RandGen:
A Program for Generating Random Numbers
Version 2.0

Jeff Miller
Department of Psychology
University of Otago
Dunedin, New Zealand

Sept 17, 2002

Contents

1 Introduction

RandGen is a general-purpose purpose program for generating random numbers, designed for use in computer simulations. The goal was to free you from having to work out the details of generating the specific kinds of random numbers that you want. Instead, you just tell RandGen what sorts of random numbers are needed, and it generates them for you.

RandGen will generate random numbers from many different kinds of probability distributions. The basic scenario is that it generates random samples of a desired size from a desired population. Each sample may be measured on a number of variables, and these variables may be correlated or independent. As a special case, RandGen will also generate random permutations of integers (see the special “shuffle” mode).

1.1 A Simple Example

A simple example will illustrate how RandGen operates. First, you prepare an input file (call it "1stEx.In") describing the random numbers to be generated (see Table 1 for an example). Then, at a command-line prompt, you issue the command

```
RandGen 1stEx
```

RandGen reads the information in 1stEx.In and produces two output files: (1) a file containing the random numbers that were generated (see Table 2 for an example), and (2) a file containing some summary statistics computed from these random numbers (see Table 3 for an example).

Table 1: An example of a RandGen input file, prepared by the user with any plain-text (ASCII) editor.

```
* Example showing how to generate 12 random numbers from a
* Normal distribution with mean 100 and standard deviation 15.
Samplesize 12
NVariables 1
Normal(100,15)
```

The numbers in 1stEx.Dat are the random numbers for use in a simulation. For example, they might be read by a program carrying out the simulation, or they might be imported directly into a statistical package for analysis. The summary information in 1stEx.Sum is provided so you can check up on the random number generation.

1.2 Capabilities of RandGen

The complete list of probability distributions that can be used for generating random numbers within RandGen, along with their parameters, can be found in section 5. RandGen can generate more than 50 different distributions of random numbers. For example, it can generate uniform, exponential, gamma, chi-square, t, and F random numbers instead of normal ones. See section 5. for more information about the possible distributions of random numbers.

RandGen can also generate more than one variable at a time, for situations where you want to simulate samples of multivariate observations (a maximum of 20 variables in the current version). You can build any desired true correlations into the underlying multivariate distribution from which these multivariate samples are taken. In the special case of bivariate random numbers, RandGen provides a choice among three different types bivariate structures (described in section 7).

Table 2: A set of random numbers generated by RandGen from the input file 1stEx.In (shown in Table 1). These numbers are written to the file 1stEx.Dat.

```
1.03567923462157E+0002
7.99005441505953E+0001
1.12283845079420E+0002
1.33209049874592E+0002
1.06709082584233E+0002
1.02176624742546E+0002
9.46441347800793E+0001
9.86770495570442E+0001
9.58360087839738E+0001
1.08312067310089E+0002
9.41720651255518E+0001
9.46809573223893E+0001
```

Table 3: RandGen's summary of the random numbers shown in Table 1. This summary is written to the file 1stEx.Sum.

Summary of random numbers in file 1stEx.Dat; SampleSize = 12

X[1]: Normal(100,15)
True distribution: Mu = 100.0000; Sigma = 15.0000
This random sample: Mean = 102.0141; SD = 12.4185
Goodness of fit ChiSqr(3) = 4.500, p = 0.212

2 Structure of the Input File

RandGen reads all of its program control information from an ASCII file of input parameters that you create with an editor prior to starting RandGen. This section describes the different parameters and how to set them. The file "Examples.In" shows a number of examples of different input files, and it may be easier to learn about the parameters by looking at the examples in the file rather than reading this section of the documentation.

Here are some general comments about the parameters:

- With some crucial exceptions, one parameter is specified on each line of the input file.
- The parameters can appear in (almost) any order.
- Most parameters are optional and have reasonable defaults.
- None of the parameters are case-sensitive. In the headings below, I have capitalized the *required* parts of the parameter names so that you can see how to abbreviate them.
- Parameters can be followed on the same line by comments. By default, the asterisk is used to start a comment, but this can be changed.

The parameters will be described approximately in order from the most-commonly-used to least-commonly-used.

Some of the terminology used in this section assumes that you have the basic concepts of random variables and their probability distributions (marginal and multivariate). If you do not understand this terminology, you may find it helpful to read the background provided in section 6, or the references listed in the reference section.

2.1 Setting the Number of Variables

The "NVARIABLES" parameter tells RandGen how many variables are to be generated for each of the random cases. Critically, this parameter must be followed by a series of lines indicating the marginal distributions of the variables to be generated, one per line. Example:

```
NVariables 3       * This indicates that 3 variables should be generated.
Normal(0,1)        * Variable 1 is normal with μ = 0 and σ = 1.
Uniform(0,100)     * Variable 2 is uniform between 0 and 100.
t(15)              * Variable 3 is a t with 15 degrees of freedom.
```

2.2 Setting the Sample Size

The SAMPLESIZE parameter determines how many random cases are generated (with NVARIABLES variables per case). Example:

```
SAMPLESIZE 100     * Generate samples of 100 cases.
```

2.3 Setting the Correlations

This parameter only has meaning when NVARIABLES is greater than one. It is used to specify the values of the desired correlations between variables in the underlying multivariate distribution. Examples:

```
* An example with 2 variables:
NVARIABLES 2        * Generate variables with the next two marginals.
Normal(0,1)
Uniform(0,1)
CORRELATION 0.7     * The underlying bivariate correlation is 0.7

* An example with 3 variables:
NVARIABLES 3
Normal(0,1)
Uniform(0,1)
Gamma(5,0.1)
CORRELATION
0.7 0.6             * The true correlations are: normal and uniform -¿ .7,
0.5                 * normal and gamma -¿ .6, and uniform and gamma = .5
```

Note that a single correlation is specified on the same line as the CORRELATION parameter, but a correlation matrix (3+ variables) starts on the next line.

2.4 Setting RhoController Directly

This parameter only has meaning when NVARIABLES is greater than one. It is used to specify the values of RandGen's internal RHOCONTROLLER parameters. These are discussed in more detail in section 7.4, but in brief these are the values that RandGen adjusts to get the desired correlations. If you specify the correlations between variables using the CORRELATION parameter, then RandGen will conduct a possibly time-consuming search to find the necessary RHOCONTROLLER values to produce those desired correlations. The RHOCONTROLLER values obtained through the search are written to the summary (.Sum) file, and from there you could copy them into the input file. That way, you would specify RHOCONTROLLER values rather than CORRELATION values for the next run (and all future runs), and RandGen would go faster on all subsequent runs because it would not have to repeat the RHOCONTROLLER search each time it was invoked. Examples:

```
* An example with 2 variables:
NVARIABLES 2            * Generate variables with the next two marginals.
Normal(0,1)
Uniform(0,1)
RHOCONTROLLER 0.7153    * Previous search showed this gives correlation = 0.7

* An example with 3 variables:
NVARIABLES 3
Normal(0,1)
Uniform(0,1)
Gamma(5,0.1)
RHOCONTROLLER           Previous search showed these give
0.7153 0.6126           * true correlations of: normal and uniform = .7,
0.5230                  * normal and gamma = .6, and uniform and gamma = .5
```

As with the correlation parameter, a single value is specified on the same line as the RHOCONTROLLER parameter, but a matrix (3+ variables) starts on the next line.

2.5 Multivariate Structures: Independent, Normal, Bands, or Mixture

When two or more variables are being generated, the relationship between the variables can be specified. If neither CORRELATION nor RHOCONTROLLER is specified, then it is assumed that the variables are stochastically independent

of one another (so the true correlation is zero). You can add the parameter "INDEPENDent" to the input file to confirm that (as a reminder to yourself).

If either CORRELATION or RHOCONTROLLER is specified, then the default is that the variables are derived from the multivariate normal structure, as described in section 7.1. You can add the parameter "NORMAL" to the input file to confirm that (as a reminder to yourself).

If either CORRELATION or RHOCONTROLLER is specified and you are generating exactly two variables, then you have two additional options for the bivariate structure: "MIXTURE", as described in section 7.2, or "BANDS", as described in section 7.3. You must add one of these two parameters to the input file to request either of these options.

In summary, here are the options for controlling the multivariate structure when generating more than one variable. You should specify no more than one of these parameters, and you need not specify any if you are happy with the default.

INDEPENDent	* Default if neither RHOController nor CORRELATIONs specified.
NORMAL	* Use normal multivariate structure. * Default if either RHOController or CORRELATIONs specified.
BANDS	* Use bands multivariate structure (NVARIABLES=2 only).
MIXTURE	* Use mixture multivariate structure (NVARIABLES=2 only).

2.6 Controlling the Format of the Random Numbers

By default, RandGen writes random numbers in scientific notation, separated by spaces, like this:

1.03567923462157E+0002 7.99005441505953E+0001

Three parameters can be used to modify this:

TAB DELIMited This parameter indicates that the random numbers should be separated by TABs instead of spaces, as preferred by some statistics packages.

COMMA DELIMited This parameter indicates that the random numbers should be separated by commas instead of spaces, as preferred by some statistics packages.

FIELDWIDTH This parameter is used to get numbers in standard decimal notation instead of scientific. *Starting on the next line*, you specify the number of characters and the number of decimal places for each variable in the output. Example:

NVariables 2 Normal(0,1) Uniform(0,100) Fieldwidths for the variables are as follows:
8 2 * 8 characters for variable 1, with 2 decimal places
9 3 * 9 characters for variable 2, with 3 decimal places

would yield output like (where b is blank)

bb103.57bbb79.901

Note that (a) RandGen uses full precision in computing summary statistics, so the sample mean, etc, reported in the summary file may not exactly match those computed from the values in the Dat file; and (b) using narrow fields with few decimal places reduces the correlations of the data in the Dat file, possibly quite a bit. So, allow plenty of precision in the output.

2.7 Generating Multiple Output Files

The NFILES parameter controls the number of output data files generated, so that you can generate more than one sample of random numbers with the same specifications in a given run if you so desire. NFILES is 1 by default. Example:

```
SAMPLESIZE 100    * Generate samples of 100 cases.
NVARIABLES 1      * One variable
Normal(100,15)    * Normal with μ = 100 and σ = 15.
NFILES 10         * Generate 10 files of 100 cases each.
```

If NFILES is specified to be more than one, the output files are named 00000001.Dat, 00000002.Dat, 00000003.Dat, ...

2.8 Omitting the Summary File

Include the command "SKIP SUMmary" to tell RandGen to skip computation of the summary file. If you do that, it will run a bit faster.

2.9 Multiple Specifications in a Single Input File

RandGen allows you to save the specifications for more than one type of random number generation in a single input file (to avoid cluttering up your disk with lots of small files). The file "Examples.In" is an example. Note, however, that RandGen will only process one set of specifications per run. If you want to process more than one, you must use a batch file (the file "Examples.Bat" is an example).

Within an input file, each of the separate sets of specifications starts with an arbitrary label and ends with the parameter "END". In principle, the label can be any alphanumeric string you like, but in practice it is convenient to use 8 or fewer characters so that the label can be used as the name for the output file (see section 2.10). For example, suppose this is the file MultSpec.In:

```
Mu10              * Label for first set of specifications.
NVariables 2      * Generate 2 variables.
Normal(10,1)      * One variable is normal with mu = 10 and sigma = 1,
Uniform(0,1)      * and the other is uniform(0,1).
Samplesize 100    * Generate 100 cases.
End               * End of first set of specifications.

Mu20              * Label for second set of specifications.
NVariables 2      * Same as the first except
Normal(20,1)      * mu = 20
Uniform(0,1)
Samplesize 100
End               * End of second set of specifications.
```

It is fine to use blank lines in the file as spacers, for readability.

To choose which set of specifications you want, invoke RandGen with a second command-line parameter to indicate the label of the specifications you want to process, as illustrated in the file "Examples.Bat". Here, for example, you could say:

```
RandGen MultSpec Mu10
```

to process the first set of specifications or

```
RandGen MultSpec Mu20
```

to process the second.

Alternatively, you may include a "GOTO" command as the first line of the input file, like this:

```
GOTO Mu20     * Skip to Mu20
```

With this as the first line of MultSpec.In, you could simply run

```
RandGen MultSpec
```

to process Mu20.

2.10 Setting the Output File Name

Output data are typically written to files called "foo.Dat", and summaries of these data are written to files called "foo.Sum". If RandGen is called with one parameter, the default for "foo" is the first parameter (i.e., the name of the input file); if RandGen is called with two parameter, the default for "foo" is the second parameter (i.e., the name of the label within the input file–see section 2.9). For example, with the command

```
RandGen Examples
```

the default for "foo" is "Examples", whereas with the command

```
RandGen Examples Exmpl10
```

the default for "foo" is "Exmpl10".

The default name of the output file can be overridden by specifying the parameter:

OUTFILE foo2

Now the output files will be called "foo2.dat" and "foo2.sum" regardless of the command-line parameters.

An exception arises if you are generating more than one output file (e.g., NFILES 100). In this case, the output data files are called 00000001.dat, 00000002.dat, 00000003.dat, ..., regardless of the command-line parameters or OUTFILE command. The summaries of all data sets will be written to a single "foo.sum" file, whose name is still determined by the command-line parameters or OUTFILE command.

2.11 Special Modes of Operation

RandGen has four special modes of operation where it does something *other than* generate random numbers as described above. Each one of these modes is invoked by specifying one of the following parameters, with the indicated result:

2.11.1 LIMITCHeck

In this mode, RandGen will simply compute the largest and smallest correlations possible with the specified marginals. These correlations are sometimes called the Frechet bounds, and they are found by letting $F_y(Y) = F_x(X)$ (largest positive correlation) or by letting $F_y(Y) = 1 - F_x(X)$ (largest negative correlation). For an example, see Examp11 in "Examples.in". If this option is specified, the bounds are written to the .Sum file; no .Dat file is written.

2.11.2 COMPUTERHO

This mode is useful when you want to check the correlation that will be obtained with a certain value of RhoController. You simply specify RhoController, and RandGen computes the correlation that will result. For examples, see Examp12, 13, and 14 in "Examples.in". If this option is specified, the computed values are written to the .Sum file; no .Dat file is written.

2.11.3 SHUFFLE HiVal

This mode and the next don't really belong in this program, but here they are anyway. This mode produces random permutations of the integers from 1 to HiVal, which can be used for sampling without replacement. To simulate a random deck of 52 cards, for example, you could use

```
SAMPLESIZE 50
SHUFFLE 52        * Produce 50 random permutations of the numbers 1-52.
```

Each line of the output Dat file will contain the numbers 1-52 in a random order, so the 50 lines correspond to 50 random permutations of the deck. For other examples, see Examp15 and Examp16 in "Examples.in". If this option is specified, the permutations are written to the .Dat file, but no .Sum file is produced.

2.11.4 SUBSHUFFLE HiVal WantVal

This mode also produces random permutations of the integers from 1 to HiVal, but it only writes out the first WantVal elements of each permutation. It can be used for sampling without replacement where you don't care to look at the full permutation. For example,

```
Samplesize 50
SUBSHUFFLE 52 5    * Produce 50 random permutations of the numbers 1-52,
                   * writing out only the first 5 numbers in each permutation.
```

For other examples, see Examp17 and Examp18 in "Examples.in". If this option is specified, the permutations are written to the .Dat file, but no .Sum file is produced.

2.12 Controlling the Accuracy of Computations

RandGen has several parameters that can be used to control numerical computations.

APPROXimation NSteps This parameter is only relevant when RandGen has to search for RhoController values that yield the desired correlations. The parameter controls the number of steps that RandGen uses in approximating each distribution during the search. For example, the default for NSteps is 200, which means that during searching each distribution is temporarily modeled as 200 equally-likely values at the points with CDF values of .0025, .0075, .0125, .0175, . . . , .9975. Higher numbers of steps yield better approximations but slower searches. Example:

```
Approx 300    * Search more slowly to get more accurate approximation.
```

SEARCHERROR max-allowable-error This parameter is also only relevant when RandGen has to search for RhoController values that yield the desired correlations. The parameter controls the maximum error that RandGen will accept between the desired correlation and that produced by a given value of RhoController. The default is 0.001, which means that RandGen will keep adjusting RhoController until it finds a value that will yield a correlation within 0.001 of the desired value.

```
SearchError 0.01    * Search until produce desired correlation ±.01
```

CHISQuare NBins In the summary output file, RandGen computes a Chi-square goodness of fit statistic to evaluate the distribution of the generated random numbers against the specified population marginal. By default, it uses 1. . . 10, 20, 50, 100, 200, 500, or 1000 bins, depending on the sample size. This parameter allows you to specify any desired number of bins you choose, overriding the default. Example:

```
CHISQuare bins 15    * Compute the chi-square(s) using 15 bins
```

INTEGRALPRECISion new-precision-value This parameter controls the required accuracy of numerical integrations. It will only be relevant when these are being computed, and you have no real way to determine whether they are except by trial and error. The default value is 0.0000001; larger values will give faster runs but less accuracy. Example:

```
INTEGRALPRECISion 0.001    * Very lenient criterion for numerical integrals.
```

INVERSEPRECISIONX new-precision-value This parameter controls the required accuracy of computing inverse CDFs by searching: how close to the desired X value does the program have to get before it starts searching? It will only be relevant when inverse CDFs are being computed, and you have no real way to determine whether they are except by trial and error. The default value is 0.0000001; larger values will give faster runs but less accuracy. There is an analogous parameter called InversePrecisionP, which controls how close the program has to get to the desired P value. Examples:

```
INVERSEPRECISIONX 0.001    * Compute inverses to within an X value of .001 (lenient).
INVERSEPRECISIONP 0.001    * Compute inverses to within a P value of .001 (lenient).
```

Important note: If you want to alter IntegralPrecision or InversePrecision, it is best to do so *before* the variables are specified with the NVARIABLES command.

2.13 Random Number Seeding

By default, RandGen starts its random number generator with a randomly generated seed each time it is run. There are three command-type parameters that you can use to modify this behavior.

SEED SAVE START filename This tells RandGen to write the initial seed-state of its random number generator (after the random start) to a file called filename.

SEED SAVE END filename This tells RandGen to write the final seed-state of its random number generator (after generating all the random numbers) to a file called filename.

SEED START filename This tells RandGen to read the initial values of its random number generator from a file called filename. This file should have been written with one of the two "SEED SAVE" commands just described.

Examples:

```
* In this example, successive runs will give identical random numbers:
* Seedfile should already have been written by using a SEED SAVE
* command in a previous run of RandGen.
SEED START seedfile
```

```
* In this example, successive runs will "continue on" the random number
* generator, guaranteeing no repetition of the random number generator
* until it reaches the end of its period. SEED START seedfile
SEED SAVE END seedfile
```

2.14 Input File Comments

By default, RandGen treats anything following an asterisk (*) as a comment–i.e., RandGen ignores it. You can, however, change the comment marker if you want. Example:

```
Comment !!      * Starting with the NEXT LINE, comments are set off by !!.
NVariables 2    !! Two variables.
Normal(0,1)     !! Standard normal, and
Uniform(0,1)    !! ... uniform
```

3 Starting RandGen

The command to run RandGen is simply "RandGen RootFileName", where RootFileName can be up to eight characters long. The parameters needed for program operation are specified in a file "RootFileName.In", which must be prepared in advance with an ASCII text editor. The random numbers are written to a file called "RootFileName.Dat", and summary output is written to a file called "RootFileName.Sum".

RandGen is sometimes pretty slow. You should be able to abort it with control-break, if you get impatient.

4 Program Output

RandGen writes two output files, called "foo.Sum" and "foo.Dat", where "foo" is determined as described in section 2.10. The "foo.Dat" file simply contains the columns of generated random numbers. Each line corresponds to a different random sample, and the different numbers on a single line are the different variables. "Foo.Sum" contains other information about the run that may be useful. Here is an example:

```
Summary of random numbers in file examp05.Dat;  SampleSize = 1000

  X[1]: Uniform(0,1)
    True distribution:  Mu   = 0.5000; Sigma = 0.2887
    This random sample: Mean = 0.5230; SD    = 0.2869
    Goodness of fit ChiSqr(20) =     16.520, p = 0.684
```

```
X[2]: Normal(0,1)
  True distribution:  Mu   = 0.0000; Sigma = 1.0000
  This random sample: Mean = 0.0699; SD    = 1.0108
  Goodness of fit ChiSqr(20) =     28.840, p = 0.091

X[3]: Gamma(4,0.01)
  True distribution:  Mu   = 400.0000; Sigma = 200.0000
  This random sample: Mean = 400.1263; SD    = 202.5984
  Goodness of fit ChiSqr(20) =     19.480, p = 0.491

Multivariate structure: Normal

  Desired correlation  matrix =
         0.7000   0.4000
                  0.2000

  RhoController        matrix =
         0.7153   0.4197
                  0.2053

  Observed correlation matrix =
         0.7355   0.4665
                  0.2488
```

For each variable generated, the true mean (mu) and standard deviation (sigma) are computed and printed out, as are the observed mean and SD for the random sample that was generated. In addition, a goodness-of-fit Chi-square test is computed, and its value printed out. The p values associated with these chi-square tests should rarely be less than 0.05 (about once in 20 runs); if they are small more often than that, there is something wrong with the random number generator for that distribution.

After all of the individual variables have been analyzed, some information is provided about the multivariate distribution. Usually, the desired correlation matrix was specified in the input file, and the RhoController matrix was computed by the search process. The observed correlation matrix is computed from the data in the random sample (standard Pearson r's).

5 Available Marginal Distributions

5.1 Continuous Distributions

Here are the primitive continuous distributions that have been at least partially implemented so far:

Beta(A, B**)** The Beta distribution is defined over the interval from zero to one, and its shape is determined by its two parameters A and B. Its PDF is

$$f(x) = \frac{1}{\beta(A,B)} x^{A-1}(1-x)^{B-1}, \quad 0 < x < 1$$

The mean is $A/(A+B)$, and the variance is $AB(A+B)^{-2}(A+B+1)^{-1}$.

Cauchy(L, S**)** This distribution is defined in terms of location and scale parameters L and $S > 0$, respectively. Its PDF is

$$f(x) = \frac{1}{\pi S\left[1 + \left\{\frac{x-L}{S}\right\}^2\right]}$$

ChiSquare(df) This is a generalization of the distribution of the sum of df independent squared standard normals. Its parameter is df — a positive real number.

Chi(df) This is the distribution of the positive square root of a ChiSquare random variable. Its parameter is *df* — a positive real number.

Cosine For $0 \leq x \leq \Pi/2$, $f(x) = \cos(x)$ and $F(x) = \sin(x)$.

ExGaussian(μ, σ, λ) This is the distribution of the sum of independent Normal and Exponential random variables. Its parameters are the μ and σ of the Normal, and the rate λ of the Exponential.

ExGaussRat(μ, σ, r) This is just a reparametrization of the ExGaussian. Its parameters are the μ and σ of the Normal, and the ratio, r, of the mean of the exponential to the sigma of the normal.

Exponential(λ) This distribution is well-known. By default, the parameter is the rate λ; the mean is $1/\lambda$.

ExpSum($r1, r2$) This is the sum (convolution) of two exponentials with *different* rates. The two parameters are the two rates, which must be different enough to avoid numerical errors. For the convolution of exponentials with the same rates, of course, you should use the Gamma.

ExpSumT($r1, r2$,Cutoff) This is the sum (convolution) of two exponentials with *different* rates truncated at a given cutoff value. The first two parameters are the two rates, which must be different enough to avoid numerical errors; the third parameter is the upper truncation point. For the convolution of exponentials with the same rates, of course, you should use the Gamma.

ExpoNo(μ, σ) I just invented this as an ad-hoc solution for a problem I was working on one time, so I don't know whether it will ever be useful again. And I certainly don't know whether I gave it a reasonable name. In any case, it is a transformation of a normal random variable X. Specifically, it is the distribution of

$$Y = \frac{e^X}{1 + e^X}$$

where X has a normal distribution with mean μ and standard deviation σ. The two parameters of this distribution are the μ and σ of the underlying normal X.

ExtremeVal(α, β) Extreme-value Type I distribution (a.k.a. Fisher-Tippett distribution, Gumbel distribution, sometimes also called the double exponential distribution, to be confused with the Laplace distribution), with parameters α and $\beta > 0$. The CDF is

$$F(x) = \exp\left[-e^{-(x-\alpha)/\beta}\right]$$

ExWald(μ, σ, a, λ) This is the distribution of the sum of two independent random variables: one from a three-parameter Wald distribution with parameters (μ, σ, a); and one from an exponential distribution with rate λ.

F(dfNumer,dfDenom) This is Fisher's distribution of the ratio of two independent normed Chi-square distributions, as commonly used in linear models (e.g., analysis of variance). The two integer parameters are the degrees of freedom of the numerator and denominator, respectively.

Gamma(N, λ) This is the distribution of the sum of N exponentials, each with rate λ. In this distribution, N must be a positive integer. In the RNGamma distribution (see below), N is any positive real.

Geary(SampleSize) The Geary statistic arises in testing to see whether a set of observations come from a normal distribution (D'Agostino, 1970).

HypTan(Scale) This is the Hyperbolic Tangent distribution, whose PDF and CDF are[1]

$$\begin{aligned} f(x) &= \frac{4 \cdot \beta}{\left[e^{\beta x} + e^{-\beta x}\right]^2} \\ F(x) &= \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}} \end{aligned}$$

where β is the scale parameter. This distribution arises as a model of psychometric functions (e.g., Strasburger, 2001).

[1] I thank Rolf Ulrich for supplying these.

Inverse Gaussian See the Wald distribution.

Laplace(L, S) Also known as the double exponential. In terms of location and scale parameters, L and $S > 0$, respectively, the PDF is

$$f(x) = \frac{1}{2S} e^{-|x-L|/S}$$

Logistic(μ, β) This distribution is defined in terms of a location parameter μ and a scale parameter β. The cumulative form of the distribution is

$$F(x) = \frac{1}{1 + e^{\frac{-(x-\mu)}{\beta}}}$$

LogNormal(μ, σ) This is the distribution of X such that $\ln(X)$ is normally distributed. The parameters are the μ and σ of the normal.

Naka-Rushton(Scale) This is the distribution of $X \geq 0$ such that

$$\begin{aligned} f(x) &= \frac{2 \cdot x \cdot \alpha^2}{(1 + (\alpha \cdot x)^2)^2} \\ F(x) &= \frac{(\alpha \cdot x)^2}{1 + (\alpha \cdot x)^2} \end{aligned}$$

where α is the scale parameter.[2] In the actual distribution, moments above the first do not exist; they do exist in RandGen's truncated version of the distribution, however.

NoncentralF(dfNumer,dfDenom,Noncentrality) This is the distribution of the ratio of independent noncentral and central chi-squares, with the former in the numerator. It is most often used in the computation of power of the F test. The noncentrality parameter is defined in terms of the dfNumer normal random variables whose sum of squares is yields the chi-square in the numerator. Specifically,

$$\lambda = \sum_{i=1}^{\text{dfNumer}} \Lambda_i^2$$

where Λ_i is the expected value of the ith random variable contributing to this sum of squares.

Normal(μ, σ) I'll bet you know this one already. Parameters are μ and σ, not σ^2.

Pareto1(K,A) This is a Pareto distribution of the first kind, as defined by Johnson, Kotz, and Balakrishnan (1994, vol 1, p 574), with PDF and CDF

$$\begin{aligned} f(x) &= A \cdot K^A \cdot x^{-(A+1)} \\ F(x) &= 1 - \left(\frac{K}{A}\right)^A \end{aligned}$$

where $K > 0$, $A > 0$, and $x > K$.

Quantal(Threshold) This distribution is related to the Poisson. This is the distribution of $X \geq 0$ such that

$$F(x) = 1 - \sum_{t=0}^{T-1} \frac{x^t}{t!} e^{-x}$$

This distribution arises as a model of psychometric functions in visual psychophysics (e.g., Gescheider, 1997, p. 85). The threshold parameter, T, represents an observer's fixed threshold for the number of quanta of light that must be detected before saying "Yes, I saw the stimulus." Quanta are assumed to be emitted from the stimulus according to a Poisson distribution with parameter x. Then, $F(x)$ is the psychometric function for the probability of saying "Yes" as a function of the mean number of quanta, x, emitted by the stimulus. Note that it makes no real sense to think of x as a random variable in this example, but the probability distribution provides a useful model anyway.

[2] I thank Rolf Ulrich for supplying the PDF.

Quick(Scale,Shape) This is the distribution of $X \geq 0$ with PDF and CDF[3]

$$\begin{aligned} f(x) &= \frac{2^{-\left(\frac{x}{\alpha}\right)^{\beta}} \cdot \left(\frac{x}{\alpha}\right)^{\beta} \cdot \beta \cdot \ln(2)}{x} \\ F(x) &= 1 - 2^{-\left(\frac{x}{\alpha}\right)^{\beta}} \end{aligned}$$

where α is the scale parameter and β is the shape parameter. This distribution arises as a model of psychometric functions (e.g., Quick, 1974; Strasburger, 2001).

Rayleigh(σ) If Y_1 and Y_2 are independent normal random variables with mean 0 and standard deviation σ, then $X = \sqrt{Y_1^2 + Y_2^2}$ has a Rayleigh distribution with scale parameter σ. The PDF is

$$f(x) = e^{-x^2/(2\sigma^2)} \frac{x}{\sigma^2}$$

RNGamma(RN, λ) See "Gamma". In this version, the shape parameter RN is a real number rather than an integer.

rPearson(SampleSize) This is the sampling distribution of Pearson's r (correlation coefficient) under the null hypothesis that the true correlation is zero (and assuming the usual bivariate normality). The parameter is *SampleSize*, the number of pairs of observations across which the correlation is computed.

t(df) Student's t-distribution, with parameter df.

Triangular(B, T) In this distribution the density function has the shape of an equilateral triangle across some range. The parameters are the bottom (B) and the top of the range (T). The PDF is then:

$$f(x) = \begin{cases} (x - B) \times H_p & \text{if } B \leq x \leq \frac{B+T}{2} \\ (T - x) \times H_p & \text{if } \frac{B+T}{2} \leq x \leq T \end{cases}$$

where H_p is the height of the PDF at its peak, adjusted to so that the total area of the triangle is 1.0.

TriangularG(B, P, T) In this (more general triangular) distribution, the density function has the shape of a not-necessarily-equilateral triangle across some range. The parameters are the bottom of the range (B), the point at which the triangle reaches its maximum (P), and the top of the range (T). The PDF is then:

$$f(x) = \begin{cases} \frac{(x-B) \times H_p}{P-B} & \text{if } B \leq x \leq P \\ \frac{(T-x) \times H_p}{T-P} & \text{if } P \leq x \leq T \end{cases}$$

where H_p is the height of the PDF at its peak, adjusted to so that the total area of the triangle is 1.0.

Uniform(B, T) This is the distribution in which all values are equally likely within some range. The parameters are the bottom and the top of the range, B and T.

UniGap(T) This is an equal-probability mixture of two uniform distributions, one extending from $-T$ to 0 and the other extending from T to $2 \cdot T$. It is "model 4" of Sternberg and Knoll (1973). The median is somewhat arbitrarily defined as $T/2$.

Wald(μ, λ) This distribution arises in problems involving the first passage time with Brownian motion and positive linear drift. As defined by Johnson, Kotz, and Balakrishnan (1994, Volume 1), the standard two-parameter inverse Gaussian has the PDF:

$$f(x) = \left[\frac{\lambda}{2\pi x^3}\right]^{1/2} \exp\left\{-\frac{\lambda}{2\mu^2 x}(x - \mu)^2\right\}$$

where $\mu > 0$, $\lambda > 0$, and $x \geq 0$. μ is the mean and μ^3/λ is the variance.

[3] I thank Rolf Ulrich for supplying these.

Wald3(μ, σ, a) This is a three-parameter version of the Wald distribution. Specifically, assume a one-dimensional Wiener diffusion process starting at position 0 at time 0 and drifting with average rate μ and variance σ^2, and consider X to be the first passage time through position a. The PDF of X is

$$f(x) = \frac{a}{\sigma\sqrt{2\pi x^3}} \cdot \exp\left[-\frac{(a-\mu x)^2}{2\sigma^2 x}\right]$$

where all three parameters and x must be positive.

Weibull(Scale,Power,Origin) As defined by Johnson & Kotz (1970, p. 250): "X has a *Weibull distribution* if there are values of the parameters $c(>0)$, $\alpha(>0)$, and ν_0 such that

$$Y = \left[\frac{(X-\nu_0)}{\alpha}\right]^c$$

has the exponential distribution with rate = 1". Here, the parameters c, α, and ν_0 are referred to as the "scale," "power," and "origin" parameters, respectively.

The CDF of the Weibull is therefore

$$F(x) = 1 - \exp(-[(x-\nu_0)/c]^\alpha)$$

Computations are increasingly inaccurate for powers less than about 0.9, however.

5.2 Discrete Distributions

Here are the primitive *discrete* distributions that have been at least partially implemented so far:

Binomial(N, p) The distribution of the number of successes in N Bernoulli trials, with probability p of success on each trial.

Constant(C) This is a degenerate distribution that always takes on the same value. Its parameter is that value. Perhaps surprisingly, it can be convenient to have this distribution available. *Warning:* For technical reasons, the constant distribution does not work well in many of the derived distributions discussed in the next section. Thus, it should be avoided whenever possible. For example, you should always use:

```
LinearTrans(Gamma(2,.01),1,100)
```

rather than the equivalent

```
Convolution(Gamma(2,.01),Constant(100))
```

Geometric(P) The distribution of the trial number of the first success in a sequence of Bernoulli trials, where P is the probability of success on each try.

List(filename) This random variable allows you to define any discrete set of X values, each with its own arbitrary probability. The command

```
List(FileName)
```

tells RandGen to read the distribution from the indicated file. The first line in the file contains the number of X values in the distribution. After that, there should be one line for each X value, with the first number on the line being the X value itself and the second number being the probability of that X value. These values need not be sorted, and in fact the same X value can appear on several different lines, in which case the associated probabilities will be summed. (If X's do appear on more than one line, then the first line in the file should actually contain the number of X-containing lines rather than the number of distinct X's.)

Poisson(U) X has a Poisson distribution with parameter U if

$$\Pr(X = x) = \frac{e^{-U}U^x}{x!}, \quad x = 0, 1, 2, \ldots, U > 0$$

The mean and variance both equal U.

UniformInt(Low,High) This is the distribution of equally likely integer values between the two integer parameters, Low and High, inclusive.

5.3 Transformation Distributions

RandGen can form a new random variable (Y) by taking a mathematical transformation of an existing one (X). The following table lists the transformations recognized by RandGen, illustrating the syntax for each. Also listed are the constraints on the values of X.

Transformation	Example of Syntax	Constraints on Values of X
ArcSin ($Y = \sqrt{(}\phi(X/2))$)	`ArcSinT(Uniform(.5,1))`	
Exponential ($Y = e^X$)	`ExpTrans(Uniform(.5,1))`	X not too far from 0.
Inverse ($Y = 1/X$)	`InverseTrans(Uniform(.5,1))`	X not too close to 0.
Linear ($Y = A \times X + B$)	`LinearTrans(Uniform(.5,1),2,10)`	
Natural Log ($Y = \ln[X]$)	`LnTrans(Uniform(0.5,1))`	$X > 0$
Power ($Y = X^p$)	`PowerTrans(Uniform(.5,1),2)`	$X > 0$

where $\phi(Z)$ is the probability that a standard normal random variable is less than Z.

5.4 Derived Distributions

RandGen also knows about various sorts of distributions that can be derived from one or more primitive or "basis" distributions. In most cases, RandGen can compute moments, PDF's, CDF's, random numbers, etc, for the derived distribution just as it can for the primitive distributions defined above.

Convolution(RV1,RV2) This is the distribution of a sum of *independent* random variables, RV1 and RV2, where RV1 and RV2 are each legal distributions in their own right. For example,

```
Convolution(Normal(0,1),Uniform(0,1))
```

specifies the convolution of these normal and uniform distributions, and

```
Convolution(Normal(0,1),Uniform(0,1),Gamma(3,0.01))
```

specifies the convolution of the three indicated distributions.

In general, to define a convolution, the user types something of the form:

```
Convolution(BasisDist1(Parms),...,BasisDistK(Parms))
```

where `Parms` stands for the parameters associated with each of the distributions. There are K random variables summed together, and the distributions of these summed variables are simply listed, separated by commas.

RandGen is not very smart about convolutions. At this point, it only knows how to compute means, variances, and random numbers in an intelligent way. Everything else is computed using (recursive) numerical integration, which tends to be pretty slow. Also, RandGen does not "realize" that some convolutions result in a new distribution about which it already knows (e.g., convolution of two normals is normal). Thus, computations involving these convolutions proceed via numerical integration even though direct computation would be possible.

The current version can handle convolutions where all distributions are discrete, all are continuous, or some are is discrete and some continuous, but it cannot handle convolutions in which one or more distributions are mixed (i.e., partly discrete and partly continuous).

I would be very happy for suggestions on how to augment RandGen's handling of convolutions, especially those accompanied by Pascal code.

ConvolutionIID(N,RV) This is just an easier way to specify a convolution when all N summed random variables have the same distribution, RV.

```
ConvolutionIID(3,Uniform(0,1))
```

is the same as

```
Convolution(Uniform(0,1),Uniform(0,1),Uniform(0,1))
```

Difference(RV1,RV2) This is the distribution of the difference of two *independent* random variables, RV1 minus RV2, where RV1 and RV2 are each legal distributions in their own right. For example,

```
Difference(Uniform(0,1),Uniform(0,1))
```

specifies a difference between two standard uniform distributions, which ranges from -1 to 1 (not uniformly). RandGen handles difference distributions dumbly, like convolutions. The current version can handle differences where both distributions are discrete, both are continuous, or one is discrete and one continuous, but it cannot handle differences in which one or both distributions are mixed (i.e., partly discrete and partly continuous).

Mixture(p1,RV1,p2,RV2,...,pk,RVk) Mixtures are distributions formed by randomly selecting one of a number of random variables. For example, `Mixture(0.5,Normal(0,1),0.5,Uniform(0,1))` defines a random variable that comes from a standard normal half the time and a standard uniform the other half of the time. In general, the format of this distribution is:

`Mixture(`p_1`,BasisDist1(Parms),`p_2`,BasisDist1(Parms),...,`p_k`,BasisDistK(Parms))`

and the p_i's must sum to one (it is also legal to omit p_k).

InfMix(RV1,MixParm,RV2(Parms)) The InfMix distribution is an infinite mixture, formed when a parameter of one distribution is itself randomly distributed according to another distribution. For example,

```
InfMix(Normal(0,5),1,Uniform(10,20))
```

defines a random variable that comes from a normal distribution with standard deviation 5. The first parameter of that distribution (as signified by the "1" between the two distribution names) follows a uniform distribution from 10 to 20. As another example, `InfMix(Normal(0,5),2,Uniform(10,20))` defines a random variable that comes from a normal distribution with mean zero and standard deviation varying uniformly from 10 to 20. In general, the format of this distribution is:

```
InfMix(ParentDist(Parms),MixParm,ParmDist(Parms))
```

where ParentDist is a distribution, MixParm is an integer indicating whether the first, second, ..., parameter of the ParentDist varies randomly, and ParmDist is the distribution of that parameter.

InfMix may be used recursively. For example,

```
InfMix(InfMix(Normal(0,5),1,Uniform(0,2)),2,Uniform(4,6))
```

defines a normal distribution in which the mean is uniform(0,2) and the standard deviation is uniform(4,6).

Limitations: (1) At present, computations of the upper and lower bounds of InfMix distributions assume that the largest and smallest values of the random variable are obtained when the underlying ParmDist is at its two extremes. (2) Extreme caution is needed with these distributions because problems often arise in numerical integration. I have found it helpful to increase the IntegralMinSteps to 10, which was enough in most of the cases I've looked at, but you may need to adjust this up (for precision) or down (for speed) in your cases.

Truncated(RV,Min,Max) A truncated distribution is a conditional distribution, conditioning on the random variable RV falling within the interval from Min to Max. For example, `Truncated(Normal(0,1),-1,1)` defines a random variable that is always between -1 and 1, and which within that interval has relative probabilities defined by the PDF of the standard normal. In general, the format of this distribution is:

```
Truncated(BasisDistribution(Parms),Min,Max)
```

It is sometimes convenient to specify the truncation boundaries in terms the probabilities you want to cut off rather than the scores themselves. For example, you might want to look at the middle 90% of a normal distribution but might not immediately know which scores cut off the top and bottom 5%. For this reason, there is a variant of the command that takes probabilities instead of values for min and max, like this:

```
TruncatedP(BasisDistribution(Parms),0.05,0.95)
```

With `TruncatedP`, RandGen will use its InverseCDF function to find the score values that correspond to the cumulative probabilities that you specify, and then truncate at those score values.

Bounded(RV,Min,Max) *I do not know if this is a standard type of distribution or not, and would appreciate any comments on it from those in the know.* A bounded distribution is similar to a truncated distribution in that the random variable must fall within the range of Min to Max. The difference is that all values less than Min are converted to Min, and all values less than Max are converted to Max. Thus, there are discrete masses of probability at Min and Max, and the probability density function between Min and Max is not conditionalized.

For example, consider the distribution `Bounded(Normal(0,1),-1,1)`. This is really a mixture of these three distributions:

Distribution	Mixture Probability
Constant(-1)	0.1587
Truncated(Normal(0,1),-1,1)	0.6826
Constant(1)	0.1587

Note that 0.1587 is the probability that a normal(0,1) score is less than -1, and also the probability that it is greater than 1. Bounding the distribution thus means taking all of the probability density higher than the upper value and massing it at that value.

As with the truncated distribution, there is a form of the Bounded distribution based on probabilities, as in:

```
BoundedP(Normal(0,1),0.1,0.9)
```

.

Order(k,RV1,RV2,RV2,...,RVn) The distribution of this order statistic is the distribution of the k'th largest observation in a sample of n independent observations from the n indicated random variables. For example,

```
Order(2,Normal(0,1),Uniform(0,1),Exponential(1))
```

defines a random variable that is the median (2nd largest) in a sample containing one score from the standard normal, one from the uniform from 0–1, and one from the exponential with rate 1. In general, the format of this distribution is:

```
Order(k,BasisDist1(Parms),...,BasisDistN(Parms))
```

In the special case where the basis distributions are all identical, it is more convenient to use the OrderIID distribution, described next.

OrderIID(k, n,RV) This is the special case of the order distribution in which the basis distributions are identical as well as independent. In general, the format of this distribution is:

`OrderIID(`k`,`N`,BasisDist(Parms))`

It is only necessary to specify the basis distribution once, since all are identical; instead, you have to specify how many there are (N).

OrdExp(i, n, λ**)** This is the special case of OrderIID in which the basis distribution is an exponential with rate λ, and you want the i'th order statistic in a sample of n ($1 \leq i \leq n$). For this case there are nice fast closed forms for the mean and variance that were given to me by Rolf Ulrich, Dept. of Psychology, Univ. of Wuppertal, Germany.

OrdBinary(i,n1,RV1,n2,RV2) This is an order distribution with two types of underlying RVs. For example,

`OrdBinary(2,5,Normal(0,1),7,Uniform(0,1))`

specifies the distribution of the second order statistic in samples of 12 made up of five standard normals and seven standard uniforms.

MinBound(RV1,RV2) Consider two arbitrary random variables X and Y, which may or may not be independent, and let $Z \equiv \min(X, Y)$. The CDFs of these three random variables must obey the inequality

$$F_z(t) \leq F_x(t) + F_y(t) \quad \text{for all } t$$

because

$$F_z(t) = F_x(t) + F_y(t) - \Pr(X \leq t \& Y \leq t)$$

Thus, for any two basis RVs X and Y, we can construct the random variable Z which is a lower bound on the distribution of $\min(X, Y)$:

$$F_z(t) = \begin{cases} F_x(t) + F_y(t) & \text{if } F_x(t) + F_y(t) < 1 \\ 1 & \text{if } F_x(t) + F_y(t) \geq 1 \end{cases}$$

MinBound implements this lower bound distribution for any two arbitrary random variables X and Y.

Because distributions are constructed recursively, it is legal within RandGen to construct weird distributions by any combination of the above. For example, this would be legal:

`Truncated(Mixture(.5,Normal(0,1),.5,OrderIID(4,5,Normal(0,1))),-1,1)`

and it indicates a truncated mixture of a normal distribution and an order statistic.

It does not appear to me that there will ever be any ambiguity about what distribution is requested within the syntax of RandGen, but let me know if you find such a case!

5.5 Bin-Based Distributions

RandGen has several bin-based distributions that can be used to construct arbitrary distributions and model any data pattern you like (e.g., ones estimated by tabulating lots of data). Each distribution is made up of NBins adjacent, non-overlapping, equal-width bins, with an arbitary probability of occurrence within its bin. The different bin-based distributions differ in their assumptions about the details of the distribution within each bin, as described below.

Histogram The histogram is a continuous distribution with a flat PDF within each bin.

Polygon The polygon is a continuous distribution with a linear but not necessarily flat PDF within each bin. More specifically, the PDF of the polygon distribution is defined by a series of NBins+1 points; the first point gives the height of the PDF at the lower bound of the distribution, and the remaining NBins points give the heights of the PDF at the top of each bin (the top of the top bin showing the PDF at the upper bound of the distribution). Within each bin, the PDF is a straight line going from the height at the bottom of the bin to the height at the top of the bin.

FreqPolygon The frequency polygon is also a continuous distribution with a linear but not necessarily flat PDF within each bin. Unlike the polygon, though, it keys on the PDF in the middle of each bin rather than the upper edge. More specifically, the PDF of this distribution is defined by a series of NBins+2 points; the first point gives the height of the PDF at the lower bound of the distribution, the last point gives the height of the PDF at the upper bound of the distribution, and the remaining NBins points give the heights of the PDF at the center of each bin. Within each bin, the PDF is formed by straight lines going from the height at the bin's center to the heights at the centers of the adjacent bins.

BinCen The "BinCenters" random variable is a discrete distribution with all of the probability mass associated with each bin assigned to the midpoint of the bin.

The commands `Histogram(FileName)`, `FreqPolygon(FileName)`, `Polygon(FileName)`, and `BinCen(FileNa` tell RandGen to read the description of the indicated distribution from the indicated file. The first line in the file must contain three numbers:

1. The minimum value in the distribution (i.e., the lower edge of the lowest bin).
2. The maximum value in the distribution (i.e., the upper edge of the highest bin).
3. The number of bins, NBins.

For example, this line might be

```
-10 100 200
```

to indicate a distribution ranging from -10 to 100, with 200 bins. (Note that in this example each bin would be $[100-(-10)]/200 = 0.55$ units wide.) Following the first line, there should be NBins+1 additional lines with one number per line. The first line is special: It should be zero for the Histogram and BinCen distributions, but for the Polygon distribution it should be the height of the PDF at the lower bound of the distribution (which may be zero but need not be). The remaining numbers correspond to the probabilities for the successive NBins bins, from smallest to largest. Note that these numbers need not actually *equal* the bin probabilities; it is sufficient for them to *be proportional to* the bin probabilities. RandGen automatically rescales them so that the total probability sums to one, so you could input frequency counts or PDF heights instead of actual bin probabilities.

5.6 Approximation Distributions

The above bin-based distributions can also be used as "approximation distributions," the purpose of which is to speed up computations with complicated underlying "basis" distributions. These approximations are particularly useful when (a) you are interested in a basis distribution for which it is time-consuming to compute values, and (b) you want to compute lots of different values from this distribution without changing its parameters. In these cases, initializing the approximation distribution will be a little slower than initializing the basis distribution, but then all further computations will be much faster with the approximation.

Some terminology and notation is used in common across all approximation distributions. Each approximation uses "bins", which are small, nonoverlapping ranges of the dependent variable. For example, a beta distribution is defined over the range from 0.0 to 1.0, and it might be approximated using 100 bins: 0.00–0.01, 0.01–0.02, ..., 0.99–1.00. The number of bins (100 in this example) will be referred to as "NBins," and the width of each bin will be referred to as "W." Of course, the approximation are slower to compute but more accurate with a larger number of bins (smaller W). I find that 200–300 bins is usually enough, and that with approximately symmetric distributions it is generally better to use an odd number of bins.

In practice, it may be somewhat tricky to decide which is the best approximation to use with a given basis distribution. I know of no sure strategy other than trial and error, but offer some comments on the different approximations based on my limited experience with them.

ApprPolygon(RV) This is a continuous approximation that can be used only for a continuous basis distribution, RV. In brief, the PDF of the approximation distribution is a set of NBins straight lines, matched to the height of the basis distribution's PDF at the bin's edges (further detail is given below). For example,

```
ApprPolygon(Convolution(Normal(0,1),Beta(2,2)),201)
```

approximates the specified convolution with a set of 201 straight lines.

ApprFreqPolygon(RV) This is a continuous approximation that can be used only for a continuous basis distribution, RV. In brief, the PDF of the approximation distribution is a set of NBins straight lines, matched to the height of the basis distribution's PDF at the bin's centers. For example,

```
ApprFreqPolygon(Convolution(Normal(0,1),Beta(2,2)),201)
```

approximates the specified convolution with a set of 201 straight lines.

This is my preferred type of approximation. It is generally quite accurate, and it is often much faster than the other approximations.

Details of construction.

Step 1: The first line starts at X_1=minimum (of the basis distribution) with height PDF at that point and goes to X_2=minimum+W/2 with height PDF=Basis.PDF(X_2). The second line continues from the end of the first line to the point with X_3=minimum+1.5*W and height PDF=Basis.PDF(X_3). And so on, with the final line segment ending at the maximum of the basis distribution and PDF at the maximum.

Step 2: The PDF just constructed is integrated, and the heights are scaled up or down appropriately so that the total area is 1.00.

ApprHistogram(RV) This is a continuous approximation that can be used for either a discrete or a continuous basis distribution, RV. In brief, the PDF is of the approximation distribution is a set of NBins flat lines, as if the basis distribution were uniform within each bin (like in a histogram). For example,

```
ApprHistogram(Convolution(Normal(0,1),Beta(2,2)),201)
```

approximates the specified convolution with a set of 201 bins with equal probability within each bin.

This approximation is more general than ApprPolygon, because it can be used with discrete distributions, and it is less sensitive to abruptly-changing PDFs. But it is usually slower to construct initially, and it is often much slower to do any computations with. The PDF has discontinuities at the bin boundaries (unlike ApprPolygon), and these make numerical integrations converge more slowly.

Details of construction. The CDF of the basis distribution is computed at the top and bottom of each bin, and from these the bin probability is computed. Then, the height of the uniform approximation PDF within that bin is adjusted to give this bin probability.

ApprBinCen(RV) This is a discrete approximation, and it can be used to approximate either a discrete or a continuous basis distribution, RV. In brief, the approximation assumes that all of the probability mass is concentrated in a single point at the center point of each bin; moreover, any value in the bin is treated as if it were that center point.

Details of construction. The CDF of the basis distribution is computed at the top and bottom of each bin, and from these the bin probability is computed. All of this probability mass is assigned to the value at the center of the bin. For purposes of PDF and CDF computations, all values within a bin are treated as equivalent to the center.

Using CUPID, approximations can be written out to files and read in from files. This is useful for saving the time-consuming initialization phase if you want to return to an approximation later, and it also allows you to prepare your own approximation distributions separately and then import them into RandGen for further analysis.

To write the current approximation from within CUPID, use the command `BinsWrite(FileName)`. To read the approximation back in, use the command `BinsRead(FileName)`. The format of the approximation file is just the same as that for the bin-based distributions described in the previous section, with one exception: The very first line of the file contains the name of the basis distribution.

5.7 Construction Distributions

Given the approximation distributions described in the previous section, it seemed natural to include another type of approximation distribution that I call "construction distributions." These are approximation distributions constructed simply by tabulating many calls to the random number generator for any arbitrary distribution. I have found them useful mostly in my own simulation work, where I know how to write an RNG for the variable that I am interested in but I don't really know anything else about how to characterize it. In these situations, I have found it convenient to construct an approximation to the true distribution by calling the RNG many times and tabulating the results. The construction distributions simply automate this process. I imagine that they will be valuable only to programmers who can compile their own RNG source code using these routines, but I describe them here for completeness.

There are three types of construction distributions:

ConsHistogramRV A constructed HistogramRV used to approximate a basis distribution.

ConsFreqPolygonRV A constructed FreqPolygonRV used to approximate a basis distribution.

ConsBinCenRV A constructed : BinCenRV used to approximate a basis distribution.

Note that no constructed PolygonRV exists.

5.8 Distributions Arising in Connection with Signal Detection Theory

In addition to the above standard and derived distributions, I have added a few distributions that corresponded to particular projects I happened to be working on. The distributions described in this section arise in connection with signal detection theory experiments, and will be of interest to some psychophysicists and perhaps engineers. If you don't know what signal detection theory is, then it is unlikely that you will care about these. Note: These are all discrete distributions, as each reflects the outcome of one or two binomial-type conditions with a finite number of trials.

ZfromP(SampleSize,TrueP,Adjust) This is the discrete distribution of Z, which is derived from the binomial distribution as follows:

1. For any sample from a Binomial(N, P), convert the number of successes k to the probability of success, $p \equiv k/N$. If $p = 0$, set $p =$ Adjust$/N$; if $p = 1$, set $p = 1 -$ Adjust$/N$. "Adjust" is a parameter between 0 and 1, specified by the user, to indicate how the extreme data values should be treated.
2. Find Z such that $p = \Pr(z \leq Z)$, where z is a random variable having the standard normal distribution.

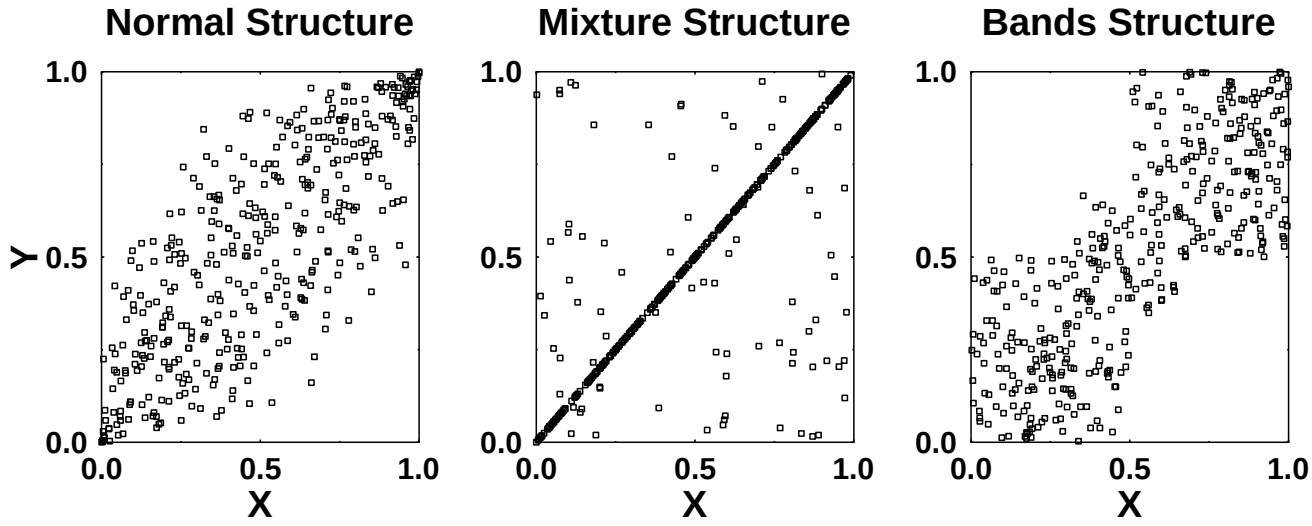
APrime(NSignalTrials,PrHit,NNoiseTrials,PrFalseAlarm) This is the distribution of the sample A' computed from an experiment with NSignalTrials signal trials each having the specified true probability of a hit, and NNoiseTrials noise trials each having the specified true probability of a false alarm. Specifically, A' is the distribution-free estimate of the area under the ROC curve computed using Equations 2 and 9 of Aaronson and Watts (1987).

APrimeSym(NTrials,PC) This is a shortcut for the previous distribution that can be used when there are equal numbers of signal and noise trials and when the probability of a correct response (hit or correct rejection) is the same for both signal and noise trials.

YNdPrime(NSignalTrials,PrHit,NNoiseTrials,PrFalseAlarm,Adjust) This is the distribution of the sample $\hat{d}'$ computed from an experiment with NSignalTrials signal trials each having the specified true probability of a hit, NNoiseTrials noise trials each having the specified true probability of a false alarm, and using the Adjust factor (between 0 and 1) to correct cases with 0% or 100% hits or false alarms (e.g., replace 0 hits with Adjust hits, and replace NSignalTrials hits with [NSignalTrials - Adjust] hits). Programming note: If PDFs are requested, this distribution is implemented using the List (smaller samples) and AppApprCen (larger samples) random variables.

YNdPrimeSym(NTrials,TrueDP,Adjust) This is the special case of YNdPrime in which NSignalTrials = NNoiseTrials and Pr(Hit) = 1 - Pr(FA). Note that the second parameter is the true d' rather than the hit probability.

Figure 1: Three bivariate random samples generated using the three available bivariate structures. In each panel, the marginal distributions of X and Y are uniform(0,1), and the underlying true correlation of X and Y was 0.8.



6 Overview of Bivariate Probability Distributions

This section provides a brief overview of bivariate probability distributions, and it can probably be skipped by readers already familiar with this topic. The next section describes the three bivariate structures available within this program. The one available multivariate (3+ variables) structure is the normal, and it is in every way analogous to the bivariate normal structure described here. If you get lost reading the following section, it might be helpful to come back and read this one. If you would like further information, you might also consult Bradley and Fleisher (1994) or Miller (1998)—some relatively non-technical references about generating correlated random numbers with non-normal distributions.

A bivariate probability distribution can be thought of as a set of possible (X, Y) pairs with an associated probability of each pair. The bivariate probability distribution uniquely determines both the marginal distributions of X and Y (i.e., the univariate distribution of each one ignoring the other) and the correlation of X and Y. The reverse is not necessarily true, however; given a pair of desired marginal distributions for X and Y and a desired value of the XY correlation, you can't always solve for the underlying bivariate probability distribution. There may be infinitely many bivariate distributions consistent with those marginals and that correlation.

This leads to a problem: Suppose you want to run some simulations with given marginal distributions and see how the model behaves when the random variables are correlated (say at a correlation of -0.2, which seems intuitively plausible to you). The problem is that (for most marginal distributions) you have not yet uniquely specified what you want to do, because there are infinitely many bivariate distributions consistent with your constraints.

This program allows you to choose among three different bivariate structures, described in the next section. Each one gives you the marginals you request, but uses a different technique to give you the correlation you want. With luck, one of the available structures may seem to capture, at least approximately, the sources of correlation that you imagine are operating in your situation.

7 Available Bivariate/Multivariate Structures

This program implements three different bivariate structures, and these are illustrated with the random samples displayed in Figure 1. I would be happy to have suggestions for other general structures to add. Only the normal structure can be used when generating three or more variables (note that this does *not* mean that the marginals must be normal).

To define these structures more formally, let F_x and F_y be the cumulative marginal distributions of X and Y, respectively.

7.1 Normal

One bivariate structure is a transformation of the bivariate normal distribution. With this structure, each (X, Y) pair is generated as follows:

1. A pair (Z_x, Z_y) is generated randomly from a bivariate normal distribution with marginal means of zero, marginal standard deviations of one, and a prespecified correlation ρ.
2. The pair (P_x, P_y) is computed, where P_x and P_y are the cumulative probabilities in the standard normal distribution associated with Z_x and Z_y, respectively.
3. X is taken as $F_x^{-1}(P_x)$, and Y is taken as $F_y^{-1}(P_y)$.

It can be seen that this method does indeed produce the desired marginals for X and Y; P_x and P_y are both uniformly distributed between zero and one by construction, so the different values of X and Y occur with the desired marginal probabilities. Moreover, the correlation of X and Y will have the same sign as ρ, although it will not necessarily have the same magnitude. As described below, the value of ρ can be adjusted to obtain the desired numerical value for the correlation between X and Y, as described in section 7.5.

The normal structure can be generalized to three or more random variables, and this technique has been discussed some in the literature on operations research (e.g., Cario & Nelson, 1997; Ghosh & Henderson, 2002). In that literature, it is referred to as the NORTA ("normal to anything" method). Unfortunately, the NORTA method does not always succeed. That is, sometimes it does not find an appropriate correlation matrix even though there does exist a multivariate distribution with the desired properties. For further information, see Ghosh and Henderson (2002).

7.2 Mixture

A second bivariate structure uses a mixture of maximally correlated pairs and uncorrelated pairs[4]. With this structure, X is randomly generated from its marginal distribution. Then, with a preselected probability p, Y is taken as $F_y^{-1}(F_x(X))$ (or, if a negative correlation is desired, as $F_y^{-1}(1 - F_x(X))$). With probability $1 - p$, however, Y is chosen from its marginal distribution independently of X.

Again, the method produces the correct marginals. X is chosen directly from its desired marginal distribution, and Y is simply a mixture of two different cases, each of which has the desired marginal distribution for Y. The strength of the correlation between X and Y is controlled by the value of the mixture probability p, which may be adjusted to obtain the desired correlation, as described in section 7.5.

7.3 Bands

The third bivariate structure divides each marginal probability distribution into percentile regions or "bands." With four bands, for example, each distribution is divided into bands from 0-25%, 25-50%, 50-75%, and 75-100%. X is generated randomly from its marginal distribution, and the program determines which band X came from. Then Y is generated randomly from the same band if a positive correlation is desired, and from the complementary band for a negative correlation (i.e., if one band goes from a to b%, then the complementary band goes from $(100 - b)$ to $(100 - a)$%). This structure also yields the desired marginals. X is generated directly from its marginal, and Y is generated from one of a set of equally likely bands within its marginal. The strength of the correlation between X and Y increases with the number of bands, which can be adjusted to give the desired correlation, as described in section 7.5.

A limitation of the band structure is that with an integral number of bands it may not be possible to produce a desired correlation. With normal and exponential marginals, for example, the correlation is 0.555 with two bands and 0.697 with three bands, so an intermediate correlation could not be produced with an integral number of bands. To overcome this limitation, the band technique was generalized to use a mixture of different numbers of bands to

[4]This structure was suggested to me by Dr. Ellen Hertz of the National Highway Traffic Safety Administration, United States Department of Transportation.

generate different (X, Y) pairs. For example, if two bands are used with probability 0.6 and three bands are used with probability 0.4, then the overall correlation is 0.612. In fact, any intermediate correlation can be produced by adjusting this probability. By convention, then, the program allows the number of bands to be a real number rather than an integer, and it uses the real part to determine the probabilities of the smaller versus larger number of bands. The real number can be thought of as the expected number of bands: 4.3 bands, for example, means there are 4 bands with probability 0.7 and 5 bands with probability 0.3.

7.4 RhoController

In the rest of this documentation, it is convenient to have a generic name for the parameter controlling strength of the correlation, and I will use the term "RhoController" for this purpose. The meaning of RhoController depends on the bivariate structure, as follows:

Structure	Meaning of RhoController
Normal	Correlation ρ between underlying normal random variables.
Mixture	Probability p of generating a perfectly correlated pair.
Bands	The number of bands.

Note that the value of RhoController is monotonically related to the XY correlation, but is not usually equal to it.

7.5 How RhoController is Adjusted

If requested to do so, RandGen will adjust the value of RhoController to try to produce a given desired correlation in the underlying bivariate distribution. For example, the user may request two underlying exponential marginals with a bivariate mixture structure, and may instruct RandGen to adjust the mixture probability to attain a true correlation of (say) 0.6 in the underlying bivariate distribution. In that case, RandGen will adjust the value of RhoController using a simple numerical search algorithm, trying to find a value of the mixture probability that produces the desired bivariate correlation of 0.6.

During the numerical search, RandGen computes the true correlation produced by each candidate value of RhoController using an $N \times N$ grid approximation of the bivariate distribution, where N is the user-specified number of steps used to approximate each distribution. If the user specifies that each distribution should be approximated by 100 points, for example, RandGen uses 100 equally-spaced percentile points of the X distribution (i.e., at percentiles of 0.5%, 1.5%, ..., 99.5%). For each of these points, it computes 100 equally-spaced percentile points of the Y distribution conditional on that value of X. In total, then, it computes 100×100 (X, Y) pairs, and computes the numerical value of the correlation as if these were all the possible pairs in the bivariate distribution and as if they were all equally likely. This is usually quite a good estimate of the true bivariate correlation obtained with that value of RhoController, as long as the grid approximation is OK.

8 Technical Notes

8.1 Installation

No special installation is required. You can simply run RandGen in the directory to which you unzipped it, or copy RandGen.EXE to any directory in your DOS path.

8.2 Underlying RNG

The basic random number generator underlying all of the distributions is the uniform generator known as the Mersenne Twister, as described by Matsumoto and Nishimura (1998).

8.3 Checking RandGen

With any new desired distribution, it is a good idea to check RandGen by generating a large sample (say 1,000,000) and checking the summary output file. The chi-square tests of the marginals should give reasonable values (p's not too

small), and the observed correlation should be close to the desired true one. If you find a problem, please e-mail me the input file.

8.4 Correlations involving Discrete Distributions

The methods described in section 7 assume that X and Y are continuous distributions. With discrete distributions, RandGen uses the very same computational methods, involving F_x, F_y, F_x^{-1}, and F_y^{-1}, as it does for continuous ones. This means that for discrete distributions the RhoController adjustment process may give a poorer approximation to the desired "true" correlations than it does with continuous distributions. You should therefore be especially careful to check the "true" correlation when using one or more discrete distributions.

8.5 Interfacing RandGen to a Simulation Program

The main problem in using RandGen to generate random numbers for one of your own simulation programs is to give your program access to the random numbers. I know of only two solutions to this problem; neither is particularly fast in terms of computer time, but they may save the all-important programmer time, compared to coding your own random-number generators from scratch.

The simpler solution is to use RandGen to generate a large disk file of random numbers from the desired distribution, and then start up your program and let it read the random numbers from the file. This solution works well enough provided that (a) the desired distributions can be specified before you start running your simulation program, and (b) your program needs few enough random numbers that you can conveniently store them all in a disk file.

The more complex solution is not subject to either of these constraints, but requires a little more programming effort. To use this solution, your program must invoke RandGen as an external program. In Pascal, for example, the syntax for this would be something like:

```
Exec('RandGen.EXE','RootFileName');
```

This would start RandGen with the command-line parameter in the second string. (All the programming languages I know of allow you to start an external program from within your own program, so I assume this is generally possible.) Of course your program would have to have already written the control file called RootFileName.In, with the desired specifications; this should not be too difficult, however, since it is a pretty straightforward ASCII file. Using this approach, the idea is to generate an appropriate batch of random numbers to a file RootFileName.Dat, read the random numbers from that file, generate a new batch, and so on. Since you can write the input file within your simulation program, you can generate the random numbers from whatever distribution the program selects at run-time, and of course you can use different distributions on successive calls to RandGen. And you can generate random numbers in batches of any convenient size. Of course, when using this strategy your program has to keep track of how many numbers it has read from the Dat file, so that it will know when to generate another batch.

A useful trick is available if you use the more complex solution. By default, RandGen generally asks for user confirmation before writing over an existing file (except it will overwrite the *.Dat output files without asking when NFiles is greater than 1). That means that your program would have to delete the old output files before RandGen will generate a new batch of random numbers, or else you would have to babysit the simulations and provide user confirmation each time a new batch was generated. The trick is to use an exclamation point as the first character of the root file name. If you do that, RandGen will (a) delete the exclamation point from the input and output file names, and (b) write the output to a file with the indicated name, *automatically overwriting the file if it already exists*.

8.6 Optimizing Speed

Using RandGen will add a little overhead to your simulation, as compared with your programming the desired random number generator yourself. The overhead arises because you must invoke RandGen as a separate program, and because it writes the random numbers to a file, from which your simulation program must read them in again. Despite the overhead, I believe that RandGen will be useful to a number of people, for three reasons: (a) the overhead will often be a negligible proportion of the total computing needed for the simulation, and (b) it will be easier to write and debug the simulation program in the first place if this random number generator toolbox is used, and (c) computer time is cheap and getting cheaper.

In any case, there are some strategies for minimizing the overhead with RandGen. First, it is worth your while to try different orders for your marginal distributions. With two distributions, for examples, you have marginals for X and Y. These two distributions are used slightly differently within the program (see descriptions of computational methods for the three bivariate structures), and so sometimes the speed of random number generation varies depending on which variable you call X and which you call Y. For example, generating random uniform numbers and random values of Pearson's r using the mixture structure, it is faster to let X be the Pearson and Y be the uniform than the reverse.

A second speed issue arises in connection with the search process described in section 7.5. It is easy to see that the time needed for searching increases with the square of the value of the "number of steps" approximation parameter described in section 2.12. In practice, I have usually found that 100 steps is adequate to give quite a good approximation, but nonetheless I usually use 200 or even 400 steps to err on the side of slowness and accuracy.

If you are interested in search speed, you should play around a bit with the "number of steps" parameter. For example, do a first run with 50 steps, then others with 100, 200, and so on. For each run, note the value of RhoController that the search process converges on, holding fixed the target correlation, of course. As you increase the number of steps, RhoController should stabilize to a relatively constant value; you can use the minimum number of steps yielding this constant asymptotic value. My general impression is that you need a lot of steps only with distributions that have really extreme tails, but I have not looked closely at this question.

A third issue is that for really complicated distributions it may be faster to use one of the approximation distributions (see CUPID documentation). The approximation distributions take more time to set up, but then they can be much faster to use, especially when you are generating correlated random numbers.

8.7 Numerical Approximations

One key point is that all distributions are represented numerically, with finite limits. RandGen's version of the standard normal distribution, for example, goes from about -5.6 to 5.6, not from $-\infty$ to ∞. Similarly, there are numerical bounds for all distributions (you can find out what bounds RandGen is using by running CUPID and using the functions `minimum` and `maximum`). In addition, RandGen sometimes has to change bounds of naturally bounded distributions in order to avoid numerical errors. Gamma distributions, for example, start at 0.00001 instead of 0.0, because the Gamma PDF cannot be evaluated at 0.0.

Although it is very general, RandGen is not always very accurate. Many values are obtained through numerical integration, and the results can be substantially off in some pathological cases, due to the vagaries of numerical approximations with finite-precision math. The moral of the story is that you should check the values that you care most about. One good check is to make very minor changes in parameter values and make sure that the results change only slightly.

9 Release History

Version 1.0, called Bivar, was released on a limited basis in April 1997.
Version 1.1, called Bivar, was released in August 1997, with improved documentation, a revised interface, and some bug fixes.
Version 1.2, called RandGen, was released in March 2000 with the ability to generate more than two variables using the normal bivariate structure.

10 Registration and Author Contact Address

I would really like to receive feedback on who is using this software, for what purposes. So, please e-mail me at miller@otago.ac.nz if you found this software useful. If you do, I will add your name to my mailing list and let you know about any new versions, bugs, or new programs that might interest you. If you use this software for any published research, I would greatly appreciate it if you would acknowledge the software in your article (e.g., in a footnote) and email me a citation to the article or, better yet, send me a reprint. Of course I would also welcome bug reports and suggestions for improvement, too, although I can't promise any fast action on those.

Here is how to contact me:

Prof Jeff Miller
Department of Psychology
University of Otago
Dunedin, New Zealand
email: miller@otago.ac.nz
FAX: (64-3)-479-8335

11 OS/2 Versions

I will happily provide executable OS/2 versions of this software to others who still use this operating system, as I do myself. The only difference from the versions described in this documentation is that the "SET CUPID=C:\CUPID" command (if needed) goes in the config.sys file rather than the autoexec.bat file. (Don't forget you have to shutdown and reboot for this to take effect.)

12 Acknowledgements

I have obtained information about probability distributions from a variety of sources, including the internet. The most important are the references listed below. Rolf Ulrich has also supplied quite a lot of information about various probability distributions and numerical methods. Thanks also to Roman Krejci, from whom I got the pascal code for the Mersenne Twister. Others who have helped, directly or indirectly, include Timo Salmi and Duncan Murdoch. Special credit goes to Alann Lopes, who showed me how object-oriented programming could be useful in the first place.

13 References

References

Aaronson, D., & Watts, B. (1987). Extensions of Grier's computational formulas for A' and B" to below-chance performance. *Psychological Bulletin, 102,* 439–442.

D'Agostino, R. B. (1970). Simple compact portable test of normality: Geary's test revisited. *Psychological Bulletin, 74,* 138–140.

Dallal, G. E., & Wilkinson, L. (1986). An analytic approximation to the distribution of Lilliefors's test statistic for normality. *The American Statistician, 40,* 294–296.

Devroye, L. (1986). *Non-uniform random variate generation.* Berlin: Springer-Verlag.

Finney, D. J. (1978). *Statistical method in biological assay.* London: Charles Griffin.

Gescheider, G. A. (1997). *Psychophysics: The fundamentals. [3rd ed.].* Hillsdale, NJ: Erlbaum.

Johnson, N. L., & Kotz, S. (1970). *Continuous univariate distributions.* New York: Houghton Mifflin.

Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation, 8,* 3–30.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1986). *Numerical recipes: The art of scientific computing.* Cambridge, England: Cambridge University Press.

Quick, R. F. (1974). A vector magnitude model of contrast detection. *Kybernetik, 16,* 65–67.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *Computer Journal, 3,* 175–184.

Sternberg, S., & Knoll, R. L. (1973). The perception of temporal order: Fundamental issues and a general model. In S. Kornblum (Ed.), *Attention and performance IV* (pp. 629–685). New York: Academic Press.

Strasburger, H. (2001). Converting between measures of slope of the psychometric function. *Perception & Psychophysics, 63,* 1348–1355.

14 Software License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

14.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

14.2 Terms of License

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS