ExGUtils: Manual

D. Gamermann*[a]

[a]Instituto Universitario de Matemática Pura y Aplicada, Universidad Politécnica de Valencia, Camino de Vera 14, 46022 Valencia, Spain.

November 6, 2013

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 environment via the Gnuplot package, whose functions we import with the following commands:

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

The reader who wishes, should adapt the plotting commands in order to visualize the plots with other plotting tools.

*daniel.gamermann@gmail.com

2 The stats module

The stats module contain functions for quickly obtain the average and standard deviation of a distribution, to generate random numbers with probability densities given by the exponential function or the ex-Gaussian function and to perform an ANOVA test.

The functions contained in the module are:

- `stats` $\rightarrow$ This function receives a list as input and returns the average and standard deviation of the numbers in the list.
- `stats_int` $\rightarrow$ Same as the function `stats`, but for data given as histograms. The function receives two lists, the first is the list of the class marks for the intervals and the second is the absolute frequency in each interval.
- `rand_exp` $\rightarrow$ This function generates a random number with probability density given by the exponential function:

  $$h(x) = \frac{1}{\tau} e^{-\frac{x}{\tau}}. \tag{1}$$

  As input the function receives one number which is the value of the τ parameter for the distribution.
- `rand_exg` $\rightarrow$ This function generates a random number with probability density given by the ex-Gaussian function:

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

  As input it receives the parameters μ, σ and τ correspondent to the Gaussian and exponential components of the ex-Gaussian function.
- `histogram` $\rightarrow$ This function creates an histogram for a set of data. It should receive at least one input which is a list containing the data, for which it returns two lists, one with the class marks and the other with the absolute frequency in each interval. Several optional inputs can be given:
  - `ini`: lower bound of the first interval (by default its the lowest value found in the input list).
  - `fin`: higher bound of the last interval (by default its the highest value found in the input list).
  - `Nint`: Number of intervals (by default its two times the square root of the length of the input list).
  - `dell`: "offset" for the values in the class marks (by default its 0.5, this input should be a number between 0 and 1, otherwise the output may make no sense).

  - accu: This allows to choose cumulative distributions to the left (accu=1), to the right (accu=-1) or absolute frequencies (accu=0). By default accu=0.

- ANOVA $\rightarrow$ As input receives a list of lists (table) and performs an ANOVA test. For output it gives in this order: the value of the variable F_s (F of snedecor), number of degrees of freedom in between, number of degrees of freedom inside and the p-value for the test.

As an example, let's generate 1000 random numbers with exponential distribution, calculate its average and standard deviation, make histograms, recalculate average and standard deviation from the histogram and plot the distribution:

```
>>> from ExGUtils.stats import *
>>>
>>> N = 1000
>>> tau = 5.
>>> nums = [rand_exp(tau) for ii in xrange(N)]
>>> [xb, sb] = stats(nums)
>>>
>>> [x0, y0] = histogram(nums)
>>> [x1, y1] = histogram(nums, dell=1., accu=1)
>>> [x2, y2] = histogram(nums, dell=0., accu=-1)
>>>
>>> [xh, sh] = stats_int(x0, y0)
>>>
>>> plot1 = gdata(x0, y0, with_="boxes", title="absolute")
>>> plot2 = gdata(x1, y1, with_="lines lw 3 lc 2", title="left tail")
>>> plot3 = gdata(x2, y2, with_="lines lw 3 lc 3", title="right tail")
>>> g.plot(plot1, plot2, plot3)
```

These commands should produce a plot similar to the one found in figure 1. Moreover, the values for the distribution's average and standard deviation can be found in the variables xb and sb when calculated from the list nums and in the variables xh and sh when calculated from the distribution in intervals (histogram). Both, the average and standard deviation should be close to 5, which is the value of the τ parameter used for generating the numbers.

As another example, let's simulate three different realizations of a reaction time experiment and perform an ANOVA test to see if the differences are statistically significant. For this, we generate three lists of random numbers from ex-Gaussian probability densities, we put these list together in a table and call the ANOVA function:

```
>>> l1 = [rand_exg(560., 75., 75.) for ii in xrange(40)]
>>> l2 = [rand_exg(520., 50., 85.) for ii in xrange(30)]
>>> l3 = [rand_exg(590., 65., 65.) for ii in xrange(20)]
```

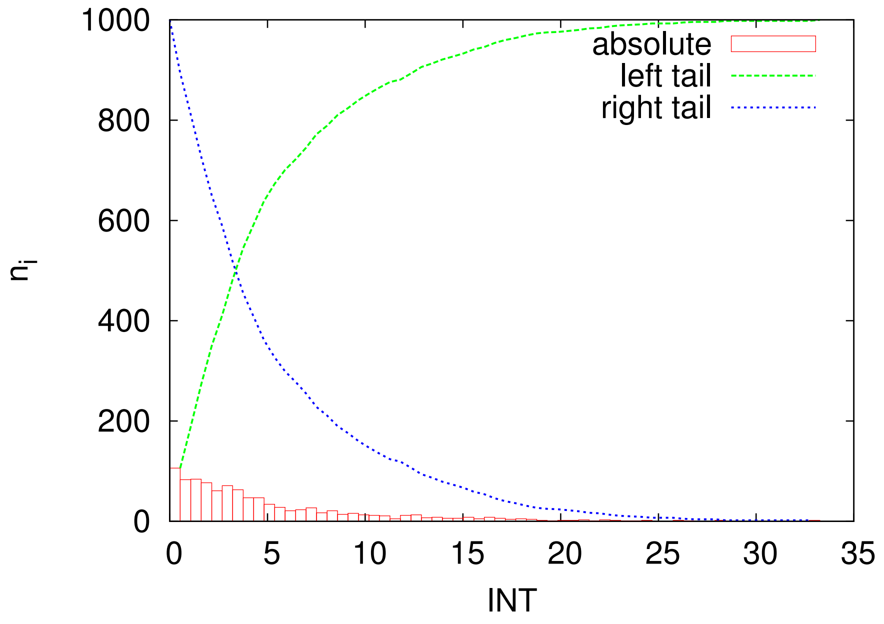

Figure 1: Distribution of random numbers from an exponential probability density with its left and right tails.

```
>>>
>>> tab = [l1, l2, l3]
>>> tup = ANOVA(tab)
>>> print " Fs = %f \n gb = %i \n gi = %i \n p-value = %f"%tuple(tup)
```

3 The nummath module

The module nummath contain four functions:

- `minsquare` → Given two lists one for the x and another for the y coordinates of a set of points, it calculates the coefficients for the best fit of a polynomial function of degree `G`.
- `integral` → Calculates the integral of the input function between two points. It divides the whole integration interval in `Nints` sub-intervals and uses the 20 points Gauss method to integrate each sub-interval.
- `fitter` → Fits any function to a set of points. This function uses the leastsq function from the scipy.optimize package, as an optional argument the function will also return the variable `ier`, which indicates success in the fitting when it is equal to 1, 2, 3 or 4.
- `zero` → finds the zero of a function. The optional argument `eps` indicates the precision and the optional argument `delt` the differential dx for calculating numerically the derivatives.

In the first example, let's generate points for a sinusoidal function with noise. Then lets fit a degree six polynomial to the points using the minsquare and the fitter routines and let's also fit a sinusoidal function using the fitter routine.

```
>>> from ExGUtils.nummath import *
>>> from math import pi
>>> from numpy import cos
>>> from random import gauss
>>>
>>> ini=0.;fin=10.;N=100;dx=(fin-ini)/N
>>> xx = [ini+dx*ii for ii in xrange(N)]
>>> k = 2.*pi/5.
>>> yy = [3*cos(k*ele+.3)+.2*gauss(0., 2.) for ele in xx]
>>>
>>> tofit1 = lambda p, x: p[0] + p[1]*x + p[2]*x**2 + p[3]*x**3 + p[4]*x**4 + \
... p[5]*x**5 + p[6]*x**6
>>> tofit2 = lambda p, x: p[0]*cos(p[1]*x+p[2]) + p[3]
>>>
>>> pms = minsquare(xx, yy, G=6)
>>> pf1 = fitter(tofit1, xx, yy, [1., 1., 1., 1., 1., 1., 1.])
>>> pf2 = fitter(tofit2, xx, yy, [1., 1., 1., 1.])
```

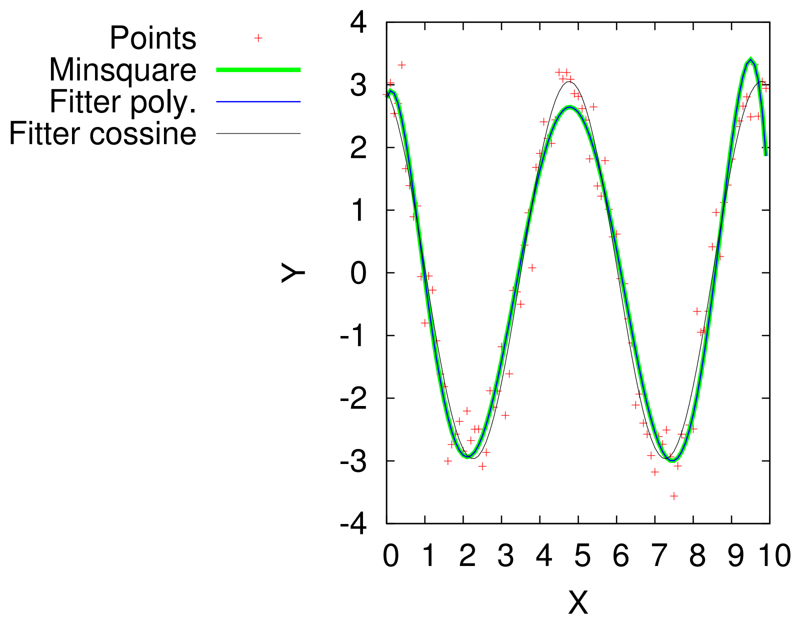

Figure 2: Set of points fitted to different functions.

```
>>>
>>> yms = [tofit1(pms, ele) for ele in xx]
>>> yf1 = [tofit1(pf1, ele) for ele in xx]
>>> yf2 = [tofit2(pf2, ele) for ele in xx]
>>>
>>> plot1 = gdata(xx, yy, with_="points", title="Points")
>>> plot2 = gdata(xx, yms, with_="lines lt 1 lw 7 lc 2", title="Minsquare")
>>> plot3 = gdata(xx, yf1, with_="lines lt 1 lw 2 lc 3", title="Fitter poly.")
>>> plot4 = gdata(xx, yf2, with_="lines lt 1 lw 1 lc 0", title="Fitter cossine")
>>> g.plot(plot1, plot2, plot3, plot4)
```

The resulting plot can be found in figure 2.

In our next example let's calculate $\sqrt{\pi}$ by integrating e^{-x^2} between minus and plus infinity:

```
>>> from math import exp, sqrt
>>> print sqrt(pi)
1.77245385091
>>> print integral(lambda x: exp(-x**2), -100, 100, 1000)
1.77245385091
```

In our last example for this module let's find the exact point where the functions $f(x) = \cos(x)$ and $f(x) = x$ intercept each other:

```
>>> x0 = zero(lambda x: cos(x)-x, 1., eps=1.e-15)
>>> print "x0 = %1.15f \ncos(x0)-x0 = %f"%(x0, cos(x0)-x0)
x0 = 0.739085133215161
cos(x0)-x0 = 0.000000
```

4 The exgauss module

The exgauss module contains functions specific to calculate properties for the ex-Gaussian probability density. An ex-Gaussian random variable is the result of the sum of an exponential random variable with a Gaussian random variable. In terms of its exponential and Gaussian component's parameters, the ex-Gaussian probability density is given by equation (2).

We should emphasize that the parameters μ and σ are not the average and standard deviation of the ex-Gaussian variable. The statistical parameters M, S and λ (average, standard deviation and normalized skewness) are related to the component's parameters via:

$$M = \mu + \tau \tag{3}$$

$$S = \sqrt{\sigma^2 + \tau^2} \tag{4}$$

$$\lambda = \sqrt[3]{\frac{t}{2}} = \frac{\tau}{\sqrt{\sigma^2 + \tau^2}} \tag{5}$$

$$\mu = M - S\lambda \tag{6}$$

$$\sigma = S\sqrt{1-\lambda^2} \tag{7}$$

$$\tau = S\lambda \tag{8}$$

The parameter t is the skewness of the distribution and we call λ the normalized skewness, its value lies between 0 and 1; 0 would mean a pure Gaussian and 1 a pure exponential. We should mention that for values of λ smaller than around 0.2 the ex-Gaussian distribution can hardly be differentiated from a Gaussian distribution, but the calculation of equation (2) becomes numerically unstable because of the multiplication of a huge number by a number smaller than the computer's machine precision, therefore it is not advisable to use the function in this module if $\lambda < 0.2$ (in these cases one can just approximate the ex-Gaussian with a Gaussian distribution).

We also use a "typified" ex-Gaussian (which means average 0 and standard deviation equal to 1). This typified ex-Gaussian depends only on one parameter, its skewness and the probability distribution for it is given by:

$$f_\lambda(z) = \frac{1}{2\lambda} e^{\frac{1}{2\lambda^2}(-2z\lambda - 3\lambda^2 + 1)} \operatorname{erfc}\left(\frac{-z + \frac{1}{\lambda} - 2\lambda}{\sqrt{2}\sqrt{1-\lambda^2}}\right). \tag{9}$$

The functions found in the module exgauss are:

- `exgauss` $\rightarrow$ The ex-Gaussian probability density in terms of its component's parameters for a point x.
- `exg_lamb` $\rightarrow$ The typified ex-Gaussian probability density in terms only of λ for a point $z = \frac{x-M}{S}$.

- `pars_to_stats` $\rightarrow$ Transformation from the component's parameters to the statistical parameters.
- `stats_to_pars` $\rightarrow$ Transformation from the statistical parameters to the component's parameters.
- `fit_exgauss` $\rightarrow$ Fits a distribution of numbers to an ex-Gaussian distribution. For performing the fit an histogram is constructed using a number of intervals given by the optional input `Nint`. It returns the values for the component's parameters μ, σ and τ and an overall multiplicative constant.
- `exgauss_lt` $\rightarrow$ Returns the left tail for the ex-Gaussian probability density.
- `exg_lamb_lt` $\rightarrow$ Returns the left tail for the typified ex-Gaussian probability density.
- `zalp_exgauss` $\rightarrow$ Returns the point x_α which leaves a right tail equal to α.
- `zalp_exg_lamb` $\rightarrow$ Returns the point z_α which leaves a right tail equal to α.

The following commands will produce plots for the ex-Gaussian distribution with average 0, standard deviation 1 for different values of λ:

```
>>> from ExGUtils.exgauss import *
>>>
>>> xx = [-4.+.001*ii for ii in xrange(10000)]
>>> y1 = [exg_lamb(ele, .9) for ele in xx]
>>> y2 = [exg_lamb(ele, .99) for ele in xx]
>>> y3 = [exg_lamb(ele, .999) for ele in xx]
>>>
>>> plot1 = gdata(xx,y1, with_="lines lw 8 lc 1 lt 1", title="lamb=0.9")
>>> plot2 = gdata(xx,y2, with_="lines lw 4 lc 2 lt 1", title="lamb=0.99")
>>> plot3 = gdata(xx,y3, with_="lines lw 2 lc 3 lt 1", title="lamb=0.999")
>>>
>>> g.plot(plot1, plot2, plot3)
```

The result can be seen in figure 3

We can also calculate the statistical parameters of the ex-Gaussian by integrating it:

```
>>> from ExGUtils.nummath import *
>>>
>>> M = integral(lambda x:x*exg_lamb(x, .99), -10., 40., Nints=1000)
>>> S = integral(lambda x:x**2*exg_lamb(x, .99), -10., 40., Nints=1000)
>>> t = integral(lambda x:x**3*exg_lamb(x, .99), -10., 40., Nints=1000)
>>> lamb = (t/2.)**(1./3)
>>> print "M = %f\nS = %f\nlamb = %f"%(M, S, lamb)
```

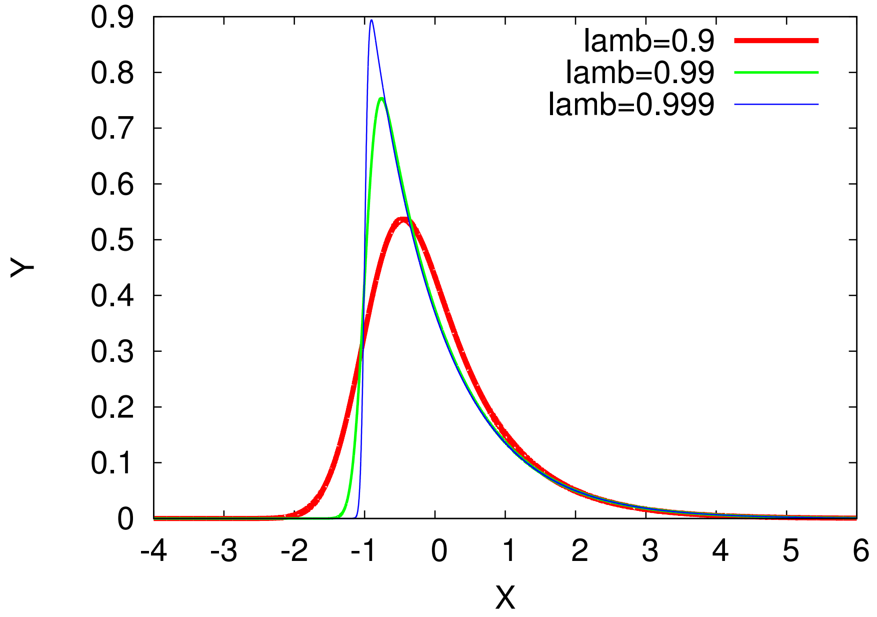

Figure 3: Typified ex-Gaussian for different values of λ.

```
M = 0.000000
S = 1.000000
lamb = 0.990000
>>> [mu, sig, tau] = stats_to_pars(M, S, lamb)
>>> print "mu = %f\nsig = %f\ntau = %f"%(mu, sig, tau)
mu = -0.990000
sig = 0.141067
tau = 0.990000
```

With the last two commands we obtained the component's parameters from the statistical we calculated.

The use of the `zalp` and `_lt` functions are straight forward. Suppose we determine, for a determined experiment the following values for the component's parameters: $\mu = 561$, $\sigma = 57$ and $\tau = 102$ and we are interested in determine the point for which 95% of the distribution is accumulated.

```
>>> from ExGUtils.exgauss import *
>>>
>>> mu = 561.; sig = 57.; tau=102;
>>> xa = zalp_exgauss(.05, mu, sig, tau, eps=1.e-12)
>>> [M, S, lamb] = pars_to_stats(mu, sig, tau)
>>> za = (xa - M)/S
>>> print exg_lamb_lt(za, lamb)
0.949999999994
```

In the last example let's simulate an experiment with 340 measurements of reaction times (with the same parameters as above for the ex-Gaussian) and fit the data with the `fit_exgauss` function.

```
>>> from ExGUtils.exgauss import *
>>> from ExGUtils.stats import *
>>>
>>> mu = 561.; sig = 57.; tau=102;
>>> nums = [rand_exg(mu, sig, tau) for ii in xrange(340)]
>>> [muf, sigf, tauf, AA] = fit_exgauss(nums)
>>> print "mu = %f\nsig = %f\ntau = %f"%(muf, sigf, tauf)
mu = 569.653673
sig = 52.678700
tau = 99.683349
>>>
>>> [xx, yy] = histogram(nums)
>>> yf = [AA*exgauss(ele, muf, sigf, tauf) for ele in xx]
>>>
>>> plot1 = gdata(xx, yy, with_="boxes", title="histogram")
>>> plot2 = gdata(xx, yf, with_="lines lw 3 lc 2 lt 1", title="fit")
>>> g.plot(plot1, plot2)
```

These will produce the (similar) fit in figure 4.

5 Bugs and Feedback

These classes 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.

6 License

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

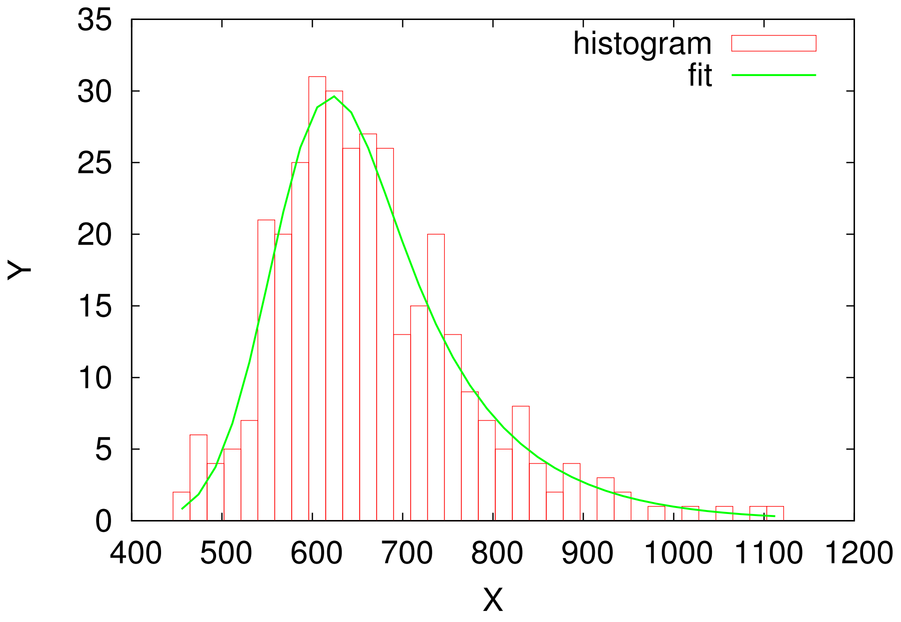

Figure 4: Ex-Gaussian fit for a set of data.