Wollensack Michael 15.05.2020

METAS UncLib Python V2.3.1
User Reference

This document is a quick reference sheet. For practical demonstrations and more details refer to the examples that are provided with the installation of the software.

The *Metas.UncLib.Python* library is an extension to Python, which supports creation of uncertainty objects and subsequent calculation with them as well as storage of the results. It has been developed with Python V3.6 using the numpy (1.16.1) and the pythonnet (2.3.0) packages. It requires the C# library *Metas.UncLib* in the background. There are three modules for uncertainty propagation: 'LinProp', 'DistProp' and 'MCProp'.

- 'LinProp' supports linear uncertainty propagation. This class is fully functional.
- 'DistProp' supports higher order uncertainty propagation, i.e. higher order terms of the Taylor expansion of the measurement equation are taken into account. This class currently only supports Gaussian input distributions and assumes Gaussian result distributions. In the future it will be possible to specify non-Gaussian inputs and to use higher moments of the results to calculate more realistic coverage intervals.
- 'MCProp' supports Monte Carlo propagation but its implementation is still preliminary and its use is not recommended yet.

1. Global uncertainty settings

`from metas_unclib import *` `use_linprop()`	Use the linear uncertainty propagation.
`from metas_unclib import *` `use_distprop(maxlevel=1)`	Use the higher order uncertainty propagation. The argument maxlevel specifies the higher order uncertainty propagation maximum level. Default value: 1 (1 corresponds to LinProp)
`from metas_unclib import *` `use_mcprop(n=100000)`	Use the Monte Carlo uncertainty propagation. The argument n specifies the Monte Carlo uncertainty propagation sample size. Default value: 100'000

2. Create an uncertainty object

Square brackets indicate vector or matrix.

`x = ufloat(value)`	Create uncertain number without uncertainty.
`x = ufloat(value, stdunc, idof=0.0, desc=None)`	Creates a new real uncertain number with value, standard uncertainty, inverse degrees of freedom (optional), and a description (optional).
`x = ucomplex(value, [covariance], desc=None)`	Creates a new complex uncertain number. Covariance size: 2x2
`x = ufloatarray(value, [covariance], desc=None)`	Creates a new real uncertain array. N: value.Length Covariance size: NxN
`x = ucomplexarray(value, [covariance], desc=None)`	Creates a new complex uncertain array. N: value.Length Covariance size: 2Nx2N
`x = ufloat(value, [sys_inputs],[sys_sensitivities])`	Create uncertain number by setting sensitivities with respect to uncertain inputs. [1]

3. Calculations with uncertainty objects

3.1 Math functions

`x + y`	`x * y`	`x ** y (power)`	
`x - y`	`x / y`	`-x`	
`umath.sqrt(x)`	`umath.sin(x)`	`umath.sinh(x)`	`umath.real(x)`
`umath.exp(x)`	`umath.cos(x)`	`umath.cosh(x)`	`umath.imag(x)`
`umath.log(x)`	`umath.tan(x)`	`umath.tanh(x)`	`umath.abs(x)`
`umath.log10(x)`	`umath.asin(x)`	`umath.asinh(x)`	`umath.angle(x)`
	`umath.acos(x)`	`umath.acosh(x)`	`umath.conj(x)`
`umath.pow(x, y)`	`umath.atan(x)`	`umath.atanh(x)`	

3.2 Linear Algebra

`ulinalg.dot(M1, M2)`	Matrix multiplication of matrix M1 and M2
`ulinalg.det(M)`	Determinate of matrix M
`ulinalg.inv(M)`	Matrix inverse of M
`ulinalg.solve(A, Y)`	Solve linear equation system: A * X = Y
`ulinalg.lstsqrsolve(A, Y)`	Least square solve over determined equation system.
`ulinalg.weightedlstsqrsolve( A, Y, W)`	Weighted least square solve over determined equation system.
`V, D = ulinalg.eig(A0)`	Eigenvalue problem [1]: A0 * V = V * D
`V, D = ulinalg.eig(A0, A1, A2, ..., An-1)`	Non-linear Eigenvalue problem [1]: A0 * V + A1 * V * D + A2 * V * D^2 + ... + A(n-1) * V * D^(n-1) = 0

3.3 Numerical Routines

`unumlib.polyfit(x, y, n)`	Fit polynom to data
`unumlib.polyval(p, x)`	Evaluate polynom
`unumlib.interpolation(x, y, n, xx)`	Interpolation
`unumlib.interpolation2(x, y, n, xx)`	Interpolation with linear unc interpolation
`unumlib.splineinterpolation(x, y, xx, boundaries)`	Spline interpolation
`unumlib.splineinterpolation2(x, y, xx, boundaries)`	Spline interpolation with linear unc interpolation
`unumlib.integrate(x, y, n)`	Integrate
`unumlib.splineintegrate(x, y, n)`	Spline integrate
`unumlib.fft(v)`	Fast Fourier transformation
`unumlib.ifft(v)`	Inverse Fast Fourier transformation
`unumlib.numerical_step(@f, x, dx)`	Numerical Step [1]
`unumlib.optimizer(@f, xStart, p)`	Optimizer [1]

4. Get properties of an uncertainty object

`get_value(y)`	Returns the expected value.
`get_fcn_value(y)`	Returns the function value.
`get_stdunc(y)`	Computes the standard uncertainty.
`get_idof(y)`	Computes the inverse degrees of freedom. [1]
`1.0 / get_idof (y)`	Computes the degrees of freedom. [1]
`get_coverage_interval(y, p)`	Computes the coverage interval.
`get_moment(y, n)`	Computes the n^{th} central moment.
`get_jacobi(y)`	Returns the sensitivities to the virtual base inputs (with value 0 and uncertainty 1). [1]
`get_jacobi2(y, x)`	Computes the sensitivities of y to the intermediate results x. [1]
`get_unc_component(y, x)`	Computes the uncertainty components of y with respect to x. [1]
`get_correlation([y1 y2 ...])`	Computes the correlation matrix
`get_covariance([y1 y2 ...])`	Computes the covariance matrix

5. Storage functions

5.1 Store a computed uncertainty object

`ustorage.save_binary_file(y, filepath)`	Binary serializes uncertainty object y to file.
`ustorage.save_xml_file(y, filepath)`	Xml serializes uncertainty object y to file.
`ustorage.to_xml_string(y)`	Xml serializes uncertainty object y to string.

5.2 Reload a stored uncertainty object

`ustorage.load_binary_file( filepath)`	Reloads uncertainty object from binary file.
`ustorage.load_xml_file( filepath)`	Reloads uncertainty object from xml file.
`ustorage.from_xml_string(s)`	Reloads uncertainty object from xml string.

[1] 'LinProp' uncertainty objects only