ExGUtils: Manual

D. Gamermann*[a]

[a]Department of Physics, Universidade Federal do Rio Grande do Sul (UFRGS) - Instituto de Fsica , Av. Bento Gonalves 9500 - Caixa Postal 15051 - CEP 91501-970 - Porto Alegre, RS, Brasil.

March 6, 2018

Abstract

This is a description of the functions in the modules of the ExGUtils package.

Keywords: exgaussian, statistical analysis, psychometrics, reaction times

1 Introduction

This document contains the description of the functions found in the modules of the package ExGUtils and presents examples of uses for them. The package has been written mainly to aide statistical analysis of data involving the ex-Gaussian function, as is usually the case of reaction times in psychometric experiments, for example.

Some examples deserve graphical representations (plots) of the results. The package ExGUtils does not contain any graphical utility, so for this purpose we recommend the use of the gnuplot software which can be used inside the python interpreter via the Gnuplot package, whose functions we import with the commands found in Listings 1:

Listing 1: Commands used to import graphical objects from GNUplot.

```
from Gnuplot import Gnuplot as gplot
from Gnuplot import Data as gdata
from Gnuplot import Func as gfunc
g = gplot(persist=1)
```

Note that the commands in Listings 1 should only be used if GNUplot and the Gnuplot package for python are installed in the computer. The reader who

*gamermann@gmail.com

wishes, should adapt the plotting commands in order to visualize the graphs with other plotting tools.

This manual refers to the version 3.0 of the ExGUtils package. Version 3.0 is very different from Versions 1.0 and 2.0.

Some users of version 2.0 reported problems when compiling the C functions in a windows system. Therefore, in this new version of the package, when installing it, the user can opt to either compile or not the C functions. If he chooses not to compile them, only the `pyexg` module will be installed. This module has all functions and algorithms which in version 2.0 were programmed in C now programmed purely in python. If the user does compile (successfully) the C functions, both modules, `pyexg` and `uts`, will be installed. They both have basically the same functions (with a few exceptions which will be discussed later), but while the functions in one module are programmed in C, in the other module they are purely programmed in python.

Only the module `pyexg` has dependencies. Some of its functions depend on the `numpy`, `scipy`, `math` and `random` packages.

Another difference between the present version and previous ones, is the nomenclature of some functions. Due to some feedback we received, we tried to standardize the function names such that they have now similar `scipy` and `numpy` nomenclature.

Many functions in this package deal with (possibly heavy) numerical calculations. Numerical calculations can be involving and become unstable. In order to properly perform (and understand the results) of numerical calculations it is very advisable that the user understands the calculation and controls some parameters of it. In this manual we explain the most involving calculations and the numerical parameters that may be controlled (precision, for example) in order to obtain meaningful results. All numerical parameters in the functions do have default values that, in principle, should give a reasonable results, but it is possible that particular functions or datasets may fall in unstable parameter regions and, in such cases, it may be useful to control some numerical parameters.

2 Modules Functions

The two modules that are installed with the `ExGUtils` package are:

- `pyexg` → All functions in this module are programmed in python language. In order to use this module, the packages `numpy`, `scipy`, `math` and `random` must previously be installed by the user (`math` and `random` usually come by default with python).
- `uts` → The functions in this module are programmed in C and must be compiled when installing the module (the user should answer `Y` when asked "`Compile C [Y/n]?`". This module has no dependency with other python packages.

Most functions have the same names and receive the same arguments in both modules. The difference will be speed (our tests indicate that the C functions run at least 50 times faster than the python ones for the most complex calculations), numerical precision and stability (again C is better than python).

The following functions can be imported from both modules:

- Random Number Generators: The basic algorith for generating homogeneous random numbers in the `uts` module has been taken from Numerical Recipes in C and it is the basic random number generator in order to construct random numbers with different distributions in this module. In the module `pyexg` all random number generators come from the `random` package, they are simply imported with different names in order to assure compatibility and consistency.
  - `drand` → Generates a random number with homogeneous distribution between 0 and 1.
  - `exp_rvs` → Generates a random number with exponential distribution.
  - `gauss_rvs` → Generates a random number with gaussian distribution.
  - `exg_rvs` → Generates a random number with ex-gaussian distribution.
- Numerical and Statistical Analysis
  - `histogram` → Produces an histogram from a numerical dataset.
  - `stats` → Calculates the statistics of a dataset.
  - `stats_his` → Calculates the statistics of the data in an histogram.
  - `correlation` → Calculates the linear correlation coefficient for two datasets.
  - `minsquare` → Adjusts a polynomial using the minimum square method to a list of points.
- Specific to Ex-gaussian function
  - `gauss_pdf` → Evaluates the gaussian distribution at a given point.
  - `gauss_cdf` → Evaluates the gaussian cumulative distribution at a given point.
  - `exg_pdf` → Evaluates the ex-gaussian distribution at a given point.
  - `exg_lamb_pdf` → Same as `exg_pdf` but uses the ex-gaussian parametrized only in terms of its asymmetry.
  - `exg_cdf` → Evaluates analytically the cumulative distribution for the ex-gaussian at a given point.

- `exg_lamb_cdf` → Same as `exg_cdf` but uses the ex-gaussian parametrized only in terms of its asymmetry.
- `exg_ppf` → Evaluates the point at which the ex-gaussian distribution leaves a given left tail. Is the inverse of `exg_cdf`.
- `exg_lamb_ppf` → Same as `exg_ppf` but uses the ex-gaussian parametrized only in terms of its asymmetry.
- `stats_to_pars` → Evaluates the parameters μ, σ and τ from a distribution statistics.
- `pars_to_stats` → Evaluates the statistics correspondent to the parameters μ, σ and τ.
- `exgLKHD` → Evaluates the likelihood and its gradient in parameter space for a dataset given the parameters μ, σ and τ.
- `maxLKHD` → Evaluate the parameters μ, σ and τ that maximize the likelihood for a dataset.
- `exgSQR` → Evaluates the sum of squares and its gradient in parameter space for the histogram built from a dataset given the parameters μ, σ and τ.
- `minSQR` → Evaluate the parameters μ, σ and τ that minimize the sum of squares for a dataset.

The following functions are present only in the `uts` module:

- `int_points_gauss` → Generates an gaussian partition of points in order to perform an integral.
- `intsum` → Calculates the integral for a function calculated for every point in a gaussian partition.

The following functions are present only in the `pyexg` module:

- `zero` → Finds the zero of a function.
- `ANOVA` → Performs the ANOVA test (ANalysis Of VAriance).
- `integral` → Integrates a function between two points.

3 Some Examples and Uses

The calls to the random number generators are straight forward. The probability density for each distribution is given in equations (1-4) for the homogeneous, exponential, gaussian and ex-gaussian distributions respectively.

Table 1: Statistics for the probability distributions.

Distribution	M	S	t
Homogeneous	$\frac{1}{2}$	$\sqrt{\frac{1}{12}}$	0
Exponential	τ	τ	2
Gaussian	μ	σ	0
Ex-Gaussian	$\mu+\tau$	$\sqrt{\sigma^2+\tau^2}$	$\frac{2\tau^3}{(\sigma^2+\tau^2)^{\frac{3}{2}}}$

$$h(x) = \begin{cases} 0 & \text{, if } x \leq 0 \text{ or } x > 1 \\ 1 & \text{, otherwise} \end{cases}, \tag{1}$$

$$e(x/\tau) = \frac{1}{\tau} e^{-\frac{x}{\tau}}, \tag{2}$$

$$g(x/\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \tag{3}$$

$$y(x/\mu,\sigma,\tau) = \frac{1}{2\tau} e^{\frac{1}{2\tau}\left(2\mu+\frac{\sigma^2}{\tau}-2x\right)} \operatorname{erfc}\left(\frac{\mu+\frac{\sigma^2}{\tau}-x}{\sqrt{2}\sigma}\right). \tag{4}$$

The average M, standard deviation S and skewness t for a given probability distribution $f(x)$ are calculated, as:

$$M = \int_{-\infty}^{\infty} x f(x) \mathbf{d}x, \tag{5}$$

$$S^2 = \int_{-\infty}^{\infty} (x-M)^2 f(x) \mathbf{d}x, \tag{6}$$

$$t = \int_{-\infty}^{\infty} \left(\frac{x-M}{S}\right)^3 f(x) \mathbf{d}x. \tag{7}$$

The result for the statistics for each one of the distributions in terms of its parameters can be found in table 1.

In Listings 2 we use the random number generators in order to generate 1000000 random numbers with each probability distribution. Then we use the function `stats` to evaluate the statistics of each set generated and compare it with the expected values of table 1. In a computer with a pentium i7 processor the commands in this listing took around 1.2 seconds to run (with the C compiled functions). Note that four sets with a million random numbers in each are generated and statistics for each set is evaluated.

Listing 2: Generation of set of random numbers with different distributions and evaluation of their statistics. Note that the results obtained are random, so in every execution of the commands, slightly different results are obtained.

```
from ExGUtils.uts import stats, drand, drand_exp, drand_gauss, drand_exg

N = 1000000
mu = 100.; sig = 50.; tau = 150.
li1 = [drand() for ii in xrange(N)]
li2 = [exp_rvs(tau) for ii in xrange(N)]
li3 = [gauss_rvs(mu, sig) for ii in xrange(N)]
li4 = [exg_rvs(mu, sig, tau) for ii in xrange(N)]
# After each result, the expected value in parenthesis
[M, S, t] = stats(li1, True)
print "Homogeneous: M=%4.4f (%4.4f)  S=%4.4f (%4.4f)   t=%4.4f (%4.4f)"% \
    (M, 0.5, S, (1./12)**.5, t, 0.)
[M, S, t] = stats(li2, True)
print "Exponential: M=%4.4f (%4.4f)  S=%4.4f (%4.4f)   t=%4.4f (%4.4f)"% \
    (M, tau, S, tau, t, 2.)
[M, S, t] = stats(li3, True)
print "Gaussian   : M=%4.4f (%4.4f)  S=%4.4f (%4.4f)   t=%4.4f (%4.4f)"% \
    (M, mu, S, sig, t, 0.)
[M, S, t] = stats(li4, True)
print "Ex-Gaussian: M=%4.4f (%4.4f)  S=%4.4f (%4.4f)   t=%4.4f (%4.4f)"% \
    (M, mu+tau, S, (sig**2+tau**2)**.5, t, 2.*(tau**3)/((sig**2+tau**2)**(3./2.)))

# Results:
#Homogeneous: M=0.4997 (0.5000)  S=0.2887 (0.2887)   t=0.0022 (0.0000)
#Exponential: M=149.7682 (150.0000)  S=149.8710 (150.0000)   t=2.0007 (2.0000)
#Gaussian   : M=100.0317 (100.0000)  S=49.9567 (50.0000)   t=0.0015 (0.0000)
#Ex-Gaussian: M=249.9907 (250.0000)  S=158.0081 (158.1139)   t=1.6990 (1.7076)
```

In the example found in Listings 2 one can also see the use of function `stats`. This function must have at least one argument (a list of numbers). Its second argument is an optional boolean which is `False` by default. The function `stats` returns the statistics of the numbers contained in the list, if the optional argument is false, it returns only the average and standard deviation for the numbers in the list, if the second argument is true, it returns the skewness as well.

Another important function in this module for statistical analysis is the `histogram` function. This function has one mandatory argument which is a list of numbers (from which it will build the histogram) and a series of optional arguments in order to control the histogram parameters. One can control the interval under which the function constructs the histogram with the parameters `ini` and `fin` that by default are the smallest and the largest values in the list. The parameter `Nint` specifies the number of intervals in the histogram and is, by default, two times the square root of the number of elements in the list. The size of the intervals are, therefore, $\frac{fin-ini}{Nint}$. The function `histogram` returns two lists: the list of the class marks (a number representing each interval), and the counts in each interval. The parameter `dell` is used to control the class marks positions, its default number is 0.5 which indicates that each point in the class mark list is the middle point of the interval. For this parameter a number between 0 and 1 should be used, 0 indicating the beginning of the interval and 1 indicating its other extreme (the use of numbers outside the interval [0;1) will not result in error, but may return senseless results, for it would shift the class marks to outside their proper intervals. There are two parameters to control the kind of histogram: The parameter `accu` can have the values 0, 1 or -1. The value 0 is the default and it counts the points in each interval, the values 1 and -1 result in cumulative distributions, -1 for the left-tail and 1 for the right tail; Finally, if `accu`=0, one can choose three types of normalization with the

parameter `norm`. The default is `norm`=0 and returns (the second list of the returned values) the absolute number of counts in each interval (the sum of all elements in the returned list will be the number of points), `norm`=1 is the "integral" normalization, so that the total sum of the area of the histogram is equal to one (the sum of all elements in the returned list times the size of the intervals will be equal to one) and `norm`=-1 means frequency histogram, the number corresponding to each interval is the proportion of points in each interval (the sum of all elements in the returned list will be equal to one).

The function `stats_his` does the same as function `stats` but for data distributed as a histogram (two lists as input, the class marks and the counts). If the count list in the histogram is not the absolute number of counts in each histogram (different normalization, `norm=1` for example), one should also know the total number of counts and enter also parameters in the `stats_his` indicating the normalization and the value for the number of points, otherwise the results might have big errors associated to them. We show in Listing 3 example of the use of the `histogram` function in order to produce some plots, and the `stats` and `stats_his` functions to obtain statistics. Also, in this example, one can already see the use of the function `exg_pdf` which is quite straightforward. In figure 1 one can see the plot generated by the commands.

Listing 3: Use of the functions `histogram`, `stats` and `stats_his`.

```
from ExGUtils.uts import stats, stats_his, histogram, exg_rvs, exg_pdf

N = 10000
mu = 100.; sig = 50.; tau = 150.
li4 = [exg_rvs(mu, sig, tau) for ii in xrange(N)]
[x, y] = histogram(li4, norm=1)
y2 = [exg_pdf(xi, mu, sig, tau) for xi in x]
d1 = gdata(x, y2, with_="lines lw 3 lc 1", title="exgaussian")
d2 = gdata(x, y, with_="boxes lc 3", title="histogram")
g.plot(d1, d2)

M, S, t = stats(li4, 1)
print "Li4 stats  :  M = %f, S = %f, t = %f"%(M, S, t)
[x, y] = histogram(li4, norm=0)
M1, S1, t1 = stats_his(x, y, assymetry=True)
print "For norm=0 :  M = %f, S = %f, t = %f      sum(yi) = %f"%(M1, S1, t1, sum(y))
[x, y] = histogram(li4, norm=-1)
M1, S1, t1 = stats_his(x, y, assymetry=1, norm=-1, N=N)
print "For norm=-1:  M = %f, S = %f, t = %f      sum(yi) = %f"%(M1, S1, t1, sum(y))
[x, y] = histogram(li4, norm=1)
M1, S1, t1 = stats_his(x, y, assymetry=1, norm=1, N=N)
print "For norm=1 :  M = %f, S = %f, t = %f      sum(yi*dx) = %f"%(M1, S1, t1, sum(y)*(x[1]-x[0]))

# output:
#Li4 stats  :  M = 251.001140, S = 161.469213, t = 1.822576
#For norm=0 :  M = 251.002816, S = 161.473770, t = 1.819510      sum(yi) = 10000.000000
#For norm=-1:  M = 251.002816, S = 161.473770, t = 1.819510      sum(yi) = 1.000000
#For norm=1 :  M = 251.002816, S = 161.473770, t = 1.819510      sum(yi*dx) = 1.000000
```

The function `correlation` evaluates, given two lists of numbers, X and Y, the linear correlation coefficient between them:

$$c = \frac{1}{N-1}\sum_{i=1}^{N}\frac{(x_i-\bar{x})(y_i-\bar{y})}{\sigma_x\sigma_y} \tag{8}$$

where c is the linear correlation coefficient, x_i and y_i are the elements in the X and Y lists, $\bar{x}$ and $\bar{y}$ are the averages of the lists and σ_x and σ_y are the standard deviations of the elements in the lists.

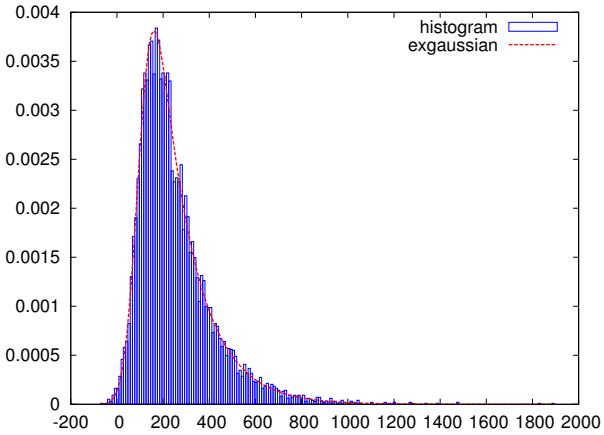

Figure 1: Histogram plotted along side the ex-gaussian function produced by the command line in Listing 3.

The function `minsquare` adjusts an polynomial to a dataset using the minimum square method. It returns a list containing the coefficients of the fitted polynomial, plus a number which is the minimized value for the sum of squares. In Listing 4 one uses the random number generator to generate noisy data, and then fits a third degree polynomial to the data using the `minsquare` function. The plot produced by the commands can be found in figure 2.

Listing 4: Produces noisy data and fits a polynomial to it.

```
from ExGUtils.uts import correlation, minsquare, drand

N = 25
dx = 8./N
x = [-3+ii*dx for ii in xrange(N)]
fu = lambda x: x**3-3*x**2-x+3
data = [fu(xi)+12*drand() for xi in x]
errs = [12*drand() for xi in x]
[coefs, chi2] = minsquare(x, data, errs, deg=3)
fit = lambda x, coefs: sum([coef*x**ii for ii, coef in enumerate(coefs)])
yfit = [fit(xi, coefs) for xi in x]
d = gdata(x, data, errs, with_="err", title="noisy_data")
dfit = gdata(x, yfit, with_="lines_lw_3", title="minsquare_fit")
g.plot(d, dfit)

print "fit_result:_y(x)_=_%3.3f_+_%3.3f_x_+_%3.3f_x^2_+_%3.3f_x^3"%tuple(coefs)
print "Linear_correlation_between_fit_and_noisy_data:_%f"%correlation(data, yfit)

# output:
#fit result: y(x) = 6.505 + -0.774 x + -2.602 x^2 + 0.947 x^3
#Linear correlation between fit and noisy data: 0.976793
```

The function `zero` receives two mandatory arguments and two keyword arguments. The mandatory arguments are the function (whose zero one wants to find) and an initial point to start the search. The function implements Newton's method and has two keyword arguments:

- `eps` → The precision within the zero must be found. In other words, the search stops when the value of the function at the point is smaller than eps.

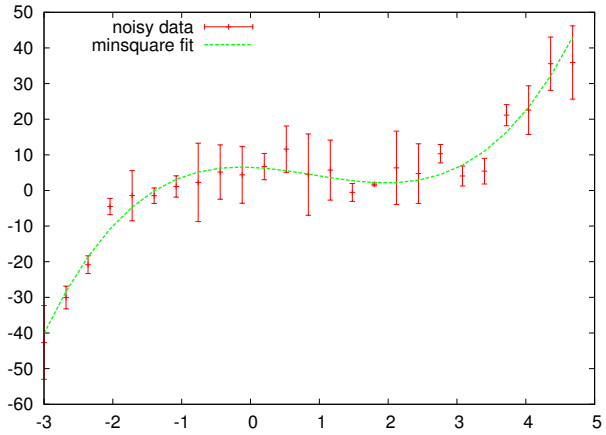

Figure 2: Polynomial fitted to noisy data..

- `delt` → Newton's method must evaluate the derivatives of the function and follow it in order to find the zero. The keyword argument delt should be a small number Δx that is used in order to evaluate numerically the derivative of the function: $\frac{df}{dx}(x) = \frac{f(x+\Delta x)-f(x)}{\Delta x}$.

Suppose we want to evaluate the value of x for which:

$$5\,(e^x - 1) - xe^x \quad = \quad 0, \tag{9}$$

(this equation appears when one tries to obtain Wien's Law from Planck's black-body raidiation spectrum).

In Listings 5 one can see the python commands used to call the zero function in order to perform this task.

Listing 5: Commands used to evaluate a zero at different precisions.

```
from ExGUtils.pyexg import zero
from math import exp

func = lambda x: 5.*(exp(x)-1.)-x*exp(x)

for ep in [1.e-4, 1.e-6, 1.e-15]:
    x = zero(func, 6., eps=ep)
    print "f( %5.18f ) = %5.18f     eps= %5.18f"%(x, func(x), ep)

# Result:
#
# f( 4.965114385178566181 ) = -0.000021223789417490     eps= 0.000100000000000000
# f( 4.965114233298355551 ) = -0.000000214967826651     eps= 0.000001000000000000
# f( 4.965114231744276907 ) = 0.000000000000000000     eps= 0.000000000000001000
```

4 Numerical Calculations

Here we make a few comments about the algorithms used and parameters that can be controlled in order to obtain meaningful results. Some functions may also take too long if some parameters are not properly chosen (it might cost to much to get a result with too much precision in some instances, for example).

The `_ppf` functions (percent point functions) use Newton's method to find the point where the cumulative distributions (`_cdf` functions) have a given value. This is an iterative method that starts at some given point. If the starting point is near an asymptote or a point where the derivative of the function is close to zero (the `_pdf` functions have small values), the method may become unstable. There is a keyword argument for the `_ppf` functions which is the starting point. By default it is zero (not the value zero, but, when it is zero, the starting point will be the average value of the distribution). If the function does not return a reasonable result (our tests indicate that it returns infinity and raises RuntimeWarnings in the python functions when the algorithm enters unstable regions) one might try to change this argument to a non-zero value (if one does wish to start the search at zero, we suggest to start at a very small value `1.e-10`, for example).

The functions `maxLKHD` and `minSQR` perform a maximum and minimum ascent/descent algorithm in order to maximize and minimize the likelihood and sum of squares, respectively for a dataset. The algorithm in both cases will follow a path given by the gradients in parameter space that are evaluated by the functions `exgLKHD` and `exgSQR`. Given an initial point, the algorithm evaluates the gradient at the point and advances following the gradient for maximizing or the opposite direction to minimize. The initial advanced step and point in the search can be controlled. If no initial point is given, the initial point is given by the parameters μ, σ and τ correspondent to the dataset statistics and the keyword argument parameter `lambda` controls the advance step. If not given, the initial advance in parameter space is 5. This step adapts itself while the algorithm runs in order to efficiently arrive to the maximum/minimum. If a positive value is given for the keyword argument parameter `lambda`, the initial advance is this number and if a negative number is entered, the adaptive step is the number times the module of the gradient.

5 Bias When Performing Fits

We have performed the following test in order to study possible bias in the three different fitting methods that can be directly used within the package:

We constructed 10000 datasets with 10000 ex-gaussian random numbers generated with the parameters $\mu = 30$, $\sigma = 20$ and $\tau = 20$.

Now, for each dataset, we evaluated the values of μ, σ and τ best fitting the data via the three methods: maximum likelihood (`maxLKHD` function), minimum squares (`minSQR` function) and moments (`stats` and `stats_to_pars` functions).

Then the average value of each parameter for each method and its standard

deviation was evaluated. The deviation from this average to the nominal value used to generate the random numbers should be compared with the standard deviation from the average divided by the square root of 10000 (the population of datasets). The results were:

```
Moments :
--------------
mub = 30.030567 +- 0.005933
sib = 20.011060 +- 0.004087
tab = 19.971779 +- 0.005933


maxLKHD :
--------------
mub = 30.015070 +- 0.004773
sib = 19.999438 +- 0.003025
tab = 19.987277 +- 0.004750


minSQR :
--------------
mub = 30.027971 +- 0.007022
sib = 20.001701 +- 0.004929
tab = 19.977125 +- 0.008647
```

The method that best performs appears to be the maximum likelihood method. In this method, the τ parameter is the one that presents the most statistically significant deviation from its nominal value. The code to generate the above results can be found in Listings 6.

Listing 6: Code to study bias in the different fitting methods.

```
mu = 30.
sig = 20.
tau = 20.
N1 = 10000
N2 = 10000
sts_mom = []
sts_lkhd = []
sts_sqr = []
for ii in xrange(N1):
    nums = [exg_rvs(mu, sig, tau) for jj in xrange(N2)]
    M, S, t = stats(nums, True)
    sts_mom.append(stats_to_pars(M, S, (.5*t)**(1./3)))
    sts_lkhd.append(maxLKHD(nums))
    [xi, yi] = histogram(nums, norm=1)
    sts_sqr.append(minSQR(xi, yi))

listas = [sts_mom, sts_lkhd, sts_sqr]
names = ["Moments", "maxLKHD", "minSQR"]
no = (1.*N1)**.5
for ii, name in enumerate(names):
    li = listas[ii]
    mus = [ele[0] for ele in li]
    sis = [ele[1] for ele in li]
    tas = [ele[2] for ele in li]
    # mu
    mm, sm = stats(mus)
    print "%s:\n--------------"%name
    print "mub = %f +- %f"%(mm, sm/no)
    # sig
    mm, sm = stats(sis)
    print "sib = %f +- %f"%(mm, sm/no)
```

```
# tau
mm, sm = stats(tas)
print "tab = %f + %f"%(mm, sm/no)
print
print
```

6 Bugs and Feedback

These functions have been thoroughly tested in a linux mint system using python 2.7. Some functions have already been tested in a windows system running python 2.7 via idle with no conflicts found until now.

Please email any detected bugs, suggestions or your feedback to the author.

7 License

ExGUtils is released under the GNU GENERAL PUBLIC LICENSE. See COPYING and README files for further information.