otpod Documentation

Release

Antoine Dumas

Oct 05, 2016

CONTENTS

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`

1.1.2 POD computation methods

`UnivariateLinearModelPOD`
`QuantileRegressionPOD`
`PolynomialChaosPOD`
`KrigingPOD`

1.1.3 Adaptive algorithms

`AdaptiveSignalPOD`
`AdaptiveHitMissPOD`

1.1.4 Tools

`PODSummary`
`DataHandling`

1.2 Examples of the API

ipynb source code

1.2.1 Linear model analysis

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

Generate data

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

Run analysis without Box Cox

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

Get some particular results

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

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

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

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

```
analysis.printResults()
```

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

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

                                                     Uncensored

Intercept coefficient :                                 -604.76
Slope coefficient :                                     3606.04
```

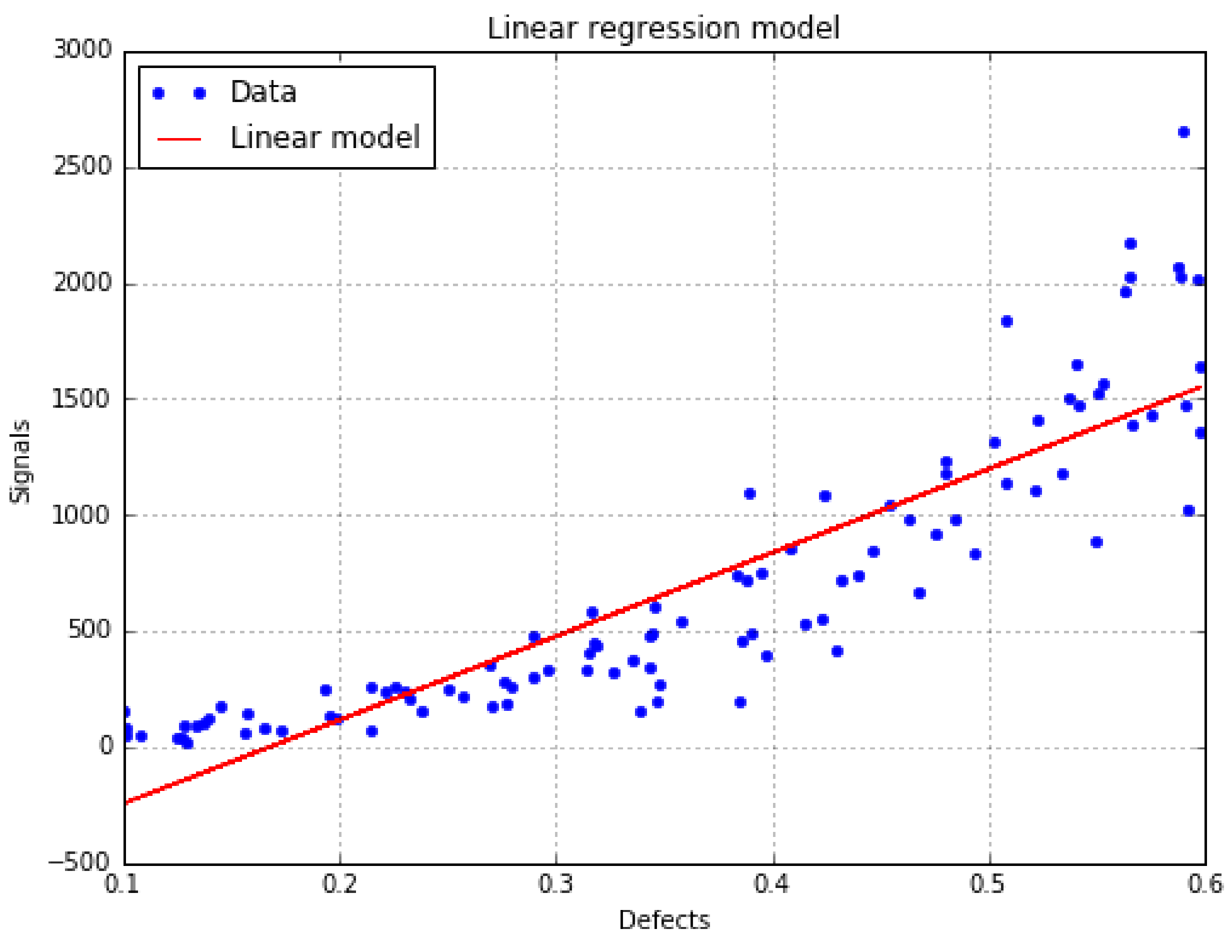
```
Standard error of the estimate :                        291.47

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

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


--------------------------------------------------------------------------------
         Residuals analysis results
--------------------------------------------------------------------------------
Fitted distribution (uncensored) :              Normal(mu = 5.95719e-13, sigma = 289.998)

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

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

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

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

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

Show graphs

The linear model is not correct

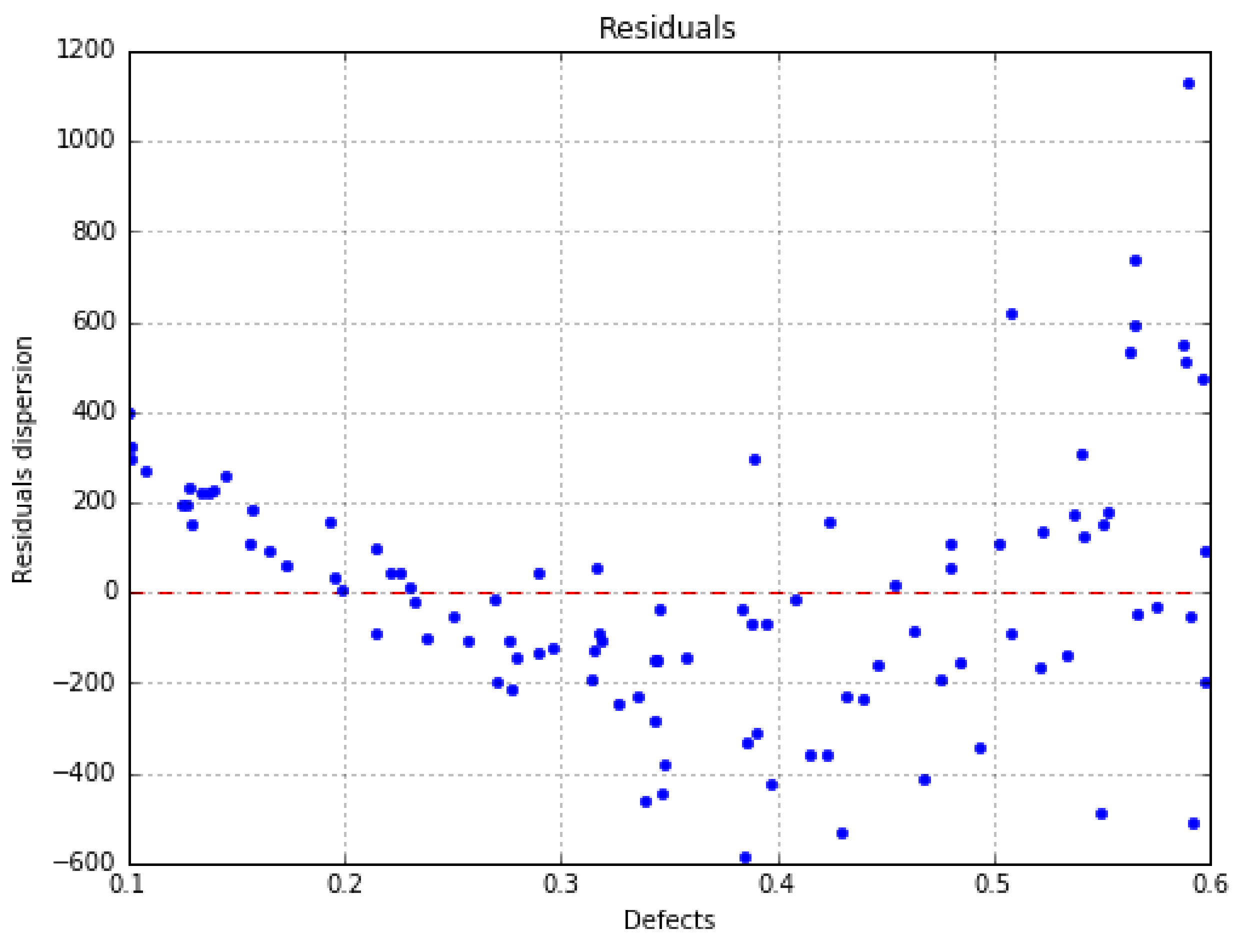
```
fig, ax = analysis.drawLinearModel()
fig.show()
```

```
/home/dumas/anaconda2/lib/python2.7/site-packages/matplotlib/figure.py:397: UserWarning: matplotlib i
  "matplotlib is currently using a non-GUI backend, "
```



The residuals are not homoskedastic

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



Run analysis with Box Cox

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

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

```
analysis.printResults()
```

```
--------------------------------------------------------------------------------
         Linear model analysis results
--------------------------------------------------------------------------------
Box Cox parameter :                                                   0.22

                                                                Uncensored

Intercept coefficient :                                               4.02
Slope coefficient :                                                  25.55
Standard error of the estimate :                                      1.34

Confidence interval on coefficients
Intercept coefficient :                                       [3.33, 4.72]
Slope coefficient :                                         [23.80, 27.31]
```

```
Level :                                                         0.95

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

--------------------------------------------------------------------------------
         Residuals analysis results
--------------------------------------------------------------------------------
Fitted distribution (uncensored) :              Normal(mu = 1.45661e-15, sigma = 1.32901)

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

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

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

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

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

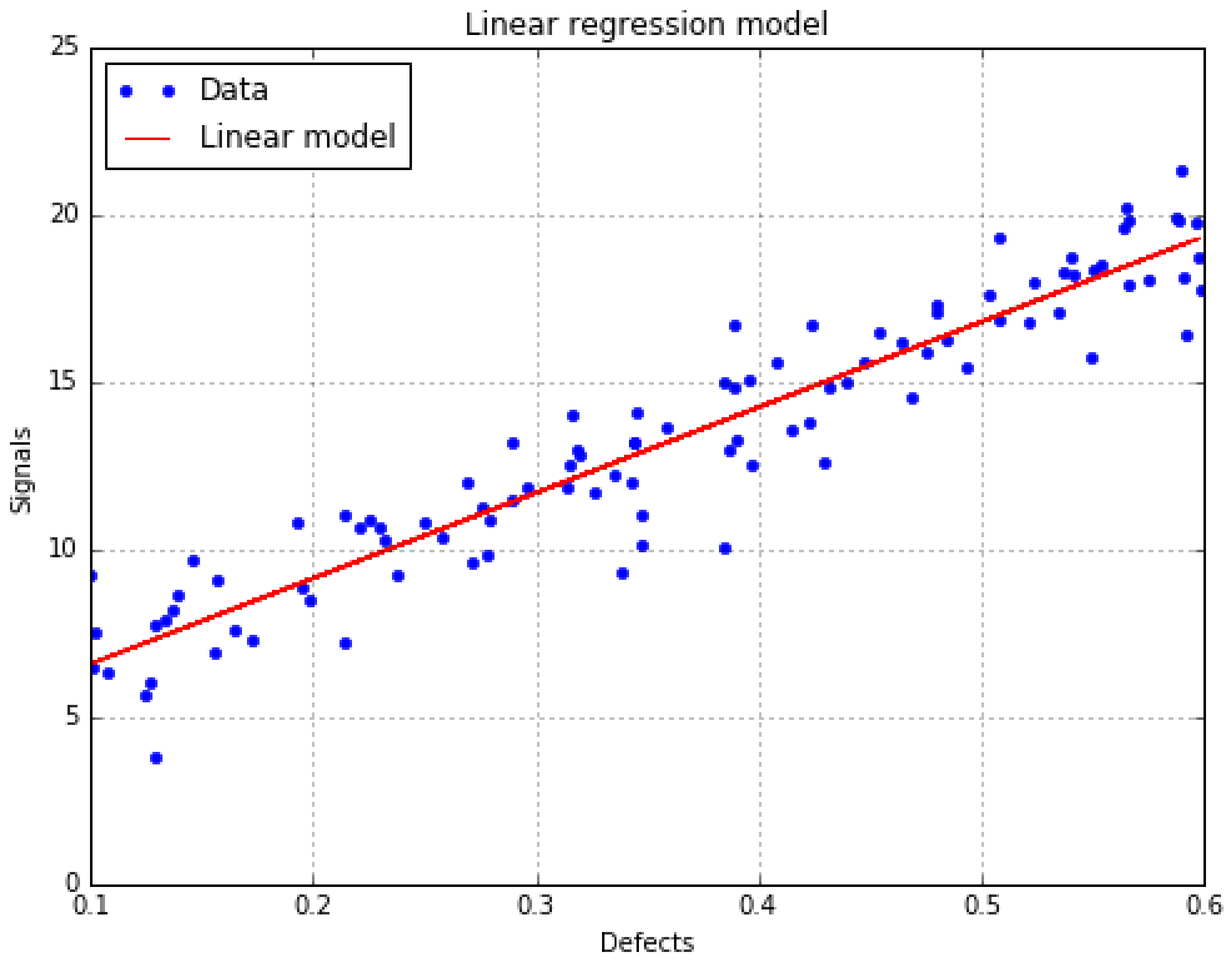
Save all results in a csv file

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

Show graphs

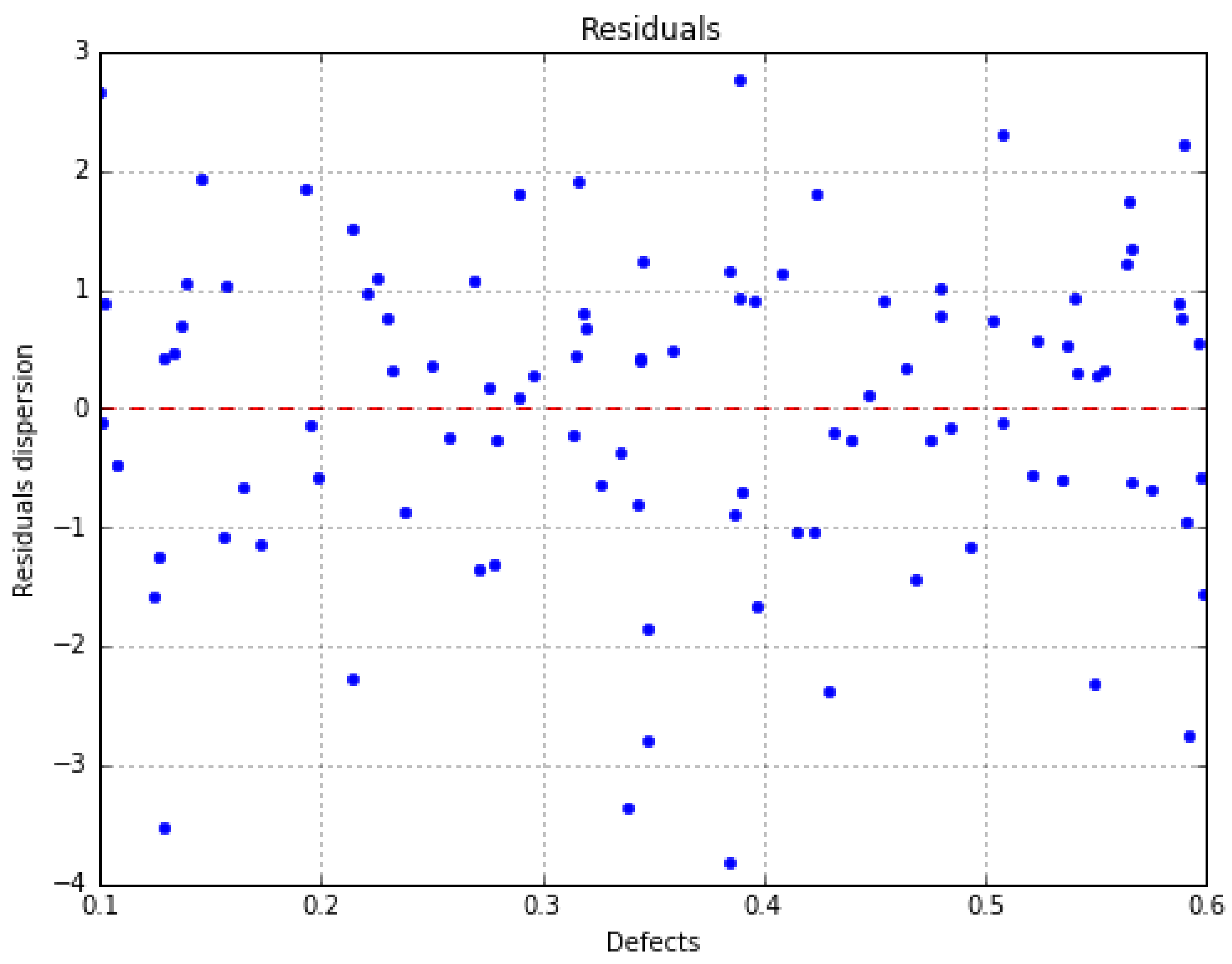
The linear regression model with data

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



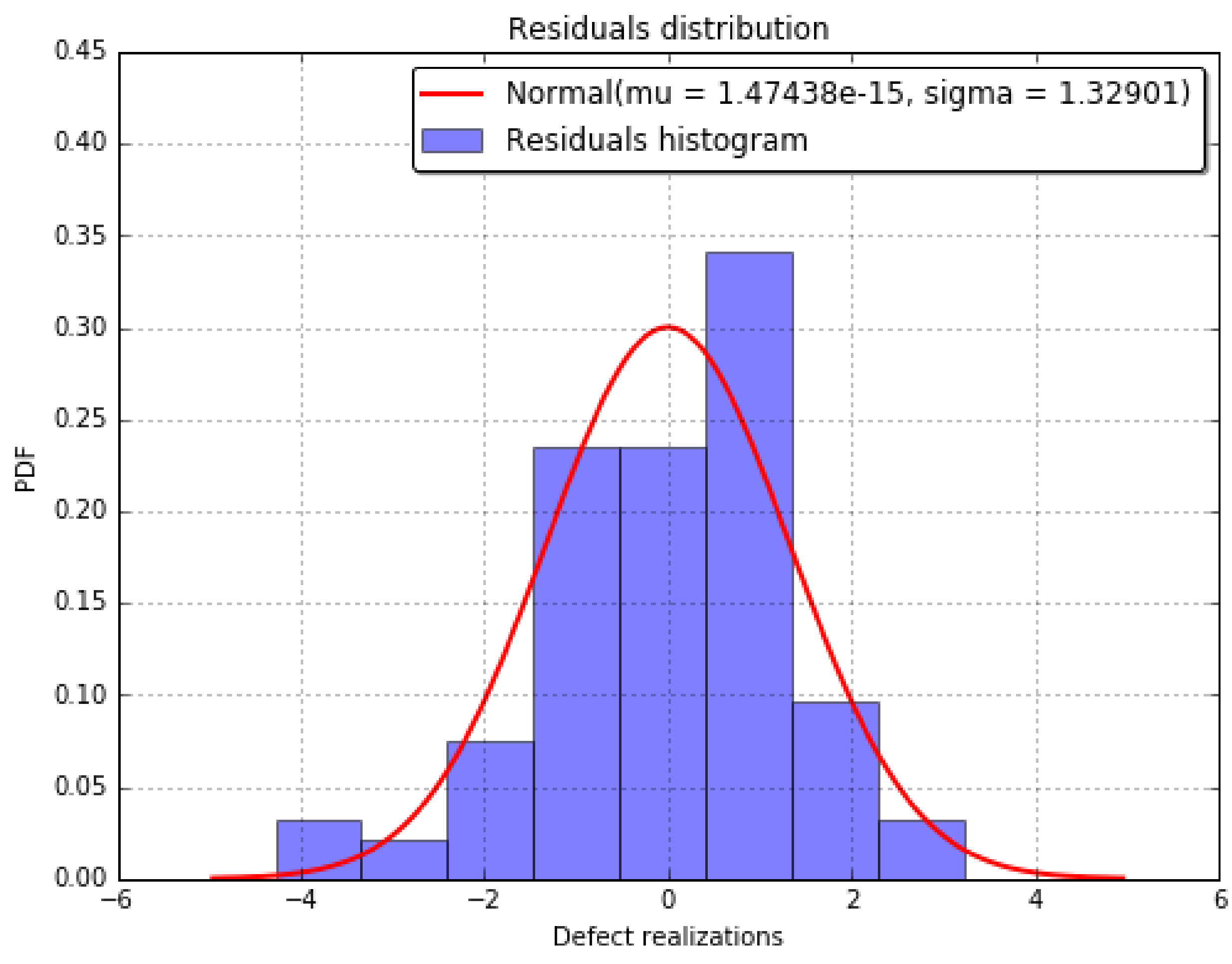
The residuals with respect to the defects

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



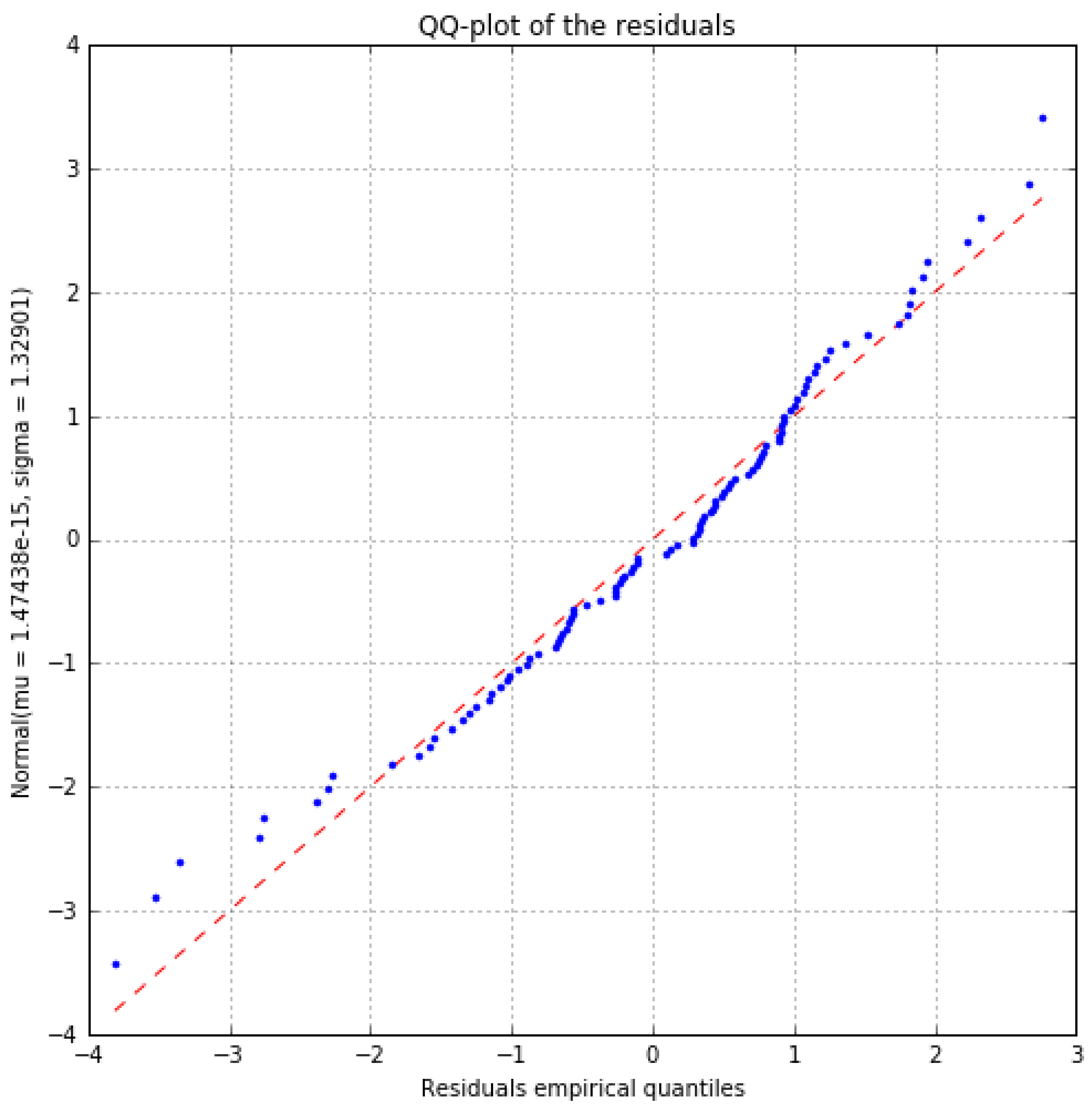
The fitted residuals distribution with the histogram

```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')
```



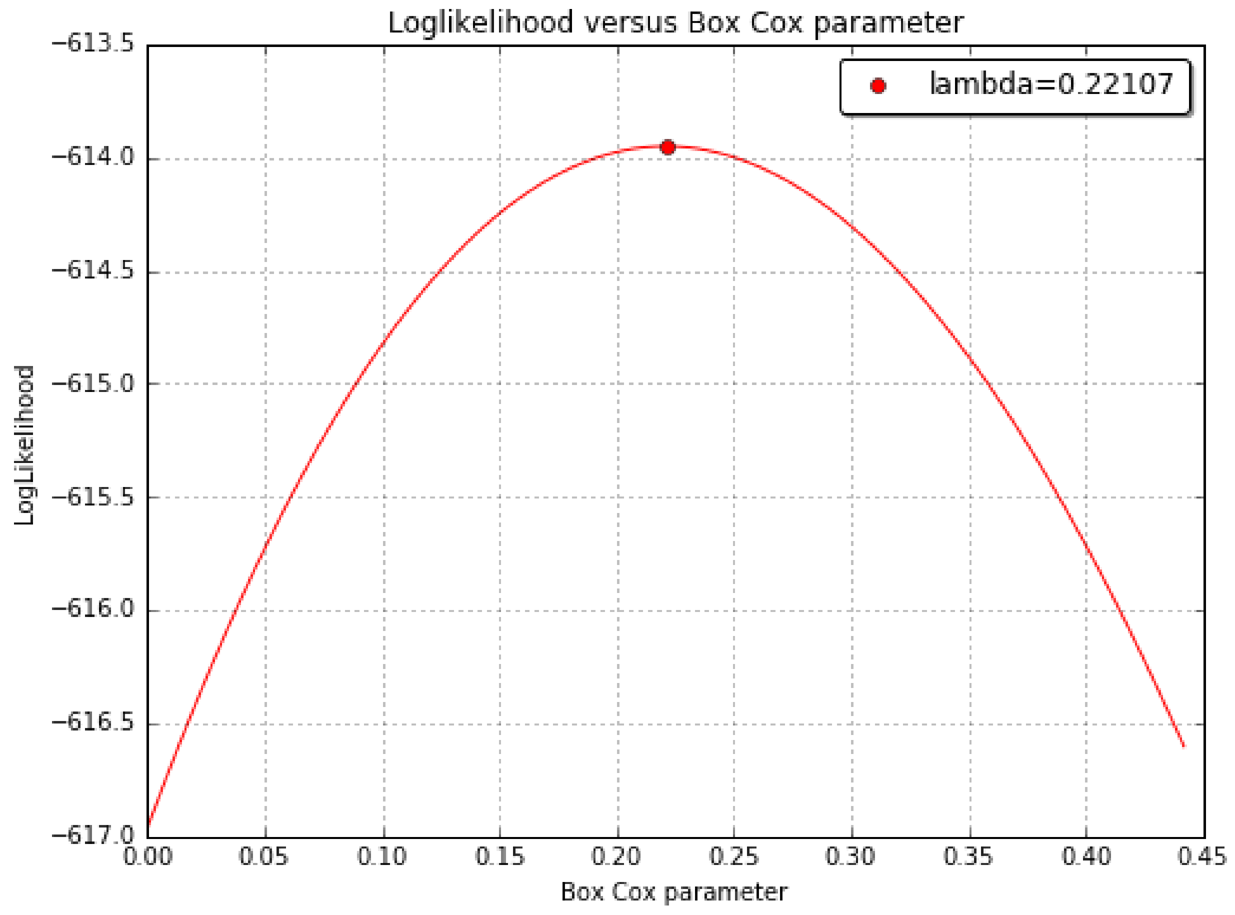
The residuals QQ plot

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

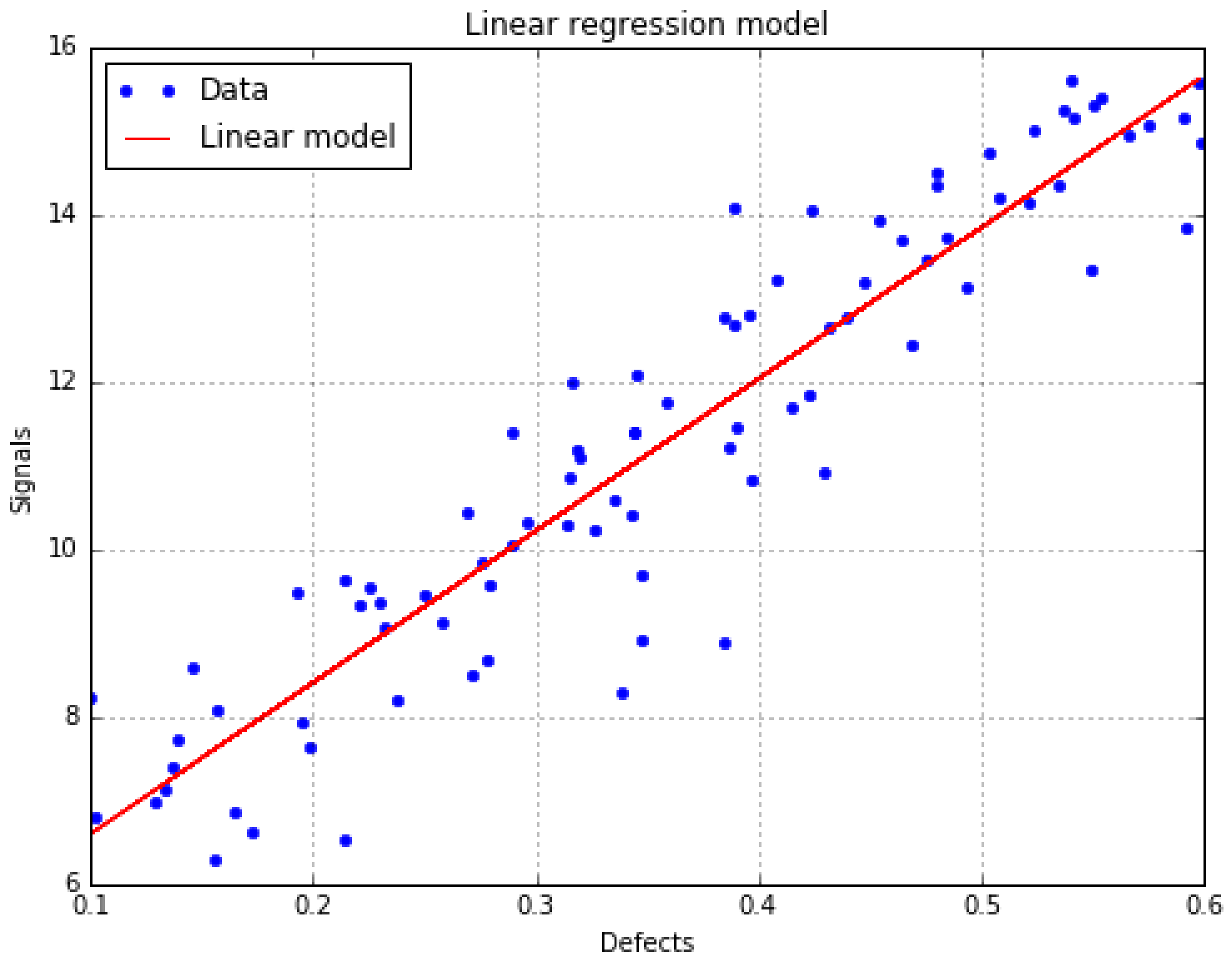
Save all results in a csv file

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

Show graphs

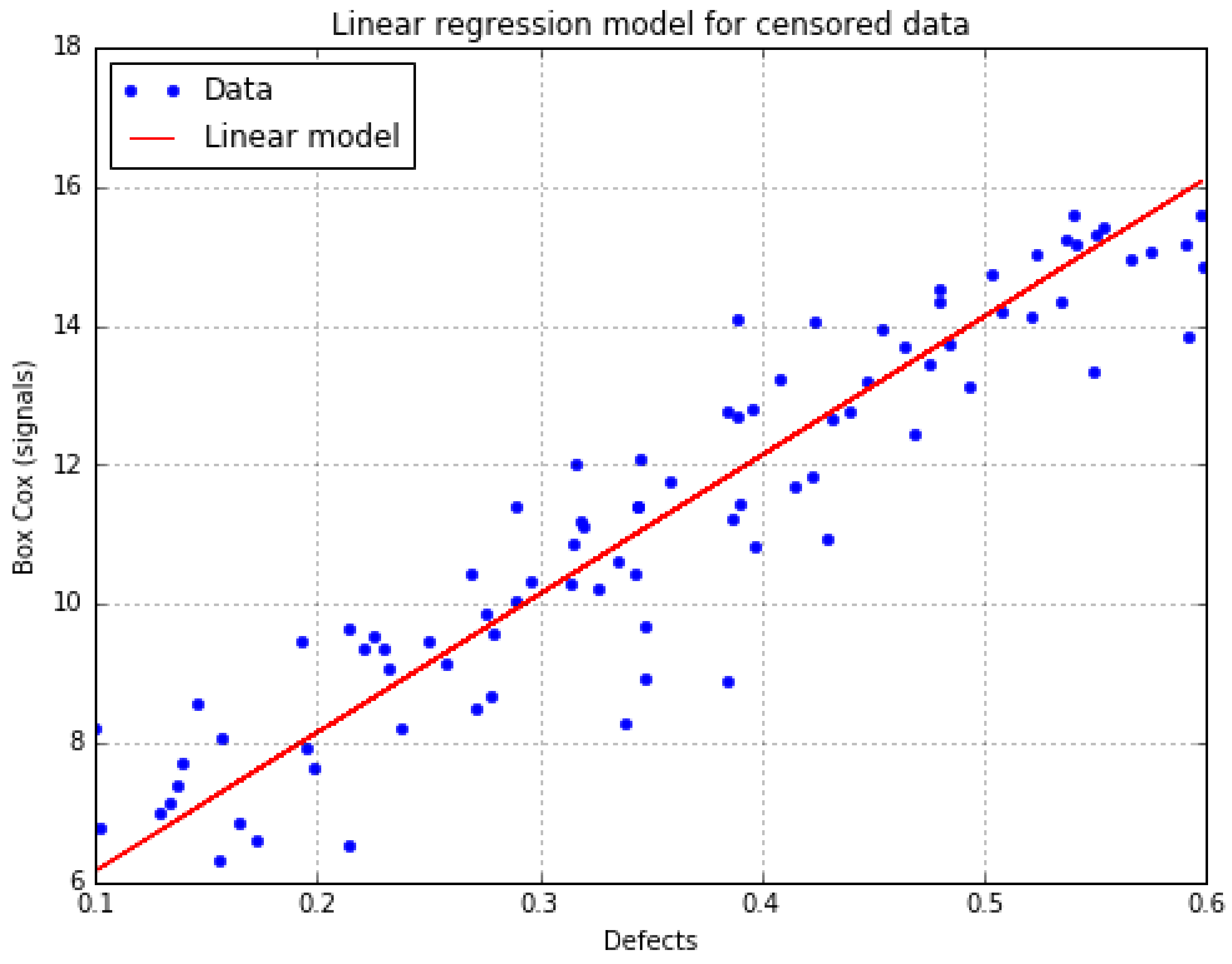
The linear regression model with data for the uncensored case (default case)

```
# 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])
```

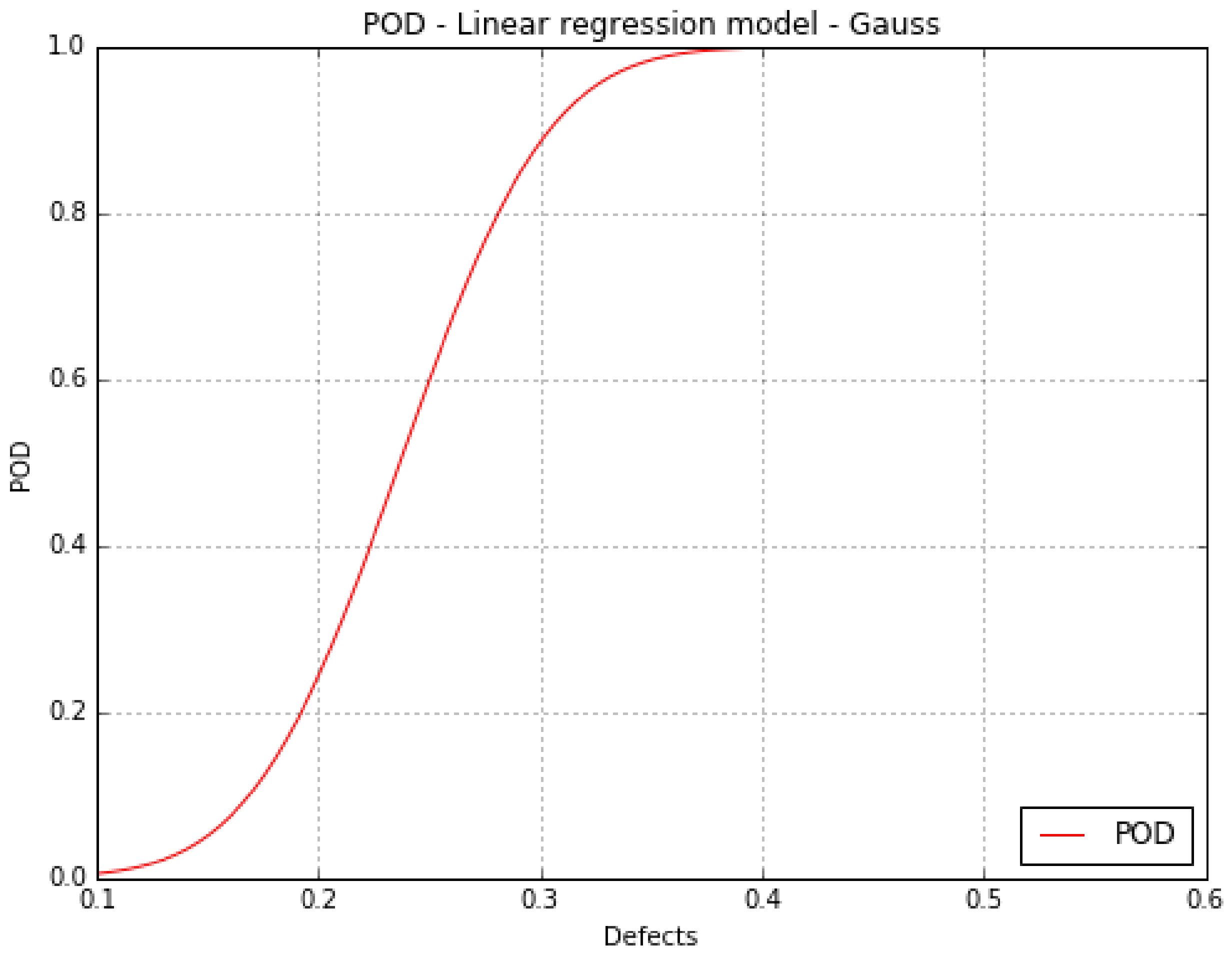
```
POD : 0.886
POD at level 0.95 : 0.834
```

Show POD graphs

Only the mean POD

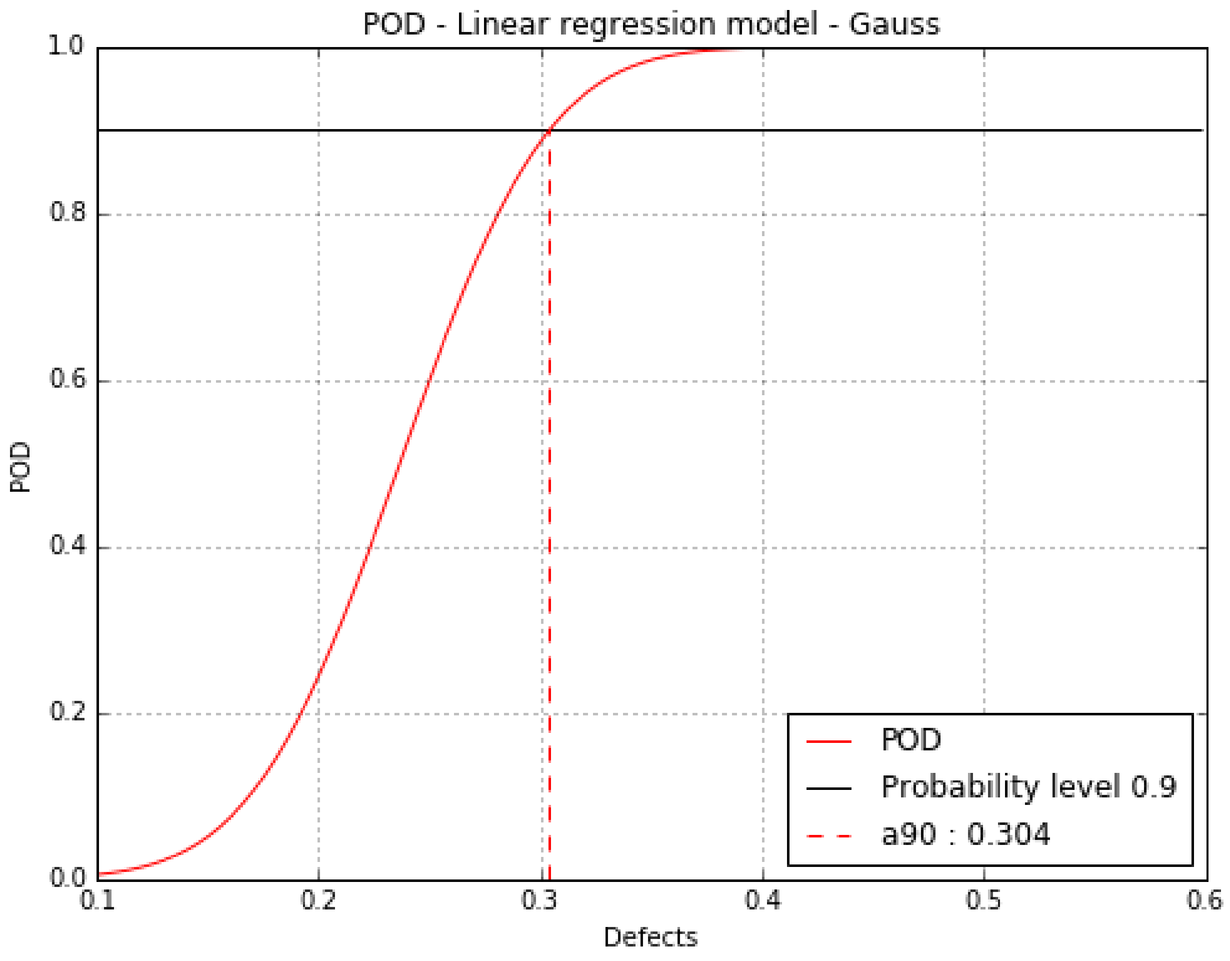
```
fig, ax = PODGauss.drawPOD()
fig.show()
```

```
/home/dumas/anaconda2/lib/python2.7/site-packages/matplotlib/figure.py:397: UserWarning: matplotlib i
  "matplotlib is currently using a non-GUI backend, "
```



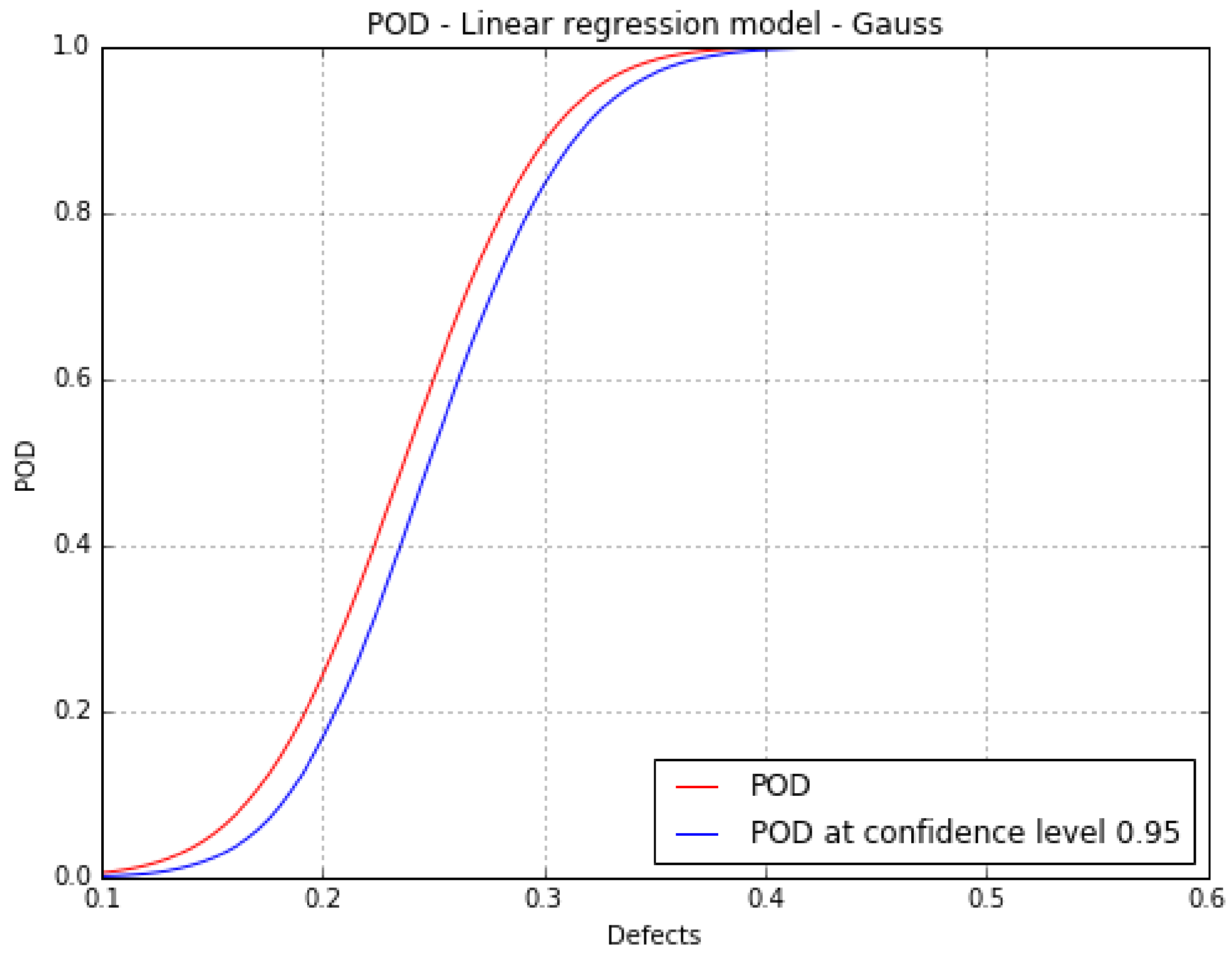
Mean POD with the detection size for a given probability level

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



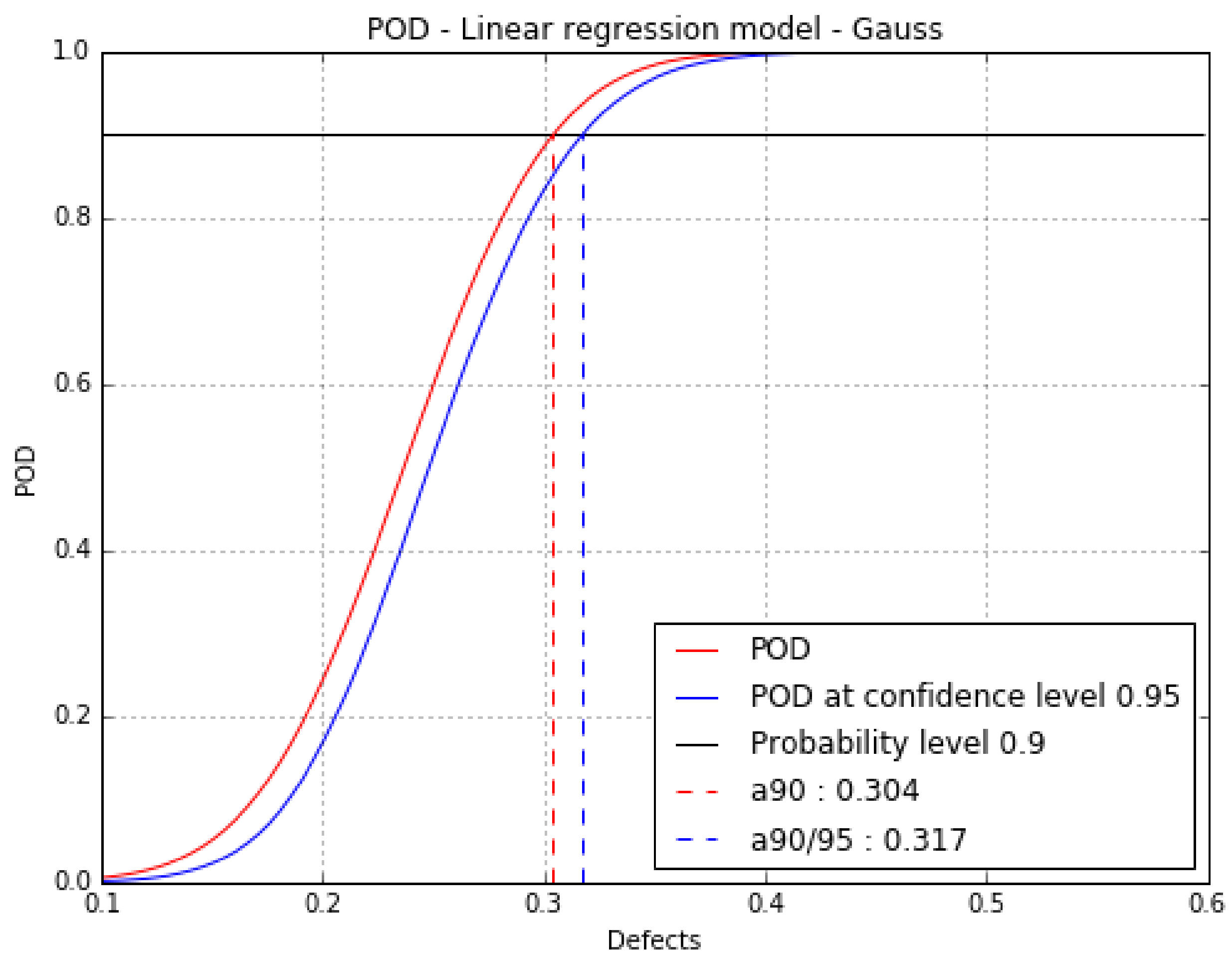
Mean POD with POD at confidence level

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



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

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



Build POD with no hypothesis on the residuals

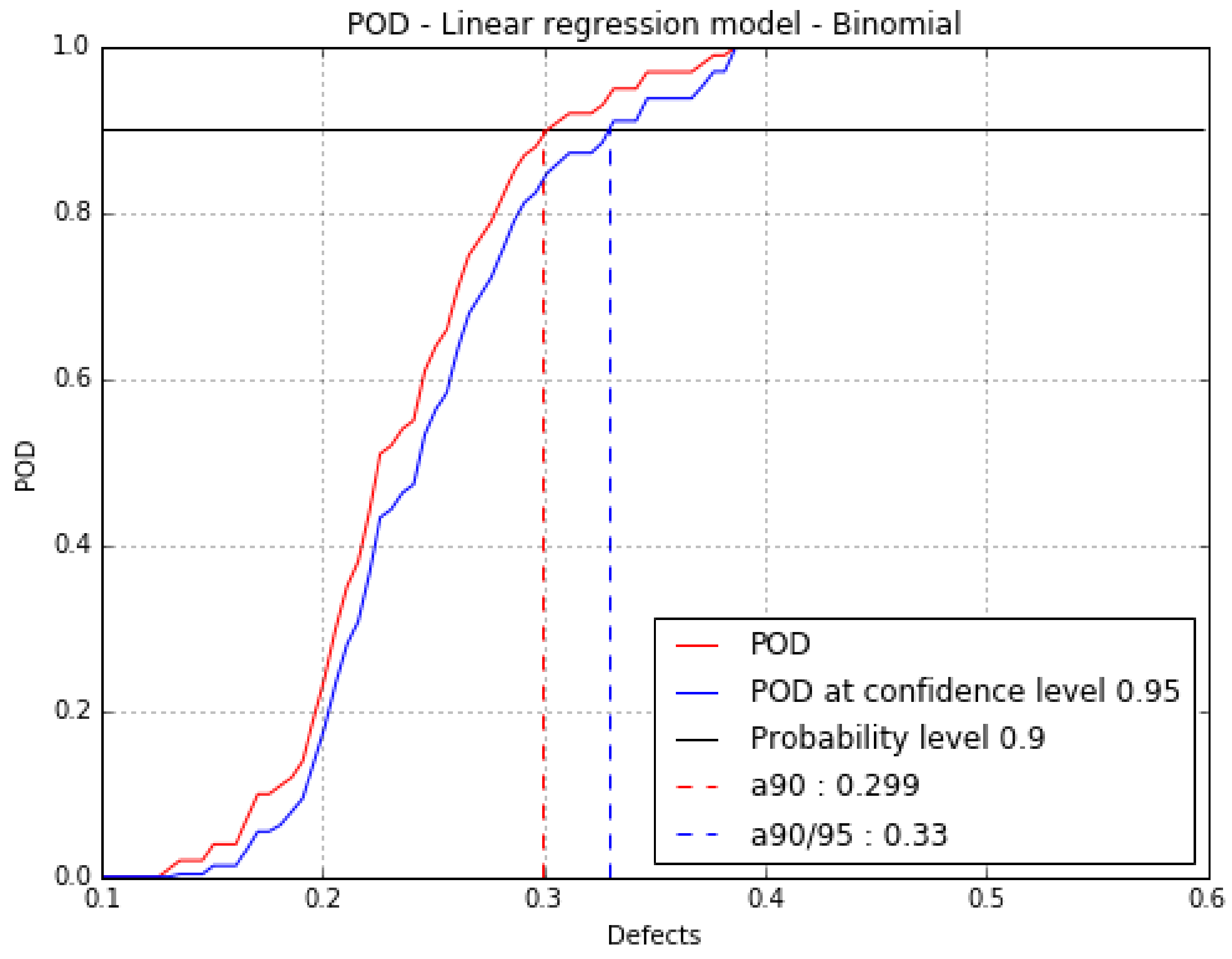
This corresponds with the Berens Binomial method.

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

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

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

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



Build POD with kernel smoothing on the residuals

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

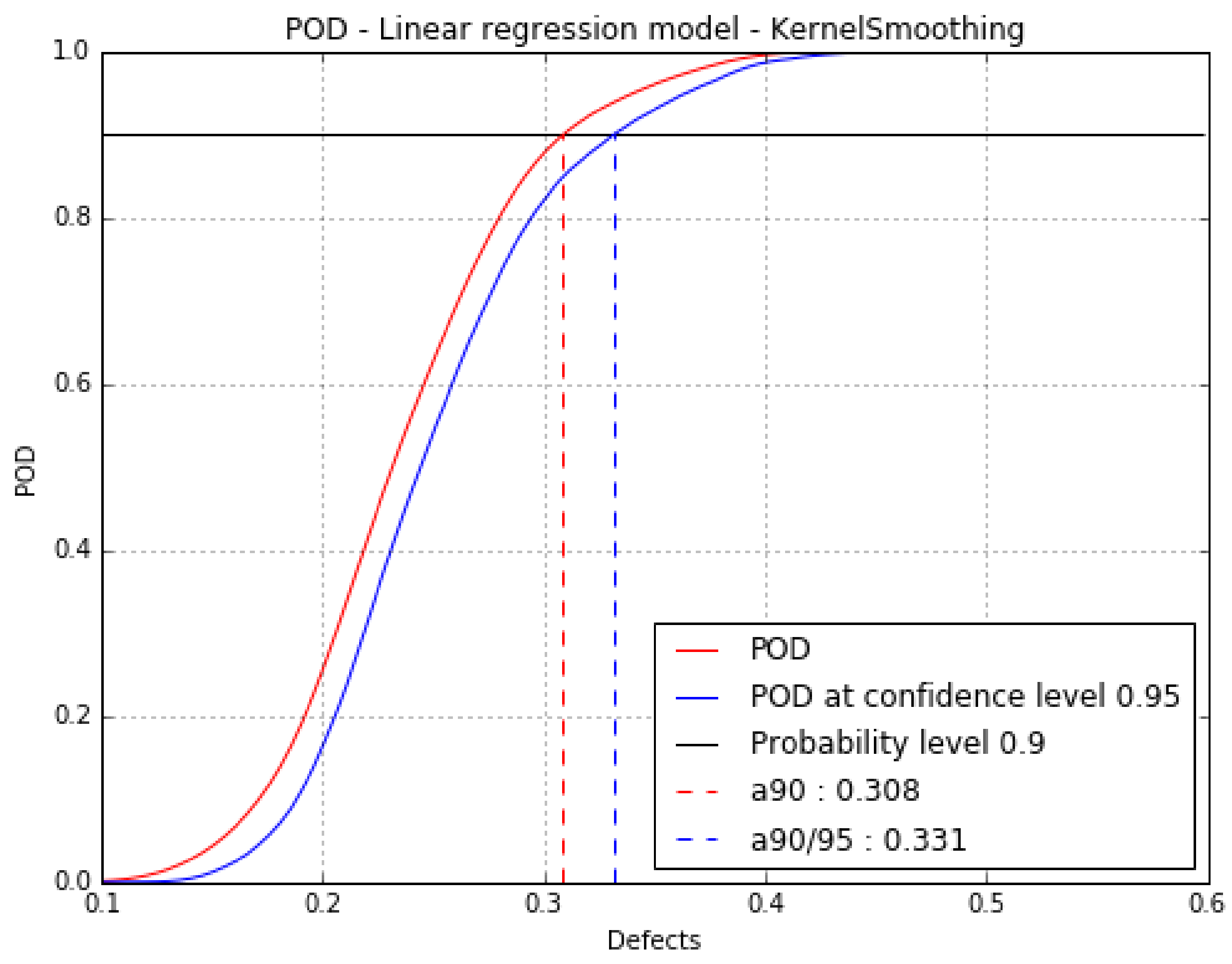
```
PODks = otpod.UnivariateLinearModelPOD(defects, signals, detection,
                                        resDistFact=ot.KernelSmoothing(),
                                        boxCox=True)
PODks.run()
```

```
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

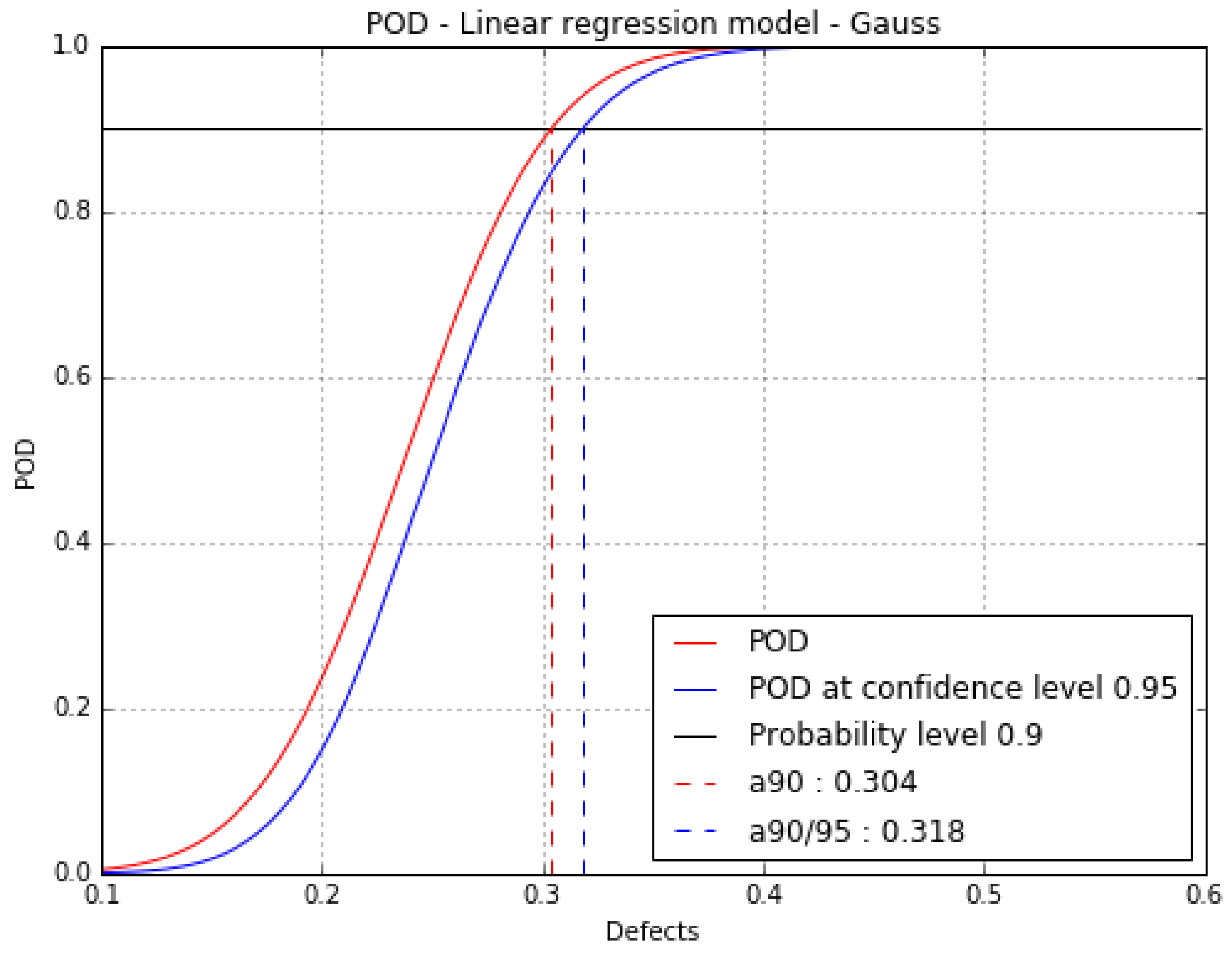
```
# get the POD model
PODmodel = PODGauss.getPODModel()
# get the POD model at the given confidence level
PODmodelCl95 = PODGauss.getPODCLModel(0.95)

# 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 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.
```

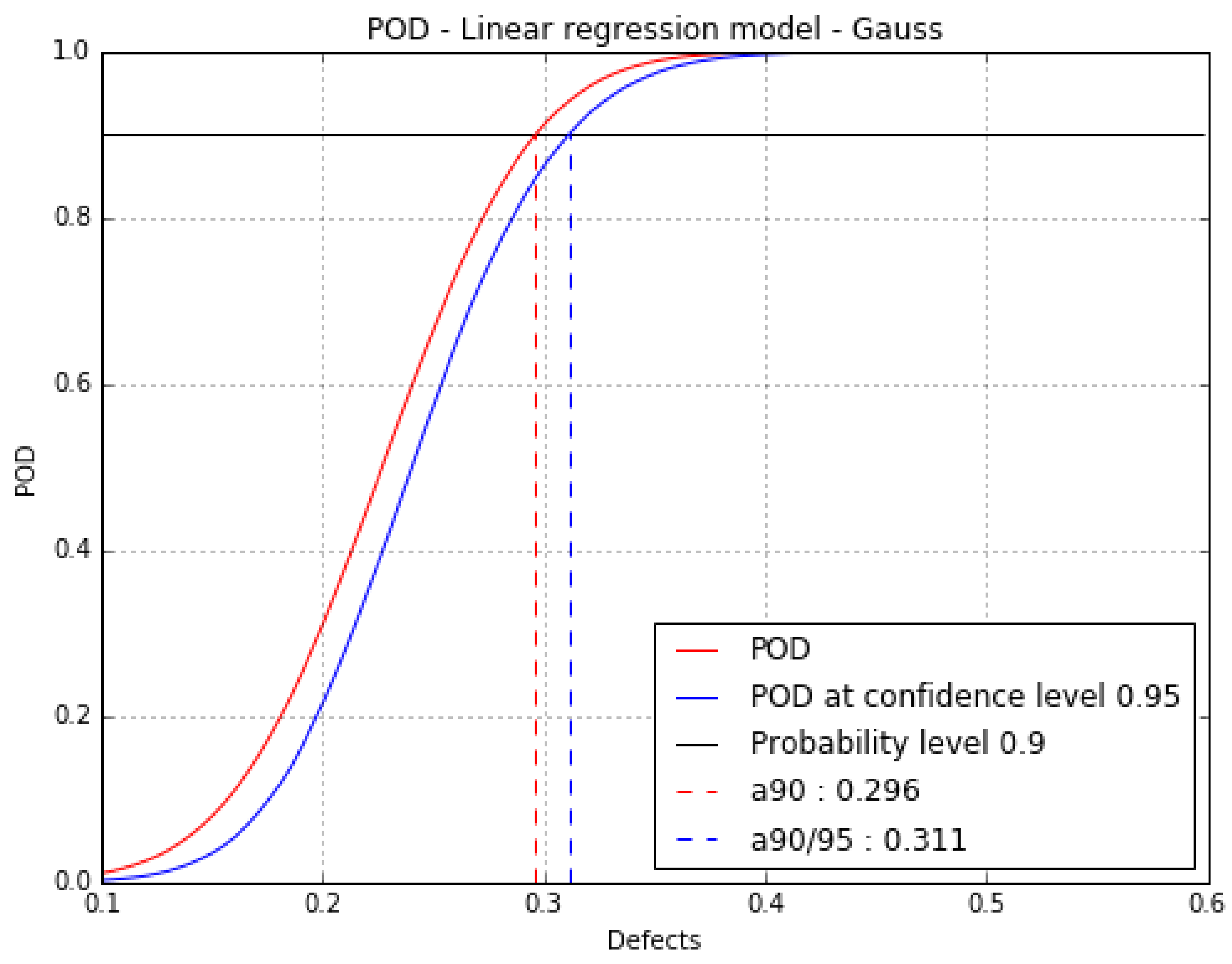
```
result = otpod.DataHandling.filterCensoredData(defects, signals,
                                               noiseThres, saturationThres)
defectsFiltered = result[0]
signalsFiltered = result[3]
```

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

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

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

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



ipynb source code

1.2.5 Quantile Regression POD

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

Generate data

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

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

Build POD with quantile regression technique

```
# signal detection threshold
detection = 200.
# The POD with censored data actually builds a POD only on filtered data.
# A warning is 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 : 209.77 s
```

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

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

```
t0 = time()
PODsimulSize100 = otpod.QuantileRegressionPOD(defects, signals, detection,
                                                  boxCox=True)
PODsimulSize100.setSimulationSize(100) # default is 1000
PODsimulSize100.run()
print 'Computing time : {:0.2f} s'.format(time()-t0)
```

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

Compute detection size

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

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

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

get POD NumericalMathFunction

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

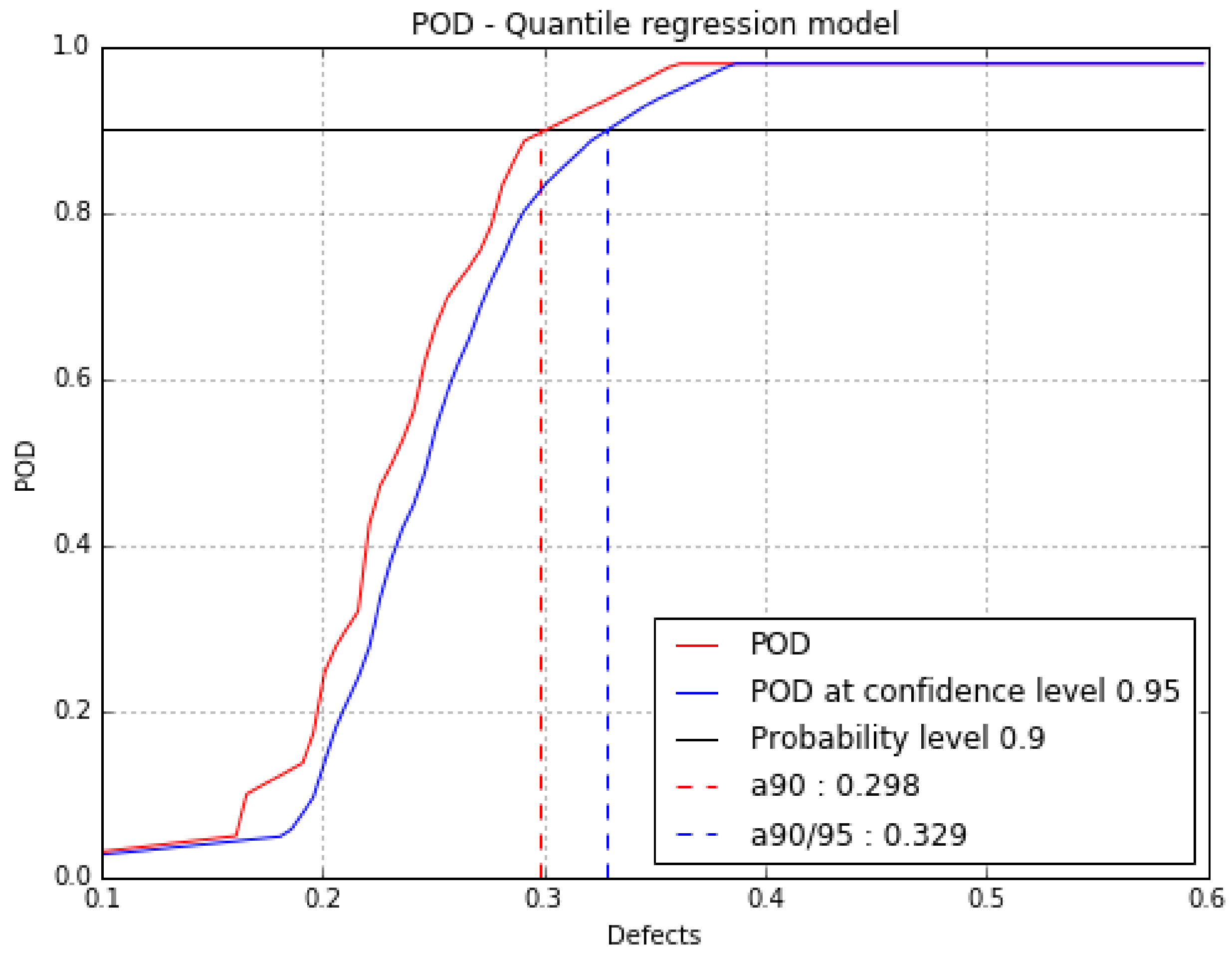
```
print 'Pseudo R2 for quantile 0.9 : {:0.3f}'.format(POD.getR2(0.9))
print 'Pseudo R2 for quantile 0.95 : {:0.3f}'.format(POD.getR2(0.95))
```

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

Show POD graphs

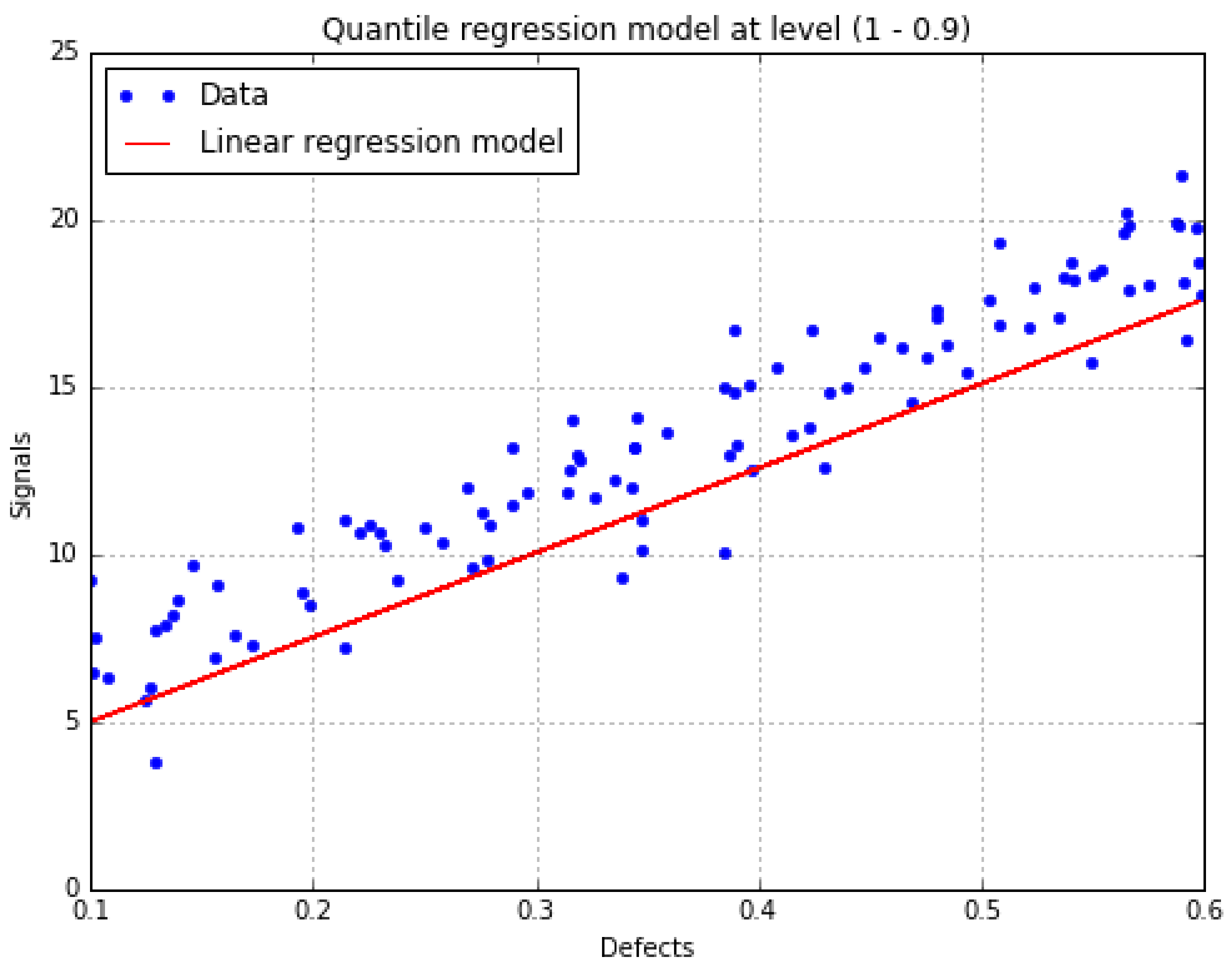
Mean POD and POD at confidence level with the detection size for a given probability level

```
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
```

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)
```

User-defined defect sizes

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

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

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

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

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

ValueError                                Traceback (most recent call last)

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


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


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

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

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

Running the polynomial chaos based POD

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

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

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

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

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

Compute detection size

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

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

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

get POD NumericalMathFunction

```
# get the POD model
PODmodel = POD.getPODModel()
# get the POD model at the given confidence level
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 : 0.871
POD at level 0.95 : 0.841
```

Compute the R2 and the Q2

Enable to check the quality of the model.

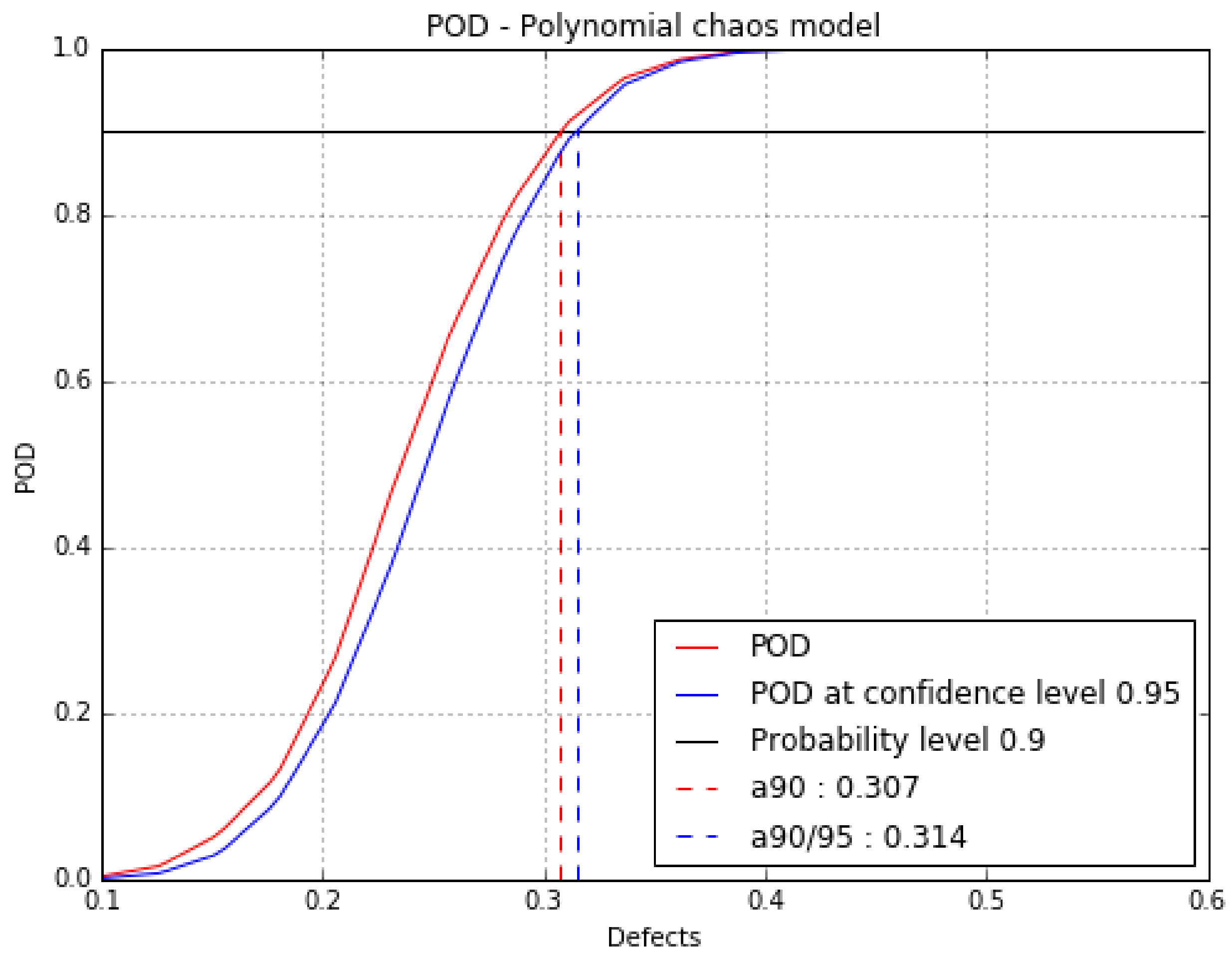
```
print 'R2 : {:0.4f}'.format(POD.getR2())
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

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

Show POD graphs

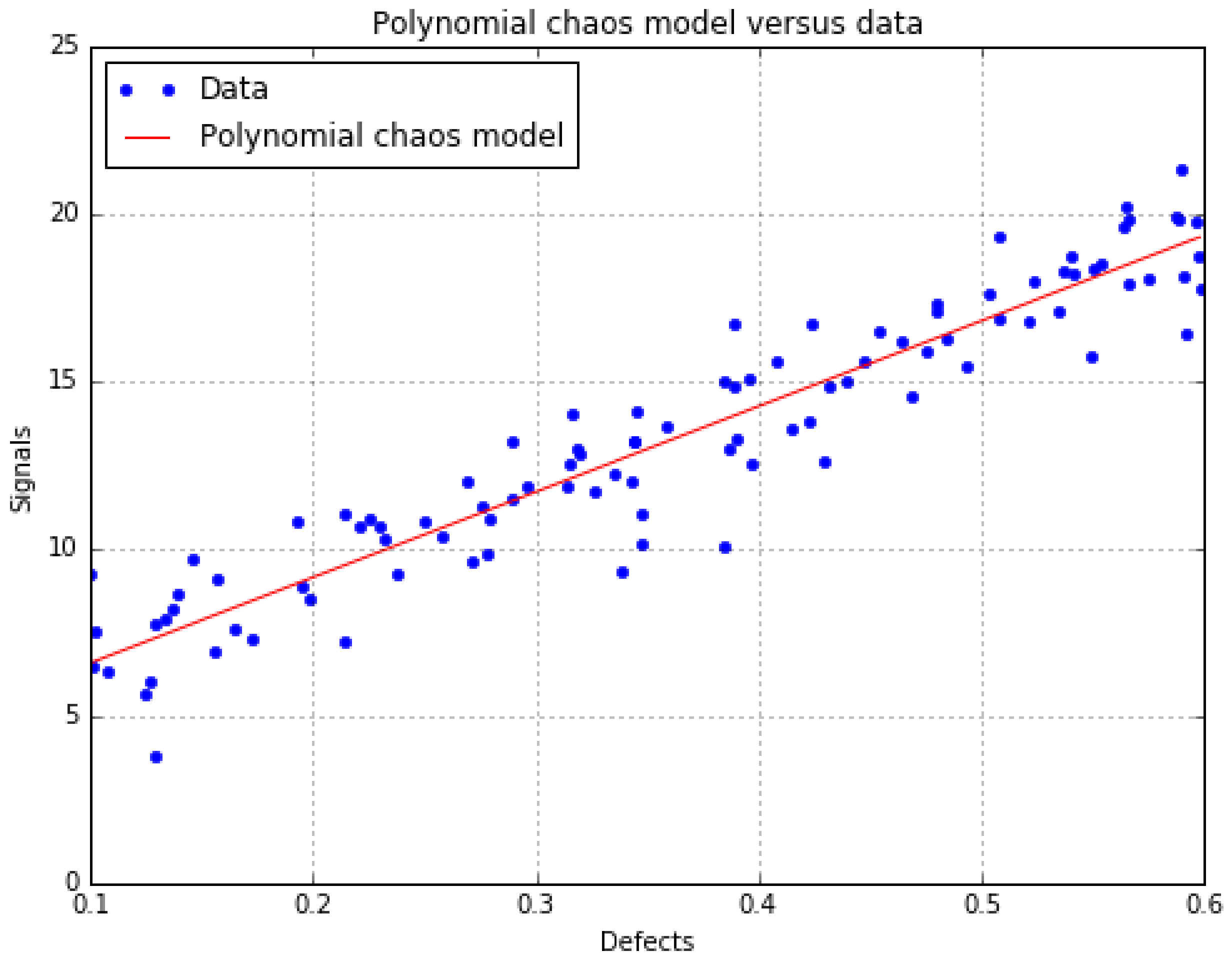
Mean POD and POD at confidence level with the detection size for a given probability level

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



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

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



Advanced user mode

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

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

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

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

```
p = 1
indexMax = enumerateFunction.getStrataCumulatedCardinal(p)
adaptiveStrategy = ot.FixedStrategy(multivariateBasis, indexMax)

PODnew.setAdaptiveStrategy(adaptiveStrategy)
```

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

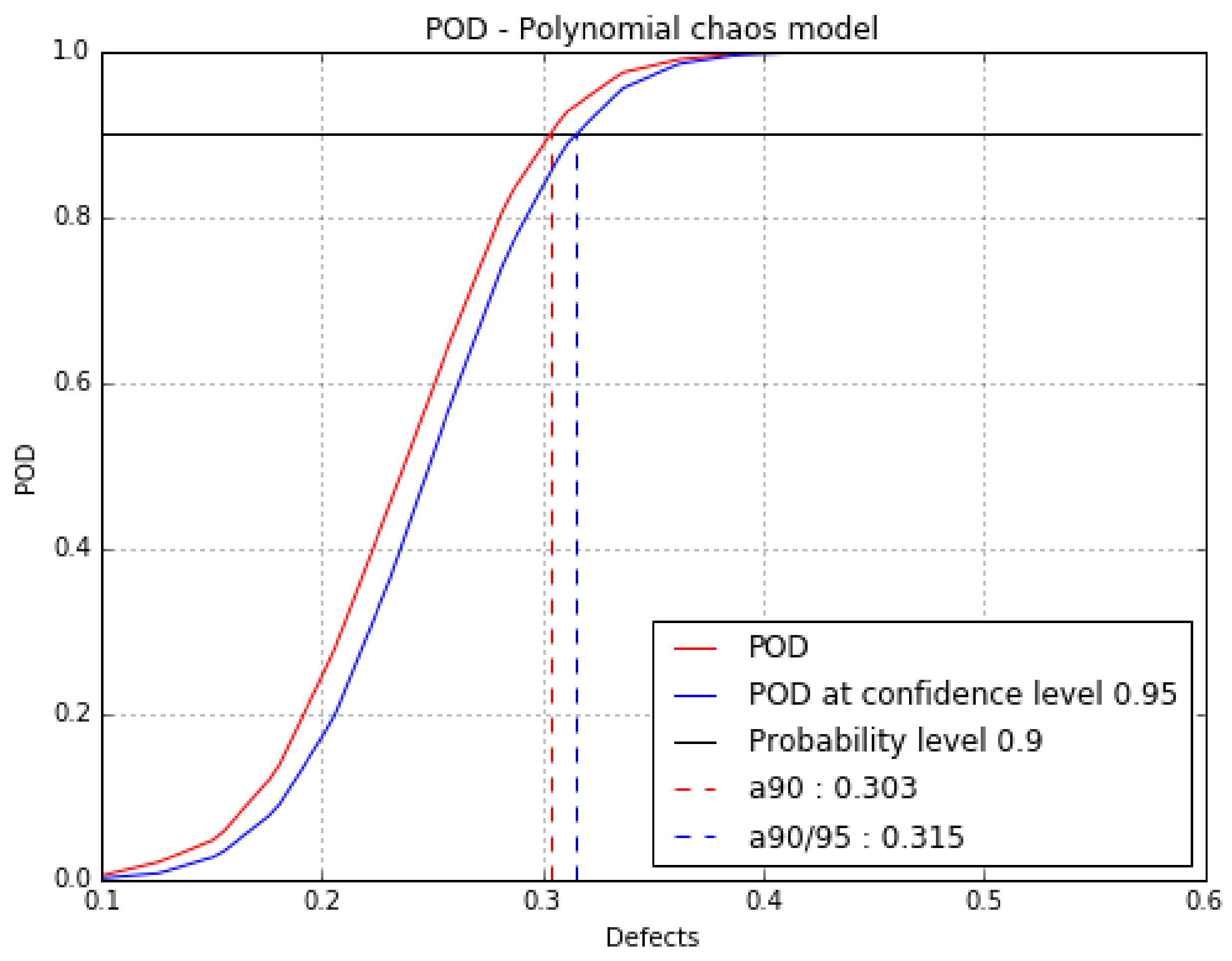
```
PODnew.run()
```

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

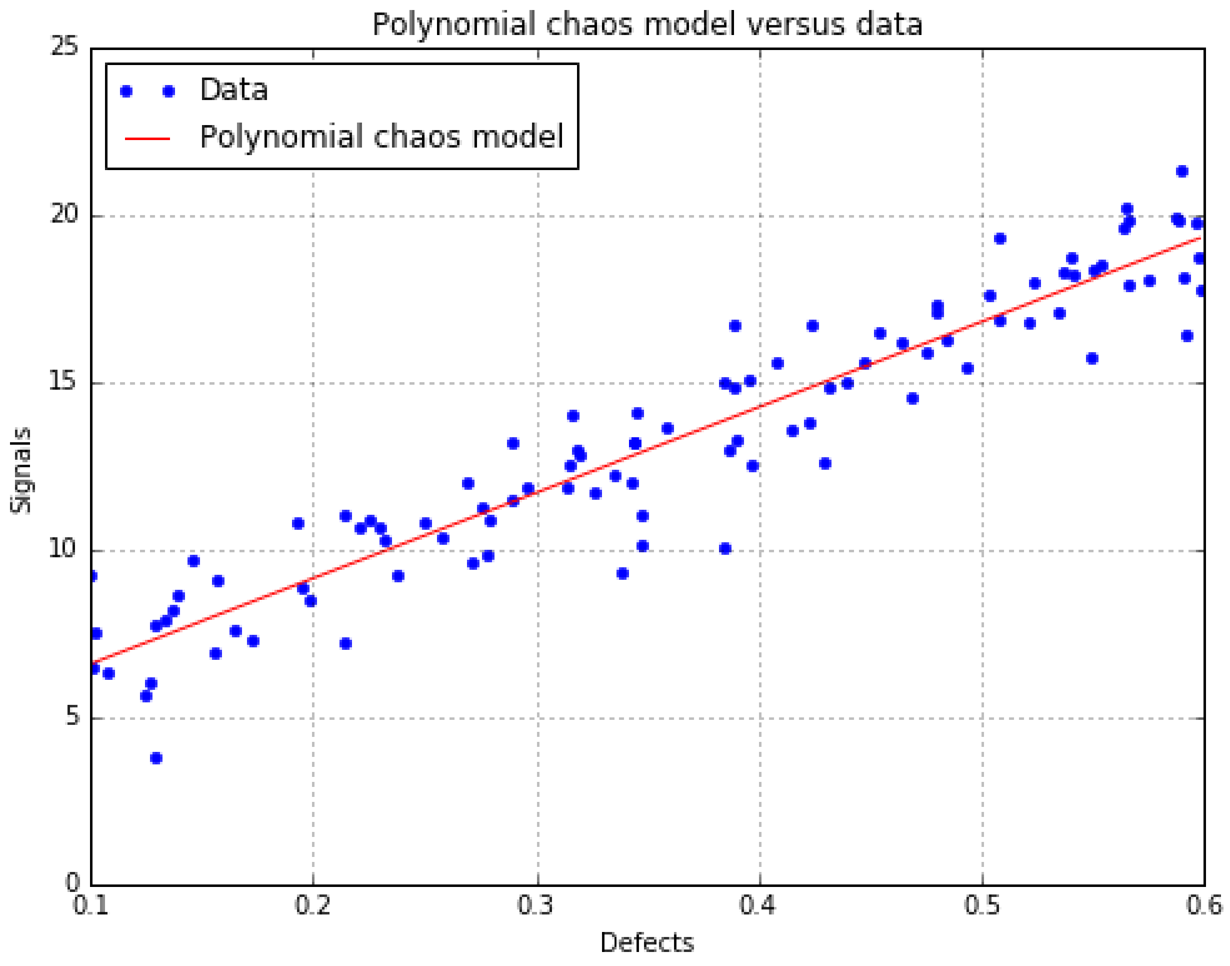
```
print PODnew.computeDetectionSize(0.9, 0.95)
print 'R2 : {:0.4f}'.format(POD.getR2())
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

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

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



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



ipynb source code

1.2.7 Kriging POD

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

Generate data

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

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.609524],
     [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469 ],
     [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.768369],
     [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.636588],
     [ 40.643948]])
```

Build POD with Kriging model

```
# signal detection threshold
detection = 38.
# The POD with censored data actually builds a POD only on filtered data.
# A warning is 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_tools.pyc in setDefectSizes(self, size)
    230             raise ValueError('Defect sizes must range between ' + \
    231                              '{:0.4f} '.format(np.ceil(minMin*10000)/10000) + \
--> 232                              'and {:0.4f}.'.format(np.floor(maxMax*10000)/10000))
    233         self._defectNumber = self._defectSizes.shape[0]
    234


ValueError: Defect sizes must range between 3.9550 and 5.1019.
```

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

Running the Kriging based POD

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

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

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

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

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

Compute detection size

```
# Detection size at probability level 0.9
# and confidence level 0.95
```

```
print POD.computeDetectionSize(0.9, 0.95)

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

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

get POD NumericalMathFunction

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

# compute the probability of detection for a given defect value
print 'POD : {:0.3f}'.format(PODmodel([4.2])[0])
print 'POD at level 0.95 : {:0.3f}'.format(PODmodelCl95([4.2])[0])
```

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

Compute the Q2

Enable to check the quality of the model.

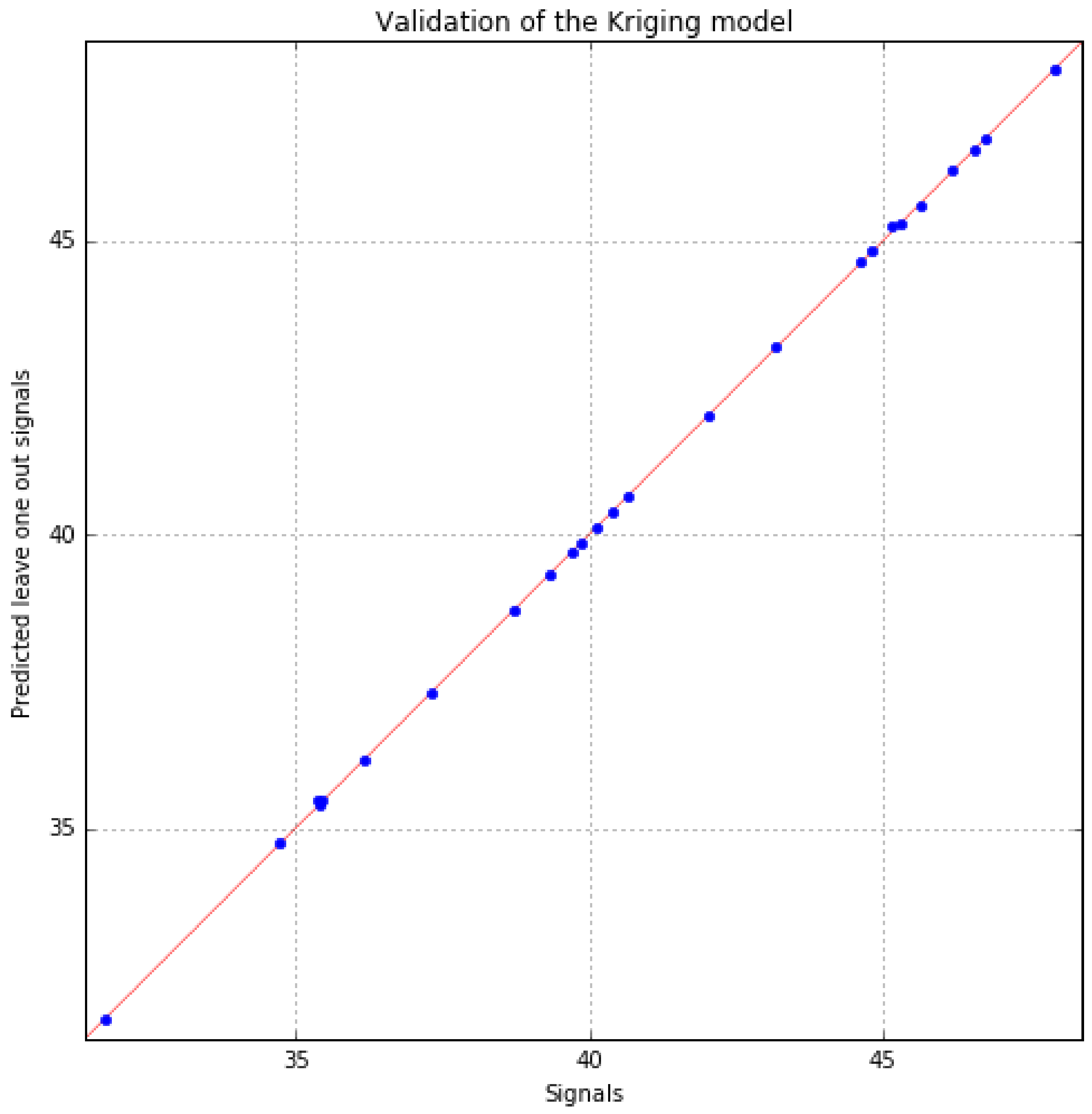
```
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

```
Q2 : 1.0000
```

Draw the validation graph

The predictions are the one computed by leave one out.

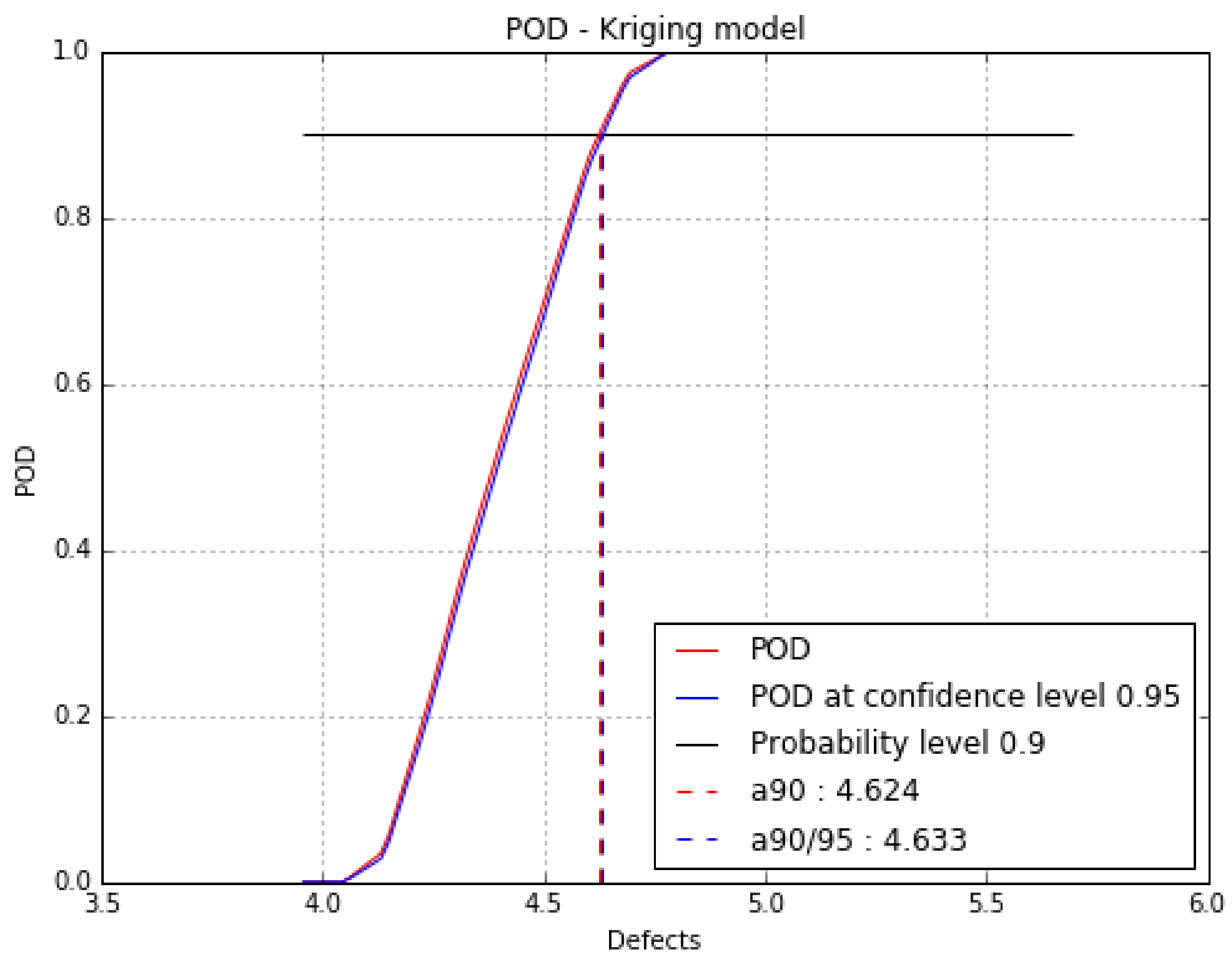
```
fig, ax = POD.drawValidationGraph()
fig.show()
```



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)
```

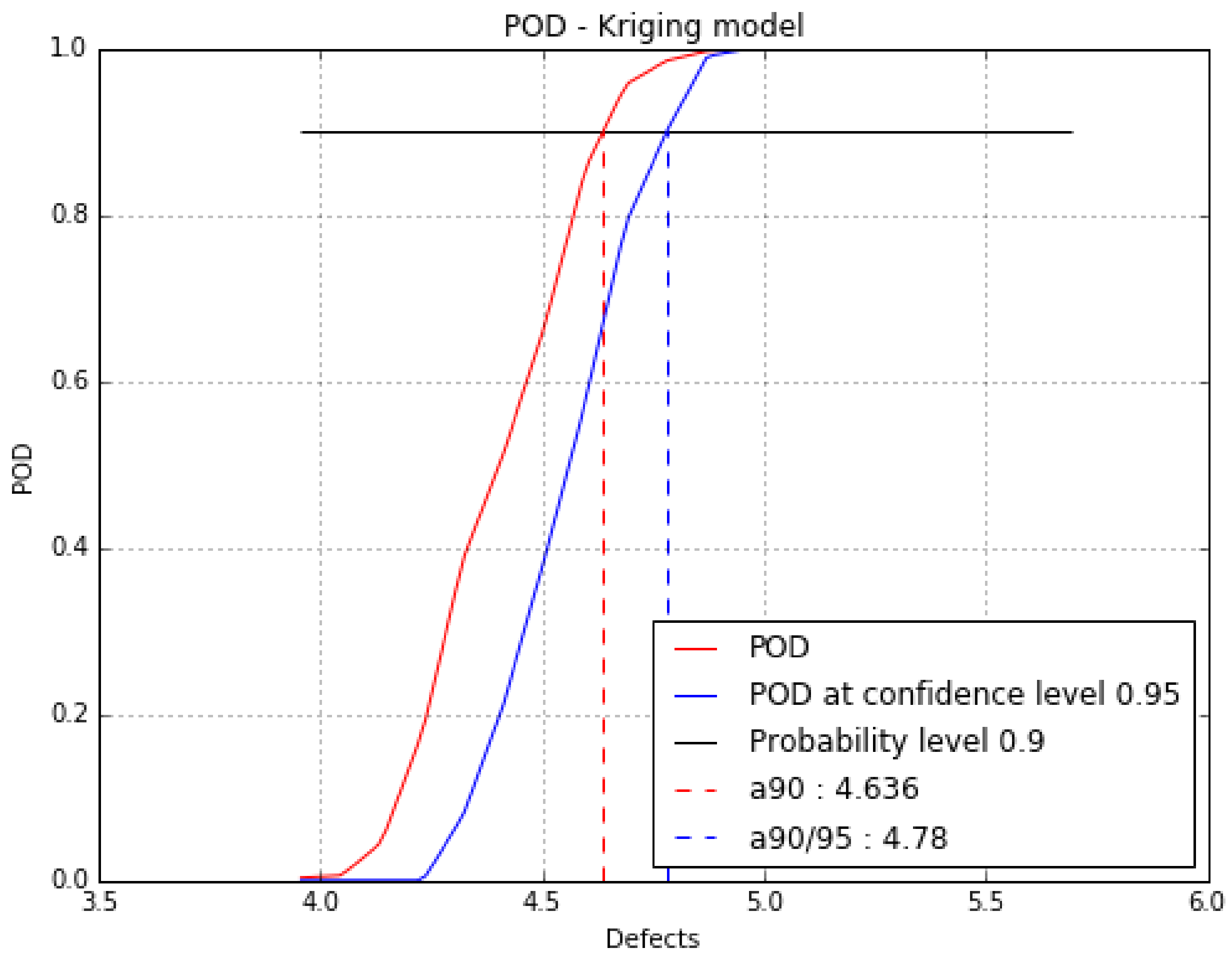
```
# set the covariance Model as an absolute exponential model
covColl = ot.CovarianceModelCollection(4)
for i in xrange(4):
    covColl[i]  = ot.AbsoluteExponential(1, 1.)
covarianceModel = ot.ProductCovarianceModel(covColl)

PODnew.setCovarianceModel(covarianceModel)
PODnew.run()
```

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

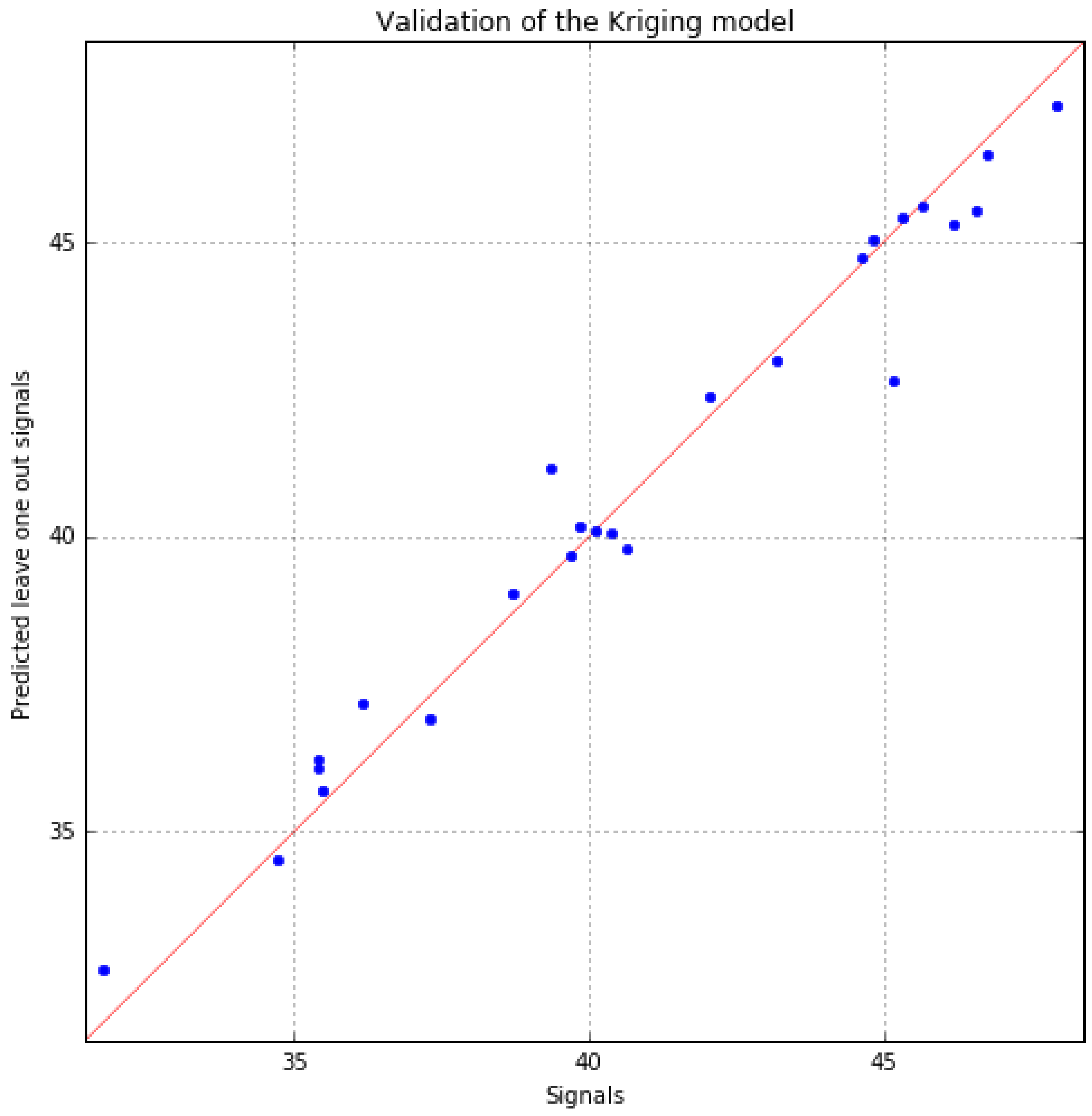
```
print PODnew.computeDetectionSize(0.9, 0.95)
print 'Q2 : {:0.4f}'.format(POD.getQ2())
```

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

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



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



ipynb source code

1.2.8 POD Summary

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

Generate data

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

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.609524],
     [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469 ],
     [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.768369],
     [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.636588],
     [ 40.643948]])
```

Compute POD with several methods

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

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

```
# signal detection threshold
detection = 38.
# The POD summary take
POD = otpod.PODSummary(inputSample, signals, detection)
# The main parameters can modified :
# The number of simulation to compute the confidence level
POD.setSimulationSize(500)
# The number of Monte Carlo simulation to compute the POD for polynomial chaos and kriging
POD.setSamplingSize(2000)
# Deactivate the quantile regression technique
POD.setMethodActive('QuantileRegression', False)
```

```
# Finally run
POD.run()
```

```
Start univariate linear model analysis...

Start univariate linear model POD with Gaussian residuals...

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

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

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

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

Access to the dictionnary of the active methods

```
POD.getMethodActive()
```

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

Show results

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

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

```
POD.printResults()
```

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

                                                      Uncensored

Intercept coefficient :                                     0.02
Slope coefficient :                                         8.71
Standard error of the estimate :                            2.29
```

```
Confidence interval on coefficients
Intercept coefficient :                        [-10.03, 10.07]
Slope coefficient :                            [6.58, 10.85]
Level :                                                  0.95

Quality of regression
R2 (> 0.8):                                              0.76
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
         Residuals analysis results
--------------------------------------------------------------------------------
Fitted distribution (uncensored) :             Normal(mu = -7.10543e-15, sigma = 2.2441)

                                                  Uncensored
Distribution fitting test
Kolmogorov p-value (> 0.05):                             0.99

Normality test
Anderson Darling p-value (> 0.05):                       0.76
Cramer Von Mises p-value (> 0.05):                       0.83

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

Homoskedasticity test (constant variance)
Breush Pagan p-value (> 0.05):                           0.09
Harrison McCabe p-value (> 0.05):                        0.21

Non autocorrelation test
Durbin Watson p-value (> 0.05):                          0.34
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
         Model validation results
--------------------------------------------------------------------------------
                                                     Uncensored    Censored

                                                  R2          Q2          R2

Linear Regression (> 0.8):                       0.76
Polynomial Chaos (> 0.8):                         1.0         1.0
Kriging (> 0.8):                                              1.0
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
         POD results
--------------------------------------------------------------------------------
                                                          a90        a90/95
Linear Regression
Gaussian residuals :                                      4.7          4.85
No residuals hypothesis :                                4.71          4.88
Kernel smoothing on residuals :                          4.75          4.82

Polynomial chaos :                                       4.63          4.65
Kriging :                                                4.62          4.63
--------------------------------------------------------------------------------
```

Results can be displayed for another probability and confidence level.

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

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

                                                    Uncensored

Intercept coefficient :                                   0.02
Slope coefficient :                                       8.71
Standard error of the estimate :                          2.29

Confidence interval on coefficients
Intercept coefficient :                          [-10.03, 10.07]
Slope coefficient :                              [6.58, 10.85]
Level :                                                   0.95

Quality of regression
R2 (> 0.8):                                               0.76
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
         Residuals analysis results
--------------------------------------------------------------------------------
Fitted distribution (uncensored) :               Normal(mu = -7.10543e-15, sigma = 2.2441)

                                                    Uncensored
Distribution fitting test
Kolmogorov p-value (> 0.05):                              0.99

Normality test
Anderson Darling p-value (> 0.05):                        0.76
Cramer Von Mises p-value (> 0.05):                        0.83

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

Homoskedasticity test (constant variance)
Breush Pagan p-value (> 0.05):                            0.09
Harrison McCabe p-value (> 0.05):                         0.21

Non autocorrelation test
Durbin Watson p-value (> 0.05):                           0.34
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
         Model validation results
--------------------------------------------------------------------------------
                                                   Uncensored   Censored

                                                    R2          Q2          R2

Linear Regression (> 0.8):                        0.76
Polynomial Chaos (> 0.8):                          1.0         1.0
Kriging (> 0.8):                                               1.0
```

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


-------------------------------------------------------------------------------
         POD results
-------------------------------------------------------------------------------
                                                              a80        a80/90
Linear Regression
Gaussian residuals :                                         4.58          4.67
No residuals hypothesis :                                    4.64          4.68
Kernel smoothing on residuals :                              4.61          4.69

Polynomial chaos :                                           4.55          4.57
Kriging :                                                    4.55          4.56
-------------------------------------------------------------------------------
```

Save results

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

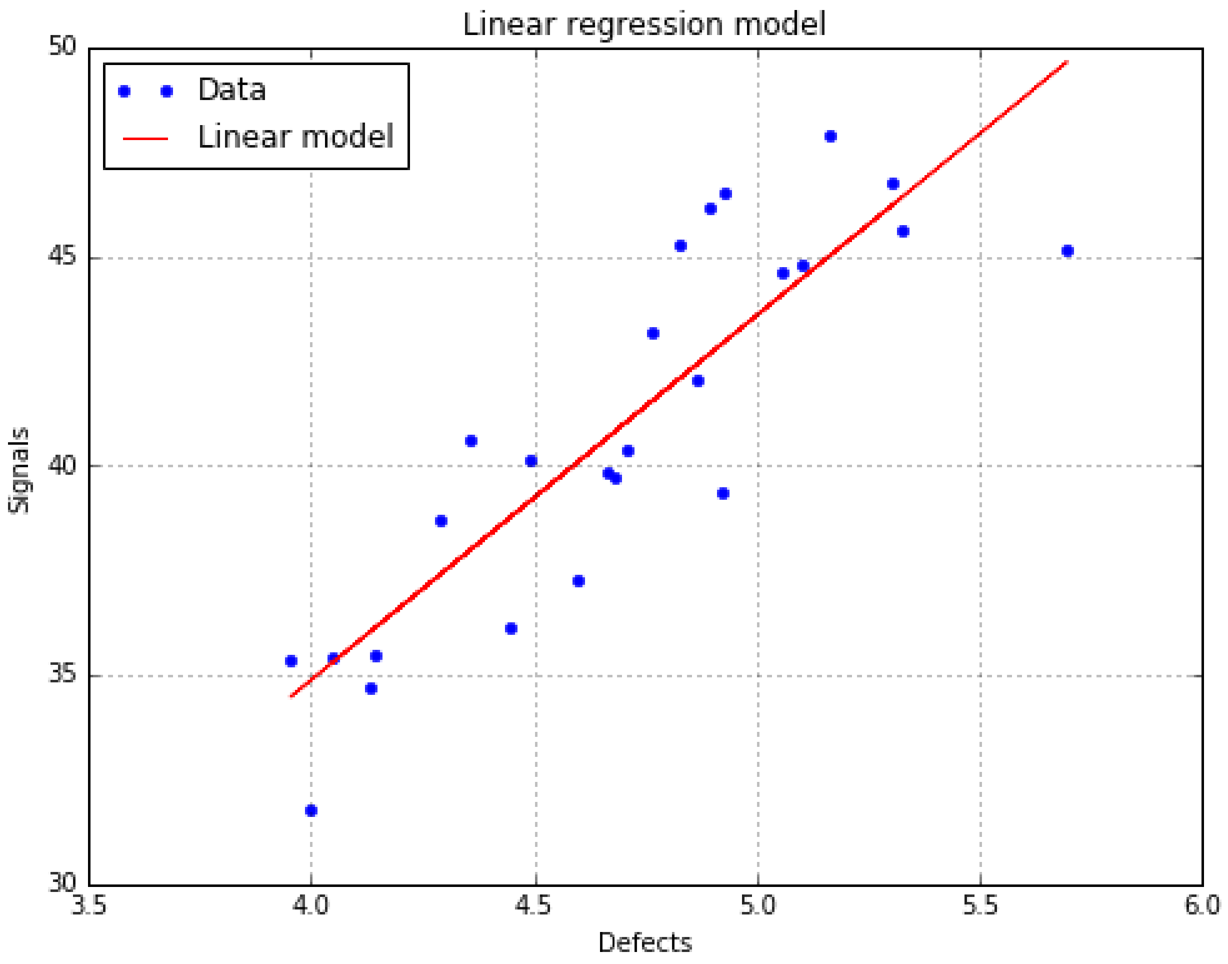
```
POD.saveResults('results.csv', probabilityLevel=0.9, confidenceLevel=0.95)
```

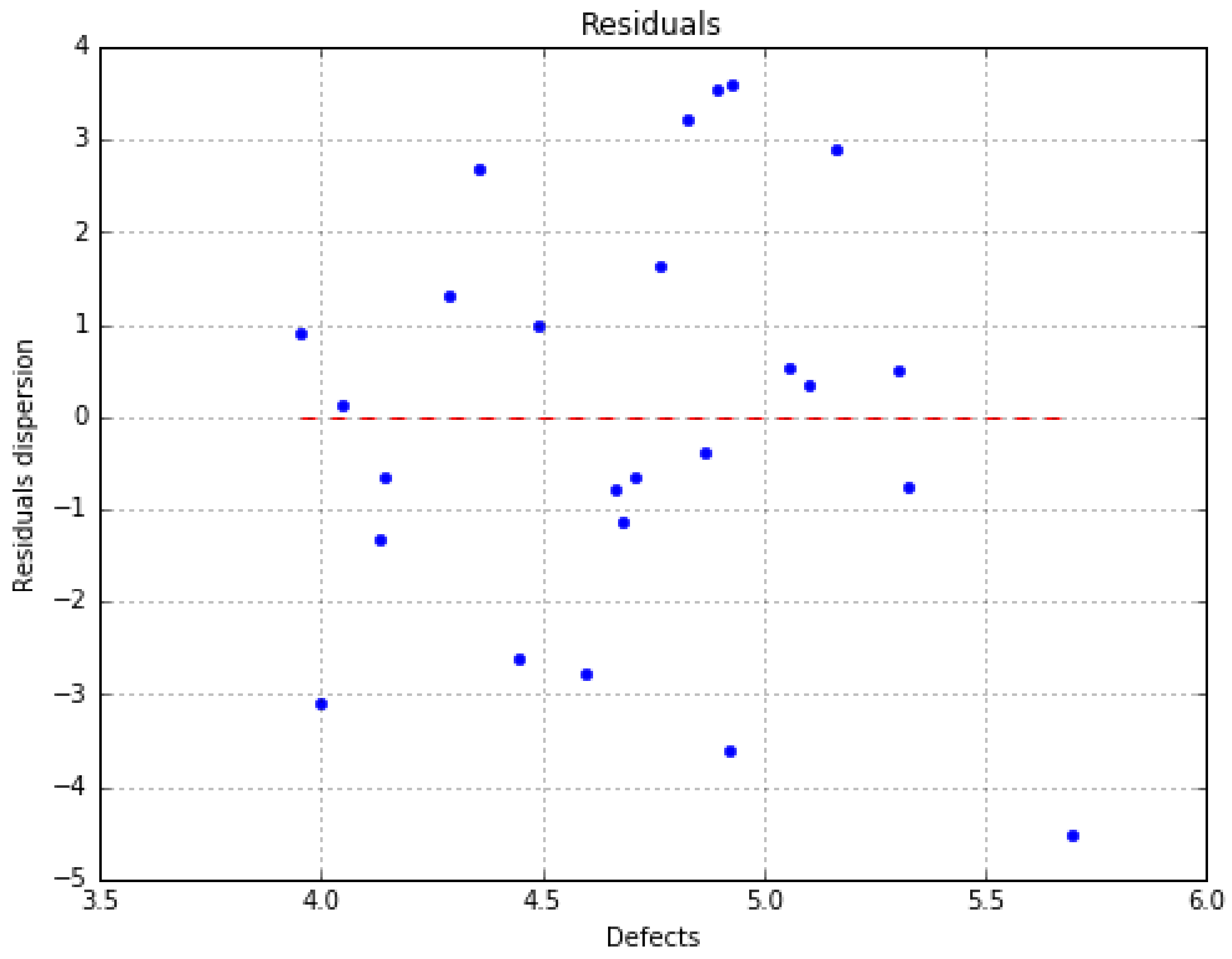
Draw and save graphs

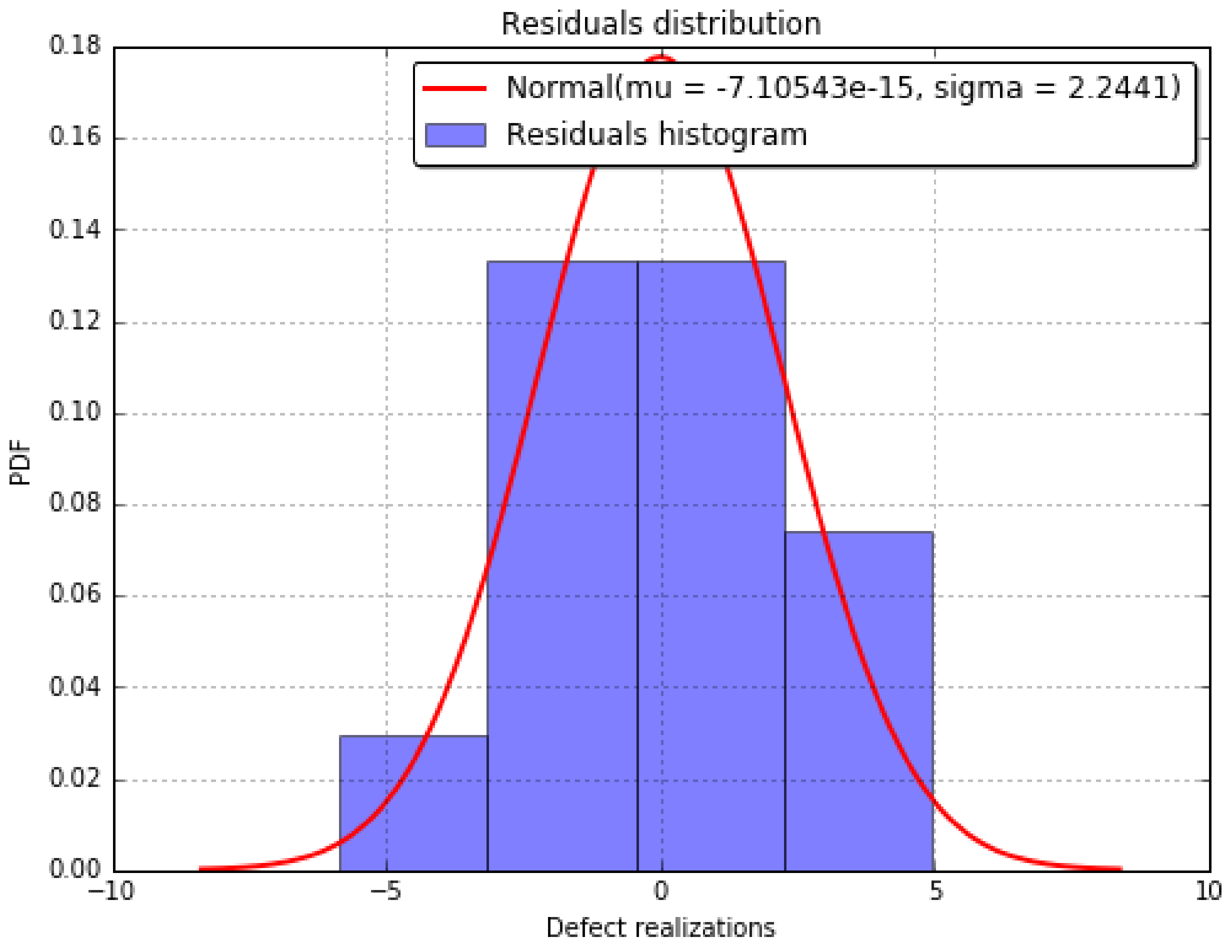
All available graphs can be saved using the method *saveGraphs*. A specific directory and the extension of the files can be given as parameters. As before the probability level and confidence level can also be chosen by the user.

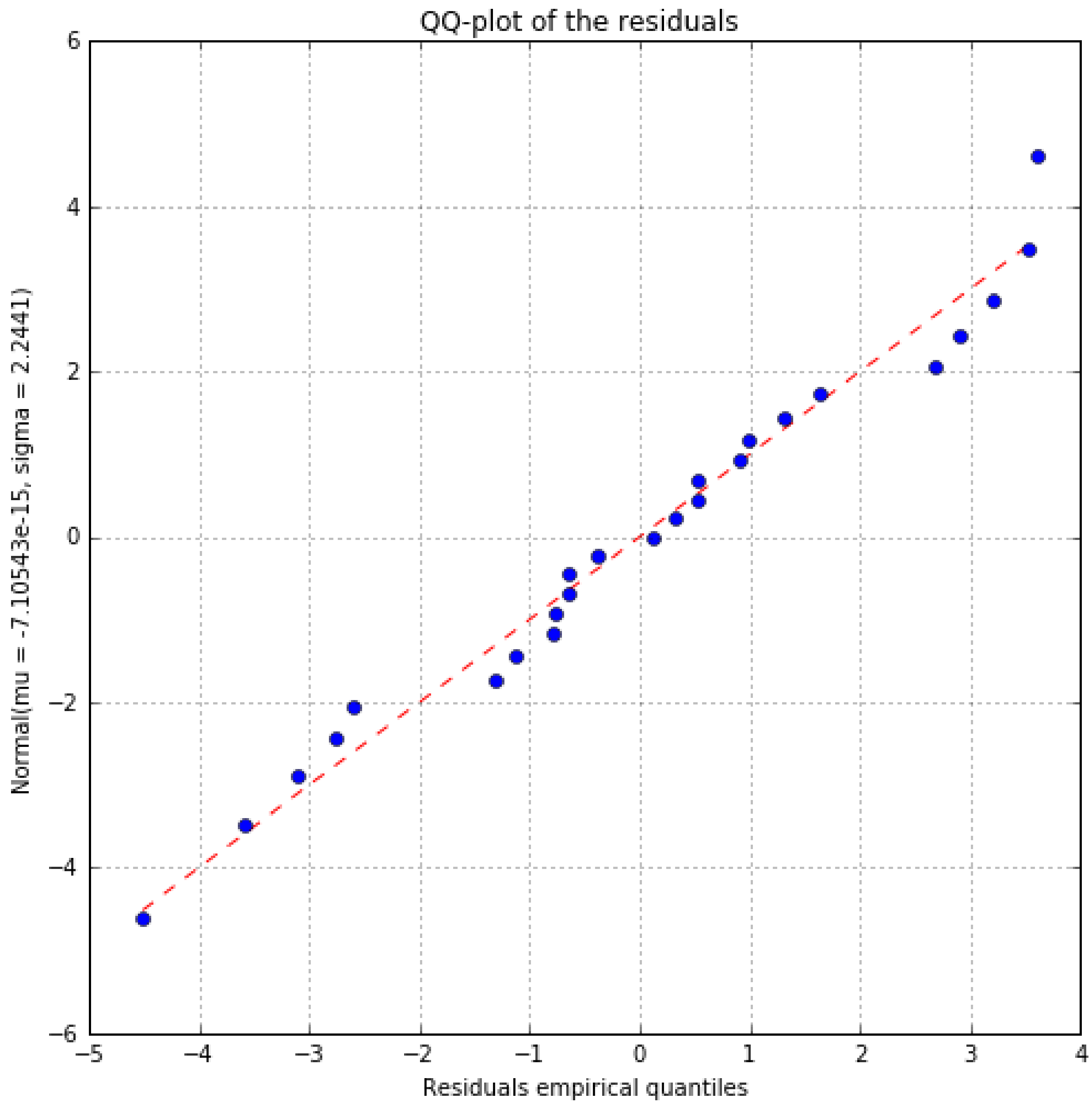
The warning is also printed here for the polynomial chaos because the detection size at the given probability level cannot be computed. A solution is to set *probabilityLevel* = *None*.

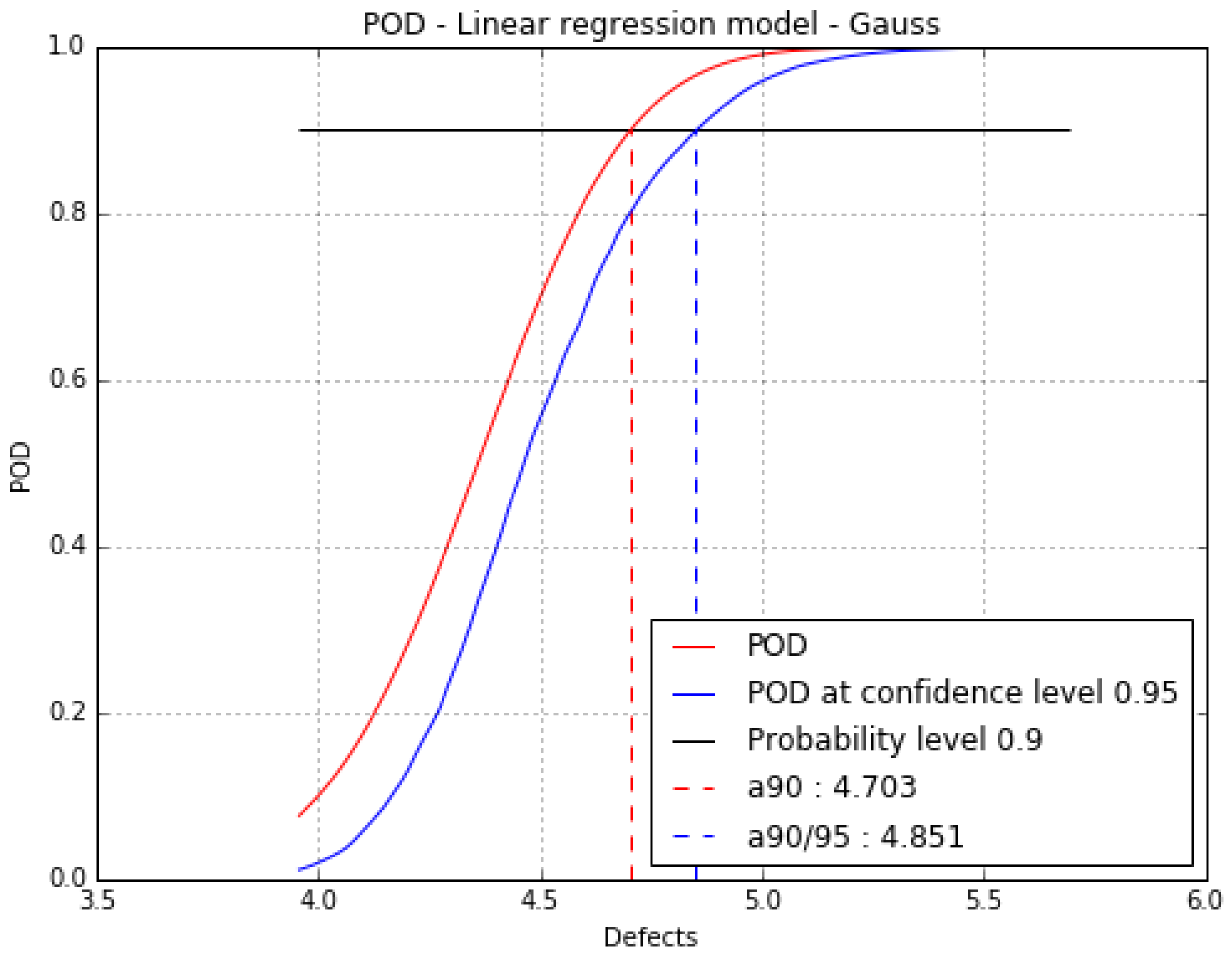
```
# return a list a figure
fig = POD.drawGraphs('./figure/', 'png', probabilityLevel=0.9, confidenceLevel=0.95)

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

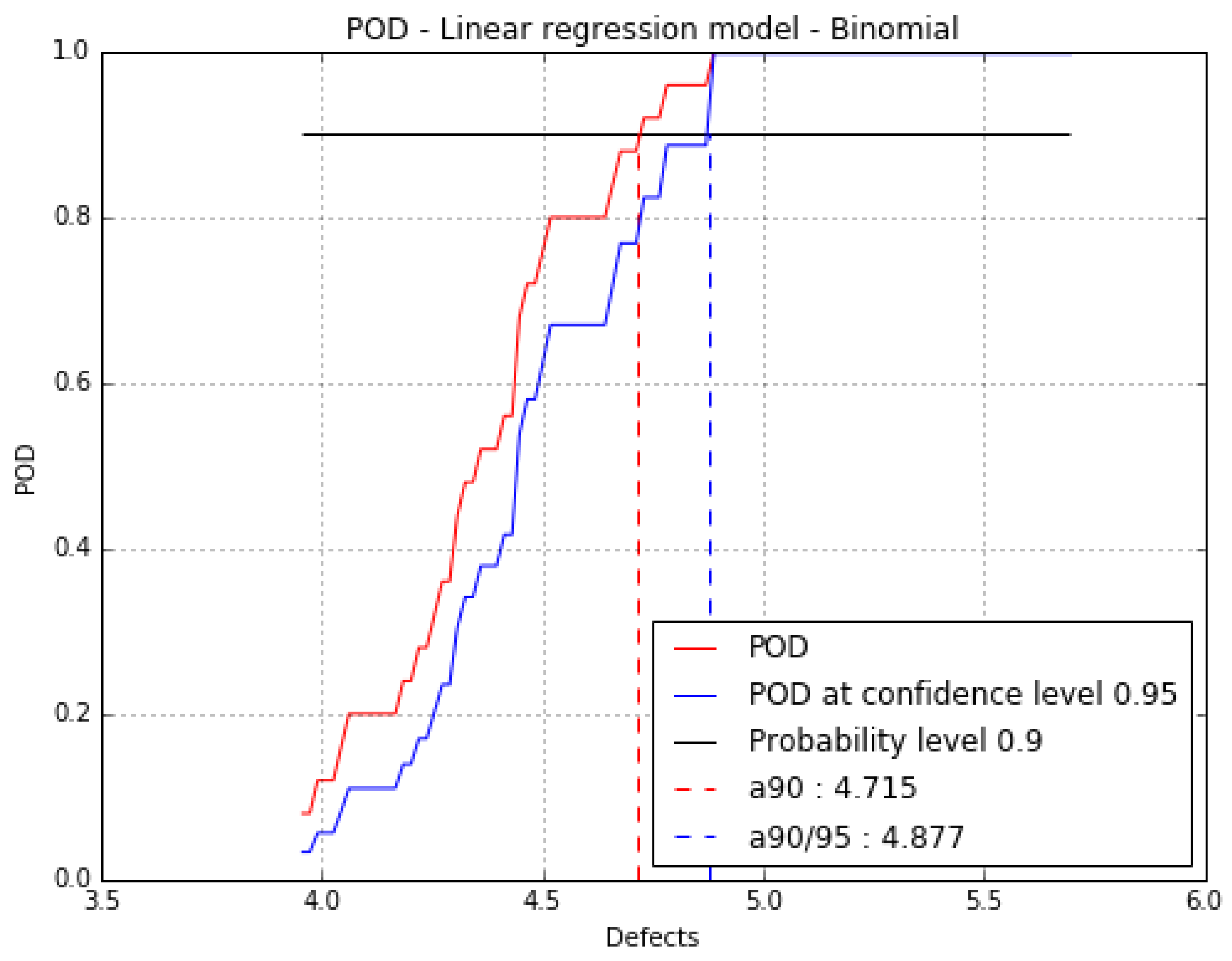

Linear regression model
Data
Linear model
Signals
Defects
50
45
40
35
30
3.5
4.0
4.5
5.0
5.5
6.0

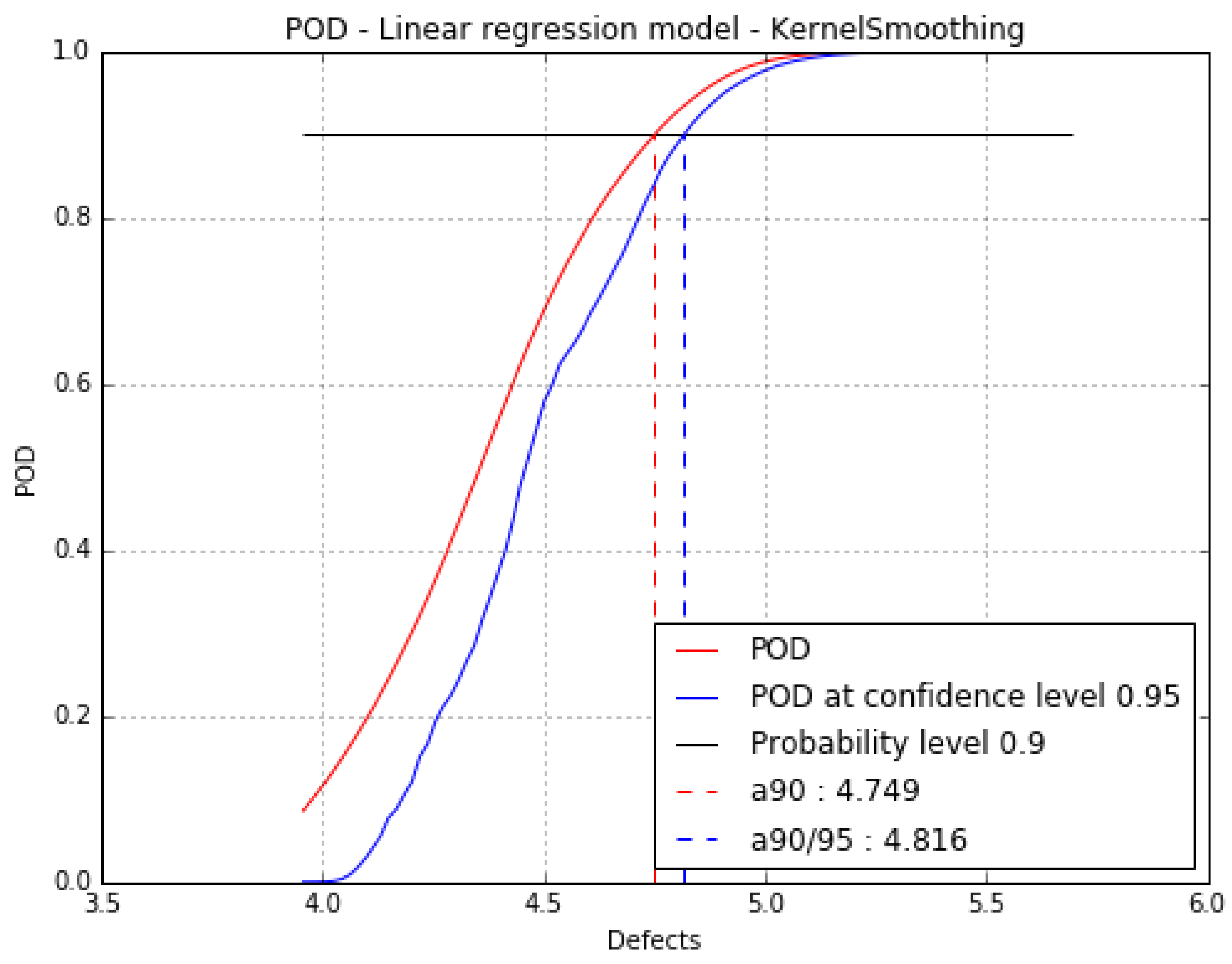

Residuals
Residuals dispersion
Defects
4
3
2
1
0
−1
−2
−3
−4
−5
3.5
4.0
4.5
5.0
5.5
6.0

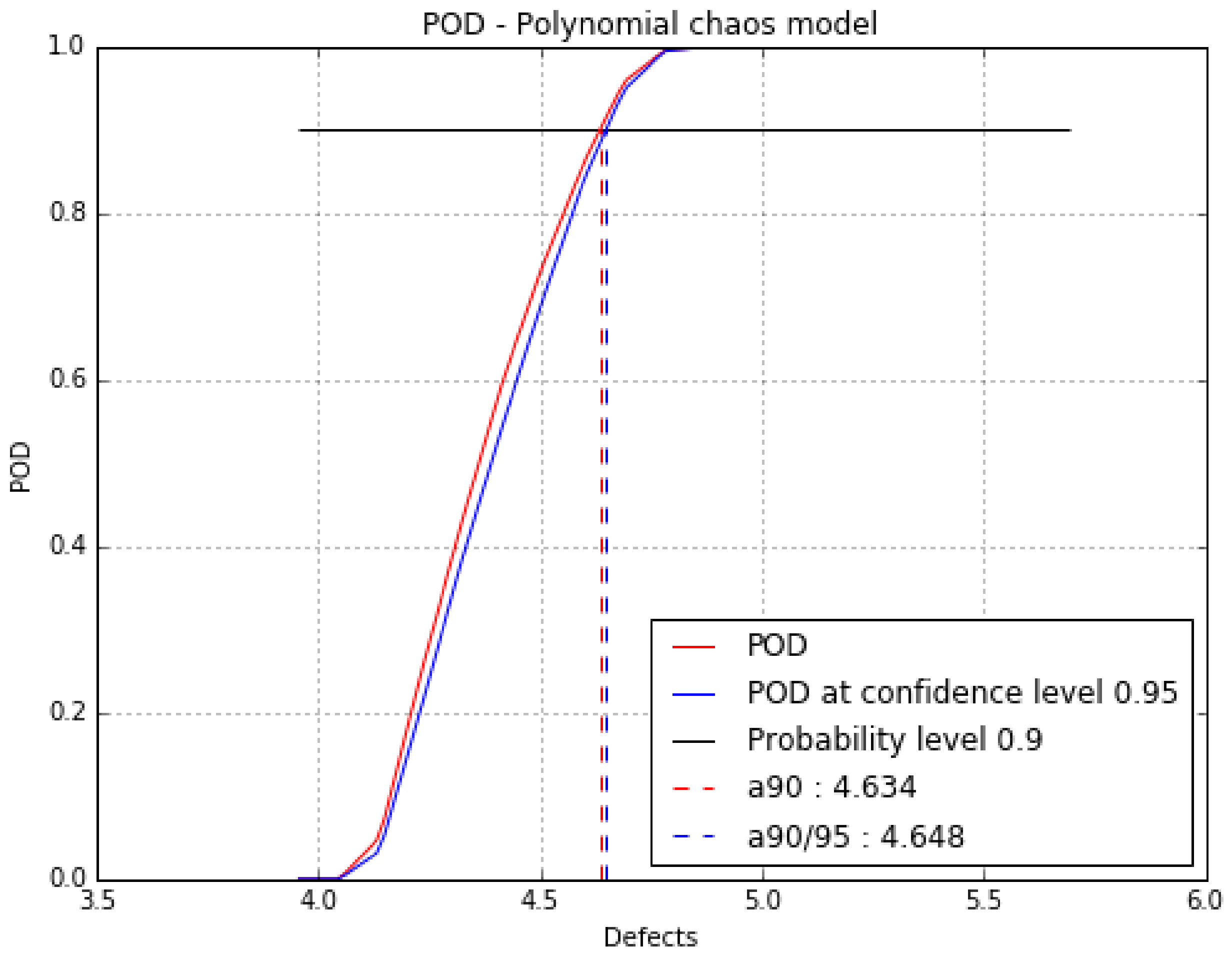

Residuals distribution
Normal(mu = -7.10543e-15, sigma = 2.2441)
Residuals histogram
PDF
Defect realizations
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
−10
−5
0
5
10

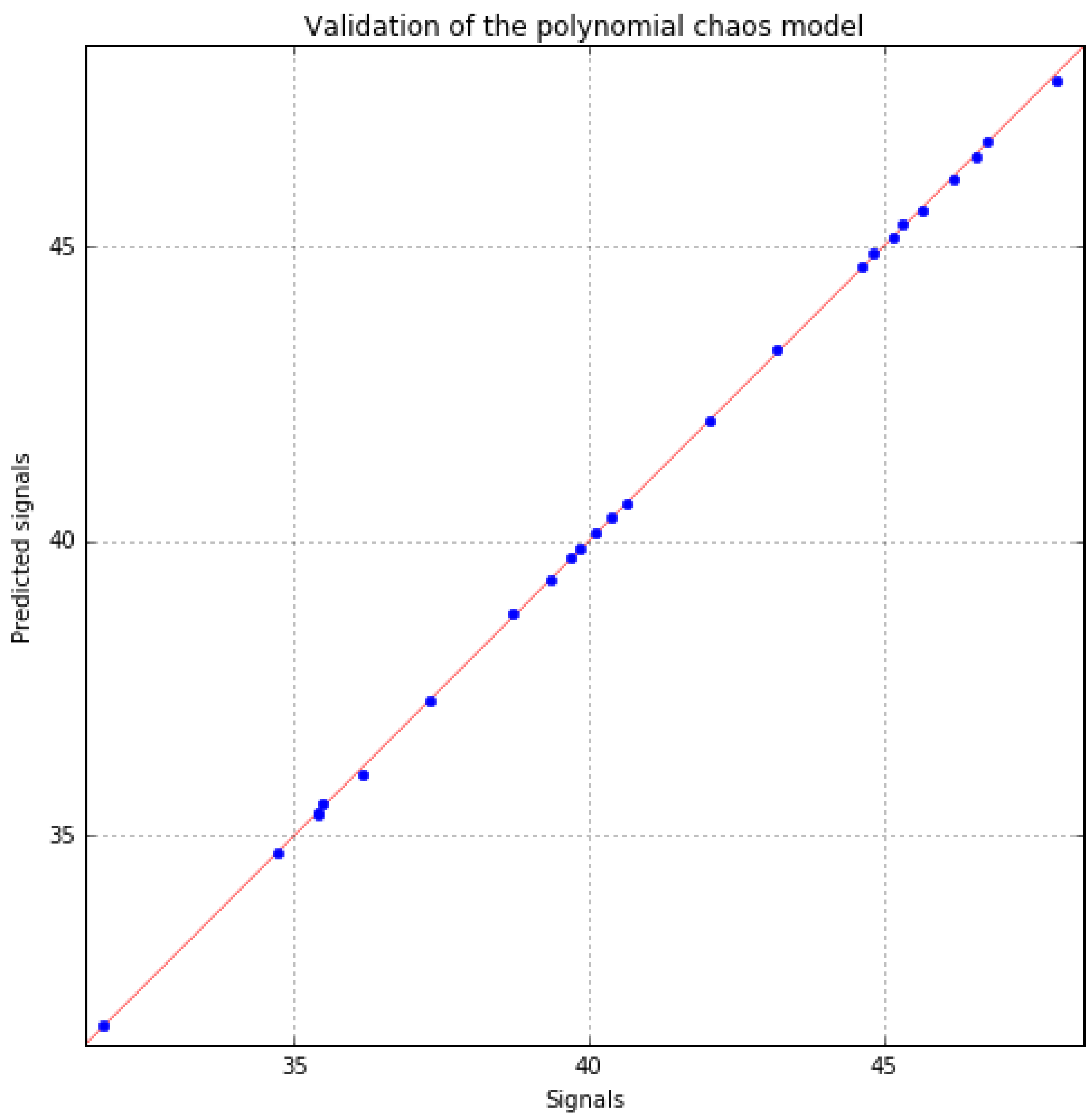

QQ-plot of the residuals
Normal(mu = -7.10543e-15, sigma = 2.2441)
Residuals empirical quantiles
6
4
2
0
−2
−4
−6
−5
−4
−3
−2
−1
0
1
2
3
4

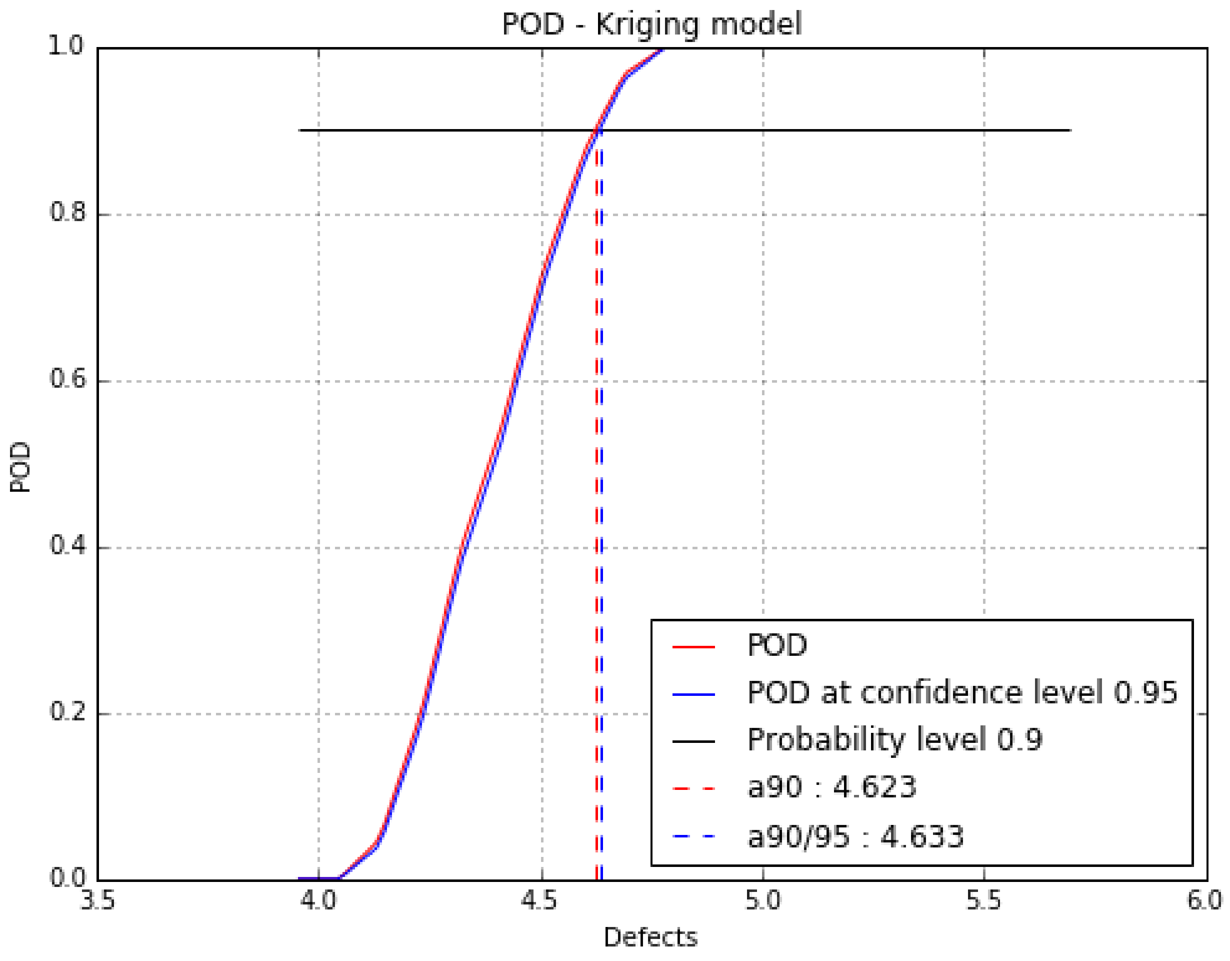

POD - Linear regression model - Gauss
1.0
0.8
0.6
0.4
0.2
0.0
POD
3.5
4.0
4.5
5.0
5.5
6.0
Defects
POD
POD at confidence level 0.95
Probability level 0.9
a90 : 4.703
a90/95 : 4.851

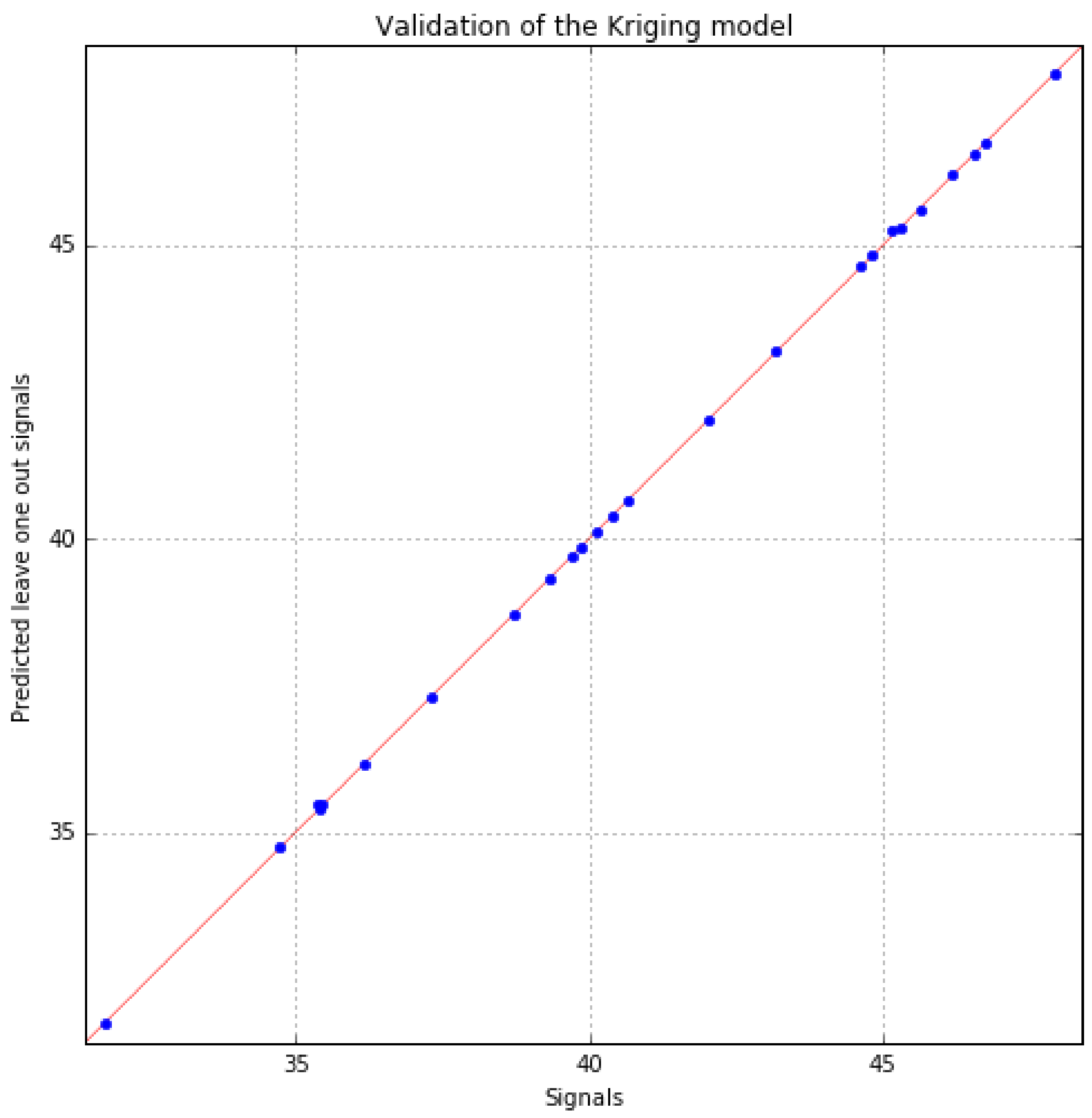

POD - Linear regression model - Binomial
POD
Defects
1.0
0.8
0.6
0.4
0.2
0.0
3.5
4.0
4.5
5.0
5.5
6.0
POD
POD at confidence level 0.95
Probability level 0.9
a90 : 4.715
a90/95 : 4.877


POD - Linear regression model - KernelSmoothing
POD
Defects
1.0
0.8
0.6
0.4
0.2
0.0
3.5
4.0
4.5
5.0
5.5
6.0
POD
POD at confidence level 0.95
Probability level 0.9
a90 : 4.749
a90/95 : 4.816


POD - Polynomial chaos model
POD
Defects
1.0
0.8
0.6
0.4
0.2
0.0
3.5
4.0
4.5
5.0
5.5
6.0
POD
POD at confidence level 0.95
Probability level 0.9
a90 : 4.634
a90/95 : 4.648


Validation of the polynomial chaos model
Predicted signals
Signals
35
40
45
35
40
45


POD - Kriging model
POD
Defects
1.0
0.8
0.6
0.4
0.2
0.0
3.5
4.0
4.5
5.0
5.5
6.0
POD
POD at confidence level 0.95
Probability level 0.9
a90 : 4.623
a90/95 : 4.633



ipynb source code

1.2.9 Adaptive Signal POD using Kriging

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

Generate data

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

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.609524],
     [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469 ],
     [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.768369],
     [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.636588],
     [ 40.643948]])

# detection threshold
detection = 38

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

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

Create the Adaptive Signal POD with Kriging model

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

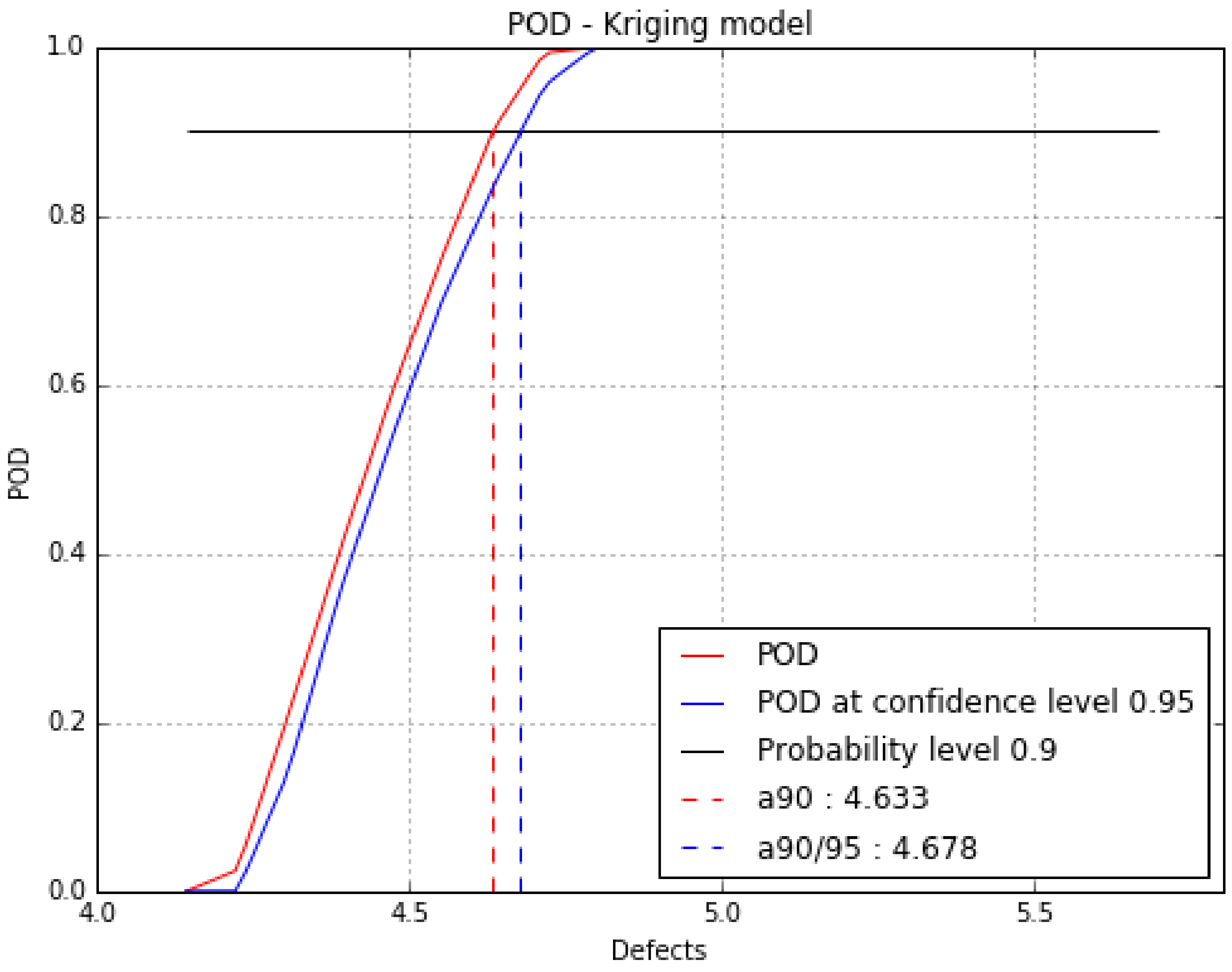
As this method is time consuming, it is more efficient to reduce the area of the defect size only in the most interesting part. To do that, an initial POD study can be run.

Run an initial POD study with the kriging technique

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

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

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



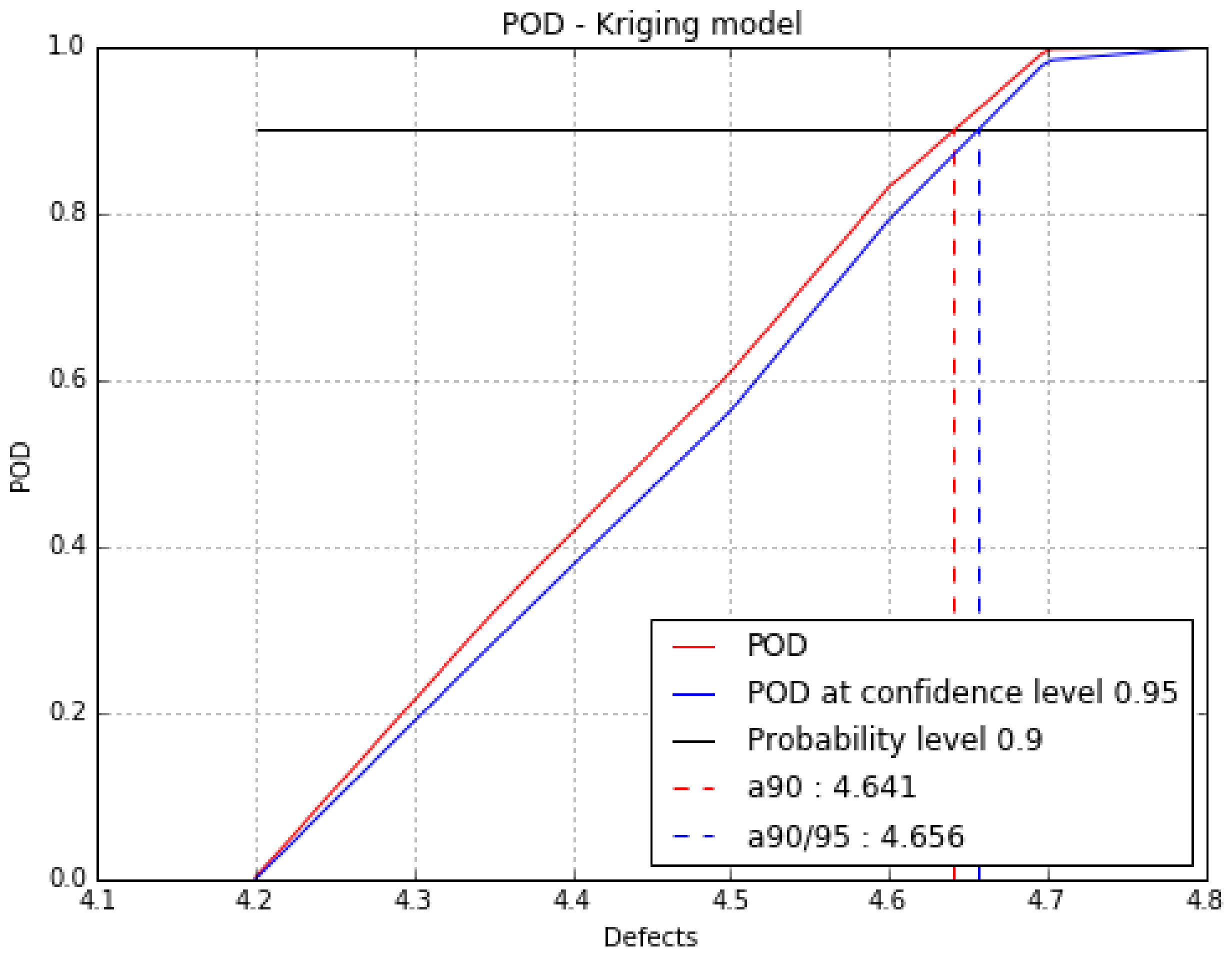
Based on this study, the interesting part for the defects ranges from 4.2 to 4.8. The adaptive signal algorithm will be then reduced to this area.

Run the adaptive algorithm

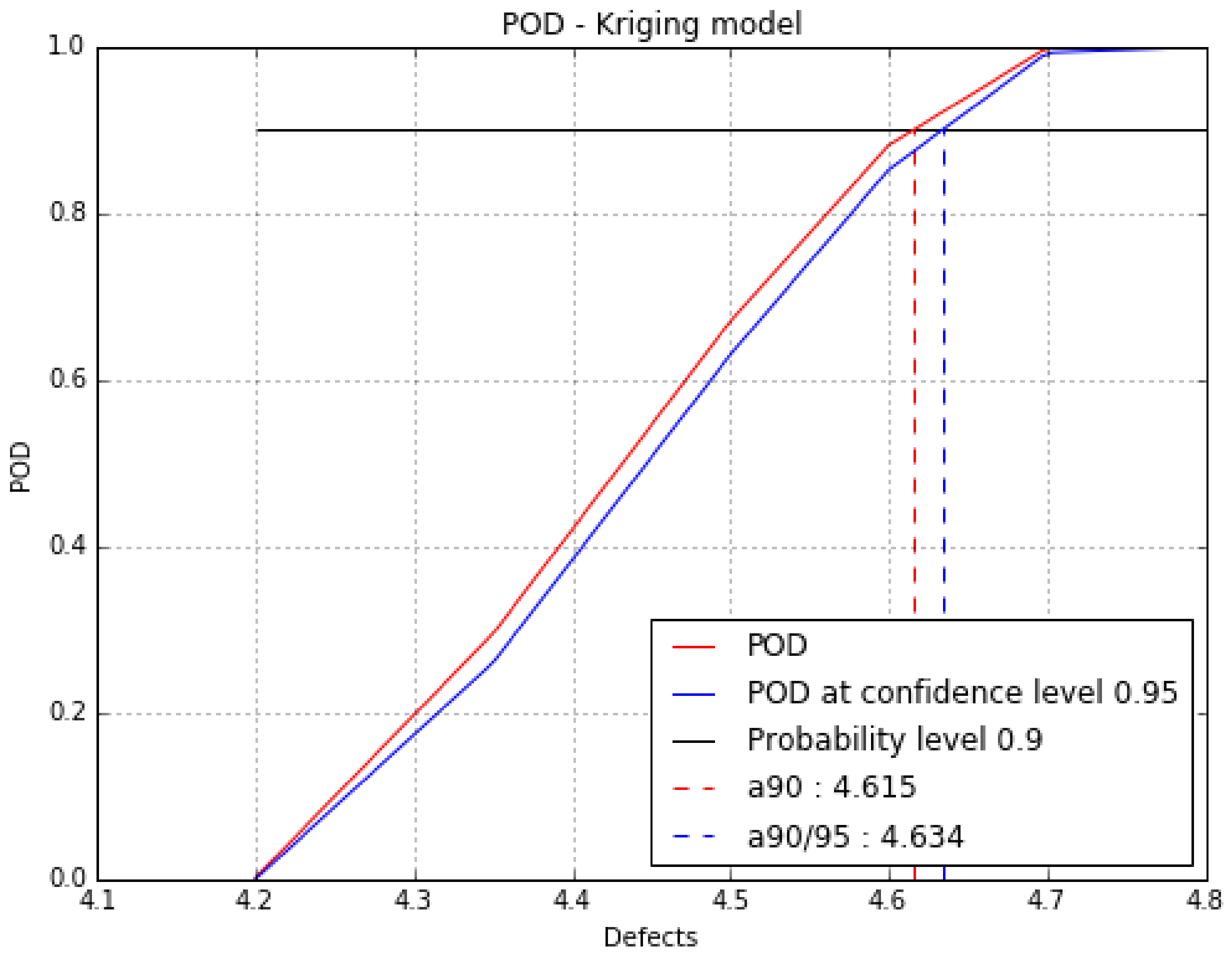
Computing the criterion is costly so the sampling and simulation size are reduced.

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

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

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

# We can change also the number of candidate points for which the critertion is computed
adaptivePOD.setCandidateSize(1000)
# we can change the sample size of the Monte Carlo simulation
adaptivePOD.setSamplingSize(500) # default is 5000
# we can also change the size of the simulation to compute the confidence interval
adaptivePOD.setSimulationSize(100) # default is 1000

# The current iteration POD graph can be displayed with multiple options :
## with or without the confidence level curve
## and with or without the intersection value at the given probability level
## if a directory is given, all graphs are saved as AdaptiveSignalPOD_i.png
adaptivePOD.setGraphActive(graphVerbose=True, probabilityLevel=0.9, confidenceLevel=0.95,
                           directory='figure/')

%time adaptivePOD.run()
```

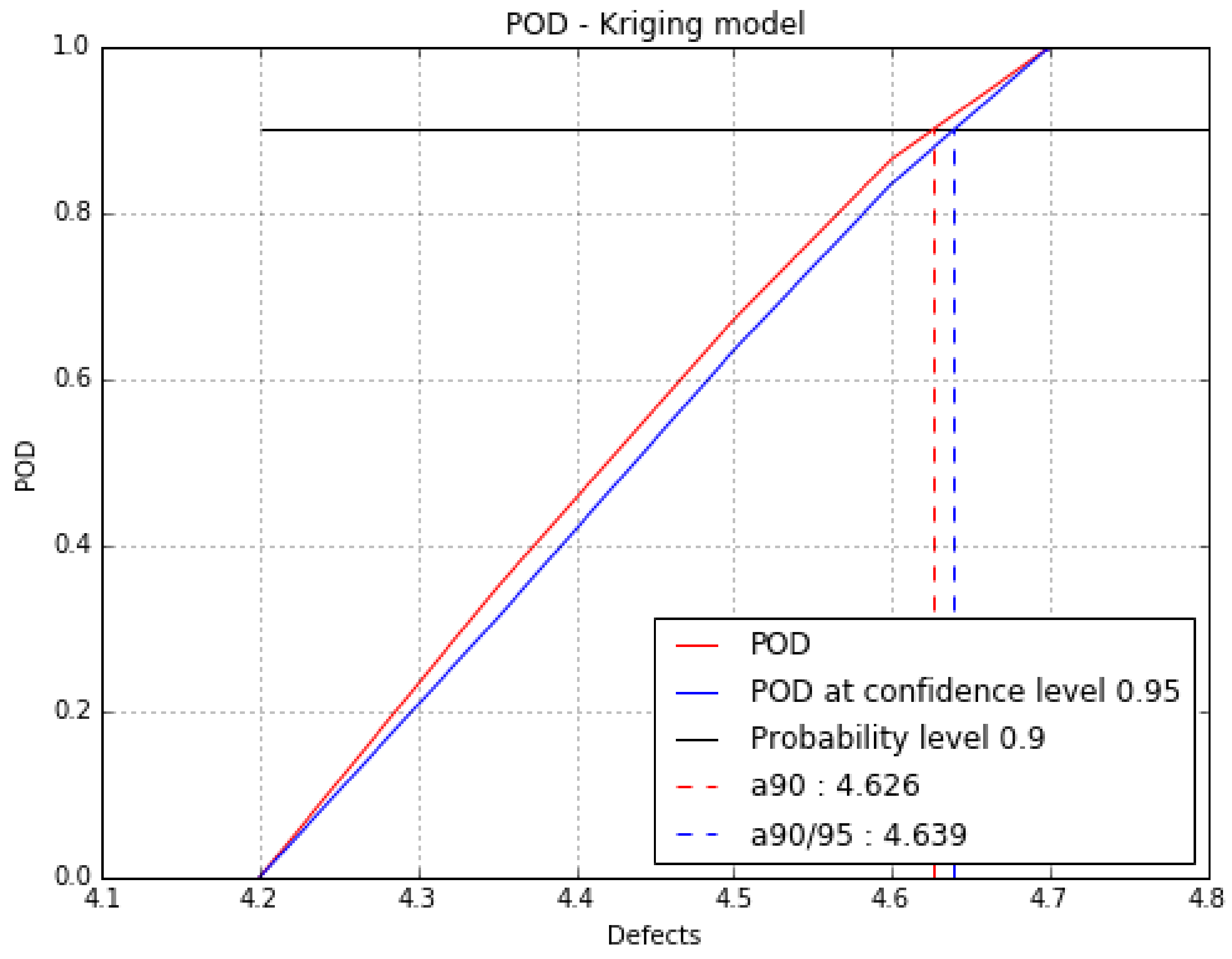
```
Build kriging model
Optimization of the covariance model parameters...
Kriging validation Q2 (>0.9): 0.9425

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



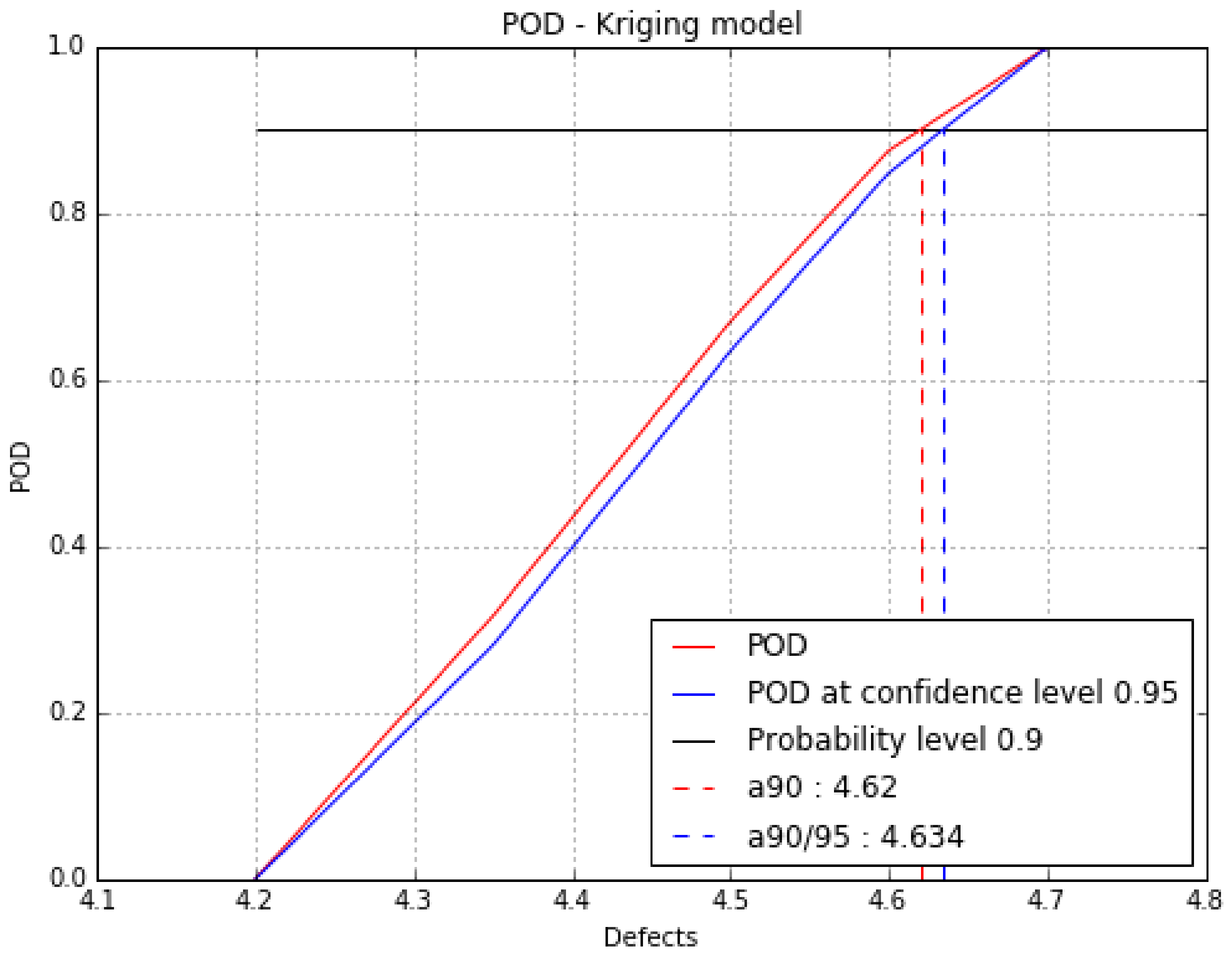
```
Iteration : 2/5
Computing criterion: [==================================================] 100.00% Done
Criterion value : 0.0158
Added point : [4.3998,0.0651376,1.06941,29.9332]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9954
```



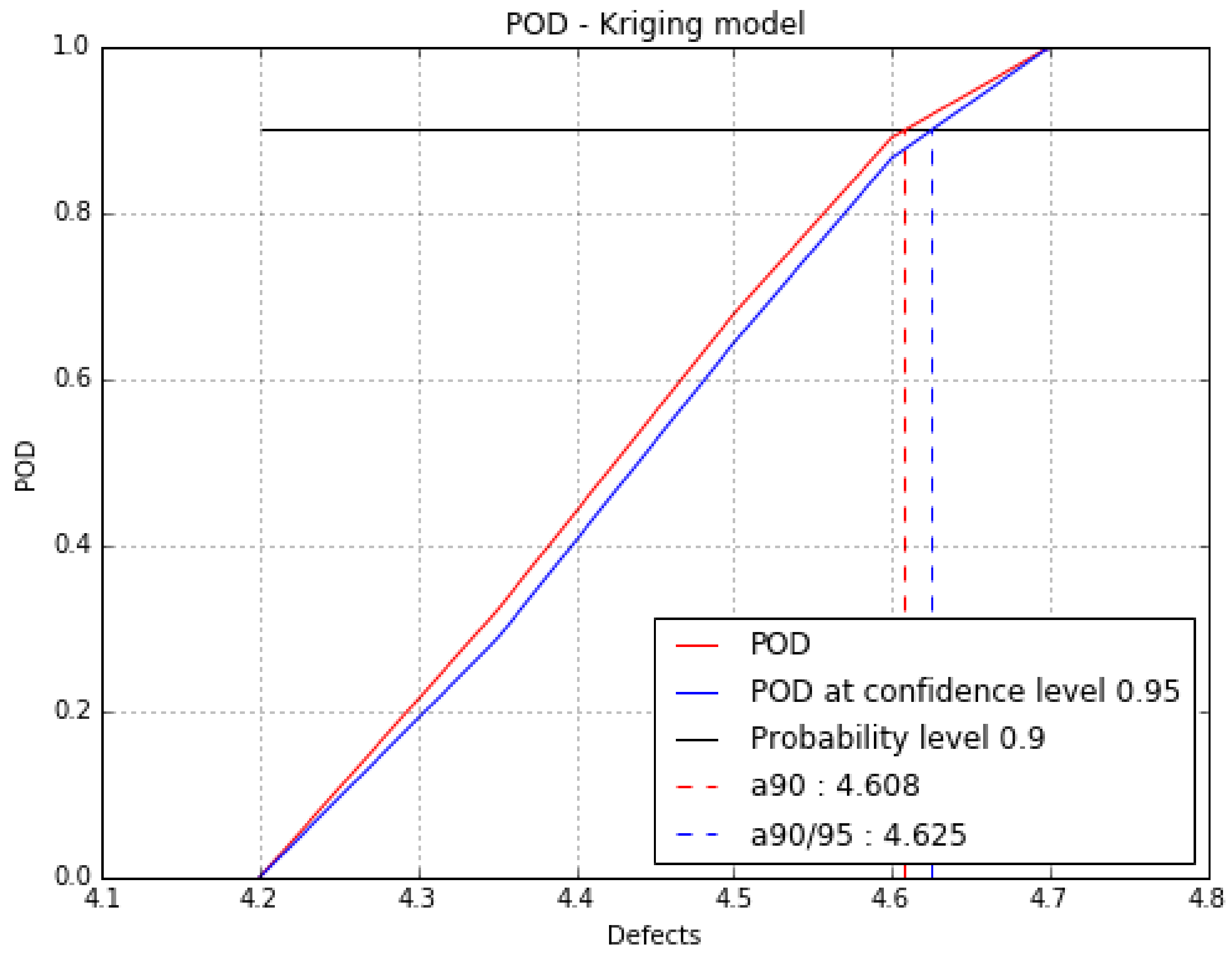
```
Iteration : 3/5
Computing criterion: [==================================================] 100.00% Done
Criterion value : 0.0157
Added point : [4.3582,0.0294083,1.0403,79.0659]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9925
```



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



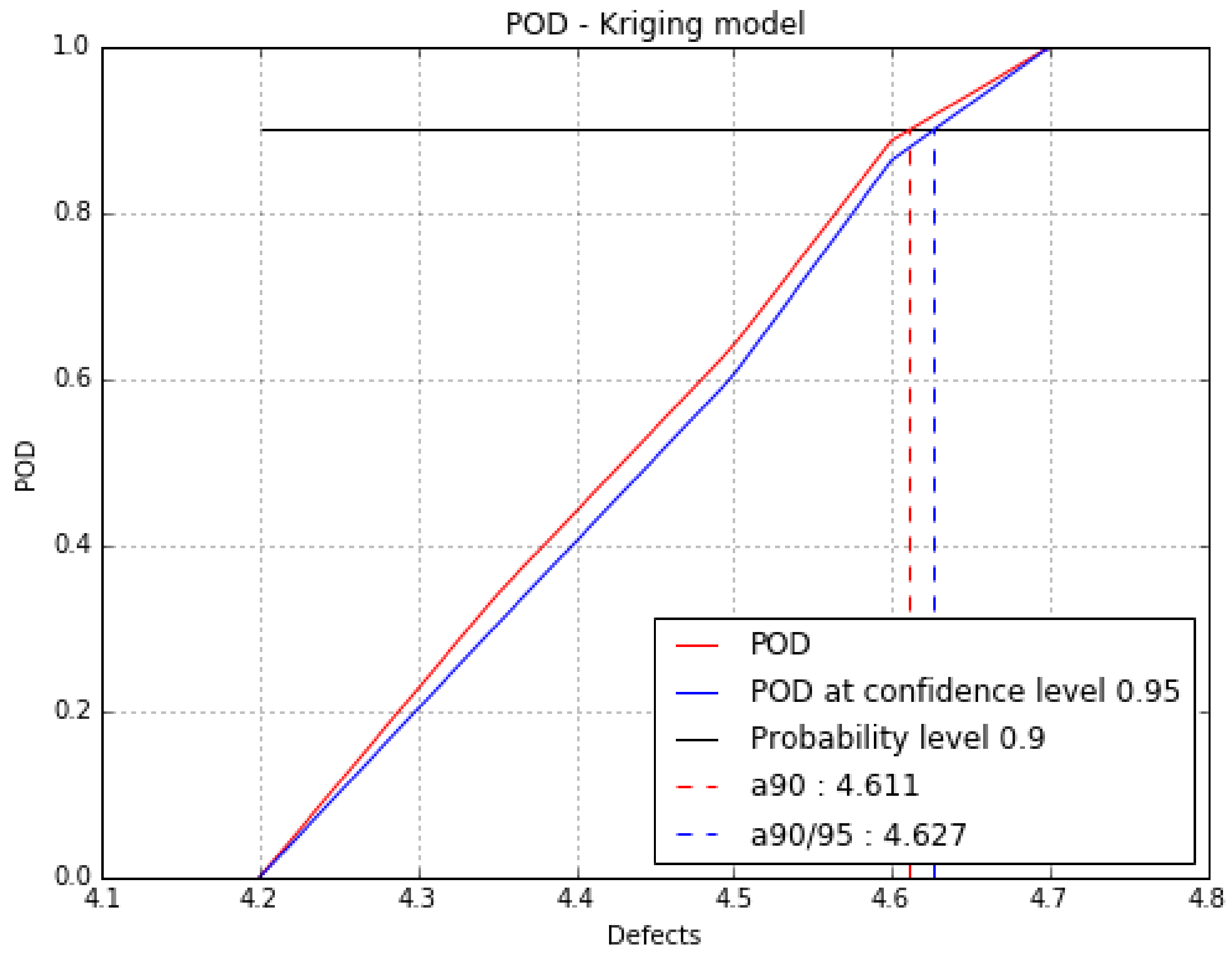
```
Iteration : 5/5
Computing criterion: [==================================================] 100.00% Done
Criterion value : 0.0139
Added point : [4.66465,0.0723142,1.02347,40.5827]
Update the kriging model
Kriging validation Q2 (>0.9): 0.9882
```



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

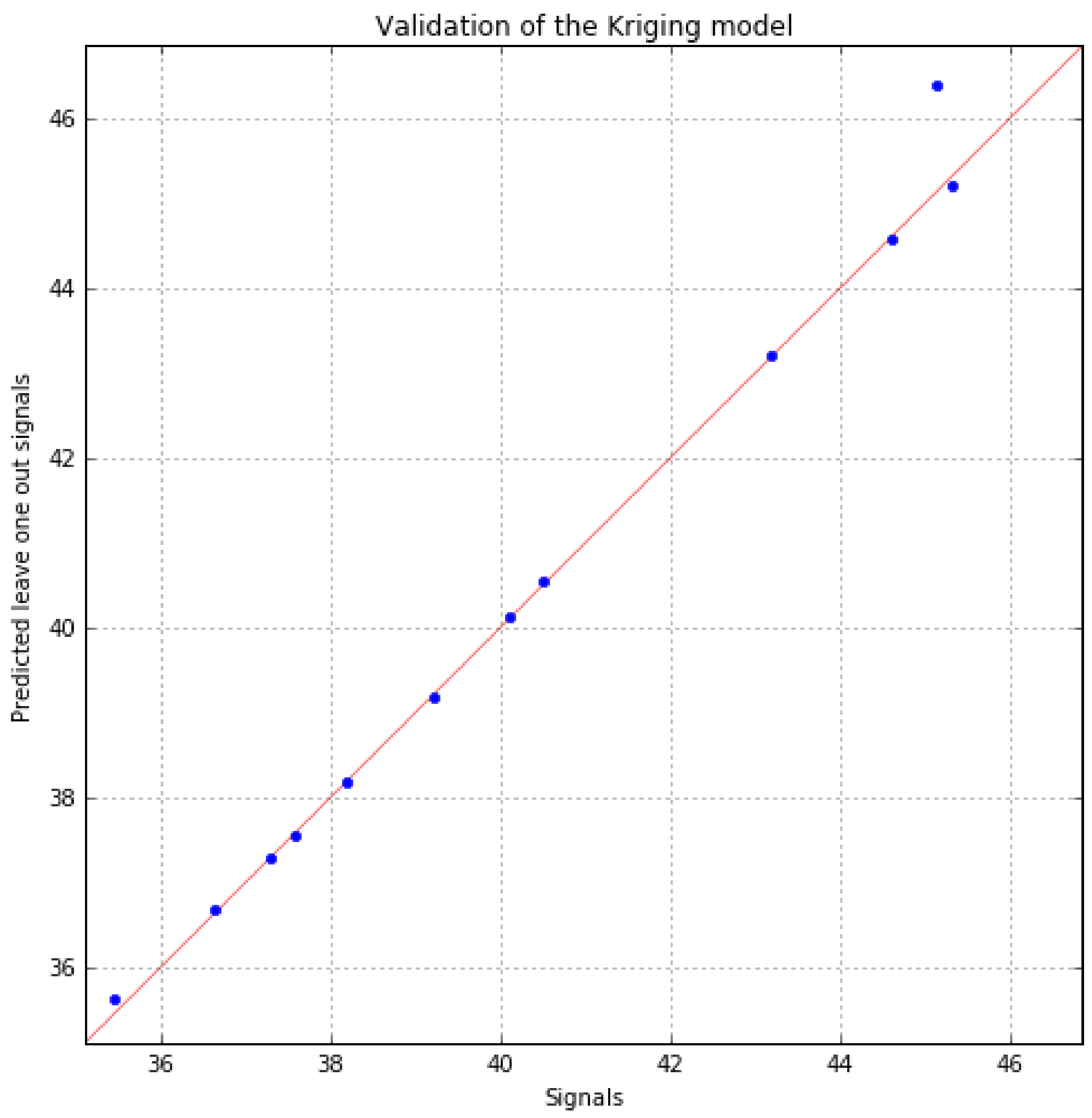
Display the POD result based on the adative kriging model

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



Diplay the validation graph

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



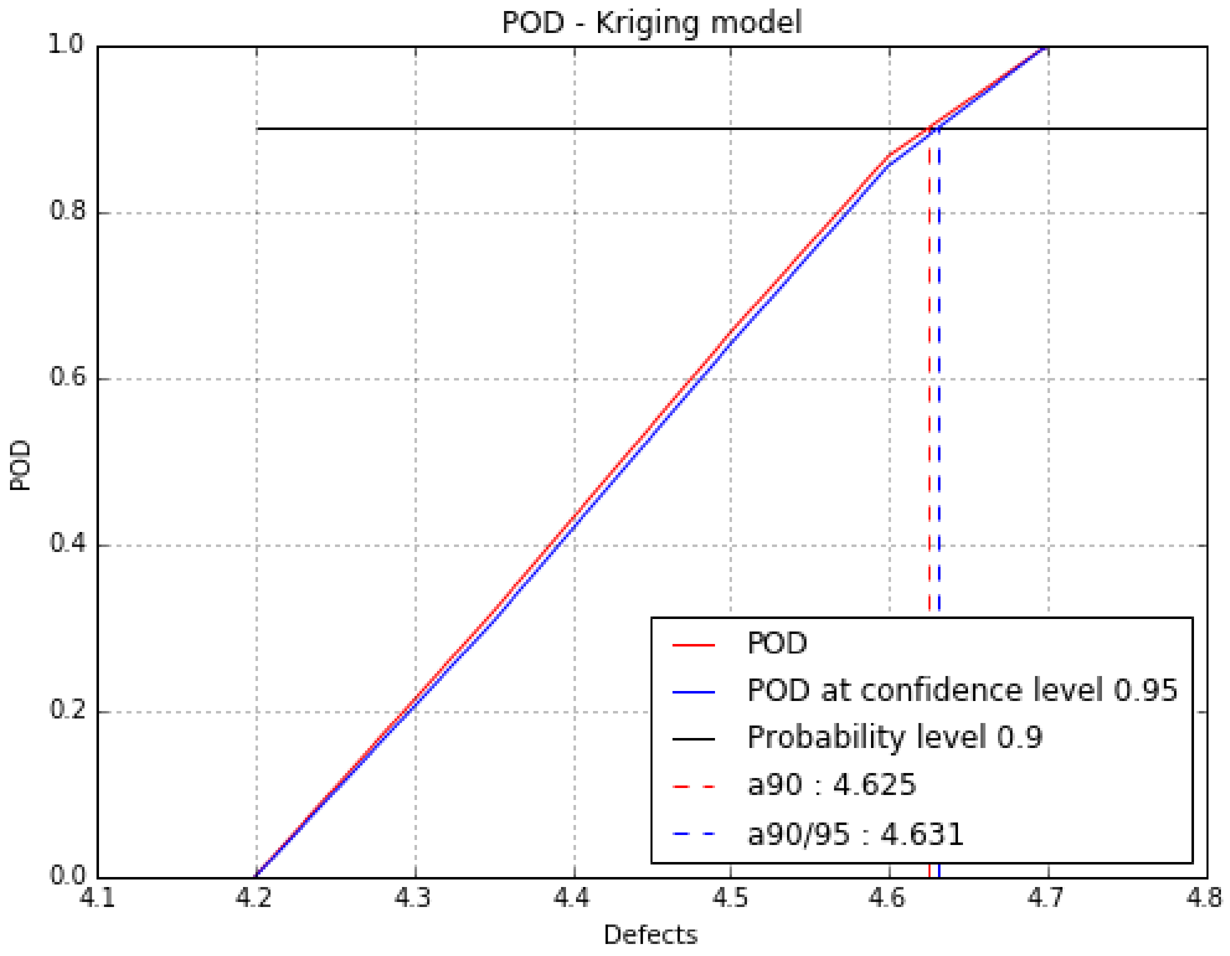
Quality improvement of the POD computation From the adaptive algorithm, the kriging result and the final DOE are available. As the number of simulations were reduced, we can compute again the POD with more accuracy than before if needed.

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

# A new POD study is launch with the DOE values
finalPOD = otpod.KrigingPOD(inputfinal, outputfinal, detection)
finalPOD.setDefectSizes(defectSizes)
```

```
# The kriging model is already known so it is given to this study
finalPOD.setKrigingResult(krigingRes)
finalPOD.run()
```

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

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



ipynb source code

1.2.10 Adaptive Hit Miss POD

```
%matplotlib inline
import numpy as np
import openturns as ot
import otpod
import warnings
warnings.filterwarnings("ignore")
```

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

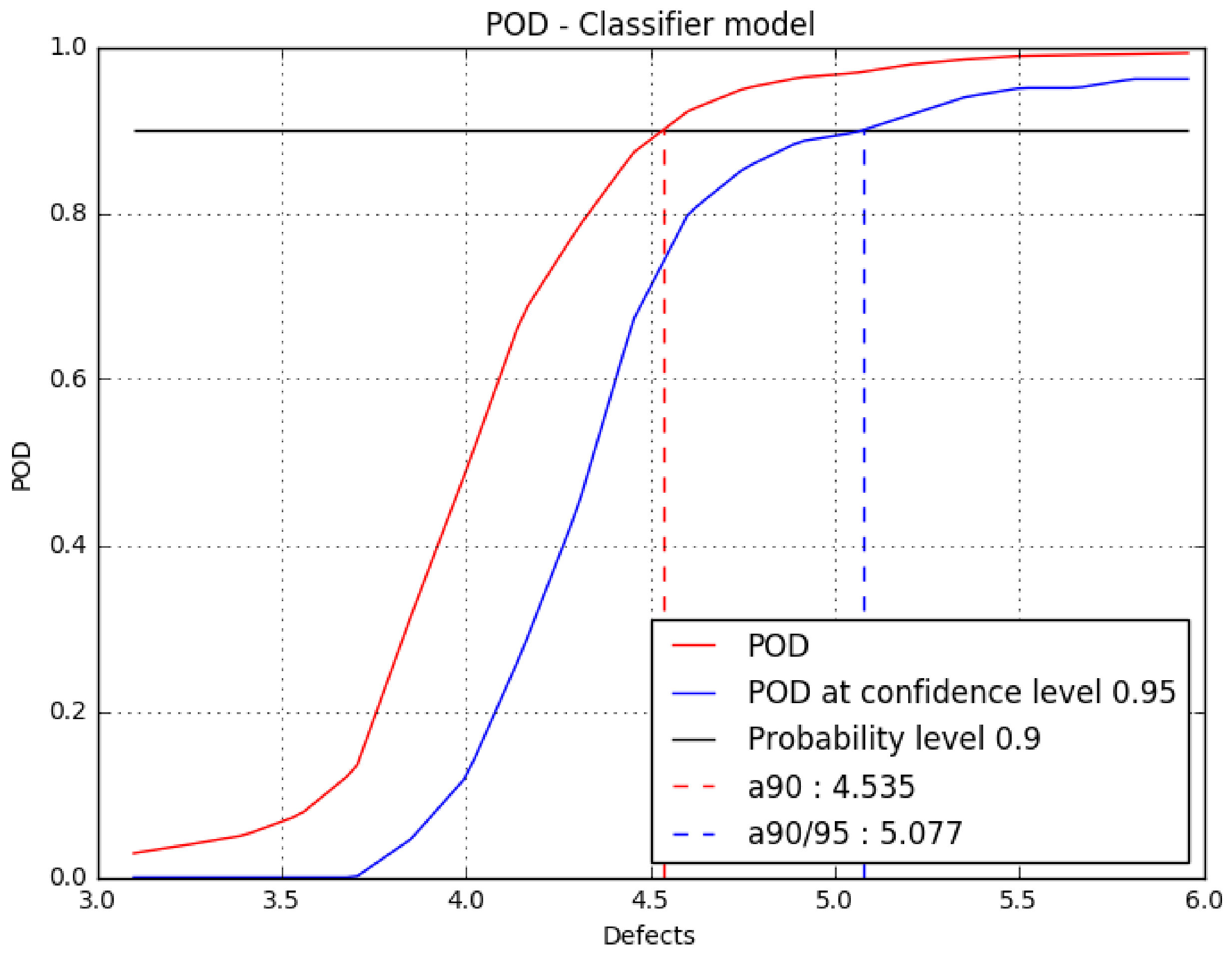
```
execfile("Make_HM.py")
```

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

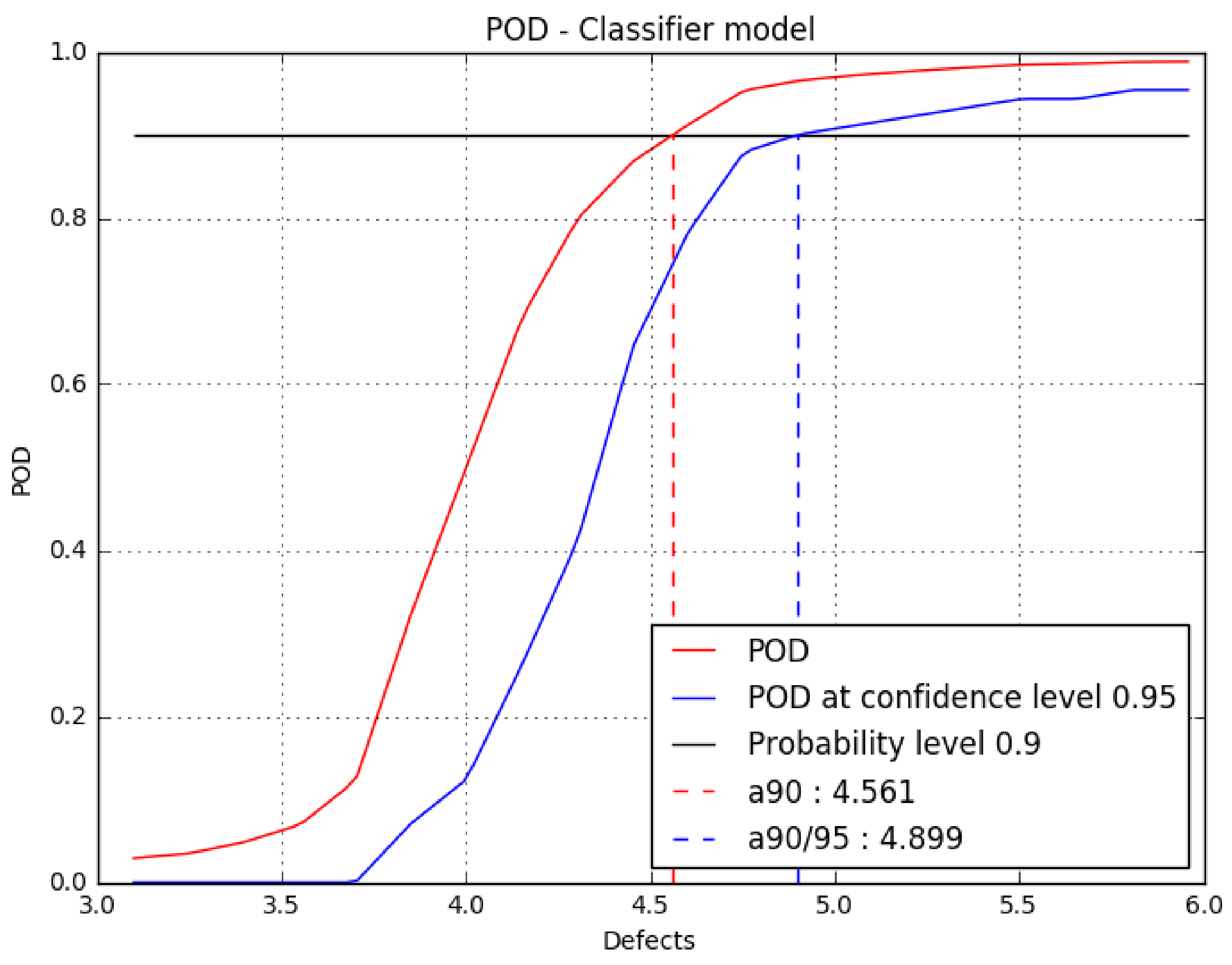
```
n_ini = 100
inputDOE = np.array([np.random.uniform(x_min[i],
             x_max[i],n_ini)
             for i in range(d)]).T
outputDOE = MyHM(inputDOE)
```

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

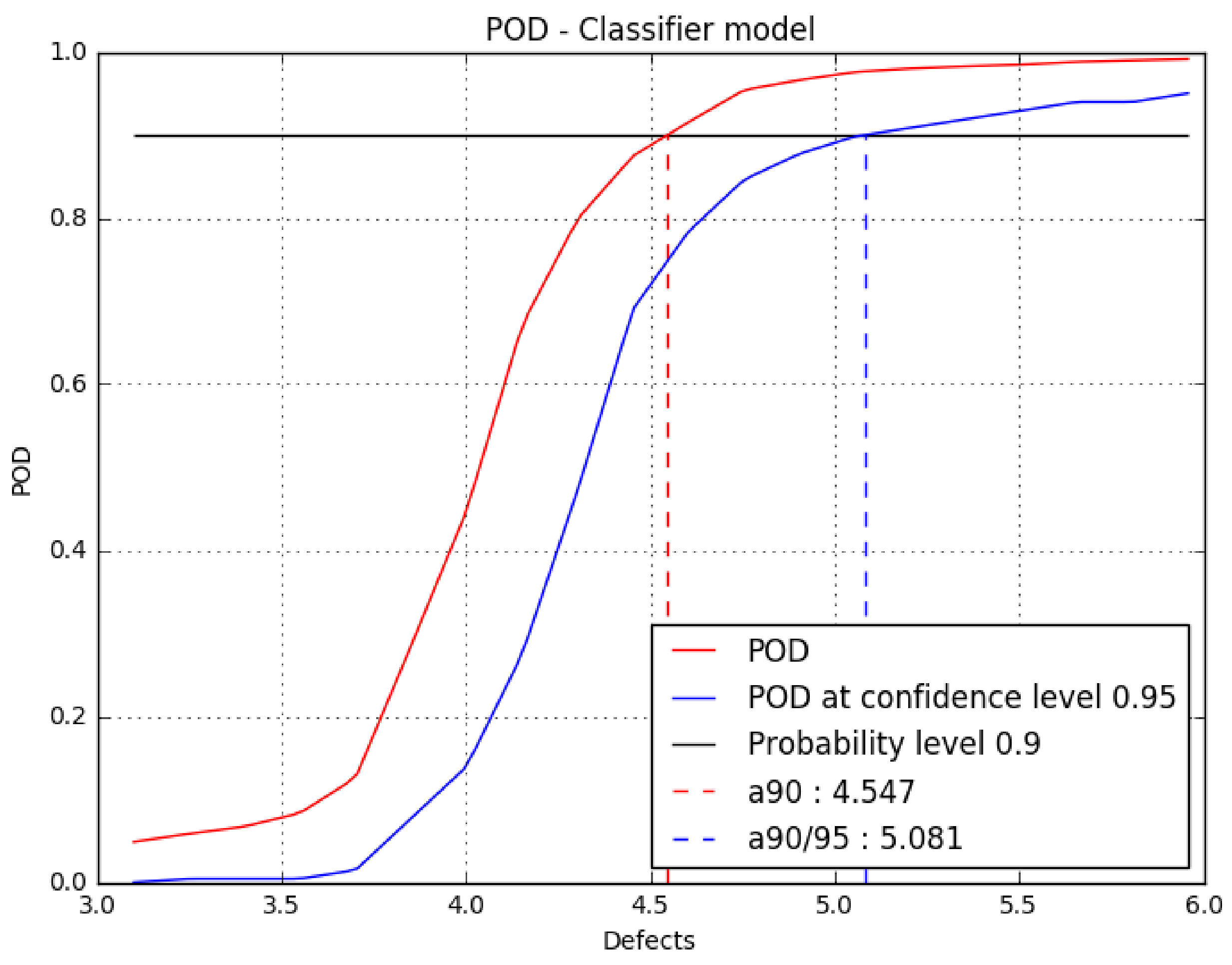
```
Building the classifier
Start the improvement loop
Adding points: [========-----------------------------------------] 16.67%
```



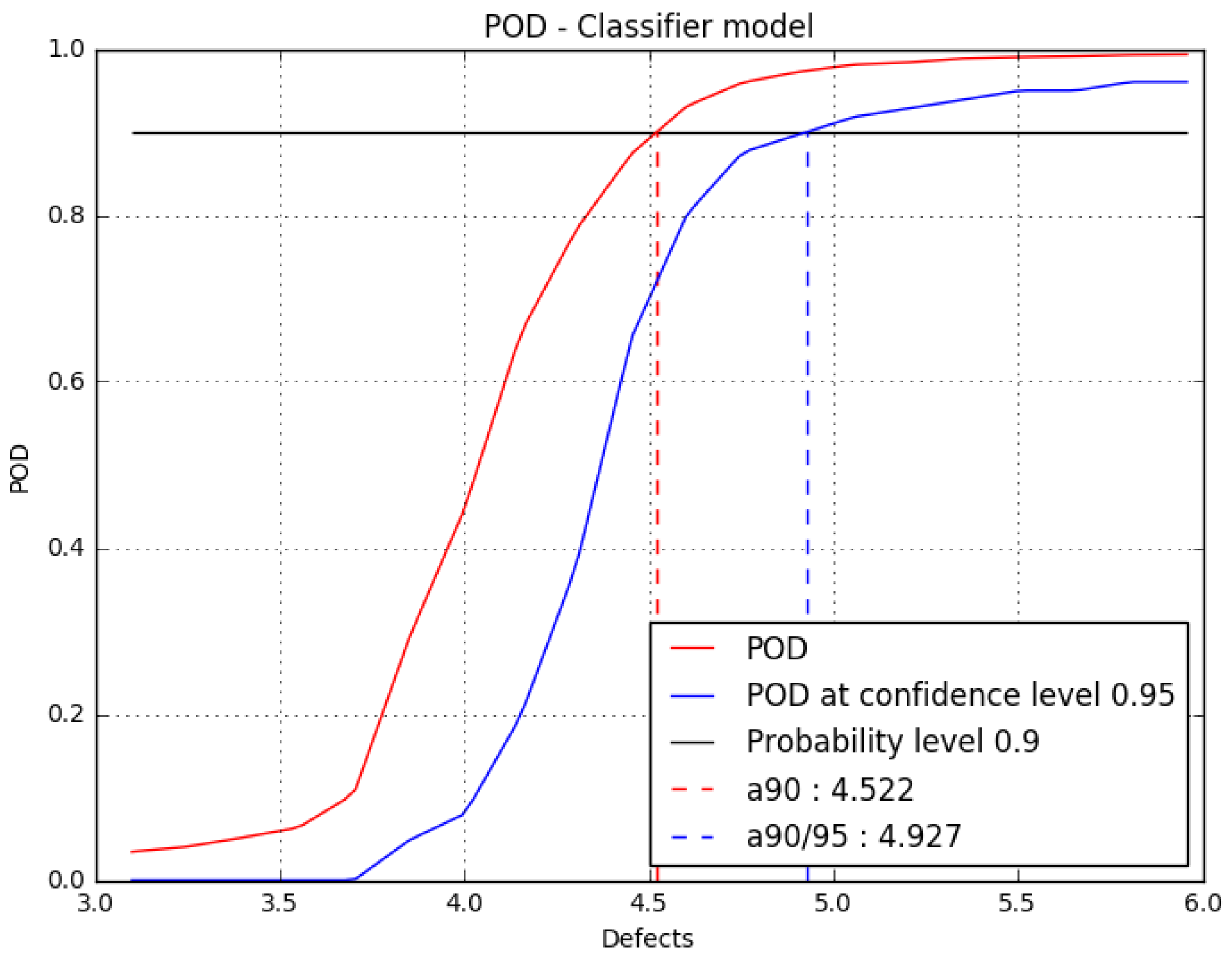
```
Adding points: [================--------------------------------] 33.33%
```



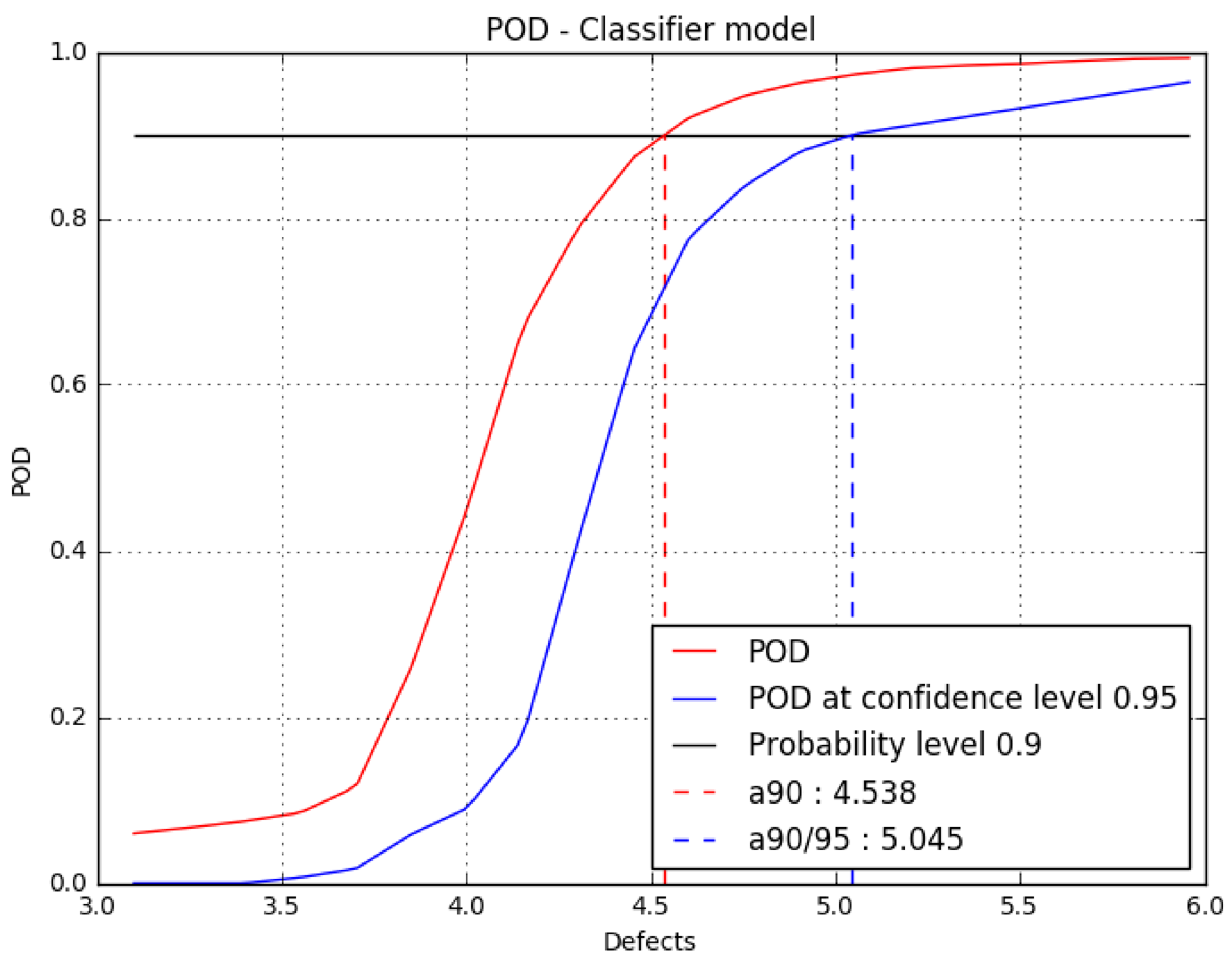
```
Adding points: [========================-------------------------] 50.00%
```



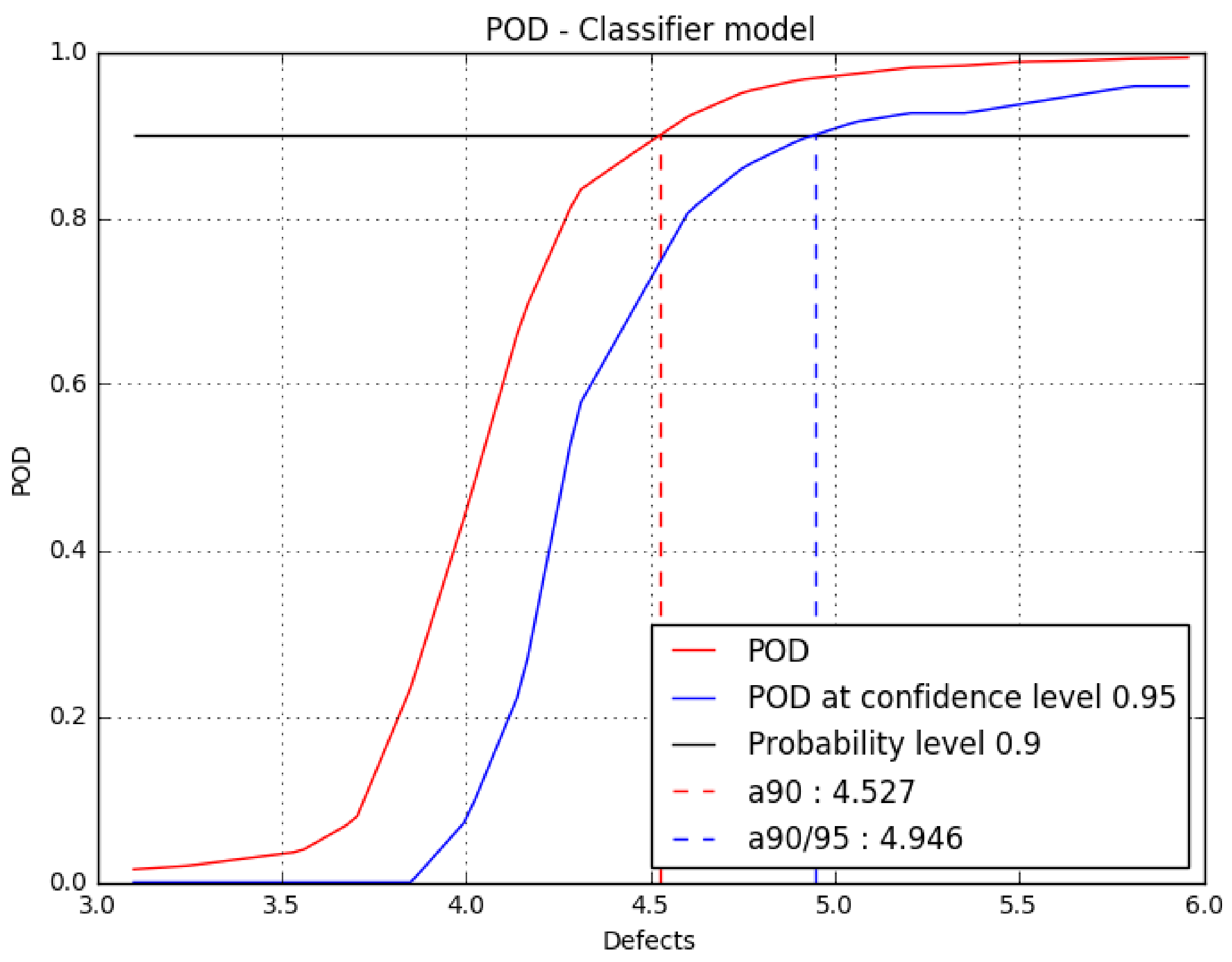
```
Adding points: [=================================================-----------------] 66.67%
```



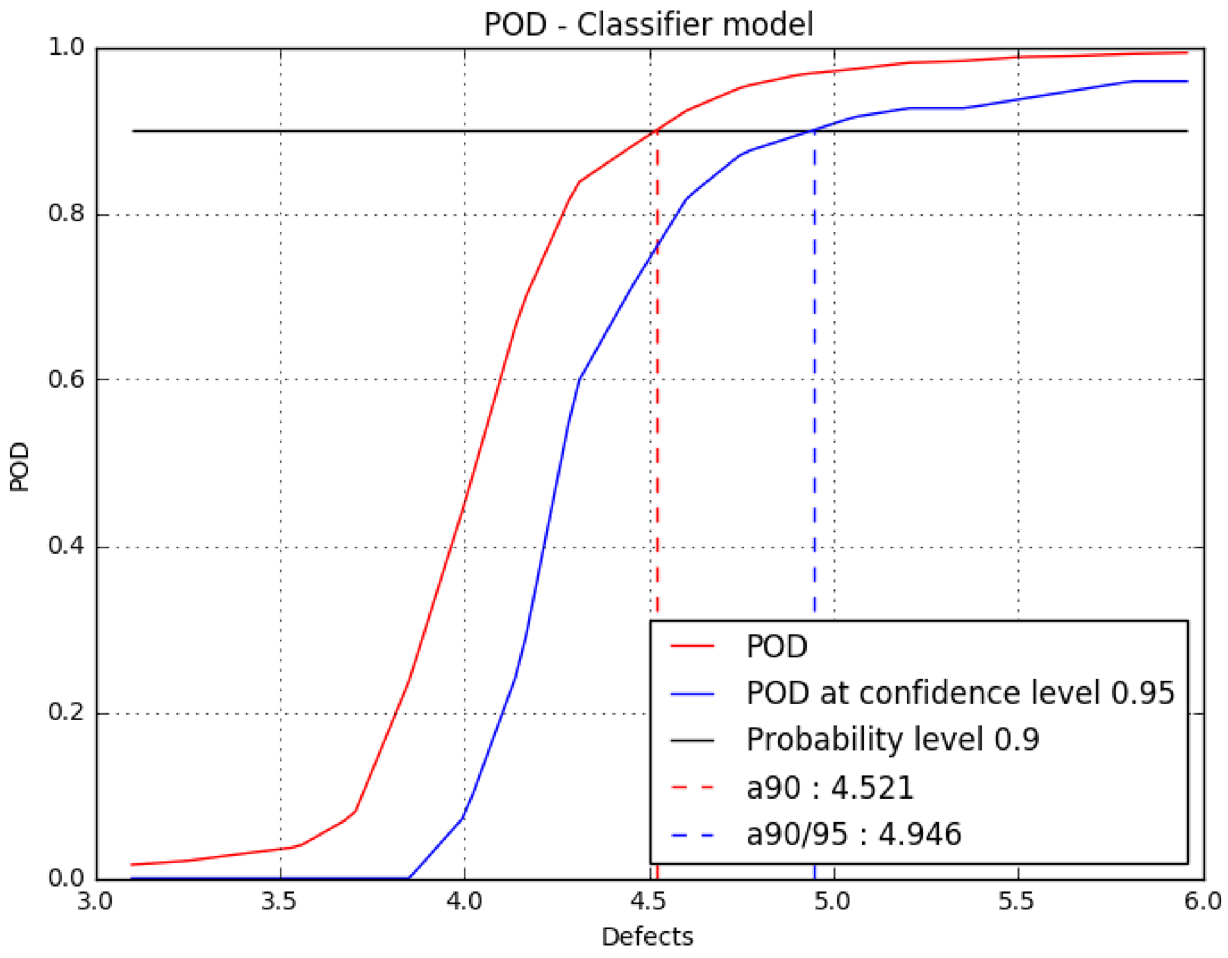
```
Adding points: [==========================================--------] 83.33%
```



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



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



1.2.11 Signal case

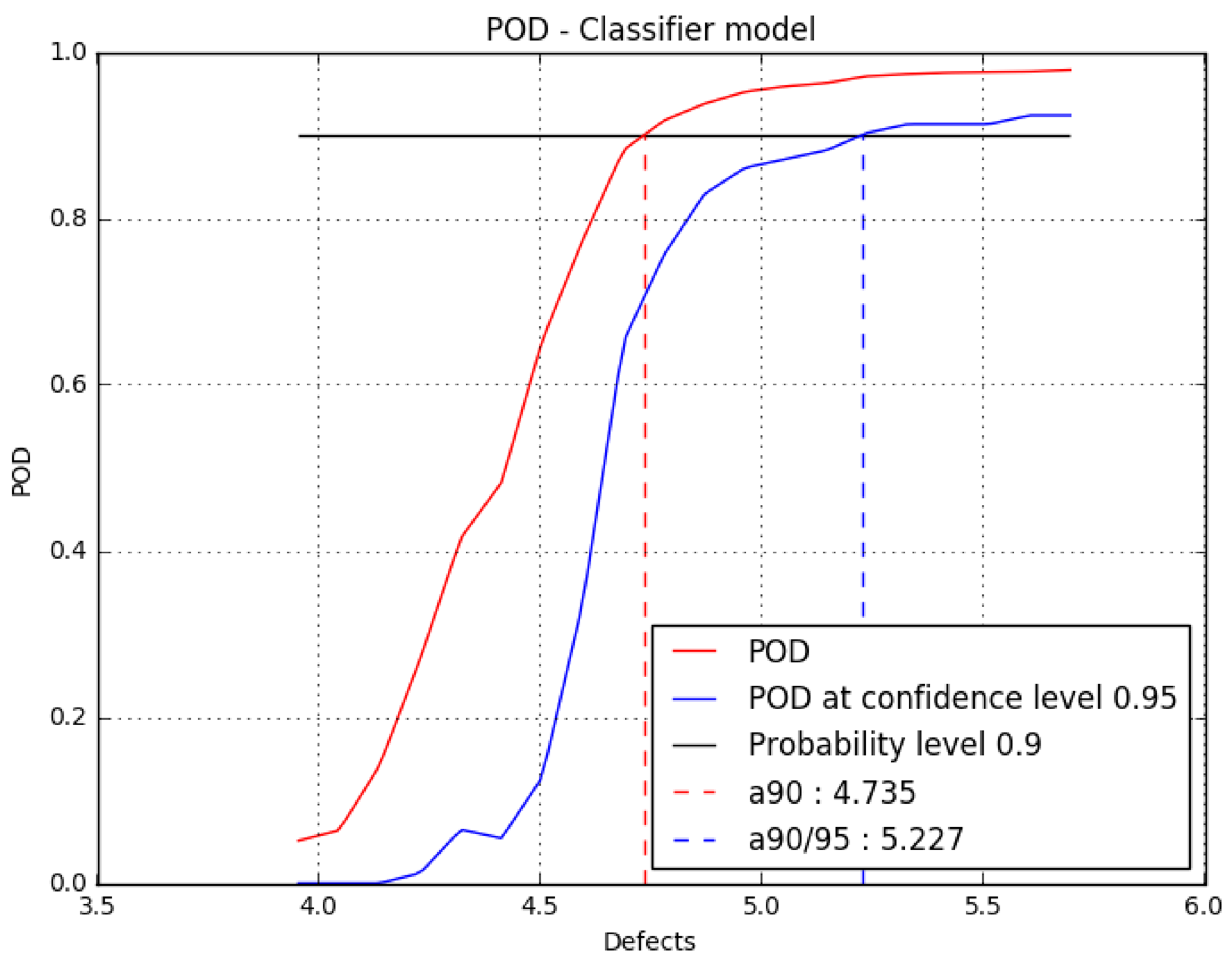
Case where the physical model is a function providing the a signal value. In this case, the detection threshold must be given. The hit miss function is built inside the AdaptiveHitMissPOD class and is then used in the algorithm.

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

signals = ot.NumericalSample(
    [[ 37.305445], [ 35.466919], [ 43.187991], [ 45.305165], [ 40.121222], [ 44.609524],
     [ 45.14552 ], [ 44.80595 ], [ 35.414039], [ 39.851778], [ 42.046049], [ 34.73469 ],
     [ 39.339349], [ 40.384559], [ 38.718623], [ 46.189709], [ 36.155737], [ 31.768369],
     [ 35.384313], [ 47.914584], [ 46.758537], [ 46.564428], [ 39.698493], [ 45.636588],
     [ 40.643948]])

# detection threshold
detection = 38

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

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

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

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

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


POD - Classifier model
1.0
0.8
0.6
0.4
0.2
0.0
POD
3.5
4.0
4.5
5.0
5.5
6.0
Defects
POD
POD at confidence level 0.95
Probability level 0.9
a90 : 4.735
a90/95 : 5.227

CHAPTER TWO

INDICES AND TABLES

- genindex
- search