CUPID: A MATLAB Toolbox for Computations with Univariate Probability Distributions

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

September 2, 2021

Contents

1 Introduction

CUPID is a library of MATLAB classes for working with probability distributions. It was developed primarily for stochastic simulation and modelling, although it can also be useful in data analysis. There is no "main program" per se—just building blocks that can be used in your own programs.

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.

Random Number Generation: The CUPID classes generate random numbers from any univariate distribution that is supported (e.g., for computer simulations). In addition, the program `DemoRandGen` shows how to generate random numbers from multivariate distributions with any marginals known to CUPID and with any desired underlying correlation matrix.

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) matching a target proportion between certain bounds, (f) maximizing the likelihood of a set of probit-type data, or (g) minimizing the chi-square fit to a set of probit-type data.

Constrained Estimation: CUPID has considerable support for fitting one or more distributions whose parameters are constrained. For example, a single parameter might be constrained to fall within a certain range, or 2+ parameters of a single distribution might be constrained to have a certain numerical relationship, or even multiple parameters of different distributions might be constrained relative to one another. See section 6 for details.

Estimation from censored data: CUPID can also find maximum-likelihood estimates of the parameters of an underlying distribution when known numbers of observations have been lost due to censoring at known lower and upper bounds. For example, you might want to estimate a distribution's parameters but have only a set of observed values within the interval 0–100, knowing that there were also k_1 unrecorded values less than 0 and k_2 unrecorded values greater than 100.

Here are some kinds of distributions that have been implemented:

Standard distributions: The CUPID classes include support for many common distributions (e.g., Normal or Gaussian, Uniform, Gamma, F, t, ...), which I refer to as "standard" or "primitive" distributions.

Transformation distributions: Other CUPID classes provide support for distributions formed by transforming the standard distributions. A very simple derived distribution is a linear transformation of a standard 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 standard distributions), the natural log ($Y = \ln(X)$ and inverse ($1/X$) transformations, and power function transformations, among others. Still more complicated transformations can be constructed by combining several of these (e.g., log transform of an inverse transform).

Derived distributions: Other CUPID classes provide support for distributions formed from the standard distributions in more complicated ways. For example:

- **Truncated distributions:** New distributions may be defined by restricting the scores from other distributions to part of their usual range (e.g., a standard normal distribution restricted to the range from -2 to +2).
- **Mixture distributions and infinite mixture distributions:** Distributions may be formed by randomly selecting one of a set of independent random variables (e.g., a distribution that is either a standard normal, with probability p, or a uniform from -2 to +2, with probability $1 - p$). Distributions may also be formed as "infinite mixtures" by letting the parameter of one distribution vary randomly according to some other (usually different) distribution. For example, suppose X follows a uniform(0,1) distribution, and Y is normally distributed with mean X and $\sigma = 1$. Y has an infinite mixture distribution, because it is formed by mixing an infinite number of normal distributions (i.e., all of which have means between 0–1).
- **Convolutions and difference distributions:** Distributions may represent the sums of values of independent random variables (e.g., the sum of an independent standard normal and a standard uniform). Difference distributions may also be formed.
- **Distributions of order statistics:** Distributions 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). The distributions of the different sample scores may be identical or different.

Distributions associated with hypothesis testing: CUPID supports certain distributions arising in connection with hypothesis testing. For example, suppose a researcher conducts a study that is analyzed with a one-sample t-test, sample size $n = 20$. According to the null hypothesis, the sampling distribution of possible t values is $t(19)$. Suppose that the null hypothesis is false, however, and the sampling distribution of possible t values is really a Noncentral t with df=19 and noncentrality parameter 0.4. Now, under these assumptions, what is the distribution of attained p values that would be expected? That is, from what distribution of p values does the researcher obtain a sample when doing a single study? This distribution can be computed exactly, and it is represented by a CUPID class:

```
pcurve=AttainP(tNoncentral(19,.4),t(19))
```

Optionally, the distribution of p can be computed for one-tailed testing (either tail) or two-tailed testing. The AttainP distribution can handle any continuous true and hypothesized distributions of the test statistic, so it could also be used with, for example, rNoncentral and r or FNoncentral and F.

For a more detailed demonstration of how the CUPID library is used and what it can do, step through the script `DemoCupid1`. This document provides detailed information about the distributions and functions available within CUPID, but it is intended mostly as a reference guide—not an introduction. Complete lists of the standard, transformation, and derived distributions implemented so far can be found in section 3, and descriptions of the functions can be found in section 4. It is easy to add new distributions and functions, so contributions are welcome.

2 User Interface

Note: For convenience, the commands illustrated in this manual can be found in the MATLAB script file `Docs.m`.

The CUPID classes are used in MATLAB's command window (i.e., there is no gui). Ordinarily, you explicitly create an object for each distribution that you want to work with. For example, `mydist1=Uniform(0,100);` creates an object implementing a uniform distribution ranging from 0 to 100. Here is a more elaborate example:

```
mydist2=Mixture(0.65,Normal(0,1),0.35,Normal(1,1))
```

This command creates a distribution that is a mixture of two normals—a standard normal with probability 0.65 and a normal with $\mu = \sigma = 1$ with probability 0.35.

After you have created a distribution, use MATLAB's regular object-oriented syntax to call its functions, as shown in `DemoCupid1`. For example, typing `my1sd=mydist1.SD` computes the standard deviation of the indicated distribution.

3 Distributions Available in CUPID

3.1 Continuous Distributions

Here are the (mostly) standard 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

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

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

ChiSqNoncentral(df,noncen) This is the distribution of the sum of *df* independent squared normals with nonzero means. The noncentrality parameter noncen is the sum across the *df* chi-squared RVs of the squared mean of the normal going into that chi-square (divided by its sd).

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

ConstantC(C) This is a continuous version of a distribution with a single constant value. Parameter estimation (i.e., adjustment of C values) is not supported for this distribution. (For convenience, there are both discrete and continuous versions of "Constant" distributions.)

Cosine(μ,s) Where μ and s are the mean and half-width of the distribution, respectively. For $\mu - s \le x \le \mu + s$, the PDF is $f(x) = \frac{1}{2s}\left[1 + \cos\left(\frac{x-\mu}{s}\pi\right)\right]$, and the CDF is $F(x) = \sin\left(\frac{x-\mu}{s}\pi\right)$.

DblMon(t_o,δ,ϵ) This is the "double monomial" distribution (see, e.g., Luce, 1986, p. 510) with PDF

$$f(x) = \frac{(\delta+1)(\epsilon-1)}{(\delta+\epsilon)\,t_o} \begin{cases} (x/t_o)^{\delta}, & 0 \le x \le t_o, \qquad \delta > 0 \\ (x/t_o)^{-\epsilon}, & t_o \le x < \infty, \qquad \epsilon > 1 \end{cases}$$

ExGamma(K, λ_G, λ_E) *DEPRECATED: Use Convolution, Convolve2, or ConvolveFFTc instead.* This is the distribution of the sum of independent gamma and exponential random variables. Its parameters are the shape K and scale λ_G of the gamma, and the rate λ_E of the exponential. Note: This distribution has not been tested extensively and there may be very bad numerical errors with some parameter values.

ExGammaMSM(μ_G, σ_G, μ_E) *DEPRECATED: Use Convolution, Convolve2, or ConvolveFFTc instead.* This is the distribution of the sum of independent gamma and exponential random variables. Its parameters are the mean μ_G and standard deviation σ_G of the gamma, and the mean μ_E of the exponential. Note: This distribution has not been tested extensively and there may be very bad numerical errors with some parameter values.

ExGauss(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.

ExGauMn(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.

ExGauRatio(μ, σ, r) This is another 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. The parameter is the rate λ; the mean is $1/\lambda$.

ExponenMn(μ) This distribution is just a reparameterization of the exponential; the parameter is now the mean value (i.e., $1/\lambda$).

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

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

ExtrVal1(α, β) Extreme-value Type I distribution (a.k.a. Fisher-Tippett distribution, Gumbel distribution, sometimes also called the double exponential distribution, not to be confused with the Laplace distribution), with parameters α and $\beta > 0$. The CDF is

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

ExtrVal2(ϵ,θ,k) Extreme-value Type II distribution with location, scale, and shape parameters, respectively. For $x \geq \epsilon$, the PDF and CDF are

$$\begin{aligned} f(x) &= \frac{k}{\theta}\left(\frac{x-\epsilon}{\theta}\right)^{-k-1} \exp\left\{-\left(\frac{x-\epsilon}{\theta}\right)^{-k}\right\} \\ F(x) &= \exp\left\{-\left(\frac{x-\epsilon}{\theta}\right)^{-k}\right\} \end{aligned}$$

ExtrVal2L(θ,k) Extreme-value Type II distribution given by Luce (1986) with scale and shape parameters, respectively. This is a special case of the ExtrVal2 distribution with $\epsilon = 0$, which I included as a convenient way to get estimates with the restriction that $\epsilon > 0$ (see AddPosTrans).

ExtrValGen(μ,σ,ϵ) This is the Generalized Extreme Value distribution, with its three parameters corresponding to location, scale, and shape, respectively. The PDF and CDF are

$$\begin{aligned} f(x) &= \frac{1}{\sigma}\left[1+\epsilon\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\epsilon-1} \exp\left\{-\left[1+\epsilon\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\epsilon}\right\} \\ F(x) &= \exp\left\{-\left[1+\epsilon\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\epsilon}\right\} \end{aligned}$$

The Gumbel or Type I extreme value distribution is a limiting version of this distribution in which $\epsilon \to 0$; the Fréchet or Type II extreme value distribution has $\epsilon > 0$; and the Weibull or Type III extreme value distribution has $\epsilon < 0$.

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

FNoncentral(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.

Frechét(shape,scale,minimum) See

`https://en.wikipedia.org/wiki/Frechet_distribution`.

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 (this distribution is often called the Erlang). In the RNGamma distribution (see below), k is any positive real.

Geary(SampleSize) The Geary statistic arises in checking skewness to test whether a set of observations come from a normal distribution (D'Agostino, 1970; Geary, 1947). The static function `obsA = Geary.ObservedA(x)` computes the observed Geary's A value for the vector of observations whose normality you want to check. Then, `Geary(SampleSize).CDF(obsA)` computes the cumulative p of that observed value.

GenNor1(Mu,Scale,Shape) This is version 1 of the "generalized normal distribution" described at

`https://en.wikipedia.org/wiki/Generalized_normal_distribution`.

It is also sometimes called the "general (or generalized) error" distribution (e.g., Evans et al., 1993, p. 57), "Subbotin's" distribution (e.g., Johnson et al., 1994, p. 195), or the "exponential power" distribution (e.g., Mineo and Ruggieri, 2005) The correct PDF is

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

although some references (i.e., 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. (1993). 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 numerical overflow errors, especially if the shape parameter is more than approximately 3.

GenNor2(Mu,Scale,Shape) This is version 2 of the "generalized normal distribution" described at

`https://en.wikipedia.org/wiki/Generalized_normal_distribution`.

ghHoag(A,B,g,h) This is the 'g&h' distribution described by Hoaglin (1985). The parameters A and B are location (median) and scale, respectively. g controls the amount and direction of skew ($g \approx 0$ is symmetric), and h controls the thickness of the tails. There are serious numerical problems with this distribution, so results are only accurate for certain ranges of parameter values. Check carefully with the parameter values of interest. Note that you may need to override the default tolerance of fzeroOpts.

HyperbolicTan(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).

JohnsonSB(Location,Scale,Alpha1,Alpha2) This is one of the Johnson (1949) system of distributions, known as the "bounded" one, which can be viewed as a transformation of the normal distribution. Specifically, consider a standard normal random variable Z. A JohnsonSB random variable J with

parameters L, S, α_1, and α_2 can be formed as $J = L + S/\left\{1 + \exp\left(-\left[Z - \alpha_1\right]/\alpha_2\right)\right\}$. Viewing the transformation in the opposite direction, $Z = \alpha_1 + \alpha_2 \cdot \ln\left\{\frac{Y}{1-Y}\right\}$, where $Y = (J - L)/S$.

Letting $X = (J - L)/S$, the PDF and CDF of X are:

$$\begin{aligned} f(x) &= \frac{\alpha_2}{S \cdot x(1-x)} \cdot \phi\left\{\alpha_1 + \alpha_2 \cdot \ln\left(x/(1-x)\right)\right\} \\ F(x) &= \Phi\left\{\ \alpha_1 + \alpha_2 \cdot \ln\left(\frac{x}{1-x}\right)\right\} \end{aligned}$$

where ϕ and Φ are the standard normal PDF and CDF, respectively.

JohnsonSU(Location,Scale,Alpha1,Alpha2) This is another one of the Johnson (1949) system of distributions, known as the "unbounded" one, which can also be viewed as a transformation of the normal distribution. Specifically, consider a standard normal random variable Z. A JohnsonSU random variable J with parameters L, S, α_1, and α_2 can be formed as $J = L + S \cdot \sinh\left\{\left[Z - \alpha_1\right]/\alpha_2\right\}$. Viewing the transformation in the opposite direction, $Z = \alpha_1 + \alpha_2 \cdot \sinh^{-1}\left(\left[J - L\right]/S\right)$.

Letting $X = (J - L)/S$, the PDF and CDF of X are:

$$\begin{aligned} f(x) &= \frac{\alpha_2}{S \cdot \sqrt{x^2+1}} \cdot \phi\left\{\alpha_1 + \alpha_2 \cdot \ln\left(x + \sqrt{x^2+1}\right)\right\} \\ F(x) &= \Phi\left\{\ \alpha_1 + \alpha_2 \cdot \ln\left(x + \sqrt{x^2+1}\right)\right\} \end{aligned}$$

where ϕ and Φ are the standard normal PDF and CDF, respectively.

KolmSmir(N) This is the distribution of Kolmogorov's D_n, where D_n is the absolute value of the maximum difference between the theoretical and observed CDFs. The parameter N is the sample size for the observed CDF. This distribution is computed entirely from its CDF, which is evaluated with Java code provided by Simard and L'Ecuyer (2011). Programming note: This distribution thus illustrates one way of calling Java code directly from inside MATLAB.

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

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

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

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

Loglogistic(scale,shape) This is the distribution of X such that $\ln(X)$ has a logistic distribution. The median is "scale", and the variance decreases as "shape" increases, with good example values aroun 3–10.

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.

LognormalMCV(μ_l, CV_l) This is the distribution of X such that $\ln(X)$ is normally distributed. The parameters are the μ_l and $CV_l = \sigma_l/\mu_l$ of the overall lognormal.

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,

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

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

NakaRush(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.

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

Pareto(K,A) This is a Pareto distribution of the first kind, as defined by (Johnson et al., 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.

ProdUni01(K) The distribution of the product of K Uniform(0,1) random variables, for $K \geq 1$, as discussed at

https://math.stackexchange.com/questions/659254/
product-distribution-of-two-uniform-distribution-what-about-3-or-more

To get the product of K Uniform(0,p) random variables, use

`MultTrans(ProdUni01(K),`p^K`)`

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

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

This distribution arises as a model of psychometric functions in visual psychophysics (e.g., Gescheider, 1997, p. 85). The threshold parameter, $T \geq 1$, 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).

r(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.

rNoncentral(SampleSize,TrueRho) This is the noncentral sampling distribution of Pearson's r (correlation coefficient) for an arbitrary true value of the correlation (i.e., ρ), assuming bivariate normality. The approximation of Konishi (1978) is used for many parameter combinations.

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

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

Recinormal(μ_X, σ_X) This is the distribution of $Y = 1/X$, where X is a normal random variable with the specified meanμ and standard deviation σ. To make sure that X does not extend across zero, its parameters must satisfy the relation $\mu > 4 \cdot \sigma$, and X is truncated at $\mu \pm 4\sigma$.

RecinormalInv($1/\mu_X, 1/\sigma_X$) This is the distribution of $Y = 1/X$, where X is a normal random variable; that is, it is the same as "Recinormal" except for the parameters. This distribution's parameters are $1/\mu_X$ and $1/\sigma_X$, which puts them on a scale somewhat closer to the scale of Y values.

RecinormalMS(μ_Y, σ_Y) This is the distribution of $Y = 1/X$, where X is a normal random variable and Y has the specified mean and standard deviation. To make sure that X does not extend across zero, its parameters must satisfy the relation $\mu_X > 4 \cdot \sigma_X$, and X is truncated at $\mu \pm 4\sigma$. WARNING: Parameter estimation is extremely slow with this distribution because μ_X and σ_X must be estimated numerically for each candidate pair of μ_Y, σ_Y. For estimation, it is better to use RecinormalInv instead, even though its parameters are not quite on the most convenient scale.

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

RNGammaMn(k, μ) This is a reparameterization of the RNGamma distribution, with the parameters specified by giving the shape parameter k and the mean μ.

RNGammaMS(μ, σ) This is a reparameterization of the RNGamma distribution, with the parameters specified by giving the mean and standard deviation of the random variable.

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

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

SkewNor(Loc,Scale,Shape) This is a skewed version of the normal distribution studied by Adelchi Azzalini and others (see, e.g., http://azzalini.stat.unipd.it/SN/index.html). In brief, the PDF is

$$f(y) = \frac{2}{\omega} \phi(\frac{y - \xi}{\omega}) \Phi(\alpha \frac{y - \xi}{\omega})$$

where ξ is the location parameter, ω is the scale parameter, α is the shape parameter (i.e., which controls skewness), and ϕ and Φ are the PDF and CDF of the standard normal distribution, respectively.

SpearKar(X,CDF) This is a Spearman-Kärber distribution, which is characterized by a CDF that increases linearly through a series of arbitrary (X,CDF) points. The distribution is defined by a vector of X values and a corresponding vector of CDF values such that Pr(y<X(i))=CDF(i). It is assumed that both the X's and the CDF's increase monotonically, and it is also assumed that CDF(1)=0 and CDF(end)=1, so these may be omitted from the CDF vector. The linear increase in the CDF from one X to the next implies that the PDF is uniform within each interval X(i) to X(i+1). For more information on the use of the Spearman-Kärber distribution in probit analysis, see section 8.4.

StochCasc2T(α_1, α_2, k) This is the distribution of waiting times for the k'th spike at stage 2 in a 2-stage stochastic cascade model developed by Schwarz (2003). Spikes are generated at stage 1 by a homogeneous Poisson process with rate α_1, and "Any spike from this first stage is transferred to the second, final stage and the transfer time between these two stages is assumed to follow an exponential density, with rate α_2" (p. 240). By default, k is fixed during parameter estimation. There are numerical problems outside the range of $k = 2$ to about $k = 50$.

StudRng(dferror,NSamples) An approximation to the distribution of the Studentized range statistic with *df* degrees of freedom for error and with *NSamples* samples. Because both parameters are integers, automatic program-based estimation of these parameters is rarely successful. Computations with the distribution are quite slow but can be speeded up substantially without toomuch loss of accuracy (for many parameter settings) by using SplineCDF [e.g., with the command `UseSplineCDF(100)`]. Acknowledgement: This distribution uses the function `cdfTukey(q,v,r)` by Peter Nagy, obtained from https://au.mathworks.com/matlabcentral/fileexchange/37450 on 9 April 2018.

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

tNoncentral(dferr,Noncentrality) This is the distribution of the t-test statistic when the null hypothesis is false, with the noncentrality parameter indexing the size of the true effect. It is most often used for computating the power of the t-test. For a one-sample t-test, the noncentrality parameter is $\sqrt{n}\frac{\mu}{\sigma}$, where μ and σ are the true mean and standard deviation. For a two-sample t-test with sample sizes n1 and n2, the noncentrality parameter is $\sqrt{n1 * n2/(n1 + n2)}\frac{\mu_1 - \mu_2}{\sigma}$, where μ_1 and μ_2 are the true means and σ is the true common standard deviation.

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:

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

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.

TriangularCW(C, W) A triangular distribution where the parameters are specified as the center C and the width W—that is, the range is from $C - W/2$ to $C + W/2$.

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:

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

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.

TriangularGCWP(C, W, P) This is a reparameterization of TriangularG. The first two parameters are specified as the center C and the width W. P, which must be between 0 and 1, indicates the location of the peak, specified as a proportion of the distance from the distribution's minimum to its maximum.

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.

UniformCW(C, W) A uniform distribution where the parameters are specified as the center C and the width W—that is, the range is from $C - W/2$ to $C + W/2$. It is often convenient to use this distribution instead of the regular Uniform(min,max) for any problem involving parameter searching, because a search with the regular Uniform will bomb if fminsearch tries parameter values where min¿=max.

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 $T/2$.

Uquad(B, T) In this distribution the density function has the shape of quadratic function across some range. The parameters are the bottom (B) and the top of the range (T).

VonMises(L, C) This is the Von Mises distribution with location and concentration parameters L and C, respectively. Values of the distribution fall within the range of $L - \pi$ to $L + \pi$. If C is large, the values are tightly concentrated around L (i.e., low variance), whereas if C is large the values are more uniform across this range. Typically, the Von Mises distribution with $L = 0$ is regarded as an analog of the normal distribution *on a circle* instead of on the real number line.

Wald(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.

Wald2(μ, a) This is a two-parameter version of the Wald distribution with $\sigma = 1$. I believe this is also called the Inverse Gaussian distribution.

Wald2MSD(μ, σ) This is a two-parameter version of the Wald distribution with $\sigma = 1$, where the two parameters are the mean and standard deviation of the resulting Wald distribution.

Weibull(Scale,Power,Origin) As defined by (Johnson and 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.

Weibull2(Scale,Power) This is a special case of the Weibull with the origin fixed at zero.

3.2 Discrete Distributions

Here are the (mostly) standard discrete distributions that have been at least partially implemented so far:

Binomial(N, P) The distribution of the *number* of successes in N independent Bernoulli trials, where the probability of success in each trial is P.

BinomialP(N, P) The distribution of the *proportion* of successes in N independent Bernoulli trials, where the probability of success in each trial is P.

BinomialMixed(P) This is also the distribution of successes in N independent trials, but each trial may have a different probability of success. P is an array of $(1, N)$, and $P(i)$ indicates the probability of success in trial i. Parameter estimation (i.e., adjustment of P values) is not supported for this distribution.

ConstantD(C) This is a discrete version of a distribution with a single constant value. Parameter estimation (i.e., adjustment of C values) is not supported for this distribution. (For convenience, there are both discrete and continuous versions of "Constant" distributions.)

Hypergeometric(N, K, n) Consider an urn with N balls of which K are white and the rest are black. We draw n balls from the urn *without replacement*. What is the probability of drawing exactly k white balls, for each k=0...min(n,K).

Geometric(P) The distribution of the number of independent trials to the first success, where P is the probability of success in each trial.

List(Xs, Ps) You can use this to represent any discrete probability distribution at all. Xs is a (k,1) vector of the random variable's k different possible values (must be in ascending order), and Ps is a corresponding (k,1) vector of the probability of each X (the Ps must sum to 1).

NegativeBinomial(N, P) The distribution of the number of failures before the Nth success in a sequence of Bernoulli trials, where the probability of success in each trial is P.

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.

RankDist(SampleSize,BasisX,BasisY) This is the distribution of the order statistic of the random variable X within a random sample consisting of one X and SampleSize-1 Y's. The values of RankDist thus vary from 1–SampleSize. The RankDist value of 1 means that X has the largest value in the sample, and the RankDist value of SampleSize means that X has the smallest value in the sample. BasisX and BasisY are the distributions of the X and Y random variables. At present, at least one of the two basis distributions must be continuous so that there is no possibility of ties.

UniformInt(Min,Max) Uniform distribution of integers in the range of Min–Max, inclusive.

3.3 Using MATLAB Distribution Objects

CUPID can also form distributions from MATLAB distribution objects. This is useful with distributions that are supported in MATLAB but not in CUPID (e.g., the Burr, Rician, and Stable distributions), because it allows these distributions to be used in ways supported by CUPID but not by MATLAB (e.g., transformations, convolutions, order statistics, etc).

For continuous distributions, this is done the special CUPID distribution `dMATLABc`, which essentially serves as a connector to MATLAB distributions. Here is an annotated example showing how to create such a distribution:

```
% The following MATLAB command makes a Burr probability
% distribution with the indicated parameter values.
burrM = makedist('Burr','a',1,'c',5,'k',1);

% The next command creates a Burr distribution within \progname.
burrC = dMATLABc(burrM,'rrr',[0 0 0],[+inf +inf +inf]);
% The first parameter is the ProbabilityDistribution object created by MATLAB.
% The second parameter is a vector of letters, one per distribution parameter.
```

```
% The letter 'r' indicates that the parameter is a real number; 'i' would
    indicate an integer.
% The third parameter is a vector indicating the lower bound for each
    parameter.
% The fourth parameter is a vector indicating the upper bound for each
    parameter.
```

Once the distribution is created—that is, `burrC` in this example—it can be used just like any other CUPID distribution. For some examples, see `Demo_dMATLAB.m`.

3.4 Transformation Distributions

CUPID can form a new random variable (Y) by taking a *monotonic* 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	Limitation on values of X
Addition ($Y = X + B$)	`AddTrans(Uniform(.5,1),10)`	
	`AddPosTrans(Uniform(.5,1),10)`	Addend constrained to be positive.
ArcSin ($Y = \sqrt{\Phi(X/2)}$)	`ArcsinTrans(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)`	
Log ($Y = \ln[X]$)	`LogTrans(Uniform(0.5,1))`	$X > 0$
Logistic ($Y = \frac{e^X}{1+e^X}$	`LogisticTrans(Normal(0.5,1))`	
Logit ($Y = \frac{X}{1-X}$)	`LogitTrans(Uniform(0.01,0.99))`	$0 < X < 1$
Multiplicative ($Y = A \times X$)	`MultTrans(Uniform(.5,1),2)`	
Phi $Y = \Phi(X)$	`PhiTrans(Uniform(-1,1))`	
Phi inverse $Y = \Phi^{-1}(P)$	`PhiInvTrans(Beta(3,3))`	$0 < P < 1$
Power ($Y = X^p$)	`PowerTrans(Uniform(.5,1),2)`	$X > 0$ (sorry!)
Square ($Y = X^2$)	`SqrTrans(Uniform(.5,1))`	$X > 0$ (sorry!)
Square root ($Y = \sqrt{X}$)	`SqrtTrans(Uniform(.5,1))`	$X > 0$

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

3.5 Derived Distributions

CUPID also knows about various sorts of distributions that can be derived from one or more standard or "basis" distributions by (for example) convolution, truncation, mixturing, and so on. In many cases, CUPID can compute moments, PDF's, CDF's, random numbers, etc, for the derived distribution just as it can for the standard distributions defined above.

ConditXLTY and ConditXGTY These two distributions are the conditional distributions of X either given that $X < Y$ or given that $X > Y$, respectively, where Y is another random variable independent of X. For example, suppose the marathon finishing times for runner A are normally distributed with $\mu = 4.32$ hours and $\sigma = 0.27$ hours, the parameters for runner B are $\mu = 4.62$ hours and $\sigma = 0.32$, and the finishing times of the two runners are independent. You can then examine the conditional distributions for each runner, conditional on that runner winning or losing the race, with commands like these:

```
RunnerA = Normal(4.32,0.27);
RunnerB = Normal(4.62,0.32);
RunnerAwinningtimes = ConditXLTY(RunnerA,RunnerB);
RunnerAwinningtimes.Mean;
RunnerAwinningtimes.PlotDens;
```

```
RunnerBlosingtimes = ConditXGTY(RunnerB,RunnerA);
RunnerBlosingtimes.SD;
```

The separate function `PrXGTY(RunnerA,RunnerB)` gives the probability that RunnerA's time is greater than RunnerB's time.

ContamShift(BasisRV,probContamination,ShiftRV) This is a mixture distribution consisting of either a basis RV or, with some probability, the basis RV is contaminated by the addition of a random value from a shift RV. For example,

```
mydist=ContamShift(Normal(0,1),0.2,Uniform(1,2)
```

is a mixture of 80% Normal(0,1) values and 20% values that are the sum of a Normal(0,1) and a Uniform(1,2).

ContamStretch(BasisRV,probContamination,StretchRV) This is a mixture distribution consisting of either a basis RV or, with some probability, the basis RV is contaminated by a stretch consisting of a multiplication of the basis RV by a random value from the stretch RV. Restriction: BasisRV and StretchRV must be strictly positive RVs. For example,

```
mydist=ContamStretch(RNGammaMS(10,2),0.1,Uniform(2,4)
```

is a mixture of 90% RNGammaMS(10,2) values and 10% values that are the product of a RNGammaMS(10,2) and a Uniform(2,4).

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. Unfortunately, at present a convolution can only be defined using two distributions, so you have to use it recursively to get convolutions of more distributions. For example,

```
mydist=Convolution(Normal(100,50),Convolution(Uniform(0,100),Gamma(3,0.01)))
```

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., the convolution of two normals is normal). Thus, computations involving these convolutions proceed via numerical integration even though more direct computation would be possible.

Another factor to keep in mind when working with convolutions is that there may be an influence of the order in which the convolutions are specified—certainly on the speed of the computations, and very probably on the accuracy as well. For example,

```
mydist1 = Convolution(Normal(0,1),Uniform(0,1));
mydist1.PlotDens; % This takes a few seconds.
mydist2 = Convolution(Uniform(0,1),Normal(0,1)); % Theoretically, this is
    the same distribution, but...
mydist2.PlotDens; % This takes many minutes!
```

Thus, when using convolutions, it may be worth experimenting a bit to see which order of distributions produces faster computations.

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

Convolve2(RV1,RV2) This is an alternative faster implementation of the preceding Convolution distribution. I find that PDF and CDF computations with this implementation take only about 70% as long as the same computations with Convolution.

For speed, this implementation uses calls to 3 private MATLAB functions that I copied into my own functions j_integralParseArgs, j_integralCalc, and j_Gauss7Kronrod15. These functions are part of MATLAB's private 'integral' functionality so I cannot distribute them, but you can make them yourself as follows:

- The file j_integralCalc.m is the output of 'type integralCalc' in R2016b 2020-04-22.
- The file j_integralParseArgs.m is the output of 'type integralParseArgs' in R2016b 2020-04-22 except that the call to Gauss7Kronrod15 was changed to j_Gauss7Kronrod15.
- The file j_Gauss7Kronrod15.m is the output of 'type Gauss7Kronrod15' in R2016b 2020-04-22.

ConvolveFFTc(RV1,RV2,...RVk) This is a further alternative way to implement the distribution of a convolution, using an approximation based on fast Fourier transforms. It is generally much faster than Convolution or Convolve2, though it may not be as accurate, and it can conveniently be used with more than two RVs. An optional parameter 'NxPoints' can be specified to adjust the number of discrete points at which the fft is evaluated. The method is similar to that described by Ruckdeschel and Kohl (2014).

ConvolveFFTcIID(k,RV) This is the convolution of k i.i.d. random variables of the indicated family and parameter values, again approximated via fft.

ConvUnif(RV1,unifMin,unifMax) This is a further special case of a convolution distribution in which the second random variable is a uniform ranging from unifMin to unifMax. For example, the following two distributions should be equivalent but most computations will be faster with mydist2.

```
mydist1 = Convolution(Normal(0,1),Uniform(0,1));
mydist2 = ConvUnif(Normal(0,1),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 stupidly, like convolutions.

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).

For a mixture distribution, the Random function produces a second output array of integers 1...K. Each integer indicates which distribution the corresponding random number was selected from.

InfMix(RV1(Parms),RV2(Parms),MixParm) The InfMix distribution is an infinite mixture, formed when a parameter of one distribution is itself randomly distributed according to another so-called "mixing" distribution. These are often called "compound" probability distributions. For example,

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

defines a random variable that comes from a normal distribution with standard deviation 5. The first parameter (as signified by the final parameter, "1") of that distribution—that is, its mean—follows a uniform distribution from 10 to 20. As another example,

```
InfMix(Normal(0,5),Uniform(10,20),2)
```

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 an InfMix distribution is:

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

where ParentDist is any 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),Uniform(0,2),1),Uniform(4,6),2)
```

defines a normal distribution in which the mean is uniform(0,2) and the standard deviation is uniform(4,6). As you might expect, computations with this distribution are quite slow (but Spline approximations reduce computation time in many cases).

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.

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

```
Product(Uniform(0,1),Uniform(0,1))
```

specifies the product of scores from two standard uniform distributions, which also ranges from 0 to 1 (not uniformly). CUPID also handles product distributions stupidly, like convolutions. At present, RV1 and RV2 must both be strictly positive RVs.

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

```
Ratio(Uniform(0,1),Uniform(1,2))
```

specifies the distribution of a standard uniform divided by one plus a standard uniform, which also ranges from 0 to 1 (not uniformly). CUPID also handles ratio distributions stupidly, like convolutions. At present, RV1 and RV2 must both be strictly positive RVs.

TruncatedX(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,

```
TruncatedX(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:

```
TruncatedX(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.

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 of three scores 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). Example:

```
mydist = OrderIID(3,10,Exponential(.1))
```

Likelihood ratio distributions Suppose data are sampled from a given probability distribution. From each observed data value, one can compute the likelihood ratio comparing the likelihood of that data value under two hypotheses, H0 and H1, i.e., Pr(data given H1) divided by Pr(data given H0). What is the distribution of the resulting likelihood ratio values?

For a continuous data distribution, this distribution can be approximated numerically:

```
mydist = LikeRatioC(Normal(0.1,1),Normal(0,1),Normal(1,1),NSplinePoints)
```

The first parameter is the distribution of the actual data values. The second and third parameters are the distributions associated with H0 and H1, respectively, so the commands can be thought of like this:

```
mydist = LikeRatioC(DataDist,H0Dist,H1Dist,NSplinePoints)
```

The distribution is approximated using a spline for the CDF, and `NSplinePoints` is the number of equally-spaced percentile points at which to evaluate the spline (e.g., 200). (Alternatively, you may specify the exact data points at which to do the evaluation by supplying a vector instead of a single value for `NSplinePoints`).

Important restriction: It is assumed that as the data values X increase, the values of the likelihood ratio increase monotonically. Among other things, this means that H0 should predict relatively small data values, and H1 should predict relatively large values.

For a discrete data distribution, the corresponding command has the similar form:

```
LikeRatioD(Binomial(10,.5),Normal(5,1),Normal(5.5,1));
```

Note that `NSplinePoints` is not needed because the distribution is computed exactly for each possible data value rather than using a spline approximation. In addition, the restriction that is needed for a continuous X distribution is not needed for a discrete data data distribution.

Analogously, you can get the distribution of the *log* likelihood values with commands like these:

```
mydist = LnLikeRatioC(DataDist,H0Dist,H1Dist,NSplinePoints)

mydist = LnLikeRatioD(DataDist,H0Dist,H1Dist)
```

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

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

because

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

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

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

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

Here is a numerical example:

```
xvalues = 0.01:.01:.99;
mydist1 = Beta(1.4,1.5);
mydist2 = Uniform(0,1);
mydistmin = MinBound(mydist1,mydist2);
mydist1cdf = mydist1.CDF(xvalues);
mydist2cdf = mydist2.CDF(xvalues);
mydistmincdf = mydistmin.CDF(xvalues);
figure;
plot(xvalues,mydistmincdf);
hold on;
plot(xvalues,mydist1cdf);
plot(xvalues,mydist2cdf);
legend({'MinBound CDF','Beta CDF','Uniform CDF'});
```

Note that the CDF of the MinBound distribution is the sum of the CDFs of the Beta and uniform distributions (up to 1.0).

3.6 Distributions Associated with Hypothesis Testing

AttainP CUPID can implement several distributions corresponding in various ways to the distribution of attained p values that would be obtained when data points sampled from any observed data distribution are judged for significance against any hypothesized "null" distribution. The general form is something like

```
AttainP(ObservedDist,NullDist[,TailsArg])
```

, where "ObservedDist" and "NullDist" are distributions corresponding to the observed data and the null hypothesis, respectively. For example,

```
mydist=AttainP(Normal(1,1),Normal(0,1))
```

represents the distribution of p values that would be obtained if data were sampled from a Normal(1,1) distribution ("observed"), and each data point was converted to its two-tailed p value within the Normal(0,1) distribution ("hypothesized null"; e.g., an observed value of 1.96 is converted to a p value of .05). Formally, for any p, $0 \leq p \leq 1$, one can determine low and high cutoffs (L and H respectively) from the null distribution such that $\Pr(X \leq L|Null) = p/2$ and $\Pr(X \geq H|Null) = p/2$. The CDF of p in the AttainP distribution is then $\Pr(X \leq L|Observed) + \Pr(X \geq H|Observed)$.

The optional TailsArg parameter controls one- versus two-tailed testing. The default value is 2, for two-tailed testing. Alternatively, you can specify -1 for one-tailed testing at the lower end of the null distribution and +1 for one-tailed testing at the upper end.

At this point, CUPID can only handle these distributions if both the observed and null distributions are continuous.

When estimating values for any of these distributions, the default is that the parameters of the null distributions are considered to be fixed.

tPowerEst(TrueDelta,Sigma,Alpha,n_1**[,**n_2 **)]** This is the distribution of the power estimates that would be obtained based on different random samples, where each random sample yields a different estimated sample variance. The power estimate is the estimated probability of observing a t greater than the critical t value (i.e., one-tailed) given an assumed true effect size of TrueDelta. For example,

```
mydist=tPowerEst(.5,1,.05,20)
```

Note that in this distribution the estimated power varies *only* because the sample variance varies (and the estimated sample variance influences the estimated noncentrality parameter of the noncentral t).

TrueDelta is the true "effect size"; for a one-sample t-test, this is the true mean; for a two-sample t-test, this is the difference between the two true means. Sigma is the true standard deviation of the scores (one-sample) or within each group (two-sample). Alpha is the alpha level used to decide whether an observed effect is significant (typically .05). Note that the computed t critical cuts off the upper alpha/2 % of the null distribution, so this is distribution pertains to a researcher who uses two-tailed cutoffs but only considers the power to reject Ho in the expected direction (i.e., positive effect). n_1 is an integer which is the sample size for a one-sample t-test or the first sample size for a two-sample t-test. n_2, if present, is the second sample size for a two-sample t-test.

3.7 Nested Specification of Distributions

The object-oriented structure of the CUPID classes allow nested construction of distributions. Thus, it is legal to construct ad hoc distributions by any combination of the above standard distributions, transformations, and derivations. For example, this would be legal:

```
TruncatedX(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. In case you are wondering, CUPID computes the mean and standard deviation of this distribution to be 0.1731 and 0.4994.

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

4 Functions That CUPID Can Compute

4.1 Basic Functions

Here are most of the functions that have been implemented so far.

4.1.1 Distribution functions (PDF, CDF, etc)

PDF `mydist.PDF(x)` requests the value of $f(x)$.

CDF `mydist.CDF(`x`)` requests the cumulative probability at x.

OMCDF `mydist.OMCDF(`x`)` requests one minus the cumulative probability at x.

TwoTailProb `mydist.TwoTailProb(`x`)` requests the two-tailed probability associated with x. If mydist.CDF(X) ≤ 0.5, this is 2*CDF(X). If mydist.CDF(X) > 0.5, this is 2*[1-CDF(X)].

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

4.1.2 Moment-based functions (Mean, Variance, etc)

Mean `mydist.Mean` requests the expected value of the random variable.

SD `mydist.SD` requests the standard deviation.

Variance `mydist.Variance` requests the variance.

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

Skewness `mydist.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 also sometimes used.

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

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

4.1.3 Percentile functions (InverseCDF, median, SIQR)

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

Median `mydist.Median` requests the x such that $F(x) = 0.5$.

SIQR `mydist.SIQR` requests the semi-interquartile range, also known as the "DL" in psychophysical work:

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

PctileSkew(P) `mydist.PctileSkew(P)` 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 an official name for it, please let me know.

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

4.1.4 Other integrals (unconditional/conditional raw/central moments, MGF, etc)

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

ConditionalRawMoment `mydist.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.

EVFun `mydist.EVFun(Fun,Min,Max)` requests

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

where $f(x)$ is the PDF of x and `Fun` is any function of x. If Min and Max are omitted, the integral is computed from the distribution's lower bound to its upper bound. Note: The function `Fun` must be able to accept a vector of x values as inputs and must produce a vector of the corresponding `Fun(x)` values as outputs. Usually this can be done by using MATLAB's element operators such as .* and ./. For example:

```
Fun=@(x) (x./(x+1)); % This would work.

Fun=@(x) (x/(x+1)); % This would produce an error.
```

EVMAD `mydist.EVMAD` requests the expected value of the absolute deviation from the median.

IntegralXtoNxPDF `mydist.IntegralXtoNxPDF(Min,Max,Power)` requests

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

where $f(x)$ is the PDF of x. Note: For both this integral and the next one, integration takes place (by default) over equally-spaced X values. By setting the property `mydist.IntegrateOverP=true;` you can force the integration to use X values that are equally-spaced in probability. This is generally slower, though, since the InverseCDF function must be called repeatedly to find the necessary X values.

IntegralX_CToNxPDF `mydist.IntegralX_CToNxPDF(Min,Max,C,Power)` requests

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

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

CDFIntegral `mydist.CDFIntegral(Min,Max,Power)` requests

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

where $F(x)$ is the CDF of x. This is useful with an all-positive RV (i.e., CDF(0)=00, because its mean equals this integral, with Power=0, over the whole range of the RV.

MGF `mydist.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.

4.1.5 Goodness-of-fit measures (χ^2 and log likelihood)

GofFChiSq This function *almost* computes the goodness-of-fit ChiSquare between the current distribution and a set of observed proportions. An example would be something like this:

```
mydist = Beta(1.4,1.5);
B = .1:.1:1;
O = [.1 .11 .09 .1 .12 .08 .1 .1 .09 .11];
GoF = mydist.GofFChiSq(B,O)
```

B(1), B(2), ... are the upper boundaries of the various bins, and they must be in ascending order (the lowest bin is assumed to extend down to $-\infty$). O(1), O(2), ... are the observed proportions in each bin. CUPID computes mydist's 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 part of the 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.

Finally, here is an example of testing the null hypothesis that a dataset X comes from a distribution `Dist`:

```
[BinProbs, BinUpperBounds] = histcounts(X, 'Normalization', 'probability')
obsChiSq = Dist.GofFChiSq(BinUpperBounds(2:end-1), BinProbs(1:end-1)) *
    numel(X);
HoChiSq = ChiSq(numel(BinProbs)-1);
critChiSq = HoChiSq.InverseCDF(.95);
if obsChiSq>critChiSq
    disp('Reject Ho')
else
    disp('Do not reject Ho')
end
```

Kolmogorov-Smirnov one-sample test The function `kstest` computes a one-sample Kolmogorov-Smirnov test of the null hypothesis that the observations in a data vector X come from the distribution. For example, here is a command to test whether the data in X are consistent with a t distribution having 10 degrees of freedom.

```
[p, Dmax] = t(10).kstest(X)
```

Dmax is the maximum absolute difference between the empirical CDF of X and the theoretical CDF—in this case, based on a t distribution with 10 df. p is the attained significance level. Typically, with $p < .05$, the conclusion would be that the values in X are not consistent with that distribution.

If you just want Dmax but don't want the p value, you can compute that with

```
Dmax = t(10).ksDmax(X)
```

LnLikelihood `mydist.LnLikelihood(`X`)` requests

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

where X is a vector.

Measures for Probit models Four additional goodness-of-fit measures used in conjunction with probit-type models are described in section 8.

4.1.6 Univariate random numbers

Random `mydist.Random` requests a random number from the distribution. `mydist.Random(N)` requests N random numbers from the distribution.

RndLnLikelihood `mydist.RndLnLikelihood(NObservations)` generates a random sample of NObservations data values, and then computes the LnLikelihood value for this sample. Of course, RndLnLikelihood returns a different value each time you call it, because it generates a new random sample each time.

4.1.7 Multivariate random numbers

Some preliminary, experimental support is provided for multivariate random number generation with a Gaussian copula and arbitrary marginals, based on a class called `RandGen.m`. See `DemoRandGen.m` for a short demonstration, including minimal documentation.

4.2 Functions of Two Distributions

CUPID can compute a few functions that depend on two distributions.

4.2.1 Delta

```
[mean, delta] = Delta(mydist1,mydist2,[p values])
```

For each of a vector of p values, $0 < p(i) < 1$, this function computes the mean and difference (delta) of the scores in the two distributions with those cumulative p values. If the vector of p values is not specified, the default is 0.005:0.01:0.995.

4.2.2 PrXGTY

```
thisp = PrXGTY(mydist1,mydist2)
```

computes the probability that a randomly selected observation from the first distribution is larger than a randomly selected observation from the second distribution (assuming independence). For example,

```
thisp = PrXGTY(Normal(0,1),Uniform(0,1))
```

produces 0.31563.

4.2.3 BhattDist

```
thisdist = BhattDist(mydist1,mydist2)
```

computes the Bhattacharyya distance between the two distributions.

4.2.4 Gini

```
GiniVal = Gini(Dist,[Dist2])
```

Compute the Gini index for a distribution or pair of distributions. If a single Dist is specified, the Gini index is computed between that distribution and a uniform distribution over the same range. Scores of 1/2 indicate that the distribution or difference between pair of distributions is flat across percentiles. Scores less than 1/2 indicate that most of the distribution or difference between a pair of distributions is in the upper tail. Scores greater than 1/2 indicate that most of the distribution or difference is in the lower tail.

4.2.5 HellingDist

```
thisdist = HellingDist(mydist1,mydist2)
```

computes the Hellinger distance between the two distributions.

4.2.6 Lorenz

```
Lp = Lorenz(p,Dist,[Dist2])
```

Compute the Lorenz function which is the integrated difference between two distributions up to a given cumulative probability p (Dist1 minus Dist2). If only one distribution is provided, Dist2 defaults to the uniform over the same range as Dist.

4.3 Other Functionality

CUPID classes support a variety of other house-keeping-type functions that can be useful. Some of these are documented here.

4.3.1 A Shortcut for Changing Parameter Values

In some cases you may want to change some parameters of a distribution but not want to retype the whole thing. For example, suppose you initially asked for

```
mydist=Convolution(TruncatedX(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 `mydist.ResetSomeParms(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. You can specify several pairs to reset several parms at the same time. For example, this would reset the first parameter to 0.033 and the seventh to .15: `mydist.ResetSomeParms(1,0.033,7,.15)`.

4.3.2 MakeBinSet(MinPr)

This function creates an output vector whose values are the tops of bins covering the RV's range, where each bin has a probability of at least `MinPr`. Thus, the last value in the vector is always the upper bound of the distribution. (The bottom of the first bin is implicitly the distribution's lower bound, and it is not included in the output vector.) As an illustration,

```
a=Uniform(0,1).MakeBinSet(0.1)
```

generates the output vector 0.1:0.1:1.0. Also,

```
Normal(0,1).MakeBinSet(1/5)
```

generates a vector of the normal distribution values at the percentiles of 20, 40, 60, 80, and 100.

4.3.3 FnAfterReset

In some cases it is useful to be able to compute values of the same function for a whole variety of parameter values. This can of course be done "manually" by re-initializing the distribution for each new set of parameter values and computing the desired function. One sometimes-convenient short-cut is built into the CUPID classes, though:

```
mydist.FnAfterReset(ParmNo,ParmValues,'FunctionName'
```

ParmNo is a number indicating which parameter should be adjusted—that is, the first, second, third, etc. ParmValues is a vector of different values that the parameter should take on, and the string FunctionName is the function that is to be computed. As an illustration,

```
Uniform(0,1).FnAfterReset(2,1:10,'Mean')
```

produces as output the means of Uniform distributions with the second parameter (i.e., upper bound) ranging from 1 to 10 in steps of one. That is, it produces the vector 0.5:0.5:5.0.

4.3.4 Plotting

For the most part you would make plots as usual in MATLAB, but there is one short-cut for all of the CUPID distributions: `mydist.PlotDens` plots the PDF and CDF of mydist.

The utility function `plotDists` can also be used to plot the PDFs or CDFs of multiple distributions on the same (sub)plot to facilitate comparisons between distributions.

```
plotDists(Dists,PdfCdf)
```

The first parameter is a cell array of the distributions to be plotted, and the second parameter is 1 for PDFs or 2 for CDFs..

5 Parameter Estimation

CUPID distributions can also adjust their parameter values in order to satisfy certain criteria (e.g., maximum likelihood, moment matching, etc), as described in detail below. The following general points apply to all of the parameter estimation procedures, but you may want to find the one you want first and then come back to these general points.

5.1 General points about parameter estimation procedures

Each of the CUPID parameter estimation routines described below works by calling MATLAB's `fminsearch`. This is important to know because `fminsearch` accepts various optional control values, and you may want to change these control values depending on your particular application. You can set new control values like this:

```
mydist = Beta(1.4,1.5);
mydist.SearchOptions.MaxFunEvals = 500;
```

This works because `mydist.SearchOptions` is a set of control values initialized by MATLAB's `optimset` function, and you can change these default values to anything you want. CUPID distributions pass their control value settings (e.g., `mydist.SearchOptions`) to `fminsearch` when they call it.

Caution: The parameter estimation processes are in real danger of getting caught in local minima, so they are not guaranteed to find the overall best parameter values. In my experience, the local minimum problem is especially bad when the starting point of the search (i.e., the current distribution parameters) is far from the correct parameter values. One way to address this problem is to start the parameter search from many different points, hopefully intelligently chosen to be likely near the true values. The function `mydist.EstManyStarts`, which is described at the end, can be used to automate this process. Also, in the case of maximum-likelihood estimation, I have started trying to provide distributions with functions to choose intelligent starting parameters from the data—see `StartParmsMLE`, described briefly below.

In some cases, CUPID will bomb during the estimation process due to numerical errors. When this happens, try harder to find starting parameter values in the general neighborhood of the best parameter estimates.

5.2 Estimate from moments

This function tells CUPID to alter the parameter values so as to optimize the fit to a given set of moments. In general, the syntax is:

```
mydist.EstMom(ObsMoms[,ParmCodes])
```

ObsMoms is a vector of the observed central moments (i.e., M1 is the mean, M2 is the variance, M3 is 3rd central moment, ...) for which you want to find the best-fitting parameters. You may specify as many moments as you like. Currently, the routine minimizes the sum of squared errors between predicted and observed moments. ParmCodes is an optional parameter described below.

For example, if you create `mydist=RNGamma(3,.4);` and you want the parameters changed so that the mean and variance are 100 and 40000, type:

```
mydist.EstMom([100,40000])
```

After a brief search, CUPID will change mydist to a RNGamma(0.25,0.0025), which has the desired mean and variance.

If you use a value of 'nan' for one of the observed moments, the routine will skip that moment and only try to fit the others. For example, if you create `mydist=Beta(3,20);` and you want the parameters changed so that the variance and third moment are 0.005 and 0.0003, type:

```
mydist.EstMom([nan,0.005,0.0003])
```

The result is Beta(3.0136,19.3265).

5.3 Estimate from ChiSquare

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

```
mydist.EstChiSq(BinUpperBounds,BinProbs[,ParmCodes])
```

BinUpperBounds is a vector of the upper bounds of the bins to be considered, and these must be in ascending order (the lowest bin is assumed to extend down to the distribution's minimum). BinProbs is a vector of the same length, and it contains the observed proportions in each bin. ParmCodes is an optional parameter described below. See the function "GofFChiSq" for further details.

For example:

```
mydist = Beta(1.4,1.5);
B = .1:.1:1;  % Values of the beta RV
O = [.1 .11 .09 .1 .12 .08 .1 .1 .09 .11];
mydist.EstChiSq(B,O)
```

After adjustment, mydist is Beta(0.97246,0.97212).

5.4 Estimate from percentiles

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

```
mydist.EstPctile(XValues,TargetCDFs[,ParmCodes])
```

XValues is a (sorted) vector of possible values of the random variable, and the TargetCDFs is the desired CDF value for each of those XValues. You may specify as many XValues 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.

For example:

```
mydist = Beta(2.4,1.5);
B = .1:.1:.9;  % Values of the beta RV
P = [.1 .21 .29 .30 .42 .48 .5 .7 .99];
mydist.EstPctile(B,P)
```

After adjustment, mydist is Beta(0.99598,0.77969).

5.5 Maximum likelihood estimation

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:

```
mydist.EstML(Obs[,ParmCodes])
```

`Obs` is a vector of data values used for the likelihood calculation.

For example:

```
mydist = Lognormal(2.4,1.5);
X = mydist.Random(100,1);  % Generate 100 random numbers
mydist.EstML(X)            % Estimate the parameters of mydist by maximum
    likelihood
[SEs, Covars] = mydist.MLSE(X);  % Use Fisher information to get the SEs of
    the estimates.
```

After adjustment, mydist will have slightly different parameters.

After calling `EstML`, the command `MLSE` produces standard errors and covariances of the parameter estimates, computed via Fisher information.

In July 2021, I started introducing functions to choose (hopefully!) reasonable starting parameter values for the ML search from the vector of observations. This is done with a function called `StartParmsMLE`, but so far this function has only been provided for a small subset of the distributions. When you `EstML` with a distribution for which this function has not yet provided, you get a warning message that the search is starting at the current parameter values, just like it did before. If you have good functions to choose starting parameter values for additional distributions, please give them to me and I will happily add them.

5.6 Maximum likelihood estimation with censored data

This function is like `EstML` except that it is used with data sets where known numbers of observations have been lost because they were outside of some observable range. The syntax is:

```
mydist.EstMLcensored(Obs,Bounds,nsTooExtreme[,ParmCodes])
```

`Obs` is the vector of observed data values within the observable range. `Bounds` is a 2-item vector giving the minimum and maximum values of the observable range. `nsTooExtreme` is a 2-item vector giving the numbers of observations that were lost because of being less than the minimum bound or greater than the maximum bound.

For example:

```
mydist = Normal(100,10);    % a true distribution
X = mydist.Random(1000,1);  % Generate random numbers
Bounds = [80, 120];     % Suppose that only values in this range can be
    observed
```

```
censoredX = X( (X>=Bounds(1)) & (X<=Bounds(2)) ); % Here are your actual
    observed data
nTooLow = sum(X<Bounds(1)); % How many data points you lost because they were
     too small
nTooHi = sum(X>Bounds(2)); % How many data points you lost because they were
     too large
mydist.EstMLcensored(censoredX, Bounds, [nTooLow, nTooHi]) %
    Estimate the parameters of mydist by maximum likelihood
[SEs, Covars] = mydist.MLSEcensored(X, Bounds, [nTooLow, nTooHi]); % Use
    Fisher information to get the SEs of the estimates.
```

After adjustment, mydist will have slightly different parameters.

After calling `EstMLcensored`, the command `MLSEcensored` produces standard errors and covariances of the parameter estimates, computed via Fisher information.

5.7 Interval probability estimation

This function tells CUPID to adjust the parameter values to produce a certain desired percentage in a given interval. The syntax is:

`mydist.EstPctBounds(LowerBound,UpperBound,TargetProb[,ParmCodes])`

The bounds define the interval to be considered, and the TargetProb is the desired probability within that interval. Note that for many distributions there are lots of different parameter combinations that yield a given TargetProb within the indicated bounds, and it is unpredictable which combination will be found by the estimation process.

For example:

```
mydist = Lognormal(2.4,1.5);
mydist.EstPctBounds(2,10,.3)
```

After adjustment, mydist will be Lognormal(2.5078,1.7811), for which mydist.CDF(10) - mydist.CDF(2) = 0.3.

5.8 ParmCodes

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

f The parameter is fixed at the current value, 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 legal real value.

If ParmCodes is not specified, reasonable default values are assumed (usually 'r'; see below).

Here are some examples of how to use ParmCodes:

- 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 `mydist=Beta(0.5,0.5)`. Then, `mydist.EstMom([0.3,0.1],'` adjusts the two parameters to produce the desired mean and variance. The distribution mydist will be changed to Beta(0.33,0.77), which has the desired moments.
- Alternatively, suppose you want to keep the second parameter of the Beta fixed at 0.5, and just adjust the first parameter to come as close as possible to the desired mean and variance. Fix the second parameter by using 'f' as the second parmcode: `mydist.EstMom([0.3,0.1],'rf')`, and the result is Beta(0.21059,0.5), with mean and variance of .29 and .12, respectively.

- Use 'i' to restrict parameter values to integers:

```
mydist=RNGamma(3.5,10.5);
mydist.EstML([0.34162, 0.52264, 0.35699, 0.40554, 0.34145, 0.37642],'ii')
```

  The parameters are set to 44 and 113.

- Suppose you create a standard normal distribution truncated at -1 and 1:

```
mydist=TruncatedX(Normal(0,1),-1,1)
```

  The command

```
mydist.EstMom([0.3,0.22],'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. CUPID sets the normal mean and standard deviation to 0.68137 and 0.74865, respectively, yielding 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."

- Suppose you create the distribution:

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

  In this case, the command

```
mydist.EstML([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.

You can check the default parameters for any distribution by examining `mydist.DefaultParmCodes`.

5.9 EstManyStarts

This function is used to search for parameter estimates using multiple different starting points. It can be used with any of the estimation functions (i.e., EstML, EstMLcensored, EstMom, EstChiSq, etc). As is illustrated with examples in `DemoEstManyStarts.m`, you must pass in to `EstManyStarts` both the function handle of the estimation function and a cell array of the parameters needed by the function. You also pass in to `EstManyStarts` a 2-dimensional array indicating the different starting points that you would like to try. Each row of this array corresponds to one starting point, and the different columns correspond to the different distribution parameters. For example, the starting point array element (i,j) is the starting value for the j'th distribution parameter at the i'th search starting point. The function `EstManyStarts` returns the best-fitting set of parameters that it finds across all of the different starting points.

6 Parameter Estimation With Constraints

This section describes more flexible parameter estimation techniques that can be used in situations not conveniently handled with the methods described in section 5. For demos, see `DemoConstrained.m`.

In one such situation you want to estimate the parameters of a single distribution imposing some constraints on the estimated values. For example, suppose you have a set of data that are thought to be normally distributed, and you would like to estimate μ and σ. But you also have prior information suggesting $\sigma \geq 0.1 \cdot \mu$, so you would like the estimates to satisfy this constraint. (This is Example 0 in `DemoConstrained.m`.)

In another situation you want to estimate the parameters of two or more distributions imposing constraints that somehow refer to several of the distributions. For example, suppose you have two sets of data that you think are normally distributed with potentially different means but equal variances; you would like to estimate μ_1, μ_2, and the common σ. (This is Example 1 in `DemoConstrained.m`.)

6.1 function `FitConstrained`

The general-purpose function `FitConstrained.m` can be used to estimate parameter values subject to arbitrary user-specified constraints in the above situations and many others. This function is actually a kind of interface to MATLAB's `fminsearch`—which does the actual searching—with `FitConstrained.m` just providing a framework that facilitates incorporating constraints into the search.

The function header is:

```
function [Dists, ErrScores, Penalty] =
FitConstrained(Dists,Datasets,sErrorFn,LinkFn,StartingVals)
```

Below are brief descriptions of the inputs and outputs, and the following subsections provide some further information. These descriptions are pretty generic, though, and it may well be easier to figure out how to use this function by looking at `DemoFitConstrained` and its associated `DemoFitConstraintFnXX` functions.

`Dists` This is a cell array of the Cupid probability distribution(s) that are to be fit. For the input values of `Dists`, the parameter values of the distributions are completely irrelevant. In the output values of `Dists`, the parameter values have been adjusted to the best values that `fminsearch` could find.

`Datasets` This is a cell array of the datasets to which the distributions are to be fit, and there is one cell array element (i.e., dataset) for each probability distribution. The structure of each cell array element (i.e., dataset) depends on the error function being minimized (see next parameter). In the simplest case of fitting by maximum likelihood [i.e., minimizing -Ln(Likelihood)], for example, the cell array for each distribution is just a vector of observations from that distribution. Other cases will be described later.

`sErrorFn` This is a string naming the function used to measure the error in the fit of Dists{i} to Datasets{i}. The string specifies one of the error functions defined in `dGeneric.m`, so it should be one of these options: '-LnLikelihood', 'MomentError', 'GofFChiSq', 'PercentileError', '-YNProbitLnLikelihood', 'YNProbitChiSq', '-mAFCProbitLnLikelihood', 'mAFCProbitChiS'. Note that the initial minus signs in some of these names indicate that their function values should be multiplied by -1, and the result minimized.

`ConstraintFn` This is a user-written function that enforces the parameter constraints. For an example, see:

```
function [Dists, Penalty] = DemoFitConstraintFn0(Dists, Parms)
```

The function takes as its inputs the cell array of the distributions being fit, and the vector of parameter values that is currently being suggested by `fminsearch`. The function outputs are (1) the cell array of distributions with their parameters adjusted according to the suggested values `Parms`, and (2) the size of a penalty to be applied, which is set to a value greater than zero if the current parameter values do not completely satisfy the desired constraints.

`StartingVals` This is a vector of the initial values of the parameters at which `fminsearch` will start. Note that these will also be the first parameter values passed to `ConstraintFn`.

`ErrScores` **and** `Penalty` These outputs are only useful occasionally. `ErrScores` is a vector with one score per `Dist`, with each score giving the value of `sErrorFn` for that `Dist` with the best parameter values. `Penalty` is a value computed by the user-supplied `ConstraintFn`; it will be zero if the user-defined constraint is completely satisfied by the best fit, and it will be greater than zero to the extent that the user-defined constraint is violated.

6.2 Datasets

The specification of the `Datasets` parameter is complicated by the fact that the different error functions require different data as input. For example, to estimate by maximum likelihood or minimum chi-square requires calling one of the these two functions to evaluate the current parameter set:

```
LnLikelihood(Observations)
GofFChiSq(BinUpperBounds,BinProbs)
```

Note that the relevant dataset for LnLikelihood has just one parameter: Observations. In contrast, the relevant dataset for GofFChiSq has two parameters: BinUpperBounds and BinProbs. Thus, the cell array `Datasets` that is passed to `FitConstrained` needs to be flexible enough to handle a different number of parameters depending on the function to be minimized.

To provide this flexibility, the `Datasets` parameter is passed as a cell array. `Datasets{1}` contains all of the parameters needed to evaluate the error function for the first distribution being fitted, `Datasets{2}` contains all of the parameters needed to evaluate the error function for the second distribution being fitted, and so on.

If the error function requires only one parameter (e.g., `LnLikelihood`), then `Datasets{i}` can simply be set equal to this parameter. See `DemoFitConstrained` Examples 0 and 1 for examples.

If the error function requires more than one parameter (e.g., `GofFChiSq`), then, for each distribution i, `Datasets{i}{1}` must be set equal to the first parameter needed by the error function, `Datasets{i}{2}` must be set equal to the second parameter needed by the error function, and so on. See `DemoFitConstrained` Example 3 for an example.

6.3 Constraint functions

You must write your own constraint function to implement the constraints that you would like satisfied. For examples, see the functions `DemoFitConstraintFnXX`. In brief, the first input to this function, `Dists`, is a cell array of the distributions that are being fit. The second input to this function, `Parms`, is a vector of the parameter values that `fminsearch` is currently considering (starting with the `StartingVals` that you provided).

Your constraint function must produce two outputs. The first output is a cell array with the distributions being fit, each having its new parameter values as computed from the values suggested by `fminsearch`. These parameter values are computed from the values in `Parms` according to your constraints.

The second output is a single real number, `Penalty`, which is to be added to the overall error score computed from the current set of parameter values. The idea is to add a penalty whenever `fminsearch` suggests parameter values that do not satisfy your constraints (see `DemoFitConstraintFn0` for an example). It is a good idea to make the penalty larger when the suggested parameter values are farther from meeting your constraints and to reduce the penalty when the suggested parameter values only violate the constraints by a little bit; this helps `fminsearch` converge more quickly on satisfactory parameter values.

7 Outlier analysis

CUPID can be used to model distributions contaminated with outliers via the special-purpose class `clOutlierModel`. In brief, the model assumes that there are `NConds` conditions, each associated with its own distribution of true data values described by `NConds` arbitrary CUPID random variables. According to the model, true data

values in each condition are contaminated—with some probability that is constant across conditions—by an outlier process described by a further CUPID random variable that I call the "contamination distribution". `clOutlierModel` can be set up to model any one of three different types of contamination:

Shift The contaminated score is formed by summing the value of the true score with the value of a score from the contamination distribution.

Stretch The contaminated score is formed by multiplying the value of the true score times the value of a score from the contamination distribution. (This option only works when all true and contamination RVs are strictly positive, however.)

Replace The contaminated score is formed by replacing the value of the true score with the value of a score from the contamination distribution.

`clOutlierModel` instantiates the observed distribution of each condition as a CUPID mixture of true and contaminated scores, so it possible to examine the PDF, CDF, mean, etc of the predicted observed scores. `clOutlierModel` also provides an `EstML` routine to find the best-fitting (maximum likelihood) values of its parameters for a given data set.

For a demo, see `clOutlierModelDemo.m`.

8 Probit Analysis

This section describes the use of CUPID in probit analysis. In brief, probit analysis involves estimating the parameters of a probability distribution (most often the normal, but any distribution is possible) from a specific type of data described below. CUPID allows you to estimate the parameters of any of its distributions from such data, with estimation procedures analogous to those described in section 5. Alternatively, the user may obtain nonparametric estimates using the Spearman-Kärber method (e.g., Epstein and Churchman, 1944; Kärber, 1931; Spearman, 1908). Based on an extensive simulation study, in fact, Miller and Ulrich (2001) recommended that the Spearman-Kärber method be used under a wide variety of circumstances (also see Ulrich and Miller, 2004).

To set the symbols and terminology, the first two subsections outline two slightly different experimental paradigms producing data typically examined with probit analysis. These will be referred to here as "yes/no paradigms" and "mAFC paradigms", based on the terminology used in the field of psychophysics. For more detailed descriptions, see, for example, Miller and Ulrich (2001, 2004) and Ulrich and Miller (2004).

The third subsection describes how CUPID can be used to estimate distribution parameters from the data from these paradigms. For a demo, see `DemoProbit.m`.

8.1 Yes/no paradigms

Yes/no paradigms can be characterized as follows:

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
$$Y_{ij} = \begin{cases} 0 & \text{if } X_{ij} > C_i \\ 1 & \text{if } X_{ij} \leq C_i \end{cases}$$

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

4. Probit analysis requires estimating the parameters of the probability distribution of the X_{ij} values from the k observed proportions, G_i/N_i, i=1...k.

Bioassays provide one concrete example of yes/no paradigms (e.g., Finney, 1978). For example, suppose a researcher wants to determine the dosage of insecticide needed to kill a particular type of insect. He selects various dosages to test; these are the C_i values. The researcher then applies each dosage separately to N_i insects, and observes whether each one dies (i.e., $Y_{ij} = 1$) or not (i.e., $Y_{ij} = 0$), for $j = 1 \ldots N_i$. As one would expect, the probability of $Y_{ij} = 1$ increases with C_i. Typically, the researcher assumes that each insect has a minimum lethal dose, X_{ij} and that the X values are distributed according to some distribution (e.g., normal). Thus, the researcher wants to estimate the parameters of this underlying normal distribution from the observed proportions, G_i/N_i.

With such data, the parameters of the distribution of the X_{ij} values can be estimated either by maximizing likelihood or by a minimizing chi-square. For any given set of parameter values, the likelihood of the specific ordered set of Y values is:

$$L = \prod_{i=1}^{k} p_i^{G_i} \cdot (1 - p_i)^{(N_i - G_i)}$$

where $p_i = CDF(C_i)$ with the given parameter values (Finney, 1971, chap. 5).[1] 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).

Alternatively, based on Guilford (1936), for any given set of parameter values, a chi-square goodness-of-fit test may be computed as:[2]

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

where $p_i = CDF(C_i)$ with the given parameter values and $\hat{p}_i = G_i/N_i$. When requested to use the ChiSq type of probit fit, CUPID adjusts parameters iteratively to minimize this value.

8.2 mAFC paradigms

These paradigms are common in psychophysical discrimination experiments, but they do not map well onto the insecticide example. As an alternative example, consider a 2AFC visual detection task. In each trial, an observer views a computer screen during two (the '2' of 2AFC) time intervals (e.g., demarked by tones). The observer knows that a dim visual stimulus will be presented during one interval but not the other, and in each trial he or she is asked to say whether the stimulus was presented in the first or second interval. The brightness of the stimulus is varied from trial to trial (varying C_i values), and for each C_i value the percentage of correct interval responses is tabulated. Note that the true percentage would always be at least 1/2, since the observer would guess when C_i was so weak that the visual stimulus could not be detected at all. In variants of this paradigm the number of intervals could instead be 3, 4, ...; this is the 'm' of mAFC.

[1] To get the likelihood of the G_i values instead, you would include the binomial coefficients $\binom{N_i}{G_i}$ in this product. This has no effect on the parameter estimation, however, because these constant multipliers can be factored out to isolate the kernel of the likelihood function, i.e.,

$$\prod_{i=1}^{k} \binom{N_i}{G_i} \cdot p_i^{G_i} \cdot (1 - p_i)^{(N_i - G_i)} = \prod_{i=1}^{k} \binom{N_i}{G_i} \prod_{i=1}^{k} p_i^{G_i} \cdot (1 - p_i)^{(N_i - G_i)}$$

I thank Rolf Ulrich for supplying the information in this footnote.

[2] 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., CDF(C_i) in a yes/no task or $1/m + (1 - 1/m) \cdot \text{CDF}(C_i)$ 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.

8.3 Parametric Distributions: Functions and Estimation

For more detail, see DemoProbit.m.

YNProbitLikelihood and mAFCProbitLikelihood These two functions compute the likelihood of a set of probit-type data for a given distribution. 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 the yes-no task, the command has the form:

```
mydist.YNProbitLnLikelihood(Cs,Ns,Gs)
```

Cs is a vector of the constants that were used; Ns has the number of tests at each constant; and Gs has the number of responses indicating that the value of X was greater than each Cs.

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

```
mydist.mAFCProbitLnLikelihood(m,Cs,Ns,Gs)
```

except the first parameter is the number of alternative responses, m, and Gs is the number of correct responses at each constant.

Parameter Estimation From Probit Data There are four functions that can be used 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—to a given set of probit-type data (see section 8 for a brief description of probit analysis).

For a yes-no task, the syntax is one of the following options:

```
mydist.EstProbitYNML(Constants,NTrials,NGreater[,ParmCodes])

mydist.EstProbitYNChiSq(Constants,NTrials,NGreater[,ParmCodes])
```

Constants is a vector of stimulus values at which test trials were run. NTrials is an equal-length vector indicating the number of trials at each stimulus value. NGreater is also an equal-length vector, and it indicates the number of trials at each stimulus for which the outcome was "Greater". (Note that each element of NGreater should be less than or equal to the corresponding element of NTrials.) ParmCodes is an optional parameter described below.

There are also two parallel functions for estimating from data obtained in an m-alternative forced-choice task:

```
mydist.EstProbitmAFCML(m,Constants,NTrials,NGreater[,ParmCodes])

mydist.EstProbitmAFCChiSq(m,Constants,NTrials,NGreater[,ParmCodes])
```

The first parameter, m, indicates the number of options in the forced-choice task, and the remaining parameters are parallel to those used for the yes-no task.

8.4 Nonparametric Estimates with the Spearman-Kärber Method

The Spearman-Kärber method provides a nonparametric method of estimating the properties (e.g., moments) of the underlying distribution—that is, a method of estimating these properties without assuming a particular underlying distributional shape (e.g., normal). Within CUPID, this is implemented via the special-purpose "SpearKar" distribution. This distribution is defined by giving an ordered vector of C_i values as the first argument and another vector with the corresponding observed CDF value of each C_i as the second argument. The C_i values must be strictly monotonically increasing, and the corresponding CDF values must also be

increasing, though ties are allowed. Furthermore, C_1 must have a CDF of 0 and the final C_k must have a CDF of 1. One can then compute nonparametric estimates of the distribution mean, variance, etc, with the usual functions.

When using the SpearKar distribution in a situation where the CDF values were estimated from observed response probabilities, it may sometimes happen by chance that the observed probabilities produce estimated CDF values that are nonmonotonic. The function SpearKar.monotonize is provided to handle this case, provided monotonically non-decreasing CDF estimates from the nonmonotonic ones.

See DemoProbit.m for examples.

9 Miscellaneous

An additional function that be useful:

9.1 function `DistRename`

This function can be used to change the name of the current distribution to any desired string. For example, `mydist1.DistRename('ShiftedBeta')` changes the name of the distribution to `ShiftedBeta`. This name will be retained even if parameters are estimated (which normally resets the distribution name to show the new parameters).

10 Spline Approximations

Warning: The spline functionality is highly experimental.

Any combination of the PDF, CDF, and InverseCDF functions of any distribution can be approximated with splines, and this functionality might be useful with distributions for which the computation of one or more of these functions is slow and is performed often. For example, to start using the spline approximation to the PDF of a distribution, the command is:

```
mydist.UseSplinePDFOn(NBinsOrListX)
```

The parameter NBinsOrListX is used to determine a set of X values at which the PDF is to be evaluated via the regular PDF function (perhaps slowly). Once the PDFs of these Xs have been computed, all further PDF values are computed as spline approximations from them, until the command `mydist.UseSplinePDFOff` is given.

NBinsOrListX can be either a single number or a vector. If it is a single number, this number should be an integer indicating the number of Xs at which the PDF should be evaluated. A list of NBinsOrList Xs is then constructed ranging from mydist.LowerBound to mydist.UpperBound in equal increments.

If NBinsOrListX is a vector, then it is the list of X values at which the PDF is to be evaluated.

There are exactly parallel commands UseSplineCDFOn and UseSplineCDFOff for handling spline estimation of the CDF function, and parallel commands UseSplineInvCDFOn and UseSplineInvCDFOff for handling spline estimation of the InverseCDF function.

Here is an example in which the spline approximation reduces the computation time by about 25% with no visible change in results:

```
mydist = InfMix(Normal(0,1),RNGamma(3,.2),1);
tic
mydist.PlotDens
toc

mydist = InfMix(Normal(0,1),RNGamma(3,.2),1);
tic
mydist.UseSplinePDFOn(50);
mydist.PlotDens
toc
```

Spline approximations are especially helpful with nested and recursive distribution definitions. For example:

```
% Recursive definition of a convolution of three random variables:
mydoubleconv1 = Convolution(Normal(0,50),Convolution(Uniform(0,100),Gamma
    (3,0.01)));
% mydoubleconv1.PlotDens;  % This would take hours
%
% Now the same definition with a spline approximation
% of the inner part:
mypart = Convolution(Uniform(0,100),Gamma(3,0.01));
mypart.UseSplinePDFOn(100);
mydoubleconv2 = Convolution(Normal(0,50),mypart);
mydoubleconv2.PlotDens;  % This only takes a few seconds
```

11 Programming with CUPID Handle Objects

Be aware that CUPID distributions are handle objects. This can produce some surprises if you forget (or just don't know) how handle objects work. The basic point to keep in mind is that "when you use the same distribution in two different cases, adjusting its parameters in one case will change its parameters in the other case also."

For example:

```
MyNorm = Normal(0,1);
MyWinner = Order(1,MyNorm,Exponential(1));
[MyWinner.Mean MyWinner.SD]    % MyWinner has a mean & sd of -0.16052 and
    0.8224.
MyNorm.ResetSomeParms(1,2);    % Change MyNorm's mean to 2
[MyWinner.Mean MyWinner.SD]    % MyWinner now has a mean & SD of 0.78103 and
    0.70097
```

Changing MyNorm also changed MyWinner, because MyWinner used MyNorm as one of its components. Worse yet, the new values reported by MyWinner may be wrong (though they are not in this case). The reason is that MyWinner actually needs to be reinitialized when one of its underlying distributions changes; for example, its upper and lower bounds might change, leading to different integration results.

Because of such complications, it is best to avoid re-using distributions if there is any chance at all that you will be changing their parameter values. For example, the complications above would be avoided if one defined

```
MyWinner = Order(1,Normal(0,1),Exponential(1));
```

because in this case MyWinner has its own separate copy of a normal distribution and would therefore be unaffected by changes to MyNorm.

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 `mydist.Minimum` and `mydist.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 some small nonzero value 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 (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 parameter search 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 with MATLAB's numerical integration routines.

13.2 Parameter searching

If no explicit formulas are available, a variant of MATLAB's built-in fminsearch is used. The variant is my fminsearcharb function, available on GitHub, which can be helpful in allowing constraints on the parameters (e.g., standard deviation of a normal must be positive).

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 MATLAB functions or 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, ex-Gaussian, lognormal, and Weibull distributions, plus all of the transformed and derived distributions. Finally, for other distributions CUPID generates a uniform random number between zero and one as the CDF, and then works backward with InverseCDF to find the number with that desired CDF. It would not be difficult to incorporate more routines from Devroye (1986) for distributions where the InverseCDF procedure is too slow or too inaccurate to give satisfactory results.

14 Testing

Because of the possibility of numerical (and programming!) errors, it is a good idea to check the computations with any distribution before relying on it. Since the errors can depend very strongly on the parameter values, be sure that you check with parameter values in the range that is of interest to you. Checking can be done in at least two ways.

1. Many tests can be performed using MATLAB's unit testing framework. This framework is very powerful but also somewhat complicated. If you want to use it but are not already familiar with it, you will need to consult the MATLAB documentation (i.e., it is not described here).

For those familiar with unit testing, the UnitTests folder has many examples of distributional tests using this framework. This folder contains a file named something like "utDist.m" for each distribution "Dist." If you want to test an existing distribution with your particular parameter values, you start by modifying the existing ut*.m file. If you want to test a new distribution that you have created, you must create a new ut*.m file for that new distribution. Hopefully, you can do that by modifying the ut*.m file for a similar existing distribution.

After you have modified or created the ut*.m file for the distribution you want to test, you need to run the unit tests. The script SingleTest.m gives a small example showing how to run a set of tests for one distribution. Right now, SingleTest.m is set up for testing the Uniform distribution, but you can easily modify it to test any desired distribution (and in some other ways) within the indicated OPTIONS section of the script.

2. CupiTest is a special-purpose MATLAB script designed to check the computations for any of CUPID's distribution classes, and it can sometimes uncover both numerical problems and programming errors. In brief, CupiTest checks any given distribution by computing functions in at least two ways (e.g., computing the CDF directly versus by numerical integration of the PDF) and comparing 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 the MATLAB command window with a parameter indicating the distribution you want to check, like this:

   ```
   CupiTest(Normal(0,1))
   ```

   I hope the output file is reasonably self-explanatory; suspicious results are marked with asterisks.

   CupiTest has many option input control parameters, which it will list out for you if you call it with no parameters at all. These are undocumented, but I will be happy to answer questions about what they do, if anyone gets that far.

15 Contributing to CUPID

Of course, I would be delighted for others to contribute to CUPID. Use the GitHub repo if you know how, and email me if you do not. If you are interested in contributing a new distribution, look at the code for a similar distribution to see what needs to be provided (Uniform.m is a good example) and what conventions need to be followed. Then, program your new distribution following these conventions, send it to me, and I'll be happy to add it to the set.

16 Release History

CUPID started out in object-oriented Turbo Pascal in the late 1980's.. Several versions of it were released as stand-alone DOS programs between about 1996–2006 (see, e.g., Miller, 1998b,a, 2006; Miller and Ulrich, 2004). This was never very satisfactory, however, because the routines fit much better into a programmer's toolbox than a stand-alone application.

Porting into MATLAB started in 2016, with the goal of producing a GitHub repo eventually. As can be seen by examining the m files, a lot of the port has been done simply be reproducing the Pascal structure within MATLAB as closely as possible. I hope to clean up the code, gradually, into something that looks more like "real MATLAB", and of course would welcome any help with that.

CUPID was first provided as a MATLAB toolbox package in 2019.

17 Acknowledgements

- The utility `allcomb.m` (c) was written by Jos van der Geest and downloaded from MATLAB File Exchange.

- The Studentized range distribution uses the function `cdfTukey(q,v,r)` by Peter Nagy, obtained from https://au.mathworks.com/matlabcentral/fileexchange/37450 on 9 April 2018.
- The SkewNor distribution CDF is computed with the MATLAB version of Owen's T function from John Burkardt, obtained from http://people.sc.fsu.edu/ jburkardt/m_src/asa005/tfn.m on 24 May 2019.
- The SpearKar monotonize function was provided by Rolf Ulrich and Karin Bausenhart.
- David Goodmanson provided crucial hints on how to implement ConvolveFFTc (see https://www.mathworks.com/matla request-for-help-computing-convolution-of-random-variables-via-fft).

References

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

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

Epstein, B. and Churchman, C. W. (1944). On the statistics of sensitivity data. *Annals of Mathematical Statistics*, 15:90–96.

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

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

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

Geary, R. C. (1947). Testing for normality. *Biometrika*, 34(3/4):209–242.

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

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

Hoaglin, D. C. (1985). Summarizing shape numerically: The g-and-h distributions. In Hoaglin, D. C., Mosteller, F., and Tukey, J. W., editors, *Exploring data tables, trends, and shapes.*, chapter 11, pages 461–513. Wiley, New York, NY.

Johnson, N. L. (1949). Systems of frequency curves generated by methods of translation. *Biometrika*, 36:149–176.

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

Johnson, N. L., Kotz, S., and Balakrishnan, N. (1994). *Continuous univariate distributions.* Wiley, New York, NY.

Kärber, G. (1931). Beiträg zur kollektiven Behandlung pharmakologischer Reihenversuche. [A contribution to the collective treatment of a pharmacological experimental series.]. *Archiv für experimentelle Pathologie und Pharmakologie*, 162:480–483.

Konishi, S. (1978). An approximation to the distribution of the sample correlation coefficient. *Biometrika*, 65:654–656.

Luce, R. D. (1986). *Response times: Their role in inferring elementary mental organization.* Oxford University Press, Oxford, England.

Miller, J. O. (1998a). Bivar: A program for generating correlated random numbers. *Behavior Research Methods, Instruments & Computers*, 30(4):720–723.

Miller, J. O. (1998b). Cupid: A program for computations with probability distributions. *Behavior Research Methods, Instruments & Computers*, 30(3):544–545.

Miller, J. O. (2006). A likelihood ratio test for mixture effects. *Behavior Research Methods*, 38(1):92–106.

Miller, J. O. and Ulrich, R. (2001). On the analysis of psychometric functions: The Spearman-Kärber method. *Perception & Psychophysics*, 63(8):1399–1420.

Miller, J. O. and Ulrich, R. (2004). A computer program for Spearman-Kärber and probit analysis of psychometric function data. *Behavior Research Methods, Instruments & Computers*, 36(1):11–16.

Mineo, A. M. and Ruggieri, M. (2005). A software tool for the exponential power distribution: The normalp package. *Journal of Statistical Software*, 12(4).

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

Ruckdeschel, P. and Kohl, M. (2014). General purpose convolution algorithm in S4-classes by means of FFT. *Journal of Statistical Software*, 59(4):1–25.

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 & Methods*, 31:2113–2121.

Schwarz, W. (2003). Stochastic cascade processes as a model of multi-stage concurrent information processing. *Acta Psychologica*, 113:231–261.

Simard, R. and L'Ecuyer, P. (2011). Computing the two-sided Kolmogorov-Smirnov distribution. *Journal of Statistical Software*, 39(11):1–18.

Spearman, C. (1908). The method of "right and wrong cases" ("constant stimuli") without Gauss's formulae. *British Journal of Psychology*, 2:227–242.

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

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

Ulrich, R. and Miller, J. O. (2004). Threshold estimation in two-alternative forced-choice (2AFC) tasks: The Spearman-Kärber method. *Perception & Psychophysics*, 66(3):517–533.