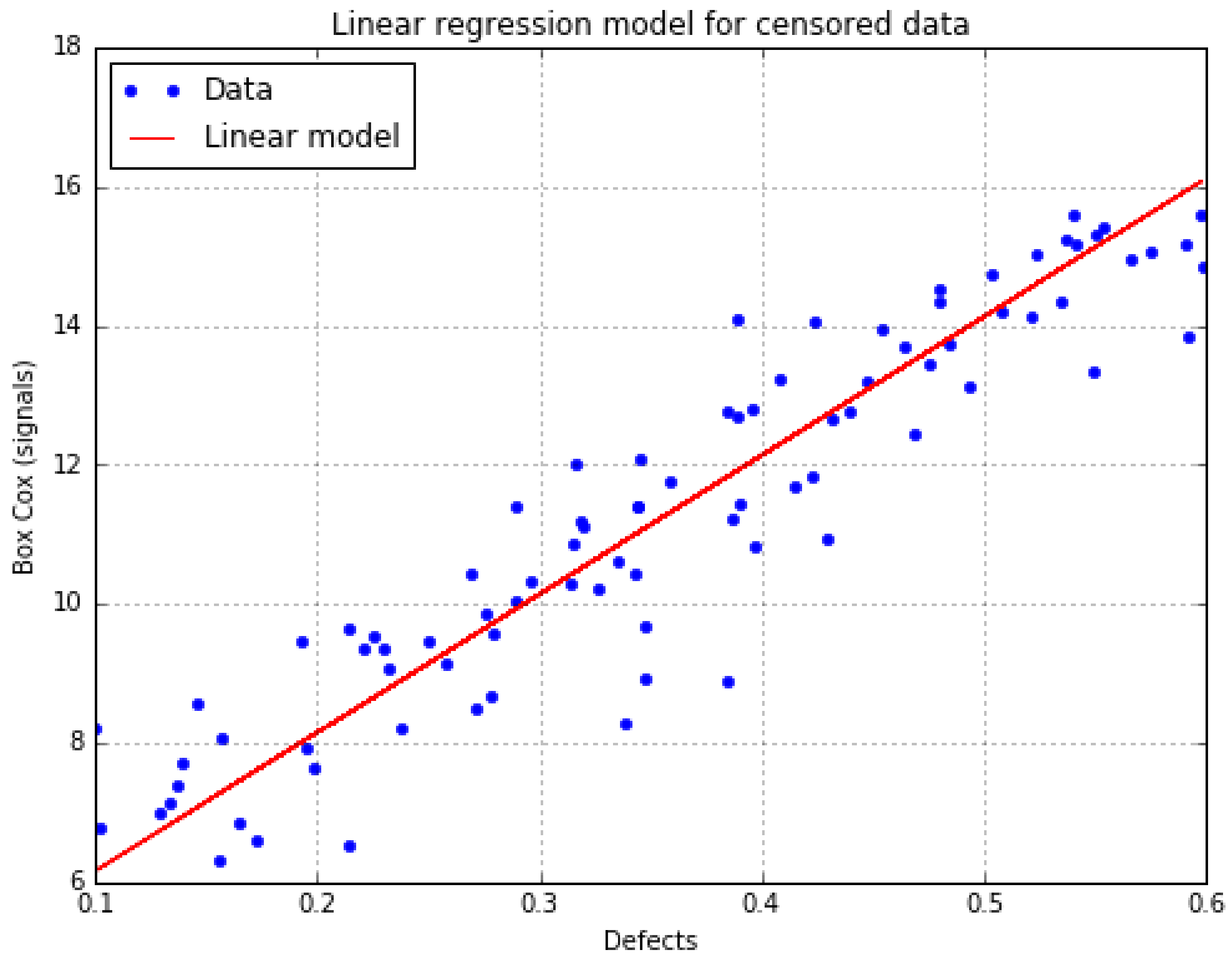
The linear regression model with data for the uncensored case (default case)

```
# 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

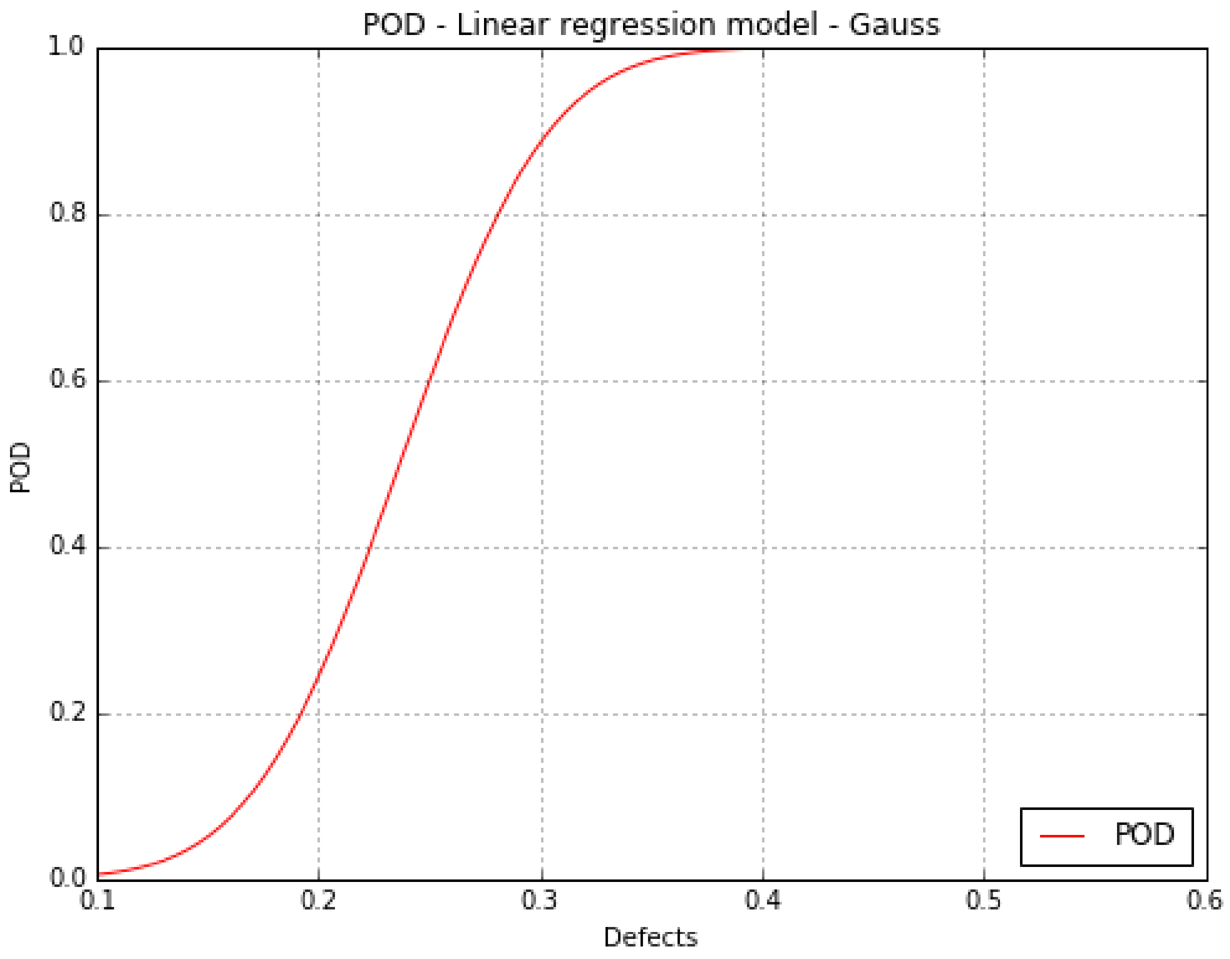
```
# get the POD model
PODmodel = PODGauss.getPODModel()
# get the POD model at the given confidence level
PODmodelCl95 = PODGauss.getPODCLModel(0.95)

# 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

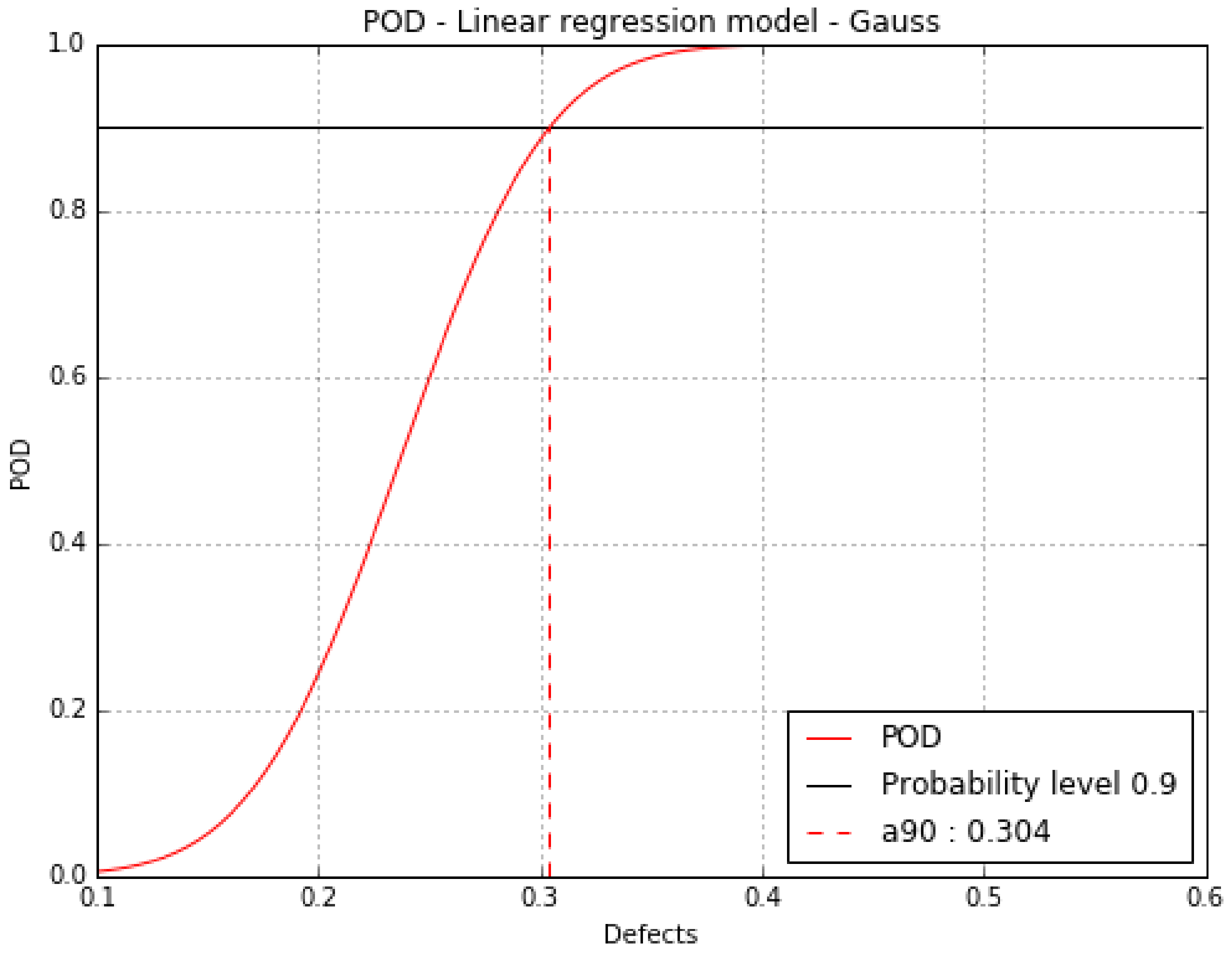
Only the mean POD

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



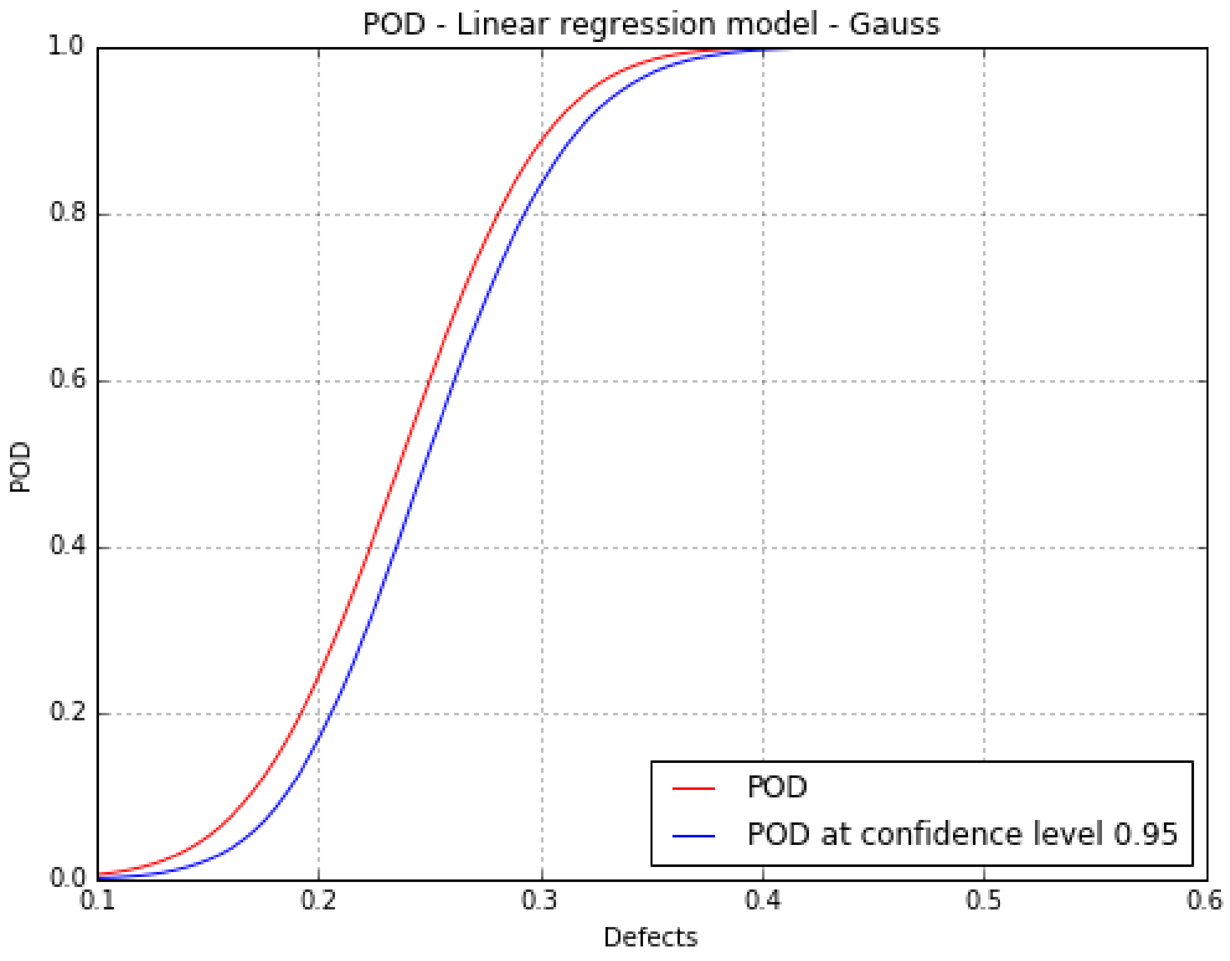
Mean POD with the detection size for a given probability level

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



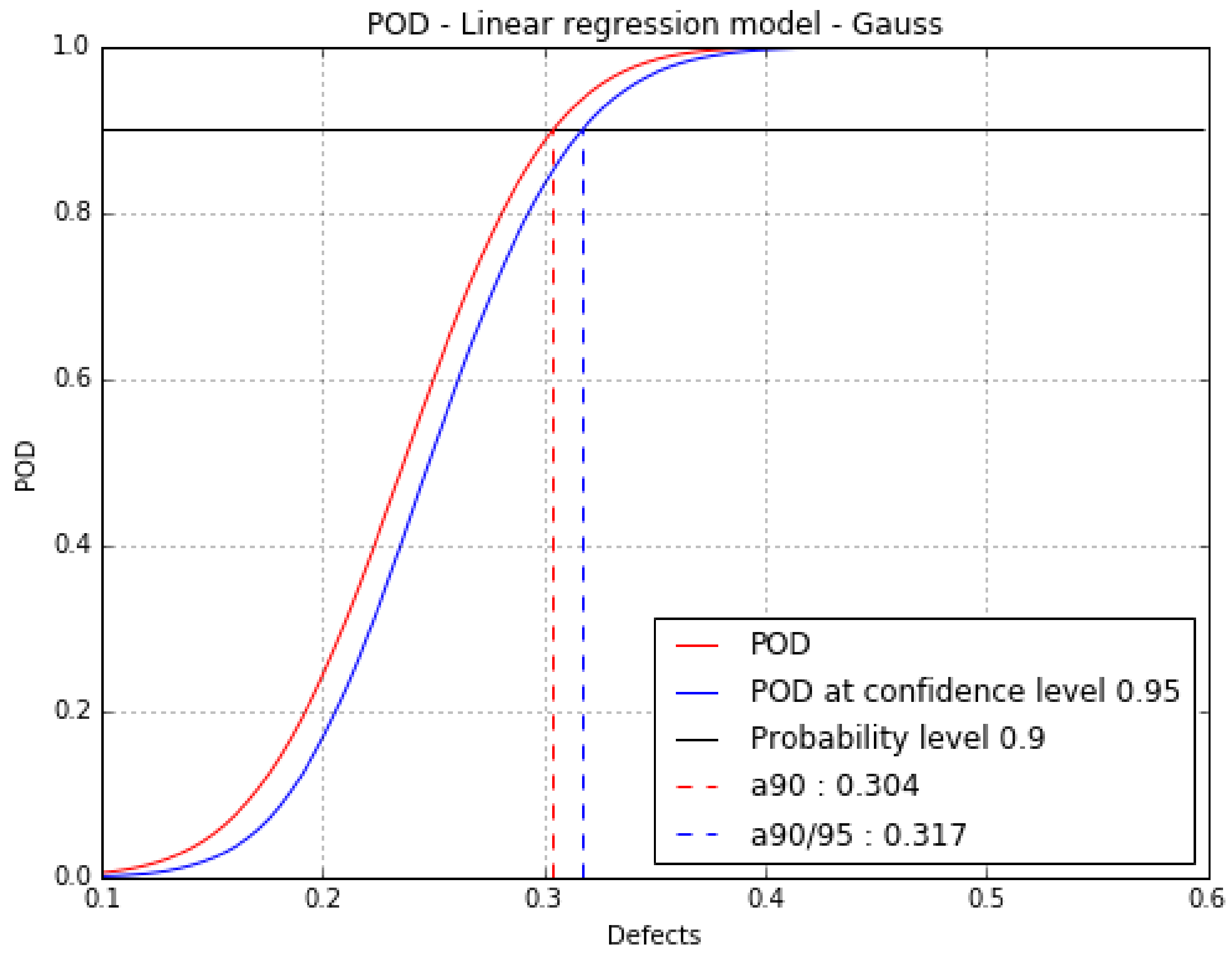
Mean POD with POD at confidence level

```
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

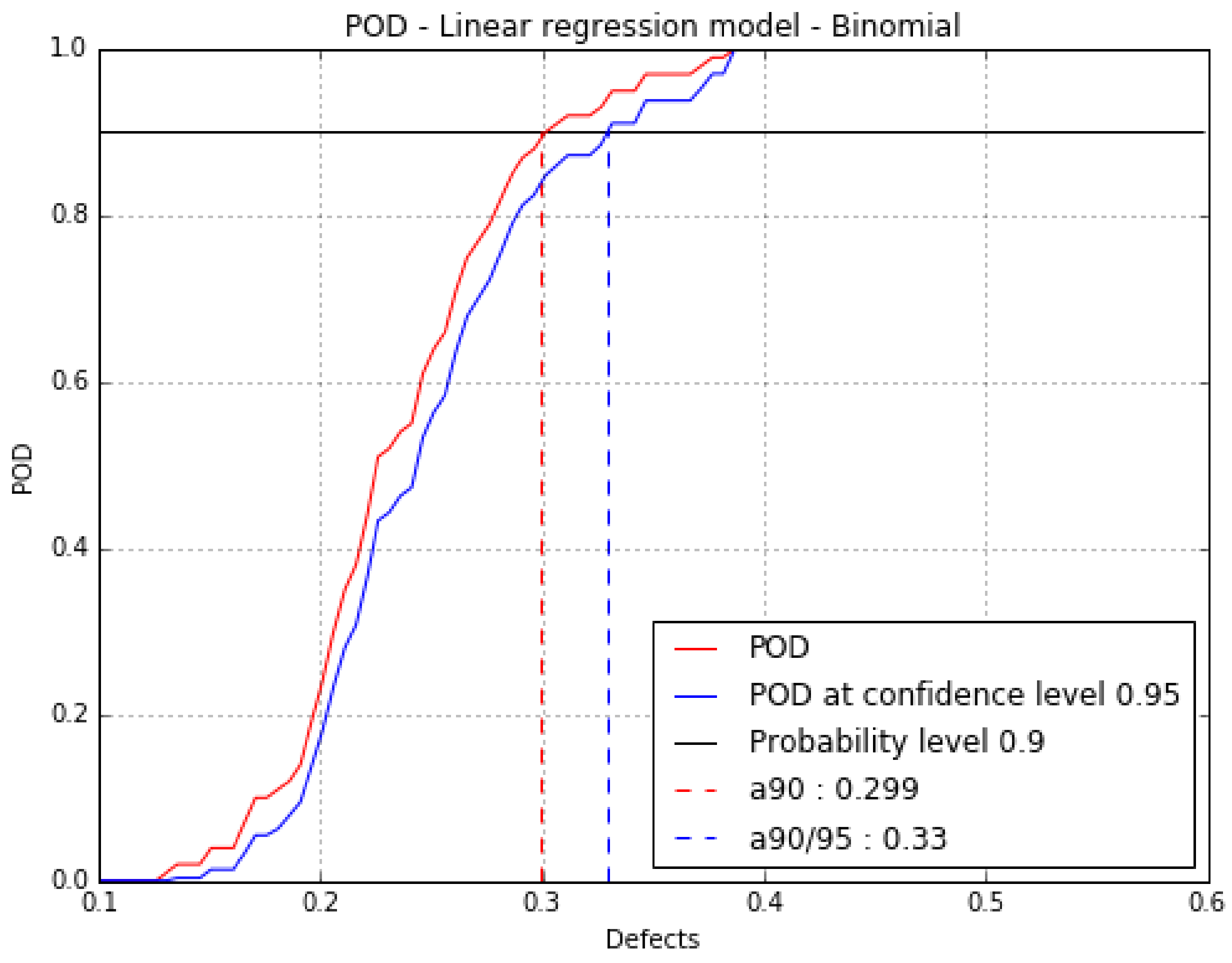
This corresponds with the Berens Binomial method.

```
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

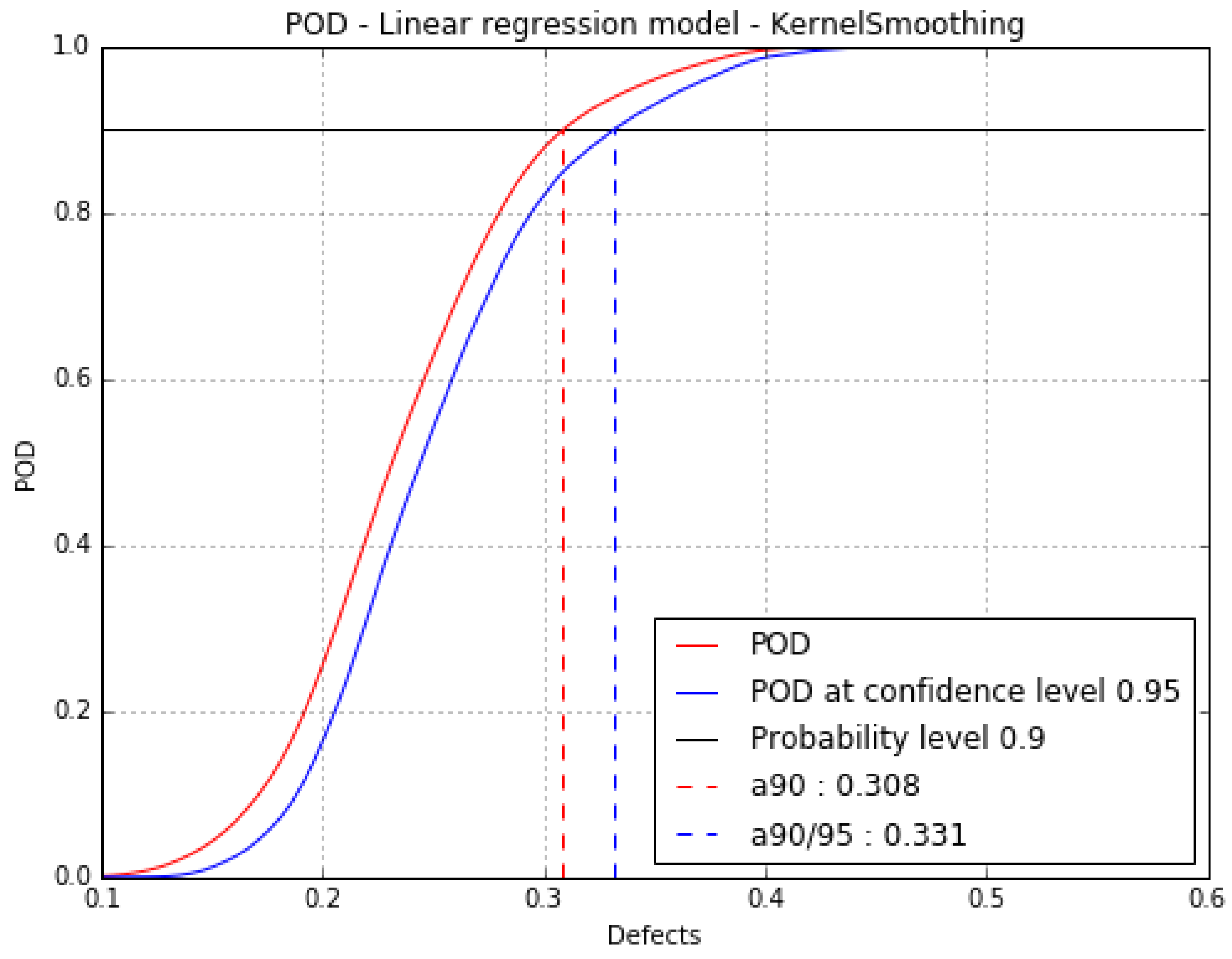
The POD at the given confidence level is built using bootstrap. It may take few seconds.

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

```
# 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

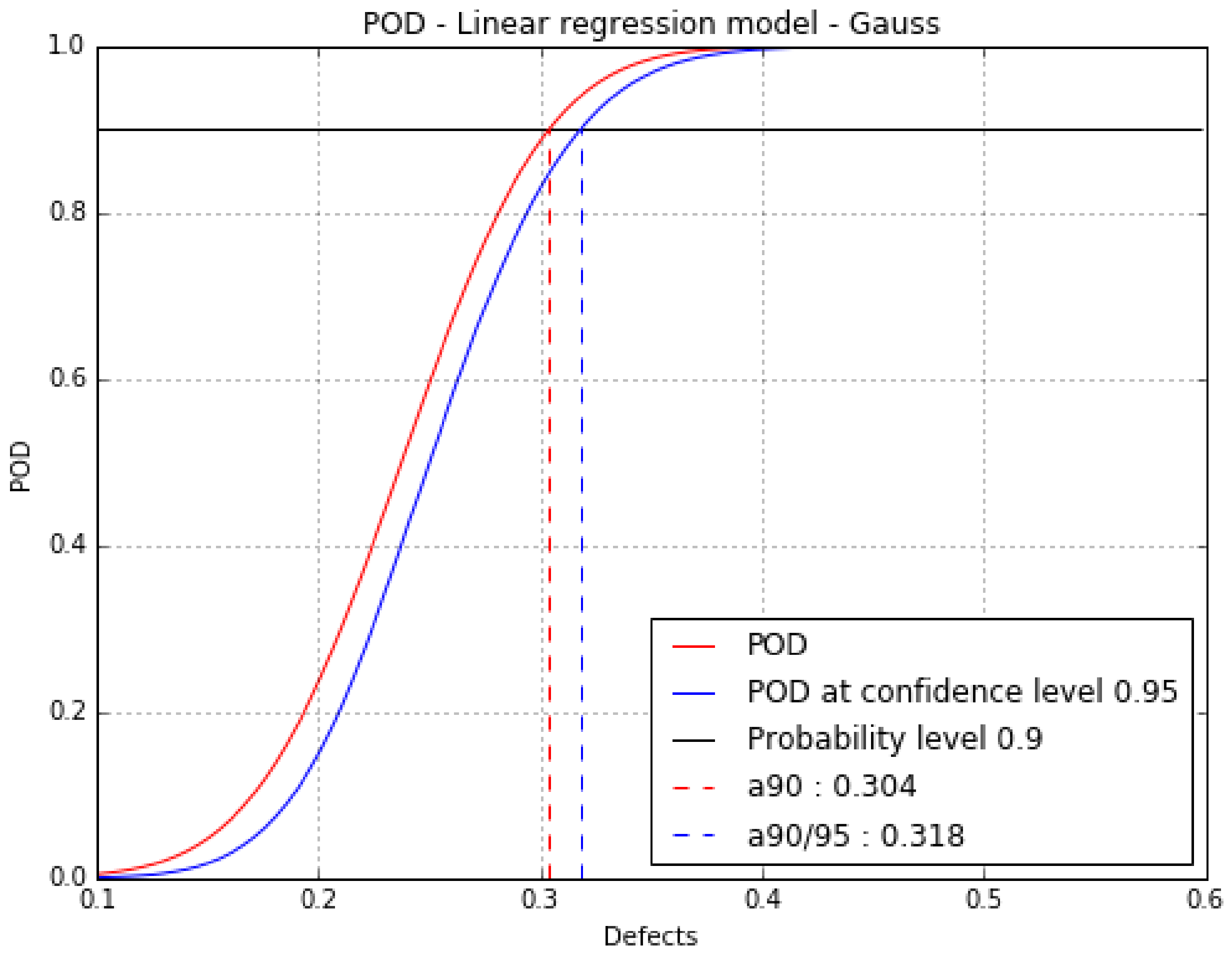
```
# get the POD model
PODmodel = PODGauss.getPODModel()
# get the POD model at the given confidence level
PODmodelCl95 = PODGauss.getPODCLModel(0.95)

# 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.
```

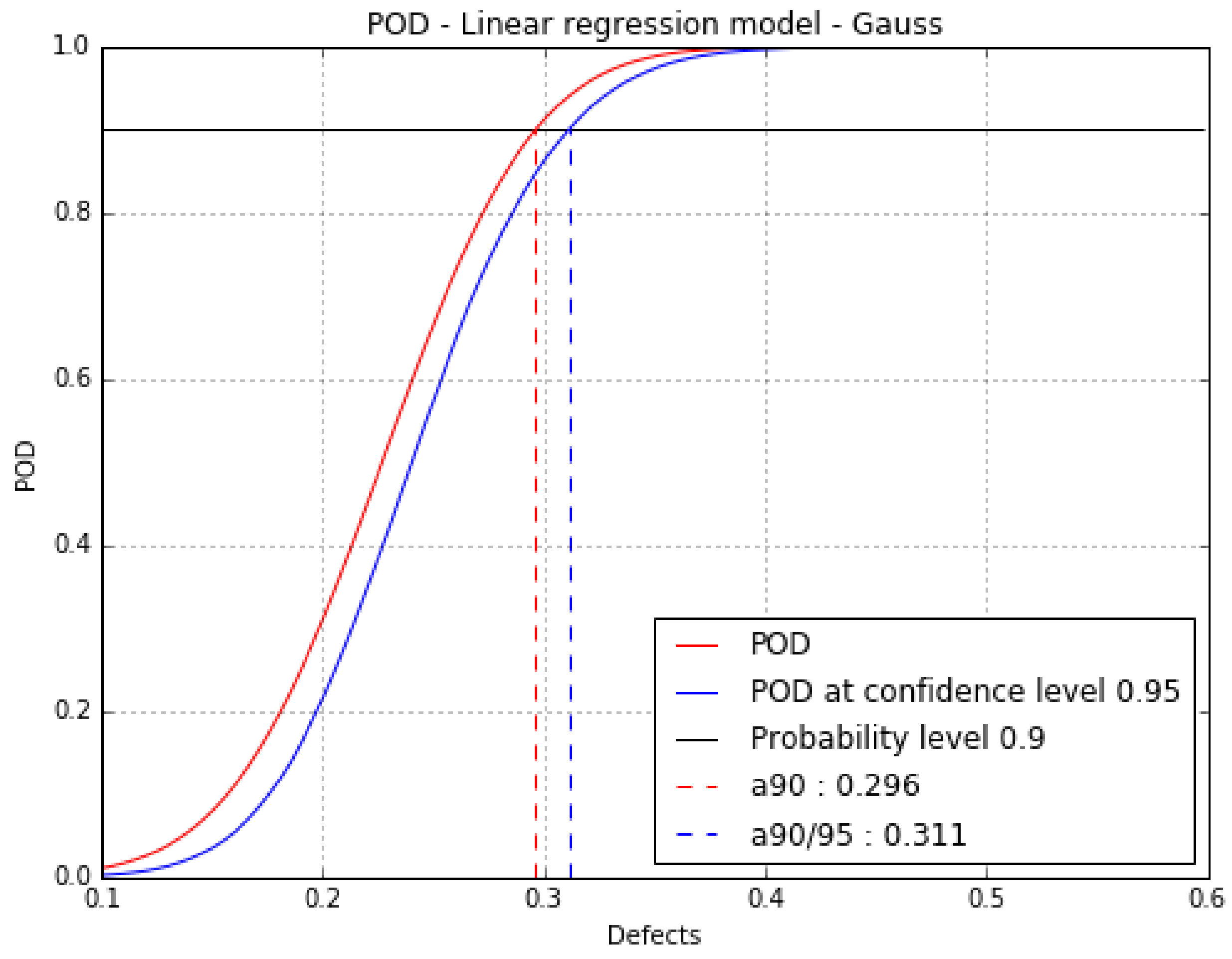
```
result = otpod.DataHandling.filterCensoredData(defects, signals,
                                               noiseThres, saturationThres)
defectsFiltered = result[0]
signalsFiltered = result[3]
```

```
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
```

```
# Due to the bootstrap technique used to compute the confidence
# interval, the run take few minutes.
t0 = time()
POD = otpod.QuantileRegressionPOD(defects, signals, detection,
                                  boxCox=True)
POD.run()
print 'Computing time : {:0.2f} s'.format(time()-t0)
```

```
Computing time : 151.43 s
```

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

```
t0 = time()
PODsimulSize100 = otpod.QuantileRegressionPOD(defects, signals, detection,
                                  boxCox=True)
PODsimulSize100.setSimulationSize(100)
PODsimulSize100.run()
print 'Computing time : {:0.2f} s'.format(time()-t0)
```

```
Computing time : 16.23 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

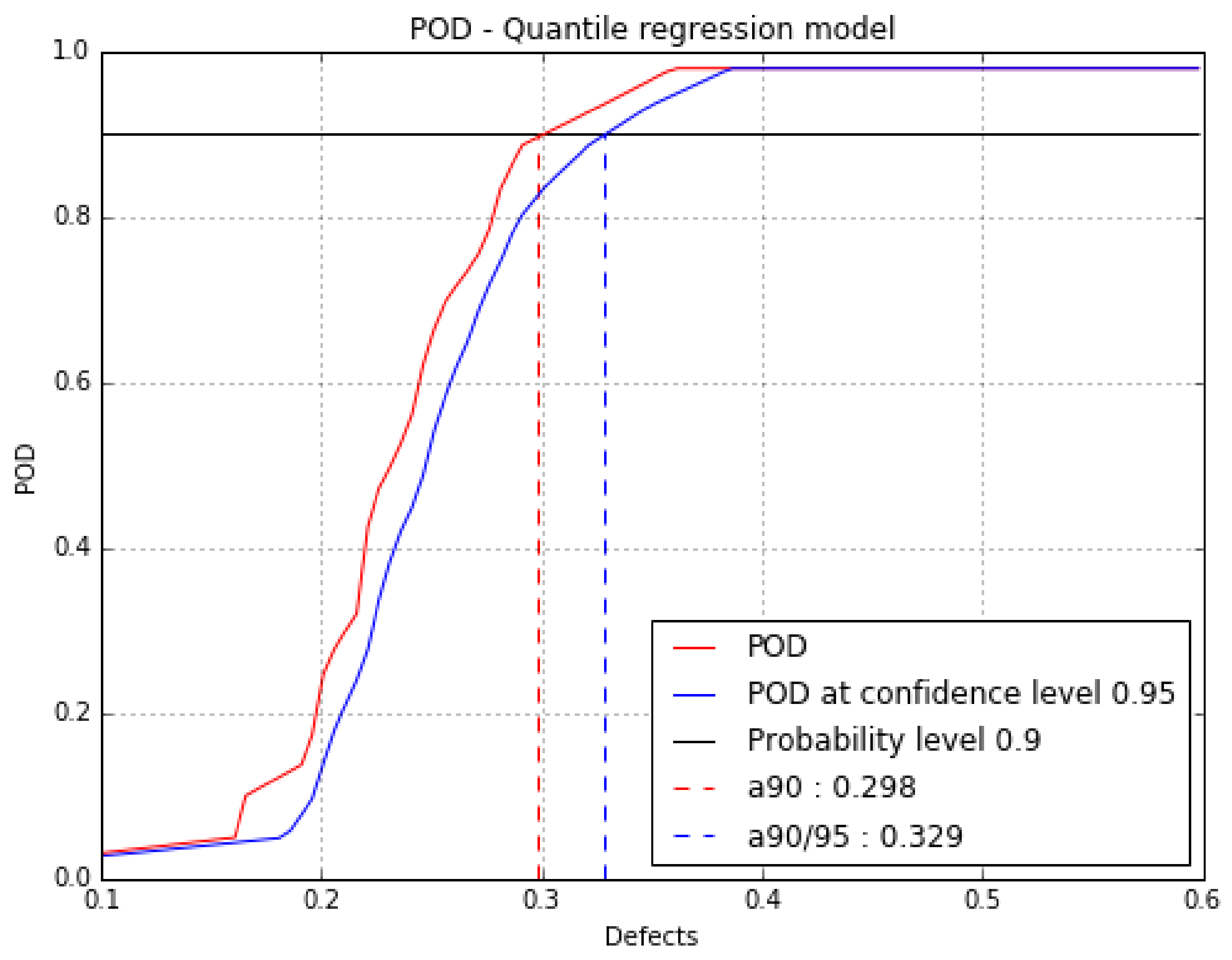
```
print 'Pseudo R2 for quantile 0.9 : {:0.3f}'.format(POD.getR2(0.9))
print 'Pseudo R2 for quantile 0.95 : {:0.3f}'.format(POD.getR2(0.95))
```

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

Show POD graphs

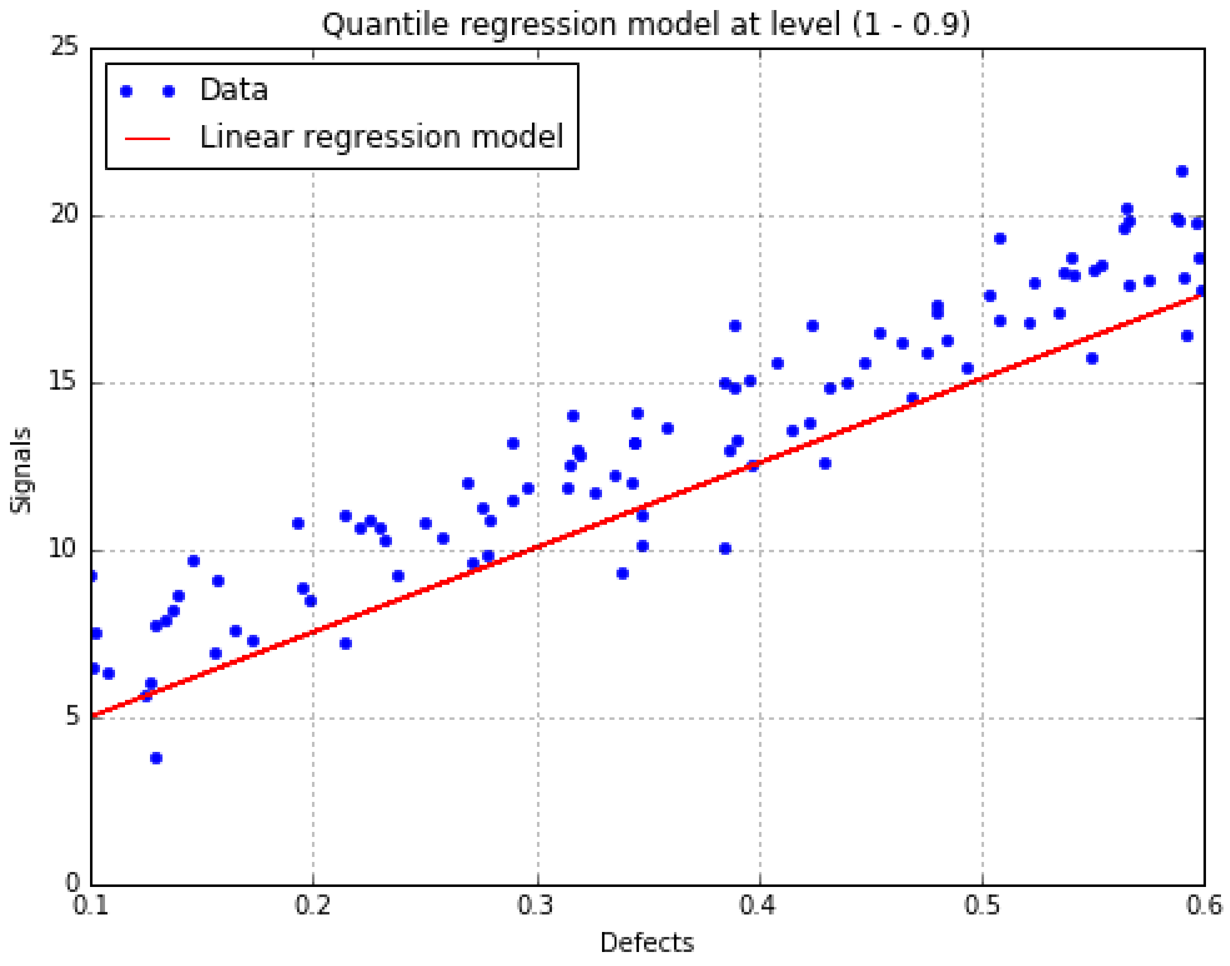
Mean POD and POD at confidence level with the detection size for a given probability level

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



Show the linear regression model at the given quantile

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


Quantile regression model at level (1 - 0.9)
Data
Linear regression model
Signals
Defects
0
5
10
15
20
25
0.1
0.2
0.3
0.4
0.5
0.6

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search