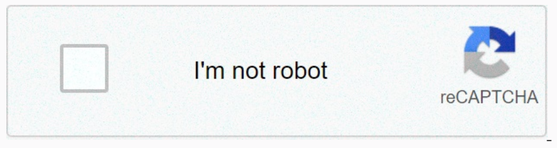

Continue

Uncertainties python logarithm

Spreading uncertainty: Lazy and Absurd Way2013 3. Well-researched problem, right? And really, there is a package of uncertainties in Python that is quite sophisticated and seems to be the gold standard for such things. Because uncertainties are extremely reasonable, they represent uncertain variables with medium and standard deviations, which analytically spread errors. It does it quite robustly, calculating the necessary derivatives magically, but analytical dissemination still fundamentally works by ignoring nonlinear terms, meaning in words of uncertainty documentation that it is therefore important that uncertainties be small. As far as I can tell, uncertainty is analytical spread as much as anything out there, but frankly, if your method can't handle a lot of uncertainty, it's pretty useless for astronomy. Well, if the analytical spread of bugs doesn't work, I think we need to do it empirically. So I wrote a little Python module. To represent 5 and 3 ± do not create a variable that stores mean=5 and stddev=3 — I create a field that stores 1024 samples taken from the normal distribution. Yes. To calculate, I used Numpy's vectorized operations. When I report the result, I look at the 16th, 50th and 50th. Ridiculous? Yes. Inefficient? Oh, yes. Effective? Also, yes, in many cases. For example: the uncertainty package does not support asymmetric error line or upper limits. My understanding is that they could be implemented, but poorly break the assumptions of the spread of analytical errors - an asymmetric error bar by definition can not be represented by a simple medium and standard deviation, and the measurement of the upper limit by definition has great uncertainty compared to its best value. But I can do math with these values simply by drawing my 1024 samples from the correct distribution – beming normal or uniform between zero and limit. I can combine perfectly known values, standard (i.e. normally distributed) uncertain values, upper limits and anything else and everything just works. (It might be hard to define uncertainty on the complex function of a mixture of all of these, but that's because it's really ill-defined – analytical dissemination is just misleading you!) Another example: uncertainties spend a lot of effort tracking correlations, so if x = 5 ± 3, then x - x = 0 exactly, not 0 ± 4.2. My access gets it for free. I've found that approaching uncertainty in this way helps clarify your thinking too. Are you afraid: are 1024 samples large enough? Well, did you really measure 5 ± 3 by taking 1,024 samples? Probably not. As Chief Hogg points out, the uncertainties on your insecurities are great. I'm pretty sure that only under extreme circumstances would the number of samples actually limit your ability to understand your insecurities. what if you are trying to calculate log(x) for x = 9 ± 3? With 1024 samples, you'll quite likely end up trying to take logarithm negative number. Well, that tells you something. In many such cases, x is something like luminosity, and while you may not be sure that it's much larger than zero, I can guarantee you that it's not actually less than zero. The assumption that x is drawn from the normal distribution fails. Now, living in the real world, you want to try to handle these corner cases, but if they happen permanently, you're told that splitting the premise is a significant enough effect that you have to figure out what to do about it. Now, of course, this approach has some serious drawbacks. But it was super easy to implement and just worked remarkably well. That's big business. I'm having a hard time with the pythons uncertainty package. I have to evaluate experimental data with Python, which I have been doing for a while but have never encountered the following problem: >>>from the uncertainty of importing ufloat >>>from the uncertainty of importing unumpy as unp >>>u = ufloat(5, 1) >>>l = unp.log(u) >>>print(l) 1.61+/-0.2 Everything seems to be fine, all right? But here comes the strange part: >>>print(type(l)) <type 'numpy.ndarray'=> Which is a huge problem because that's how I met him: >>>print(l.n) AttributeError: the 'numpy.ndarray' object doesn't have the attribute 'n' Right now in my work I desperately need denominations and standard devivations separately for linear regression. What's really weird about it and makes me think it actually is a mistake is the fact that printing variables actually works as intended, but python thinks its type is a field when its actually supposed to be ufloat. Any ideas or tips for an easy solution? Do you think it's a mistake, or did we miss something, and it's actually my fault? To prevent anyone from asking why I'm doing such a simple calculation: This is just an example of course. In my actual work, I have many much more complex values stored in fields. Edit1: Edit2: Ok here's the code I'm actually having problems with, the above was just supposed to illustrate the problem. d, t, n = loadtxt('mess_blei_gamma.txt', unpack=True) fh = open('table_blei.txt', 'w') nn = [] ln = [] for i in the range(0, len(d)): nn.append(norm(n[i], t[i], n0)) ln.append(unp.log(nn[i])) fh.write(tex(str(d[i])+ & +str[t])+ & +str(n[i])+ & +str(nn[i])+ & +str(ln[i]) #works and as it should, the table is perfectly fine fh.close() printing(unp.nominal_values(nn)) #works in fine print (unp.nominal_values(ln)) #error The Uncertainty Package is a free platform-specific program that transparently processes calculations with uncertainty numbers (for example, 3.14±0.01). It can also bring derivatives of any expression. The uncertainty package takes pain and complexity from uncertainty calculations. Propagation of errors cannot be feared </type>Calculations of results with uncertainties or derivatives can be performed either in an interactive session (as with a calculator) or in programs written in the Python programming language. Existing calculation code can be run with little or no change. Regardless of the complexity of the calculation, this package returns its result with uncertainty as predicted by linear error propagation theory. It automatically calculates derivatives and uses them to calculate uncertainties. Almost all uncertainty calculations are made analytically. Correlations between variables are automatically processed, which distinguishes this module from many existing error propagation codes. You may want to check the following related calculations of Python uncertainty packages to see if they better suit your needs: soerp (higher order of approximations) and mcerp (Monte-Carlo approach). Calculations involving numbers with uncertainties can be performed without knowing anything about the Python programming language. After installing this package and invoking the Python interpreter, you can transparently perform calculations with automatic error propagation (i.e. through the usual syntax of mathematical formulas): >>> import ufloat >>> from uncertainties.umath import * # sin(), etc. >>> x = ufloat(1, 0.1) # x = 1+/-0.1 >>> print 2 *x 2.00+/-0.20 >>> sin(2*x) # Python shell printing is optional0.9092974 268256817+/-0.08322936730942848 The current calculation code for common numbers can therefore run with numbers with uncertainties without or small adjustments. Another advantage of this package is the correct processing of correlations. For example, the following quantity is exactly zero, although x has uncertainty: Many other propagation errors return codes to return an incorrect value of 0±0.1414... because they incorrectly assume that both subtracted quantities are independent of random variables. Fields of numbers with uncertainties are also transparently processed. Derivatives are similarly very easy to obtain: >>> (2* x + 1000).derivatives[x] 2.0 They are calculated by a quick method. The user guide details many of the features of this package. The Uncertainties in Fields section describes how to create and use fields of numbers with uncertainties. The technical manual provides advanced technical details. There is also a pdf version of the documentation. Additional information is available through the pydoc command, which provides access to many of the documentation strings contained in the code. The installation commands below should run in the DOS or Unix command environment (not python). In Windows (version 7 and earlier) you can get the shell command by running cmd.exe (via Run ... menu from the Start menu). Under Unix (Linux, Mac OS X,...), the Unix shell is available when opening the terminal (in Mac OS X, the terminal program is located in the Utilities folder, which can be accessed via the Go menu in the Finder). One of the The installation or upgrade procedures below may work on your system if you have a Python package installer or are using certain Linux distributions. Within Unix, you may need to prefix the sudo commands below for Setup to have sufficient access rights to the system. If you have pip, you can try to install the latest version with pip install --upgrade uncertainty If you have an installation backup, you can try to automatically install or upgrade this package with easy_install --upgrade uncertainty Package uncertainty is also available for Windows through Python(x,y) distribution. It can also be included in Christoph Gohlke's basic distribution of Python scientific packages. Mac OS X users who use macports package manager can install uncertainties with sudo py**-uncertainties port installation and upgrade it by upgrading the sudo py**-uncertainties port, where ** represents the desired version of Python (27, 33, etc.). The uncertainty package is also available through the following Linux distributions and software platforms: Ubuntu, Fedora, openSUSE, Debian and Maemo. Alternatively, you can simply download the package archive from the Python Package Index (PyPI) and expand it. The package can then be installed by going to the expanded directory (uncertainty-...) and running the provided setup.py program (where the default python interpreter must generally be replaced by the Python version for which the package should be installed: python3, python3.3, etc.). To install with Python 2.6+ in the Python user library (without additional access rights) setup.py python installation --user To install in your own my_directory directory: python setup.py install --install-lib my_directory If additional access rights are needed (Unix): sudo python setup.py installation You can also simply move the uncertainty directory-py* that best matches your Python version to a location from which Python can import (directory , in which scripts are run using uncertainties, etc.); the selected uncertainty directory py* should then be renamed uncertainties. Python 3 users should then run 2to3 -w . from this directory so that the code automatically adapts to Python 3. The latest, bleeding, but work code and documentation source are available on GitHub. The uncertainty package is written in pure Python and has no external dependency (numpy package is optional). Contains about 7,000 lines of code. 75% of these lines are documentation strings and notes. The remaining 25% is divided between unit tests (15% of the total) and the actual calculation code (10% of the total). uncertainty is therefore a lightweight, portable package with rich documentation and tests. Some incompatible changes were introduced in version 2 of Uncertainty (see version history). While the version 2 line will support version 1 syntax for some time, we recommend that you update your existing programs as soon as possible. This through the automatic update program delivery system. The automatic updater acts as a Python 2to3 updater. It can be run (in unix or DOS shell) with: python -m uncertainty.1to2 For example, an update of one Python program can be done with python -m uncertainty.1to2 -w example.py All Python programs contained in the Programs directory (including nested subcans) can be automatically updated with python -m uncertainty.1to2 -w Backup programs are automatically created if -n option is listed. Some manual edits may be required after you run the update program (incorrectly edited lines, untouched by outdated syntax). While the updater creates backup copies by default, it is generally useful to first create a backup of the modified directory or alternatively use some version control system. Checking for changes using the file comparison tool can also be useful. What others say¶ Planned future developments include (starting with the most in-need): processing complex numbers with uncertainties; increased NumPy support: Fourier transformation with uncertainties, automatic wrapping of functions that accept or create arrays, standard deviation of fields, more convenient matrix formation, new linear algebra methods (custom value and QR decomposition, determinant,...), input of fields with uncertainties as strings (as in NumPy),...; Support for JSON; adding real and imag attributes, for increased compatibility with existing code (Python numbers have these attributes); Add new features from a math module; installation of routines that conveniently process data with uncertainties; Re-correlate a function that puts a correlation back between data that has been stored in separate files; support for multi-faceted numbers with uncertainties. Call for contributions: I have several requests for complex numbers with uncertainties, support for fourier transformation, and automatic wrapping of functions that receive or create fields. Please contact me if you are interested in contributing. Patches are welcome. They must have a high standard of readability and quality to be accepted (otherwise it is always possible to create a new Python package by branching this package, and I would still be happy to help you with the effort). Please encourage further development of this program by donating $10 or more through PayPal (PayPal account required). I like modern board games so it will go towards making my friends and I some special game time! If you use this package for publication (magazine, web, etc.), provide as much information as possible from the following information: Uncertainties: Python Uncertainty Calculation Package, Eric O. LEBIGOT, . Adding a version number is optional. The author wants to thank all the people who have made generous donations: they help keep this project alive by providing positive feedback. I really appreciate it obtained key technical information from Arnaud Delobell, Pierre Cladé and Sebastian Walter. Patches Pierre Cladé, Tim Head, José Sabater Montes, Martijn Pieters, Ram Rachum, Christoph Deil, Gabi Davar and Roman Yurchak are gratefully recognized. I would also like to thank users who have contributed feedback and suggestions, which greatly helped improve this program: Joaquin Abian, Jason Moore, Martin Lutz, Víctor Terrón, Matt Newville, Matthew Peel, Don Peterson, Mika Pflueger, Albert Puig, Abraham Lee, Arian Sanusi, Martin Laloux, Jonathan Whitmore, Federico Vaggi, Marco A. Ferra, Hernan Grecco, David Zwicker, James Hester, Andrew Nelson and many others. I am also grateful to Gabi Davar and Pierre Raybaut for its inclusion in Python (x, y) to Christoph Gohlke for his basic distribution of Python scientific packages for Windows, and anaconda, macOS and Linux distribution administrators of this package (Jonathan Stickel, David Paleino, Federico Ceratto, Roberto Colistete Jr, Filipe Pires Alvarenga Fernandes, and Felix Yan). The software is issued under a dual license; You can choose one of the following options: Revised BSD license (© 2010-2016, Eric O. LEBIGOT [EOL]). Any other license, if obtained from the creator of this package. Package.