User Manual

Open-Source pyMAPOD Framework – v.beta

Developed by Computational Design Laboratory (CODE Lab)

Department of Aerospace Engineering, Iowa State University

Leading Developers:

Professor: Leifur Leifsson

Ph.D Student: Xiaosong Du

We gratefully acknowledge the support of the Center for Nondestructive Evaluation Industry/University Cooperative Research Program at Iowa State University

Code and updates are available at the link: https://github.com/Fightingwolf2105/MAPOD

Content

1. Introduction

This user manual aims at providing details and utilization examples on the open-source pyMAPOD framework. For the theoretical background, please go through the associated document, Theory Manual of this framework.

The Computational Design (CODE) lab would like to thank the center of nondestructive evaluation (CNDE) for funding this program. In addition, we also want to say thanks to all the colleagues, Dr. Jiming Song, Dr. William Meeker, Dr. Ronald Roberts, Dr. Leonard Bond, etc. for providing physics-based NDT simulation models, valuable ideas and suggestions.

2. Utilization

This MAPOD framework is written for various types of implementations.

Firstly, it can handle existing data, directly, meaning users only need to provide their data in specified format, no matter which type of physics-based simulation model the data is generated.

Secondly, the framework can be linked with real physics-based simulation models, which can be in the format of python or Matlab (will add more types in upcoming versions). The only requirement is that the users need to modify the physics model script slightly into a specified format, which will be discussed later in this section.

Thirdly, this open-source framework provides the users the options of constructing stochastic metamodel, the PCE. Then an arbitrary number of predicted model response are generated for the calculation of "ahat vs. *a*" linear regression and POD analysis.

And of course, the users are very welcome to go through the open-source code and provide any feedback. We are always glad to keep improving our work. In the following parts of this section, the details of the framework and how to link it with the existing data, physics-based simulation model and efficient PCE metamodel will be discussed.

2.1 Flowchart

The flowchart of this MAPOD framework is given as follows

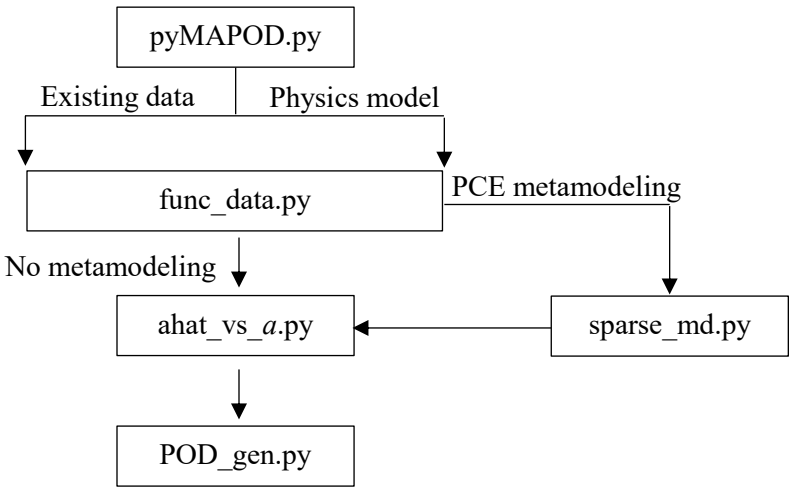

Figure 1. General process of MAPOD framework.

Now, let's go through each python file in details:

pyMAPOD.py: the main function of this framework, used as a configuration file for the POD calculation. The information needed here includes:

a) The type of inputs, existing data or simulation model, *input_type*;
b) The file name of Excel-format data file, *fileName*;
c) The specific sheet containing the data of interest in the Excel file, *sheetName*;
d) Read data, and decide whether view the data or not, *view_data*;
e) The format of simulation model, *funcForm*;
f) The name of callable function of simulation model, *funcName*;
g) The defect sizes, the parameter "*a*" of "ahat vs. *a*" plots;
h) The statistical distributions of random inputs, *x_prob*;
i) The number of sample points, using LHS method, *x_sample*;
j) The matrix variable, containing sample points, *x_exp*;
k) Run the simulation model on *x_exp*, to obtain model response, and assign them to formatted matrix *data*;
l) Decide whether to construct PCE metamodel, *PCE_gen*;
m) The degree of PCE order, *n_deg*;
n) The points used for prediction, x_prediction
o) Construct PCE model, using sparse_md.meta_gen, and return formatted matrix *data*, and corresponding statistical moments *statistics*;
p) Run the "ahat vs. *a*" linear regression, view the regression line through the data, and check the metrics of interest, such as the interception and slope;
q) Set up the detection threshold, *threshold*, and run the POD calculation;
r) Obtain metrics of interest, such as a_{50}, a_{90}, and $a_{90/95}$;
s) View the POD curves.

func_data.py: this file contains only one function: *read_view_data*, used for
a) Read data from specific sheet of a user-provided Excel-format file, and return data to main function;
b) View the data points, if the variable *vew_data = True*.

sparse_md.py: this file consists of 14 functions:
a) *meta_gen*: generate metamodel based on *x_exp* and corresponding model response, using the function *sparse_md*; calculate the statistics at each defect size if provided with discrete sizes, or the statistics of the whole range of defect size if provided as a uniform distribution; calculate the model response of prediction points, x_pred; and finally return statistics and predicted model response;
b) *sparse_md*: construct multi-dimensional PCE model, and solve for corresponding coefficients;
c) *gen_basis_md*: generate multi-dimensional PCE basis, based on the specified statistical distributions, and return PCE basis terms;
d) *gen_legendre_md*: generate Legendre polynomial basis series for Uniform distribution, based on Quadrature points, for each dimension of random inputs;
e) *gen_hermite_md*: generate Hermite polynomial basis series for Gaussian distribution, based on Quadrature points, for each dimension of random inputs;
f) *coef_legendre_md*: generate Legendre polynomial basis series for Uniform distribution, based on sample points, for each dimension of random inputs;
g) *coef_hermite_md*: generate Hermite polynomial basis series for Gaussian distribution, based on sample points, for each dimension of random inputs;
h) *iter_basis*: iteratively generate the multi-dimensional basis of PCE model;
i) *iter_weights*: iteratively generate the multi-dimensional weighting factors of Quadrature points;
j) multichoose: select the order of truncated PCE terms;
k) *prediction_md*: make predictions using generated multi-dimensional PCE model;
l) *prediction_legendre_md*: generate Legendre polynomial basis for prediction use of PCE model;
m) *prediction_hermit_md*: generate Hermite polynomial basis for prediction use of PCE model;
n) *stats_md*: utilize obtained PCE coefficients, and return the statistical moments of the PCE model;

ahat_vs_a.py: this file consists of three functions:
a) *regression*: make linear regression, using numerical expression as shown in Eqn. 14 - 16, on the provided data, the return the interception, slope of regression line, and standard deviation of random error back to the main function;
b) *view_reg*: generate "ahat vs *a*" plots, including data points, linear regression line, 95% confidence bounds, and 95% prediction bounds, based on the results from the function, *regression*;
c) *cov_para2*: calculate and return covariance matrix of two linear regression parameters, interception and slope, using Fisher information matrix, for the generation of 95% confidence bounds and 95% prediction bounds in the function, *view_reg*.

pod_gen.py: this file consists of five functions:
a) *pod_cal*: calculate related parameters as shown in Eqn. 20, 21, then return μ, β, and covariance matrix of them back to the main function;
b) *pod_para*: calculate POD parameters of interest, namely, a_{50}, a_{90}, and $a_{90/95}$, and return them back to the main function;
c) *pod_view*: view the results of POD, including the POD curve and 95% confidence lower bound;
d) *cov_para3*: calculate and return covariance matrix of three linear regression parameters, interception, slope, and standard deviation of random error, using Fisher information matrix, for the generation of 95% confidence bounds in the function, *pod_cal*.
e) *pod_ci*: calculate the 95% confidence low bound, and return it back to the function, *pod_para* for $a_{90/95}$ and *pod_view*.

2.2 Utilization

a) Utilization with existing data

As mentioned above, this MAPOD framework is able to connect with existing data from any physics-based simulation NDT model or experimental. The only required format of the data is shown in Fig. 2.

index	size	response
1	0.1	0.803879
2	0.1	1.180379
3	0.1	0.642051
4	0.1	0.124154
5	0.1	1.047103
6	0.1	1.208215
7	0.1	0.773787
8	0.1	0.447201

Figure 2. Formatted data in Excel file, taking "test_MAPOD.xlsx" as an example.

Details on the required format are as following:
1) Generate model response at various defect sizes, put them into an Excel file, and specify this file name in *pyMAPOD.py* as mentioned in Section 2.1;
2) Give a reasonable name of each Excel sheet within the data file, although users can select to use default name. Specify this sheet name in *pyMAPOD.py* as mentioned in Section 2.1;
3) Within each Excel sheet containing the data of interest, it has three columns totally. The first column has the index of the data, the second column has the current defect size, and the third column has the corresponded model response;

4) The excel sheet should have a title in each column, for example in Fig. 2, we give "*index*", "*size*", "*response*" for each column, respectively.

b) Utilization on physics model

this MAPOD framework is able to connect with real physics-based simulation NDT model, which can be the format of python or Matlab. The only requirement of this type of utilization is that the simulation model has to be a function requesting two arguments, the first one is defect size, and the second one is an array containing all random inputs.

3. Example

After introducing the mathematical background above, it is time to apply the framework to some test cases. The results will be compared against the MIL-HDBK-1823. For the users who are only interested in how to utilize the code directly, feel free skip the part for problem description.

3.1 Ultrasonic

3.1.1 Problem description

This is an ultrasonic benchmark case, called spherically-void-defect case, developed by the World Federal Nondestructive Evaluation Center in 2004. The experimental setup and validation on physics based simulation model are shown in Fig. 3. The spherically void defect, whose radius is 0.34 mm, is included in a fused quartz block, which is surrounded by water. A spherically focused transducer, the radius of which is 6.23mm, is used to detect this defect. The frequency range is set to be [0, 10MHz].

The analytical model, used in this work, is known as the Thompson-Gray model. This model is based on paraxial approximation of the incident and scattered ultrasonic waves, computing the spectrum of voltage at the receiving transducer in terms of the velocity diffraction coefficients of the transmitting/receiving transducers, scattering amplitude of the defect and a frequency-dependent coefficient known as the system-efficiency function. In this work, velocity diffraction coefficients were calculated using the multi-Gaussian beam model and scattering amplitude of the spherical-void was calculated using the method of separation of variables. The system efficiency function, which is a function of the properties and settings of the transducers and the pulser, was taken from the WFNDEC archives. The time-domain pulse-echo waveforms are computed by performing FFT on the voltage spectrum. The foregoing system model was shown to be very accurate in predicting pulse-echo from the spherical void if the paraxial approximation is satisfied and radius of the void is small. To guarantee the effectiveness of this analytical model on the benchmark problem mentioned above, it is validated on this case with experimental data, given in Fig. 3, through which shows that the results match well.

For the POD calculation, we set three uncertainty parameters for the original problem: the probe angle of transducer ~ N(0deg, 0.5deg); the F number of frequency ~ U(13, 15); and the x location of transducer ~ U(0mm, 1mm). Simulations will be run at the defect size: 0.1mm, 0.2mm, 0.3mm, 0.4mm, and 0.5mm, respective, for the data collection before POD calculation. The detection threshold is set as 6.5mV.

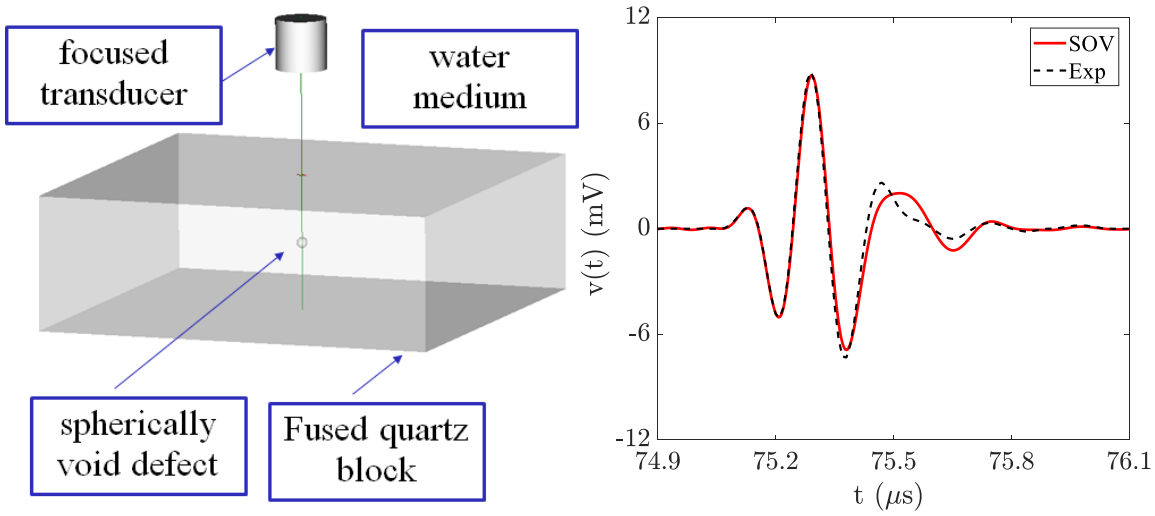

Figure 3. Setup of the spherically-void-defect benchmark case (left) and results of comparison between experimental data (Exp) and the analytical solution (SOV).

3.1.2 Configuration code for POD generation

```
import func_data
import ahat_vs_a
import pod_gen
import collections
import prob_distrs
import numpy as npy

input_type = 'dat'

beta0, beta1, tau = ahat_vs_a.regression(data)

ahat_vs_a.view_reg(data, beta0, beta1, tau)

threshold = 6.5

mu, sigma, pcov = pod_gen.pod_cal(data, beta0, beta1, tau, threshold)

a_50, a_90, a_90_95 = pod_gen.pod_para(mu, sigma, pcov)

pod_gen.pod_view(mu, sigma, pcov)
```

3.1.3 Results

Results on "ahat vs. a" regression are shown in Fig. 4. And related results on POD are shown in Fig.5. It is clear that both sets of plots look very similar with each other. This is reasonable, because we apply the same method to make calculation and add 95% confidence bounds on the plots.

Moreover, the key parameters, such as β_0, β_1, and τ in "ahat vs. a" plots, μ, σ and corresponding covariance matrix in POD curves, have the same values.

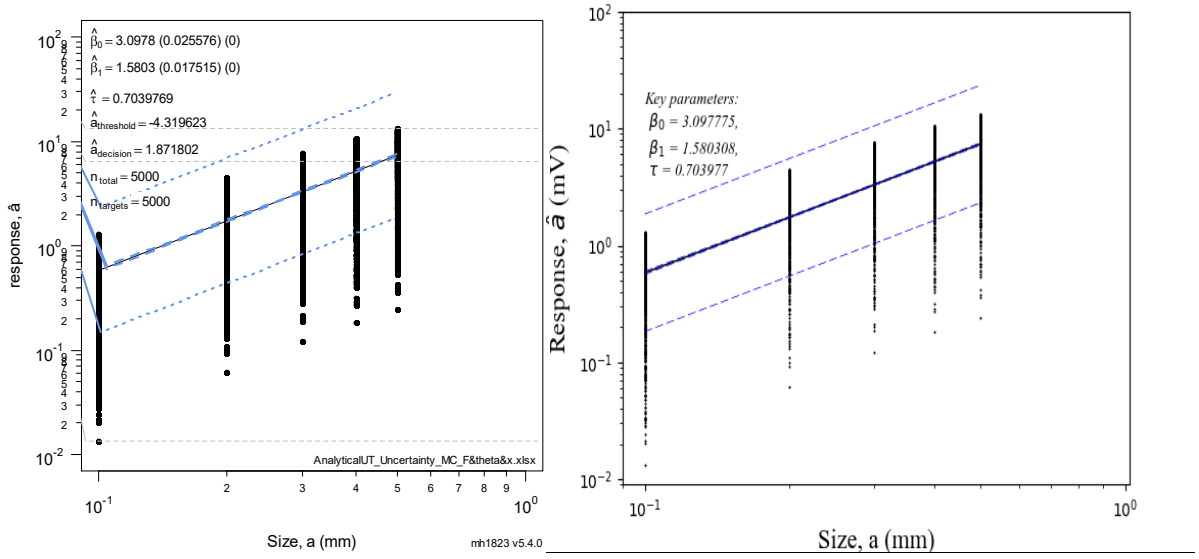

Figure 4. Comparison of "ahat vs. *a*" regressions: left) MH1823; right) pyMAPOD.

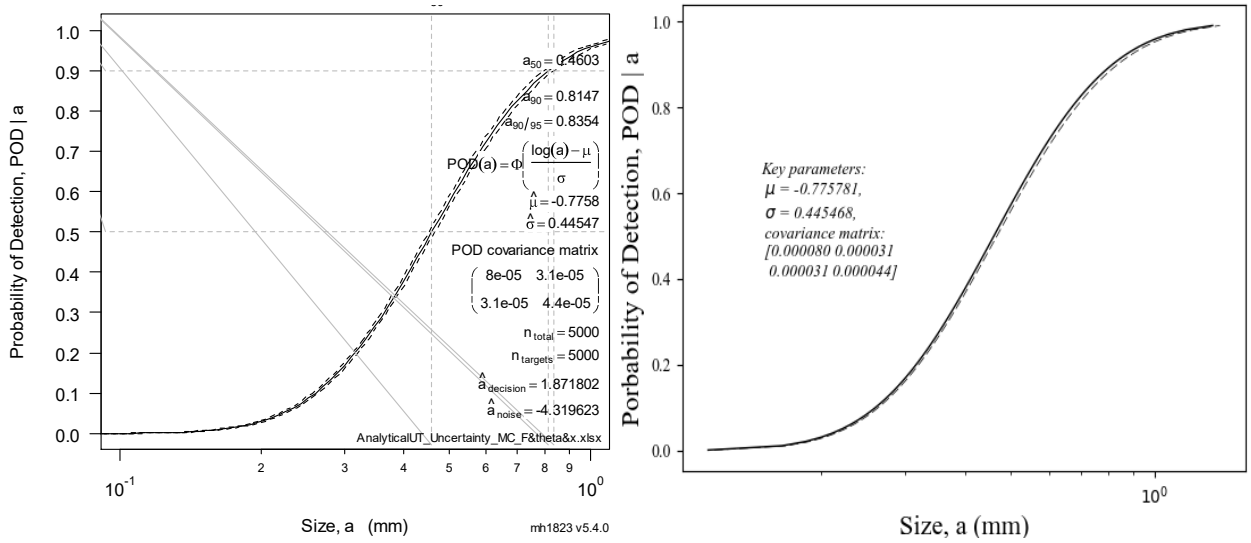

Figure 5. Comparison of POD curves: left) MH1823; right) pyMAPOD.

3.2 Numerical example

3.2.1 Problem description

This MAPOD framework can also be connected with physics-based simulation model, directly. Here we will take a simple numerical case as an example.

$$y = e^{k\cdot\ln(a)+b}, \tag{50}$$

where k has the distribution of *Uniform* (3, 4), b has the distribution of *Normal* (5, 0.5), a can be taken as five discrete points [0.1, 0.2, 0.3, 0.4, 0.5], or with a distribution of *Uniform* (0.1, 0.5).

3.2.2 python function

Here, we assume that the physics-based simulation model is of python format. The python file, test_func.py, with the function simpleFunc which takes two arguments, is provided. Then we only need to take care of the pyMAPOD.py file, and can always follow the steps below to run this pyMAPOD framework.

```
import func_data
import ahat_vs_a
import pod_gen
import collections
import prob_distrs
import numpy as npy
import test_func

input_type = 'func'

funcForm = ‘python’

funcName = test_func.simpleFunc

a = npy.array([0.1, 0.2, 0.3, 0.4, 0.5])

x_prob = collections.OrderedDict([(‘Uniform1’, (3, 4)),
                                  (‘Gaussian1’, (5, 0.5))])

x_sample = {‘LHS’ : 100}

PCE_gen = False

beta0, beta1, tau = ahat_vs_a.regression(data)

ahat_vs_a.view_reg(data, beta0, beta1, tau)

threshold = 0.5

mu, sigma, pcov = pod_gen.pod_cal(data, beta0, beta1, tau, threshold)

a_50, a_90, a_90_95 = pod_gen.pod_para(mu, sigma, pcov)

pod_gen.pod_view(mu, sigma, pcov)
```

The resulting plots are as follows.

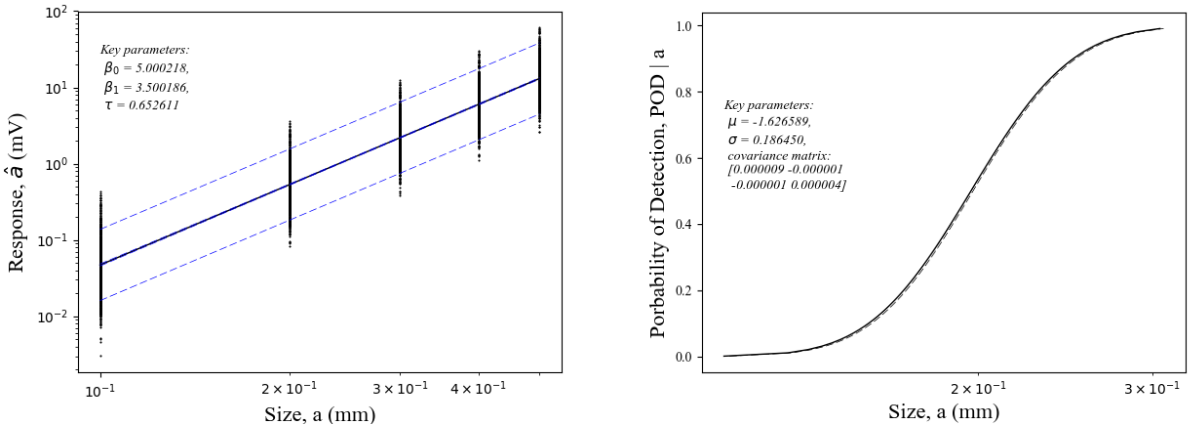

Figure 6. "ahat vs. *a*" plots and POD curves.

We can also use the independent variable, *a*, as Uniform (0.1, 0.5). Then all the settings are the same as above, except the followings:

```
a = npy.array([])

x_prob = collections.OrderedDict([('Uniform1', (0.1, 0.5)),
                                  ('Uniform2', (3, 4)),
                                  ('Gaussian1', (5, 0.5))])

x_sample = {'LHS' : 1000}
```

The resulting plots are as follows.

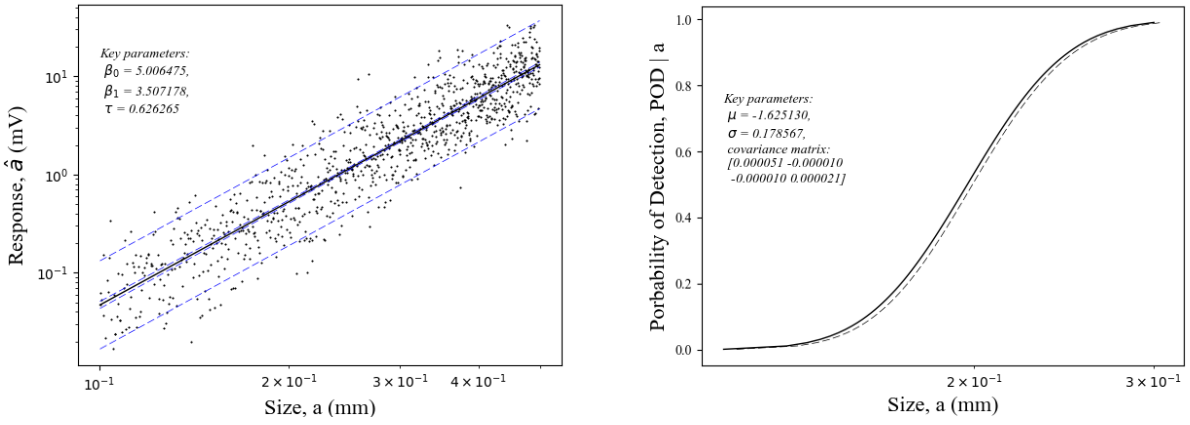

Figure 7. "ahat vs. *a*" plots and POD curves.

3.2.3 PCE model on python function

For the same problem as mentioned in section B, we can construct PCE model first, then generate POD curves, following the steps below. *a* is set as [0.1, 0.2, 0.3, 0.4, 0.5]. The resulting plots are in Fig. 8.

```
import func_data
import ahat_vs_a
import pod_gen
import collections
import prob_distrs
import numpy as npy
import sparse_md
import test_func

input_type = 'func'

funcForm = ‘python’

funcName = test_func.simpleFunc

a = npy.array([0.1, 0.2, 0.3, 0.4, 0.5])

x_prob = collections.OrderedDict([(‘Uniform1’, (3, 4)),
                                  (‘Gaussian1’, (5, 0.5))])

x_sample = {‘LHS’ : 45}

PCE_gen = True

n_deg = 8

x_meta = {'LHS' : 1000}

beta0, beta1, tau = ahat_vs_a.regression(data)

ahat_vs_a.view_reg(data, beta0, beta1, tau)

threshold = 0.5

mu, sigma, pcov = pod_gen.pod_cal(data, beta0, beta1, tau, threshold)

a_50, a_90, a_90_95 = pod_gen.pod_para(mu, sigma, pcov)

pod_gen.pod_view(mu, sigma, pcov)
```

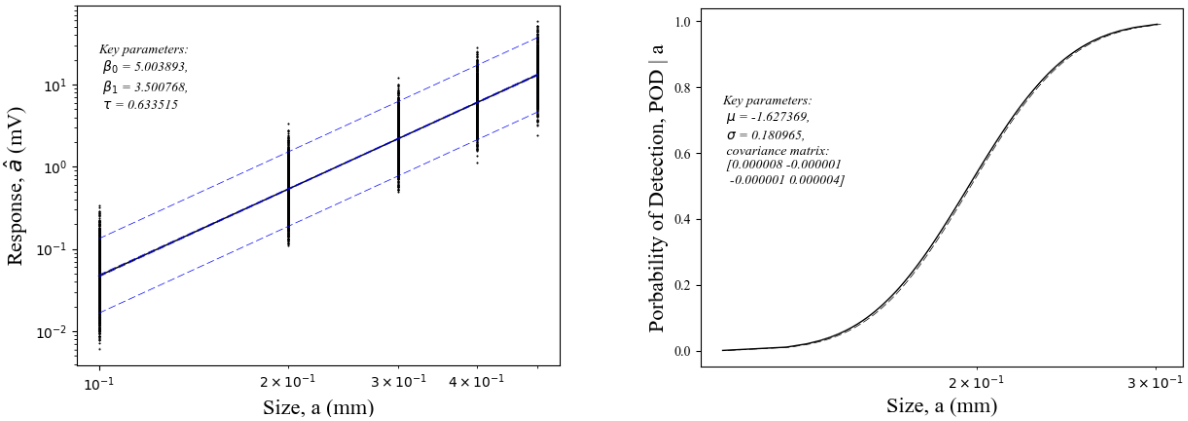

Figure 8. "ahat vs. *a*" plots and POD curves.

a can be set as a random variable, with Uniform (0.1, 0.5). Then,

```
a = npy.array([])

x_prob = collections.OrderedDict([('Uniform1', (0.1, 0.5)),
                                  ('Uniform2', (3, 4)),
                                  ('Gaussian1', (5, 0.5))])

x_sample = {'LHS' : 120}

n_deg = 7
```

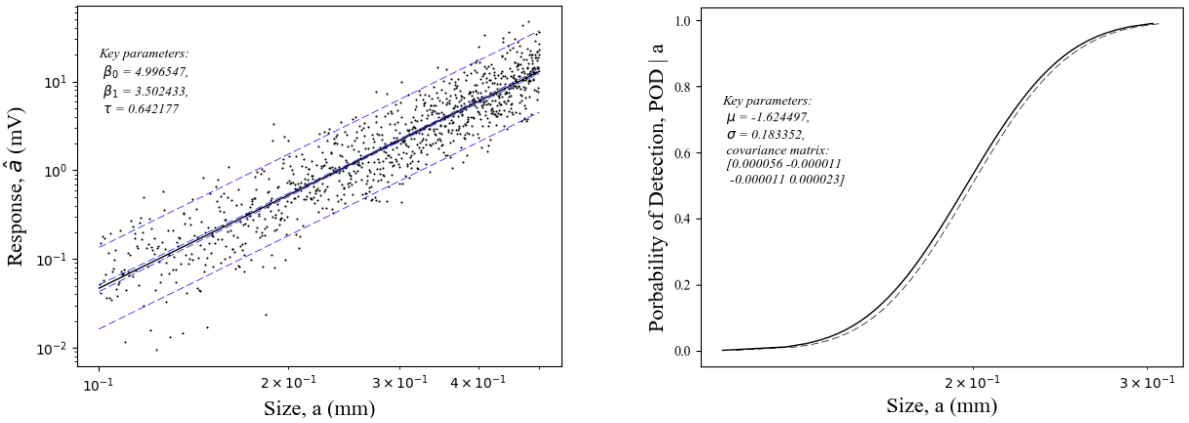

Figure 9. "ahat vs. *a*" plots and POD curves.

3.2.4 Matlab function

Similarly as section B, we can also link this MAPOD framework to Matlab-format physics-based simulation model. The resulting plots are shown in Fig. 10.

```
import func_data
import ahat_vs_a
import pod_gen
import collections
import prob_distrs
import numpy as npy

input_type = 'func'

funcForm = 'matlab'

funcName = 'test_func_mat'

a = npy.array([0.1, 0.2, 0.3, 0.4, 0.5])

x_prob = collections.OrderedDict([('Uniform1', (3, 4)),
                                  ('Gaussian1', (5, 0.5))])

x_sample = {'LHS' : 100}

PCE_gen = False

beta0, beta1, tau = ahat_vs_a.regression(data)

ahat_vs_a.view_reg(data, beta0, beta1, tau)

threshold = 0.5

mu, sigma, pcov = pod_gen.pod_cal(data, beta0, beta1, tau, threshold)

a_50, a_90, a_90_95 = pod_gen.pod_para(mu, sigma, pcov)

pod_gen.pod_view(mu, sigma, pcov)
```

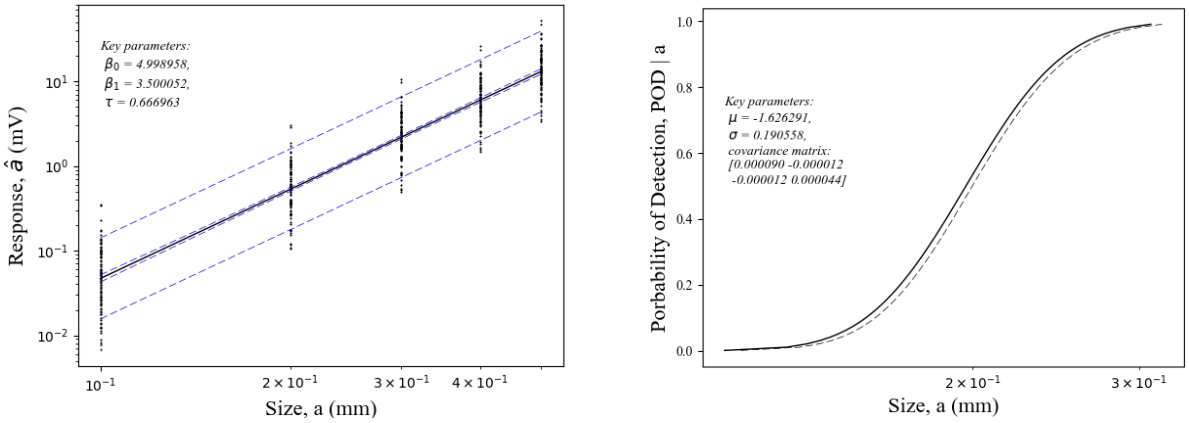

Figure 10. "ahat vs. *a*" plots and POD curves.

We can also use the independent variable, *a*, as Uniform (0.1, 0.5). Then all the settings are the same as above, except the followings:

```
a = npy.array([])

x_prob = collections.OrderedDict([('Uniform1', (0.1, 0.5)),
                                  ('Uniform2', (3, 4)),
                                  ('Gaussian1', (5, 0.5))])

x_sample = {'LHS' : 1000}
```

The resulting plots are as follows.

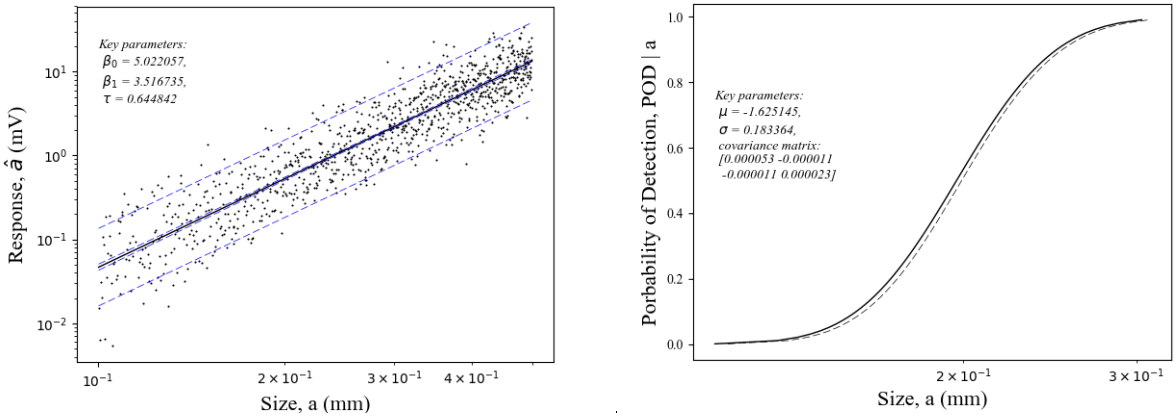

Figure 11. "ahat vs. *a*" plots and POD curves.

3.2.5 PCE model on Matlab function

For the same problem as mentioned in section B, we can construct PCE model first, then generate POD curves, following the steps below. *a* is set as [0.1, 0.2, 0.3, 0.4, 0.5]. Resulting plots are shown in Fig. 12.

```
import func_data
import ahat_vs_a
import pod_gen
import collections
import prob_distrs
import numpy as npy
import sparse_md

input_type = 'func'

funcForm = ‘matlab’

funcName = ‘test_func_mat’

a = npy.array([0.1, 0.2, 0.3, 0.4, 0.5])

x_prob = collections.OrderedDict([(‘Uniform1’, (3, 4)),
                                  (‘Gaussian1’, (5, 0.5))])

x_sample = {‘LHS’ : 45}

PCE_gen = True

n_deg = 8

x_meta = {'LHS' : 1000}

beta0, beta1, tau = ahat_vs_a.regression(data)

ahat_vs_a.view_reg(data, beta0, beta1, tau)

threshold = 0.5

mu, sigma, pcov = pod_gen.pod_cal(data, beta0, beta1, tau, threshold)

a_50, a_90, a_90_95 = pod_gen.pod_para(mu, sigma, pcov)

pod_gen.pod_view(mu, sigma, pcov)
```

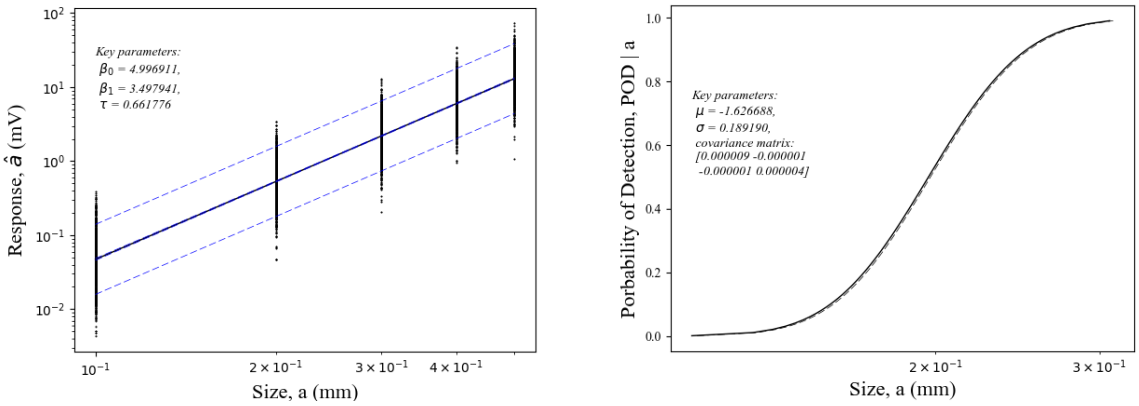

Figure 12. "ahat vs. *a*" plots and POD curves.

We can also use the independent variable, *a*, as Uniform (0.1, 0.5). Then all the settings are the same as above, except the followings:

```
a = npy.array([])

x_prob = collections.OrderedDict([('Uniform1', (0.1, 0.5)),
                                  ('Uniform2', (3, 4)),
                                  ('Gaussian1', (5, 0.5))])

x_sample = {'LHS' : 1000}
```

The resulting plots are as follows.

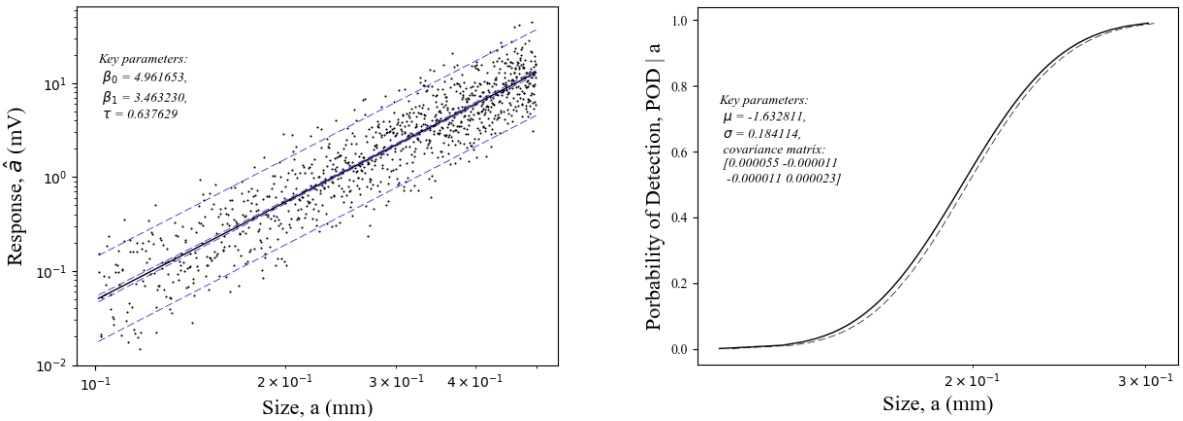

Figure 13. "ahat vs. *a*" plots and POD curves.