CUPID: A Program for Computations with Univariate Probability Distributions

Version 2.13

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

Jan, 2006

If you use this software, please register it by sending me your email address, as described in section 18. There is no charge to register, and your email address will be used only to notify you of program bugs or updates.

Contents

Part I

Introduction

1 Overview of the Package

CUPID is a package for working with probability distributions. It was developed primarily for stochastic simulation and modeling, although it can also be useful in data analysis.

2 Introduction

CUPID is useful for doing computations with univariate probability distributions.

- Here are the kinds of things it does:

  Computation: It computes numerical values of many quantities associated with probability distributions, including probability densities, cumulative probabilities, inverse cumulative probabilities, mean, variance, skewness, etc. Such computations are often needed to determine the quantitative predictions of stochastic models, and CUPID can often be used instead of a special-purpose program to do such computations. In addition, because cumulative and inverse cumulative probabilities are available, the program provides a handy set of on-line statistical tables for researchers doing statistical hypothesis testing (i.e., what is the critical F value with 7 and 143 degrees of freedom? what is the p level associated with this F value? etc.).

  Random Number Generation: CUPID generates random numbers from any desired distribution. This can be an easy way to generate random numbers for computer simulations (but see the program "RandGen", based on CUPID, for a more power random-number generation tool).

  Estimation: CUPID can often find the best estimates of the parameters of a distribution, with "best" defined in terms of either (a) maximizing the likelihood of a given set of observations, (b) matching certain observed percentile values, (c) matching certain observed moments, (d) minimizing the chi-square fit to a set of observed bin proportions, (e) maximizing the likelihood of a set of probit-type data, or (f) minimizing the chi-square fit to a set of probit-type data.

  Graphing: CUPID can make some fairly crude plots of the PDF, CDF, MGF, and Hazard functions. In addition, for users wanting more sophisticated graphs or graphs that can be printed, it has a "Columns" command to generate columns of X and Y values that can be used as input to your own graphing program.

- Here are some of the kinds of distributions it knows about:

  Standard distributions: CUPID knows about many standard distributions (e.g., Normal or Gaussian, Uniform, Gamma, F, t, . . .), which I refer to as "primitive" distributions. The program also knows how to work with various kinds of distributions that can be derived from these primitive distributions, as indicated by the remaining distributions in this list.

  Transformations: A very simple derived distribution is a linear transformation of a primitive one (e.g., an exponential plus a constant). More complicated transformations include the exponential transform (e.g., $Y = e^X$, where X has one of the primitive distributions), the natural log ($Y = \ln(X)$ and inverse ($1/X$) transformations, and power function transformations. More complicated transformations can be constructed by combining several of these.

Truncated distributions: New random variables may be defined by restricting other variables to part of their usual range (e.g., a standard normal restricted to the range from -2 to +2).

Mixture distributions: Random variables may be formed by randomly selecting one of a set of independent random variables (e.g., an RV that is a standard normal with probability p & a uniform from -2 to +2 with probability $1 - p$).

Infinite mixture distributions: Random variables may be formed by letting one of their parameters vary randomly according to some (usually different) distribution. For example, Y might be distributed following a normal distribution with $\mu = X$ and $\sigma = 1$, where X follows a uniform distribution between 0 and 1.

Convolutions: Random variables may be formed by summing values of independent random variables (e.g., sum of independent standard normal and uniform from -2 to +2).

Order distributions: Random variables may be formed by choosing the i'th order statistic from a sample of n independent random variables (e.g., the minimum of 10 standard normals).

Approximation distributions: For complicated distributions that require slow numerical integration, CUPID has some approximations available to speed things up.

Complete lists of the primitive distributions and functions implemented so far can be found in sections 5 and 6, respectively, and the possibility of adding new distributions and functions to the program is discussed in Section 15.

3 Installation Instructions

1. Unzip the distribution file CUPID.ZIP.
2. Open the file 00Readme.txt and follow the instructions in it for a brief demo.
3. If you want to be able to run the programs from a command window opened within any directory, move all of the unzipped EXE files to a directory in your path.

4 User Interface

CUPID is command-driven, and all commands are entered at the program's prompt (>). Commands that look like distribution names simply tell the program which distribution you want to use for the next set of computations. For example, typing `Uniform(0,100)` in response to the program prompt tells the program that, until further notice, it should use a Uniform distribution ranging from 0 to 100. Here is a more elaborate example:

```
Mixture(0.95,Normal(0,1),0.05,Normal(1,1))
```

This command instructs CUPID to work with a distribution that is a mixture of two normals—a standard normal with probability 0.95 and a normal with $\mu = \sigma = 1$ with probability 0.05.

Other commands tell the program what quantities to compute from the distribution currently in use. For example, typing `SD` tells the program to print out the standard deviation of the distribution currently in use. There are also a number of commands that control the program (e.g., opening an output file for results) rather than doing any statistical computation (see Section 10).

The program also keeps a history buffer of commands issued previously. You can scroll through previous commands with the up and down arrows, and you can edit these commands with left and right arrows, backspace, delete, etc.

Furthermore, you do not need to type the full names of distributions, functions, and commands. It is sufficient to type enough letters at the beginning of the string to uniquely specify what you want. The following two input lines are equivalent, for example:

```
Mixture(0.95,Normal(0,1),0.05,Normal(1,1))

Mix(0.95,Nor(0,1),0.05,Nor(1,1))
```

because “Mixture” is the only recognized string starting with “Mix” and “Normal” is the only string starting with “Nor.” Similarly, “logn” is a legal abbreviation for “lognormal”, and so on. If you type a too-short string that is ambiguous (e.g., “exg” could be “exgaussian” or “exgaussrat”), CUPID will reject it as an unrecognized command.

While perhaps not as nice for the novice user as a friendly menu system, the command-driven approach has the advantage that the program can also take commands from ASCII files (see section 10.20). or from the command line. All commands operate the same way whether they are typed in by the user, read in from files or entered as command-line parameters when the program is invoked, like this:

```
C:> CUPID Outfile(test) Convolution(Exponential(1.5),Exponential(1))
        Repeat(xxx,.01,.01,5)CDF(xxx)
```

Obviously, for slow and/or repetitive computations it is often very convenient to let the commands come from a file prepared in advance, while the user goes home to visit the family. (Note that an outfile is specified first so that the results will be saved on the disk as well as written to the screen.) Another advantage, for programmers, is that it is possible to execute CUPID from within another program, effectively using CUPID like a subroutine, albeit a slow one.[1]

5 Distributions Available in CUPID

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 $0 \le x \le \Pi/2$, $f(x) = \cos(x)$ 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. You can change to specifying the exponential in terms of its mean instead of its rate by first issuing the command `UseExpMean`, described in Section 10.

ExGaussMn(μ, σ, μ_e) This is a reparameterization of the ExGaussian. Its parameters are the μ and σ of the Normal, and the mean of the exponential, μ_e.

---

[1] It would be even nicer to have CUPID available as a dll, but I'm not familiar with dll's so I haven't yet figured out how to do that.

ExGaussRat(μ, σ, r) This is just a reparameterization 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 $1/\lambda$. You can change to specifying Mean instead of Rate with the `UseExpMean`, described near the end of Section 10.

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. I don't know whether it has ever been considered before or 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$(\mu, \sigma, a, \lambda)$ 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. Schwarz (2001, 2002) describes the ex-Wald distribution in detail.

ExWaldMn(μ, σ, a, μ_e) This is a reparameterization of the ExWald. The first three parameters are the same as in the plain ExWald, and the fourth parameter is the mean of the exponential, μ_e, instead of the rate.

ExWaldMSM(μ, sd, μ_e) This is a further reparameterization of the ExWald. The first two parameters are the mean and standard deviation (*not* σ!) of the Wald component, and the third parameter is the mean of the exponential, μ_e. The Wald parameter a is set to 1.0.

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 k exponentials, each with rate λ. In this distribution, k must be a positive integer. In the RNGamma distribution (see below), k is any positive real. You can change to specifying Mean instead of Rate with the `UseExpMean`, described near the end of Section 10.

Geary(SampleSize) The Geary statistic arises in testing to see whether a set of observations come from a normal distribution (D'Agostino, 1970). CUPID will not compute the test for you, but it will give you significance levels of values computed by some other program.

GenErr(Mu,Scale,Shape) This is the general (a.k.a. "generalized") error distribution (e.g., Evans, Hasting, & Peacock, 1993, p. 57), also known as Subbotin's distribution (e.g., Johnson, Kotz, & Balakrishnan, 1995, 2nd Ed., Vol 2, p. 195). The correct PDF is

$$f(x) = \frac{\exp\left[-|x - \mathrm{Mu}|^{\mathrm{Shape}} / (2 \cdot \mathrm{Scale})\right]}{\mathrm{Scale}^{1/Shape} \cdot 2^{(1+1/\mathrm{Shape})} \cdot \Gamma(1 + 1/\mathrm{Shape})}$$

although both EHP and JKB give it with incorrect exponents of the Scale parameter in the denominator. This version uses the shape parameter denoted λ by Evans et al. Note that the Laplace, normal, and uniform distributions are special cases of this distribution with this shape parameter equal to 1, 2, and approaching ∞, respectively. In practice, lots of combinations of parameter values give overflow errors, especially if the shape parameter is more than approximately 3.

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

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

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

Inverse Gaussian See the Wald3 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

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

Lilliefors(SampleSize) The Lilliefors statistic arises in testing whether a set of observations come from a normal distribution (Dallal & Wilkinson, 1986). CUPID will not compute the test for you, but it will give you the critical values for the test. The approximation given by Dallal and Wilkinson is only accurate for upper tail probabilities (i.e., CDF values of 0.90 or greater), so this distribution really should really be used *only* for the computation of critical values.

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_n, σ_n) This is the distribution of X such that $\ln(X)$ is normally distributed. The parameters are the μ_n and σ_n of the underlying normal.

LogNormalMS(μ_l, σ_l) This is the distribution of X such that $\ln(X)$ is normally distributed. The parameters are the μ_l and σ_l of the overall lognormal. Comparing parameters of the two lognormal distributions,

$$\begin{aligned} \mu_l &= \exp(\mu_n) \cdot \exp(\sigma_n^2/2) \\ \sigma_l^2 &= \left[\exp(\mu_n)\right]^2 \cdot \exp(\sigma_n^2) \cdot \left[\exp(\sigma_n^2) - 1\right] \\ \sigma_n^2 &= \ln\left(\frac{\sigma_l^2}{\mu_l^2} + 1\right) \\ \mu_n &= \ln\left(\frac{\mu_l}{\exp(\sigma_l^2/2)}\right) \end{aligned}$$

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. In the actual distribution, moments above the first do not exist; they do exist in CUPID'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 $x \geq K$. This is a model for income, where K is some minimum income and $F(x)$ is the probability that a randomly selected income is less than or equal to x.

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, $T \geq 1$, 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.

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

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

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

$$f(x) = \frac{x^{k-1} \exp(-x/\lambda)}{\Gamma(k)\lambda^k} \quad \text{for } x > 0$$

where $\Gamma(k) = \int_0^\infty s^{k-1} \exp(-s) ds$.

Rosin(D_m, P) This distribution is generally known as the Rosin-Rammler, and it arises in analyses of particle sizes. Its CDF is

$$F(x) = 1 - \exp\left[-\left(\frac{x}{D_m}\right)^P\right]$$

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.

StudRng(df,NSamples) Distribution of Studentized range statistic with *df* degrees of freedom for error and $NSamples$ samples. Because both parameters are integers, automatic program-based estimation of these parameters is rarely successful.

t(df) Student's t-distribution, with degrees of freedom equal to *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 $T/2$.

VonMises(L, S) This is the Von Mises distribution with location and scale parameters L and S, respectively. It is defined on the range of 0 to 2π, and L must be in this range. Typically, the Von Mises distribution is regarded as an analog of the normal distribution *on a circle* instead of on the real number line.

Wald3(μ, σ, a) This is the general, 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. Note: By default, the sigma parameter is fixed in all parameter searches.

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.

NegativeBinomial(N, p) The distribution of the number of failures before reaching N successes in a sequence of Bernoulli trials, with probability p of success on each trial.

NeymanA(μ_1, μ_2) This is the Neyman type A distribution, with mass on the nonnegative integers. It has mean $\mu_1 \cdot \mu_2$ and variance $\mu_1 \cdot \mu_2 \cdot (1+\mu_2)$. The PDF is defined by:

$$\begin{aligned} \Pr(X=0) &= \exp\left[-\mu_1\left\{1-\exp(-\mu_2)\right\}\right] \\ \Pr(X=k) &= \frac{\mu_1}{k}e^{-\mu_2}\sum_{j=1}^{k}\mu_2^j\frac{\Pr(X=k-j)}{(j-1)!} \qquad \text{for } k>0 \end{aligned}$$

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))
```

If the constant distribution does not work where you need it, try instead a normal distribution with a really small standard deviation.

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 CUPID 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

CUPID 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 CUPID, 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

CUPID also knows about various sorts of distributions that can be derived from one or more primitive or "basis" distributions. In most cases, CUPID 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.

CUPID 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, CUPID 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 CUPID'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). CUPID 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`, CUPID 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 ($1 \le i \le n$). For this case there are nice fast closed forms for the mean and variance that were given to me by Rolf Ulrich.

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

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

because

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

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 $\min(X,Y)$:

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

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

Because distributions are constructed recursively, it is legal within CUPID 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. CUPID computes the mean and standard deviation of this distribution to be 0.1731 and 0.4994—now you know.

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

5.5 Bin-Based Distributions

CUPID 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 arbitrary 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(FileName)` tell CUPID 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. CUPID 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

CUPID can also use bin-based distributions as "approximation distributions," the purpose of which is to speed up computations with complicated underlying "basis" distributions (e.g., convolutions). 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. For example, compare the speed of

```
repeat(xxx,-1,0.01,1)CDF(xxx)
```

with these two distributions:

```
Convolution(OrderIID(1,100,normal(0,1)),Normal(1,1))
```

versus the approximation distribution (see below)

```
ApprPolygon(Convolution(OrderIID(1,100,normal(0,1)),Normal(1,1)),200)
```

5.6.1 The automatic approximation

The simplest way to request an approximation distribution is by placing an asterisk (i.e., "*") before a distribution name. This method of requesting an approximation is called the "automatic" approximation, and it signals CUPID to expand the distribution into a ApprPolygon approximation (this approximation is described below). For example, the distribution specification

```
*Convolution(OrderIID(1,100,normal(0,1)),Normal(1,1))
```

is automatically expanded into

```
ApprPolygon(Convolution(OrderIID(1,100,normal(0,1)),Normal(1,1)),2001)
```

This approximation works so well that I can recommend that you consider using it to approximate continuous distributions (e.g., to speed up convolutions) even if you don't have time to delve into the details of how it works (explained further below).

5.6.2 Bin-based approximations

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,NBins) 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,NBins) 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,NBins) 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,NBins) 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 other programs 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 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 $p = \text{Adjust}/N$; if $p = 1$, set $p = 1 - \text{Adjust}/N$. "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.

6 Functions That CUPID Can Compute

Here are the functions that have been implemented so far. Each function returns the requested value for the *currently defined* distribution, so you always define the distribution first and then ask for functions of it.

CDF "CDF(x)" requests the cumulative probability at x.

CV "CV" requests the coefficient of variation, defined as the ratio of the standard deviation to the mean.

ConditionalRawMoment "ConditionalRawMoment(Min,Max,Power)" requests the expected value of the Power'th raw moment of X conditionalized on X falling within the range of Min to Max.

GofFChiSq This function almost computes the goodness-of-fit ChiSquare between the current distribution and a set of observed proportions. The syntax is

```
GofFChiSq(B0,O1,B1,O2,B2,...,Ok,Bk)
```

O1, O2, ... are the observed proportions in each bin (e.g., 0.1, 0.9). B0 is the lower boundary of the lowest bin, and B1, B2, ... are the upper boundaries of the various bins, and they must be in ascending order.

CUPID computes the predicted proportion in each bin, P_i, and returns

$$\sum_{i=1}^{k} \frac{(O_i - P_i)^2}{P_i}$$

The actual value of the chi-square statistic is the sample size times this value; since the sample size is not known to CUPID, you must do this computation yourself.

You may specify as many proportion/bin pairs as you like. Implicitly, there may be one extra bin above the last one you specify. The observed proportion in this bin is 1 minus the sum of the specified O_i values, and the predicted value in this bin is of course determined by the current distribution and its parameters.

The number of degrees of freedom associated with this chi-square is the number of specified bins (plus one, if there is any predicted or observed probability in the implicit extra bin) minus the number of free parameters adjusted.

Hazard "Hazard(x)" requests the value of the hazard function at x: PDF(x) / (1 - CDF(x)).

IntegralXtoNxPDF "IntegralXtoNxPDF(Min,Max,Power)" requests

$$\int_{\text{Min}}^{\text{Max}} x^{\text{Power}} f(x)dx$$

where $f(x)$ is the PDF of x.

CDFIntegral "CDFIntegral(Min,Max,Power)" requests

$$\int_{\text{Min}}^{\text{Max}} x^{\text{Power}} F(x)dx$$

where $F(x)$ is the CDF of x.

InverseCDF "InverseCDF(P)" requests X such that $\Pr(x \leq X) = P$.

Kurtosis "Kurtosis" requests $E[(x-\mu)^4]/E[(x-\mu)^2]^2$.

LnLikelihood "LnLikelihood($X_1, X_2, \ldots, X_N$)" requests

$$\ln\left[\prod_{i=1}^{N} f(X_i)\right]$$

Mean This requests the expected value of the random variable.

Median x such that $F(x) = 0.5$.

MGF "MGF(Theta)" requests the value of the moment generating function at Theta (θ), defined here as

$$\int_{\text{Min}}^{\text{Max}} e^{\theta x} f(x)dx$$

where $f(x)$ is the PDF of x and Min and Max are the lower and upper limits of the distribution.

MMMSD I don't know what to call this, but it is (Mean - Median) / SD. I like it as a measure of skewness. If anyone knows a real name for it, please let me know.

PctileSkew(P) Here is another measure of skewness. It is

$$\frac{\text{InverseCDF(P)+InverseCDF(1-P)-2}\times\text{Median}}{\text{InverseCDF(P)-InverseCDF(1-P)}}$$

If anyone knows a real name for it, please let me know.

PDF "PDF(x)" requests the value of $f(x)$.

PDFBin "PDFBin(x,Bin)" requests the value of $F(x+Bin)-F(x)$. This is particularly useful with discrete distributions because $f(x)$ is zero at lots of intermediate points.

Random "Random" requests a random number from the distribution. "Random(N)" requests N random numbers from the distribution, and these are printed out as a list with one per line. N can be any integer from 1 to 32767.

RawMoment "RawMoment(Power)" requests the Power'th raw moment of the distribution.

SIQR This is the semi-interquartile range, also known as the "DL" in psychophysical work:

$$\frac{\text{InverseCDF(.75)-InverseCDF(.25)}}{2}$$

Skewness "Skewness' requests $E[(x-\mu)^3]/E[(x-\mu)^2]^{3/2}$. This seems to be the most standard measure of skewness, although many other measures are used too.

SD Standard deviation.

RawSkewness "RawSkewness" requests $E[(x-\mu)^3]^{1/3}$, which is another potentially useful measure of skewness.

Variance $E[X^2]-(E[X])^2$.

YNProbitLikelihood and mAFCProbitLikelihood These two incredibly special-purpose functions compute the likelihood of a set of probit-type data under the current distribution (see section 11 for a brief description of probit analysis). The first is appropriate when the data come from a yes-no task, and the second is appropriate when the data come from an m-alternative forced-choice task. For example,

```
YNProbitLnLikelihood(N,Cs,Ns,NGs)
```

N is the number of stimulus constants used; the Cs are the constants themselves (there will be a list of these); the Ns are the number of tests at each constant (there will be a list of these); and the NGs are the number of positive test outcomes at each constant (there will be a list of these).

The command for the m-alternative forced-choice task is almost the same

```
mAFCProbitLnLikelihood(m,N,Cs,Ns,NGs)
```

except the first parameter is the number of alternative responses, m.

7 Functions That CUPID Can Plot

With the Plot command, you can get CUPID to make a plot of the PDF, CDF, MGF, or Hazard function of the current distribution. The format is: `Plot(PDF)` for the PDF, or you can substitute "CDF", "MGF" or "Hazard". If you want, you can restrict the range of the plot to a certain interval. For example, `Plot(PDF,1,2)` yields a plot of the PDF of the distribution in the range from 1–2.

By default, CUPID makes its plots in a separate graphics window. This graphics window is displayed automatically, but you have to return to the text window in order to continue on with the program. The text window displays the message:

```
Inspect Graph Window, then type ENTER to continue:
```

Thus, after you have looked at the graph, you have to come back to the main CUPID text window in order to type ENTER and continue the program.

Also by default, CUPID computes the to-be-plotted value of the requested function (e.g., PDF) with half of the maximum density allowed by the graphics window. For example, if the graphics window has 400 pixels across, then CUPID computes the value of the PDF at each of 200 equally-spaced points throughout the range of the random variable. If the computations are slow, though, you might wish to compute the value less frequently—say, only every third or fourth pixel across the range. You can adjust the plotting density with a command like `GraphicsPlotDensity(3)`. The integer parameter specifies the number of pixels between computed values, so the default is `GraphicsPlotDensity(2)`, and `GraphicsPlotDensity(20)` would plot only 1/10th as many points (faster but cruder). For highest resolution, of course, use `GraphicsPlotDensity(1)`.

Finally, an alternative plot format can be requested using the command `AsciiPlot`. In this format, the plots appear in a crude ASCII format in the same window where the user types commands. This plot format is mainly useful when you are running CUPID in batch mode and you would like just to get a crude idea of the shape of the plot. If you want to switch back to graphics plots after invoking the `AsciiPlot` option, the format for that is of course `GraphicsPlot`.

8 Generating Columns of Function Values

Unfortunately, CUPID won't save or print the onscreen plots that it produces. There is, however, a "Columns" command that you can use to save the computed X and Y values needed for plotting, and then you can feed these into the plotting program of your choice. (I recommend the freeware program GLE, by the way; the learning curve is a little steep at first, but it is very well worth it.)

An example of the columns command is

```
Columns(PDF,CDF)
```

which produces three columns of output: namely, X, PDF(X), and CDF(X). You can list any combination of 1–3 of the three functions of X (i.e., PDF, CDF, hazard), and CUPID will produce the appropriate number of columns.

By default, CUPID varies X in 100 equal steps from (nearly) the lower bound to (nearly) the upper bound of the distribution. You can change the number of steps by adding an extra parameter at the end. For example, this command produces a table with 1000 lines:

```
Columns(CDF,PDF,1000)
```

You can also specify the range over which CUPID varies X, with a further elaboration of the command that specifies three numeric parameters at the end:

```
Columns(Hazard,PDF,CDF,0,10,0.1)
```

This command varies X from 0 to 10 in steps of 0.1.

Finally, you can get moment-generating function values in tabular format with the same sorts of commands, for example:

```
Columns(MGF) or Columns(MGF,1000) or Columns(MGF,-1,1,0.01)
```

A restriction is that you may not request columnar output for the moment-generating function within the same "Columns" command as one of the other functions. This is because the MGF may need a very different range of X values than th PDF, CDF, and hazard function.

9 Parameter Estimation

If instructed to do so, CUPID will adjust the parameter values of the current distribution, creating a new current distribution in order to satisfy certain criteria. There are four available criteria:

Estimate From Moments This function tells CUPID to alter the parameter values of the current distribution so as to optimize the fit to a given set of moments. The syntax is:

```
EstFromMoments(M1,M2,...[,ParmCodes])
```

M1 is the mean, and M2, M3, ... are the observed central moments for which you want to find the best-fitting parameters. You may specify as many moments as you like. Currently, the program minimizes the sum of squared errors between predicted and observed moments. ParmCodes is an optional parameter described below.

For example, if the current distribution is Lognormal(3,.4) and you want the parameters changed so that the mean and variance are 100 and 40000, type:

```
EstFromMoments(100,40000)
```

After a brief search, CUPID will change to a lognormal(3.8,1.269), which has the desired mean and variance.

Note that after the search CUPID reports a “simplex minimum”. If this value is greater than zero, then CUPID could not find parameter values giving the exact moments desired. Maybe they don’t exist, or maybe the search algorithm can’t find them. You might be able to improve the fit by repeating the same command again from different starting parameter values.

Warning: This and the other estimation processes are in real danger of getting caught in local minima. Use with caution! Indeed, in some cases CUPID will bomb during the estimation process due to a numerical error. When this happens, try harder to find starting parameter values in the general neighborhood of the best parameter estimates.

Estimate From ChiSquare This function tells CUPID to alter the parameter values of the current distribution so as to optimize the fit to a given set of bin proportions. The syntax is:

```
EstFromChiSquare(B0,O1,B1,O2,B2,...[,ParmCodes])
```

O1, O2, ... are the observed proportions in each bin (e.g., 0.1). B0 is the lower boundary of the lowest bin, and B1, B2, ... are the upper boundaries of the various bins, and they must be in ascending order. ParmCodes is an optional parameter described below. See the function “GofFChiSq” for further details.

Estimate From Percentiles This function tells CUPID to alter the parameter values of the current distribution so as to optimize the fit to a given set of observed percentile values. The syntax is:

```
EstFromPercentiles(P1,O1,P2,O2,...[,ParmCodes])
```

P1, P2, ... are the percentile values (e.g., 0.1, 0.9) for which you want to find the best-fitting parameters, and O1, O2, ... are the observed values of those percentiles. You may specify as many pairs as you like. Currently, the program minimizes the sum of squared errors between predicted and observed percentile values. ParmCodes is an optional parameter described below.

Estimate From Observations This function tells CUPID to alter the parameter values of the current distribution so as to give the maximum likelihood fit to a given set of data points (observations). The syntax is either:

```
EstFromObs(Obs1,Obs2,...[,ParmCodes])
```

or

```
EstFromObs(FileName[,ParmCodes])
```

In the first version, the data values used for the likelihood calculation are given as Obs1, Obs2, Obs3, and so on. In the second version, FileName specifies the name of an ASCII file with data points listed one per line (maximum of 1000 data points). ParmCodes is an optional parameter described below.

Estimate From Probit Data Eight functions tell CUPID to alter the parameter values of the current distribution so as to give the optimal fit—in either a maximum likelihood sense or a minimum chi-square sense, as described further below—to a given set of Probit-type data (see section 11 for a brief description of probit analysis). The syntax is one of the following four options:

```
EstFromYNProbitLnLike(C1,N1,G1,C2,N2,G2,C3,N3,G3,...[,ParmCodes])

EstFromYNProbitChiSq(C1,N1,G1,C2,N2,G2,C3,N3,G3,...[,ParmCodes])

EstFromYNProbitLnLike(FileName[,ParmCodes])

EstFromYNProbitChiSq(FileName[,ParmCodes])
```

In the first two versions, the data values used for the calculations are given as C1,N1,G1, and so on. In the third and fourth versions, FileName specifies the name of an ASCII file with each line containing one Ci,Ni,Gi triple separated by spaces (e.g., “1.25 40 25”), with a maximum of 1000 triples. ParmCodes is an optional described below.

There are also four options parallel to the above for estimation from data obtained in an m-alternative forced-choice task. Here is one example:

```
EstFrommAFCProbitLnLike(m,C1,N1,G1,C2,N2,G2,C3,N3,G3,...[,ParmCodes])
```

The first parameter, m, indicates the number of options in the forced-choice task. The other three options are exactly parallel to the options for the yes-no task, with the additional m parameter first.

ParmCodes This is an optional parameter that can be specified for any of the estimation procedures just described, and you use it to tell CUPID how you want it to handle each of the parameters. Specifically, ParmCodes is a sequence of letters—one letter for each free numerical parameter in the current distribution. Each letter should be F, I, or R, with these meanings:

F The parameter is fixed at the value used in the current distribution, so the parameter search routine is not allowed to adjust it.

I The parameter can be adjusted, but it can only be set to integer values.

R The parameter can be adjusted to any real value.

Here are some examples:

- Suppose you want to work with a beta distribution with mean 0.3 and variance 0.1. Not knowing immediately what parameters to use, you guess to enter a Beta(0.5,0.5). Then, type `EstFromMoments(0.3,0.1,RR)` to tell CUPID to adjust the two parameters to produce the desired mean and variance. CUPID will search briefly and then change the current distribution to Beta(0.33,0.77), which has the desired moments.
- Suppose the current distribution is a standard normal distribution truncated at -1 and 1:

  ```
  Truncated(Normal(0,1),-1,1)
  ```

  The command

  ```
  EstFromMoments(0.3,0.8,RRFF)
  ```

tells CUPID to hold the truncation limits fixed at -1 and 1, but adjust the parameters of the normal to yield a truncated distribution with a mean of 0.3 and a variance of 0.22. It takes about 10 seconds on my 486, but CUPID eventually finds that setting the normal mean and standard deviation to 1.039 and 0.9717, respectively, yields a truncated distribution with the desired mean and variance. Note that the truncated distribution has four parameters, corresponding (from left to right) to the normal mean and sd and the lower and upper truncation cutoffs. The ParmCodes of RRFF says, "vary the first two but leave the last two alone." If you used IIFF instead, CUPID would try to find the best solution in which the normal mean and sd were integers.

- Suppose the current distribution is set to:

  ```
  Mixture(0.6,Normal(0,1),0.4,Exponential(1))
  ```

  In this case, the command

  ```
  EstFromObs(4,1.1,3.2,-0.3,0.8,RFFF)
  ```

  tells CUPID to look for different mixture probabilities maximizing the likelihood of these four observations.

Note that the parameters are mapped to ParmCode letters in strict left to right order, with one letter for each of the numbers appearing in the distribution name. The only exception to this rule arises in the case of mixture distributions, where the final mixture probability (0.4 in the third example just above) *has no letter assigned to it because it is not a free parameter.*

ParmCodes is optional; if it is not specified, CUPID allows most parameters to be varied according to their true types (i.e., real or integer). By default, though, certain parameters of derived distributions are held constant. These include:

- The bounds of truncated and bounded distributions.
- The multiplier and addend of linear transformation distributions.
- The mixture probabilities of mixture distributions.
- The order and number for Order, OrderIID, OrdBin, and OrdExp distributions.
- The power for Power transformation distributions.

10 Program Control

There are a number of CUPID commands that serve to control the program itself rather than to perform statistical computations. For example, they allow you to get some on-line help, save output to a file, exit the program, define your own aliases, get debugging information, iterate some commands, skip some commands, set parameters controlling integration and simplex searchs, read commands from a file, and insert comments into your output files. These commands are described in this section, except for those controlling random number generation, which are described in the next section.

10.1 Getting Help

There really isn't much in the way of on-line help available. The command "help" displays a one-screen summary of the program. In addition, there are these "Show" commands, each of which lists a particular type of information:

`Show(Distributions)`	Show distributional families.
`Show(Functions)`	Show functions that CUPID can compute.
`Show(Commands)`	Show commands for control of CUPID.
`Show(Aliases)`	Show currently defined aliases (see below).

10.2 Saving Output to a File

The command `OutFile(FileName)` opens an output file with whatever filename you specify. Thereafter, anything that is written to the screen is also written to the file. If you want to close the output file, just type `OutFile` without a filename. If you forget to close the output file, CUPID will do so for you when you quit the program. If you try to open an output file with the name of a file that already exists, CUPID will ask if you want to append to it or over write it.

10.3 Exiting the Program

The command to quit the program is `Stop`.

10.4 Continuation Lines

If the last character on a line is a backslash, CUPID regards the next line as a continuation of it. This can be useful, because some commands are longer than 80 characters (e.g., see “Repeat”). The maximum length of command is 255 characters—after expansion of any aliases (see below).

10.5 Color and Mono

The commands “color” and “mono” select their respective display modes. The default mode is color, but the mono mode may be more readable on black-and-white monitors (e.g., laptops).

10.6 Notes & Comments

You may insert notes and comments to yourself—to be ignored by CUPID—with the “Note” command. For example, CUPID will ignore the following line:

```
Note(Next compute predicted CDF’s without truncation:)
```

This feature is useful for adding your own notes to your output files and also in leaving notes to yourself within command files when running CUPID in batch mode.

10.7 Aliases: User-defined Macros & Abbreviations

You may want to define your own macros, or your own abbreviations for some of CUPID’s longer key words. For example, if you use the Exponential distribution a lot, you may want to abbreviate it with “Exp”. Or, if you want to do the same set of computations for many different distributions, you may want to define a macro to carry out that set, just to avoid retyping it. Both these functions can be accomplished using CUPID’s “alias” feature. Basically, CUPID maintains a list of abbreviations in memory. When you type a command, it checks your command for any string that matches one of the stored abbreviations; if it finds a match, it replaces the abbreviation with the corresponding full form.

To define your own alias, the basic command is:

```
AliasAdd(Shortform,Longform)
```

This defines the string Shortform as an abbreviation for Longform, as in the example

```
AliasAdd(Exp,Exponential)
```

Another example is

```
AliasAdd(4Moms,Mean Variance Skewness Kurtosis)
```

Unlike other CUPID commands, the Alias command may not be followed by other commands on the same line.

Aliases can be written to and read from files, so that you can save them across CUPID sessions. The command `AliasToFile(FileName)` writes the current set of aliases to the specified file, and the command

`AliasFromFile(FileName)` reads a set of aliases from the specified file. With the latter command, if some aliases are already defined, the new aliases are simply added to the current list.

For convenience, CUPID allows you to maintain a standard list of aliases that are loaded up automatically. Each time you start CUPID, it looks for a default alias file called CUPID.ALI. It looks first in the current directory, and it automatically loads in this file of aliases if it is found. If there is no CUPID.ALI in the current directory, CUPID also looks for CUPID.ALI in the CUPID path (see section 10.19). In short, this means you should:

- Save an alias file CUPID.ALI in the current directory if you want that alias file loaded automatically any time you start CUPID in that directory.
- Save an alias file CUPID.ALI in the CUPID path (see section 10.19) if you want that alias file loaded automatically any time you start CUPID from any directory.

Other stuff about aliases:

- The Alias command cannot be aliased.
- The maximum number of aliases is 100.
- An alias file is a regular ASCII file—one line per alias—with a very obvious format. You can edit it with an ASCII editor to make changes.
- If you ever want to clear all the currently defined aliases, just type `AliasFromFile` without specifying a file name.

10.8 @Self

The string “@Self” is essentially a special alias that refers to the currently defined distribution, and you can use it to save retyping this distribution’s name. For example, suppose the current distribution is “ExGaussian(123.456,12.345,0.012345)”, and you would like to truncate it between 150 and 200. You can simply type:

```
Truncated(@self,150,200)
```

and this will have exactly the same effect as if you had typed:

```
Truncated(ExGaussian(123.456,12.345,0.012345),150,200)
```

because “@self” is always replaced with the name of the current distribution.

“@Self” can also be useful in defining aliases. After adjusting the parameters of a distribution, you may want to define an alias for it so that you can use it again later. You can define `FirstD` as an alias for the current distribution by typing:

```
AliasAdd(FirstD,@self)
```

10.9 Getting Debugging Information

CUPID will sometimes bomb if you ask for computations that lead to floating point errors. If you are curious to find out where the errors occur, you can issue the command `DebugMode`. This turns on debugging mode, and CUPID then prints out lots of information about what it is doing at each moment. You’ll probably want to have opened an output file, because there will be more than a screenful.

10.10 Iteration — the "Repeat" Command

CUPID has a "repeat" command to handle some types of repetitive calculations automatically. The format is:

```
Repeat(VarName,FromX,Increment,ToX)Command
```

For example, if you want to compute the inverse of the CDF for the 10th–90th percentiles, you could type:

```
Repeat(XXX,0.1,0.1,0.9)InverseCDF(XXX)
```

VarName is any alphanumeric string. It is like a place-holder; it indicates where in the command you want to use the numbers whose values range from FromX to ToX by Increment. This may seem conceptually superfluous in the above example (though it may not be omitted), because the InverseCDF command takes only one argument—obviously you want the repeat command to apply to that argument. But other commands take more arguments.

The Repeat command executes at the end of the current logical line of input, which means that the scope of the command is just one line. For example, the command

```
Repeat(Mu,0,0.1,1) Normal(Mu,1) CDF(1)
```

will print out the CDF(1) for each of 11 normal distributions, but

```
Repeat(Mu,0,0.1,1) Normal(Mu,1)
```

will just cycle through the distributions, initializing each one without printing out anything. You can use a backslash at the end of a physical line to signal that you want to continue the current logical line onto the next physical line, up to a total of 255 characters.

Repeats can be nested. For example, the following command, given across two lines, will produce a very complete *F* table (note the backslash is used to continue onto the next line):

```
Repeat(Num,1,1,100)Repeat(Denom,1,1,100) F(Num,Denom) \

Repeat(Prob,0.90,0.01,0.99)InverseCDF(Prob)
```

The repeat command has two significant quirks:

- Processing of variable names is *case sensitive*, so `Repeat(X,0.1,0.1,0.9)InverseCDF(x)` will not work.
- The variable name must be a unique string occurring no where else in the command, or else CUPID will get confused. For example, `Repeat(C,0.1,0.1,0.9)InverseCDF(C)` will not work because the variable name "C" occurs within the string "InverseCDF". Just use CC instead.

10.11 SkipTo

It is sometimes handy to be able to skip over lines in a command file of the sort invoked with "InFile". For example, you might make a long series of commands in a file, start executing the file, and then have CUPID halt because of a typo in a command towards the end of the file. In that case, you might like to restart where you left off, without disturbing the command file more than necessary. To do that, you can put a command like `SkipTo(continue)` at the beginning of the commands you want to skip over, and then put the line `continue` at the point where you want CUPID to start processing again. The intervening commands will simply be ignored. Note that the name of the target label ("continue" in this example) is case-sensitive.

10.12 IntegralPrecision

In many cases where explicit formulas have not been programmed in, CUPID computes values by numerical integration. The speed and accuracy of this iterative procedure are controlled by three parameters called "IntegralMinSteps," "IntegralPrecisionAbs," and "IntegralPrecisionRel". The integral is accepted as having converged when the difference (D) between two successive approximations is less than IntegralPrecisionAbs in absolute terms **or** is less than IntegralPrecisionRel times the current approximation. Thus, both of these parameters should be small positive numbers (by default, both are 1.0e-7), and smaller values produce more accurate integrals, but take longer to compute. You may change the value of either parameter by typing its name and the new value as its argument (e.g., `IntegralPrecisionRel(0.00002)`). If you just type the parameter name with no argument (e.g., `IntegralPrecisionAbs`), CUPID prints the current value of the parameter so you can see what it is. In addition, the numerical integration routine requires at least IntegralMinSteps before convergence (essentially, it divides the range of integration into 2^{Steps} segments on each step). By increasing IntegralMinSteps from its default of 2, you can insure a finer sampling of the integral to make sure that you don't miss a localized bump.

There is also an additional parameter called "IntegralPrecisionGlobal" that is the default setting for IntegralPrecisionAbs and IntegralPrecisionAbs, used each time a new distribution is created. You can also set IntegralPrecisionGlobal with the analogous command.

10.13 InversePrecisionP and InversePrecisionX

When CUPID must find the inverse of a CDF

```
InverseCDF(P)
```

for a distribution with no explicit InverseCDF function built in, it uses a numerical search algorithm to find the desired X value corresponding to the specified P. These parameters control the accuracy required for the search to stop (defaults are 1.0e-7). The former controls the tolerance for the desired P, and the latter controls the tolerance for the desired X value. They can be set and interrogated in a manner analogous to the IntegralPrecision parameters.

10.14 ZExtreme

By default, CUPID considers the normal distribution to range between Z values of ±20, but such wide limits can sometimes cause numerical errors. You can change these Z limits, for example to ±10, with a command of the form:

```
ZExtreme(10)
```

Note that a change in ZExtreme only affects normal distributions defined after the new ZExtreme value is in place, not ones that are already in use.

10.15 StepSize and MinStepSize

These two commands allow you to set two parameters that control the Simplex search routines that carry out the estimation process. The first parameter controls the initial step size (i.e., the sizes of the first steps that are tried), and the second parameter controls the minimum step size at which the search is considered to have converged. Each command works like the IntegralPrecision commands described earlier: Type the command by itself to find out the current value of the parameter, or type the command with an argument to set a new value of the parameter.

10.16 A Shortcut for Changing Parameter Values

In some cases you may want to change one parameter of the distribution you are working with, and not want to retype the whole thing. For example, suppose you initially asked for

```
Convolution(Truncated(Normal(0,1),-1,1),OrderIID(3,5,Exponential(.10))
```

and then you decided you wanted to try the same distribution but with an exponential rate of .15 instead of .10. Instead of retyping the whole thing with the new exponential rate, you can just type `ChangeParm(7,.15)` or even `Cha(7,.15)` This reinitializes the current distribution, changing the value of the 7th parameter to .15. Parameters are always counted from left to right within the distribution name.

10.17 UseExpRate / UseExpMean

By default, the parameter of the exponential distribution is defined as the rate, and the mean equals 1.0 / rate; the analogous definition is used in the ExGaussian, Gamma, and RNGamma distributions. If you would rather think in terms of the mean than the rate, you may indicate this to CUPID by issuing the command `UseExpMean`. Thereafter (until changed back by `UseExpRate`), each time you enter this parameter it will be assumed that you have entered the mean rather than the rate.

10.18 Seeding the Random Number Generator

CUPID uses the "Mersenne Twister" uniform random number generator of Matsumoto and Nishimura (1998). This section describes how CUPID handles seeding of that random number generator. Not everyone will be interested in such an arcane topic, but it may be useful for serious simulation work, because the user can control whether the random number generator "continues on from where it left off" or "restarts from the same spot" on successive runs of CUPID.

- When it starts up, CUPID checks the current directory for a file called CUPID.MT. If this file is found, the initial state of the random number generator (i.e., at the start of CUPID) is retrieved from this file.
- If no file CUPID.MT is found in the current directory, then CUPID checks for the file CUPID.MT in the CUPID path (see section 10.19), and reads the initial state of the random number generator from this file if it is found.
- If no CUPID.MT file is found, then CUPID always starts the random number generator by seeding it randomly when the program begins. Obviously, this means that the random number generator starts in a different state each time the program starts up.
- If a CUPID.MT file was found at startup, then by default CUPID will save the new state of the random number generator in this same file (same directory) when it quits. The net effect is that CUPID's random number generator starts up from where it left off each time, insuring a fresh sequence of random numbers for each run (by default).

Note that CUPID's random number generator will only "continue on" by default if it finds the file CUPID.MT in either the current directory or the CUPID path.

CUPID provides four commands for altering its seeding procedure, for use in cases when the above default behavior is not what you want:

Randomize This command generates a new random state for the random number generator. This command takes no parameters.

RandRdState The command `RandRdState(Filename)` reads the state of the random number generator from the file "filename", which should previously have been written using the command described next. If the file name is omitted, CUPID rereads the same CUPID.MT file that it read when it started.

RandWrState The command `RandWrState(Filename)` writes the current state of the random number generator to the file "Filename", so that this same state can later be reinstated using the RandRdState command. If the file name is omitted, CUPID writes over the same CUPID.MT file that it read when it started.

RandSave This command is a toggle that controls whether CUPID does or does not save the final state of its random number generator. It is here in case you want to turn off CUPID's default procedure of updating its random number generator seed at the end of each run. In other words, if you want to start CUPID with the random number generator in the same state on two successive runs, issue the command RandSave during the first run so that CUPID will not update its seed file.

10.19 The CUPID Path

You can define a CUPID path to tell CUPID where to look for the random number generator state file CUPID.MT and the default alias file CUPID.ALI. This path is specified by the environment variable "CUPID". So, for example, you can set the environment variable CUPID=C:\CUPID, and then store your random number generator and default aliases in that directory.

10.20 Control of CUPID from a command file

CUPID accepts commands from an ASCII text file and behaves just as if you had typed in the commands interactively. This feature is handy if you want to do a long series of slow calculations — you can set them up with any ASCII editor and then let CUPID do them without your attention.

The command `InFile(FileName)` reads a sequence of commands from an input file with whatever filename you specify. The commands in the file should be just the same as you would type at the CUPID prompt, and they are processed in the same way.

With this feature, you can create files of commonly used command sequences and effectively invoke them like subroutines.

This feature is also useful when you want to prepare lists of commands for unattended computation (e.g., because they are slow). For example, you might store a list of slow commands in the file `CupCmd01.txt` and then process that file with:

```
C:> CUPID Outfile(OutFile01.txt) Infile(CupCmd01.txt)
```

(Again, note that an outfile is specified first so that the results will be saved on the disk as well as written to the screen.)

10.21 Control of CUPID from the Command Line

CUPID can also take commands from the command line as a convenience, for real quick and dirty computations. For example, at your command prompt you could type:

```
C:> CUPID CDF(2.34)
```

to ask CUPID to give you the cumulative probability of 2.34 from its default distribution (i.e., the standard normal).

You can also give several commands on the command line, like

```
C:> CUPID Exponential(.10) SD CDF(2.34)
```

to get the standard deviation and CDF of 2.34 for the exponential distribution with the indicated rate.

When commands are given on the command line, CUPID automatically quits after it executes the last command. This too is a feature.

10.22 Table of Known Functions

CUPID computes many quantities by default numerical approximation techniques (e.g., numerical integration) that work for any distribution, but it also has many explicit functions built-in for particular distributions. These are called "known functions". An example is the CDF of an exponential distribution, which can be computed directly as $1 - e^{-\lambda x}$ rather than by numerical integration of the PDF. You can see which functions are known to CUPID by using the command "TableOfKnowns". Given this command, CUPID will print out a large table in which the rows correspond to different probability distributions and the columns correspond to different functions (e.g., PDF, CDF, raw moment). (Be sure to specify an output file, because the table is far too big to fit on one screen.) An X in a cell of the table means that CUPID uses a built-in function to compute that quantity for that distribution, not a general numerical approximation technique. Of course, in some cases the built-in functions are themselves approximations, but they are usually better (in terms of speed, accuracy, or both) than simple numerical integration.

11 Probit Analysis

In brief, the Probit analysis is used in the following situation (cf. Finney, 1978):

1. A researcher selects an ordered set of k constants $C_1, C_2, C_3, \ldots, C_k$ (e.g., $5, 10, 15, 20, \ldots, 50$) roughly spanning a probability distribution.

2. For each constant C_i, the researcher takes N_i independent random samples X_{ij} from the distribution, $j = 1, \ldots N_i$. The value of X_{ij} cannot be observed directly, however. Instead, the researcher observes *only* Y_{ij}, where
$$Y_{ij} = \begin{cases} 0 & \text{if } X_{ij} \leq C_i \\ 1 & \text{if } X_{ij} > C_i \end{cases}$$

3. The data are summarized by counting the number of observations greater than each C_i:
$$G_i = \sum_{j=1}^{N_i} Y_{ij}$$

4. The problem is to estimate the probability distribution of the X_{ij} values from the k observed proportions, G_i/N_i, i=1…k.

With such data, the parameters of the distribution of the X_{ij} values can be estimated by either maximizing likelihood or minimizing chi-square. For any given set of parameter values, the likelihood of the specific ordered set of X values is
$$L = \prod_{i=1}^{k} p_i^{(N_i - G_i)} \cdot (1 - p_i)^{G_i}$$
where $p_i = CDF(C_i)$ with the given parameter values (Finney, 1971, chap. 5).[2] When requested to use the maximum-likelihood type of probit fit, CUPID adjusts parameters iteratively to maximize this value (actually, to minimize the negative of the natural logarithm of this value) using the numerical search algorithm of Rosenbrock (1960). The simplex minimum value that it reports at the end of its search is -Ln(L).

---

[2] To get the likelihood of the G_i values instead, you would include the binomial coefficients $\left(\begin{array}{c} N_i \\ G_i \end{array} \right)$ in this product. This has no effect on the parameter estimation, however, because these constant multipliers can be factored out, i.e.,
$$\prod_{i=1}^{k} \left(\begin{array}{c} N_i \\ G_i \end{array} \right) \cdot p_i^{(N_i - G_i)} \cdot (1 - p_i)^{G_i} = \prod_{i=1}^{k} \left(\begin{array}{c} N_i \\ G_i \end{array} \right) \prod_{i=1}^{k} \cdot p_i^{(N_i - G_i)} \cdot (1 - p_i)^{G_i}$$
I thank Rolf Ulrich for supplying the information in this footnote.

Alternatively, for any given set of parameter values, a chi-square goodness-of-fit test may be computed as (Guilford, 1936)[3]

$$\chi^2 = \sum_{i=1}^{k} N_i \frac{(\hat{p}_i - p_i)^2}{p_i \cdot (1 - p_i)}$$

where $p_i = CDF(C_i)$ with the given parameter values and $\hat{p}_i = (N_i - G_i)/N_i$. When requested to use the ChiSq type of probit fit, CUPID adjusts parameters iteratively to minimize this value. The simplex minimum value that it reports at the end of its search is the minimum obtained value of χ^2.

12 Disclaimer and Warnings

All distributions are represented numerically, with finite limits. CUPID's version of the standard normal distribution, for example, goes from approximately -20 to 20, not from $-\infty$ to ∞. Similarly, there are numerical bounds for all distributions (you can find out what bounds CUPID is using with the functions `minimum` and `maximum`). In addition, CUPID sometimes has to change the bounds of naturally bounded distributions in order to avoid numerical errors. Beta distributions, for example, start at 0.00001 instead of 0.0, because the method used for evaluating the Beta PDF produces a numerical error at 0.0.

In most cases, it seems that CUPID's bounding does not have much effect on the accuracy of the computations. But there are a few cases where it does. For example, CUPID's Cauchy distributions are really truncated Cauchys, so CUPID will compute moments for them, despite the fact that true Cauchy moments do not really exist!

Because it is very general, CUPID 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 (the CupiTest program can help with this; see section 14). One good check is to make very minor changes in parameter values and make sure that the results change only slightly.

13 Algorithms

Serious users may want to know something about the algorithms used in CUPID. This section briefly describes the numerical integration and simplex algorithms, and gives further information about how random numbers are generated.

13.1 Numerical Integration

In most cases where explicit formulas have not been programmed in, CUPID computes values by numerical integration. I wrote the integration algorithm myself, and it is pretty crude. Much better algorithms are available in the book by Press et al. (1986), but they are copyright'ed so I can't distribute them.

13.2 Simplex

The simplex algorithm used for parameter searching is that of Rosenbrock (1960). My main modification of it was to force it to round parameter values that are constrained to be integers rather than real numbers. Another trick I often use is to force bounded parameters into their acceptable ranges (e.g., binomial p must be between 0 and 1) using logit and inverse logit transformations.

---

[3] The chi-square test can be derived by conceiving of the probit data set as a multinomial with k categories. Let N_i be the number of independent observations at each C_i, let G_i and $N_i - G_i$ be the numbers of successes and failures, and let p_i be the predicted probability of a success (i.e., $\mathrm{CDF}(C_i)$ in a yes/no task or $1/m + (1 - 1/m) \cdot \mathrm{CDF}(C_i)$ in an m-alternative forced-choice task). Then the standard chi-square test for a multinomial is computed as

$$\chi^2 = \sum_{i=1}^{k} \left[\frac{(G_i - p_i \cdot N_i)^2}{p_i \cdot N_i} + \frac{((N_i - G_i) - (1 - p_i) \cdot N_i)^2}{(1 - p_i) \cdot N_i} \right]$$

This formula can be simplified to obtain the formula given by Guilford (1936). I thank Rolf Ulrich for supplying the information in this footnote.

13.3 Random Number Generation

CUPID uses various techniques for generating random numbers from non-uniform distributions, depending on the distribution. For some distributions, random numbers are generated using algorithms obtained from Devroye (1986). This includes the beta, exponential, and normal distributions. For other distributions, a random number is generated from an appropriate parent distribution (e.g., a chi-square is generated from a gamma). This includes the chi-square, F, and t distributions. For still other distributions, a random number is generated by construction (e.g., a gamma is generated by summing an appropriate number of random exponentials). This includes the gamma, exgaussian, ExpoNo, lognormal, and Weibull distributions, plus all the derived distributions. Finally, for the rest of the distributions CUPID generates a uniform random number between zero and one as the CDF, and then works backward with InverseCDF to find out what random number has that desired CDF. It would not be difficult to incorporate more routines from Devroye for distributions where the InverseCDF procedure is too slow or too inaccurate to give satisfactory results.

14 CupiTest

CupiTest is a special-purpose program designed to check CUPID's handling of any distribution, and it can sometime uncover both numerical problems and programming errors. In brief, for any distribution to be checked, it computes many functions in at least two ways (e.g., computing the CDF directly versus by numerical integration of the PDF) and compares results, noting any differences that are found. CupiTest should always be run several times, with various combinations of parameter values in the appropriate range, as a check on whether CUPID's numerical approximations are adequate.

To use CupiTest, invoke it from a command window with two parameters, like this:

```
CupiTest Normal(0,1) testout.txt
```

The first parameter is the name of the distribution to be checked, just as you would type it into CUPID. The second parameter is the name of an output file into which to save the results. I hope the output file is reasonably self-explanatory; suspicious results are marked with !!! in the output file.

15 Contributing New Routines to CUPID

No doubt many people will wish that CUPID contained other probability distributions or functions of distributions, wondering how I could possibly have left out the exact one that they most commonly need. Sorry!

In fact, I've worked much harder on developing a cohesive structure and interface for this tool than I have on making it complete. On the other hand, I will be happy to add new probability distributions or computational functions to the program, if others will send me the appropriate algorithms. The literature on statistical algorithms is huge and enormously helpful, but the algorithms would be a lot more convenient for everybody if they were already "in the computer" rather than sitting on a shelf in the Science and Engineering Library.

16 Release History

Version 2.13 was released later in January 2006, mainly to coincide with the re-release of the newly renamed program FitDist. Some changes for this release:

- The "columns" command was added, eliminating the need for CupCols.
- The automatic approximation feature was added (i.e., via the asterisk).
- The Rosin-Rammler distribution was added.

Version 2.12 was released in January 2006, with these improvements:

- Plots shown in graphics windows.

- Basic facilities for entering and editing the input lines typed by the user, including a history buffer.
- CupiTest was added.
- Some further new distributions.

Version 1.2 was released on a limited basis in January 1999, with these improvements:

- These new distributions were added: Bounded, Cosine, ExGaussRat, ExpSum, ExpSumT, InfMix, MinBound, OrdExp, and UniformInt, Wald.
- All of the approximation distributions were added.
- The programs CupCols and CupCols2 were added.
- Command-completion was added, so that it was only necessary to type enough of a command or distribution name to identify it uniquely.
- The automatic @Self abbreviation was added.
- A bug in parsing large integer values was fixed.
- PDFBin was added.
- EstFromChiSquare was added.
- Fixed famous Pascal "CRT bug" that causes run-time errors on fast machines.

Version 1.1 was released in January 1997 (DOS only), containing minor bug fixes and improved documentation. Version 1.0 was released in October 1996 (DOS only).

17 Related Programs

CUPID is one of a family of programs built from the same core code defining an object-oriented implemention of probability distributions. If this program is useful to you, you may also be interested in one or more of the others. Here is a complete list:

Cupid Interactive computations (pdf, cdf, moments, etc) with probability distributions. Can be used (for example) as an on-line table for distributions.

RandGen Generates random values from a specified probability distribution.

FitDist Estimates best-fitting parameters of a given probability distribution for a given data set.

MixTest Computes a likelihood ratio test to see whether the difference between two conditions (say "experimental" versus "control") is a "uniform effect" or a "mixture effect". With a uniform effect, all of the scores in the experimental condition are increased relative to what they would have been in the control condition. With a mixture effect, however, only some of the scores in the experimental condition are affected; the rest of the scores in this condition are the same as they would have been without the manipulation (i.e., the same as they would have been in the control condition).

Pmetric Estimates the parameters of a probability distribution from a data set relating the proportion of a certain (binary) response to a physical quantity. This type of analysis is often called "probit" analysis, and it is used (for example) in bioassay (analysis of dose/response curves) and psychophysics (analysis of psychometric functions).

18 Author Contact Address

I welcome bug reports and suggestions for improvement (regarding the software and/or the documentation). I would also welcome suggestions for further probability distributions to be added, although I can't promise any fast action on those.

I would also really like to receive feedback on who is using this software, and for what purposes. So, please e-mail me at miller@psy.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.

Here is how to contact me:

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

19 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. Wolf Schwarz provided all the information and code for the ex-Wald distribution, Angelo M. Mineo provided crucial information about the general error distribution, and Angelo Mazza provided information about the Neyman Type A distribution. 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, Duncan Murdoch, and Ellen Hertz. Special credit goes to Alann Lopes, who showed me how object-oriented programming could be useful in the first place.

20 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.

Evans, M., Hastings, N., & Peacock, B. (1993). *Statistical distributions. (2nd ed.)* New York: Wiley.

Finney, D. J. (1971). *Probit analysis: A statistical treatment of the sigmoid response curve. [3rd ed.].* Cambridge: Cambridge University Press.

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

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

Guilford, J. P. (1936). *Psychometric methods.* New York: McGraw-Hill.

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: 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.

Schwarz, W. (2001). The ex-Wald distribution as a descriptive model of response times. *Behavior Research Methods, Instruments, & Computers, 33,* 457–469.

Schwarz, W. (2002). On the convolution of inverse Gaussian and exponential random variables. *Communications in Statistics: Theory and Methods, 31,* 2113–2121.

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.

21 Software License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

21.1 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.

21.2 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