DistFit: A Program to Fit Probability Distributions to Data Sets

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

DistFit Version 1.1 — Cupid Version 2.1— Documentation Version July 29, 2005

If you use this software, please register it by sending me your email address, as described in section 11. There is no charge to register, and your email address will be used only to notify you of program bugs or updates.

Contents

1 Overview

DistFit is a general-purpose program to fit probability distributions to sets of observed data values. It can be used for answering such questions as “Are my data better fit by a normal distribution or a gamma distribution?” or “Are my data consistent with the assumption of a lognormal distribution?” or “What are the best-fitting logistic distribution parameter values for my data set?”

DistFit can fit many different kinds of probability distributions to data sets; the alternative distributions are listed and briefly described in section 8. It can fit these distributions (i.e., estimate their parameters) either by maximizing the likelihood of a set of data points, by minimizing the chi-square of observed versus predicted frequencies in data bins, by trying to match a set of observed moments of the data set (i.e., mean, variance, etc), or by trying to match certain observed percentiles.

2 Detailed Introduction

Researchers are sometimes interested in knowing which probability distribution best fits an observed data set. To find out, they must fit the various candidate distributions to the observed data set and evaluate the goodness-of-fit of each.[1] DistFit can help with this.

DistFit can fit many different probability distributions, including the beta, logistic, lognormal, uniform, exponential, gamma, Wald, and Weibull distributions, among others. It can also fit various distributions derived from these basic distributions, including convolutions, mixture distributions, distributions of order statistics, truncated (censored) distributions, etc. Section 8 has more information about the possible probability distributions that can be fit.

2.1 Parameter Estimation Methods

DistFit can fit distributions with four different parameter estimation methods: maximum likelihood, minimum chi-square, moments, and percentile matching. These different methods are appropriate for different types of data sets. Here are detailed descriptions of the four methods, in what I believe to be the order of most-used to least-used:

Maximum Likelihood Estimation For a given probability distribution, the likelihood (L) of a set of observed data values is

$$L = \prod_{i=1}^{N} f(X_i)$$

where f is the distribution's probability density function (PDF), and the values X_i, $i = 1 \ldots N$, are the observed data values. For maximum likelihood estimation, the parameters of the distribution are chosen to be whatever parameter values maximize the value of L for the given data set. Thus, this estimation method uses all of the observed data values.

Minimum Chi-square Estimation As is explained in section 5.3.4, the "chi-square goodness of fit test" is a standard way of assessing the fit of a predicted theoretical distribution to an observed data set. For this test, the observed data set is divided into K mutually exclusive and exhaustive bins, and the number of observed scores in each bin is simply counted. This tabulation of the data is called a "histogram." Then, this observed histogram is compared against a theoretically predicted histogram (i.e., bin counts) in order to compute a measure of the deviation between the predicted and observed distributions. This measure is called the observed chi-square value, χ^2_o, with larger values indicating larger discrepancies between the predicted and observed distributions. For minimum chi-square estimation, then, the parameter values of the distribution are chosen to be whichever ones produce the smallest value of χ^2_o. Thus, this estimation method uses a histogram of the observed data values (i.e., a count of the number of data values in each of a set of distinct bins).

Moment-based Estimation With this estimation method, the distribution's parameter values are chosen to match the observed data (as closely as possible) with respect to the mean (first moment), variance (second moment), and possibly 3rd, 4th, etc *central* moments as well. Thus, this estimation method uses only these moment-based summaries of the observed data set.

Percentile Matching Estimation With this estimation method, the distribution's parameter values are chosen to match the observed data (as closely as possible) with respect to the values at a certain

---

[1] Throughout this documentation, "to fit a probability distribution" means "to estimate the parameters of that distribution most consistent with the observed data values."

specified set of percentiles of the distribution. Thus, this estimation method uses only a set of percentiles of the observed data set.

2.2 A Simple Example

A simple example will illustrate how DistFit operates. First, you use a plain-text editor (e.g., Notepad) to prepare an input file describing the fitting problem in terms of DistFit's control parameters described in this documentation. Table 1 shows an example of such a file, which is in fact the file called `MLE100b.Rsp` within the `Examples` subdirectory distributed with the program.

This program runs in a CMD window (also known as a DOS window), at least under Windows 2000 and XP. At a command-line prompt, you issue the command:

```
C> DistFit @MLE100b
```

DistFit reads the information in `MLE100b.Rsp`, and it produces two output files called `MLE100b.Txt` and `MLE100b.Tab`, which contain the results of the fitting process. **Notice the @ sign preceding the file name on the command line**—this is an essential detail, as is explained in section 4.

Table 1: An example of a DistFit input file, prepared by the user with any plain-text editor. This is the file `MLE100b.Rsp`.

```
* A sample file illustrating the format of DistFit input files.
* This file illustrates maximum-likelihood fitting with the data
* contained in a separate file (MLE100b.Dat).

* The comment following each parameter indicates what information
* it conveys to the program.

NDists 2              * Two distributions are to be fit to the data.
  Normal(0,1)         * The first distribution is the normal,
  Laplace(0,1)        * and the second distribution is the Laplace.

* In the preceding two lines, 0 and 1 are guesses for the parameters.
* For each distribution, the program adjusts these to the best-fitting
* values, but it is important to choose reasonable guesses.

TEXTOUTPUT   * This line instructs the program to write the *.txt output file.
TABOUTPUT    * This line instructs the program to write the *.tab output file.

MLEFIT MLE100b.Dat  * The fitting is to be done by MLE (Max likelihood).
                    * The data to be fit are contained in the file MLE100b.dat
                    * This is a plain ASCII file with one data value per line.
```

As an example of the output, Table 2 shows the output file `MLE100b.txt` obtained using the input file shown in Table 1. The format of the output depends somewhat on the estimation method; this is the output format for maximum likelihood estimation, as used in this example.

- The first line shows the best-fitting parameter values.
- The next line shows the obtained value of -2 Ln(Likelihood) corresponding to the best fit. By definition, the maximum likelihood value is the one that minimizes -2 Ln(Likelihood).
- The next set of lines, down to and including the ChiSquare line, report the results of a chi-square goodness of fit test comparing the observations against the fitted distribution. The column "Range" refers to ranges of observed values. The column "Observed" indicates how many observed values were

in each range, and the column “Predicted” indicates how many values are predicted to be in each range by the distribution with the best-fitting parameter values. The ChiSquare value at the bottom gets larger as the fit is poorer; if the *p* value at the end of the line is less than 0.05, then it should probably be concluded that the observed values do not come from this theoretical distribution.

- The last line simply records the program’s default setting to look for the best parameter estimates with a plain simplex search; this setting and other possible settings are explained in section 7.4.

The block of lines just described is simply repeated for each of the distributions to which the observed data were fit. If multiple data sets are fit, there is a block like this for each distribution for each data set.

Table 2: DistFit’s summary of the results of the fitting exercise shown in Table 1. This summary is written to the file `MLE100b.Txt`.

```
Best-fitting parameters for distribution 1: Normal(0.05253,0.8675)
-2 Ln(Likelihood):   127.6766
               Range    Observed      Predicted
 -4.0708 to  -1.0593:         16          9.997
 -1.0593 to  -0.6774:          4         10.008
 -0.6774 to  -0.4020:          7         10.009
 -0.4020 to  -0.1669:         13         10.002
 -0.1669 to   0.0525:          8          9.984
  0.0525 to   0.2719:         12          9.984
  0.2719 to   0.5071:         10         10.002
  0.5071 to   0.7825:         13         10.009
  0.7825 to   1.1644:          8         10.008
  1.1644 to   4.1758:          9          9.997
ChiSquare =    11.2119 with 7 degrees of freedom (p=0.130)
Search mode for distribution 1: Simplex

Best-fitting parameters for distribution 2: LaPlace(0.1239,0.6844)
-2 Ln(Likelihood):   131.3999
               Range    Observed      Predicted
-12.0096 to  -0.9777:         17         10.000
 -0.9777 to  -0.5033:         10         10.000
 -0.5033 to  -0.2258:         10         10.000
 -0.2258 to  -0.0288:          8         10.000
 -0.0288 to   0.1239:          5         10.000
  0.1239 to   0.2766:         10         10.000
  0.2766 to   0.4735:         10         10.000
  0.4735 to   0.7510:         12         10.000
  0.7510 to   1.2255:          9         10.000
  1.2255 to  12.2574:          9         10.000
ChiSquare =     8.4000 with 7 degrees of freedom (p=0.299)
Search mode for distribution 2: Simplex
```

The other output file type—the `*.tab` output file—is somewhat redundant, containing the same resulting parameter estimates and goodness-of-fit summaries. While the `*.txt` output file is fairly readable, though, the `*.tab` output file is in a columnar format that may be more convenient for further processing with other programs (e.g., Excel). This is useful if you want to summarize many separate sets of parameter estimates—e.g., across replications.

This documentation describes the various options available with DistFit, and it is intended to be used as a reference manual. It is also possible—and maybe easier—to get a lot of information by looking at the example files `*.Rsp` and `*.Dat` supplied with the program. These files illustrate many of the more common options, and many of the files are heavily commented.

3 Installation & Test

Now that you know what DistFit does, perhaps you would like to try it out.

1. Unzip the distribution file. It will be named something like DistFit.ZIP, except that the last few letters of DistFit will be changed to numbers to reflect the current version (e.g., DistFit11.ZIP is version 1.1). Note that subdirectories called `Examples\DistFit\-In` and `Examples\DistFit\-Out.OK` are created; these are for test purposes (see point after next).
2. Move the program file DistFit.EXE to a directory in your path. This program runs in a CMD window (also known as a DOS window), at least under Windows 2000 and XP.
3. If you like, you can run some quick tests of the program to make sure that everything is working properly on your machine and your version of the operating system. To do the quick tests, open a command window and change its current directory into the subdirectory `Examples\DistFit\-In`. Then run the batch file `Go.Bat`. The batch file should run a series of test runs, and it should produce output files as described later. When the batch file is done, check that every newly created file in the `Examples\DistFit\-In` subdirectory is identical to the file with the same name in the `Examples\DistFit\-Out.OK` subdirectory. If the files match, it is likely that everything is OK. If it fails, contact the author.

4 Starting the Program

DistFit is invoked from the command line of a CMD or DOS window. Parameters controlling its behavior can be specified on the command line or in `Rsp` files, as is illustrated in the file Go.Bat. Here are some examples:

```
C> DistFit @Exampl1
C> DistFit @Exampl1a @Exampl1b
C> DistFit @Exampl2 TabOutput
C> DistFit @Exampl1 MLEFit_MyDataFileName.Dat
C> DistFit @GotoExample1(My1stGotoLabel)
```

The ampersand (@) character at the beginning of a parameter says that this parameter is the name of a control file from which parameters should be read. Control files names must end with the extension `.Rsp`. In the first DistFit command, for example, the parameter “@Exampl1” tells the program to read input parameters from the file `Exampl1.Rsp`, starting from the first line in that file.

The first example is all you really need to know for now. The other examples provide additional flexibility (not additional capability). If you are in a hurry, you may want to skip the rest of this section for now, and come back to it when you find you want a little more flexibility in specifying the program control parameters on the command line.

Here is a description of several other, more advanced options for specifying program control parameters on the command line:

- Parameters may be specified on the command line itself, as in the third and fourth example DistFit commands shown above. In the third example command, for instance, DistFit reads parameters from `Exampl2.Rsp` and then sets one further parameter called `TabOutput` (described later).
- You can read program control parameters from two (or more) `Rsp` files, simply listing all of them on the command line. In the example on the second line, for instance, program control parameters are read from both `Exampl1a.Rsp` and `@Exampl1b.Rsp`.

- Sometimes control parameters in the `Rsp` file consist of two or more terms separated by white space, as you will see descriptions of the different control parameters. To specify such parameters on the command line, use an underscore instead of white space, as shown in the fourth command (`MLEFit_MyDataFileName.Dat`).
- You may want to use a single `Rsp` file to control several slightly different runs, with somewhat different parameter settings. When the `Rsp` file name is followed on the command line by an extra parameter in parentheses [e.g., `@GotoExample1(My1stGotoLabel)`], parameters are read from the indicated `Rsp` file starting at the point in the file indicated by the label in parentheses, rather than starting at the beginning of the `Rsp` file. The method of labelling a point in the `Rsp` file is explained further in section 5.5.

These options can be combined in all of the various ways that you would expect.

5 Structure of the Input File

DistFit reads all of the data and program control information from one or more plain-text input files that you create with an editor prior to starting DistFit (see Table 1 for an example). This section describes the different parameters and data format options. The sample files `*.Rsp` and `*.Dat` illustrate the use of many of these options.

Here are some general comments about the parameters:

- No parameters are case-sensitive.
- With some crucial exceptions, one parameter is specified on each line of the input file.
- The parameters can appear in (almost) any order.
- Most parameters are optional and have reasonable defaults.
- Parameters can be followed on the same line by comments. By default, the asterisk is used to start a comment, but this can be changed.
- Indenting (leading whitespace) is almost always optional (i.e., it is optional unless clearly noted otherwise). It is used in the examples to improve readability and indicate grouping.

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

Much of the terminology used in this section assumes that you have the basic concepts of random variables and their probability distributions. If you do not understand this terminology, then you may find it helpful to read the background provided in section 1, or consult a basic text on probability distributions.

5.1 Specifying the Distribution(s) to Fit

The “NDistsToFit” parameter tells DistFit how many distributions are to be fit for each data set. It may be abbreviated “NDists”. Critically, this parameter must be followed by a series of lines, with one line for each distribution to be fit. These lines specify the distributions to be fit and give the starting parameter values for each one. Example:

```
NDists  4           * This indicates that 4 distributions should be fit.
  Normal(0,1)       * Distribution 1 is normal, starting with mean=0 and sigma=1.
  Uniform(-10,10)   * Distribution 2 is uniform, starting from -10 to 10.
  t(15)             * Distribution 3 is a t distribution with 15 degrees of freedom.
  Normal(0,1)     FR * Distribution 4 is normal, starting with mean=0 and sigma=1.
```

Optionally, these lines may also specify codes fixing one or more parameters for each distribution, as shown in the normal distribution at the bottom of the list. The optional parameter code specification “FR” after this normal distribution indicates that the normal mean is fixed at zero (i.e., only the standard deviation will be allowed to vary). Section 7.3 explains the parameter codes in more detail.

5.2 Specifying the Estimation Method and the Data

You have to choose one of four options to indicate the method used to estimate the parameters, and this choice determines the format of the data that you must provide to DistFit. The following subsections describe the four possible estimation methods and the default data format for each one. Some additional options for input data formats are described in section 5.6.

5.2.1 Maximum Likelihood Estimation

This type of estimation is selected with the command `MLEFit`. This command is followed on the same line either with the name of a file containing the observations to be fit, like this:

```
MLEFit  MLE100b.dat   * Perform max likelihood fit to data in MLE100b.dat
```

Or else, the command is followed on the same line with the number of observations to be fit, like this:

```
MLEFit  4   * Perform max likelihood fit to following 4 data values
  0.123
  0.456
  0.777
  0.989
```

In the former case, the specified file should contain the observations for the data set, with one observation per line (see the file `MLE100b.Dat` for an example). In the latter case, the observations should follow immediately after the MLEFit line in the `Rsp` file as shown above (also see the file `MLE100.Rsp` for an example).

5.2.2 Minimum Chi-square Estimation

This type of estimation is selected with the command `ChiSqFit`. This command is followed on the same line either with the name of a file containing the histogram to be fit, or with the number of bins in the histogram to be fit. In the former case, the specified file should contain the bins for the histogram set, with one bin per line plus one extra line at the beginning (see the file `ChiSq1b.Dat` for an example). In the latter case, the bins of the histogram should follow immediately after the `ChiSqFit` line in the `Rsp` file, as shown below (see the file `ChiSq1.Rsp` for another example).

```
ChiSqFit  4   * Perform min chi-sq fit to data set with 4 bins as follows:
  0           * Bottom value of first bin
  10 4        * 10 is top value of 1st bin; the bin  0-10 has  4 data values.
  20 12       * 20 is top value of 2nd bin; the bin 10.0001-20 has 12 data values.
  30 12       * 30 is top value of 3rd bin; the bin 20.0001-30 has 12 data values.
  40 4        * 40 is top value of 4th bin; the bin 30.0001-40 has  4 data values.
```

In both cases, the first line is special, because it simply indicates the value of bottom of the smallest bin in the histogram. The remaining lines (one per bin) show the top value of each bin and the number of data values in that bin. For bins after the first, the bottom of the bin is the top of the preceding bin (with scores exactly at the bin edge counted in the lower bin).

Note: When tabulating data for a histogram, scores exactly on a bin boundary should be included in the lower bin.

5.2.3 Moment-based Estimation

This type of estimation is selected with the command `MomentFit`. This command is followed on the same line either with the name of a file containing the moments to be fit, or with the number of moments to be fit. In the former case, the specified file should contain the moments of the data set, with one moment per line (see the file `Mom1b.Dat` for an example). In the latter case, the observations should follow immediately after the MomentFit line in the `Rsp` file, like this (also see the file `Mom1.Rsp` for an example).

```
MomentFIT 2  * Number of moments in this data set
   0.500     * The desired mean
   1.500     * The desired variance
```

5.2.4 Percentile Matching Estimation

This type of estimation is selected with the command `PctileFit`. This command is followed on the same line either with the name of a file containing the percentiles to be fit, or with the number of percentiles to be fit. In the former case, the specified file should contain the percentiles of the data set, with one percentile point specified per line (see the file `Pct1b.Dat` for an example). In the latter case, the indicated number of percentile points should follow immediately after the `PctileFit` line in the `Rsp` file (see the file `Pct1.Rsp` for an example). As is illustrated in the following example, each line has two values: The first is the percentile (ranging from 0–1, not from 0–100), and the second is the observed score at that percentile.

```
PctileFit 5  * There are 5 percentile points to be fit.
 0.1 1       * The data value at the 10th percentile is 1
 0.3 2       * The data value at the 30th percentile is 2
 0.5 2.5     * The data value at the 50th percentile is 2.5
 0.7 3       * The data value at the 70th percentile is 3
 0.9 4       * The data value at the 90th percentile is 4
```

5.3 Controlling the Output

5.3.1 Choosing Output File(s)

By default, the program produces the `*.txt` output file but not the `*.tab` output file. You can override these defaults either or both of these output file types with the commands `TabOutput` and `NoTextOutput`. The analogous commands `NoTabOutput` and `TextOutput` are also defined, although rarely needed because these are the defaults.

5.3.2 Adjusting Number Widths and Numbers of Decimal Places

You can change the field width and number of decimal places used to write out numbers. For example, you could use the following two commands to say that you would like the estimated parameter values written as numbers that are 10 spaces wide (including decimal point) with 5 decimal places.

```
ParmWid  10
ParmDP    5
```

Analogously, you can use similar commands to control the format in which other types of numbers are written:

Commands:	Control writing of:
CountWid, CountDP	Observed and predicted bin counts.
ScoreWid, ScoreDP	Observed and predicted data values.
ParmWid, ParmDP	Estimated parameter values.
ErrorWid, ErrorDP	Measures of goodness-of-fit error.
ProbWid, ProbDP	Probability values.

5.3.3 Setting the Output File Name

DistFit tries to generate a unique and reasonable file name for its `*.tab` and/or `*.txt` output file(s). It does this by concatenating the names of any `Rsp` files that have been read and any goto labels that were referenced. For example, if you invoke the program with

```
DistFit @Test1(Goto1)
```

then by default the output files will be called `Test1Goto1.*`.

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

```
Outfile foo2
```

Now the output files will be called `foo2.txt` and `foo2.tab`.

You can build up the output file name in stages, too, with a command like this:

```
AppendOutfile MLEa
```

This appends MLEa to the current output file name. So, the sequence

```
Outfile foo2
AppendOutfile MLEa
```

produce output file names `foo2MLEa.*`. Of course this is a silly example—you might as well just say `Outfile foo2MLEa`—but the "AppendOutfile" command can actually be quite useful when using multiple `Rsp` files and/or labels within `Rsp` files.

5.3.4 Controlling Bins for the Chi-Square Goodness of Fit Test

In many cases a researcher wants to know whether a given assumed theoretical distribution provides an adequate model for a given set of data (e.g., "are my data normally distributed?"). There are many possible tests to decide this, including both some tests that can be used only with certain specific distributions and other tests that can be used with any distribution.

DistFit uses a general test—the chi-square goodness of fit test—to check the fit of the data to the assumed distribution in two cases: with maximum likelihood estimation, and with minimum chi-square estimation. Specifically, the fit of a given theoretical distribution to a given data set can be assessed by an observed chi-square value (χ_o^2), defined as

$$\chi_o^2 = \sum_{k=1}^{K} \frac{\left[f_o(k) - f_e(k)\right]^2}{f_e(k)}$$

where the given distribution is divided into K bins, and where $f_o(k)$ and $f_e(k)$ are the observed and expected frequencies in each bin. Larger values of χ_o^2 correspond to poorer fits to the given theoretical distribution. Moreover, with a large enough sample size (and under certain assumptions) the value of χ_o^2 can be checked to see whether the data deviate "significantly" from the assumed theoretical distribution. Specifically, the deviation is significant if the value of χ_o^2 is larger than the critical value from a chi-square table, with degrees of freedom equal to $K - P - 1$, where P is the number of distributional parameters estimated from the data (e.g., P would be 2 if the theoretical distribution were the normal and if the normal mean and standard deviation were estimated from the data).

By default, DistFit reports the value of χ_o^2 and its associated significance level for each maximum likelihood fit to a data set of 25 or more points, and for every minimum chi-square fit. By default, if there are N data points in the maximum likelihood fit, the number of bins K is set to the greatest integer less than $\sqrt{N}$. The bin boundaries are chosen so that each bin is expected to contain an equal number of observations (i.e., the bins are equal in probability mass). The user can override the default number of bins like this:

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

The number of bins cannot be adjusted with the minimum chi-square estimation method, because it is determined by the specified bins in the data set.

Also, the user can request that the bins be adjusted to be equal in numerical width (on the random variable)

```
ChisquareGOFBinsEqualInX   * Construct chi-square(s) bins equally wide in scores
```

This option is often very useful with discrete probability distributions, because DistFit can get confused about choosing equal-probability bins with those distributions (i.e., equal-probability bins may not exist).

As noted above, by default the bins are constructed to have equal probability mass; this default can also be requested with the command

```
ChisquareGOFBinsEqualInP   * Construct chi-square(s) bins to have equal probability
```

5.3.5 Adding a Title to the Output File

You can add a title to the top of the output file with the command

```
Title This is an example title for my output file.
```

Everything after the `Title` command is simply copied into the output file.

5.4 Comments in the Input File

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

```
Comment !          * Starting with the NEXT LINE, comments are set off by !
Outfile MyOutName  ! Set the output file name
```

5.5 Multiple Specifications in a Single `Rsp` File

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

Within an `Rsp` file, each of the separate sets of specifications starts with an arbitrary label and ends with the control specification `End`. In principle, the label can be any alphanumeric string you like, but in practice it is convenient to use a set of characters that can be used in the name for the output file (see section 5.3.3). For example, suppose this is the file `MultSpec.Rsp`:

```
TypeA            * Label for first set of specifications.
...              * Here is the set of specifications for one run.
End              * End of first set of specifications.

TypeB            * Label for second set of specifications.
...              * Here is the set of specifications for another run.
Goto TypesBandC  * Go to the indicated label to process more commands.
End              * End of second set of specifications.

TypeC            * Label for third set of specifications.
...              * Here is the set of specifications for another run.
Goto TypesBandC  * Go to the indicated label to process more commands.
End              * End of third set of specifications.

TypesBandC       * Label that is the target for the above GOTO commands.
...              * A set of commands to be used for both TypeB and TypeC.
End
```

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

To choose which set of specifications you want, invoke DistFit with a label in parentheses after the name of the `Rsp` file, like this:

```
DistFit MultSpec(TypeA)
DistFit MultSpec(TypeB)
```

Each run will then process the appropriate set of specifications. The output files will be named `MultSpecTypeA.*` and `MultSpecTypeB.*`.

As is also illustrated in the example, you may also use the `GOTO` command in the input file. When this command is encountered, the program reads ahead in the `Rsp` file until it finds the label that is the target of the GOTO command, and it continues processing at that point. Note that GOTO can only go *forward* in the `Rsp` file, though.

5.6 Multiple Data Sets in a Single Run

DistFit can also process data in a columnar format, as is described in this section. Using this format, DistFit can process more than one data set in a given run, fitting each distribution to each data set (see the files `Examples\DistFit\-In\ColEx*.*` for examples).

To indicate that you have columnar format data, include the command `ColumnFormat` in the `Rsp` file. *This command must appear before the command requesting a specific fit type (e.g., maximum likelihood), and it can only be used when the data appear in an external file–not with data inside the* `Rsp` *file.*

5.6.1 Columnar Data for Maximum Likelihood Estimation

For example, the `Rsp` file might contain these lines:

```
ColumnFormat            * Data is in a separate file in columnar format.
MLEFit ColExMLE.Col     * ColExMLE.Col has 1 line per dataset.
```

Then, the file `ColExMLE.Col` should contain *one data set on each line*, with the different data values in each set included in successive columns on that line. As is shown in the example below, the data values on a given line must be separated by white space, and there can be different numbers of data values in different data sets:

```
220 180 800 540 670 320 280 600 940 710
221 182 803 544 1010 833 675 326 287 608 949 710
```

5.6.2 Columnar Data for Minimum Chi-Square Estimation

For example, the `Rsp` file might contain these lines:

```
ColumnFormat                    * Data is in a separate file in columnar format.
BinEdges 200 250 300 350 400 450 500 550 600   * List of the high bin edge values
MLEFit ColExChiSq.Col           * ColExChiSq.Col has 1 line per dataset.
```

Note the new command `BinEdges`; this command specifies the values used to define the bins that were used in tabulating the data file. The first value (here, 200) is the *bottom* of the lowest bin, and the subsequent values are the *tops* of the successive bins. So, there are eight bins in this example, each 50 points wide, starting at the lowest value of 200. Again, the file `ColExChiSq.Col` should contain *one data set on each line*, with the counts for the bins included in successive columns on that line. Here is an example:

```
   1 10 13 16 22 13 11  9  5
   2 10 13 16 22 13 11  9  4
```

Note that in this format the bin specifications are not included with the data as they were in the format described in section 2.1.

5.6.3 Columnar Data for Moment-Based Estimation

For example, the `Rsp` file might contain these lines:

```
ColumnFormat            * Data is in a separate file in columnar format.
MomentFit ColExMom.Col  * ColExMom.Col has 1 line per dataset.
```

Then, the file `ColExMom.Col` should contain *one data set on each line*, with the different moments in each set included in successive columns on that line. As is shown in the example below, the moments on a given line must be separated by white space:

```
  240 2000 128000
  250 2100 134000
```

5.6.4 Columnar Data for Percentile-Based Estimation

For example, the `Rsp` file might contain these lines:

```
ColumnFormat                        * Data is in a separate file in columnar format.
Percentiles 0.1 0.3 0.5 0.7 0.9    * Data values are for these percentiles.
PercentileFit ColExPct.Col          * ColExPct.Col has 1 line per dataset.
```

Note that the `Percentiles` command is used to specify which the percentile values for the different data points to be fit. Then, the file `ColExPct.Col` should contain *one data set on each line*, with the different percentiles in each set included in successive columns on that line, like this:

```
  220  250  280  320 380
  220  250  285  330 410
```

For example, in both of these data sets the value at the 10th percentile was 220.

6 Program Output

DistFit writes a maximum of two output files, called `foo.txt` and `foo.tab`, where `foo` is determined as described in section 5.3.3. The `*.tab` file is a tab-delimited output file with the fitting results in a pure numerical format, as might be appropriate for analysis with another program. The `*.txt` file is an ASCII text file with the same information in a more human-readable format. The example files in `-out.ok\*.*` include plenty of examples, and you can easily determine the meanings of the columns in the `*.tab` files by comparing the numbers against the corresponding `*.txt` files.

I believe the information in the output files is basically self-explanatory (see also the explanation of Table 2 in section 2.2). Some explanation is needed, however, for the lines like this

```
Sum of squared errors:     0.0016
```

These lines appear in the output files for moment-based and percentile-based estimation, and they are simply the sum of the squared differences between predicted and observed values.

7 Miscellaneous Advanced Options

7.1 Interrupting DistFit

DistFit is sometimes pretty slow. You should be able to abort it with control-break if you need to do so. Otherwise, just close its window.

7.2 Running DistFit at Low Priority

If you want to do a DistFit run that will take a long time, you probably want to run it at low priority so that you can use the computer for other things at the same time (i.e., without noticing a big slowdown). To do that, use the Windows start command like this:

```
C> start /low DistFit @MyParms
```

This will open up a new window in which DistFit will run at low priority. I find this hardly slows down the program completion time at all, and yet it makes the computer much less sluggish when working in other applications at the same time.

7.3 ParmCodes

In some situations, it may be desirable to fix one or more parameters of one or more of the distributions being fit.

For example, you might know on a priori grounds that the mean of the underlying normal distribution was 0 and you might want to allow only the standard deviation to vary during maximum likelihood estimation. DistFit provides this capability through a mechanism called "ParmCodes", which allows you to specify whether each parameter of a distribution is Fixed, Real, or an Integer.

To constrain one or more parameters of a fitted distribution, you enter an optional extra string of letters (F, R, and I) after the name of the distribution (and separated from the name by at least one space). This extra string is called the "ParmCodes" string. Note that:

- The ParmCodes string should have exactly one character for each parameter in the distribution.
- Each character in the ParmCodes string should be either F, R, or I, with no other characters allowed.
- The order of the characters in the string should match the left-to-right order of the parameters in the distribution name.
- there is more complete information on ParmCodes setting in the documentation for CUPID.

For example,

```
Normal(0,1) FR
```

indicates that the parameter search routine should fit the data to a normal distribution, starting at the values of $\mu = 0$ and $\sigma = 1$, *and that the mean is fixed at 0.* You can specify a ParmCodes string after any or all of the distribution names.

7.4 Search Modes

DistFit obtains all parameter estimates with the simplex search algorithm (Rosenbrock, 1960). Like all numerical search algorithms, this algorithm may fail to find the best parameter estimates, because it may get trapped by a set of estimates that are the best within a certain region of the parameter space but are not the best overall (the so-called "local minimum" problem, referring to minimum error). In that case, the best parameters actually provide a somewhat better fit than the one given by the program.

This section describes some steps that you can take to diagnose and overcome the local minimum problem, thereby increasing your chances of getting the true best parameter estimates.

The best way to find out whether your parameter estimations suffer from the local minimum problem is to run DistFit several times and use widely different starting parameter values for the different runs. If the search always produces the same parameter estimates regardless of the starting guesses, then the values it finds probably really are the best estimates. If the search produces different parameter estimates when you use different starting guesses, however, that is a sure sign that there are local minimum problems.

When you find that your parameter estimation task suffers from the local minimum problem, there are a few things that you can do to increase your chances of getting DistFit to find the actual best estimates. One approach is to improve your starting guesses; as those get closer to the true values, the local minimum problem is less likely to arise. Unfortunately, this is not always a very helpful strategy in practice, because often you just don't have a good idea what the true parameter values are.

The other approach is to run the simplex search from many different starting guesses. The best result across all of the different searches is obviously more likely to be the overall best result than is the result of any individual search. DistFit provides "grid search" and "random search" modes to automate this process (i.e., carrying out many simplex searches from different starting values), as described in this section. Of course, grid searches and random searches slow down the program considerably, so you have to be prepared to wait hours or even days to get the best estimates if you want to go all out with these options.

DistFit has three search modes:

Simplex This search mode uses the simplex algorithm of Rosenbrock (1960).

Grid search This search mode tries all combinations of parameter values on a user-defined grid, possibly starting a new simplex search from each of the grid points.

Random search This search mode tries randomly generated combinations of parameter values within user-defined bounds, possibly starting a new simplex search from each randomly generated combinations.

The default search mode is simplex. To get one of the other two modes, you need to include specifications for it, as described in the next two subsections. Both the grid search and random search modes can optionally perform a simplex search starting at some or all of the parameter combinations considered by the search.

7.4.1 Specifying the Grid Search Mode

To use the grid search mode, you need to include a set of specifications like those shown in Table 3. Here is an explanation of the lines in that table:

Table 3: Example of specifications for grid search mode.

```
GridSearch 1   * Use a grid search for search type 1.
 3 0.5 1.1     * Special line: See notes below at @@
 100 500 10    * Parameter 1 should range from 100 to 500 in 10 steps.
  10 200 10    * Parameter 2 should range from 10 to 200 in 10 steps.
   1 400 10    * Parameter 3 should range from 1 to 400 in 10 steps.

* @@ Notes for the above special line:
* 3 indicates that there are three free parameters in this distribution.
* 0.5 indicates that a simplex search should sometimes be started from a grid point.
*     The alternatives are:
*          0.0  Never start simplex search from a grid point.
*          1.0  Always start simplex search from a grid point.
* 1.1 indicates that the simplex search should start from a grid point if the error
*     at that grid point is no more than CurrentBestError*1.1, where CurrentBestError
*     is the best error found so far.
```

The “GridSearch” word on the first line is simply the keyword to indicate that you want to use the grid search option. The integer “1” on that same line indicates that you want to specify this grid search for the first distribution being fit. Alternatively, you can use “2” here to specify grid (or random–see below) searches for the second distribution, or “3” to indicate that you want grid or random searches for third distribution, etc. If you want to specify all searches to be grid or random searches, you need to include one block of the sort shown in the table for each distribution.

The second line has three numbers with these meanings:

- The first number specifies the number of parameters to be varied in the search (more generally, the number of free parameters in the distribution being fitted to the data).
- The second number is used to specify whether the simplex search should be called at each of the grid points. There are three possible cases:
    1. If this value is less than or equal to 0, simplex is never called for any grid point.
    2. If this value is greater than or equal to 1, simplex is always called for every grid point.
    3. If this value is between 0 and 1, simplex is sometimes called, depending on the third number on this line.
- The third number, which will be called “ErrorFactor” is only used when the second number is between 0 and 1. In those cases, DistFit first evaluates the error associated with the current combination of parameters; call this error value “CurrentError”. It then computes the ratio of CurrentError divided by

ErrorFactor. If this ratio is less than the best error obtained previously, then DistFit starts a simplex search at the current point; otherwise, it skips the simplex for this point. I usually use ErrorFactor values of around 1.1–1.5, with higher values doing more simplex searches (thus, taking more time, but also giving a better chance of finding the best overall result). Intuitively, what is going on here? Well, CurrentError measures the error at the current parameter combination. If that is fairly small, then it seems like we might be close to the right parameter combination so it would be worthwhile to start a simplex search from this point to refine it further. How small is "fairly small"? That is defined by comparing the current error to the best error score so far.

The last three lines in Table 3 correspond to the three parameters varied in the search (one line per parameter). The first number on each line is the minimum value for the parameter, and the second number on each line is the maximum value for the parameter. The third number is the number of steps on the grid going from the minimum value to the maximum value. For example, for a parameter with a minimum of 100, a maximum of 500, and 10 steps, the search routine will try values of 100, 144.44, 188.88, 233.33, . . . 455.56, 500. The order of the parameter lines must correspond to the order of the parameters within the distribution being fit. For example, suppose you are specifying a grid search for a normal distribution. Then the first parameter line corresponds to the normal mean, and the second parameter corresponds to the normal standard deviation.

Notice that the total number of grid points considered is the product of the number of grid points on each parameter. So, for example, if you ask for 100 grid points on each of four parameters, DistFit will attempt as many as $100^4 = 100,000,000$ searches. That's going to take some time!

7.4.2 Specifying the Random Search Mode

As is shown in Table 4, the specifications for a random search are quite similar to those of a grid search, so I will just describe the differences.

Table 4: Example of specifications for random search mode.

```
RandomSearch 1
 3 0.5 1.1 1000    * See notes below at @@
 100 500           * Parameter 1 should range from 100 to 500.
  10 200           * Parameter 2 should range from 10 to 200.
   1 400           * Parameter 3 should range from 1 to 400.

* @@ Notes for the above line:
* 3 indicates that there are three free parameters in this distribution.
* 0.5 indicates that a simplex search should sometimes be started from a grid point.
*     The alternatives are:
*          0.0  Never start simplex search from a grid point.
*          1.0  Always start simplex search from a grid point.
* 1.1 indicates that the simplex search should start from a grid point if the error
*     at that grid point is no more than CurrentBestError*1.1, where CurrentBestError
*     is the best error found so far.
* 1000 indicates that 1000 starting points are to be randomly generated.
```

One difference is that there is a fourth number on the second line, here "1000". This is the number of random parameter combinations to generate and test.

The other difference is that you don't need to specify the number of steps between the minimum and maximum values for each parameter. Random parameter values are generated anywhere within the legal range, using a uniform distribution without reference to any steps.

Note that the control of simplex searching from parameter combinations generated randomly is just like the control from parameter combinations generated from a grid. Only the source of starting combination (grid versus random) is different.

7.4.3 Special Output with Grid and Random Searches

When you request that a search be carried out using a grid or random search, the program produces a little extra output information that may be informative, on a line that looks like this:

```
Results: 620 searches & 68 new bests found.
```

The number of searches (620, in this example) indicates the number of times that a simplex search was carried out. If you ask for a simplex search from *every* combination of parameter values, then this is simply the number of combinations generated (i.e., the number of grid combinations, or the number of random points generated). If you ask for a simplex search only *sometimes*, however, this number may be helpful to you in revealing the consequences of your particular value of ErrorFactor.

The “number of new bests found” (68, in this example) indicates the number of times that a simplex search resulted in a better parameter combination than any previously-tried parameter combination. This number can give you some indication of how serious the local minimum problem is, with lower numbers tending to less serious problems.

For example, suppose that your search problem is so well-behaved that the simplex search process will always find the best solution *regardless of the starting parameter values.* In that case, the very first simplex search will find the best solution, regardless of where you start it from. That search will automatically increment the “number of new bests found” to 1, because it is necessarily the best search so far. Critically, no subsequent search will do better, no matter how many subsequent searches are carried out, so the “number of new bests” will remain at 1. This suggests that there are no local minimum problems at all.

On the other hand, suppose that your search problem is so badly-behaved that the simplex search process settles on a different final solution from every different set of starting parameter values. (This situation corresponds to the worst case of lots of local minima.) In this case, the first simplex search process is very unlikely to give the best result, and later searches will tend to do better. You might wonder exactly how many times a new best is likely to be found (me too!), but for the present argument all that matters is that it is likely to increase with the number of local minima in the search space. So, larger numbers of new bests will tend to indicate more local minima, meaning that you need to be more cautious about concluding that you have found the overall best set of parameter values. And that means you want to put more computational effort into the problem, perhaps by increasing the number of grid points or the number of random searches across several runs until it appears that you have really found the best overall parameter estimates.

7.4.4 Simplex Search Parameters

Finally, some further options provide control over the simplex parameter search algorithm used to find the parameter estimates. These options apply in the same fashion whether the simplex procedure is used by the plain simplex search mode or as an augmentation of the grid search mode or random search mode.

MinStepSize This option is used to control the minimum step size used by the simplex parameter search algorithm. For example:

```
MinStepSize 0.00001
```

The default is 1.0e-8.

SimplexStartingStep This option is used to control the starting step size used by the simplex parameter search algorithm. For example:

```
SimplexStartingStep 0.01
```

The default is 0.2.

SimplexMaxIteration This option is used to restrict the maximum number of points examined by the simplex parameter search algorithm. For example:

```
SimplexMaxIteration 10000
```

would allow up to 10000 points in the parameter space to be examined in a given simplex search. The default is 5000.

7.5 Controlling the Accuracy of Numerical Integration and Inversion

One limitation of DistFit (as well as all of the other CUPID-related programs–see section 10) is that all distributions are represented numerically, with finite limits. DistFit's version of the standard normal distribution, for example, goes from about -5.6 to 5.6, not from $-\infty$ to ∞. Similarly, there are numerical bounds for all distributions. (You can find out what bounds DistFit is using by running CUPID and using the functions `minimum` and `maximum`). In addition, DistFit sometimes has to change bounds of naturally bounded distributions in order to avoid numerical errors. DistFit's Gamma distributions, for example, start at 0.00001 instead of 0.0, because the Gamma PDF cannot be evaluated at 0.0.

It is also important to realize that DistFit computes values by numerical integration and approximation in many cases where explicit formulas do not exist or have not been programmed in. You have some control over the accuracy of these numerical procedures via parameter settings described in this section. These settings are relevant when the affected numerical procedures are being used, and you have no real way to determine whether they are except by trial and error.

The speed and accuracy of the numerical integration procedure are controlled by a parameter called `IntegralPrecision`, which may be set with a command like

```
IntegralPrecision 0.00001
```

Larger values give faster convergence but less accuracy. The default is 1.0e-7.

Similarly, when DistFit must find the inverse of a CDF for a distribution with no explicit InverseCDF function built in, it uses a numerical search algorithm to find the desired X value corresponding to the specified P. You may control the accuracy required for the search to stop with commands like:

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

The IntegralPrecisionX parameter controls the required accuracy of computing inverse CDFs by searching: how close to the desired X value does the program have to get before it stops searching? There is an analogous parameter called InversePrecisionP, which controls how close the program has to get to the desired P value. The defaults are 1.0e-7 for InverseprecisionP and 1.0e-3 for InverseprecisionX; larger values will give faster runs but less accuracy.

Important note: If you want to alter IntegralPrecision or InversePrecision, it is best to do so *before* you define any probability distributions.

7.6 Checking DistFit

Although it is very general, DistFit is not always very accurate. Because many values are obtained through numerical integration, the results can be substantially off in some pathological cases, due to the vagaries of numerical approximations with finite-precision math. It is therefore important that you check the values that you care most about. One good check is to make very minor changes in data values and make sure that the program's results change only slightly. An even better check, if possible, is to carry out some runs of DistFit where you know exactly what the results should be. It is especially important to perform checks when using any new desired distribution for the first time, since different distributions have different susceptibilities to numerical problems, and I have by no means tested all of the distributions.

8 Available Probability Distributions

8.1 Continuous Distributions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ExWald(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.

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

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

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

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

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

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

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

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

Inverse Gaussian See the Wald3 distribution.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

where $K > 0$, $A > 0$, and $x \geq K$. This is a model for income, where K is some minimum income and $F(x)$ is the probability that a randomly selected income is less than or equal to x.

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

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

This distribution arises as a model of psychometric functions in visual psychophysics (e.g., Gescheider, 1997, p. 85). The threshold parameter, $T \geq 1$, represents an observer's fixed threshold for the number of quanta of light that must be detected before saying "Yes, I saw the stimulus." Quanta are assumed to be emitted from the stimulus according to a Poisson distribution with parameter x. Then, $F(x)$ is the psychometric function for the probability of saying "Yes" as a function of the mean number of quanta, x, emitted by the stimulus. Note that it makes no real sense to think of x as a random variable in this example, but the probability distribution provides a useful model anyway.

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

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

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

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

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

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

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

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

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

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

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

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.

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.

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

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

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

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

where all three parameters and x must be positive. Note: By default, the sigma parameter is fixed in all parameter searches.

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

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

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

The CDF of the Weibull is therefore

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

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

8.2 Discrete Distributions

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

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

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

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

rather than the equivalent

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

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

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

```
List(FileName)
```

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

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

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

The mean and variance both equal U.

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

8.3 Transformation Distributions

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

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

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

8.4 Derived Distributions

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

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

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

specifies the convolution of these normal and uniform distributions, and

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

specifies the convolution of the three indicated distributions.

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

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

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

DistFit 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, DistFit does not “realize” that some convolutions result in a new distribution about which it already knows (e.g., convolution of two normals is normal). Thus, computations involving these convolutions proceed via numerical integration even though direct computation would be possible.

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

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

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

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

is the same as

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

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

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

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

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

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

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

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

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

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

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

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

InfMix may be used recursively. For example,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

because

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

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

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

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

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

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

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

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

8.5 Approximation Distributions

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

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

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

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

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

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

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

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

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

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

Details of construction.

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

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

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

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

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

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

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

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

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

8.6 Distributions Arising in Connection with Signal Detection Theory

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

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

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

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

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

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

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

9 Release History

Version 1.1 was released widely in July 2005.
Version 1.0 was released on a limited basis in 1999.

10 Related Programs

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

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

RandGen Generates random values from a specified probability distribution.

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

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

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

11 Author Contact Address

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

I would also really like to receive feedback on who is using this software, and for what purposes. So, please e-mail me at miller@psy.otago.ac.nz if you found this software useful. If you do, I will add your name to my mailing list and let you know about any new versions, bugs, or new programs that might interest you. If you use this software for any published research, I would greatly appreciate it if you would acknowledge the software in your article (e.g., in a footnote) and email me a citation to the article or, better yet, send me a reprint.

Here is how to contact me:

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

12 Acknowledgements

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

13 References

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

14 Software License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

14.1 Preamble

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

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

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

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

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

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

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

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

14.2 Terms of License

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NO WARRANTY

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

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

END OF TERMS AND CONDITIONS