# Punpy Documentation

***Release 1.0.0***

**CoMet Toolkit Team**

**Apr 02, 2024**

# FOR USERS


**1 Acknowledgements** **3**

**2 Project status** **5**
- 2.1 Getting Started . . . 5
- 2.2 User Guide . . . 7
- 2.3 Algorithm Theoretical Basis . . . 23
- 2.4 API reference . . . 27

**Index** **67**

The **punpy** module is a Python software package to propagate random, structured and systematic uncertainties through a given measurement function.

**punpy** can be used as a standalone tool, where the input uncertainties are inputted manually. Alternatively, **punpy** can also be used in combination with digital effects tables created with **obsarray**. This documentation provides general information on how to use the module (with some examples), as well as a detailed API of the included classes and function.

Quickstart Guide New to *punpy*? Check out the quickstart guide for an introduction.

User Guide The user guide provides a documentation and examples how to use **punpy** either standalone or in combination with *obsarray* digital effects tables.

ATBD ATBD mathematical description of **punpy** (under development).

API Reference The API Reference contains a description of each of the **punpy** classes and functions.

CHAPTER

ONE

# ACKNOWLEDGEMENTS

**punpy** has been developed by Pieter De Vis.

The development has been funded by:

- The UK's Department for Business, Energy and Industrial Strategy's (BEIS) National Measurement System (NMS) programme
- The IDEAS-QA4EO project funded by the European Space Agency.

CHAPTER

# TWO

# PROJECT STATUS

**punpy** is under active development. It is beta software.

## 2.1 Getting Started

### 2.1.1 Dependencies

Punpy has the following dependencies:

- Python (3.7 or above)
- numpy
- scipy
- emcee
- comet_maths
- obsarray

### 2.1.2 Installation

The easiest way to install punpy is using pip:

```
$ pip install punpy
```

Ideally, it is recommended to do this inside a virtual environment (e.g. conda).

Alternatively, for the latest development version, first go to the folder where you want to save punpy and clone the project repository from GitLab:

```
$ git clone git@github.com:comet-toolkit/punpy.git
```

Then go into the created directory and install the module with pip:

```
$ cd punpy
$ pip install -e .
```

### 2.1.3 Example Usage

For examples on how to use punpy either as a standalone package or with digital effects tables, we refer to the examples section on the CoMet Website. There some jupyter notebooks (hosted on google colab) are available with examples.

Below, we show an example for using punpy standalone for illustration purposes. For more complete examples with more detailed explanations, we refer to the CoMet website examples.

In this example, we use punpy to propagate uncertainties through a calibration process:

```
import punpy
import numpy as np

# your measurement function
def calibrate(L0,gains,dark):
   return (L0-dark)*gains

# your data
L0 = np.array([0.43,0.8,0.7,0.65,0.9])
dark = np.array([0.05,0.03,0.04,0.05,0.06])
gains = np.array([23,26,28,29,31])

# your uncertainties
L0_ur = L0*0.05  # 5% random uncertainty
dark_ur = np.array([0.01,0.002,0.006,0.002,0.015])  # random uncertainty

gains_ur = np.array([0.5,0.7,0.6,0.4,0.1])  # random uncertainty
gains_us = np.array([0.1,0.2,0.1,0.4,0.3])  # systematic uncertainty
# (different for each band but fully correlated)
gains_utemp = gains*0.03

corr_temp = []

prop=punpy.MCPropagation(10000)
L1=calibrate(L0,gains,dark)
L1_ur=prop.propagate_random(calibrate,[L0,gains,dark],
      [L0_ur,gains_ur,dark_ur])
L1_us=prop.propagate_systematic(calibrate,[L0,gains,dark],
      [L0_us,gains_us,np.zeros(5)])
L1_ut=(L1_ur**2+L1_us**2)**0.5
L1_cov=punpy.convert_corr_to_cov(np.eye(len(L1_ur)),L1_ur)+\
       punpy.convert_corr_to_cov(np.ones((len(L1_us),len(L1_us))),L1_ur)

print(L1)
print(L1_ur)
print(L1_us)
print(L1_ut)
print(L1_cov)
```

## 2.2 User Guide

In this user guide, you will find detailed descriptions and examples that describe many common tasks that you can accomplish with **punpy**.

### 2.2.1 Punpy as a standalone package

In this section we give a short overview of some of the key capabilities of punpy for propagating uncertainties through a measurement function. When using punpy as a standalone tool, the input quantities and their uncertainties are manually specified. The code in this section is just as illustration and we refer to the the CoMet website examples for examples with all required information for running punpy. The punpy package can propagate various types of uncertainty through any python measurement function. In the next subsection we discuss how these python measurement functions are defined.

#### Measurement functions

The measurement function can be written mathematically as:

$$y = f\left(x_i, \ldots, x_N\right)$$

where:

- $f$ is the measurment function;
- $y$ is the measurand;
- $x_i$ are the input quantities.

The measurand and input quantities are often vectors consisting of multiple numbers. E.g. in spectroscopy, the input quantities and measurand each have measurements for multiple wavelengths. These wavelengths are the same for the input quantities and the measurand. We refer to the *Algorithm Theoretical Basis* section for more details on the vocabulary used and the various types of uncertainty.

When using punpy, the measurement function can be any python function that takes a number of input quantities as arguments (as floats or numpy arrays of any shape) and returns a measurand (as a float or numpy array of any shape). For example:

```
def measurement_function(x1,x2,x3):
   y=x1+x2-x3 # here any real measurement function can be implemented
   return y
```

Any function that can be formatted to take only numpy arrays or floats/ints as its inputs and only returns one or more numpy arrays and or floats/ints can be used in punpy. Note that this measurement function can optionally be part of a class, which can allow to do an additional setup step to set further, non-numerical parameters:

```
class Func:
   def __init__(self,option1,option2):
      self.option1 = option1
      self.option2 = option2

   def measurement_function(self,x1,x2,x3):
      if self.option1=="green":
         y=x1+x2-x3 # here any real measurement function can be implemented
      else:
```

(continues on next page)

(continued from previous page)

```
        y=self.option2
    return y
```

### Monte Carlo and Law of Propagation of Uncertainty methods

Once this kind of measurement function is defined, we can use the various punpy methods to propagate uncertainties though this measurement function. In order to do this, we first create a prop object (object of punpy MCPropagation of LPUPropagation class):

```
import punpy

prop=punpy.MCPropagation(10000) # Here the number is how
# many MC samples will be generated

# Or if you have a measurement function that does not accept
# higher dimensional arrays as argument:
prop=punpy.MCPropagation(10000,parallel_cores=1)

# Alternatively it is possible to use LPU methods to
# propagate uncertainties
prop=punpy.LPUPropagation()
```

In order to do propagate uncertainties, punpy uses Monte Carlo (MC) methods (see Section *Monte Carlo Method*) or Law of Propagation of Uncertainties (LPU) methods (see Section *Law of Propagation of Uncertainty Method*). MC methods generate MC samples from the input quantities (which can be correlated to eachother or not), and then propagate these samples through the measurement function. The LPU methods implement the law of propagation of uncertainties from the GUM (Guide to the Expression of Uncertainty in Measurement) by calculating the Jacobian and using this to propagate the uncertainties.

For the MC method, the number of MC samples that is used is set as the first argument when creating the MCPropagation object (see example above). Two approaches can be followed to propagate the MC samples of the input quantities to the measurand, depending on whether the measurement function can be applied to numpy arrays of arbitrary size.

The most computationally efficient way is to pass an array consisting of all MC steps of an input quantity instead of the input quantity themselves. Each of the input quantities will thus get an additional dimension in the MC sample. If the measurement function can deal with these higher dimensional arrays by just performing numpy operations, this gives the most computationally efficient MC propagation. This method is the default in punpy (which corresponds to setting the optional `parallel_cores` keyword to 0).

However, this is not the case for every measurement function. If the inputs to the measurement function are less flexible, and don't support additional dimensions, it is possible to instead run the MC samples one by one. In order to pass each MC sample individually to the measurement function, it is possible to set the optional `parallel_cores` keyword to 1. In *Managing memory and processing speed in punpy*, we will show how the same keyword can be used to do parallel processing for such measurement functions.

For the LPU methods, the numdifftools package (used within comet_maths) is used to calculate the Jacobian. This package automatically determines the stepsize in the numerical differentiation, unless a manual stepsize is set. For some measurement functions, it can be necessary to set a manual stepsize (e.g. because of the limited range of the input quantities). It is possible to set the stepsize to be passed to the numdifftools jacobian method by setting the `step` keyword when creating the propagation object:

```
prop = punpy.LPUPropagation(step=0.01)
```

Both the MC and LPU options also have the `verbose` keyword to set the verbosity, which allows the user to get additional information such as the time the runs took and additional warnings:

```
prop=punpy.MCPropagation(10000,verbose=True)
```

### Propagating random and systematic uncertainties

Once a prop object has been made (see previous sections), a number of methods can be used to propagate uncertainties, depending on the kind of uncertainties that need to be propagated. We start by showing how to propagating random and systematic uncertainties. For random uncertainties, the errors associated with these uncertainties are entirely independent of each-other (errors for different elements of the input quantity are uncorrelated). For systematic uncertainties, the errors of the different elements (along the different dimensions of the input quantity) are entirely correlated. This typically means they are all affected by the same effect (e.g. if the calibration gain of a sensor is wrong, all its measurements will be wrong by the same factor).

For given values (arrays or numbers) of the input quantities xn, and their random (ur_xn) or systematic (us_xn) uncertainties, punpy can be used to propagate these uncertainties as follows:

```
y = measurement_function(x1, x2, x3)
ur_y = prop.propagate_random(measurement_function,
       [x1, x2, x3], [ur_x1, ur_x2, ur_x3])
us_y = prop.propagate_systematic(measurement_function,
       [x1, x2, x3], [us_x1, us_x2, us_x3])
```

In addition to returning the uncertainties, punpy can also be used to return the correlation matrix. This is not particularly useful when the input quantities all have a random or all have a systematic error correlation as in this section, but is very relevant when dealing with input quantities that have other error correlations (see next section). This is done by setting the `return_corr` keyword to True:

```
ur_y, corr_y = prop.propagate_random(measurement_function,
       [x1, x2, x3], [ur_x1, ur_x2, ur_x3], return_corr=True)
```

Here, the returned error correlation matrix will cover all dimensions of the associated uncertainty. If ur_y has shape (k,l), corr_y has shape (k*l,k*l). The order of the correlation coefficient elements corresponds to the order for a flattened version of ur_y (ur_y.flatten()). By default, the returned correlation matrices are forced to be positive-definite. This means they are sometimes slightly changed in order to accomodate this. If this is the case, a warning is shown. To just return the unmodified correlation matrices, it is possible to set the `PD_corr` keyword to False.

### Propagating uncertainties when measurements are correlated (within input quantity)

Sometimes the elements of an input quantity xn are not simply independent (random uncertainties) or fully correlated (systematic uncertainty), but rather something in between. In this case, it is possible to specify an error-correlation matrix between the different elements (at different coordinates/indices) of the input quantity, which gives the correlation coefficient between the errors for different elements within the input quantities. If the input quantity is one-dimensional of size (k), the error correlation matrix will be a matrix of shape (k,k). If the input quantity is two dimensional of size (k,l), the error correlation matrix will be of size (k*l,k*l).

This error correlation matrix cannot be calculated from the uncertainties themselves (it is not the correlation between the values of the uncertainties or something like that) but instead required knowledge of how the measurements were done and the sensors used. It is a matrix that needs to be provided. Fur more detailed information on error correlation matrices, we refer to the Guide to the expression of uncertainty in measurement and the FIDUCEO website.

If such an error-correlation matrix corr_xn is known for every xn, punpy can use them to propage the combined uncertainty:

```
uc_y, corrc_y = prop.propagate_standard(measurement_function,
                [x1, x2, x3], [us_x1, us_x2, us_x3], corr_x=[corr_x1, corr_x2, corr_x3])
```

Here the corr_xn can either be a square array with the appropriate error-correlation coefficients, or alternatively the string "rand" or "syst" for random and systematic error correlations respectively. In the case of random or systematic error-correlations, the error correlation matrices are always the same (diagonal matrix of ones and full matrix of ones for random and systematic respectively), and the string is thus sufficient to define the complete error correlation matrix. Note that these error-correlation matrices can also optionally be passed to the propagate_random() and propagate_systematic() functions. In this case, the only difference with propagate_standard, is that in case no error_correlation matrix is provided (i.e. None), the error correlation matrix defaults to the random or systematic case. The following four expressions are entirely equivalent:

```
uc_y, corrc_y = prop.propagate_standard(measurement_function,
                 [x1, x2, x3], [us_x1, us_x2, us_x3], corr_x=[np.eye(us_x1.flatten()),␣
↪corr_x2, np.eye(us_x1.flatten())])
uc_y, corrc_y = prop.propagate_standard(measurement_function,
                 [x1, x2, x3], [us_x1, us_x2, us_x3], corr_x=["rand", corr_x2, "rand"])
uc_y, corrc_y = prop.propagate_random(measurement_function,
                 [x1, x2, x3], [us_x1, us_x2, us_x3], corr_x=[np.eye(us_x1.flatten()),␣
↪corr_x2, np.eye(us_x1.flatten())], return_corr=True)
uc_y, corrc_y = prop.propagate_random(measurement_function,
                 [x1, x2, x3], [us_x1, us_x2, us_x3], corr_x=[None, corr_x2, None],␣
↪return_corr=True)
```

Instead of working with error-correlation matrices, it is also possible to use error covariance matrices. It is straightforward to convert between error correlation and covariance matrices using comet_maths:

```
import comet_maths as cm
cov_x1 = cm.convert_corr_to_cov(corr_x1, us_x1)

#and back to corr and uncertainty:
corr_x1 = cm.convert_cov_to_corr(cov_x1, us_x1)
corr_x1 = cm.correlation_from_covariance(cov_x1)
us_x1 = cm.uncertainty_from_covariance(cov_x1)
```

Using covariance matrices, the uncertainties can be propagated using:

```
uc_y, corr_y = prop.propagate_cov(measurement_function, [x1, x2, x3],
                [cov_x1, cov_x2, cov_x3])
```

If required, the resulting measurand correlation matrix can easily be converted to a covariance matrix as:

```
cov_y = cm.convert_corr_to_cov(corr_y, u_y)
```

Note that propagate_standard() and propagate_cov() by default return the correlation matrix, yet propagate_random() and propagate_systematic() return only the uncertainties on the measurand (because the correlation matrices are trivial in their standard use case). However all these functions have an optional `return_corr` argument that can be used to define whether the correlation matrix should be returned.

### Propagating uncertainties when input quantities are correlated (between different input quantities)

In addition to the elements within an input quantity being correlated, it is also possible the input quantities are correlated to eachother. If this is the case, this functionality can be included in each of the functions specified above by giving an argument to the optional keyword `corr_between`. This keyword needs to be set to the correlation matrix between the input quantities, and thus needs to have the appropriate shape (e.g. 3 x 3 array for 3 input quantities):

```
ur_y = prop.propagate_random(measurement_function, [x1, x2, x3],
       [ur_x1, ur_x2, ur_x3], corr_between = corr_x1x2x3)
uc_y, corr_y = prop.propagate_cov(measurement_function, [x1, x2, x3],
               [cov_x1, cov_x2, cov_x3], corr_between = corr_x1x2x3)
```

### More advanced punpy input and output

When returning the error correlation functions, rather than providing this full correlation matrix, it is also possible to get punpy to only calculate the error correlation matrix along one (or a list of) dimensions. If `return_corr` is set to True, the keyword `corr_dims` can be used to indicate the dimension(s) for which the correlation should be calculated. In this case the correlation coefficients are calculated using only the first index along all dimensions other than `corr_dims`. We note that `corr_dims` should only be used when the error correlation matrices do not vary along the other dimension(s). A warning is raised if any element of the correlation matrix varies by more than 0.05 between using only the first index along all dimensions other than `corr_dims` or using only the last index along all dimensions other than `corr_dims`. The `corr_dims` keyword can be passed to any of the propagate_* functions:

```
ur_y, corr_y = prop.propagate_random(measurement_function,
       [x1, x2, x3], [ur_x1, ur_x2, ur_x3], return_corr=True, corr_dims=0)
```

If ur_y again had shape (k,l), corr_y would now have shape (k,k).

For the MC method, it is also possible to return the generated samples by setting the optional `return_samples` keyword to True:

```
prop = punpy.MCPropagation(10000)
ur_y, samplesr_y, samplesr_x = prop.propagate_random(
measurement_function, [x1, x2, x3], [ur_x1, ur_x2, ur_x3],
corr_between=corr_x1x2x3, return_samples=True)

ub_y, corr_y, samplesr_y, samplesr_x = prop.propagate_systematic(
measurement_function, [x1, x2, x3], [us_x1, us_x2, us_x3],
return_corr=True, return_samples=True)
```

It is also possible to pass a sample of input quantities rather than generating a new MC sample. This way, the exact same sample can be used as n a previous run, or one can generate a sample manually:

```
ub_y, corr_y = prop.propagate_systematic(
measurement_function, [x1, x2, x3], [us_x1, us_x2, us_x3],
return_corr=True, samples=samplesr_x)
```

For the LPU method, it is possible to additionally return the calculated Jacobian matrix by setting the `return_Jacobian` keyword to True. In addition, instead of calculating the Jacobian as part of the propagation, it is also possible to give a precomputed Jacobian matrix, by setting the `Jx` keyword. This allows to use the Jacobian matrix from a previous step or an analytical prescription, which results in much faster processing:

```
prop = punpy.LPUPropagation()
ur_y, Jac_x = prop.propagate_random(
measurement_function, [x1, x2, x3], [ur_x1, ur_x2, ur_x3],
corr_between=corr_x1x2x3, return_Jacobian=True)

ub_y, corr_y = prop.propagate_systematic(
measurement_function, [x1, x2, x3], [us_x1, us_x2, us_x3],
return_corr=True, Jx=Jac_x)
```

It is not uncommon to have measurment functions that take a number of input quantities, where each input quantity is a vector or array. If the measurand and each of the input quantities all have the same shape, and the measurement function is applied independently to each element in these arrays, then most of the elements in the Jacobian will be zero (all except the diagonal elements for each square Jacobian matrix corresponding to each input quantity individually). Rather than calculating all these zeros, it is possible to set the `Jx_diag` keyword to True which will automatically ignore all the off-diagonal elements and result in faster processing:

```
prop = punpy.LPUPropagation()
ub_y, corr_y = prop.propagate_systematic(
measurement_function, [x1, x2, x3], [us_x1, us_x2, us_x3],
return_corr=True, Jx_diag=True)
```

### Multiple measurands

In some cases, the measurement function has multiple outputs (measurands):

```
def measurement_function(x1,x2,x3):
   y1=x1+x2-x3 # here any real measurement function can be implemented
   y2=x1-x2+x3 # here any real measurement function can be implemented
   return y1,y2
```

These functions can still be handled by punpy, but require the `output_vars` keyword to be set to the number of outputs:

```
us_y, corr_y, corr_out = prop.propagate_systematic(measurement_function,
                         [x1, x2, x3], [us_x1, us_x2, us_x3],
                         return_corr=True, corr_dims=0, output_vars=2)
```

Note that now the `corr_dims` keyword is set to a list with the corr_dims for each output variable, and there is an additional output corr_out, which gives the correlation between the different output variables (in the above case a 2 by 2 matrix). Here the correlation coefficients between the 2 variables are averaged over all measurements.

When there are multiple output variables, it is also possible to specify separate corr_dims for each measurand. This is done by setting the `separate_corr_dims` keyword to True, and passing a list of corr_dims:

```
us_y, corr_y, corr_out = prop.propagate_systematic(measurement_function,
                         [x1, x2, x3], [us_x1, us_x2, us_x3],
                         return_corr=True, corr_dims=[0,1],separate_corr_dims=True,␣
↪output_vars=2)
```

It is also possible to set one of the separate `corr_dims` to None if you do not want the error correlation to be calculated for that measurand. In that case None will be returned (as corr_y[1] in below example):

```
us_y, corr_y, corr_out = prop.propagate_systematic(measurement_function,
                         [x1, x2, x3], [us_x1, us_x2, us_x3],
```

(continues on next page)

(continued from previous page)

```
                        return_corr=True, corr_dims=[0,None],separate_corr_dims=True,␣
↪output_vars=2)
```

### Different Probability Density Functions

The standard probability density function in punpy is a Gaussian distribution. This means the generated MC samples will follow a gaussian distribution with the input quantity values as mean and uncertainties as standard deviation. However other probabilty density functions are also possible. Currently there are two additional options (with more to follow in the future).

The first alternative is a truncated Gaussian distribution. This distribution is just like the Gaussian one, except that there are no values outside a given minimum or maximum value. A typical use case of this distribution is when a certain input quantity can never be negative. In such a case the uncertainty propagation could be done like this:

```
ur_y = prop.propagate_random(measurement_function, [x1, x2, x3],
       [ur_x1, ur_x2, ur_x3], corr_between = corr_x1x2x3, pdf_shape="truncated_gaussian",
↪ pdf_params={"min":0.})
```

When the alternative probability density functions require additional parameters, these can be passed in the optional pdf_params dictionary. For the truncated Gaussian example, this dictionary can contain a value for "min" and "max" for the minimum and maximum allowed values respectively.

The second alternative is a tophat distribution. In this case the MC sample will have a uniform probabilty distribution from the value of the input quantity minus its uncertainty to the value of the input quantity plus its uncertainty. We note that for these modified probability density functions, the standard deviation of the MC sample is not the same as the uncertainty anymore.

### Handling errors in the measurement function

There are cases where a measurement function occasionally runs into an error (e.g. if certain specific combinations of input quantities generated by the MC approach are invalid). This can completely mess up a long run even if it happens only occasionally. In cases where the measurement function does not raise an error but returns a measurand which has only non-finite values (np.nan or np.inf as one of the values), that MC sample of the measurand will automatically be ignored by punpy and not used when calculating the uncertainties or any of the other outputs.

In cases where an error is raised, one can catch this error using a try statement and instead return np.nan. punpy will ignore all these nan's in the measurand MC sample, and will just calculate uncertainties and its other output without these nan samples:

```
import numpy as np
def function_fail(x1, x2):
   zero_or_one=np.random.choice([0,1],p=[0.1,0.9])
   with np.errstate(divide='raise'):
      try:
        return 2 * x1 - x2/zero_or_one
      except:
        return np.nan

prop = punpy.MCPropagation(1000)
ur_y = prop.propagate_random(function_fail, [x1, x2], [ur_x1, ur_x2])
```

Here the measurement will fail about 10% of the time (by raising a FloatingPointError due to division by zero). The resulting sample of valid measurands will thus have about 900 samples, which should still be enough to calculate the uncertainties.

By default, numpy will only ignore MC samples where all the values are non-finite. However, it is also possible to ignore all MC samples where any of the values are non-finite. This can be done by setting the `allow_some_nans` keyword to False.

#### Shape of input quanties within the measurement function

When setting parallel_cores to 1 or more, the shape of the input quantities used for each iteration in the measurement function matches the shape of the input quantities themselves. However, when parallel_cores is set to 0, all iterations will be processed simultaneously and there will be an additional dimension for the MC iterations. Generally within punpy, the MC dimension in the samples is the first one (i.e. internally as well as when MC samples are returned). However, when processing all iterations simultaniously, in most cases it is more practical to have the MC dimension as the last dimension. This is because we use numpy arrays and these are compatible when the last dimensions match following the numpy broadcasting rules. So as default, the shape of the input quantities when using parallel_cores will have the MC iterations as its last dimension. However, in some cases it is more helpful to have the MC iterations as the first dimension. If this is the case, the MC iteration dimension can be made the first dimension by setting the `MClastdim` keyword to False:

```
prop = punpy.MCPropagation(1000,MCdimlast=False)
```

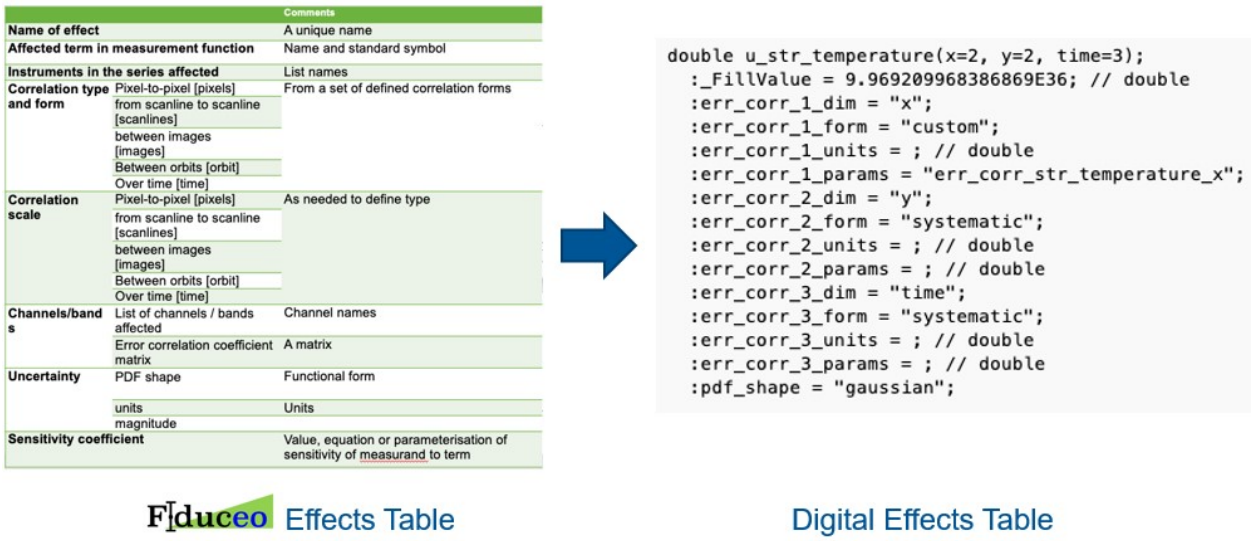
### 2.2.2 punpy in combination with digital effects tables

In this section we explain how punpy can be used for propagating uncertainties in digital effects tables through a measurement function. For details on how to create these digital effects tables, we refer to the obsarray documentation. Once the digital effects tables are created, this is the most concise method for propagating uncertainties. The code in this section is just as illustration and we refer to the the CoMet website examples for example with all requied information for running punpy. The punpy package can propagate the various types of correlated uncertainties that can be stored in digital effects tables through a given measurement function. In the next subsection we discuss how these measurement functions need to be defined in order to use the digital effects tables.

#### Digital Effects Tables

Digital Effects tables are created with the obsarray package. The documentation for obsarray is the reference for digital effect tables. Here we summarise the main concepts in order to give context to the rest of the Section.

Digital effects tables are a digital version of the effects tables created as part of the FIDUCEO project. Both FIDUCEO effects tables and digital effects tables store information on the uncertainties on a given variable, as well as its error-correlation information (see Figure below). The error-correlation information often needs to be specified along multiple different dimensions. For each of these dimensions (or for combinations of them), the correlation structure needs to be defined. This can be done using an error-correlation matrix, or using the FIDUCEO correlation forms. These FIDUCEO correlation forms essentially provide a parametrisation of the error correlation matrix using a few parameters rather than a full matrix. These thus require much less memory and are typically the preferred option (though this is not always possible as not all error-correlation matrices can be parameterised in this way). Some correlation forms, such as e.g. “random” and “systematic” do not require any additional parameters. Others, such as “triangle_relative”, require a parameter that e.g. sets the number of pixels/scanlines being averaged.



| | | Comments |
|---|---|---|
| **Name of effect** | | A unique name |
| **Affected term in measurement function** | | Name and standard symbol |
| **Instruments in the series affected** | | List names |
| **Correlation type and form** | Pixel-to-pixel [pixels] | From a set of defined correlation forms |
| | from scanline to scanline [scanlines] | |
| | between images [images] | |
| | Between orbits [orbit] | |
| | Over time [time] | |
| **Correlation scale** | Pixel-to-pixel [pixels] | As needed to define type |
| | from scanline to scanline [scanlines] | |
| | between images [images] | |
| | Between orbits [orbit] | |
| | Over time [time] | |
| **Channels/bands** | List of channels / bands affected | Channel names |
| | Error correlation coefficient matrix | A matrix |
| **Uncertainty** | PDF shape | Functional form |
| | units | Units |
| | magnitude | |
| **Sensitivity coefficient** | | Value, equation or parameterisation of sensitivity of measurand to term |

```
double u_str_temperature(x=2, y=2, time=3);
  :_FillValue = 9.969209968386869E36; // double
  :err_corr_1_dim = "x";
  :err_corr_1_form = "custom";
  :err_corr_1_units = ; // double
  :err_corr_1_params = "err_corr_str_temperature_x";
  :err_corr_2_dim = "y";
  :err_corr_2_form = "systematic";
  :err_corr_2_units = ; // double
  :err_corr_2_params = ; // double
  :err_corr_3_dim = "time";
  :err_corr_3_form = "systematic";
  :err_corr_3_units = ; // double
  :err_corr_3_params = ; // double
  :pdf_shape = "gaussian";
```



*Figure 1 - left: FIDUCEO effects table template. right: obsarray digital effects table for one uncertainty component.*

The obsarray package which implements the digital effects tables, extends the commonly used xarray package. xarray objects typically have multiple variables with data defined on multiple dimensions and with attributes specifying additional information. In digital effects tables, each of these variables has one (or more) uncertainty variables associated with it. Each of these uncertainty components is clearly linked to its associated variable, and has the same dimensions. These uncertainty components, unsurprisingly, have uncertainties as the data values. As attributes, they have the information defining the error-correlation structure. If the FIDUCEO correlation forms can be used, the form name and optionally its parameters are stored directly in the attributes of the uncertainty component. If the FIDUCEO correlation forms cannot be used, the form in the attributes is listed as "err_corr_matrix" and as parameter it has the name of another variable in the xarray dataset that has the correlation matrix.

Multiple uncertainty components can be added for the same data variable, and obsarray provide functionality to combine these uncertainties, either as the total uncertainties for a given variable, or as separate random, systematic, and structured components.

### Measurement Function

Generally, a measurement function can be written mathematically as:

$$y = f\left(x_i, \ldots, x_N\right)$$

where:

- $f$ is the measurment function;
- $y$ is the measurand;
- $x_i$ are the input quantities.

The measurand and input quantities are often vectors consisting of multiple numbers. Here, we choose an example of an ideal gas law equivalent:

$$V = \frac{8.314 \times nT}{P}$$

where:

- $V$ is the volume;
- $n$ is the amount of substance (number of moles);

- $T$ is the temperature;
- $P$ is the pressure.

Here $V$ is now the measurand, and $n$, $T$ and $P$ are the input quantities. Digital effects tables for $n$, $T$ and $P$ will thus need to be specified prior, and punpy will create a digital effects table for $V$ as output.

In order to be able to do the uncertainty propagation with these digital effects tables, the measurement functions now need to be defined within a subclass of the MeasurementFunction class provided by punpy. In this subclass, one can then define the measurement function in python as a function called "function":

```
from punpy import MeasurementFunction

class GasLaw(MeasurementFunction):
   def function(self, P, T, n):
      return (8.134 * n * T)/P
```

In some cases, it can also be useful to define the measurand name and input quantity names directly in this class:

```
from punpy import MeasurementFunction

class GasLaw(MeasurementFunction):
   def function(self, P, T, n):
      return (8.134 * n * T)/P

   def get_measurand_name_and_unit(self):
      return "volume", "m^3"

   def get_argument_names(self):
      return ["pressure", "temperature", "n_moles"]
```

These names will be used as variable names in the digital effects tables. This means they have to match the expected names in e.g. the input digital effects tables that are used. Providing the names of the input quantities and measurand in this way is not a requirement, as this information can also be provided when initialising the object of this class.

### Propagating uncertainties in digital effects tables

Once this kind of measurement function class is defined, we can initialise an object of this class. In principle there are no required arguments when creating an object of this class (all arguments have a default). However, in practise we will almost always provide at least some arguments. The first argument `prop` allows to pass a MCPropagation or LPUpropagaion object. It thus specifies whether the Monte Carlo (MC) method (see Section *Monte Carlo Method*) or Law of Propagation of Uncertainties (LPU) method (see Section *Law of Propagation of Uncertainty Method*) should be used. These prop objects can be created with any of their options (such as parallel_cores):

```
prop = MCPropagation(1000, dtype="float32", verbose=False, parallel_cores=4)

gl = IdealGasLaw(prop=prop)
```

If no argument is provided for prop, a MCPropagation(100,parallel_cores=0) object is used. The next arguments are for providing the input quantity names and the measurand name and measurand unit respectively:

```
gl = IdealGasLaw(prop=prop, xvariables=["pressure", "temperature", "n_moles"], yvariable=
↪"volume", yunit="m^3")
```

In the `xvariables` argument, one needs to specify the names of each of the input quantities. These names have to be in the same order as in the specified function, and need to correspond to the names used for the variables in the

digital effects tables. These variable names can be provided as optional arguments here, or alternatively using the get_argument_names() function in the class definition. If both options are provided, they are compared and an error is raised if they are different.

Similarly, the `yvariable` gives the name of the measurand (or list of names if multiple measurands are returned by measurement function) and `yunit` specifies its associated unit(s). Alternatively, these can also be provided using the get_measurand_name_and_unit() function in the class definition (they will be cross-checked if both are provided). There are many more optional keywords that can be set to finetune the processing of the uncertainty propagation. These will be discussed in the *Options when creating MeasurementFunction object* section.

Once this object is created, and a digital effects table has been provided (here as a NetCDF file), the uncertainties can be propagated easily:

```
import xarray as xr
ds_x1 = xr.open_dataset("digital_effects_table_gaslaw.nc")
ds_y = gl.propagate_ds(ds_x1)
```

This generates a digital effects table for the measurand, which could optionally be saved as a NetCDF file, or passed to the next stage of the processing. The measurand effects table will have separate contributions for the random, systematic and structured uncertainties, which can easily be combined into a single covariance matrix using the obsarray functionalities of the digital effects tables. It is quite common that not all the uncertainty information is available in a single digital effects table. In such cases, multiple digital effects tables can simply be provided to "propagate_ds". punpy will then search each of these effects tables for the input quantities provided when initialising the MeasurementFunction object. For example, if $n$, $T$ and $P$, each had their own digital effects tables, these could be propagated as:

```
import xarray as xr
ds_nmol = xr.open_dataset("n_moles.nc")
ds_temp = xr.open_dataset("temperature.nc")
ds_pres = xr.open_dataset("pressure.nc")
ds_y = gl.propagate_ds(ds_pres, ds_nmol, ds_temp)
```

These digital effects tables can be provided in any order. They can also contain numerous other quantities that are not relevant for the current measurement function. When multiple of these digital effects tables have a variable with the same name (which is used in the measurement function), an error is raised.

### functions for propagating uncertainties

In the above example, we show an example of using the propagate_ds() function to obtain a measurand effects table that has separate contributions for the random, systematic and structured uncertainties. Depending on what uncertainty components one is interested in, there are a number of functions that can be used: * propagate_ds: measurand digital effects table with separate contributions for the random, systematic and structured uncertainties. * propagate_ds_tot: measurand digital effects table with one combined contribution for the total uncertainty (and error correlation matrix). * propagate_ds_specific: measurand digital effects table with separate contributions for a list of named uncertainty contributions provided by the user. * propagate_ds_all: measurand digital effects table with separate contributions for all the individual uncertainty contributions in the input quantities in the provided input digital effects tables.

It is worth noting that the uncertainty components labelled in the measurand digital effect tables as "random" or "systematic" (either in propagate_ds, propagate_ds_specific or propagate_ds_all), will contain the propagated uncertainties for all uncertainty components on the input quantities that are random or systematic respectively along all the measurand dimensions. Any uncertainty components on the input quantities where this is not the case (e.g. because the error correlation along one dimension is random and along another is systematic; or because one of the error correlations is provided as a numerical error correlation matrix) will be propagated to the structured uncertainty components on the measurand.

This is somewhat further complicated by the fact that the input quantity dimensions are not always the same as the measurand dimensions. If any of the measurand dimensions is not in the input quantity dimensions, some assumption

needs to made about how this input quantity will be correlated along that measurand dimension. Often, such a situation will simply mean that the same value of the input quantity will be used for every index along the measurand dimension (broadcasting). This often leads to a systematic correlation along this measurand dimension (a typical example would be the same spectral gains being applied to multiple spectral scans in a measurement, where the gains have a wavelength dimension and the spectral scans have wavelength and scan index dimensions; any error in the gains, will affect all scans equally). There are however also scenarios where the introduced error-correlation along the measurand dimension should be random (e.g. if a constant temperature is assumed and applied along the time dimension, but we know in reality the temperature is fluctuating randomly w.r.t. to assumed temperature). It can also be structured. Detailed understanding of the problem is thus required when the measurand dimensions are not present along the measurand dimensions. By default, punpy assumes that the error correlation along the missing dimensions is systematic. If another error correlation is required, this can be done by setting the `expand` keyword to True and the `broadcast_correlation` to the appropriate error correlation (either "rand", "syst" or an error correlation matrix as a numpy array). Depending on how this broadcast error correlation combines with the error correlations in the other dimensions, can also affect which measurand uncertainty component (random, systematic or structured) it contributes to when using propagate_ds.

Sometimes one wants to propagate uncertainties one input quantity at a time. This can be the case no matter if we are propagating total uncertainties or individual components. When creating the MeasurementFunction object, it is possible to specify on which input quantities the uncertainties should be propagated using the `uncxvariables` keyword:

```
gl = IdealGasLaw(prop=prop,
                 xvariables=["pressure", "temperature", "n_moles"],
                 uncxvariables=["pressure"]
                 yvariable="volume",
                 yunit="m^3")
ds_y = gl.propagate_ds(ds_pres, ds_nmol, ds_temp)
```

In the above example, only the uncertainties on pressure will be propagated. This behaviour could also be obtained by removing the unc_comps in the temperature and n_moles variables in their respective datasets, but the solution shown above is easier. If no uncxvariables are provided, the uncertainties on all input quantities are propagated.

### Options when creating MeasurementFunction object

A number of additional options are available when creating the MeasurementFunction object, and when running one of the propagate_ds functions. We refer to the API for a full list of the keywords, but here highlight some of the ones that were not previously explained.

When creating the MeasurementFunction object, we previously discussed the `prop`, `xvariables`, `uncxvariables`, `yvariable` and `yunit` keywords. Next, there are a number of keywords that are the same as the keywords for using punpy as standalone. These are `corr_between`,`param_fixed`, `repeat_dims`, `corr_dims`, `seperate_corr_dims`, `allow_some_nans`. Here these keywords work in the same way as for standalone punpy and we refer to the *Punpy as a standalone package* Section for further explanation. The one difference is that here, the repeat_dims and corr_dims can be provided as dimension names rather than dimension indices (dimension indices are also still allowed). If a string corr_dim is provided that is present in some but not all of the measurand dimensions (only relevant when there are multiple differently-shaped measurands), the `seperate_corr_dims` will automatically be set to True, and the appropriate separate corr_dims will be worked out automatically.

The options we have not previously explained are the `ydims`, `refxvar` and `sizes_dict`. These all have to do with the handling of dimensions when they differ between input quantities (or between input quantities and measurand). In the typical punpy usecase, the dimensions of the measurand are the same as the dimensions of the input quantities. If this is not the case, the `ydims` keyword should be set to a list of the measurand dimensions (in order matching the shape). If one of these dimensions is not in the input quantities, one should also provide `sizes_dict`, which is a dictionary with all dimension names as keys, and the dimension size as the value. Alternatively, if the dimensions of the measurand match the dimensions of one (but not all) of the input quantities, the measurnad shape can be automatically set if `refxvar` is provided, where `refxvar` is the name of the input quantity with matching shape.

Finally the `use_err_corr_dict` is explained in the *Managing memory and processing speed in punpy* Section.

#### Options when running propagate_ds functions

There are also a few options when running the propagate_ds (or similar) functions. The `store_unc_percent` keyword simply indicates whether the measurand uncertainties should be stored in percent or in the measurand units (the latter is the default). The `expand` keyword indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand, prior to passing to the measurement function (defaults to False).

`ds_out_pre` allows to provide a pre-generated xarray dataset (typically made using obsarray) in which the results will be stored. This can be used to add additional variables to the dataset prior to running the uncertainty propagation, or to concatenate multiple uncertainty propagation results into one file. By default, the measurand variables and associated uncertainty and error correlation will be overwritten, but all other variables in the dataset remain. If one does not want to get punpy to work out the error correlations iteself, but just use the ones in the template (e.g. because in complex cases, punpy can run into problems), this can be specified by setting the `use_ds_out_pre_unmodified` keyword to True. In this case, only the values of the variables will be changed, but none of the attributes.

Finally the `include_corr` keyword can be set to False if error correlations should be omited from the calculation. The latter results in faster processing but can lead to wrong results so should be used with caution.

### 2.2.3 Managing memory and processing speed in punpy

#### Storing error correlation matrices separately per dimension

Error correlation matrices typically take up a lot of memory as they give the error correlation coefficients between each pair of measurements. The MC samples themselves can also take up a lot of memory when the number of samples is large. One easy way to limit the amount of memory used is by setting the `dtype` of the MC samples (and resulting uncertainties and error correlation matrices):

```
prop = MCPropagation(20000, dtype=np.float32)
```

Random and systematic uncertainty components take up relatively little space, as each of their error correlation dimensions are by defnition parameterised as random or systematic. For structured components with error correlation matrices stored as separate variables, it is not uncommon for these matrices to take up a lot of memory. This is especially the case when each of the dimensions is not parametrised separately, and instead an error correlation matrix is provided along the combination of angles. E.g. for a variable with dimensions (x,y,z), which correspond to a shape of e.g. (20,30,40), the resulting total error correlation matrix will have shape (20*30*40,20*30*40) which would contain 575 million elements. The shape chosen here as an example is quite moderate, so it is clear this could be an issue when using larger datasets.

The solution to this is to avoid storing the full (x*y*z,x*y*z) error correlation matrix when possible. In many cases, even though the errors for pixels along a certain dimension (e.g. x) might be correlated, this error correlation w.r.t x does not change for different values of y or z. In that case, the error correlation for x can be separated and stored as a matrix of shape (x,x).

One way to reduce the memory load is by separately storing the error-correlation matrices in the output dataset. In the *Punpy as a standalone package* section, we showed that the `corr_dims` keyword can be used to output the error correlation matrix for a single dimension rather than the full error correlation matrix. This can also be used to separately store the error correlation for each dimension by passing a list of all dimension indices for `corr_dims`:

```
ur_y, corr_y = prop.propagate_random(measurement_function,
       [x1, x2, x3], [ur_x1, ur_x2, ur_x3], return_corr=True, corr_dims=[0,1])
```

where now corr_y will have two elements, the first of which will be a (k,k) shaped matrix and the second one a (l,l) matrix. Saving the error correlations in this manner often required less memory than storing the full error correlation matrix. However, because of the averaging over the other dimension(s) when calculating these one-dimensional errror correlation matrices, some information in lost. In fact, this approach should only be done when the error correlation

matrices do not vary along the other dimension(s). Whether this method can be applied thus depends on the measurement function itself and it should be used with caution (a good rule of thumb to decide whether this approach can be used is whether the measurement function could be applied without passing data along the averaged dimension(s) at once). There are cases where the measurement function can only partially be decomposed (e.g. a measurement function where the first nd second dimensions of the measurand have some complex correlation, but the third dimension can easily be separated out as effectively the calculation could be done entirely separate for each index along this third dimension). In such cases, the corr_dims keyword for the dimensions that cannot be separted can be given a string with the dimension indices separated by a dot:

```
ur_y, corr_y = prop.propagate_random(measurement_function,
        [x1, x2, x3], [ur_x1, ur_x2, ur_x3], return_corr=True, corr_dims=["0.1",2])
```

Here, if the measurand (and ur_y) are of shape (k,l,m), corr_y will have two elements. The first element will be of shape (k*l,k*l) and the second element of shape (m,m).

We note that for the digital effects table use case, when creating the MeasurementFunction object, it is possible to provide the `corr_dims` keyword as strings with the dimension names rather than dimension indices (both options work). When using dimension names as strings, they can still be separated by a dot to indicate combined error correlation matrices in the outputs:

```
gl = GasLaw(prop, ["n_moles", "temperature", "pressure"], corr_dims=["x.y","time"])
```

### Using error correlation dictionaries

In addition to the large error correlation matrices in the output, another memory issue comes from the calculation of the error-correlation for each of the input quantities (which often have the same dimensions). When using punpy as standalone, one can pass the error correlation for separate dimensions using a dictionary:

```
ufb, ucorrb = prop.propagate_systematic(
    functionb,
    xsb,
    xerrsb,
    corr_x=[{"0":np.eye(len(xerrb)),"1":"syst"}, "syst"],
    return_corr=True,
)
```

This avoids having to pass the correlation matrix as one large array. When using digital effects tables with punpy, the use of these correlation dictionaries can be handled internally. This can be achieved by setting the use_err_corr_dict keyword:

```
gl = IdealGasLaw(
   prop=prop,
   xvariables=["pressure", "temperature", "n_moles"],
   yvariable="volume",
   yunit="m^3",
   use_err_corr_dict=True,
)
```

### Input quantities with repeated measurements along one axis

In general, random uncertainties are uncorrelated between repeated measurements, and systematic uncertainties are fully correlated between repeated measurements. If the input quantities are arrays and no further information is specified, punpy assumes that all the different values in the array are repeated measurements, and the correlation between the values is treated accordingly.

However, it is also possible that the arrays provided in the input quantities have multiple dimensions, one of which is for repeated measurements, and one is another dimension. E.g. when propagating uncertainties in spectra, often one of the input quantities is a 2D array where along one dimension there are repeated measurements and along another there are different wavelengths. In this case the `repeat_dims` keyword can be set to an integer indicating which dimension has repeated measurements. When the `repeat_dims` keyword is set, punpy also splits the calculations and does them separately per repeated measurement. This reduces the memory requirements and as a result speeds up the calculations. It is however possible that not all of the input quantities have repeated measurements. E.g. one of the input quantities could be an array of three calibration coefficients, whereas another input quantity is an array with repeated spectral measurements which are being calibrated. If the `repeat_dims` keyword does not apply to one of the input quantities, this can be specified by the `param_fixed` keyword. This keyword then needs to be set to a list of bools where each bool indicates whether the corresponding input quantity should remain fixed (True) or should be split along repeat_dims (False).

If x1, x2, us_x1, us_x2 are all arrays with shape (n_wav,n_repeats) where n_wav is the number of wavelengths and n_repeats is the number of repeated measurements, and x3 is an array with some calibration coefficients (with uncertainties u_x3):

```
import numpy as np

corr_wav_x1= np.eye(len(wavelengths))  # This is a diagonal (i.e.
# uncorrelated) correlation matrix with shape (n_wav,n_wav) where
# n_wav is the number of wavelengths.

corr_wav_x2= np.ones((len(wavelengths),len(wavelengths))  # This is
# a correlation matrix of ones (i.e. fully correlated) with shape
#(n_wav,n_wav) where n_wav is the number of wavelengths.

corr_wav_x3= None  # When set to None, the correlation between
# wavelength defaults to the same as the correlation between repeated
# wavelengths (i.e. fully correlated for propagate_systematic()).

param_fixed_x1x2x3 = [False,False,True] # indicates that x1 and x2
# have repeated measurements along repeat_dims and calculations will
# be split up accordingly, and x3 will remain fixed and not split up
# (x3 does not have the right shape to be split up)

us_y, corr_y = prop.propagate_systematic(measurement_function,
               [x1, x2, x3], [us_x1, us_x2, us_x3],
               corr_x=[corr_wav_x1,corr_wav_x2,corr_wav_x3],
               param_fixed=param_fixed_x1x2x3, fixed return_corr=True,
               repeat_dims=1, corr_dims=0)
```

Here only one matrix is returned for corr_y with a shape matching the provided corr_dims, rather than a correlation matrix per repeated measurement. The matrices for each repeated measurement have been averaged. We note that if no corr_dims are set, the default option is to return a combined error correlation matrix for all dimensions that are not in repeat_dims.

In some cases, when there are multiple measurands with different shapes, it is not clear what dimension the repeat_dim refers to. In such cases, the `refyvar` keyword should be set to the index of the measurand with the most dimensions

and the repeat_dims indexes should correspond to this measurand.

### Processing the MC samples in parallel

At the start of this section we already saw that the optional `parallel_cores` keyword can be used to running the MC samples one-by-one through the measurement function rather than all at once as in the standard case. It is also possible to use the same keyword to use parallel processing. Here, only the processing of the input quantities through the measurement function is done in parallel. Generating the samples and calculating the covariance matrix etc is still done as normal. Punpy uses the multiprocessing module which comes standard with your python distribution. The gain by using parallel processing only really outweighs the overhead if the measurement function is relatively slow (of the order of 0.1 s or slower for one set of input quantities).

Parallel processing for MC can be done as follows:

```
if __name__ == "__main__":
   prop = punpy.MCPropagation(10000,parallel_cores=4)
   ur_y = prop.propagate_random(measurement_function, [x1, x2, x3],
          [ur_x1, ur_x2, ur_x3])
   us_y = prop.propagate_systematic(measurement_function, [x1, x2, x3],
          [us_x1, us_x2, us_x3])
```

Note that the use of 'if __name__ == "__main__":' is required when using a Windows machine for multiprocessing and is generally good practise. When processing in parallel, child processes are generated from the parent code, and the above statement is necessary in Windows to avoid the child processes to generate children themselves. Everything using the results of the multiprocessing needs to be inside the 'if __name__ == "__main__"'. However the measurement function itself needs to be outside this since the child processes need to find this.

One other important aspect is that in order for the parallel processing to work, the measurement function cannot be a normal function of a class. It can however be a static function of a class. This means that if we want to do parallel processing for a measurement function in a punpy MeasurementFunction class in order to use digital effects tables, we need to define it as a static function:

```
# Define your measurement function inside a subclass of MeasurementFunction
class IdealGasLaw(MeasurementFunction):
    @staticmethod
    def meas_function(pres, temp, n):
        return (n * temp * 8.134) / pres
```

Measurement function for which multiprocessing can be used can thus not have self as their first argument.

For the LPU method, it is also possible to use parallel processing, though only if the `repeat_dims` keyword is set. In this case each of the repeated measurements is processed in parallel:

```
if __name__ == "__main__":
   prop = punpy.LPUPropagation(parallel_cores=4)
   ur_y = prop.propagate_random(measurement_function, [x1, x2, x3],
          [ur_x1, ur_x2, ur_x3],repeat_dims=0)
   us_y = prop.propagate_systematic(measurement_function, [x1, x2, x3],
          [us_x1, us_x2, us_x3],repeat_dims=0)
```

#### Separating MC propagation in different stages

In some cases, it is necessary to run a large MC sample but the measurement function requires too much memory to run all the MC samples in one single run. In such cases it is possible to break up the punpy processing in differnet stages. Generally, there are 4 stages: - Generating the MC sample of the input quantities. - Running these samples through the measurement function. - Combining the MC samples of measurands. - Processing the MC measurand sample to produce the required outputs (sush as uncertainties and error correlation matrices).

In code, this looks like:

```
MC_x = prop.generate_MC_sample(xsd, xerrsd, corrd)
MC_y1 = prop.run_samples(functiond, MC_x, output_vars=2, start=0, end=10000)
MC_y2 = prop.run_samples(functiond, MC_x, output_vars=2, start=10000, end=20000)
MC_y = prop.combine_samples([MC_y1, MC_y2])

ufd, ucorrd, corr_out = prop.process_samples(
   MC_x, MC_y, return_corr=True, corr_dims=0, output_vars=2
)
```

Here the run has been broken up into two seperate calls to run the samples, which can be controlled by specifying the start and end indices of the MC sample of input quantities (i.e. which MC iterations should be processed by this call). This can be broken up into any number of samples. The runnning of these samples through the measurand can even be distributed on different computers. The different measurand samples could then simply be stored in files, before bringing them all together and analysing the combined measurand MC sample. This also allows detailed controll (e.g. quality checks) on the measurand MC samples, prior to processing the samples.

#### Additional options

For both MC and LPU methods there are some cases, when there is only one correlation matrix contributing to the measurand (e.g. a complicated measurement function where all but one of the input quantities are known with perfect precision, i.e. without uncertainty), it can be beneficial to just copy this correlation matrix to the measurand rather than calculating it (since copying is faster and does not introduce MC noise). When the `fixed_corr_var` is set to True, punpy automatically detects if there is only one term of uncertainty, and if so copies the relevant correlation matrix to the output instead of calculating it. If `fixed_corr_var` is set to an integer, the correlation matrix corresponding to that dimension is copied without any checks:

```
prop = punpy.MCPropagation(10000)
ur_y = prop.propagate_random(
measurement_function, [x1, x2, x3], [ur_x1, ur_x2, ur_x3],
corr_between=corr_x1x2x3, fixed_corr_var=True)
```

## 2.3 Algorithm Theoretical Basis

### 2.3.1 Principles of Uncertainty Analysis

The Guide to the expression of Uncertainty in Measurement (GUM 2008) provides a framework for how to determine and express the uncertainty of the measured value of a given measurand (the quantity which is being measured). The International Vocabulary of Metrology (VIM 2008) defines measurement uncertainty as:

> *"a non-negative parameter characterizing the dispersion of the quantity values being attributed to a measurand, based on the information used."*

The standard uncertainty is the measurement uncertainty expressed as a standard deviation. Please note this is a separate concept to measurement error, which is also defined in the VIM as:

> *"the measured quantity value minus a reference quantity value."*

Generally, the "reference quantity" is considered to be the "true value" of the measurand and is therefore unknown. Figure 1 illustrates these concepts.

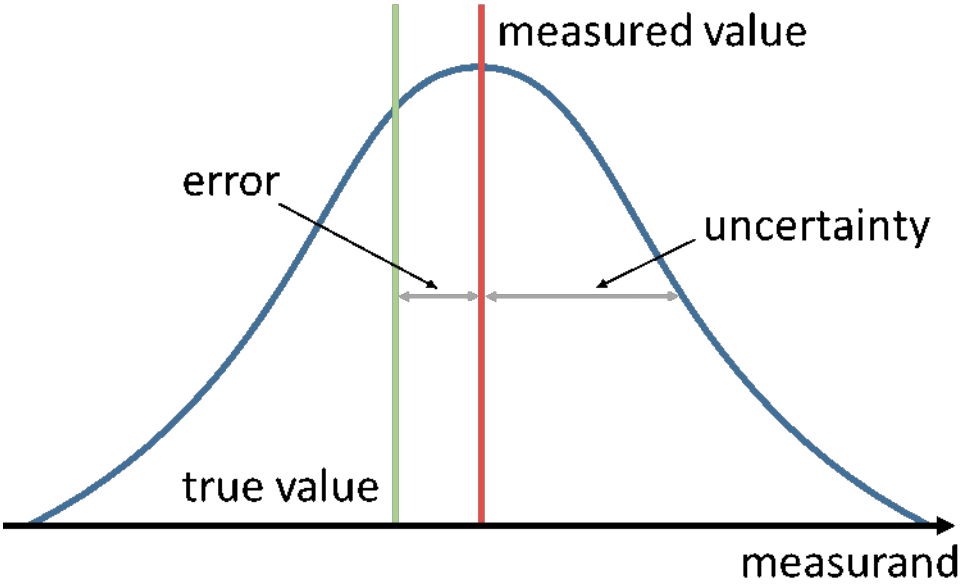



*Figure 1 - Diagram illustrating the different concepts of measured value and true value, uncertainty and error.*

In a series of measurements (for example each pixel in a remote sensing Level 1 (L1) data product) it is vital to consider how the errors between the measurements in the series are correlated. This is crucial when evaluating the uncertainty of a result derived from these data (for example a Level 2 (L2) retrieval of geophysical parameter from a L1 product). In their vocabulary the Horizon 2020 FIDUCEO[1] (Fidelity and uncertainty in climate data records from Earth observations) project (see FIDUCEO Vocabulary, 2018) define three broad categories of error correlation effects important to satellite data products, as follows:

- **Random effects**: *"those causing errors that cannot be corrected for in a single measured value, even in principle, because the effect is stochastic. Random effects for a particular measurement process vary unpredictably from (one set of) measurement(s) to (another set of) measurement(s). These produce random errors which are entirely uncorrelated between measurements (or sets of measurements) and generally are reduced by averaging."*
- **Structured random effects**: *"means those that across many observations there is a deterministic pattern of errors whose amplitude is stochastically drawn from an underlying probability distribution; "structured random" therefore implies "unpredictable" and "correlated across measurements"…"*
- **Systematic (or common) effects**: *"those for a particular measurement process that do not vary (or vary coherently) from (one set of) measurement(s) to (another set of) measurement(s) and therefore produce systematic errors that cannot be reduced by averaging."*

[1] See: https://www.fiduceo.eu

### 2.3.2 Law of Propagation of Uncertainty Method

Within the GUM framework uncertainty analysis begins with understanding the measurement function. The measurement function establishes the mathematical relationship between all known input quantities (e.g. instrument counts) and the measurand itself (e.g. radiance). Generally, this may be written as

$$Y = f\left(X_i, \ldots, X_N\right)$$

where:

- $Y$ is the measurand;
- $X_i$ are the input quantities.

Uncertainty analysis is then performed by considering in turn each of these different input quantities to the measurement function, this process is represented in Figure 2. Each input quantity may be influenced by one or more error effects which are described by an uncertainty distribution. These separate distributions may then be combined to determine the uncertainty of the measurand, $u^2(Y)$, using the *Law of Propagation of Uncertainties* (GUM, 2008),

$$u^2\left(Y\right) = \mathrm{C}\ \mathrm{S}\left(\mathbf{X}\right)\mathbf{C}^T$$

where:

- $\mathbf{C}$ is the vector of sensitivity coefficients, $\partial Y/\partial X_i$;
- $\mathbf{S}(\mathbf{X})$ is the error covariance matrix for the input quantities.

This can be extended to a measurement function with a measurand vector (rather than scalar) $\mathbf{Y} = (Y_i, \ldots, Y_N)$. The uncertainties are then given by:

$$\mathbf{S}(\mathbf{Y}) = \mathbf{J}\ \mathbf{S}(\mathbf{X})\mathbf{J}^T$$

where:

- $\mathbf{J}$ is the Jacobian matrix of sensitivity coefficients, $J_{ni} = \partial Y_n/\partial X_i$;
- $\mathbf{S}(\mathbf{Y})$ is the error covariance matrix (n*n) for the measurand;
- $\mathbf{S}(\mathbf{X})$ is the error covariance matrix (i*i) for the input quantities.

The error covariances matrices define the uncertainties (from the diagonal elements) as well as the correlation between the different quantities (off-diagonal elements).

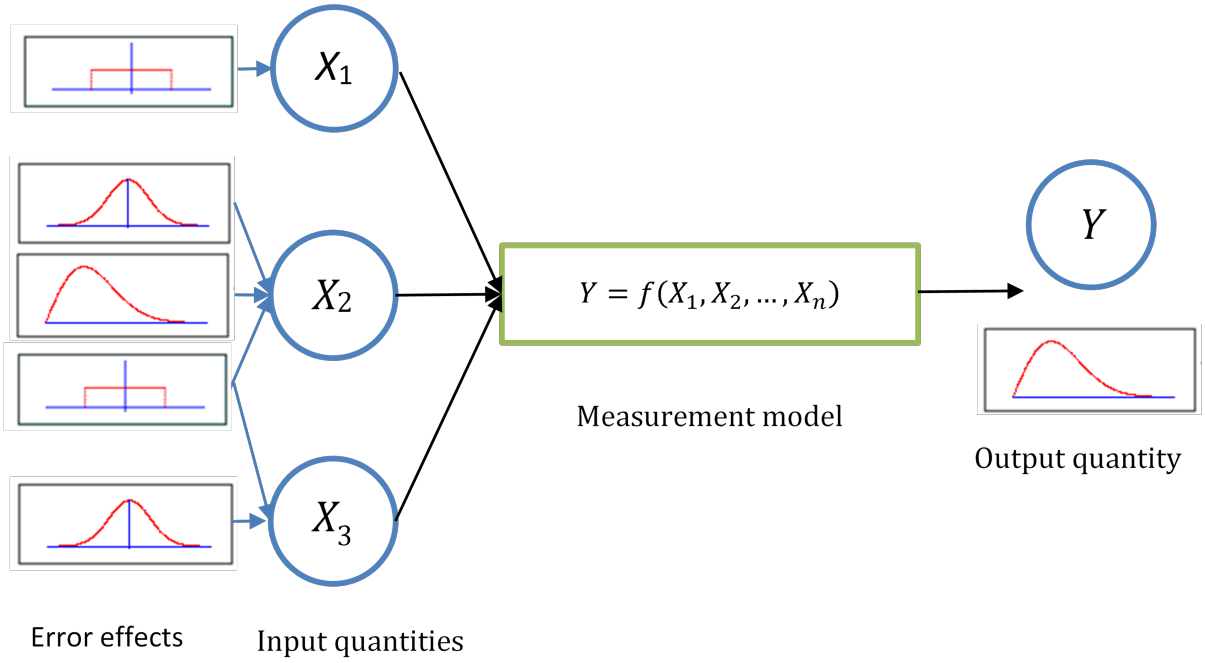



*Figure 2 - Conceptual process of uncertainty propagation.*

### 2.3.3 Monte Carlo Method

For a detailed description of the Monte Carlo (MC) method, we refer to Supplement 1 to the "Guide to the expression of uncertainty in measurement" - Propagation of distributions using a Monte Carlo method.

Here we summarise the main steps and detail how these were implemented. The main stages consist of:

- Formulation: Defining the measurand (output quantity Y), the input quantities $X = (X_i, \ldots, X_N)$, and the measurement function (as a model relating Y and X). One also needs to asign Probability Density Functions (PDF) of each of the input quantities, as well as define the correlation between them (through joint PDF).
- Propagation: propagate the PDFs for the $X_i$ through the model to obtain the PDF for Y.
- Summarizing: Use the PDF for Y to obtain the expectation of Y, the standard uncertainty u(Y) associated with Y (from the standard deviation), and the covariance between the different values in Y.

The MC method implemented in punpy consists of generating joint PDF from the provided uncertainties and correlation matrices or covariances. Punpy then propagates the PDFs for the $X_i$ to Y and then summarises the results through returning the uncertainties and correlation matrices.

As punpy is meant to be widely applicable, the user can define the measurand, input quantities and measurement function themselves. Within punpy, the input quantities and measurand will often be provided as python arrays (or scalars) and the measurement function in particular needs to be a python function that can take the input quantities as function arguments and returns the measurand.

To generate the joint PDF, comet_maths is used. The ATBD for the comet_maths PDF generator is given here.

The measurand pdf is then defined by processing each draw $\xi_i$ to Y:

$Y = f(\xi)$.

The measurands for each of these draws are then concatenated into one array containing the full measurand sample. The uncertainties are than calculated by taking the standard deviation along these draws.

When no `corr_dims` are specified, but `return_corr` is set to True, the correlation matrix is calculated by calculating the Pearson product-moment correlation coefficients of the full measurand sample along the draw dimension. When `corr_dims` are specified, the correlation matrix is calculated along a subset of the full measurand sample. This subset is made by taking only the first index along every dimension that is not the correlation dimension. We note that `corr_dims` should only be used when the error correlation matrices do not vary along the other dimension(s). A warning is raised if any element of the correlation matrix varies by more than 0.05 between the one using a subset taking only the first index along each other dimension and the one using a subset taking only the last index along each other dimension.

## 2.4 API reference

This page provides an auto-generated summary of **punpy**'s API. For more details and examples, refer to the relevant chapters in the main part of the documentation.

### 2.4.1 MCPropagation

| | |
|---|---|
| *mc.mc_propagation.MCPropagation*(steps[, ...]) | Class to propagate uncertainties using Monte Carlo (MC) |
| *mc.mc_propagation.MCPropagation.propagate_standard*(...) | Propagate uncertainties through measurement function with n input quantities. |
| *mc.mc_propagation.MCPropagation.propagate_random*(...) | Propagate random uncertainties through measurement function with n input quantities. |
| *mc.mc_propagation.MCPropagation.propagate_systematic*(...) | Propagate systematic uncertainties through measurement function with n input quantities. |
| *mc.mc_propagation.MCPropagation.propagate_cov*(...) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *mc.mc_propagation.MCPropagation.generate_MC_sample*(x, ...) | function to generate MC sample for input quantities |
| *mc.mc_propagation.MCPropagation.generate_MC_sample_cov*(x, ...) | function to generate MC sample for input quantities from covariance matrix |
| *mc.mc_propagation.MCPropagation.propagate_cov_flattened*(...) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *mc.mc_propagation.MCPropagation.run_samples*(...) | process all the MC samples of input quantities through the measurand function |
| *mc.mc_propagation.MCPropagation.combine_samples*(...) | Function to combine MC samples from individual runs |
| *mc.mc_propagation.MCPropagation.process_samples*(...) | Run the MC-generated samples of input quantities through the measurement function and calculate correlation matrix if required. |

### punpy.mc.mc_propagation.MCPropagation

**class** punpy.mc.mc_propagation.**MCPropagation**(*steps*, *parallel_cores=0*, *dtype=None*, *verbose=False*, *MCdimlast=True*)

Class to propagate uncertainties using Monte Carlo (MC)

**Parameters**

- **steps** (*int*) – number of MC iterations
- **parallel_cores** (*int*) – number of CPU to be used in parallel processing
- **dtype** (*numpy dtype*) – numpy dtype for output variables
- **verbose** (*bool*) – bool to set if logging info should be printed
- **MCdimlast** (*bool*) – bool to set whether the MC dimension should be moved to the last dimension when running through the measurment function (when parallel_cores==0). This can be useful for broadcasting within the measurement function. defaults to False

**__init__**(*steps*, *parallel_cores=0*, *dtype=None*, *verbose=False*, *MCdimlast=True*)

#### Methods

| | |
|---|---|
| *__init__*(steps[, parallel_cores, dtype, ...]) | |
| *combine_samples*(MC_samples) | Function to combine MC samples from individual runs |
| *generate_MC_sample*(x, u_x, corr_x[, ...]) | function to generate MC sample for input quantities |
| *generate_MC_sample_cov*(x, cov_x[, ...]) | function to generate MC sample for input quantities from covariance matrix |
| *process_samples*(MC_x, MC_y[, return_corr, ...]) | Run the MC-generated samples of input quantities through the measurement function and calculate correlation matrix if required. |
| *propagate_cov*(func, x, cov_x[, param_fixed, ...]) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *propagate_cov_flattened*(func, x, cov_x[, ...]) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *propagate_random*(func, x, u_x[, corr_x, ...]) | Propagate random uncertainties through measurement function with n input quantities. |
| *propagate_standard*(func, x, u_x, corr_x[, ...]) | Propagate uncertainties through measurement function with n input quantities. |
| *propagate_systematic*(func, x, u_x[, corr_x, ...]) | Propagate systematic uncertainties through measurement function with n input quantities. |
| *run_samples*(func, MC_x[, output_vars, ...]) | process all the MC samples of input quantities through the measurand function |

### punpy.mc.mc_propagation.MCPropagation.propagate_standard

MCPropagation.**propagate_standard**(*func*, *x*, *u_x*, *corr_x*, *param_fixed=None*, *corr_between=None*, *samples=None*, *return_corr=False*, *return_samples=False*, *repeat_dims=-99*, *corr_dims=-99*, *separate_corr_dims=False*, *fixed_corr_var=False*, *output_vars=1*, *PD_corr=True*, *refyvar=0*, *pdf_shape='gaussian'*, *pdf_params=None*, *allow_some_nans=True*)

Propagate uncertainties through measurement function with n input quantities. Correlations must be specified in corr_x. Input quantities can be floats, vectors (1d-array) or images (2d-array). Systematic uncertainties arise when there is full correlation between repeated measurements. There is a often also a correlation between measurements along the dimensions that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of systematic uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array]*) – list of correlation matrices (n,n) along non-repeating axis. Can be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None
- **samples** (*list[array], optional*) – allows to provide a Monte Carlo sample previously generated. This sample of input quantities will be used instead of generating one from the uncertainties and error-correlation. Defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to False
- **return_samples** (*bool, optional*) – set to True to return generated samples, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated. When the combined correlation of 2 or more (but not all) dimensions is required, they can be provided as a string containing the different dimension integers, separated by a dot (e.g. "0.2"). When multiple error_correlations should be calculated, they can be provided as a list.
- **separate_corr_dims** (*bool, optional*) – When set to True and output_vars>1, corr_dims should be a list providing the corr_dims for each output variable, each following the format defined in the corr_dims description. Defaults to False

- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **PD_corr** (*bool, optional*) – set to True to make sure returned correlation matrices are positive semi-definite, default to True
- **refyvar** (*int, optional*) – Index of output variable with reference shape (only relevant when output_vars>1; should be output variable with most dimensions; affects things like repeat_dims)
- **pdf_shape** (*str, optional*) – string identifier of the probability density function shape, defaults to gaussian
- **pdf_params** (*dict, optional*) – dictionaries defining optional additional parameters that define the probability density function, Defaults to None (gaussian does not require additional parameters)
- **allow_some_nans** (*bool, optional*) – set to False to ignore any MC sample which has any nan's in the measurand. Defaults to True, in which case only MC samples with only nan's are ignored.

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.mc.mc_propagation.MCPropagation.propagate_random

MCPropagation.**propagate_random**(*func*, *x*, *u_x*, *corr_x=None*, *param_fixed=None*, *corr_between=None*, *samples=None*, *return_corr=False*, *return_samples=False*, *repeat_dims=-99*, *corr_dims=-99*, *separate_corr_dims=False*, *fixed_corr_var=False*, *output_vars=1*, *PD_corr=True*, *refyvar=0*, *pdf_shape='gaussian'*, *pdf_params=None*, *allow_some_nans=True*)

Propagate random uncertainties through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). Random uncertainties arise when there is no correlation between repeated measurements. It is possible (though rare) that there is a correlation in one of the dimensions that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of random uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array], optional*) – list of correlation matrices (n,n) along non-repeating axis, defaults to None. Can optionally be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).

- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None
- **samples** (*list[array], optional*) – allows to provide a Monte Carlo sample previously generated. This sample of input quantities will be used instead of generating one from the uncertainties and error-correlation. Defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to False
- **return_samples** (*bool, optional*) – set to True to return generated samples, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated. When the combined correlation of 2 or more (but not all) dimensions is required, they can be provided as a string containing the different dimension integers, separated by a dot (e.g. "0.2"). When multiple error_correlations should be calculated, they can be provided as a list.
- **separate_corr_dims** (*bool, optional*) – When set to True and output_vars>1, corr_dims should be a list providing the corr_dims for each output variable, each following the format defined in the corr_dims description. Defaults to False
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimiension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **PD_corr** (*bool, optional*) – set to True to make sure returned correlation matrices are positive semi-definite, default to True.
- **refyvar** (*int, optional*) – Index of output variable with reference shape (only relevant when output_vars>1; should be output variable with most dimensions; affects things like repeat_dims).
- **pdf_shape** (*str, optional*) – string identifier of the probability density function shape, defaults to gaussian.
- **pdf_params** (*dict, optional*) – dictionaries defining optional additional parameters that define the probability density function, Defaults to None (gaussian does not require additional parameters).
- **allow_some_nans** (*bool, optional*) – set to False to ignore any MC sample which has any nan's in the measurand. Defaults to True, in which case only MC samples with only nan's are ignored.

**Returns**

uncertainties on measurand

**Return type**
array

### punpy.mc.mc_propagation.MCPropagation.propagate_systematic

MCPropagation.**propagate_systematic**(*func*, *x*, *u_x*, *corr_x=None*, *param_fixed=None*, *corr_between=None*, *samples=None*, *return_corr=False*, *return_samples=False*, *repeat_dims=-99*, *corr_dims=-99*, *separate_corr_dims=False*, *fixed_corr_var=False*, *output_vars=1*, *PD_corr=True*, *refyvar=0*, *pdf_shape='gaussian'*, *pdf_params=None*, *allow_some_nans=True*)

Propagate systematic uncertainties through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). Systematic uncertainties arise when there is full correlation between repeated measurements. There is a often also a correlation between measurements along the dimensions that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of systematic uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array], optional*) – list of correlation matrices (n,n) along non-repeating axis, defaults to None. Can optionally be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None
- **samples** (*list[array], optional*) – allows to provide a Monte Carlo sample previously generated. This sample of input quantities will be used instead of generating one from the uncertainties and error-correlation. Defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to False
- **return_samples** (*bool, optional*) – set to True to return generated samples, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated. When the combined correlation of 2 or more (but not all) dimensions is required, they can be provided as a string containing the different dimension integers, separated by a dot (e.g. "0.2"). When multiple error_correlations should be calculated, they can be provided as a list.

- **separate_corr_dims** (*bool, optional*) – When set to True and output_vars>1, corr_dims should be a list providing the corr_dims for each output variable, each following the format defined in the corr_dims description. Defaults to False
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimiension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **PD_corr** (*bool, optional*) – set to True to make sure returned correlation matrices are positive semi-definite, default to True
- **refyvar** (*int, optional*) – Index of output variable with reference shape (only relevant when output_vars>1; should be output variable with most dimensions; affects things like repeat_dims)
- **pdf_shape** (*str, optional*) – string identifier of the probability density function shape, defaults to gaussian
- **pdf_params** (*dict, optional*) – dictionaries defining optional additional parameters that define the probability density function, Defaults to None (gaussian does not require additional parameters)
- **allow_some_nans** (*bool, optional*) – set to False to ignore any MC sample which has any nan's in the measurand. Defaults to True, in which case only MC samples with only nan's are ignored.

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.mc.mc_propagation.MCPropagation.propagate_cov

MCPropagation.**propagate_cov**(*func*, *x*, *cov_x*, *param_fixed=None*, *corr_between=None*, *samples=None*, *return_corr=True*, *return_samples=False*, *repeat_dims=-99*, *corr_dims=-99*, *separate_corr_dims=False*, *fixed_corr_var=False*, *output_vars=1*, *PD_corr=True*, *refyvar=0*, *pdf_shape='gaussian'*, *pdf_params=None*, *allow_some_nans=True*)

Propagate uncertainties with given covariance matrix through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). The covariance matrix can represent the full covariance matrix between all measurements in all dimensions. Alternatively if there are repeated measurements specified in repeat_dims, the covariance matrix is given for the covariance along the dimension that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **cov_x** (*list[array]*) – list of covariance matrices on input quantities (usually numpy arrays). In case the input quantity is an array of shape (m,o), the covariance matrix is typically given as an array of shape (m*o,m*o).

- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – covariance matrix (n,n) between input quantities, defaults to None
- **samples** (*list[array], optional*) – allows to provide a Monte Carlo sample previously generated. This sample of input quantities will be used instead of generating one from the uncertainties and error-correlation. Defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to True
- **return_samples** (*bool, optional*) – set to True to return generated samples, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated. When the combined correlation of 2 or more (but not all) dimensions is required, they can be provided as a string containing the different dimension integers, separated by a dot (e.g. "0.2"). When multiple error_correlations should be calculated, they can be provided as a list.
- **separate_corr_dims** (*bool, optional*) – When set to True and output_vars>1, corr_dims should be a list providing the corr_dims for each output variable, each following the format defined in the corr_dims description. Defaults to False
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimiension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **PD_corr** (*bool, optional*) – set to True to make sure returned correlation matrices are positive semi-definite, default to True
- **refyvar** (*int, optional*) – Index of output variable with reference shape (only relevant when output_vars>1; should be output variable with most dimensions; affects things like repeat_dims)
- **pdf_shape** (*str, optional*) – string identifier of the probability density function shape, defaults to gaussian
- **pdf_params** (*dict, optional*) – dictionaries defining optional additional parameters that define the probability density function, Defaults to None (gaussian does not require additional parameters)
- **allow_some_nans** (*bool, optional*) – set to False to ignore any MC sample which has any nan's in the measurand. Defaults to True, in which case only MC samples with only nan's are ignored.

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.mc.mc_propagation.MCPropagation.generate_MC_sample

MCPropagation.**generate_MC_sample**(*x*, *u_x*, *corr_x*, *corr_between=None*, *pdf_shape='gaussian'*, *pdf_params=None*, *comp_list=False*)

function to generate MC sample for input quantities

**Parameters**

- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of systematic uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array]*) – list of correlation matrices (n,n) along non-repeating axis. Can be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.
- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None
- **pdf_shape** (*str, optional*) – string identifier of the probability density function shape, defaults to gaussian
- **pdf_params** (*dict, optional*) – dictionaries defining optional additional parameters that define the probability density function, Defaults to None (gaussian does not require additional parameters)
- **comp_list** (*bool, optional*) – boolean to define whether u_x and corr_x are given as a list or individual uncertainty components. Defaults to False, in which case a single combined uncertainty component is given per input quantity.

**Returns**
MC sample for input quantities

**Return type**
list[array]

### punpy.mc.mc_propagation.MCPropagation.generate_MC_sample_cov

MCPropagation.**generate_MC_sample_cov**(*x*, *cov_x*, *corr_between=None*, *pdf_shape='gaussian'*, *pdf_params=None*)

function to generate MC sample for input quantities from covariance matrix

**Parameters**

- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of systematic uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array]*) – list of correlation matrices (n,n) along non-repeating axis. Can be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.

- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None
- **pdf_shape** (*str, optional*) – string identifier of the probability density function shape, defaults to gaussian
- **pdf_params** (*dict, optional*) – dictionaries defining optional additional parameters that define the probability density function, Defaults to None (gaussian does not require additional parameters)

**Returns**
MC sample for input quantities

**Return type**

### punpy.mc.mc_propagation.MCPropagation.propagate_cov_flattened

MCPropagation.**propagate_cov_flattened**(*func*, *x*, *cov_x*, *param_fixed=None*, *corr_between=None*, *samples=None*, *return_corr=True*, *return_samples=False*, *repeat_dims=-99*, *corr_dims=-99*, *separate_corr_dims=False*, *fixed_corr_var=False*, *output_vars=1*, *PD_corr=True*, *refyvar=0*, *pdf_shape='gaussian'*, *pdf_params=None*)

Propagate uncertainties with given covariance matrix through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). The covariance matrix can represent the full covariance matrix between all measurements in all dimensions. Alternatively if there are repeated measurements specified in repeat_dims, the covariance matrix is given for the covariance along the dimension that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **cov_x** (*list[array]*) – list of covariance matrices on input quantities (usually numpy arrays). In case the input quantity is an array of shape (m,o), the covariance matrix needs to be given as an array of shape (m*o,m*o).
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – covariance matrix (n,n) between input quantities, defaults to None
- **samples** (*list[array], optional*) – allows to provide a Monte Carlo sample previously generated. This sample of input quantities will be used instead of generating one from the uncertainties and error-correlation. Defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to True
- **return_samples** (*bool, optional*) – set to True to return generated samples, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save

memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..

- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated. When the combined correlation of 2 or more (but not all) dimensions is required, they can be provided as a string containing the different dimension integers, separated by a dot (e.g. "0.2"). When multiple error_correlations should be calculated, they can be provided as a list.
- **separate_corr_dims** (*bool, optional*) – When set to True and output_vars>1, corr_dims should be a list providing the corr_dims for each output variable, each following the format defined in the corr_dims description. Defaults to False
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimiension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **PD_corr** (*bool, optional*) – set to True to make sure returned correlation matrices are positive semi-definite, default to True
- **refyvar** (*int, optional*) – Index of output variable with reference shape (only relevant when output_vars>1; should be output variable with most dimensions; affects things like repeat_dims)
- **pdf_shape** (*str, optional*) – string identifier of the probability density function shape, defaults to gaussian
- **pdf_params** (*dict, optional*) – dictionaries defining optional additional parameters that define the probability density function, Defaults to None (gaussian does not require additional parameters)

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.mc.mc_propagation.MCPropagation.run_samples

MCPropagation.**run_samples**(*func*, *MC_x*, *output_vars=1*, *start=None*, *end=None*, *sli=None*, *allow_some_nans=True*)

process all the MC samples of input quantities through the measurand function

**Parameters**

- **func** (*function*) – measurement function
- **MC_x** (*array[array]*) – MC-generated samples of input quantities
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.

- **start** (*integer, optional*) – set this parameter to propagate the input quantities through the measurement function starting from a specfic index. All input quantities before this index are ignored. Defaults to None, in which case no input quantities are ignored.
- **end** (*integer, optional*) – set this parameter to propagate the input quantities through the measurement function up until a specfic index. All input quantities after this index are ignored. Defaults to None, in which case no input quantities are ignored.
- **sli** (*slice, optional*) – set this parameter to a slice to set which input quantities will be processed through the measurment function. All other input quantities are ignored. Can only be used if start and end are not set. Defaults to None, in which case no input quantities are ignored.
- **allow_some_nans** (*bool, optional*) – set to False to ignore any MC sample which has any nan's in the measurand. Defaults to True, in which case only MC samples with only nan's are ignored.

**Returns**
MC sample of measurand

**Return type**
array[array]

### punpy.mc.mc_propagation.MCPropagation.combine_samples

MCPropagation.**combine_samples**(*MC_samples*)

Function to combine MC samples from individual runs

**Parameters**
**MC_samples** (*List(np.array)*) – list of samples to be concatenated

**Returns**
concatenated MC sample

### punpy.mc.mc_propagation.MCPropagation.process_samples

MCPropagation.**process_samples**(*MC_x*, *MC_y*, *return_corr=False*, *return_samples=False*, *yshapes=None*, *corr_dims=-99*, *separate_corr_dims=False*, *fixed_corr=None*, *PD_corr=True*, *output_vars=1*)

Run the MC-generated samples of input quantities through the measurement function and calculate correlation matrix if required.

**Parameters**

- **MC_x** (*array[array]*) – MC-generated samples of input quantities
- **MC_y** (*array[array]*) – MC sample of measurand
- **return_corr** (*bool*) – set to True to return correlation matrix of measurand
- **return_samples** (*bool*) – set to True to return generated samples
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated. When the combined correlation of 2 or more (but not all) dimensions is required, they can be provided as a string containing the different dimension integers, separated by a dot (e.g. "0.2"). When multiple error_correlations should be calculated, they can be provided as a list.

- **separate_corr_dims** (*bool, optional*) – When set to True and output_vars>1, corr_dims should be a list providing the corr_dims for each output variable, each following the format defined in the corr_dims description. Defaults to False
- **fixed_corr** (*array*) – correlation matrix to be copied without changing, defaults to None (correlation matrix is calculated rather than copied)
- **PD_corr** (*bool, optional*) – set to True to make sure returned correlation matrices are positive semi-definite, default to True
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.

**Returns**
uncertainties on measurand

**Return type**
array

### 2.4.2 LPUPropagation

| | |
|---|---|
| *lpu.lpu_propagation.LPUPropagation*([...]) | Class to propagate uncertainties using the Law of Propagation of Uncertainty |
| *lpu.lpu_propagation.LPUPropagation.propagate_standard*(...) | Propagate uncertainties through measurement function with n input quantities. |
| *lpu.lpu_propagation.LPUPropagation.propagate_random*(...) | Propagate random uncertainties through measurement function with n input quantities. |
| *lpu.lpu_propagation.LPUPropagation.propagate_systematic*(...) | Propagate systematic uncertainties through measurement function with n input quantities. |
| *lpu.lpu_propagation.LPUPropagation.propagate_cov*(...) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *lpu.lpu_propagation.LPUPropagation.propagate_flattened_cov*(...) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *lpu.lpu_propagation.LPUPropagation.process_jacobian*(J, ...) | |

#### punpy.lpu.lpu_propagation.LPUPropagation

**class** punpy.lpu.lpu_propagation.**LPUPropagation**(*parallel_cores=0*, *Jx_diag=False*, *step=None*, *verbose=False*)

Class to propagate uncertainties using the Law of Propagation of Uncertainty

**Parameters**

- **parallel_cores** (*int*) – number of CPU to be used in parallel processing
- **Jx_diag** – Bool to indicate whether the Jacobian matrix can be described with semi-diagonal elements. With this we mean that the measurand has the same shape as each of the input quantities and the square jacobain between the measurand and each of the input quantities individually, only has diagonal elements. Defaults to False
- **step** (*float*) – Defines the spacing used when calculating the Jacobian with numdifftools
- **verbose** (*bool*) – bool to set if logging info should be printed

**Rtype Jx_diag**
bool, optional

**__init__**(*parallel_cores=0*, *Jx_diag=False*, *step=None*, *verbose=False*)

### Methods

| | |
|---|---|
| *__init__*([parallel_cores, Jx_diag, step, ...]) | |
| *process_jacobian*(J, covx, shape_y, ...[, ...]) | |
| *propagate_cov*(func, x, cov_x[, param_fixed, ...]) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *propagate_flattened_cov*(func, x, flat_cov_x) | Propagate uncertainties with given covariance matrix through measurement function with n input quantities. |
| *propagate_random*(func, x, u_x[, corr_x, ...]) | Propagate random uncertainties through measurement function with n input quantities. |
| *propagate_standard*(func, x, u_x, corr_x[, ...]) | Propagate uncertainties through measurement function with n input quantities. |
| *propagate_systematic*(func, x, u_x[, corr_x, ...]) | Propagate systematic uncertainties through measurement function with n input quantities. |

## punpy.lpu.lpu_propagation.LPUPropagation.propagate_standard

LPUPropagation.**propagate_standard**(*func*, *x*, *u_x*, *corr_x*, *param_fixed=None*, *corr_between=None*, *return_corr=False*, *return_Jacobian=False*, *repeat_dims=-99*, *corr_dims=-99*, *fixed_corr_var=False*, *output_vars=1*, *Jx=None*, *Jx_diag=None*)

Propagate uncertainties through measurement function with n input quantities. Correlations must be specified in corr_x. Input quantities can be floats, vectors (1d-array) or images (2d-array). Systematic uncertainties arise when there is full correlation between repeated measurements. There is a often also a correlation between measurements along the dimensions that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of systematic uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array]*) – list of correlation matrices (n,n) along non-repeating axis. Can optionally be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None

- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to False
- **return_Jacobian** (*bool, optional*) – set to True to return Jacobian matrix, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated.
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **Jx** – Jacobian matrix, evaluated at x. This allows to give a precomputed jacobian matrix, which could potentially be calculated using analytical prescription. Defaults to None, in which case Jx is calculated numerically as part of the propagation.
- **Jx_diag** – Bool to indicate whether the Jacobian matrix can be described with semi-diagonal elements. With this we mean that the measurand has the same shape as each of the input quantities and the square jacobain between the measurand and each of the input quantities individually, only has diagonal elements. Defaults to False

**Rtype Jx**
array, optional

**Rtype Jx_diag**
bool, optional

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.lpu.lpu_propagation.LPUPropagation.propagate_random

LPUPropagation.**propagate_random**(*func*, *x*, *u_x*, *corr_x=None*, *param_fixed=None*, *corr_between=None*, *return_corr=False*, *return_Jacobian=False*, *repeat_dims=-99*, *corr_dims=-99*, *fixed_corr_var=False*, *output_vars=1*, *Jx=None*, *Jx_diag=None*)

Propagate random uncertainties through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). Random uncertainties arise when there is no correlation between repeated measurements. It is possible (though rare) that there is a correlation in one of the dimensions that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of random uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array], optional*) – list of correlation matrices (n,n) along non-repeating axis, defaults to None. Can optionally be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to False
- **return_Jacobian** (*bool, optional*) – set to True to return Jacobian matrix, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated.
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **Jx** – Jacobian matrix, evaluated at x. This allows to give a precomputed jacobian matrix, which could potentially be calculated using analytical prescription. Defaults to None, in which case Jx is calculated numerically as part of the propagation.
- **Jx_diag** – Bool to indicate whether the Jacobian matrix can be described with semi-diagonal elements. With this we mean that the measurand has the same shape as each of the input quantities and the square jacobain between the measurand and each of the input quantities individually, only has diagonal elements. Defaults to None, in which case the object value is used.

**Rtype Jx**
array, optional

**Rtype Jx_diag**
bool, optional

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.lpu.lpu_propagation.LPUPropagation.propagate_systematic

LPUPropagation.**propagate_systematic**(*func*, *x*, *u_x*, *corr_x=None*, *param_fixed=None*, *corr_between=None*, *return_corr=False*, *return_Jacobian=False*, *repeat_dims=-99*, *corr_dims=-99*, *fixed_corr_var=False*, *output_vars=1*, *Jx=None*, *Jx_diag=None*)

Propagate systematic uncertainties through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). Systematic uncertainties arise when there is full correlation between repeated measurements. There is a often also a correlation between measurements along the dimensions that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **u_x** (*list[array]*) – list of systematic uncertainties on input quantities (usually numpy arrays)
- **corr_x** (*list[array], optional*) – list of correlation matrices (n,n) along non-repeating axis, defaults to None. Can optionally be set to "rand" (diagonal correlation matrix), "syst" (correlation matrix of ones) or a custom correlation matrix.
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – correlation matrix (n,n) between input quantities, defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to False
- **return_Jacobian** (*bool, optional*) – set to True to return Jacobian matrix, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated.
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.

- **Jx** – Jacobian matrix, evaluated at x. This allows to give a precomputed jacobian matrix, which could potentially be calculated using analytical prescription. Defaults to None, in which case Jx is calculated numerically as part of the propagation.
- **Jx_diag** – Bool to indicate whether the Jacobian matrix can be described with semi-diagonal elements. With this we mean that the measurand has the same shape as each of the input quantities and the square jacobain between the measurand and each of the input quantities individually, only has diagonal elements. Defaults to False

**Rtype Jx**
array, optional

**Rtype Jx_diag**
bool, optional

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.lpu.lpu_propagation.LPUPropagation.propagate_cov

LPUPropagation.**propagate_cov**(*func*, *x*, *cov_x*, *param_fixed=None*, *corr_between=None*, *return_corr=False*, *return_Jacobian=False*, *repeat_dims=-99*, *corr_dims=-99*, *fixed_corr_var=False*, *output_vars=1*, *Jx=None*, *Jx_diag=None*)

Propagate uncertainties with given covariance matrix through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). The covariance matrix can represent the full covariance matrix between all measurements in all dimensions. Alternatively if there are repeated measurements specified in repeat_dims, the covariance matrix is given for the covariance along the dimension that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **cov_x** (*list[array]*) – list of covariance matrices on input quantities (usually numpy arrays). In case the input quantity is an array of shape (m,o), the covariance matrix needs to be given as an array of shape (m*o,m*o).
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split (param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).
- **corr_between** (*array, optional*) – covariance matrix (n,n) between input quantities, defaults to None
- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to True
- **return_Jacobian** (*bool, optional*) – set to True to return Jacobian matrix, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save

memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..

- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated.
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **Jx** – Jacobian matrix, evaluated at x. This allows to give a precomputed jacobian matrix, which could potentially be calculated using analytical prescription. Defaults to None, in which case Jx is calculated numerically as part of the propagation.
- **Jx_diag** – Bool to indicate whether the Jacobian matrix can be described with semi-diagonal elements. With this we mean that the measurand has the same shape as each of the input quantities and the square jacobain between the measurand and each of the input quantities individually, only has diagonal elements. Defaults to False

**Rtype Jx**
array, optional

**Rtype Jx_diag**
bool, optional

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.lpu.lpu_propagation.LPUPropagation.propagate_flattened_cov

LPUPropagation.**propagate_flattened_cov**(*func*, *x*, *flat_cov_x*, *return_corr=False*, *return_Jacobian=False*, *corr_dims=-99*, *output_vars=1*, *Jx=None*, *Jx_diag=None*)

Propagate uncertainties with given covariance matrix through measurement function with n input quantities. Input quantities can be floats, vectors (1d-array) or images (2d-array). The covariance matrix can represent the full covariance matrix between all measurements in all dimensions. Alternatively if there are repeated measurements specified in repeat_dims, the covariance matrix is given for the covariance along the dimension that is not one of the repeat_dims.

**Parameters**

- **func** (*function*) – measurement function
- **x** (*list[array]*) – list of input quantities (usually numpy arrays)
- **flat_cov_x** (*list[array]*) – flattened covariance matrix on flattened (concatenated) input quantities (usually numpy arrays). In case there is n input quantity arrays of shape (m,o), the covariance matrix needs to be given as an array of shape (n*m*o,n*m*o).
- **param_fixed** (*list of bools, optional*) – when repeat_dims>=0, set to true or false to indicate for each input quantity whether it has repeated measurements that should be split

(param_fixed=False) or whether the input is fixed (param fixed=True), defaults to None (no inputs fixed).

- **return_corr** (*bool, optional*) – set to True to return correlation matrix of measurand, defaults to True
- **return_Jacobian** (*bool, optional*) – set to True to return Jacobian matrix, defaults to False
- **repeat_dims** (*integer or list of 2 integers, optional*) – set to positive integer(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated.
- **fixed_corr_var** (*bool or integer, optional*) – set to integer to copy the correlation matrix of the dimension the integer refers to. Set to True to automatically detect if only one uncertainty is present and the correlation matrix of that dimension should be copied. Defaults to False.
- **output_vars** (*integer, optional*) – number of output parameters in the measurement function. Defaults to 1.
- **Jx** – Jacobian matrix, evaluated at x. This allows to give a precomputed jacobian matrix, which could potentially be calculated using analytical prescription. Defaults to None, in which case Jx is calculated numerically as part of the propagation.
- **Jx_diag** – Bool to indicate whether the Jacobian matrix can be described with semi-diagonal elements. With this we mean that the measurand has the same shape as each of the input quantities and the square jacobain between the measurand and each of the input quantities individually, only has diagonal elements. Defaults to False

**Rtype Jx**
array, optional

**Rtype Jx_diag**
bool, optional

**Returns**
uncertainties on measurand

**Return type**
array

### punpy.lpu.lpu_propagation.LPUPropagation.process_jacobian

LPUPropagation.**process_jacobian**(*J*, *covx*, *shape_y*, *return_corr*, *corr_dims*, *fixed_corr=None*, *output_vars=1*, *return_Jacobian=False*)

### 2.4.3 Digital Effects Tables

| | |
|---|---|
| *digital_effects_table.measurement_function.MeasurementFunction*([...]) | MeasurementFunction class which provides all the functionality for propagating uncertainties using obsarray digital effects tables. |
| *digital_effects_table.measurement_function.MeasurementFunction.meas_function*(...) | meas_function is the measurement function itself, to be used in the uncertainty propagation. |
| *digital_effects_table.measurement_function.MeasurementFunction.get_argument_names*() | This function allows to return the names of the input quantities as a list of strings. |
| *digital_effects_table.measurement_function.MeasurementFunction.get_measurand_name_and_unit*() | This function allows to return the name and unit of the measurand as strings. |
| *digital_effects_table.measurement_function.MeasurementFunction.update_measurand*(...) | |
| *digital_effects_table.measurement_function.MeasurementFunction.setup*(...) | This function is to provide a setup stage that can be run before propagating uncertainties. |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_ds*(*args) | Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the combined random, systematic and structured uncertainties on the measurand |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_ds_total*(*args) | Function to propagate the total uncertainties present in the digital effects tables in the input arguments, through the measurement function to produce an output digital effects table with the total uncertainties on the measurand |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_ds_specific*(...) | Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the uncertainties of specific components listed in comp_list. |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_ds_all*(*args) | Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the combined random, systematic and structured uncertainties on the measurand |
| *digital_effects_table.measurement_function.MeasurementFunction.run*(*args) | Function to calculate the measurand by running input quantities through measurement function. |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_total*(*args) | Function to propagate uncertainties for the total uncertainty component. |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_random*(*args) | Function to propagate uncertainties for the random uncertainty component. |

continues on next page

Table 1 – continued from previous page

| | |
|---|---|
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_systematic*(*args) | Function to propagate uncertainties for the systemtic uncertainty component. |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_structured*(*args) | Function to propagate uncertainties for the structured uncertainty component. |
| *digital_effects_table.measurement_function.MeasurementFunction.propagate_specific*(...) | Function to propagate uncertainties for a specific uncertainty component. |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils*(...) | Class with utility functions to help in the uncertainty propagation and handling of error correlations |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.find_comps*(...) | Function to find the components corresponding to provided uncertainty form (name or type). |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.get_input_qty*(*args) | Function to extract input quantities from datasets and return as list of arrays. |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.get_input_unc*(...) | Function to extract uncertainties on the input quantities from datasets and return as list of arrays. |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_unc*(...) | Function to extract uncertainties of given form on given variable from the given datasets and return as array. |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_unc_missingdim*(...) | Function to extract uncertainties of given form on given variable from the given datasets when there are missing dimensions. |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.get_input_corr*(...) | Function to extract error-correlation matrices for the input quantities from datasets and return as list of arrays. |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_corr*(...) | Function to extract error-correlation matrices of given form on given variable from the given datasets and return as array. |
| *digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_corr_missingdim*(...) | Function to extract error-correlation matrices of given form on given variable from the given datasets when there are missing dimensions. |
| *digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates*(...) | DigitalEffectsTableTemplates class allows to make templates for digital effects table creation for measurand |
| *digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.make_template_main*(...) | Make the digital effects table template for the case where random, systematic and structured uncertainties are propagated seperately |
| *digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.make_template_tot*(...) | Make the digital effects table template for the case where uncertainties are combined and only the total uncertainty is returned. |

continues on next page

Table 1 – continued from previous page

| | |
|---|---|
| *digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.make_template_specific*(...) | Make the digital effects table template for the case where uncertainties are combined and only the total uncertainty is returned. |
| *digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.remove_unc_component*(ds, ...) | Function to remove an uncertainty component from a dataset |
| *digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.join_with_preexisting_ds*(ds, ...) | Function to combine digital effects table, with previously populated dataset. |

### punpy.digital_effects_table.measurement_function.MeasurementFunction

**class** punpy.digital_effects_table.measurement_function.**MeasurementFunction**(*prop=None*, *xvariables=None*, *uncxvariables=None*, *yvariable=None*, *yunit=''*, *corr_between=None*, *param_fixed=None*, *repeat_dims=None*, *corr_dims=None*, *separate_corr_dims=False*, *allow_some_nans=True*, *ydims=None*, *refxvar=None*, *sizes_dict=None*, *use_err_corr_dict=False*, *broadcast_correlation='syst'*)

MeasurementFunction class which provides all the functionality for propagating uncertainties using obsarray digital effects tables. This class needs to be subclassed, and a meas_function needs to be provided for the measurement function to propagate uncertainties through.

**Parameters**

- **prop** (*punpy.MCPropagation or punpy.LPUPropagation*) – punpy MC propagation or LPU propagation object. Defaults to None, in which case a MC propagation object with 100 MC steps is used.
- **xvariables** (*list(str), optional*) – list of input quantity names, in same order as arguments in measurement function and with same exact names as provided in input datasets. Defaults to None, in which case get_argument_names function is used.
- **uncxvariables** (*list(str), optional*) – list of input quantity names for which uncertainties should be propagated. Should be a subset of input quantity names. Defaults to None, in which case uncertainties on all input quantities are used.

- **yvariable** (*str, optional*) – name of measurand. Defaults to None, in which case get_measurand_name_and_unit function is used.
- **yunit** (*str, optional*) – unit of measurand. Defaults to "" (unitless).
- **corr_between** (*numpy.ndarray , optional*) – Allows to specify the (average) error correlation coefficient between the various input quantities. Defaults to None, in which case no error-correlation is assumed.
- **param_fixed** (*list of bools, optional*) – set to true or false to indicate for each input quantity whether it has to remain unmodified either when expand=true or when using repeated measurements, defaults to None (no inputs fixed).
- **repeat_dims** (*str or int or list(str) or list(int), optional*) – Used to select the axis which has repeated measurements. Axis can be specified using the name(s) of the dimension, or their index in the ydims. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to -99, for which there is no reduction in dimensionality..
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated.
- **separate_corr_dims** (*bool, optional*) – When set to True and output_vars>1, corr_dims should be a list providing the corr_dims for each output variable, each following the format defined in the corr_dims description. Defaults to False
- **allow_some_nans** (*bool, optional*) – set to False to ignore any MC sample which has any nan's in the measurand. Defaults to True, in which case only MC samples with only nan's are ignored.
- **ydims** (*list(str), optional*) – list of dimensions of the measurand, in correct order. list of list of dimensions when there are multiple measurands. Default to None, in which case it is assumed to be the same as refxvar (see below) input quantity.
- **refxvar** (*string, optional*) – name of reference input quantity that has the same shape as measurand. Defaults to None
- **sizes_dict** (*dict, optional*) – Dictionary with sizes of each of the dimensions of the measurand. Defaults to None, in which cases sizes come from input quantites.
- **use_err_corr_dict** (*bool, optional*) – when possible, use dictionaries with separate error-correlation info per dimension in order to save memory
- **broadcast_correlation** (*str*) – correlation form ("rand" or "syst" to use when broadcasting

**__init__**(*prop=None*, *xvariables=None*, *uncxvariables=None*, *yvariable=None*, *yunit=''*, *corr_between=None*, *param_fixed=None*, *repeat_dims=None*, *corr_dims=None*, *separate_corr_dims=False*, *allow_some_nans=True*, *ydims=None*, *refxvar=None*, *sizes_dict=None*, *use_err_corr_dict=False*, *broadcast_correlation='syst'*)

### Methods

| | |
|---|---|
| *__init__*([prop, xvariables, uncxvariables, ...]) | |
| *get_argument_names*() | This function allows to return the names of the input quantities as a list of strings. |
| *get_measurand_name_and_unit*() | This function allows to return the name and unit of the measurand as strings. |
| *meas_function*(*args, **kwargs) | meas_function is the measurement function itself, to be used in the uncertainty propagation. |
| *propagate_ds*(*args[, store_unc_percent, ...]) | Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the combined random, systematic and structured uncertainties on the measurand |
| *propagate_ds_all*(*args[, store_unc_percent, ...]) | Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the combined random, systematic and structured uncertainties on the measurand |
| *propagate_ds_specific*(comp_list, *args[, ...]) | Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the uncertainties of specific components listed in comp_list. |
| *propagate_ds_total*(*args[, ...]) | Function to propagate the total uncertainties present in the digital effects tables in the input arguments, through the measurement function to produce an output digital effects table with the total uncertainties on the measurand |
| *propagate_random*(*args[, expand]) | Function to propagate uncertainties for the random uncertainty component. |
| *propagate_specific*(form, *args[, expand, ...]) | Function to propagate uncertainties for a specific uncertainty component. |
| *propagate_structured*(*args[, expand, ...]) | Function to propagate uncertainties for the structured uncertainty component. |
| *propagate_systematic*(*args[, expand]) | Function to propagate uncertainties for the systemtic uncertainty component. |
| *propagate_total*(*args[, expand, return_corr]) | Function to propagate uncertainties for the total uncertainty component. |
| *run*(*args[, expand]) | Function to calculate the measurand by running input quantities through measurement function. |
| *setup*(*args, **kwargs) | This function is to provide a setup stage that can be run before propagating uncertainties. |
| *update_measurand*(measurand, measurand_unit) | |

### punpy.digital_effects_table.measurement_function.MeasurementFunction.meas_function

**abstract** MeasurementFunction.**meas_function**(**args*, ***kwargs*)

meas_function is the measurement function itself, to be used in the uncertainty propagation. This function must be overwritten by the user when creating their MeasurementFunction subclass.

### punpy.digital_effects_table.measurement_function.MeasurementFunction.get_argument_names

MeasurementFunction.**get_argument_names**()

This function allows to return the names of the input quantities as a list of strings. Can optionally be overwritten to provide names instead of providing xvariables as a keyword.

**Returns**
names of the input quantities

**Return type**
list of strings

### punpy.digital_effects_table.measurement_function.MeasurementFunction.get_measurand_name_and_unit

MeasurementFunction.**get_measurand_name_and_unit**()

This function allows to return the name and unit of the measurand as strings. Can optionally be overwritten to provide names instead of providing yvariable as a keyword.

**Returns**
name of the measurand, unit

**Return type**
tuple(str, str)

### punpy.digital_effects_table.measurement_function.MeasurementFunction.update_measurand

MeasurementFunction.**update_measurand**(*measurand*, *measurand_unit*)

### punpy.digital_effects_table.measurement_function.MeasurementFunction.setup

MeasurementFunction.**setup**(**args*, ***kwargs*)

This function is to provide a setup stage that can be run before propagating uncertainties. This allows to set up additional class attributes etc, without needing to edit the initialiser (which is quite specific to this class). This function can optionally be overwritten by the user when creating their MeasurementFunction subclass.

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_ds

MeasurementFunction.**propagate_ds**(**args*, *store_unc_percent=False*, *expand=False*, *ds_out_pre=None*, *use_ds_out_pre_unmodified=None*, *include_corr=True*)

Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the combined random, systematic and structured uncertainties on the measurand

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **store_unc_percent** (*bool (optional)*) – Boolean defining whether relative uncertainties should be returned or not. Default to False (absolute uncertainties returned)
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.
- **ds_out_pre** (*xarray.dataset (optional)*) – Pre-existing output dataset in which the measurand & uncertainty variables should be saved. Defaults to None, in which case a new dataset is created.
- **use_ds_out_pre_unmodified** – bool to specify whether the ds_out_pre should be used unmodified, or whether the error-correlations etc should be worked out by punpy. defaults to None, in which case it is automatically set to True if yvariable is present as a variable in ds_out_pre, and else to False.
- **include_corr** (*bool (optional)*) – boolean to indicate whether the output dataset should include the correlation matrices. Defaults to True.

**Returns**
digital effects table with uncertainties on measurand

**Return type**
obsarray dataset

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_ds_total

MeasurementFunction.**propagate_ds_total**(**args*, *store_unc_percent=False*, *expand=False*, *ds_out_pre=None*, *use_ds_out_pre_unmodified=None*, *include_corr=True*)

Function to propagate the total uncertainties present in the digital effects tables in the input arguments, through the measurement function to produce an output digital effects table with the total uncertainties on the measurand

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **store_unc_percent** (*bool (optional)*) – Boolean defining whether relative uncertainties should be returned or not. Default to True (relative uncertaintie returned)
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.
- **ds_out_pre** (*xarray.dataset (optional)*) – Pre-existing output dataset in which the measurand & uncertainty variables should be saved. Defaults to None, in which case a new dataset is created.
- **include_corr** (*bool (optional)*) – boolean to indicate whether the output dataset should include the correlation matrices. Defaults to True.

**Returns**
digital effects table with uncertainties on measurand

**Return type**
obsarray dataset

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_ds_specific

MeasurementFunction.**propagate_ds_specific**(*comp_list*, **args*, *comp_list_out=None*, *store_unc_percent=False*, *expand=False*, *ds_out_pre=None*, *use_ds_out_pre_unmodified=None*, *include_corr=True*, *simple_random=True*, *simple_systematic=True*)

Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the uncertainties of specific components listed in comp_list.

**Parameters**

- **comp_list** – list of uncertainty contributions to propagate
- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **store_unc_percent** (*bool (optional)*) – Boolean defining whether relative uncertainties should be returned or not. Default to True (relative uncertaintie returned)
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.
- **ds_out_pre** (*xarray.dataset (optional)*) – Pre-existing output dataset in which the measurand & uncertainty variables should be saved. Defaults to None, in which case a new dataset is created.
- **include_corr** (*bool (optional)*) – boolean to indicate whether the output dataset should include the correlation matrices. Defaults to True.

**Rtype comp_list**
list of strings or string

**Returns**
digital effects table with uncertainties on measurand

**Return type**
obsarray dataset

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_ds_all

MeasurementFunction.**propagate_ds_all**(**args*, *store_unc_percent=False*, *expand=False*, *ds_out_pre=None*, *use_ds_out_pre_unmodified=False*, *include_corr=True*)

Function to propagate the uncertainties on the input quantities present in the digital effects tables provided as the input arguments, through the measurement function to produce an output digital effects table with the combined random, systematic and structured uncertainties on the measurand

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **store_unc_percent** (*bool (optional)*) – Boolean defining whether relative uncertainties should be returned or not. Default to True (relative uncertaintie returned)
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.

- **ds_out_pre** (*xarray.dataset (optional)*) – Pre-existing output dataset in which the measurand & uncertainty variables should be saved. Defaults to None, in which case a new dataset is created.
- **include_corr** (*bool (optional)*) – boolean to indicate whether the output dataset should include the correlation matrices. Defaults to True.

**Returns**
digital effects table with uncertainties on measurand

**Return type**
obsarray dataset

### punpy.digital_effects_table.measurement_function.MeasurementFunction.run

MeasurementFunction.**run**(**args*, *expand=False*)

Function to calculate the measurand by running input quantities through measurement function.

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.

**Returns**
measurand

**Return type**
numpy.ndarray

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_total

MeasurementFunction.**propagate_total**(**args*, *expand=False*, *return_corr=True*)

Function to propagate uncertainties for the total uncertainty component.

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.
- **return_corr** (*bool (optional)*) – boolean to indicate whether the measurand error-correlation matrices should be returned. Defaults to True.

**Returns**
uncertainty on measurand for total uncertainty component, error-correlation matrix of measurand for total uncertainty component

**Return type**
tuple(numpy.ndarray, numpy.ndarray)

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_random

MeasurementFunction.**propagate_random**(**args*, *expand=False*)

Function to propagate uncertainties for the random uncertainty component.

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.

**Returns**

uncertainty on measurand for random uncertainty component

**Return type**

numpy.ndarray

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_systematic

MeasurementFunction.**propagate_systematic**(**args*, *expand=False*)

Function to propagate uncertainties for the systemtic uncertainty component.

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.

**Returns**

uncertainty on measurand for systematic uncertainty component

**Return type**

numpy.ndarray

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_structured

MeasurementFunction.**propagate_structured**(**args*, *expand=False*, *return_corr=True*)

Function to propagate uncertainties for the structured uncertainty component.

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.
- **return_corr** (*bool (optional)*) – boolean to indicate whether the measurand error-correlation matrices should be returned. Defaults to True.

**Returns**

uncertainty on measurand for structured uncertainty component, error-correlation matrix of measurand for structured uncertainty component

**Return type**

tuple(numpy.ndarray, numpy.ndarray)

### punpy.digital_effects_table.measurement_function.MeasurementFunction.propagate_specific

MeasurementFunction.**propagate_specific**(*form*, **args*, *expand=False*, *return_corr=False*)

Function to propagate uncertainties for a specific uncertainty component.

**Parameters**

- **form** (*str*) – name or type of uncertainty component
- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **expand** (*bool (optional)*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand. Defaults to False.
- **return_corr** (*bool (optional)*) – boolean to indicate whether the measurand error-correlation matrices should be returned. Defaults to True.

**Returns**

uncertainty on measurand for specific uncertainty component, error-correlation matrix of measurand for specific uncertainty component

**Return type**

tuple(numpy.ndarray, numpy.ndarray)

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils

**class** punpy.digital_effects_table.measurement_function_utils.**MeasurementFunctionUtils**(*xvariables*, *uncxvariables*, *ydims*, *str_repeat_dims*, *str_repeat_noncorr_di*, *verbose*, *templ*, *use_err_corr_dict*, *broadcast_correlation*, *param_fixed*)

Class with utility functions to help in the uncertainty propagation and handling of error correlations

**Parameters**

- **xvariables** (*list(str)*) – list of input quantity names, in same order as arguments in measurement function and with same exact names as provided in input datasets.
- **uncxvariables** (*list(str), optional*) – list of input quantity names for which uncertainties should be propagated. Should be a subset of input quantity names. Defaults to None, in which case uncertainties on all input quantities are used.
- **ydims** (*list(str), optional*) – list of dimensions of the measurand, in correct order. list of list of dimensions when there are multiple measurands. Default to None, in which case it is assumed to be the same as refxvar (see below) input quantity.
- **str_repeat_dims** (*list(str)*) – Strings used to select the axis which has repeated measurements. Axis can be specified using the name(s) of the dimension.

- **str_repeat_noncorr_dims** (*list(str)*) – list of dimension names to be used as repeated dims
- **verbose** (*bool*) – boolean to set verbosity
- **templ** (*punpy.digital_effects_table_template*) – templater object
- **use_err_corr_dict** (*bool, optional*) – when possible, use dictionaries with separate error-correlation info per dimension in order to save memory
- **broadcast_correlation** (*str*) – correlation form ("rand" or "syst" to use when broadcasting
- **param_fixed** (*list of bools, optional*) – set to true or false to indicate for each input quantity whether it has to remain unmodified either when expand=true or when using repeated measurements, defaults to None (no inputs fixed).

**__init__**(*xvariables*, *uncxvariables*, *ydims*, *str_repeat_dims*, *str_repeat_noncorr_dims*, *verbose*, *templ*, *use_err_corr_dict*, *broadcast_correlation*, *param_fixed*)

### Methods

| | |
|---|---|
| *__init__*(xvariables, uncxvariables, ydims, ...) | |
| *calculate_corr*(form, ds, var) | Function to extract error-correlation matrices of given form on given variable from the given datasets and return as array. |
| *calculate_corr_missingdim*(form, ds, var[, ...]) | Function to extract error-correlation matrices of given form on given variable from the given datasets when there are missing dimensions. |
| *calculate_unc*(form, ds, var) | Function to extract uncertainties of given form on given variable from the given datasets and return as array. |
| *calculate_unc_missingdim*(form, ds, var[, ...]) | Function to extract uncertainties of given form on given variable from the given datasets when there are missing dimensions. |
| *find_comps*(form, dataset, var[, ...]) | Function to find the components corresponding to provided uncertainty form (name or type). |
| *get_input_corr*(form, *args[, ydims, ...]) | Function to extract error-correlation matrices for the input quantities from datasets and return as list of arrays. |
| *get_input_qty*(*args[, ydims, sizes_dict, expand]) | Function to extract input quantities from datasets and return as list of arrays. |
| *get_input_unc*(form, *args[, ydims, ...]) | Function to extract uncertainties on the input quantities from datasets and return as list of arrays. |

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.find_comps

MeasurementFunctionUtils.**find_comps**(*form*, *dataset*, *var*, *store_unc_percent=False*, *ydims=None*)

Function to find the components corresponding to provided uncertainty form (name or type).

**Parameters**

- **form** (*str*) – name or type of uncertainty component
- **dataset** (*xarray.dataset*) – dataset being querried
- **var** (*str*) – variable for which uncertainty components are being listed.
- **store_unc_percent** (*bool (optional)*) – Boolean defining whether relative uncertainties should be returned or not. Default to True (relative uncertaintie returned)
- **ydims** (*list(str)*) – list of dimensions of the measurand, in correct order.

**Returns**

list of names of uncertainty components

**Return type**

list(str)

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.get_input_qty

MeasurementFunctionUtils.**get_input_qty**(**args*, *ydims=None*, *sizes_dict=None*, *expand=False*)

Function to extract input quantities from datasets and return as list of arrays.

**Parameters**

- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **ydims** (*list(str)*) – list of dimensions of the measurand, in correct order.
- **sizes_dict** (*dict*) – Dictionary with sizes of each of the dimensions of the measurand.
- **expand** (*bool*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand.

**Returns**

list of values (as np.ndarray) for each of the input quantites.

**Return type**

list(np.ndarray)

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.get_input_unc

MeasurementFunctionUtils.**get_input_unc**(*form*, **args*, *ydims=None*, *sizes_dict=None*, *expand=False*, *corr_dims=[]*)

Function to extract uncertainties on the input quantities from datasets and return as list of arrays.

**Parameters**

- **form** (*str*) – name or type of uncertainty component
- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **ydims** (*list(str)*) – list of dimensions of the measurand, in correct order.

- **sizes_dict** (*dict*) – Dictionary with sizes of each of the dimensions of the measurand.
- **expand** (*bool*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand.
- **corr_dims** (*integer, optional*) – set to positive integer to select the axis used in the correlation matrix. The correlation matrix will then be averaged over other dimensions. Defaults to -99, for which the input array will be flattened and the full correlation matrix calculated.

**Returns**
list of uncertainty values (as np.ndarray) for each of the input quantites.

**Return type**
list(np.ndarray)

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_unc

MeasurementFunctionUtils.**calculate_unc**(*form*, *ds*, *var*)

Function to extract uncertainties of given form on given variable from the given datasets and return as array.

**Parameters**

- **form** (*str*) – name or type of given uncertainty component
- **ds** (*xarray.dataset*) – given dataset
- **var** (*str*) – given variable

**Returns**
uncertainty values for the given variable in the given dataset. returns None if uncertainty component is not present in dataset.

**Return type**
np.ndarray

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_unc_missingdim

MeasurementFunctionUtils.**calculate_unc_missingdim**(*form*, *ds*, *var*, *ydims=None*, *sizes_dict=None*, *expand=False*)

Function to extract uncertainties of given form on given variable from the given datasets when there are missing dimensions. With missing dimension, we here mean a dimension that is present in the measurand, but not in the input quantity being considered.

**Parameters**

- **form** (*str*) – name or type of uncertainty component
- **ds** (*xarray.dataset*) – given dataset
- **var** (*str*) – given variable
- **ydims** (*list(str)*) – list of dimensions of the measurand, in correct order.
- **sizes_dict** (*dict*) – Dictionary with sizes of each of the dimensions of the measurand.
- **expand** (*bool*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand.

**Returns**
uncertainty values for the given variable in the given dataset. returns None if uncertainty component is not present in dataset.

**Return type**
np.ndarray

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.get_input_corr

MeasurementFunctionUtils.**get_input_corr**(*form*, **args*, *ydims=None*, *sizes_dict=None*, *expand=False*)

Function to extract error-correlation matrices for the input quantities from datasets and return as list of arrays.

**Parameters**

- **form** (*str*) – name or type of uncertainty component
- **args** (*obsarray dataset(s)*) – One or multiple digital effects tables with input quantities, defined with obsarray
- **ydims** (*list(str)*) – list of dimensions of the measurand, in correct order.
- **sizes_dict** (*dict*) – Dictionary with sizes of each of the dimensions of the measurand.
- **expand** (*bool*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand.

**Returns**
list of error-correlation matrix values (as np.ndarray) for each of the input quantities.

**Return type**
list(np.ndarray)

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_corr

MeasurementFunctionUtils.**calculate_corr**(*form*, *ds*, *var*)

Function to extract error-correlation matrices of given form on given variable from the given datasets and return as array.

**Parameters**

- **form** (*str*) – name or type of given uncertainty component
- **ds** (*xarray.dataset*) – given dataset
- **var** (*str*) – given variable

**Returns**
error-correlation matrix values for the given variable in the given dataset. returns None if uncertainty component is not present in dataset.

**Return type**
np.ndarray

### punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils.calculate_corr_missingdim

MeasurementFunctionUtils.**calculate_corr_missingdim**(*form*, *ds*, *var*, *ydims=None*, *sizes_dict=None*, *expand=False*)

Function to extract error-correlation matrices of given form on given variable from the given datasets when there are missing dimensions. With missing dimension, we here mean a dimension that is present in the measurand, but not in the input quantity being considered.

**Parameters**

- **form** (*str*) – name or type of uncertainty component
- **ds** (*xarray.dataset*) – given dataset
- **var** (*str*) – given variable
- **ydims** (*list(str)*) – list of dimensions of the measurand, in correct order.
- **sizes_dict** (*dict*) – Dictionary with sizes of each of the dimensions of the measurand.
- **expand** (*bool*) – boolean to indicate whether the input quantities should be expanded/broadcasted to the shape of the measurand.

**Returns**

error-correlation matrix values for the given variable in the given dataset. returns None if uncertainty component is not present in dataset.

**Return type**

np.ndarray

### punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates

**class** punpy.digital_effects_table.digital_effects_table_templates.**DigitalEffectsTableTemplates**(*yvariable*, *yunit*, *output_vars*)

DigitalEffectsTableTemplates class allows to make templates for digital effects table creation for measurand

**Parameters**

- **yvariable** (*string*) – name of the measurand variable
- **yunit** (*string*) – unit of the measurand

**__init__**(*yvariable*, *yunit*, *output_vars*)

### Methods

| | |
|---|---|
| *__init__*(yvariable, yunit, output_vars) | |
| *join_with_preexisting_ds*(ds, ds_pre[, drop]) | Function to combine digital effects table, with previously populated dataset. |
| *make_template_main*(dims, dim_sizes[, ...]) | Make the digital effects table template for the case where random, systematic and structured uncertainties are propagated seperately |
| *make_template_specific*(comp_list, dims, ...) | Make the digital effects table template for the case where uncertainties are combined and only the total uncertainty is returned. |
| *make_template_tot*(dims, dim_sizes[, ...]) | Make the digital effects table template for the case where uncertainties are combined and only the total uncertainty is returned. |
| *remove_unc_component*(ds, variable, u_comp[, ...]) | Function to remove an uncertainty component from a dataset |

## punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.make_template_main

DigitalEffectsTableTemplates.**make_template_main**(*dims*, *dim_sizes*, *str_corr_dims=[]*, *separate_corr_dims=False*, *str_repeat_noncorr_dims=[]*, *store_unc_percent=False*, *repeat_dim_err_corrs=[]*)

Make the digital effects table template for the case where random, systematic and structured uncertainties are propagated seperately

**Parameters**

- **dims** (*list*) – list of dimensions
- **u_xvar_ref** (*xarray.Variable*) – reference uncertainty component that is used to populate repeated dims
- **str_repeat_noncorr_dims** (*str or list of str, optional*) – set to (list of) string(s) with dimension name(s) to select the axis which has repeated measurements. The calculations will be performed seperately for each of the repeated measurments and then combined, in order to save memory and speed up the process. Defaults to [], for which there is no reduction in dimensionality..

**Returns**

measurand digital effects table template to be used by obsarray

**Return type**

dict

### punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.make_template_tot

DigitalEffectsTableTemplates.**make_template_tot**(*dims*, *dim_sizes*, *str_corr_dims=[]*, *separate_corr_dims=False*, *str_repeat_noncorr_dims=[]*, *store_unc_percent=False*, *repeat_dim_err_corrs=[]*)

Make the digital effects table template for the case where uncertainties are combined and only the total uncertainty is returned.

**Parameters**

- **dims** (*list*) – list of dimensions
- **u_xvar_ref** (*xarray.Variable*) – reference uncertainty component that is used to populate repeated dims

**Returns**
measurand digital effects table template to be used by obsarray

**Return type**
dict

### punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.make_template_specific

DigitalEffectsTableTemplates.**make_template_specific**(*comp_list*, *dims*, *dim_sizes*, *str_corr_dims=[]*, *separate_corr_dims=False*, *str_repeat_noncorr_dims=[]*, *store_unc_percent=False*, *repeat_dim_err_corrs=[]*, *simple_random=True*, *simple_systematic=True*)

Make the digital effects table template for the case where uncertainties are combined and only the total uncertainty is returned.

**Parameters**

- **dims** (*list*) – list of dimensions
- **u_xvar_ref** (*xarray.Variable*) – reference uncertainty component that is used to populate repeated dims

**Returns**
measurand digital effects table template to be used by obsarray

**Return type**
dict

### punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.remove_unc_component

DigitalEffectsTableTemplates.**remove_unc_component**(*ds*, *variable*, *u_comp*, *err_corr_comp=None*)

Function to remove an uncertainty component from a dataset

**Parameters**

- **ds** (*xr.Dataset*) – dataset from which uncertainty should be removed
- **variable** (*str*) – variable from which uncertainty should be removed
- **u_comp** (*str*) – name of uncertainty component that should be removed

- **err_corr_comp** (*str*) – optionally, the name of the error correlation matrix component associated with this uncertainty component can be provided, so it is removed as well.

**Returns**
dataset which uncertainty component removed

**Return type**
xr.Dataset

### punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates.join_with_preexisting_ds

DigitalEffectsTableTemplates.**join_with_preexisting_ds**(*ds*, *ds_pre*, *drop=None*)

Function to combine digital effects table, with previously populated dataset. Only the measurand is overwritten.

**Parameters**

- **ds** (*xarray.Dataset*) – digital effects table for measurand, created by punpy
- **ds_pre** (*xarray.Dataset*) – previously populated dataset, to be combined with digital effects table
- **drop** – list of variables to drop
- **drop** – List(str)

**Returns**
merged digital effects table

**Return type**
xarray.Dataset

# INDEX

## Symbols


__init__() (*punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates method*), 62
__init__() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 50
__init__() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 58
__init__() (*punpy.lpu.lpu_propagation.LPUPropagation method*), 40
__init__() (*punpy.mc.mc_propagation.MCPropagation method*), 28


## C


calculate_corr() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 61
calculate_corr_missingdim() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 62
calculate_unc() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 60
calculate_unc_missingdim() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 60
combine_samples() (*punpy.mc.mc_propagation.MCPropagation method*), 38


## D


DigitalEffectsTableTemplates (*class in punpy.digital_effects_table.digital_effects_table_templates*), 62


## F


find_comps() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 59


## G


generate_MC_sample() (*punpy.mc.mc_propagation.MCPropagation method*), 35
generate_MC_sample_cov() (*punpy.mc.mc_propagation.MCPropagation method*), 35
get_argument_names() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 52
get_input_corr() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 61
get_input_qty() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 59
get_input_unc() (*punpy.digital_effects_table.measurement_function_utils.MeasurementFunctionUtils method*), 59
get_measurand_name_and_unit() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 52


## J


join_with_preexisting_ds() (*punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates method*), 65


## L


LPUPropagation (*class in punpy.lpu.lpu_propagation*), 39


## M


make_template_main() (*punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates method*), 63
make_template_specific() (*punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates method*), 64
make_template_tot() (*punpy.digital_effects_table.digital_effects_table_templates.DigitalEffectsTableTemplates method*), 64
MCPropagation (*class in punpy.mc.mc_propagation*), 28
meas_function() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 52
MeasurementFunction (*class in punpy.digital_effects_table.measurement_function*), 49
MeasurementFunctionUtils (*class in punpy.digital_effects_table.measurement_function_utils*), 57

## P

process_jacobian() (*punpy.lpu.lpu_propagation.LPUPropagation method*), 46
process_samples() (*punpy.mc.mc_propagation.MCPropagation method*), 38
propagate_cov() (*punpy.lpu.lpu_propagation.LPUPropagation method*), 44
propagate_cov() (*punpy.mc.mc_propagation.MCPropagation method*), 33
propagate_cov_flattened() (*punpy.mc.mc_propagation.MCPropagation method*), 36
propagate_ds() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 52
propagate_ds_all() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 54
propagate_ds_specific() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 54
propagate_ds_total() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 53
propagate_flattened_cov() (*punpy.lpu.lpu_propagation.LPUPropagation method*), 45
propagate_random() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 56
propagate_random() (*punpy.lpu.lpu_propagation.LPUPropagation method*), 41
propagate_random() (*punpy.mc.mc_propagation.MCPropagation method*), 30
propagate_specific() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 57
propagate_standard() (*punpy.lpu.lpu_propagation.LPUPropagation method*), 40
propagate_standard() (*punpy.mc.mc_propagation.MCPropagation method*), 29
propagate_structured() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 56
propagate_systematic() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 56
propagate_systematic() (*punpy.lpu.lpu_propagation.LPUPropagation method*), 43
propagate_systematic() (*punpy.mc.mc_propagation.MCPropagation method*), 32
propagate_total() (*punpy.digital_effects_table.measurement_function.MeasurementFunction method*), 55

## R

remove_unc_component() (*punpy.digital_effects_table.digital_effects_table_templates.Digit method*), 64
run() (*punpy.digital_effects_table.measurement_function.MeasurementFu method*), 55
run_samples() (*punpy.mc.mc_propagation.MCPropagation method*), 37

## S

setup() (*punpy.digital_effects_table.measurement_function.Measurement method*), 52

## U

update_measurand() (*punpy.digital_effects_table.measurement_function. method*), 52