Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Federal Institute of Metrology METAS

Introduction to METAS UncLib

Michael Wollensack

7. October 2021

Outline

Introduction

Problem

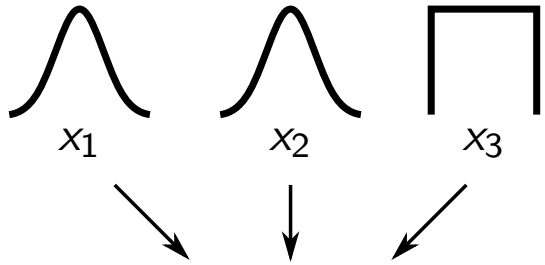
Computation of measurement uncertainties according to ISO-GUM.

Solution

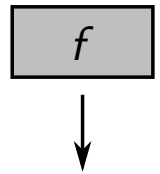
1. Identify all uncertainty influences.
2. Set up a measurement model.
3. Propagate all uncertainties through this model.

Uncertainty according to ISO-GUM (scalar)

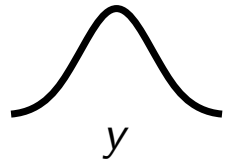
Input quantities

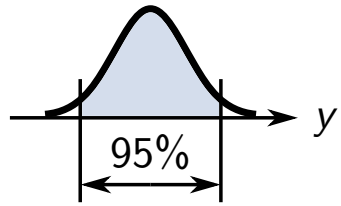


Measurement model



Output quantity





Linear uncertainty propagation

Input standard uncertainties $u(x_1)$, $u(x_2)$, $u(x_3)$

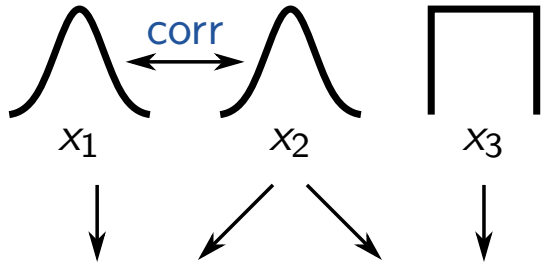
Sensitivities $\frac{\partial f}{\partial x_1}$, $\frac{\partial f}{\partial x_2}$, $\frac{\partial f}{\partial x_3}$

Output standard uncertainty $u^2(y) =$
$\left(\frac{\partial f}{\partial x_1}\right)^2 u^2(x_1) + \left(\frac{\partial f}{\partial x_2}\right)^2 u^2(x_2) + \left(\frac{\partial f}{\partial x_3}\right)^2 u^2(x_3)$

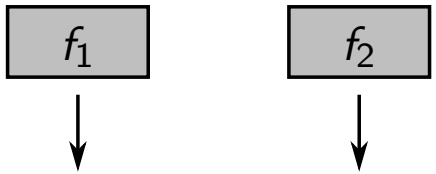
Expanded uncertainty $U^2(y) = (1.96)^2\, u^2(y)$

Uncertainty according to ISO-GUM (multivariate)

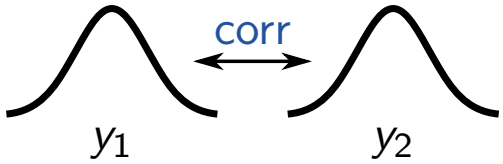
Input quantities

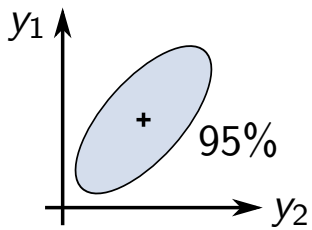


Measurement model



Output quantities





Linear uncertainty propagation

Covariance matrix $\mathbf{V}_X = \begin{bmatrix} u^2(x_1) & u(x_1,x_2) & u(x_1,x_3) \\ u(x_2,x_1) & u^2(x_2) & u(x_2,x_3) \\ u(x_3,x_1) & u(x_3,x_2) & u^2(x_3) \end{bmatrix}$

$$\mathrm{corr}(x_i, x_j) = \frac{u(x_i,x_j)}{u(x_i)\,u(x_j)}$$

Jacobian matrix $\mathbf{J}_{YX} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix}$

Output covariance matrix $\mathbf{V}_Y = \mathbf{J}_{YX}\mathbf{V}_X\mathbf{J}_{YX}{}'$

Correlations occur due to common influences

Expanded uncertainty $\mathbf{U}_Y = \chi^2_{N=2,p=0.95}\mathbf{V}_Y$

Derivative $f'(x) = \frac{d}{dx} f(x)$

The derivative $f'(x)$ of a function $y = f(x)$ of a real variable measures the sensitivity to change of the function value (output value y) with respect to a change in its argument (input value x).

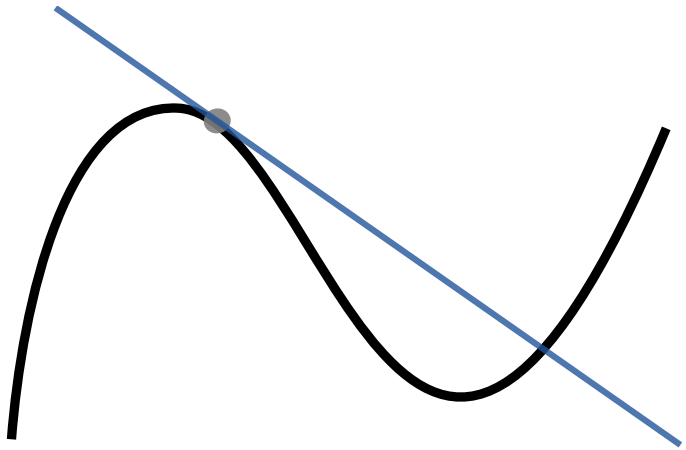


Figure: The graph of a function, drawn in black, and a tangent line to that function, drawn in blue. The slope of the tangent line is equal to the derivative of the function at the marked point.

The basic rules for derivatives

Derivatives $f'(x) = \frac{d}{dx} f(x)$

- $\frac{d}{dx} ax = a$
- $\frac{d}{dx} x^b = bx^{b-1}$
- $\frac{d}{dx} e^x = e^x$
- $\frac{d}{dx} \ln(x) = \frac{1}{x}$
- $\frac{d}{dx} \sin(x) = \cos(x)$
- $\frac{d}{dx} \cos(x) = -\sin(x)$

Partial derivatives $\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N)$

- $\frac{\partial}{\partial a}(a+b) = 1$, $\frac{\partial}{\partial b}(a+b) = 1$
- $\frac{\partial}{\partial a} ab = b$ and $\frac{\partial}{\partial b} ab = a$

Sum rule

$$(\alpha f + \beta g)' = \alpha f' + \beta g'$$

Chain rule

If $f(x) = h(g(x))$ then

$$f'(x) = h'(g(x)) \cdot g'(x)$$

If $z = h(y)$ and $y = g(x)$ then

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

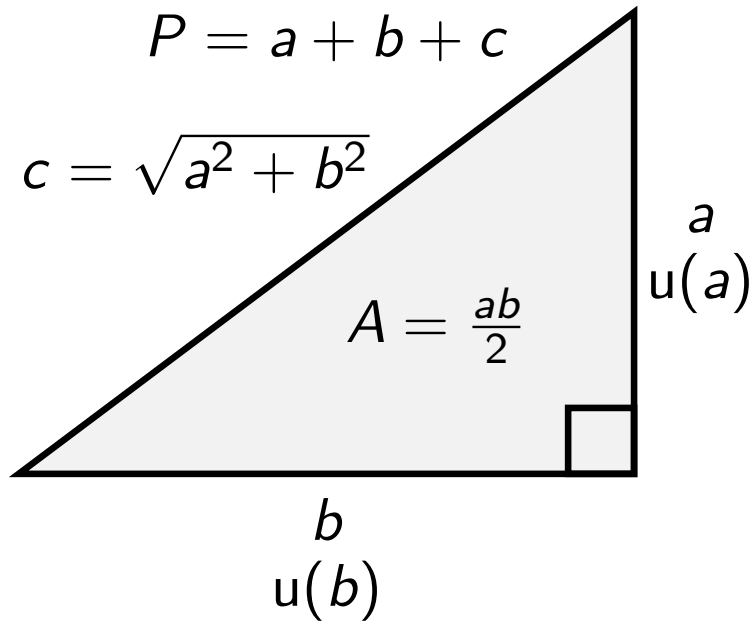
Multivariate Chain rule

$$\mathbf{J}_{ZX} = \mathbf{J}_{ZY} \mathbf{J}_{YX}$$

Right triangle example

- Cathetus $a = 3$, $\mathrm{u}(a) = 0.3$
- Cathetus $b = 4$, $\mathrm{u}(b) = 0.4$

- What's the value and unc of the hypotenuse c?
- What's the value and unc of the perimeter P?
- What's the value and unc of the area A?
- What's the correlation between P and A?



Solving the right triangle example by hand

Step	Function	Partial Derivatives	Chain Rule
1, 2	$a = 3,\ b = 4$		
3	$x_1 = a^2 = 9$	$\frac{\partial x_1}{\partial a} = 2a = 6$	
4	$x_2 = b^2 = 16$	$\frac{\partial x_2}{\partial b} = 2b = 8$	
5	$x_3 = x_1 + x_2 = 25$	$\frac{\partial x_3}{\partial x_1} = 1,\ \frac{\partial x_3}{\partial x_2} = 1$	$\cancel{\frac{\partial x_3}{\partial x_1}}\cancel{\frac{\partial x_1}{\partial a}} = 6,\ \frac{\partial x_3}{\partial b} = 8$
6	$c = \sqrt{x_3} = 5$	$\frac{\partial c}{\partial x_3} = \frac{1}{2\sqrt{x_3}} = 0.1$	$\frac{\partial c}{\partial a} = 0.6,\ \frac{\partial c}{\partial b} = 0.8$
7	$x_4 = a + b = 7$	$\frac{\partial x_4}{\partial a} = 1,\ \frac{\partial x_4}{\partial b} = 1$	
8	$P = x_4 + c = 12$	$\frac{\partial P}{\partial x_4} = 1,\ \frac{\partial P}{\partial c} = 1$	$\frac{\partial P}{\partial a} = 1.6,\ \frac{\partial P}{\partial b} = 1.8$
9	$x_5 = ab = 12$	$\frac{\partial x_5}{\partial a} = b,\ \frac{\partial x_5}{\partial b} = a$	
10	$A = \frac{x_5}{2} = 6$	$\frac{\partial A}{\partial x_5} = 0.5$	$\frac{\partial A}{\partial a} = 2,\ \frac{\partial A}{\partial b} = 1.5$

Solving the right triangle example by hand

11. Covariance matrix $\mathbf{V}_{in} = \begin{bmatrix} \mathrm{u}^2(a) & 0 \\ 0 & \mathrm{u}^2(b) \end{bmatrix} = \begin{bmatrix} 0.09 & 0 \\ 0 & 0.16 \end{bmatrix}$

12. Jacobian matrix $\mathbf{J} = \begin{bmatrix} \frac{\partial c}{\partial a} & \frac{\partial c}{\partial b} \\ \frac{\partial P}{\partial a} & \frac{\partial P}{\partial b} \\ \frac{\partial A}{\partial a} & \frac{\partial A}{\partial b} \end{bmatrix} = \begin{bmatrix} 0.6 & 0.8 \\ 1.6 & 1.8 \\ 2 & 1.5 \end{bmatrix}$

13. Output covariance matrix $\mathbf{V}_{out} = \mathbf{J}\mathbf{V}_{in}\mathbf{J}' =$
$$\begin{bmatrix} \mathrm{u}^2(c) & \mathrm{u}(c,P) & \mathrm{u}(c,A) \\ \mathrm{u}(c,P) & \mathrm{u}^2(P) & \mathrm{u}(P,A) \\ \mathrm{u}(c,A) & \mathrm{u}(P,A) & \mathrm{u}^2(A) \end{bmatrix} = \begin{bmatrix} 0.1348 & 0.3168 & 0.3000 \\ 0.3168 & 0.7488 & 0.7200 \\ 0.3000 & 0.7200 & 0.7200 \end{bmatrix}$$

Solving the right triangle example by hand

13. Output covariance matrix $\mathbf{V}_{out} = \mathbf{J}\mathbf{V}_{in}\mathbf{J}' =$

$$\begin{bmatrix} \mathrm{u}^2(c) & \mathrm{u}(c,P) & \mathrm{u}(c,A) \\ \mathrm{u}(c,P) & \mathrm{u}^2(P) & \mathrm{u}(P,A) \\ \mathrm{u}(c,A) & \mathrm{u}(P,A) & \mathrm{u}^2(A) \end{bmatrix} = \begin{bmatrix} 0.1348 & 0.3168 & 0.3000 \\ 0.3168 & 0.7488 & 0.7200 \\ 0.3000 & 0.7200 & 0.7200 \end{bmatrix}$$

14. What's the value and unc of the hypotenuse c?
    - $c = 5$ and $\mathrm{u}(c) = \sqrt{0.1348} = 0.3672$
15. What's the value and unc of the perimeter P?
    - $P = 12$ and $\mathrm{u}(P) = \sqrt{0.7488} = 0.8653$
16. What's the value and unc of the area A?
    - $A = 6$ and $\mathrm{u}(A) = \sqrt{0.7200} = 0.8485$
17. What's the correlation between P and A?
    - $\mathrm{corr}(P,A) = \frac{\mathrm{u}(P,A)}{\mathrm{u}(P)\,\mathrm{u}(A)} = \frac{0.7200}{\sqrt{0.7488}\sqrt{0.7200}} = 0.9806$

Solving the right triangle example using a PC

Which steps could be automated?

1. Decomposition of the measurement model into elementary functions. Implemented in any compiler or interpreter.
2. Partial derivatives of elementary functions $\frac{\partial}{\partial x_i} f(x_1, \ldots, x_N)$.
3. Sum $(\alpha f + \beta g)' = \alpha f' + \beta g'$ and Chain rule $\mathbf{J}_{ZX} = \mathbf{J}_{ZY}\mathbf{J}_{YX}$.
4. Uncertainty propagation $\mathbf{V}_{out} = \mathbf{J}\mathbf{V}_{in}\mathbf{J}'$.

Combination of steps 2 and 3 is called automatic differentiation. METAS UncLib implements steps 2 to 4 in the LinProp module.

Solving the right triangle example using MATLAB

```
%% Definition of the input uncertainty objects

unc = @LinProp;                          % linear unc propagation
a = unc(3.0, 0.3, 'a');                  % value 3, stdunc 0.3
b = unc(4.0, 0.4, 'b');                  % value 4, stdunc 0.4

%% Compute output quantities
c = sqrt(a.^2 + b.^2)                    % hypotenuse
P = a + b + c                            % perimeter
A = a.*b./2                              % area

m = get_correlation([P A]);              % correlation matrix
corrPA = m(2,1)                          % between P and A
```

```
--> c =  (5.0 +/- 0.367151)
--> P = (12.0 +/- 0.865332)
--> A =  (6.0 +/- 0.848528)
--> corrPA = 0.9806
```

Solving the right triangle example using Python

```
#  Definition of the input uncertainty objects
from metas_unclib import *          # import METAS UncLib
use_linprop()                       # linear unc propagation
a = ufloat(3.0, 0.3, desc='a')      # value 3, stdunc 0.3
b = ufloat(4.0, 0.4, desc='b')      # value 4, stdunc 0.4

#  Compute output quantities
c = umath.sqrt(a**2 + b**2)         # hypotenuse
P = a + b + c                       # perimeter
A = a * b / 2.0                     # area

m = get_correlation([P, A])         # correlation matrix
corrPA = m[1, 0]                    # between P and A
```

```
--> c =  (5.0 +/- 0.367151)
--> P = (12.0 +/- 0.865332)
--> A =  (6.0 +/- 0.848528)
--> corrPA = 0.9806
```

METAS UncLib

- METAS UncLib is a C# software library.
- It 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.
- Object-oriented implementation and overloaded operators hide the complexity from the user.

- It has been developed with Visual Studio 2008, 2013 and 2019.
- METAS UncLib V2.5 requires the .NET Framework V4.5.
- An Installer (msi-file) for METAS UncLib is available for free at www.metas.ch/unclib.

Propagation modes

METAS UncLib supports the following propagation modes:

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] preliminary implementation

LinProp uncertainty object

A LinProp uncertainty object has the following properties:

Value

$$z_i$$

Dependencies

$$\begin{bmatrix} x_1 & x_2 & x_3 & \dots \end{bmatrix}$$

Sensitivies

$$\begin{bmatrix} \frac{\partial z_i}{\partial x_1} & \frac{\partial z_i}{\partial x_2} & \frac{\partial z_i}{\partial x_3} & \dots \end{bmatrix}$$

- The value is simply calculated by performing the arithmetic or functional operation on the value(s) of the input object(s).
- Dependencies are updated by selecting the union of the dependencies of the inputs.
- Sensitivities are updated by applying the sum and chain rule.

Uncertainties are not stored in the objects, but can be calculated on demand.

Complex LinProp uncertainty object

A Complex LinProp object has the following fields:

Real part (LinProp object)

Value

z_{re}

Dependencies

$$\begin{bmatrix} x_1 & x_2 & x_3 & \dots \end{bmatrix}$$

Sensitivities

$$\begin{bmatrix} \frac{\partial z_{re}}{\partial x_1} & \frac{\partial z_{re}}{\partial x_2} & \frac{\partial z_{re}}{\partial x_3} & \dots \end{bmatrix}$$

Imag part (LinProp object)

Value

z_{im}

Dependencies

$$\begin{bmatrix} x_1 & x_2 & x_3 & \dots \end{bmatrix}$$

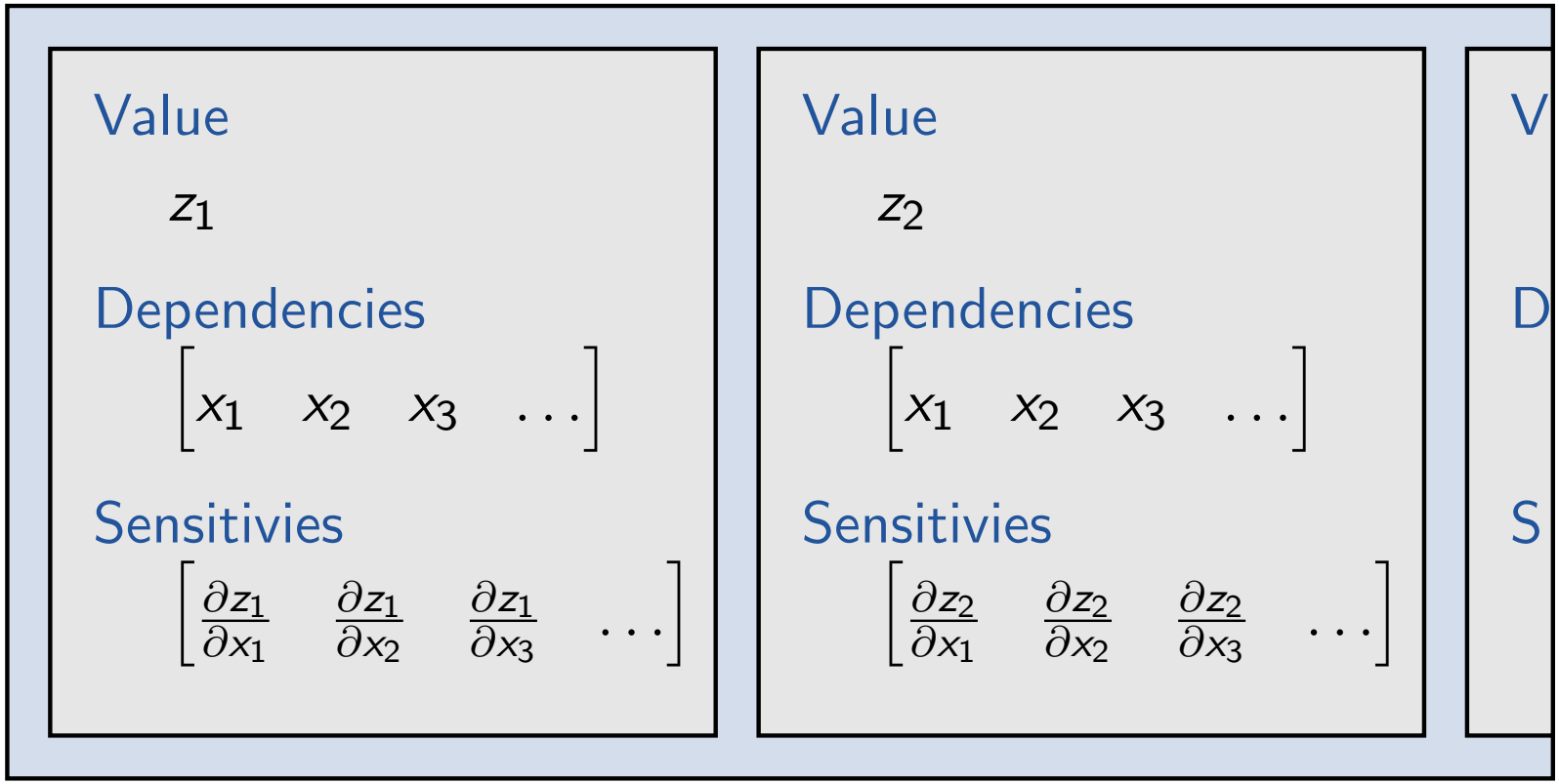
Sensitivities

$$\begin{bmatrix} \frac{\partial z_{im}}{\partial x_1} & \frac{\partial z_{im}}{\partial x_2} & \frac{\partial z_{im}}{\partial x_3} & \dots \end{bmatrix}$$

Complex math has been implemented for Complex LinProp objects.

Array of LinProp uncertainty objects

A vector or a matrix is an array of either LinProp or Complex LinProp uncertainty objects.



Linear algebra and numerical routines have been implemented.

MATLAB Wrapper

- METAS UncLib MATLAB is an extension to MATLAB.
- It 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).
- 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.
- The MATLAB wrapper, several examples and a tutorial are part of the METAS UncLib Installer (msi-file).

Global uncertainty settings

`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.

`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

Create an uncertainty object

`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: $2x2$

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

`unc([value], [covariance], (description))` Creates a new complex uncertain array. Covariance size: $2Nx2N$

Math functions

- `x + y`
- `x - y`
- `x.*y`
- `x./y`
- `x.^y`

- `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)`

Linear algebra

`M1*M2`	Matrix multiplication of matrix $\mathbf{M_1}$ and $\mathbf{M_2}$
`lu(M)`	LU decomposition of matrix $\mathbf{M}$
`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
`lscov(A, y, W)`	Weighted least square solve over determined equation system
`[V, D] = eig(A0)`	Eigenvalue problem [2]: $\mathbf{A_0V} = \mathbf{VD}$

[2] `LinProp` uncertainty objects only

Numerical routines

`polyfit(x, y, n)`	Fit polynom to data
`polyval(p, x)`	Evaluate polynom
`interpolation(x, y, n, xx)`	Interpolation
`fft(v)`	Fast Fourier transformation
`ifft(v)`	Inverse Fast Fourier transformation
`numerical_step(@func, x, dx)`	Numerical step[3]
`optimizer(@func, xStart, p)`	Optimizer[3]

[3] `LinProp` uncertainty objects only

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 properties of an uncertainty object cont.

The following methods are only available for `LinProp` uncertainty objects:

`get_idof(y)`	Computes the inverse degrees of freedom.
`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 **y** to the intermediate results **x**.
`get_unc_component(y, x)`	Computes the uncertainty components of **y** with respect to **x**.
`unc_budget(y)`	Shows the uncertainty budget.

Storage functions

Store a computed uncertainty object

`binary_file(y, filepath)`	Binary serializes **y** to file.
`xml_file(y, filepath)`	XML serializes **y** to file.
`xml_string(y)`	XML serializes **y** to string.

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.

Python Wrapper

- METAS UncLib Python is an extension to Python.
- It 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 (numpy, pythonnet).
- It requires the C# library METAS UncLib in the background.
- The classes `ufloat`, `ucomplex`, `ufloatarray` and `ucomplexarray` wrap METAS UncLib to Python over the .NET interface (pythonnet).
- The Python wrapper, several examples and a tutorial are part of the METAS UncLib Installer (msi-file) or the metas-unclib (pypi package).

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

Create an uncertainty object

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

`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).

`ucomplex(value, [covariance], desc=None)` Creates a new complex uncertain number. Covariance size: $2x2$

`ufloatarray([value], [covariance], desc=None)` Creates a new real uncertain array. Covariance size: NxN

`ucomplexarray([value], [covariance], desc=None)` Creates a new complex uncertain array. Covariance size: $2Nx2N$

Math functions - umath

- `x + y`
- `x - y`
- `x * y`
- `x / y`
- `x ** y`

- `umath.sqrt(x)`
- `umath.exp(x)`
- `umath.log(x)`
- `umath.log10(x)`

- `umath.pow(x, y)`

- `umath.sin(x)`
- `umath.cos(x)`
- `umath.tan(x)`
- `umath.asin(x)`
- `umath.acos(x)`
- `umath.atan(x)`
- `umath.sinh(x)`
- `umath.cosh(x)`
- `umath.tanh(x)`
- `umath.asinh(x)`
- `umath.acosh(x)`
- `umath.atanh(x)`

- `umath.real(x)`
- `umath.imag(x)`
- `umath.abs(x)`
- `umath.angle(x)`
- `umath.conj(x)`

Linear algebra - ulinalg

`dot(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 $\mathbf{M}$
`solve(A, Y)`	Solve linear equation system: $\mathbf{Ax} = \mathbf{y}$
`lstsqrsolve(A, Y)`	Least square solve over determined equation system
`weightedlstsqrsolve(A, y, W)`	Weighted least square solve over determined equation system
`V, D = eig(A0)`	Eigenvalue problem [4]: $\mathbf{A_0V} = \mathbf{VD}$

[4] `LinProp` uncertainty objects only

Numerical routines - unumlib

`polyfit(x, y, n)`	Fit polynom to data
`polyval(p, x)`	Evaluate polynom
`interpolation(x, y, n, xx)`	Interpolation
`fft(v)`	Fast Fourier transformation
`ifft(v)`	Inverse Fast Fourier transformation
`numerical_step(@func, x, dx)`	Numerical step[5]
`optimizer(@func, xStart, p)`	Optimizer[5]

[5] `LinProp` uncertainty objects only

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 properties of an uncertainty object cont.

The following methods are only available for `LinProp` uncertainty objects:

`get_idof(y)`	Computes the inverse degrees of freedom.
`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 **y** to the intermediate results **x**.
`get_unc_component(y, x)`	Computes the uncertainty components of **y** with respect to **x**.
`unc_budget(y)`	Shows the uncertainty budget.

Storage functions - ustorage

Store a computed uncertainty object

`save_binary_file(y, filepath)` Binary serializes **y** to file.

`save_xml_file(y, filepath)` XML serializes **y** to file.

`to_xml_string(y)` XML serializes **y** to string.

Reload a stored uncertainty object

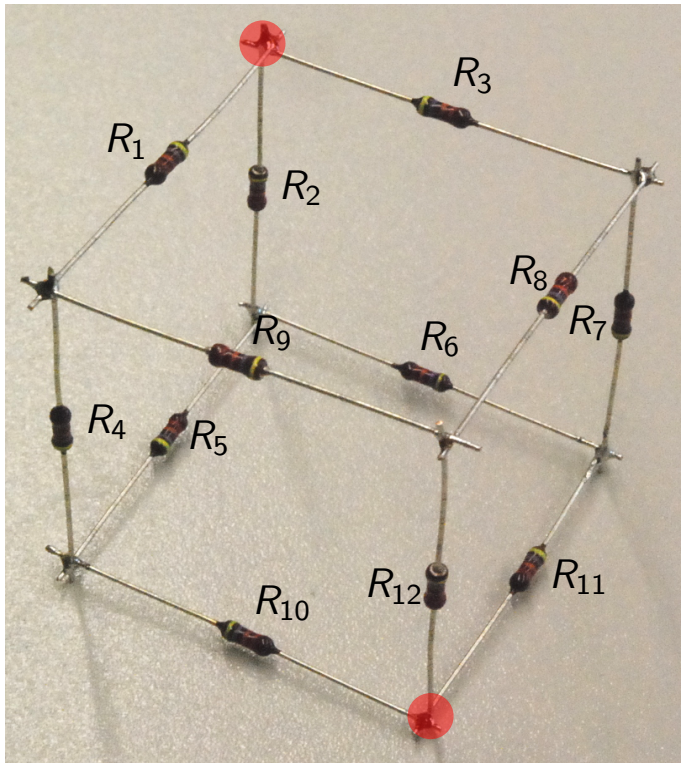
`load_binary_file(filepath)` Reloads uncertainty object from binary file.

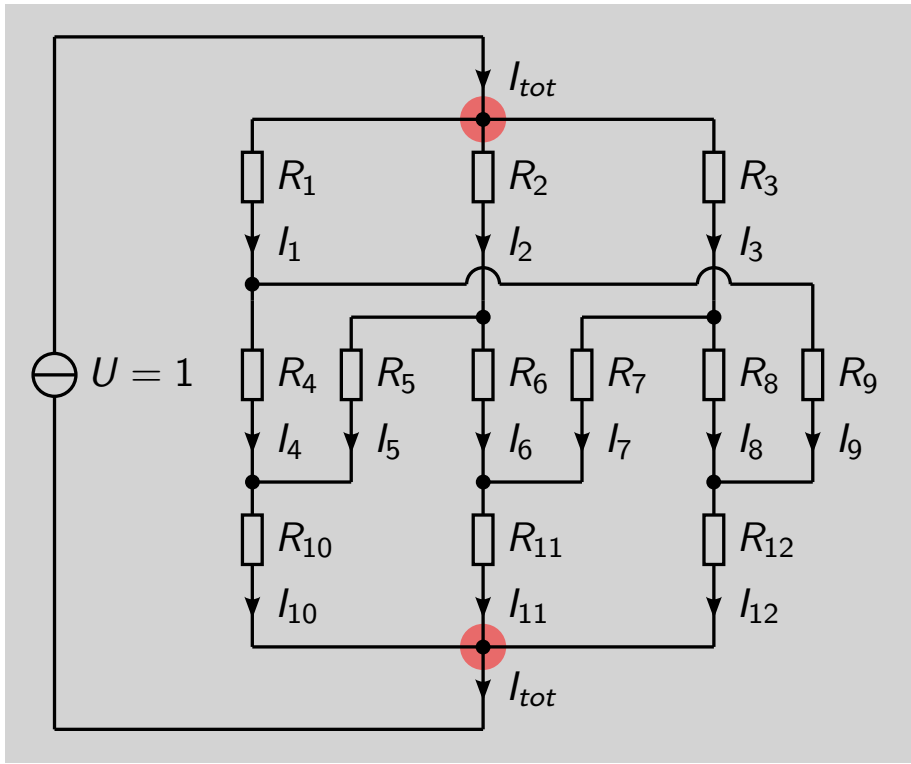
`load_xml_file(filepath)` Reloads uncertainty object from XML file.

`from_xml_string(s)` Reloads uncertainty object from XML string.

Resistor cube example

- What's the equivalent resistor of the cube between the two red junctions? And the uncertainty of it?
- $R_{eq} = \frac{U}{I_{tot}}$ with $I_{tot} = I_1 + I_2 + I_3$ and $\mathbf{I} = \mathbf{R}^{-1}\mathbf{U}$





Solving the resistor cube example using MATLAB

```
%% Definition of input unc objects R01 ... R12 and U

unc = @LinProp;                    % linear unc propagation

U   = 1.0;
R01 = unc(50.0, 0.1, 'R01');          % value 50.0 Ohm
R02 = unc(50.0, 0.1, 'R02');          % stdunc 0.1 Ohm
R03 = unc(50.0, 0.1, 'R03');
R04 = unc(50.0, 0.1, 'R04');
R05 = unc(50.0, 0.1, 'R05');
R06 = unc(50.0, 0.1, 'R06');
R07 = unc(50.0, 0.1, 'R07');
R08 = unc(50.0, 0.1, 'R08');
R09 = unc(50.0, 0.1, 'R09');
R10 = unc(50.0, 0.1, 'R10');
R11 = unc(50.0, 0.1, 'R11');
R12 = unc(50.0, 0.1, 'R12');
```

Solving the resistor cube example using MATLAB

```
%% Kirchhoff's circuit laws --> linear equation system

Ux = [0 0 0 0 0 0 U U U U U U]';


Rx = [-1   0   0   1   0   0   0   0   1   0   0   0 ;...
       0  -1   0   0   1   1   0   0   0   0   0   0 ;...
       0   0  -1   0   0   0   1   1   0   0   0   0 ;...
       0   0   0   1   1   0   0   0   0  -1   0   0 ;...
       0   0   0   0   0   1   1   0   0   0  -1   0 ;...
       0   0   0   0   0   0   0   1   1   0   0  -1 ;...
     R01   0   0  R04   0   0   0   0   0  R10   0   0 ;...
     R01   0   0   0   0   0   0   0  R09   0   0  R12;...
       0  R02   0   0   0  R06   0   0   0   0  R11   0 ;...
       0  R02   0   0  R05   0   0   0   0  R10   0   0 ;...
       0   0  R03   0   0   0   0  R08   0   0   0  R12;...
       0   0  R03   0   0   0  R07   0   0   0  R11   0 ];
```

Solving the resistor cube example using MATLAB

```
%% Solve linear equation system

Ix = Rx\Ux;


%% Compute equivalent resistor of the cube

Itot = Ix(1) + Ix(2) + Ix(3);
R = U/Itot
```

```
--> R = (41.6667 +/- 0.0280542)
```

Solving the resistor cube example using Python

```
#  Definition of input unc objects R01 ... R12 and U
from metas_unclib import *  # import METAS UncLib
use_linprop()               # linear unc propagation

U   = 1.0
R01 = ufloat(50.0, 0.1, desc='R01') # value 50.0 Ohm
R02 = ufloat(50.0, 0.1, desc='R02') # stdunc 0.1 Ohm
R03 = ufloat(50.0, 0.1, desc='R03')
R04 = ufloat(50.0, 0.1, desc='R04')
R05 = ufloat(50.0, 0.1, desc='R05')
R06 = ufloat(50.0, 0.1, desc='R06')
R07 = ufloat(50.0, 0.1, desc='R07')
R08 = ufloat(50.0, 0.1, desc='R08')
R09 = ufloat(50.0, 0.1, desc='R09')
R10 = ufloat(50.0, 0.1, desc='R10')
R11 = ufloat(50.0, 0.1, desc='R11')
R12 = ufloat(50.0, 0.1, desc='R12')
```

Solving the resistor cube example using Python

```
#  Kirchhoff's circuit laws --> linear equation system

Ux = np.array([0, 0, 0, 0, 0, 0, U, U, U, U, U, U])

Rx = np.array(
     [[-1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 ],
      [ 0 ,-1 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ],
      [ 0 , 0 ,-1 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0 ],
      [ 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0 ,-1 , 0 , 0 ],
      [ 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 ,-1 , 0 ],
      [ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 ,-1 ],
      [R01, 0 , 0 ,R04, 0 , 0 , 0 , 0 , 0 ,R10, 0 , 0 ],
      [R01, 0 , 0 , 0 , 0 , 0 , 0 , 0 ,R09, 0 , 0 ,R12],
      [ 0 ,R02, 0 , 0 , 0 ,R06, 0 , 0 , 0 , 0 ,R11, 0 ],
      [ 0 ,R02, 0 , 0 ,R05, 0 , 0 , 0 , 0 ,R10, 0 , 0 ],
      [ 0 , 0 ,R03, 0 , 0 , 0 , 0 ,R08, 0 , 0 , 0 ,R12],
      [ 0 , 0 ,R03, 0 , 0 , 0 ,R07, 0 , 0 , 0 ,R11, 0 ]])
```

Solving the resistor cube example using Python

```
#  Solve linear equation system

Ix = ulinalg.solve(Rx, Ux)


#  Compute equivalent resistor of the cube

Itot = Ix[0] + Ix[1] + Ix[2]
R = U/Itot
```

```
--> R = (41.6667 +/- 0.0280542)
```

Conclusion

METAS UncLib

- Generic software library for propagation of uncertainties.
- Is able to handle complex-valued and multivariate quantities.
- Can be used from C# (Visual Studio), MATLAB or Python.
- Used in many labs at METAS and around the world.
- Used in large applications like METAS VNA Tools.
- Available for free since 2009 at www.metas.ch/unclib.

Validation

- Only derivatives of elementary functions are programmed. Elementary functions are easy to check.
- Overloaded operators hide the complexity from the user.
- User has the possibility to check if the unc budget is plausible.