



METAS UncLib Python - User Reference V2.5.3

Michael Wollensack

February 2022

Contents

1	Introduction	2
2	Global uncertainty settings	2
3	Create an uncertainty object	3
4	Calculations with uncertainty objects	4
4.1	Math functions	4
4.2	Linear algebra	4
4.3	Numerical routines	5
5	Get properties of an uncertainty object	6
6	Storage functions	6
6.1	Store a computed uncertainty object	6
6.2	Reload a stored uncertainty object	6



METAS UncLib Python - User Reference V2.5.3

1 Introduction

This document is a quick reference sheet. For practical demonstrations and more details refer to the tutorial and 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's able to handle complex-valued and multivariate quantities. 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 $V_{out} = J V_{in} J'$.

DistProp supports higher order uncertainty propagation, i.e. higher order terms of the Taylor expansion of the measurement equation are taken into account.¹

MCProp supports Monte Carlo propagation.¹

2 Global uncertainty settings

`from metas_unclib import *` Import METAS UncLib.

`use_linprop()` Use the linear uncertainty propagation.

`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](#))

`use_mcprop(n=100000)` Use the Monte Carlo uncertainty propagation.

The argument `n` specifies the Monte Carlo uncertainty propagation sample size. Default value: 100000

¹preliminary implementation



3 Create an uncertainty object

Square brackets indicate vector or matrix.

`x = ufloat(value)` Creates a new uncertain number without uncertainties.

`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: 2×2

`x = ufloatarray([value], [covariance], desc=None)` Creates a new real uncertain array. Covariance size: $N \times N$

`x = ucomplexarray([value], [covariance], desc=None)` Creates a new complex uncertain array.

`x = ufloatfromsamples([samples], desc=None, p=0.95)` Creates a new real uncertain number from samples with a description (optional) and a probability (optional).

`x = ucomplexfromsamples([samples], desc=None, p=0.95)` Creates a new complex uncertain number from samples with a description (optional) and a probability (optional). The complex uncertain number contains the correlation between real and imaginary parts.

`x = ufloatarrayfromsamples([samples], desc=None, p=0.95)` Creates a new real uncertain array from samples with a description (optional) and a probability (optional). The real uncertain array contains the correlation between the different entries.

`x = ucomplexarrayfromsamples([samples], desc=None, p=0.95)` Creates a new complex uncertain array from samples with a description (optional) and a probability (optional). The complex uncertain array contains the correlation between real and the imaginary parts and the different entries.

`x = ufloatsystem(value, [sys_inputs], [sys_sensitivities])` Creates a new real uncertain number by setting sensitivities with respect to uncertain inputs.²

²[LinProp](#) uncertainty objects only



4 Calculations with uncertainty objects

4.1 Math functions

- `x + y`
- `x * y`
- `x ** y`³
- `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)`

4.2 Linear algebra

`ulinalg.dot(M1, M2)` Matrix multiplication of matrix M_1 and M_2

`ulinalg.det(M)` Determinate of matrix M

`ulinalg.inv(M)` Matrix inverse of M

`ulinalg.solve(A, Y)` Solve linear equation system: $Ax = y$

`ulinalg.lstsqrsolve(A, Y)` Least square solve over determined equation system

`ulinalg.weightedlstsqrsolve(A, Y, W)` Weighted least square solve over determined equation system

$V, D = \text{ulinalg.eig}(A_0)$ Eigenvalue problem²: $A_0V = VD$

$V, D = \text{ulinalg.eig}(A_0, A_1, A_2, \dots, A_{n-1})$ Non-linear eigenvalue problem²: $A_0V + A_1VD + A_2VD^2 + \dots + A_{(n-1)}VD^{(n-1)} = 0$

²`LinProp` uncertainty objects only

³`**` is the power operator



4.3 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 uncertainty propagation

`unumlib.splineinterpolation(x, y, xx, boundaries)` Spline interpolation

`unumlib.splineinterpolation2(x, y, xx, boundaries)` Spline interpolation with linear uncertainty propagation

`unumlib.integrate(x, y, n)` Integrate

`unumlib.splineintegrate(x, y, boundaries)` Spline integrate

`unumlib.fft(v)` Fast Fourier transformation

`unumlib.ifft(v)` Inverse Fast Fourier transformation

`unumlib.numerical_step(@f, x, dx)` Numerical step²

`unumlib.optimizer(@f, xStart, p)` Optimizer²

²`LinProp` uncertainty objects only



5 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_coverage_interval(y, p)` Computes the coverage interval.

`get_moment(y, n)` Computes the n-th central moment.

`get_correlation([y1 y2 ...])` Computes the correlation matrix.

`get_covariance([y1 y2 ...])` Computes the covariance matrix.

`get_idof(y)` Computes the inverse degrees of freedom.²

`1.0 / get_idof(y)` Computes the degrees of freedom.²

`get_jacobi(y)` Returns the sensitivities to the virtual base inputs (with value 0 and uncertainty 1).²

`get_jacobi2(y, x)` Computes the sensitivities of y to the intermediate results x.²

`get_unc_component(y, x)` Computes the uncertainty components of y with respect to x.²

`unc_budget(y)` Shows the uncertainty budget.²

6 Storage functions

6.1 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.

6.2 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.

²LinProp uncertainty objects only