RichValues
– Python Library –
User Guide

Andrés Megías Toledano
Center for Astrobiology (CAB), Madrid

Version 1.0
November 2022

Index

1. Introduction

The library RichValues is a Python 3 library whose purpose is to deal with numeric values with uncertainties and upper/lower limits, which we will call *rich values*.[1] With it, one can easily import rich values written in plain text documents in an easily readable format, operate with them propagating the uncertainties automatically, and export them in the same formatting style as the import.

For example, to represent the value $(6.3 \pm 0.4) \cdot 10^3$, we could write `6.3 +/- 0.4 e3`, and for the upper limit $< 1.2 \cdot 10^5$ we could write `< 1.2 e5`. Then these text strings would be parsed as Python objects containing all the necessary information to describe the rich value, and they will be displayed in the screen in the same formatting style. The main way to do so is to write the text string between simple quotation marks and write it inside the function `rich_value`; for example, `rich_value('6.3 +/- 0.4 e3')`. Then, we could make arithmetic operations between them and other rich values or even usual numbers, like addition, subtraction, multiplication, and division.

2. Installation

This library RichValues works in Python 3. It requires the modules `copy`, `math`, and `itertools`, from the Python's standard library, and also the libraries NumPy and Pandas. This library is published in GitHub:

https://github.com/andresmegias/richvalues .

In order to install it, you can use the Python Package Installer (PyPI), running from the terminal the following command:

```
pip3 install richvalues
```

.

You can also use the Conda package installer, running instead the following command:

```
conda install richvalues -c richvalues
```

.

Alternatively, you can install the library manually in your computer. To do so, you have to download the file `richvalues.py` and copy it to one of the Python system paths. To display the list of all these paths, you can run Python (for example, from the terminal, writing `python3`), and run the two following commands:

```
import sys
sys.path
```

For example, for Mac, one of these paths would be:

```
/Users/<user>/Library/Python/<3.x>/lib/python/site-packages
```

,

with `<user>` being your username and `<3.x>` your exact Python version.

Now, RichValues can be used in a Python script like any other library, with the name richvalues. Therefore, to import it you would write:

1. Like *rich text* for text with information about the font type, size, weight, etc.

import richvalues .

You can also use an abbreviation for the library name, like rv:

import richvalues as rv .

In the following examples of code, we will consider that we already imported RichValues with this abbreviation.

3. Formatting style for representing rich values

The RichValues library uses a specific formatting style to represent the different kinds of rich values with plain text. This is the way in which they are displayed in the screen and in which they can be imported and exported. It can also be used to create rich values within a script. Below are the rules for this formatting style:

- If the value has no uncertainty, just put the number.
- It it has uncertainty, you can join the central value and the uncertainty with `+/-` or `+-`, using blank spaces if you want; for example: `5.2 +/- 0.4`.
- If it has a lower and an upper uncertainty, you should write the lower uncertainty just after the central value preceded by `-`, and then write the upper uncertainty preceded by `+`, using blank spaces if you want; for example: `5.2 -0.3+0.4`.
- If you want to use scientific notation (exponential notation with decimal base), for the central value and the uncertainty (or uncertainties), you just have to write an `e`, separated by a blank space from the numbers and followed by the exponent argument; for example: `5.2 -0.3+0.4 e-3`.
- If you want to specify an upper or lower limit, just write `<` or `>` before the value (without uncertainty), putting a blank space if you want; for example: `< 5.2 e2`.
- If you want to specify the domain of the value, that is, the minimum and maximum values that the magnitude corresponding to this value could take, you have to write it between brackets, using the text `inf` to represent infinity; for example: `5.2 -0.3+0.4 e3 [0,inf]`. By default (that is, if it is not specified), the domain is all the real numbers, that is: `[-inf,inf]`.
- If you wanted to specify a constant interval of values, you should write the edges of the interval separated by two hyphens (`--`), without uncertainties; for example; `9 e3 -- 62 e3`.

4. Creation of rich values

With the library RichValues, we can create individual rich values and use them in tuples, lists, dictionaries, etc., but we can also create arrays –based on NumPy arrays– and tables –based on Pandas dataframes.

4.1. Individual rich values

There are mainly two ways to create a single rich value: with the function `rich_value` and with the class RichValue, being the first one the easiest way.

4.1.1. Function rich_value

Below are the arguments of this function, although the first one is the only required:

- **text**. Text string representing the rich value, using the formatting style explained in section 3.
- **num_sf**. Number of significant figures to use for displaying the elements of the rich array, taking into account the uncertainties. By default it is the number of scientific figures used in the rich value represented by the input text string.
- **min_exp**. Minimum exponent in absolute value to be used for displaying the rich value in scientific notation. By default it is 4.
- **domain**. List containing the edges of the domain of all the entries of the rich array, that is, the minimum and maximum values that the magnitude corresponding to this array could take. By default, it is the domain showed in the input text string, but if it is not specified it will be [-np.inf, np.inf].[2]
- **len_sample**. In case a function has to be applied to any element of the rich array and there is no analytic formula for propagating the uncertainties, it will be calculated from distributions with a sample of this size. By default it is 10^4, that is, 10000.
- **allow_log_scale**. If this logical variable is set to True, the elements of the rich array will be displayed as a decimal power in case that the interval defined by the uncertainties is greater than 5 times the central value. By default it is True if the input rich value is displayed as an exponent in decimal base and False else.

The default values for these arguments, and for most of the arguments of the following functions, can be modified as explained in section 7.

Below is a simple example of how we would create two different rich values:

```
x = rv.rich_value('5.2 +/- 0.3')
y = rv.rich_value('2.1 -0.4+0.5')
```

In this way, we have created a Python object of class RichValue. These objects will be displayed on screen with the previous formatting style except for the domain, which will be hidden. For example:

```
[in]    rv.rich_value('0.327 -0.12+0.18 [-1,1]')
[out]   0.33-0.12+0.18
```

Instead of doing this, we could have defined our rich value directly with the class RichValue.

4.1.2. Class RichValue

This is the main class of the library, which is based around it.

Arguments

Below are the arguments of this Python class, being the first one the only required one:

2. Assuming that the library NumPy was imported as the abbreviation np (import numpy as np).

- **center**. Central value of the rich value, or value of the upper/lower limit.[3]
- **unc**. Uncertainty associated with the central value. If two values are given, they would be the lower and upper uncertainties. By default it is 0.
- **is_lolim**. Logical variable that determines if the rich value is a lower limit. By default it is False.
- **is_uplim**. Logical variable that determines if the rich value is an upper limit. By default it is False.
- **num_sf**. Number of significant figures to use for displaying the elements of the rich array, taking into account the uncertainties. By default it is 1.
- **min_exp**. Minimum exponent in absolute value to be used for displaying the rich value in scientific notation. By default it is 4.
- **domain**. List containing the edges of the domain of all the entries of the rich array, that is, the minimum and maximum values that the magnitude corresponding to this array could take. By default, it is [-np.inf, np.inf].[4]
- **len_sample**. In case a function has to be applied to any element of the rich array and there is no analytic formula for propagating the uncertainties, it will be calculated from distributions with a sample of this size. By default it is 10000.
- **allow_log_scale**. If this logical variable is set to True, the elements of the rich array will be displayed as a decimal power in case that the interval defined by the uncertainties covers more than one order of magnitude. By default it is False.

As an example, below is a code to create the same rich values as in the examples of the previous section:

```
x = rv.RichValue(5.2, 0.3)
y = rv.RichValue(2.1, [0.4,0.5])
z = rv.RichValue(0.327, [0.12,0.18], domain=[-1,1])
```

Instance variables

All of the arguments of the RichValue class correspond to instance variables with the same name.[5] To access to them, you should write a dot after the name of the rich value Python variable and then the name of the instance variable. For example, to access to the instance variable center of a certain rich value x, you would write x.center. Additionally, there are other two instance variables for this class:

- **is_lim**. Logical variable that is True if the rich value is an upper/lower limit.
- **is_range**. Logical variable that is True if the rich value is a constant interval of values or an upper/lower limit.

---

3. If you wanted to specify a constant interval of values for the rich value, this should be a list with its bounds.
4. Assuming that the library NumPy was imported as the abbreviation np (import numpy as np).
5. An instance variable is a variable contained by an object of a certain class.

- **lim_for_extra_sf**. If the significand/mantissa[6] of the uncertainty of the rich value (if it is a centered value) or its main value (if it is an upper/lower limit, or an edge of an interval) is lower than this limit number, the rich value will be displayed with an additional significant figure. By default it is 2.5 (see section 7 for changing this default value).

Methods

The RichValue class have several methods.[7] Some of them just provide additional information on the rich value but others are more complex and can be used to modify the rich value. Below is the list of all of them with their arguments between brackets:

- **rel_unc**(self).[8] It returns the relative uncertainties of the rich value.
- **ampl**(self). It returns the distances between the central value and the bounds of the domain, which we call *amplitudes*.
- **rel_ampl**(self). It returns the distances between the central value and the bounds of the domain divided by the uncertainties.
- **latex**(self, mult_symbol). It returns a text string of a LaTeX code representing the rich value, using mult_symbol as the multiplication symbol in scientific notation (by default, mult_symbol = \\cdot).[9]
- **check_limit**(self, sigmas). If the rich value is not an upper/lower limit, it checks if the interval defined by the central value plus and minus sigmas times its uncertainties surpasses the domain of the rich value in any direction. If this only happens in one direction, the rich value will be converted to an upper/lower limit; if it happens in both directions, the rich value will be considered to be undetermined, and will be displayed as nan.
- **check_interval**(self). If any of the uncertainties of the rich value is greater or equal than the distance to the closest domain edge, it treats the rich value as a finite interval defined by the central value and the uncertainties.
- **set_lims_factor**(self, c). If the rich value is an upper/lower limit, the uncertainty instance variable (unc) will be replaced as the central value divided by the factor c (by default, c = 4.0). This can be useful when plotting upper/lower limits with the function errorbar from the library Matplotlib.
- **pdf**(self, x). It applies the probability density function (PDF) associated with the rich value to the given array x, returning the resulting array.
- **sample**(self, N). It returns a sample of size N of the distribution associated with the rich value, whose corresponding PDF can be explored with the previous method, pdf.

6. The number multiplying the decimal power.
7. A method is a function that can be called within an object of a certain class.
8. The arguments self means that the method has to be called within an object of this class, but it is not actually written when calling the method; for example: if x is a rich value, to call the method rel_unc we should write x.rel_unc().
9. In Python, two backslashes are needed in order to display just one in a text string.

By default, N is equal to the variable len_sample.

- **inv**(self). It returns the inverse of the rich value.
- **function**(self, function, unc_function, sigmas, domain). It applies the given function (function) to the rich value. If the uncertainties are relatively small compared with the distance to the closest bound of the domain, the calculation can be optimized providing the function of the uncertainty (unc_function) with respect to the central value and the uncertainty (so it should have only those two arguments, in that order); the exact criterion to apply this is that any of the distances from the central value to the edges of the domain (*amplitudes*) relative to the corresponding uncertainty is greater than the number sigmas. The domain of the resulting rich value can be fixed with the argument domain, in order to reduce a bit the computation time. By default, domain = [-np.inf, np.inf] and sigmas = 20.0.

The last two methods allow to apply a function to the rich value and therefore to obtain a new one. See next section for more details on making operations with rich values.

4.1.3. Operations with rich values

The RichValue class has special methods for the basic arithmetic operations: addition, subtraction, multiplication, division, and power. Therefore, you can just use the arithmetic operators (+, -, *, /, **) to combine a rich value with another rich value or with a usual number. The uncertainties will be propagated automatically, and the condition of an upper/lower limit (or even a finite interval) will be taken into account to obtain the final result. For example:

```
[in]    x = rv.rich_value('5.2 +/- 0.3')
[in]    y = rv.rich_value('2.1 -0.4+0.5')
[in]    x + y
[out]   7.3-0.5+0.6

[in]    x = rv.rich_value('< 5.2 [0,inf]')
[in]    x + 3
[out]   3.0 -- 8.2

[in]    x = rv.rich_value('5.2 +/- 0.3')
[in]    x**2
[out]   27+/-3
```

At this point, there is one thing that must be taken into account. Every rich value variable present in a mathematical expression will be treated by Python as an independent variable, even if there is one variable repeated in the expression. Therefore, in that case you may not get the desired result. For example, x*x is not equal to x**2, as in the first case Python will treat each x as independent from the other x. In any case, if the relative uncertainties are small, the differences will be mainly in the uncertainties. For example:

```
[in]    x = rv.rich_value('5.20 +/- 0.30')
[in]    print(x*x, x**2)
[out]   27.04+/-2.21  27.0+/-3.1

[in]    x = rv.rich_value('3.60 +/- 0.40')
[in]    print(x+x, 2*x)
```

```
[out]   7.20+/-0.57  7.20+/-0.80
```

In case you cannot simplify your expression so that it only contains each variable once, you can use the function `function_with_rich_values`, where you can also write the mathematical expression you want to compute, and this time each repeated variable will be treated as it. Also, if you want to apply more than 2 or 3 operations and the uncertainties are relatively high, it will be faster and a bit more precise to use `function_with_rich_values`.

Function `function_with_rich_values`

Using this function one can compute any expression involving rich values. Below is the full list of its arguments, although only the two first ones are mandatory:

- **`function`**. Python function to be applied to the given rich values. If its expression is short, it can be defined within the list of arguments using a lambda function.[10]
- **`args`**. List with the input rich values, in the same order as the arguments of the given function.
- **`len_args_sample`**. Size of the samples of the arguments. The default is the number of arguments times the mean of the sample size for all the arguments.
- **`unc_function`**. Python function used to estimate the uncertainties, in case that analytic uncertainty propagation can be applied. The arguments should be the central values first and then the uncertainties, with the same order as in the input function.[11] If it is not specified, the analytic uncertainty propagation will never be used.
- **`sigmas`**. Threshold to apply uncertainty propagation. The value is the distance to the bounds of the domain relative to the uncertainty. By default it is `20.0`.
- **`use_sigma_combs`**. Logical variable that determines if the calculation of the uncertainties is optimized when uncertainty propagation can be performed but there is no uncertainty function provided. It performs combinations of the central value plus and minus the uncertainties for every argument and applies the function, taking the minimum and maximum of the resulting values as bounds to compute the uncertainties. By default it is `False`, as it is not fully tested for more than one argument.
- **`lims_fraction`**. In case the resulting value is an upper/lower limit, this factor is used to calculate the limit. It can take values from 0 to 1, and the closest it is to 1, the closest the resulting limit will be to the function applied to the central/limit value of the arguments. By default it is `0.1`.
- **`num_reps_lims`**. Number of repetitions of the sampling done in the cases of having an upper/lower limit for better estimating its value. By default it is `4`. Greater values are recommended if `lims_fraction` is greater than `0.1`.

10. A lambda function is defined in the following way: first you write `lambda` followed by a blank space; now, you write the arguments of the function with any desired letter and separated by commas, ending with a colon, and followed optionally by a blank space; finally, you write the mathematical expression of the function using the same letters you used before. For example: `lambda a,b: a+b` .

11. For example: `lambda a,b,da,db: da+db` .

- **num_sf**. Number of significant figures to display the resulting value. By default, it is the maximum of this variable for all the arguments.
- **min_exp**. Minimum exponent in absolute value to be used for displaying the resulting rich value in scientific notation. By default it is the minimum of this variable for all the arguments.
- **domain**. Domain of the result. If not specified, it will be estimated automatically.
- **len_sample**. Sample size of the resulting rich value. By default, it is the mean of this variable for all the arguments.
- **allow_log_scale**. If this logical variable is set to True, the elements of the resulting rich array will be displayed as a decimal power in case that the interval defined by the uncertainties covers more than one order of magnitude. By default, it is True if this variable is True for any of the arguments.

Let's see an example of the use of this function.

```
[in]    x = rv.rich_value('5.2 +/- 0.3')
[in]    y = rv.rich_value('2.1 -0.4+0.5')
[in]    rv.function_with_rich_values(lambda a,b: a+b, [x,y])
[out]   7.3-0.5+0.6
```

That would be it. As you can see, the basic use of this function is quite simple.

4.2. Arrays of rich values

There are three ways to create arrays of rich values within this library: using just NumPy arrays of rich values, using the function rich_array, and using directly the class RichArray. We will call these arrays *rich arrays*.

4.2.1. NumPy arrays

Using the functions from the library NumPy (like the function array), one can create arrays whose elements are rich values (of class RichValue). Below is a simple example of a creation of an array of rich values using the NumPy function array and the two ways of creating rich values mentioned in section 2.1.[12]

```
import numpy as np
u = np.array([rich_value('1.21 +/- 0.14'), rich_value('< 4')])
u = np.array([RichValue(1.21, 0.14), RichValue(4, is_uplim=True)])
```

With this method, we do not need any additional functions or classes more than those of NumPy. However, we recommend using one of the two other ways of creating arrays of rich values, as they are simpler and are more flexible.

4.2.2. Function rich_array

Below are the arguments of this function, although the first one is the only required:

- **array**. It should be a list, an array or similar. The elements of the input argument can

12. For the rest of code examples, we will assume that we imported NumPy as np (import numpy as np).

be either a rich value (of class RichValue) or a text string representing a rich value, that is, with the formatting style explained in section 3.

- **num_sf**. Number of significant figures to use for displaying the elements of the rich array, taking into account the uncertainties. By default, it will be the maximum of this variable for all the elements of the input array.
- **min_exp**. Logical variable that determines if the elements of the rich array will be displayed in scientific notation. By default, it will be True if any of the elements of the input array has this variable as True. Minimum exponent in absolute value to be used for displaying the elements of the rich array in scientific notation, if use_exp is True. By default it is 0, that is, there is no minimum exponent.
- **domain**. List containing the edges of the domain of all the entries of the rich array, that is, the minimum and maximum values that the magnitude corresponding to this array could take. By default, it will be the union of the domains of all the elements of the input array.
- **len_sample**. In case a function has to be applied to any element of the rich array and there is no analytic formula for propagating the uncertainties, it will be calculated from distributions with a sample of this size. By default, it will be the mean of this variable for all the elements of the input array.
- **allow_log_scale**. If this logical variable is set to True, the elements of the rich array will be displayed as an exponent in decimal base. By default, it will be True if any of the elements of the input array has this variable as True.
- **check_limits**. If this logical variable is set to True, if any of the 1 σ interval bounds of any element of the resulting rich array equals or exceeds the domain, the value will be considered as an upper/lower limit. By default it is True.
- **check_intervals**. If this logical variable is set to True, if any of the uncertainties of each of the entries of the rich array is greater or equal than the distance to the closest domain edge, it treats the corresponding rich value as a finite interval defined by the central value and the uncertainties.. By default it is True.

For example, the creation of the same array as in the previous example would be:

```
u = rv.rich_array(['1.21 +/- 0.14', '< 4'])
```

In this way, we create an object of class RichArray, which is basically the NumPy class ndarray with some additional features. Alternatively, we could have created the rich value directly using the class RichArray.

4.2.3. Class RichArray

This class is inherited from the class ndarray from NumPy. An object of this class is basically a NumPy array but with some additional instance variables and methods. It is useful when you already have an array with the central values of a certain variable and another array with the corresponding uncertainties.

Arguments

Below are the arguments of this Python class, being the first element the only required:

- **centers**. Array of central values.
- **uncs**. Array of lower and upper uncertainties associated with the central values. By default, it is an array of 0 values.
- **are_lolims**. Array of logical variables that indicate if each central value is actually a lower limit. By default, it is an array of False values.
- **are_uplims**. Array of logical variables that indicate if each central value is actually an upper limit. By default, it is an array of False values.
- **are_ranges**. Array of logical variables that indicate if each central value is actually a finite interval or an upper/lower limit. By default, it is an array of False values.

Additionally, you can specify the arguments num_sf, min_exp, domain, len_sample, and allow_log_scale, to set these RichValue properties to all the elements of the rich array.

As an example, below is a code to create the same rich array as in the previous section:

```
u = rv.RichArray([1.21,4], [0.14,0], are_uplims=[False,True])
```

Instance variables

All of the proper arguments of the RichArray class correspond to instance variables with the same name.[13] Additionally, there are other two instance variables for this class:

- **are_lims**. Array of logical variables that are True if the corresponding entry of the rich array is an upper/lower limit.
- **are_ranges**. Array of logical variables that are True if the corresponding entry of the rich array is a finite interval or an upper/lower limit.

Methods

The RichArray class have several proper methods. Some of them just provide additional information on the rich array but others are more complex and can be used to modify it. Below is the list of all of them with their arguments between brackets:

- **rel_unc**(self). It returns the relative uncertainties of each element of the rich array.
- **ampl**(self). It returns the distances between the central value and the bounds of the domain (which we call *amplitudes*), for each element of the rich array.
- **rel_ampl**(self). It returns the distances between the central value and the bounds of the domain divided by the uncertainties, for each element of the rich array.
- **latex**(self, mult_symbol). It returns an array of text strings of a LaTeX code representing each entry of the rich array, using mult_symbol as the multiplication symbol in scientific notation (by default, mult_symbol = \\cdot).
- **check_limits**(self, sigmas). For each element of the rich array, if the rich value is not an upper/lower limit, it checks if the interval defined by the central value plus and minus

13. As it was explained in section 2.1.2, to access to an instance variable you should write a dot after the name of the rich value Python variable and then the name of the instance variable; for example, to access to the instance variable centers of a certain rich array u, you would write u.centers.

sigmas times its uncertainties surpasses the domain of the rich value in any direction. If this only happens in one direction, the rich value will be converted to an upper/lower limit; if it happens in both directions, the rich value will be considered to be undetermined, and will be displayed as nan.

- **check_intervals**(self). For each element of the rich array, if any of the uncertainties of the rich value is greater or equal than the distance to the closest domain edge, it treats the corresponding rich value as a finite interval defined by the central value and the uncertainties.. By default it is True.
- **set_lims_factor**(self, c). For each element of the rich array, if the rich value is an upper/lower limit, the uncertainty instance variable (unc) will be defined as the central value divided by the factor c (by default, c = 4.0). If two values are provided in the variable c, the first one will be user for lower limits and the second one for upper limits. This can be useful when plotting upper/lower limits with the function errorbar from the library Matplotlib.
- **sample**(self, N). For each entry of the rich array, it returns a sample of size N of the distribution associated with the corresponding rich value. By default, N is equal to the variable len_sample.
- **function**(self, function, unc_function, sigmas, domain). It applies the given function (function) to every element of the rich array. If the uncertainties are relatively small compared with the distance to the closest bound of the domain, the calculation can be optimized providing the function of the uncertainty (unc_function) with respect to the central value and the uncertainty (so it should have only those two arguments, in that order); the exact criterion to apply this is that any of the distances from the central value to the edges of the domain (*amplitudes*) relative to the corresponding uncertainty is greater than the number sigmas. The domain of the resulting rich array can be fixed with the argument domain, in order to reduce a bit the computation time. By default, domain = [-np.inf, np.inf] and sigmas = 10.0.

The last method allows to apply a function to the rich array and therefore to obtain a new one. See next section for more details on making operations with rich arrays.

There is one important thing that must be taken into account regarding the inherited methods from NumPy arrays. When using an inherited method which is not exclusive from the class RichArray, the resulting object may loose all the instance variables and methods proper of the class RichArray, depending on the method. In this case you should convert the resulting array to a rich array using the function rich_array. This is not necessary for slicing nor for the methods transpose, reshape, flatten, and ravel.

4.2.4. Operations with arrays of rich values

The RichArray class has special methods for the basic arithmetic operations: addition, subtraction, multiplication, division, and power. Therefore, you can just use the arithmetic operators (+, -, *, /, **) to combine a rich array with another rich array, another rich value or a usual number. The uncertainties will be propagated automatically, and the condition of an upper/lower limit (or even a finite interval) will be taken into account to obtain the final result. For example:

```
[in]     u = rv.rich_array(['1.2 +/- 0.4', '5.8 +/-0.9'])
[in]     v = rv.rich_array(['8 +/- 3', '< 21'])
[in]     u * v
[out]    RichArray([10+/-5, < 150], dtype=object)
```

```
[in]     u = rv.rich_array(['1.2 +/- 0.4', '5.8 +/-0.9'])
[in]     u + rich_value(2.0 +/- 0.3)
[out]    RichArray([3.2+/-0.5, 7.8+/-0.9], dtype=object)
```

If instead of working with objects of class RichArray you work with NumPy arrays (class ndarray) whose elements are rich values (class RichValue), you can perform this kind of operations as well, as the class RichValue also have methods for them, as explained is section 2.1.3.

However, similar to what happens with the class RichValue, if in the mathematical expression one rich array appears more than once, it will not be treated as the same variable appearing twice but as two independent variables. Therefore, for those cases, you should use the function function_with_rich_arrays, which is like function_with_rich_values but for rich arrays. If instead of obtaining a rich array as an output you would like to obtain just one rich value, you can use function_with_rich_values with the elements of the input rich arrays as arguments. Additionally, there is another function that you can use to apply a custom mean to a rich array, called rich_fmean. This three ways are explained below.

Function function_with_rich_arrays

Using this function one can compute any expression involving rich arrays, element by element. Its arguments are exactly the same as function_with_rich_values, except that in args the arguments, that should be contained in a list, are rich arrays.

Let's see an example of the use of this function.

```
[in]     u = rv.rich_array(['1.2 +/- 0.4', '5.8 +/-0.9'])
[in]     v = rv.rich_array(['8 +/- 3', '< 21'])
[in]     rv.function_with_rich_arrays(lambda a,b: a*b, [u,v])
[out]    RichArray([9-4+5, < 150], dtype=object)
```

That would be it. As with function_with_rich_values, the basic use of this function is quite simple.

Function function_with_rich_values

This function can be used in order to obtain one rich value as a result of an operation applied to rich arrays. To do so, the arguments have to be the elements of the input rich arrays, and the function to apply has to be defined with respect to these arguments in the same order.

Below there is an example for computing the scalar product between two rich arrays u and v. First, we define the function to apply (dot) and then we use function_with_rich_values.

```
[in]     u = rv.rich_array(['1.2 +/- 0.4', '5.8 +/-0.9'])
```

```
[in]    v = rv.rich_array(['8 +/- 3', '< 21'])
[in]    def dot(*args):
[in]        L = len(args) // 2
[in]        u, v = args[:L], args[L:]
[in]        y = sum([ui*vi for ui,vi in zip(u,v)])
[in]        return y
[in]    rv.function_with_rich_values(dot, [*u,*v])
[out]   < 170
```

Function rich_fmean

This function applies a generalized quasi-arithmetic mean (or f-mean) along all the elements of the input array, which can be a rich array.[14] To use it, you have to specify the function to be used for the mean and its inverse. Below is the list of its arguments:

- **array**. Input array to apply the mean.
- **function**. Function that defines the mean. By default, it is the identity (which corresponds to the arithmetic mean).
- **inverse_function**. Inverse of the function that defines the mean. By default, it is the identity (which corresponds to the arithmetic mean).
- **weights**. Weights to be applied to the values of the input array. By default, they are equal weights.
- **weight_function**. Function to be applied to the weights before normalization. By default, it is the identity (no operation is applied).

Besides these arguments, you can write more arguments of the function function_with_rich_arrays (or, equivalently, of function_with_rich_values).

Let's see an example of the use of this function for calculating the arithmetic mean.

```
[in]    u = rv.rich_array(['1.2 +/- 0.4', '5.8 +/-0.9'])
[in]    rv.rich_fmean(u)
[out]   3.5+/-0.5
```

And now, the same example but with the geometric mean.

```
[in]    u = rv.rich_array(['1.2 +/- 0.4', '5.8 +/-0.9'], domain=[0,np.inf])
[in]    rv.rich_fmean(u, function=np.log, inverse_function=np.exp)
[out]   2.6+/-0.5
```

4.3. Tables of rich values

With this library one can also create tables of rich values, that is, groups of rich values labeled with a column name and an index. This is done using Pandas dataframes, so we will call this objects *rich dataframes*. There are three ways of creating this kind of object: with Pandas dataframes, with the function rich_dataframe, and with the class RichDataFrame.

14. If f is a function with an inverse function f^{-1}, $\{x_i\}$ is a set of input numbers, and w_i are a set of weights for the input numbers, the generalized f-mean of $\{x_i\}$ is: $\langle x \rangle = f^{-1}\left(\frac{1}{\sum_i w_i} \sum_i w_i\, f(x_i)\right)$.

4.3.1. Pandas dataframes

Using the library Pandas, one can create arrays whose elements are rich values (of class RichValue). Below is a simple example of a creation of an array of rich values using the Pandas class DataFrame.[15]

```
import pandas as pd
m = rv.rich_array([['2.1+/-0.3','3.4+/-0.4','<4'],['5','<6','8+/-1']])
df = pd.DataFrame(m, columns=['a','b','c'])
```

We could have created the dataframe from a Python dictionary as well.[16]

```
d = {'a': rich_array(['2.1+/-0.3','5']),
     'b': rich_array(['3.4+/-0.4','<6']),
     'c': rich_array(['<4','8+/-1'])}
df = pd.DataFrame(d)
```

With this method, we do not need any additional functions or classes more than those of Pandas. However, we recommend using one of the two other ways of creating dataframes of rich values, as they are simpler and are more flexible.

4.3.2. Function `rich_dataframe`

Below are the arguments of this function, although the first one is the only required:

- **`df`**. It should be a dataframe, whose elements can be either a rich value (of class RichValue) or a text string representing a rich value, that is, with the formatting style explained in section 2.1.1.
- **`num_sf`**. Dictionary containing, for each column, the number of significant figures to use for displaying the elements of the rich array, taking into account the uncertainties. By default, it is `1` for all columns. By default, for each column, it will be the value for the first entry if it is a rich value, and `1` if not.
- **`min_exp`**. Dictionary containing, for each column, the minimum exponent in absolute value used to display the elements of the rich array in scientific notation. By default, for each column, it will be the value for the first entry if it is a rich value, and `False` if not.
- **`len_sample`**. Dictionary containing, for each column, the size of the sample of the elements of the dataframe, if they are rich values. By default, for each column, it will be the value for the first entry if it is a rich value, and `10000` if not.
- **`domain`**. Dictionary containing, for each column, the domain of its elements, that is, the minimum and maximum values that the magnitude corresponding to each column could take. By default, for each column, it will be the value for the first entry if it is a rich value, and `[-np.inf, np.inf]` if not.
- **`allow_log_scale`**. Dictionary containing, for each column, a logical variable that deter-

15. For the rest of code examples, we will assume that we imported Pandas as `pd` (`import pandas as pd`).

16. A Python dictionary is an object than includes several variables identified by a keyword. To create one, you have to write the pair of keywords and variables (name of the keyword, a colon, and the variable) separated by commas, and all of this enclosed by keys; for example: `d = {'a': 1, 'b': 2}`.

mines if the elements of the rich array will be displayed as a decimal power in case that the interval defined by the uncertainties is greater than 5 times the central value. By default, for each column, it will be the value for the first entry if it is a rich value, and False if not.

- **check_limits**. If this logical variable is set to True, if any of the 1 σ interval bounds of any element of the rich dataframe equals or exceeds the domain, the value will be considered as an upper/lower limit. By default it is True.
- **check_intervals**. If this logical variable is set to True, if any of the uncertainties of each of the entries of the rich dataframe is greater or equal than the distance to the closest domain edge, it treats the corresponding rich value as a finite interval defined by the central value and the uncertainties.. By default it is True.

As an example, below is a code to create a simple dataframe either an array or a dictionary, as in the previous section.

```
m = [['2.1+/-0.3','3.4+/-0.4','<4'],['5','<6','8+/-1']]
df = rv.rich_dataframe(m, columns=['a','b','c'])
d = {'a': ['2.1+/-0.3','5'],  'b': ['3.4+/-0.4','<6'], 'c': ['<4','8+/-1']}
df = rv.rich_dataframe(d)
```

As you can see, this method is easier and faster than the one of the previous section. Also, if any element of the input list/array/dictionary for creating the dataframe is a non-numeric text string, it will be preserved to the final rich dataframe. In this way, we create an object of class RichDataFrame, which is basically the Pandas class DataFrame with some additional features. Alternatively, we could have created the rich dataframe using the class RichDataFrame, although it is always easier and more practical to use the function rich_dataframe.

4.3.3. Class RichDataFrame

This class is inherited from the class DataFrame from Pandas. An object of this class is basically a Pandas dataframe but with some additional methods.

Argument

The only argument required for this class is a dataframe containing rich values. It cannot have entries with text strings representing rich values; in that case, the function rich_dataframe should be applied to the input dataframe before calling the class RichDataFrame.

Methods

The RichDataFrame class has all the DataFrame methods that allow to modify its values, and also has some additional methods exclusive of the RichDataFrame class. Below is the list of all of these additional methods:

- **get_params**(self). It returns a dictionary of dictionaries, each of them containing the value of each of the RichValue parameters (num_sf, min_exp, domain, len_sample, and allow_log_scale) used for each column of the dataframe.
- **latex**(self, return_df, row_sep, mult_symbol). If return_df = True, it returns a

dataframe of text strings of a LaTeX code representing each entry of the original dataframe, using mult_symbol as the multiplication symbol in scientific notation (by default, mult_symbol = \\cdot). Instead, if return_df = False (by default), it returns a text strings in LaTeX formatting style representing the content of a table, using row_sep as the text that indicates the end of a row (by default, row_sep = \\tabularnewline).

- **set_lims_factor**(self, limits_factors). For each element of each column of the dataframe, if the rich value is an upper/lower limit, the uncertainty instance variable (unc) will be replaced as the central value divided by the factor specified for each column by the variable limits_factors (by default, it is 4.0 for every column). If two values are provided in the variable for a certain column, the first one will be user for lower limits and the second one for upper limits. This can be useful when plotting upper/lower limits with the function errorbar from the library Matplotlib.
- **create_column**(self, function, columns, **kwargs). It returns a new column of type rich array obtained applying the given function (function) to the specified columns (columns) of the dataframe. The rest of the arguments (**kwargs) are the same as in funtion_with_rich_values.
- **create_row**(self, function, rows, **kwargs). It returns a new row obtained applying the given function (function) to the specified rows (rows) of the dataframe. The rest of the arguments (**kwargs) are the same as in function_with_rich_values.

The last method allows to apply a function to the rich dataframe and obtain a new column or row, which can then be added to the dataframe.

4.3.4. Operations with rich tables of rich values

The class RichDataFrame has special methods for the basic arithmetic operations: addition, subtraction, multiplication, division, and power. Therefore, you can just use the arithmetic operators (+, -, *, /, **) to combine a rich dataframe with another rich dataframe, another rich value or a usual number. The uncertainties will be propagated automatically, and the condition of an upper/lower limit (or even a finite interval) will be taken into account to obtain the final result. Note that this will only work if all the entries of the dataframe are numeric.

Similar to what happens with the classes RichValue and RichArray, if in the mathematical expression one rich dataframe variable appears more than once, it will not be treated as the same variable appearing twice but as two independent variables. Therefore, for those cases, you should use the methods create_column and create_rows. These methods are also useful if you want to make operations with the data within the dataframe.

Below is an example of how to create a new column in a dataframe combining its columns with the method create_column.

```
d = {'a': ['6.4+/-0.5','8'], 'b': ['3.4+/-0.4','<6'], c': ['4','8+/-1']}
df = rv.rich_dataframe(d, domain=[0,np.inf])
new_column = df.create_column(lambda a,b,c: a/b+c, ['a','b','c'])
```

```
df['d'] = new_column
```

5. Importing and exporting of rich values

The importing and exporting of rich values is very simple.

5.1. Importing

For the importing from a plain text file, you can write text strings representing rich values, with the formatting style explained in section 2.1.1. Then, any function that can import text strings will be fine, like NumPy's `loadtxt` or Pandas' `read_csv`. For example, below is a simple table from a file in .csv that could be imported into a rich dataframe.

```
Source \ Molecule   HCCCN              CH3CN
L1517B              5.16+/-0.03 e13    2.1+/-0.3 e11
L1498               1.6+/-0.3 e13      < 8.3 e10
L1544               1.0+/-0.3 e14      1.5+/-0.2 e11
B1-a                4.2+/-1.2 e12      4.9+/-1.1 e11
B1-c                4.2+/-3.4 e12      3.5+/-0.6 e11
SVS 4-5             1.1+/-0.3 e13      5.2+/-0.9 e11
GM Aur              1.9-0.4+0.4 e13    2.1-0.1+0.2 e12
As 209              2.9-0.5+0.5 e13    1.7-0.2+0.2 e12
HD 163296           7.3-1.9+2.5 e13    2.3-0.2+0.2 e12
MWC 480             7.8-2.7+3.9 e13    3.5-0.2+0.2 e12
# (units: /cm2)
46P                 < 0.003            0.017+/-0.001
67P                 0.0004             0.0059
# (units: % /H2O)
```

It contains the abundances of two molecules (HCCCN and CH_3CN) in different astronomical objects. Assuming that the file is named `table.csv`, we could import the data as:

```
df = pd.read_csv(table.csv, index_col=0, comment='#')
df = rv.rich_dataframe(df, domain=[0,np.inf])
```

As the abundances can only be positive, we specified a domain of `[0,np.inf]`; alternatively, we could have specified the domain in the first entry of each numeric column, as the function `rich_dataframe` will take, for each column, the domain of the first rich value as the domain of the whole column; that is, we should have written `5.16+/-0.03 e13 [0,inf]` and `2.1+/-0.3 e13 [0,inf]` as the first value of the columns labeled as HCCCN and CH3CN, respectively. The result of the importing will be a rich dataframe whose indexes are the first original column of the table in the .csv file, and with two columns with numeric values named HCCCN and CH3CN.

5.2. Exporting

For the exporting of a variable containing rich values to a plain text file, you can use any function that can export text strings, like NumPy's `savetxt` or Pandas' `to_csv` (which is a method for the class `DataFrame`). For example, below is a code for adding a new column to the example table of the previous section as the ratio between the two numeric columns, and exporting it to a .csv file.

```
df['ratio'] = df['HCCCN'] / df['CH3CN']
df.to_csv(table-ratio.csv)
```

And the resulting file would look like the table below.

Source \ Molecule	HCCCN	CH3CN	ratio
L1517B	5.16+/-0.03 e13	2.1+/-0.3 e11	240-30+40
L1498	1.6+/-0.3 e13	< 8 e10	> 100
L1544	1.0+/-0.3 e14	1.50+/-0.20 e11	670-210+230
B1-a	4.2+/-1.2 e12	4.9+/-1.1 e11	8-3+4
B1-c	4+/-3 e12	3.5+/-0.6 e11	12-9+11
SVS 4-5	1.1+/-0.3 e13	5.2+/-0.9 e11	21-6+8
GM Aur	1.9+/-0.4 e13	2.10-0.10+0.20 e12	8.9-1.9+2.0
As 209	2.9+/-0.5 e13	1.70+/-0.20 e12	17-3+4
HD 163296	7.3-1.9+2.5 e13	2.30+/-0.20 e12	32-8+11
MWC 480	8-3+4 e13	3.50+/-0.20 e12	22-8+11
46P	< 3 e-3	1.70+/-0.10 e-2	< 0.21
67P	4.0 e-4	5.9 e-3	0.068

We lost the comments (starting with #) of the original table, but we could actually have added it adding to the dataframe a row with the comment as the index and an empty string ('') as the value for each column. Or, alternatively, we could just add the comments directly to the resulting .csv file.

6. Changing the default parameters

Most of the default parameters of the functions and classes of this library can be modified through the variable defaultparams. This is a Python dictionary containing the values of the default variables with a specific name. Below is a list with all the parameter names, the corresponding variable names used in the library and their description:

- **number of significant figures**. Number of significant figures to display the rich values (usually num_sf). By default it is 1.
- **minimum exponent for scientific notation**. Minimum exponent used to display the number in scientific notation (usually min_exp). By default it is 4.
- **domain**. Domain of the rich values, that is, the minimum and maximum values that the variables associated with the rich values can take. By default it is all the real numbers, that is, [-np.inf, np.inf].
- **size of samples**. Size of the sample of the distribution associated with the rich value used to estimate the uncertainty propagation (usually len_sample). By default it is 10^4, that is, 10000.
- **allow logarithmic scale**. Logical variable (usually allow_log_scale) that allows the rich value to be displayed in exponential notation as a decimal power in case that the interval defined by the uncertainties is greater than the central value times a certain factor, defined by the variable minimum relative uncertainty range to use logarithmic scale (which is 5.0 by default). By default, this variable is False.

- **limit for extra significant figure**. If the significand/mantissa[17] of the uncertainty of the rich value (if it is a centered value) or its main value (if it is an upper/lower limit, or an edge of an interval) is lower than this limit number, the rich value will be displayed with an additional significant figure. By default it is 2.5, and it is called lim_for_extra_sf in some functions. If you want that limit to be excluded from the extra significant figure, you can define this variable as your limit minus a tiny number; for example: `2.5 - 1e-15`.
- **minimum relative uncertainty range to use logarithmic scale**. If the variable allow logarithmic scale is True, rich values will be displayed as a decimal power in case that the interval defined by the uncertainties relative to the central value is greater than this value. By default it is 5.0.
- **minimum relative distance to the domain edges to apply analytic uncertainty propagation**. This value defines when to apply the analytic approximation for uncertainty propagation in the operations with rich values. If the distances from the central value to the domain edges divided by the corresponding uncertainties are lower than this value, the approximation will be applied instead of creating samples from the distributions associated with the rich values, hence reducing the computation time. By default it is 20.0, and it is called sigmas in some functions.
- **use 1-sigma combinations to estimate uncertainty propagation**. Logical variable (usually called use_sigma_combs) that determines if the propagation of uncertainties is approximated with the use of the corresponding function applied to the central value plus and minus its uncertainties. By default it is False.
- **fraction of the central value for upper/lower limits**. Variable used in function_with_rich_values with the name lims_fraction. In case the rich value resulting of applying the corresponding function is an upper/lower limit, this factor is used to calculate the limit. It can take values from 0 to 1, and the closest it is to 1, the closest the resulting limit will be to the function applied to the central/limit value of the arguments. By default it is 0.1.
- **number of repetitions to estimate upper/lower limits**. Variable used in function_with_rich_values with the name num_reps. It is the number of repetitions of the sampling done in the cases of having an upper/lower limit for better estimating its value. By default it is 4. Greater values are recommended if fraction of the central value for upper/lower limits is greater than 0.1.
- **decimal exponent to define zero**. When performing operations and creating samples from distributions, any number lower in absolute value than the decimal power of this value will be considered as 0. By default it is -90.0.
- **decimal exponent to define infinity**. When performing operations and creating samples from distributions, any number lower in absolute value than the decimal power of this value will be considered as ∞. By default it is 90.0.

17. The number multiplying the decimal power.

- **`multiplication symbol for scientific notation in LaTeX`**. Symbol to be used when displaying a value in scientific notation in LaTeX mathematical mode for the multiplication of the significand/mantissa and the decimal power. By default it is `\\cdot`.[18]

7. Mathematical basis of operations with rich values

Here is explained the algorithm used to make the operations between rich values, that is, to apply a certain function f to a group of rich values. Let's consider a group of n rich values x_i, which can be a value with a central value μ_i and lower and upper uncertainties $(\sigma_{i,1}, \sigma_{i,2})$, an upper/lower limit, or even a finite interval. Each rich value has a probability density function (PDF) associated with it, depending on the properties of the rich value.

As each rich value has a PDF, for each of them we can draw a sample of a large number of values (which we choose to be $m \gtrsim 10^4\, n$), obtaining a set of n distributions $\{x_i\}$; or, grouped different, we have m groups of n values, each group containing a specific value of each variable x_i. Then, we can apply the function f to each of the m groups of values, obtaining a new distribution of values, $\{f(\{x_i\})\}$. Lastly, we have to determine if the distribution is localized around a certain value, in which case it would represent a rich value with a central value and lower and upper uncertainties, of it is more sparse, in which case it would an upper/lower limit or a finite interval.

In the following subsections we will see the different PDFs used for each kind of rich value. As for the exact algorithm of detecting the type of rich value that corresponds to the final distribution, it is not explained, but it can be inspected in the source code of the function `function_with_rich_values`.

7.1. Centered values

Let's consider a group of n variables x_i, with central values μ_i and uncertainties σ_i. This means that the probability density function (PDF) of the variable x_i is centered around μ_i with a width of the order of σ_i, so that the $1\,\sigma$ confidence interval (which includes 68.27 % of the distribution) is $(\mu_i - \sigma_i,\ \mu_i + \sigma_i)$. We define the left and right amplitudes, a_1 and a_2, as the distances between the limits of the domain and the median, that is, $a_j = |b_j - \mu|$ for $j = 1,2$. Now, as these amplitudes can be different, we will split our desired PDF in two halfs, one for $x < \mu$ and other for $x \geq \mu$. Then, we will use an amplitude a as a reference, which must be greater than the uncertainty, $a > \sigma$.

To propagate the uncertainties through a function f applied to the variables $\{x_i\}$, we can draw a sample of a large number of values ($\gtrsim 10^4\, n$) of each variable x_i, apply the function to each of the elements of the samples, and then obtain a central value and an uncertainty for the resulting distribution. To do so, we need two things: an appropiate PDF for converting each variable x_i to a distribution of values, and a proper method to obtain a central value and an uncertainty from the resulting distribution. For the last task, we can use the mean or the

18. In Python, two backslashes are needed in order to display just one in a text string.

median as the central value,[19] and the 1 σ (68.27 %) to obtain the lower and upper uncertainty (with respect to the central value). As for the PDF, it will depend of the domain of the variable.

We will define our PDFs for the case of a variable x with a central value μ and an uncertainty σ, building the final PDF with two halfs with amplitude $a = a_j$, $j = 1,2$. In case we had lower and upper uncertainties, σ_1 and σ_2, we should just replace σ by σ_1 for the left half of the PDF and by σ_2 for the right half. This construction of the final PDF by two halfs creates a discontinuity in the median, which is noticeable if any of the two relative amplitudes (a_j / σ_j) is relatively small (high uncertainty) and both of them are considerably different. Ideally, this discontinuity should not appear; however, as long as we do not have a PDF that can directly accept the parameters (μ, σ_1, σ_2, a_1, a_2), this procedure can be considered a good approximation.

7.1.1. Normal distribution

If the domain of the variable x is $(-\infty, \infty)$, a proper function is the well-known gaussian function:

$$f(x) = \frac{1}{\tau^{1/2}\sigma} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) ,$$

with $\tau \equiv 2\pi$.[20] This would led to a normal distribution.

7.1.2. Bounded normal distribution

If the domain of the variable is not $(-\infty, \infty)$, the normal distribution would be incorrect. Therefore, we have to use another function as the PDF.

Let's suppose a domain (b_1, b_2). If the amplitude is quite greater than the uncertainty, $a \gg \sigma$, a good PDF would be just the normal distribution truncated to the domain (b_1, b_2). However, for amplitudes closer to the uncertainty, it would be clearly incorrect, as the truncation shifts the median of the distribution and modifies the confidence intervals, and thus the uncertainties.

To fix this, we make the following variable change:

$$x - \mu \;\rightarrow\; \tilde{x} - \mu \;\equiv\; \frac{4}{\tau}\, a \tan\left(\frac{\tau}{4}\, \frac{x-\mu}{a}\right) .$$

Using this new variable $\tilde{x}$ with a normal distribution, we are able to compress the original domain of $(-\infty, \infty)$ to $(-a, a)$. However, we get two disadvantages. Firstly, the normalization constant is now different, and second, the relation between the parameter of the standard deviation of the original normal distribution and the 1 σ confidence interval (from which we define the uncertainty, σ) is now different; therefore, we should rename the standard deviation of the original normal distribution to s, which we will call *width*.

The first change is not a problem, as the PDF will be normalized computationally for each

19. We prefer to use the median, as it is more robust to outliers.
20. See https://tauday.com/tau_manifesto.pdf .

case. As for the second one, we have characterized computationally the relation between σ and s. It happens that σ/s decreases with a/σ, having $\sigma/s \simeq 2.65$ when $a/\sigma = 2$ and $\sigma/s \to 1$ when $a/\sigma \to \infty$. We have then a relation between the width, the uncertainty and the amplitude, that is, $s = s(\sigma, a)$.

Therefore, the resulting PDF, which we call *bounded gaussian function*, would be the following:

$$f(x) \propto \exp\left(-\frac{1}{2}\left(\frac{\frac{4}{\tau}\; a\; \tan\left(\frac{\tau}{4}\;\frac{x-\mu}{a}\right)}{s(\sigma\,,a)}\right)^2\right) \;,$$

which would led to a *bounded normal distribution* (see figure 1). However, it happens that σ/s increases quasi-exponentially as a/σ approaches a minimum value of ~ 1.55. Therefore, we must use another PDF for the cases in which $a/\sigma \lesssim 1.55$. Actually, for that limit case the shape of this PDF is almost that of a uniform distribution between $-a$ and a. We can model this PDF as a trapezoidal function with a rectangular core and triangles in the edges. Doing so, one can easily demonstrate that this kind of PDF would only work if a/σ is greater than the inverse of the integrated area corresponding to the 1 σ confidence interval (~ 0.683); that is, $a/\sigma \gtrsim 1.46$, which is consistent with the found behaviors of the calculated relationship $s(\sigma, a)$. In order to be conservative and for the sake of simplicity, the bounded normal will only be used for $a \geq 2\,\sigma$. Note that for $a \gg \sigma$, this PDF tends to a gaussian, which is a good behaviour.

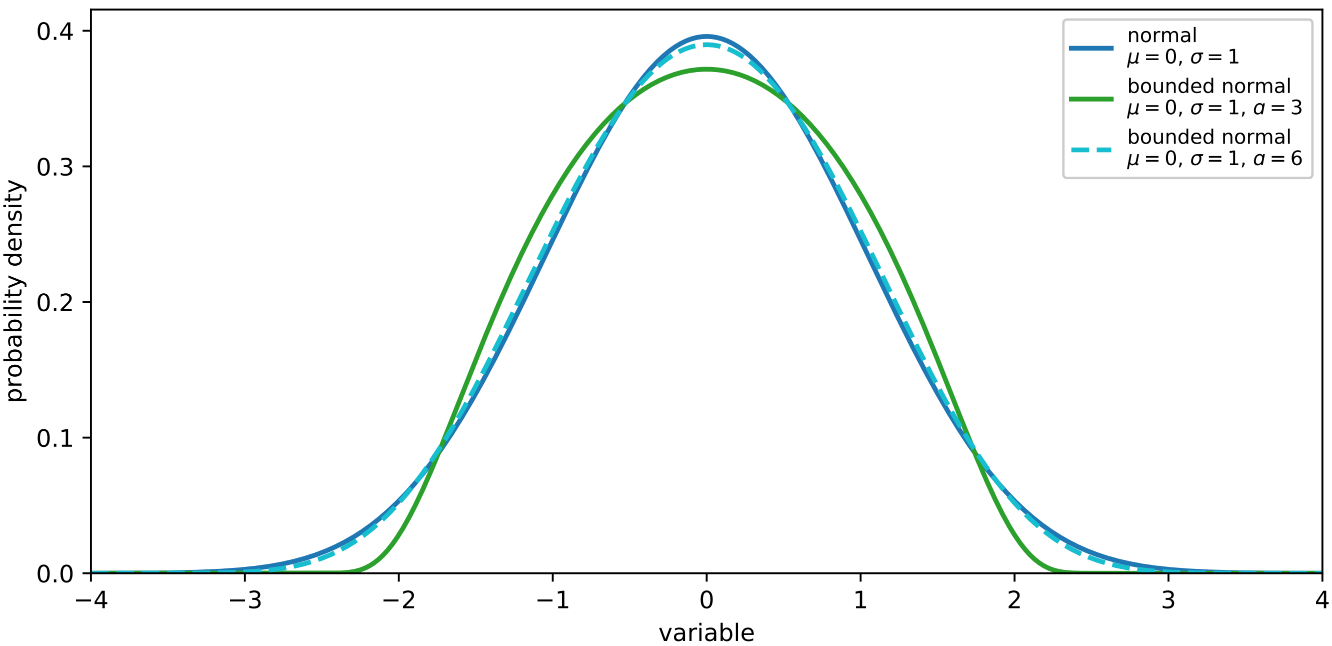

Figure 1. Probability density functions for a normal distribution and two bounded normal distributions.

7.1.3. Mirrored lognormal distribution

Finally, for $\sigma < a < 2\,\sigma$, we use a *mirrored log-gaussian function*:

$$f(x) = \frac{1}{\tau^{1/2}\,\ln(1-\sigma/a)}\,\frac{1}{\left|x-(\mu-a)\right|}\,\exp\left(-\frac{1}{2}\left(\frac{\left|x-(\mu-a)\right|-\ln(a)}{\ln(1-\sigma/a)}\right)^2\right) \;,$$

which is a lognormal function shifted and mirrored, and which would led to a *mirrored lognormal distribution* (see figure 2). This PDF meets our requirements: the median is equal to μ and the uncertainty associated with the 1 σ interval is σ. It may seem that the shape of the function is not good, as its median is not equal to its maximum. However, there is no way to avoid this if the domain is too small. We could think about using a trapezoidal function, like discussed in the previous section, but we showed that this would only work for $a/\sigma \gtrsim 1.46$. Therefore, the PDF should have some kind of central dip, like in the mirrored lognormal distribution. Although ideally we would like to have a continuous function for all the possible values of $a/\sigma \geq 1$ (for example, a continuous transition between the bounded gaussian and the mirrored log-gaussian), for the moment this can be considered a good approximation to that ideal situation.

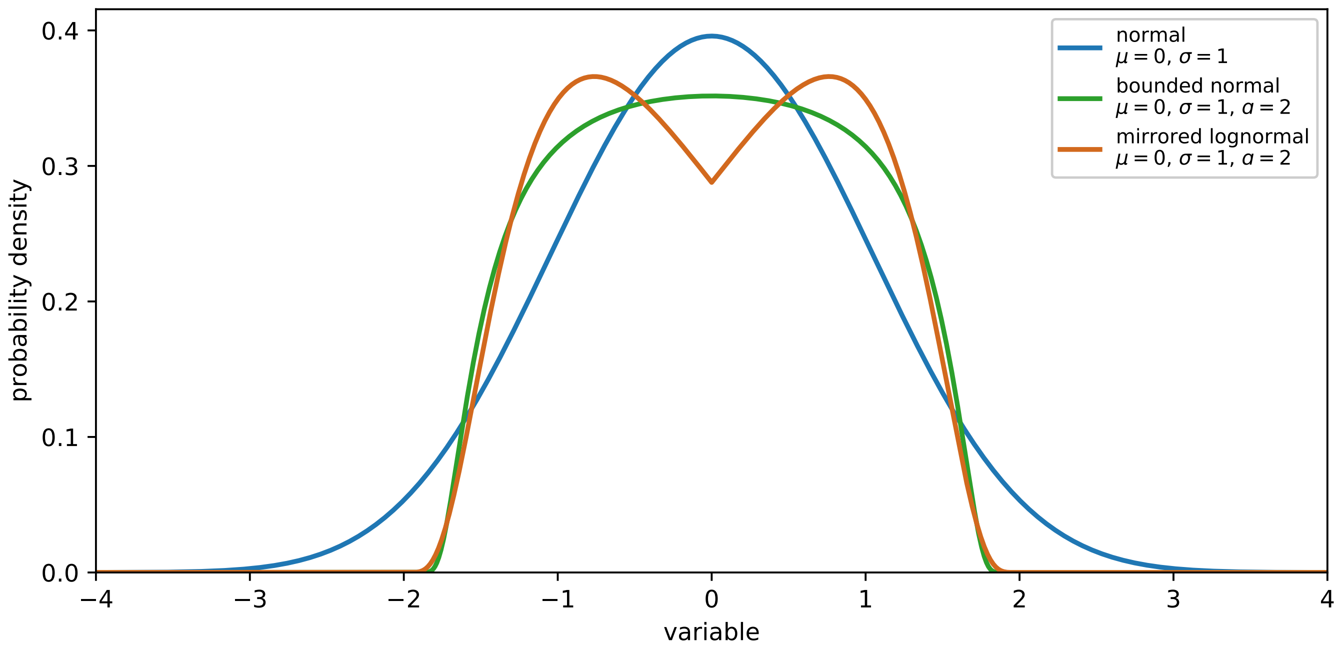

Figure 1. Probability density functions for a normal distribution, a bounded normal distribution and a mirrored lognormal distribution.

7.2. Upper/lower limits and finite intervals

Lastly, we should also address the case of a variable with an upper/lower limit or even a finite interval. Let's consider an interval (x_1, x_2), which may be finite or infinite (and which can represent an upper/lower limit). If it is finite, we choose a uniform distribution between x_1 and x_2 as the corresponding PDF, with finite thresholds for 0, which we set to $\pm\,10^{-90}$. But if it is infinite, we choose a symmetric loguniform distribution with finite thresholds for 0 and $\pm\,\infty$, which we set to $\pm\,10^{-90}$ and $\pm\,10^{90}$. For example, for an interval of (–100, ∞), we would build a sample $\{x_-\}$ from a uniform distribution between –90 and 2 and a sample $\{x_+\}$ from a uniform distribution between –90 and 90. Our final distribution would be the joining of the samples of $\{-10^{\{x-\}}\}$ and $\{10^{\{x+\}}\}$.

7.3. Summary

To sum up, if we have a set of variables x_i with central values μ_i and uncertainties σ_1, σ_2 we first build distributions $\{x_i\}$ using the mentioned PDFs. Then, we apply the function to the distributions, $f(\{x_i\})$, obtaining a new distribution. Finally, we use an algorithm to detect if the distribution corresponds to an interval (that can be an upper/lower limit) or a defined value with uncertainties, and derive the corresponding parameters.

8. Additional useful functions

Besides the functions explained throughout this document, the RichValues libraries uses some additional functions to work. This section contains explanations of most of them, as they may be of interest.

8.1. Rounding numbers

The following functions are used for displaying the rich values with the correct number of significant figures, rounding the numbers accordingly to the type of rich value and, if existent, its uncertainties.

Function `round_sf`

If rounds the given number to the given number of significant figures. The arguments are:

- **`x`**. Input number.
- **`n`**. Number of significant figures. The default is `1`.
- **`min_exp`**. Minimum exponent to display the number in scientific notation. The default is `4`.
- **`lim_for_extra_sf`**. If the significand/mantissa of the number is lower than this limit value, the number will be displayed with an additional significant figure. By default it is 2.5.

Function `round_sf_unc`

If rounds the given value and uncertainty to the given number of significant figures. The arguments are:

- **`x`**. Input value.
- **`dx`**. Uncertainty of the input value.
- **`n`**. Number of significant figures. The default is `1`.
- **`min_exp`**. Minimum exponent to display the numbers in scientific notation. The default is `4`.
- **`lim_for_extra_sf`**. If the significand/mantissa of the uncertainty is lower than this limit value, the numbers will be displayed with an additional significant figure. By default it is 2.5.

Function `round_sf_uncs`

If rounds the given value and uncertainties to the given number of significant figures. The arguments are:

- **`x`**. Input value.
- **`dx`**. List containing the lower and upper uncertainties of the input value.
- **`n`**. Number of significant figures. The default is `1`.
- **`min_exp`**. Minimum exponent to display the numbers in scientific notation. The default is `4`.
- **`lim_for_extra_sf`**. If the significand/mantissa of the lower uncertainty is lower than this limit value, the number will be displayed with an additional significant figure. By default it is `2.5`.

8.2. Creating distributions

The following functions are used for creating the distributions associated with the rich values.

Function `bounded_gaussian`

It applies the bounded gaussian function defined in section 7.1.2, which is the probability density function (PDF) of a bounded gaussian distribution. The arguments are:

- **`x`**. Input array of values to apply the function.
- **`m`**. Median of the curve. By default it is `0`.
- **`s`**. Width of the curve (similar to the standard deviation). By default it is `1`.
- **`a`**. Amplitude of the curve (distance from the median to the domain edges). By default it is `10`.
- **`norm`**. Logical variable that determines if the resulting curve is normalized. By default it is `False`.

Function `mirrored_loggaussian`

It applies the mirrored log-gaussian function defined in section 7.1.3, which is the probability density function (PDF) of a mirrored lognormal distribution. The arguments are:

- **`x`**. Input array of values to apply the function.
- **`m`**. Median of the curve. By default it is `0`.
- **`s`**. Width of the curve (similar to the standard deviation). By default it is `1`.
- **`a`**. Amplitude of the curve (distance from the median to the domain edges). By default it is `10`.
- **`norm`**. Logical variable that determines if the resulting curve is normalized. By default it is `False`.

Function `sample_from_pdf`

It draws a sample from the distribution specified with the given probability density function

(PDF). The arguments are:

- **pdf**. Input PDF of the distribution.
- **size**. Size of the sample.
- **low**. Minimum of the input values for the PDF.
- **high**. Maximum of the input values for the PDF.

Additionally, you can use any of the keyword arguments of the input PDF.[21]

loguniform_distribution

It draws a sample from a lognormal distribution, with finite thresholds for 0 and $\pm\infty$.

- **low**. Minimum of the input values for the PDF.
- **high**. Maximum of the input values for the PDF.
- **size**. Size of the sample.
- **zero_log**. Decimal logarithm of the minimum value in absolute value that can be returned.
- **infinity_log**. Decimal logarithm of the maximum absolute in absolute value that can be returned.

distribution_with_rich_values

It creates a distribution resulting from applying the given function to the given rich values, which will be represented by their corresponding distributions.

- **function**. Function to be applied to the input rich values.
- **args**. Input rich value arguments.
- **len_args_samples**. Size of the samples of the arguments. By default, it is the number of arguments times the mean of the sample size of each argument.

8.3. Evaluating distributions

The following functions are used for evaluating distributions in order to interpret them as rich values.

Function center_and_uncs

It returns the central value and uncertainties of the given distribution.

- **distr**. Input distribution.
- **function**. Function used to define the central value. By default, it is the median, calculated with np.median.
- **interval**. Confidence interval, in percent, used to define the uncertainties. The default is the 1 σ confidence interval (~ 68.27 %), that is, 68.27.
- **fraction**. Fraction of the input distribution (centered in the calculated central value)

21. Keyword arguments are arguments that have a default value. For example, in bounded_gaussian and symmetric_loggaussian, the arguments m, s, and a are keyword arguments.

used to compute the uncertainties. By default, it is `1.0`.

Function `evaluate_distr`

It interprets the given distribution as a rich value.

- **`distr`**. Input distribution
- **`domain`**. Domain of the result, in case it is already known. By default it is all the real numbers, that is, `[-np.inf, np.inf]`.
- **`zero_log`**. Decimal logarithm of the threshold in absolute value for 0, used to calculate the resulting domain and the range of the resulting distribution. The default is `-90.0`.
- **`infinity_log`**. Decimal logarithm of the threshold in absolute value for $\pm\infty$, used to calculate the resulting domain and the range of the resulting distribution. The default is `90.0`.

Citation of the library

If you use the library for your work, we would be very grateful if you cite this library. You can put the link to the GitHub repository, where this user guide can also be downloaded:

https://github.com/andresmegias/richvalues .

Useful links

The following links may be of interest:

- **Python.**
  https://www.python.org/
- **Python's standard library.**
  https://docs.python.org/3/library/
- **NumPy.**
  https://numpy.org/
- **Pandas.**
  https://pandas.pydata.org/

Credits

This software has been developed at the Center for Astrobiology (*Centro de Astrobiología*, CAB), in Madrid (Spain), within the subgroup of Astrochemistry and Chemical Complexity (Group of Interstellar and Circumstellar Medium, Department of Astrophysics).

Coding and testing
Andrés Megías Toledano

Testing
Álvaro López Gallifa

Additional ideas
Jacobo Aguirre Araujo, Eva Herrero Cisneros

License