METAS UncLib MATLAB - User Reference V2.8.2

Michael Wollensack

April 2024

Contents

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 `MCProp` wrap METAS UncLib to MATLAB over the .NET interface.

LinProp supports linear uncertainty propagation $\mathbf{V}_{out} = \mathbf{J}\mathbf{V}_{in}\mathbf{J}'$.

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]

1.1 Object Behavior

Scalar `LinProp`, `DistProp` and `MCProp` 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 `MCProp` 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 = @MCProp` Set function handle `unc` to Monte Carlo uncertainty propagation.

2.2 Additional global settings

`LinPropGlobalDofMode(mode)` Set the degrees of freedom mode to WelchSatterthwaite or to StudentT. Default value: WelchSatterthwaite

`LinPropGlobalFromSamplesMode(mode)` Set the from samples mode to Dof or to Expand-InputCovariance. Default value: ExpandInputCovariance

[1]preliminary implementation

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

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

3 Create an uncertainty object

Square brackets indicate vector or matrix.

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

`unc([value])` Creates a new array without uncertainties.

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

`unc(value, [covariance], (idof), (id), (description))` Creates a new complex uncertain number. Covariance size: 2×2. Covariance normalized to $dof = n - 2$.

`unc([value], [covariance], (idof), (id), (description))` Creates a new real uncertain array. Covariance size: $N \times N$. Covariance normalized to $dof = n - N$.

`unc([value], [covariance], (idof), (id), (description))` Creates a new complex uncertain array. Covariance size: $2N \times 2N$. Covariance normalized to $dof = n - 2N$.

`unc([samples], 'samples', (id), (description), (probability))` Creates a new real or complex uncertain number or array from samples with an ID (optional), a description (optional) and a probability (optional). Samples size: $n \times N$ where n is the number of observations and N is the number of dimensions. The result contains the correlation between the different entries.

`unc([samples], 'randomchoices', (id), (description))` Creates a new real or complex uncertain number or array from random choices with an ID (optional) and a description (optional). Samples size: $n \times N$ where n is the number of observations and N is the number of dimensions. The result contains the correlation between the different entries.

`unc(distribution, (id), (desc))` Creates a new real uncertain number from a distribution with an ID (optional) and a description (optional).

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

[2] `LinProp` uncertainty objects only

3.1 Distributions

`StandardNormalDistribution()` Creates a normal distribution with $\mu = 0$ and $\sigma = 1$.

`NormalDistribution(mu, sigma)` Creates a normal distribution with μ and σ.

`StandardUniformDistribution()` Creates an uniform distribution between $a = 0$ and $b = 1$.

`UniformDistribution(a, b)` Creates an uniform distribution between a and b.

`CurvilinearTrapezoidDistribution(a, b, d)` Creates a curvilinear trapezoid distribution between $a \pm d$ and $b \pm d$.

`TrapezoidalDistribution(a, b, beta)` Creates a trapezoidal distribution between a and b with β.

`TriangularDistribution(a, b)` Creates a triangular distribution between a and b.

`ArcSineDistribution(a, b)` Creates an arc sine distribution between a and b.

`ExponentialDistribution(mu)` Creates an exponential distribution with μ.

`GammaDistribution(a, b)` Creates a gamma distribution with shape a and scale b.

`ChiSquaredDistribution(k)` Creates a chi-squared distribution with degrees of freedom k.

`StudentTDistribution(mu, sigma, dof)` Creates a Student T distribution with μ, σ and dof.

`StudentTFromSamplesDistribution([samples])` Creates a Student T distribution from samples.

`RandomChoicesFromSamples(seed, [samples])` Creates random choices from samples with a seed.

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

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

4.2 Linear algebra

`M1*M2` Matrix multiplication of matrix $\mathbf{M_1}$ and $\mathbf{M_2}$

`det(M)` Determinate of matrix $\mathbf{M}$

`inv(M)` Matrix inverse of $\mathbf{M}$

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

`A\y` Least square solve over determined equation system using QR decomposition

`lscov(A, y, w)` Weighted least square solve over determined equation system using QR decomposition

`lscov(A, y, V)` General least square solve over determined equation system using QR decomposition

`[L, U, P] = lu(M)` LU decomposition of matrix $\mathbf{M}$

`R = chol(M)` Cholesky decomposition of matrix $\mathbf{M}$

`[Q, R] = qr(M)` QR decomposition of matrix $\mathbf{M}$

`[U, S, V] = svd(M)` Single value decomposition of matrix $\mathbf{M}$

`[V, D] = eig(A0)` Eigenvalue problem[2]: $\mathbf{A_0V} = \mathbf{VD}$

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

[2] `LinProp` uncertainty objects only

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

`dft(v)` Discrete Fourier transformation[2]

`idft(v)` Inverse discrete Fourier transformation[2]

`numerical_step(@func, x, dx)` Numerical step[2]

`optimizer(@func, xStart, p)` Optimizer[2]

4.4 Special routines

`LinProp2MCProp(x)` Converts LinProp objects to MCProp objects where

> `x` are the input LinProp objects.

`MCProp2LinProp(yMC, xMC, x)` Converts MCProp objects back to LinProp objects where

> `yMC` are the output MCProp objects,
> `xMC` are the input MCProp objects and
> `x` are the input LinProp objects.

Example of usage:

```
xMC = LinProp2MCProp(x)
yMC = f(xMC)
y = MCProp2LinProp(yMC, xMC, x)
```

The expected values of `y` are the same as the expected values of `yMC`. The covariance of `y` is the same as the covariance of `yMC`.

[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).

`get_jacobi2(y, x)` Computes the sensitivities of $\mathbf{y}$ to the intermediate results $\mathbf{x}$.

`get_unc_component(y, x)` Computes the uncertainty components of $\mathbf{y}$ with respect to $\mathbf{x}$.

`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 $\mathbf{y}$ to file.

`xml_file(y, filepath)` XML serializes uncertainty object $\mathbf{y}$ to file.

`xml_string(y)` XML serializes uncertainty object $\mathbf{y}$ 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.

[2] `LinProp` uncertainty objects only

A Physical constants

`unc.Const` [3] is equal to the newest physical constants `unc.Const2018`, see subsection A.3.

A.1 CODATA 2014

The following list contains the exact physical constants:

`unc.Const2014.deltavCs` Hyperfine transition frequency of Cs-133 in Hz

`unc.Const2014.c0` Speed of light in vacuum in m/s

`unc.Const2014.mu0` Vacuum magnetic permeability in Vs/Am

`unc.Const2014.ep0` Vacuum electric permittivity in As/Vm

`unc.Const2014.Kcd` Luminous efficacy in lm/W

`unc.Const2014.Mu` Molar mass constant in kg/mol

The following list contains the physical constants with uncertainties:

`unc.Const2014.G` Newtonian constant of gravitation[4] in $m^3/(kg*s^2)$

`unc.Const2014.alpha` Fine-structure constant[4]

`unc.Const2014.Ryd` Rydberg constant[4] in 1/m

`unc.Const2014.mpsme` Proton-electron mass ratio[4]

`unc.Const2014.Na` Avogadro constant[4] in 1/mol

`unc.Const2014.Kj` Josephson constant[4] in Hz/V

`unc.Const2014.k` Boltzmann constant[4] in J/K

`unc.Const2014.Rk` von Klitzing constant in Ohm

`unc.Const2014.e` Elementary charge in C

`unc.Const2014.h` Planck constant in Js

`unc.Const2014.me` Electron mass in kg

`unc.Const2014.mp` Proton mass in kg

`unc.Const2014.u` Atomic mass constant in kg

`unc.Const2014.F` Faraday constant in C/mol

`unc.Const2014.R` Molar gas constant in J/(mol*K)

`unc.Const2014.eV` Electron volt in J

[3] `unc.Const` cannot be used directly, because MATLAB is not supporting dot indexing for function handles. Therefore `unc` is just a place holder for `LinProp`, `DistProp` or `MCProp`, e.g.: `LinProp.Const.c0`.

[4] The correlation matrix of this physical constants is used in METAS UncLib to generate uncertainty objects which are correlated. The other physical constants are computed out of this set and the exact physical constants, e.g.: `Rk = mu0*c0/(2*alpha)` and `e = 2/(Kj*Rk)`.

A.2 CODATA 2014 for conventional electrical units 90

The following list contains the exact physical constants:

`unc.Const2014_90.deltavCs` Hyperfine transition frequency of Cs-133 in Hz

`unc.Const2014_90.c0` Speed of light in vacuum in m/s

`unc.Const2014_90.mu0` Vacuum magnetic permeability in Vs/Am

`unc.Const2014_90.ep0` Vacuum electric permittivity in As/Vm

`unc.Const2014_90.Kcd` Luminous efficacy in lm/W

`unc.Const2014_90.Mu` Molar mass constant in kg/mol

`unc.Const2014_90.Kj` Conventional value of Josephson constant in Hz/V

`unc.Const2014_90.Rk` Conventional value of von Klitzing constant in Ohm

`unc.Const2014_90.e` Elementary charge in C

`unc.Const2014_90.h` Planck constant in Js

The following list contains the physical constants with uncertainties:

`unc.Const2014_90.Na` Avogadro constant in 1/mol

`unc.Const2014_90.F` Faraday constant in C/mol

`unc.Const2014_90.k` Boltzmann constant in J/K

A.3 CODATA 2018

The following list contains the exact physical constants:

`unc.Const2018.deltavCs` Hyperfine transition frequency of Cs-133 in Hz

`unc.Const2018.c0` Speed of light in vacuum in m/s

`unc.Const2018.h` Planck constant in Js

`unc.Const2018.e` Elementary charge in C

`unc.Const2018.k` Boltzmann constant in J/K

`unc.Const2018.Na` Avogadro constant in 1/mol

`unc.Const2018.Kcd` Luminous efficacy in lm/W

`unc.Const2018.Kj` Josephson constant in Hz/V

`unc.Const2018.Rk` von Klitzing constant in Ohm

`unc.Const2018.F` Faraday constant in C/mol

`unc.Const2018.R` Molar gas constant in J/(mol*K)

`unc.Const2018.eV` Electron volt in J

The following list contains the physical constants with uncertainties:

`unc.Const2018.G` Newtonian constant of gravitation[5] in $m^3/(kg*s^2)$

`unc.Const2018.alpha` Fine-structure constant[5]

`unc.Const2018.mu0` Vacuum magnetic permeability in Vs/Am

`unc.Const2018.ep0` Vacuum electric permittivity in As/Vm

`unc.Const2018.Ryd` Rydberg constant[5] in 1/m

`unc.Const2018.me` Electron mass in kg

`unc.Const2018.are` Electron relative atomic mass[5]

`unc.Const2018.arp` Proton relative atomic mass[5]

`unc.Const2018.mpsme` Proton-electron mass ratio

`unc.Const2018.mp` Proton mass in kg

`unc.Const2018.u` Atomic mass constant in kg

`unc.Const2018.Mu` Molar mass constant in kg/mol

[5] The correlation matrix of this physical constants is used in METAS UncLib to generate uncertainty objects which are correlated. The other physical constants are computed out of this set and the exact physical constants, e.g.: `mu0 = 2*h/(e*e*c0)*alpha` and `ep0 = 1.0/(c0*c0*mu0)`.