



METAS UncLib MATLAB - User Reference V2.5.0

Michael Wollensack

September 2021

Contents

1	Introduction	2
1.1	Object Behavior	2
2	Global uncertainty settings	2
2.1	Set function handle	2
2.2	Additional global settings	2
3	Create an uncertainty object	3
4	Calculations with uncertainty objects	3
4.1	Math functions	3
4.2	Linear algebra	4
4.3	Numerical routines	4
5	Get properties of an uncertainty object	5
6	Storage functions	5
6.1	Store a computed uncertainty object	5
6.2	Reload a stored uncertainty object	5



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 MATLAB** library is an extension to MATLAB, 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 MATLAB V8.3 (R2014a) and it requires the C# library **METAS UncLib** in the background. The classes **LinProp**, **DistProp** and **MCTProp** wrap **METAS UncLib** to MATLAB over the .NET interface.

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

MCTProp supports Monte Carlo propagation.¹

1.1 Object Behavior

Scalar **LinProp**, **DistProp** and **MCTProp** objects behave like MATLAB fundamental types with respect to copy operations. Copies are independent values. Operations that you perform on one object do not affect copies of that object.

Non-scalar **LinProp**, **DistProp** and **MCTProp** objects are referenced by their handle variable. Copies of the handle variable refer to the same object. Operations that you perform on a handle object are visible from all handle variables that reference that object.

`B = copy(A)` copies each element in the array of handles A to the new array of handles B.

2 Global uncertainty settings

2.1 Set function handle

`unc = @LinProp` Set function handle `unc` to linear uncertainty propagation.

`unc = @DistProp` Set function handle `unc` to higher order uncertainty propagation.

`unc = @MCTProp` Set function handle `unc` to Monte Carlo uncertainty propagation.

2.2 Additional global settings

`DistPropGlobalMaxLevel(1)` Set the higher order uncertainty propagation maximum level.
Default value: 1 (1 corresponds to **LinProp**)

`MCTPropGlobalN(n)` Set the Monte Carlo uncertainty propagation sample size. Default value: 100000

¹preliminary implementation



3 Create an uncertainty object

Square brackets indicate vector or matrix.

`unc(value)` Creates a new uncertain number or array without uncertainties.

`unc(value, stdunc, (idof))` Creates a new real uncertain number with value, standard uncertainty and inverse degrees of freedom (optional).

`unc(value, stdunc, (description))` Creates a new real uncertain number with value, standard uncertainty and a description (optional).

`unc(value, [covariance], (description))` Creates a new complex uncertain number. Covariance size: 2×2

`unc([value], [covariance], (description))` Creates a new real uncertain array. Covariance size: $N \times N$

`unc([value], [covariance], (description))` Creates a new complex uncertain array. Covariance size: $2N \times 2N$

`unc([samples], 'samples', (description), (probability))` Creates a new real or complex uncertain number or array from samples with a description (optional) and a probability (optional). The result contains the correlation between the different entries.

`unc(value, standard_unc, idof, id, description)` Creates a new real uncertain number with value, standard uncertainty, inverse degrees of freedom, input id and description.²

`unc(value, [sys_inputs], [sys_sensitivities], 'system')` Create uncertain number by setting sensitivities with respect to uncertain inputs.²

4 Calculations with uncertainty objects

Use MATLAB call `methods(y)` on uncertainty object `y` to obtain a full list of supported methods.

4.1 Math functions

- | | | | | |
|----------------------|--------------------------|------------------------|-------------------------|----------------------------|
| • <code>x + y</code> | • <code>sqrt(x)</code> | • <code>sin(x)</code> | • <code>sinh(x)</code> | • <code>real(x)</code> |
| • <code>x - y</code> | • <code>exp(x)</code> | • <code>cos(x)</code> | • <code>cosh(x)</code> | • <code>imag(x)</code> |
| • <code>x.*y</code> | • <code>log(x)</code> | • <code>tan(x)</code> | • <code>tanh(x)</code> | • <code>abs(x)</code> |
| • <code>x./y</code> | • <code>log10(x)</code> | • <code>asin(x)</code> | • <code>asinh(x)</code> | • <code>angle(x)</code> |
| • <code>x.^y</code> | • <code>log(x, y)</code> | • <code>acos(x)</code> | • <code>acosh(x)</code> | • <code>conj(x)</code> |
| | • <code>sign(x)</code> | • <code>atan(x)</code> | • <code>atanh(x)</code> | • <code>atan2(x, y)</code> |

²`LinProp` uncertainty objects only



4.2 Linear algebra

`M1*M2` Matrix multiplication of matrix M_1 and M_2

`lu(M)` LU decomposition of matrix M

`det(M)` Determinate of matrix M

`inv(M)` Matrix inverse of M

`A\y` Solve linear equation system: $Ax = y$

`A\y` Least square solve over determined equation system

`lscov(A, y, W)` Weighted least square solve over determined equation system

`[V, D] = eig(A0)` Eigenvalue problem²: $A_0V = VD$

`[V, D] = eig(A0, A1, A2, ..., An)` Non-linear eigenvalue problem²: $A_0V + A_1VD + A_2VD^2 + \dots + A_nVD^n = 0$

4.3 Numerical routines

`polyfit(x, y, n)` Fit polynom to data

`polyval(p, x)` Evaluate polynom

`roots(p)` Roots of the polynom

`interpolation(x, y, n, xx)` Interpolation

`interpolation2(x, y, n, xx)` Interpolation with linear uncertainty propagation

`spline(x, y, xx, boundaries)` Spline interpolation

`spline2(x, y, xx, boundaries)` Spline interpolation with linear uncertainty propagation

`integrate(x, y, n)` Integrate

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

`fft(v)` Fast Fourier transformation

`ifft(v)` Inverse Fast Fourier transformation

`numerical_step(@func, x, dx)` Numerical step²

`optimizer(@func, 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./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

`binary_file(y, filepath)` Binary serializes uncertainty object y to file.

`xml_file(y, filepath)` XML serializes uncertainty object y to file.

`xml_string(y)` XML serializes uncertainty object y to string.

6.2 Reload a stored uncertainty object

`unc(filepath, 'binary_file')` Reloads uncertainty object from binary file.

`unc(filepath, 'xml_file')` Reloads uncertainty object from XML file.

`unc(xml_string)` Reloads uncertainty object from XML string.

²LinProp uncertainty objects only