MixTest: A Program to Compute a Maximum Likelihood Test for a Mixture Effect

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

MixTest Version 1.02 — Cupid Version 2.13— Documentation Version January 17, 2006

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 **1**

2 Detailed Introduction **1**
2.1 Control condition . . . 2
2.2 Experimental condition, uniform model . . . 2
2.3 Experimental condition, mixture model . . . 2
2.4 The Likelihood Ratio Test . . . 2
2.5 Distributional Assumptions and Outliers . . . 3

3 Installation & Test **3**

4 Starting the Program **3**

5 Data Analysis Mode **4**
5.1 Input File Formats . . . 4
5.1.1 Data File . . . 4
5.1.2 Control File . . . 5
5.2 Output File Format . . . 6

5.2.1 Case-by-Case Output . . . 6
5.2.2 Results Summary . . . 6
5.2.3 Aggregated Results . . . 8
5.2.4 Fitting Conditions . . . 8
5.3 Data Analysis Options . . . 8
5.3.1 DataFile . . . 9
5.3.2 Setting the Output File Name . . . 9
5.3.3 DataSpacers . . . 9
5.3.4 ControlFit, ExptlFitUni, ExptlFitMix . . . 10
5.3.5 StartingP . . . 10
5.3.6 AddExptlFitUni . . . 10
5.3.7 Excluding Data Sets . . . 11
5.3.8 Title . . . 11
5.3.9 MissingValue . . . 11
5.4 Search Modes . . . 12
5.4.1 Specifying the Grid Search Mode . . . 12
5.4.2 Specifying the Random Search Mode . . . 14
5.4.3 Special Output with Grid and Random Searches . . . 14
5.4.4 Simplex Search Parameters . . . 15
5.5 Parameter Penalties . . . 16
5.5.1 Specifying Constraints on the Parameters of the Mixture Model . . . 16

6 Simulation Mode 17
6.1 Additional Control Options for Simulation . . . 17
6.1.1 ControlGen, ExptlGen,TrueP . . . 18
6.1.2 SampleSizes . . . 18
6.1.3 NIterations . . . 18
6.1.4 Controlling Simulation Output . . . 18
6.1.5 Excluding Iterations . . . 18
6.1.6 Aggregation Analyses . . . 19
6.1.7 Controlling the Seed of the Random Number Generator . . . 19
6.2 Simulation Output . . . 20
6.2.1 The Output DAT File . . . 20
6.2.2 The Output Summary TXT File . . . 20

7 Miscellaneous Advanced Options 21
7.1 AppendName . . . 21
7.2 ParmCodes . . . 21
7.3 FieldWidth and DecPlace . . . 21
7.4 Controlling the Accuracy of Numerical Integration and Inversion . . . 22
7.5 Checking MixTest . . . 22
7.6 Comment Character . . . 22
7.7 Flow of Control . . . 23
7.8 Start Simplex with Estimates from Moments . . . 23

8 Available Probability Distributions 24
8.1 Continuous Distributions . . . 24
8.2 Discrete Distributions . . . 28
8.3 Transformation Distributions . . . 29
8.4 Derived Distributions . . . 29
8.5 Approximation Distributions . . . 33
8.5.1 The automatic approximation . . . 33
8.5.2 Bin-based approximations . . . 33
8.6 Distributions Arising in Connection with Signal Detection Theory . . . 34

9 Release History **35**

10 Related Programs **35**

11 Author Contact Address **36**

12 Acknowledgements **36**

13 References **36**

14 Software License **37**
14.1 Preamble 37
14.2 Terms of License 38

1 Overview

This document describes MixTest, a program that can be used to compute a likelihood ratio test to see whether the difference between two conditions is a "uniform effect" or a "mixture effect", as defined further below. The data from the two conditions can be any form of numerical measurements. In addition to analyzing real data to test for uniform versus mixture effects, the program can also be used to generate simulated data to explore the statistical properties of the likelihood ratio test under conditions similar to those present in a particular experimental setting.

2 Detailed Introduction

What does this program do? This section describes the basic statistical scenario and defines some terminology.

Suppose a researcher collects numerical scores from two conditions and finds that the two conditions have significantly different means. To make the scenario a little more concrete, call the conditions "experimental" and "control", and suppose the mean is larger in the experimental condition.

Intuitively, the difference in means might reflect either a "uniform effect" or a "mixture effect", defined as follows: 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).

To take a traditional sort of example, suppose we want to evaluate the effectiveness of adding a certain chemical to a standard fertilizer. 100 corn plants are given the standard fertilizer as a control group, and 100 are given the standard fertilizer enhanced with the added chemical as an experimental group. Average productivity turns out to be higher for the experimental plants than the controls.

One interpretation of the higher scores for the experimental condition is that the added chemical facilitates growth of corn plants in general, so productivity tends to be larger for each plant in this group than it would have been without the chemical. This would be an example of a uniform effect.

Another possibility, however, is that the added chemical facilitates growth for only a proportion of the plants, with other plants being completely unaffected by it (presumably because of some genetic variation). This would be an example of a mixture effect.

Looking only at the means, there is no way to decide whether there was a uniform effect or a mixture effect. You might be able to see that there was a mixture effect by looking more closely at the observed distributions of scores, however. For example, you might find a bimodal frequency distribution in the experimental group; one mode might match the mode of the control group, and the other mode might be larger. In that case, you would have strong evidence of a mixture effect. But looking for a bimodal distribution isn't likely to be the most powerful approach, because (a) it requires a lot of data, and (b) it can only work for mixture effects that produce distinct modes.

This program tests between uniform and mixture effects with a likelihood ratio test taking into account the full distributions of scores in the two conditions. In brief, it works by fitting two different models to the

data—one corresponding to a uniform effect, and the other corresponding to a mixture effect. These two models will be called the "uniform model" and the "mixture model" throughout this documentation.

Each model is fit via maximum likelihood estimation of its parameters. Roughly speaking, the likelihood of a data set under a given model is the product of the probabilities (under that model) of all of the actual observations. So, maximum likelihood estimation adjusts the model parameters so that the model assigns the highest likelihood that it can to the actual observed data set.

Once each model has been fit by maximum likelihood estimation, a chi-square likelihood ratio test is used to decide whether the mixture model fits significantly better than the uniform model. In brief, this test just looks at whether the data are significantly more likely under the mixture model than under the uniform model. If so, the researcher can conclude that there is a mixture effect. If not, the null hypothesis of a uniform effect model cannot be rejected. The uniform model is simpler, so it can never fit better than the mixture model.

The program can be run in either of two modes, for data analysis or simulation. In the data analysis mode, it performs the likelihood ratio test on some data supplied by the user. In the simulation mode, the program iteratively generates random datasets (according to your specifications) and analyzes each one with the likelihood ratio test, tabulating the results across iterations. The simulation mode can be used to see how well the test works with a particular type of data that are of interest, as is illustrated in Miller (2006).

I will next get a bit more technical and introduce a little terminology which is used throughout the remainder of this documentation.

2.1 Control condition

Both models assume that there is some probability distribution for the observations in the control condition, call it f_c. For example, you might believe that the observations in the control condition have a normal distribution, so $f_c(x)$ is the function giving the normal curve over the region of x's (data values) of interest.

2.2 Experimental condition, uniform model

According to the uniform model, there is some other probability distribution for the observations in the experimental condition, call it f_{eu}. For example, if you thought the control distribution was normal, you would usually think that the observations in the experimental condition also had a normal distribution, although with a different mean and possibly a different standard deviation too. This other distribution is f_{eu}.

2.3 Experimental condition, mixture model

According to the mixture model, there are two possibilities for each observation in the experimental condition. Some experimental observations are unaffected by the manipulation, so they simply come from the control distribution, f_c. The rest of the experimental observations are affected by the manipulation, however, so they come from some other probability distribution, f_{em}, representing the distribution of *affected* scores. (I will sometimes refer to this as the "effect-present distribution".) For example, if you thought the control distribution was normal, you would usually think that the affected observations in the experimental condition also had a normal distribution, although with a different mean and possibly a different standard deviation. Finally, the mixture model has one extra parameter, which is the "effect probability", P. This is the probability that an observation in the experimental condition is actually affected by the experimental manipulation—i.e., the probability that it comes from the effect-present distribution rather than the control distribution. Thus, $1 - P$ is the probability that the observation is not affected, so it comes from the control distribution.

2.4 The Likelihood Ratio Test

In brief, MixTest computes a likelihood ratio test to compare the fits of two models to the data from the experimental and control conditions (see Miller, 2006 for more details). Both models have the distribution f_c in the control condition. The uniform model says that the data in the experimental condition come from

some other distribution f_{eu}, whereas the mixture model says that these data are a mixture of f_c amd f_{em}. MixTest tries various parameter values for these models and finds the ones that give the maximum likelihood for your data. Using those maximum-likelihood parameter estimates, it then computes a likelihood ratio test to evaluate whether the more flexible mixture model fits significantly better than the simpler uniform model.

2.5 Distributional Assumptions and Outliers

A limitation of MixTest is that the researcher has to specify the probability distribution family (e.g., normal, lognormal, gamma) of the observations. In some cases, the correct family may be known on a priori grounds or from extensive prior research in the area. If not, the researcher must examine the control condition data carefully to try to determine a reasonable distributional family for this condition before starting MixTest, and the program FitDist may be helpful in this regard (see section 10). Note that it is probably not a good idea to examine the experimental condition data in the same fashion, because you don't know when you start whether this condition should be fit by a single family or by a mixture.

Another important limitation of MixTest is that it is rather sensitive to the presence of outliers. It is thus extremely important to check the data carefully for outliers, and exclude any that are found. If there are some identifiable points that may or may not be outliers—you cannot tell for sure—then it would be prudent to run MixTest twice (i.e., once with the possible outliers and once without them) and to interpret the results only if the two runs lead to the same overall conclusion.

3 Installation & Test

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

1. Unzip the distribution file. It will be named something like MixTest.ZIP, except that the last few letters of MixTest will be changed to numbers to reflect the current version (e.g., MixTest11.ZIP is version 1.1). Note that subdirectories called `Examples\MixTest\In` and `Examples\MixTest\Out.OK` are created; these are for test purposes (see point after next).

2. Move the program file MixTest.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\MixTest\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\MixTest\In` subdirectory is identical to the file with the same name in the `Examples\MixTest\Out.OK` subdirectory. If the files match, it is likely that everything is OK. If it fails, contact the author.

4 Starting the Program

MixTest 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> MixTest @Exampl1
C> MixTest @Exampl1a @Exampl1b
C> MixTest @Exampl2 DataSpacers
C> MixTest @Exampl1 DataFile_MyDataFileName
C> MixTest @Example3(Simulation)
```

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 MixTest 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:

- 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`.
- Parameters may be specified on the command line itself, as in the third and fourth example MixTest commands shown above. In the third example command, for instance, MixTest reads parameters from `Exampl2.Rsp` and then sets one further parameter called `DataSpacers` (described later).
- 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 (`DataFile_MyDataFileName`).
- 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., `@Example3(Simulation)`], 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 7.7.

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

5 Data Analysis Mode

This section describes how to use MixTest in the data analysis mode. The simulation mode is described in section 6, but that section assumes you are already familiar with this one.

5.1 Input File Formats

MixTest reads the data and analysis options from plain-text (ASCII) files that you create in advance with an editor (although only one file is needed for simulation options, as is described in section 6). The next section describes the formats of both input files, and it lists and explains the various parameters. Examples of both types of files can be found in the `In` subdirectory, and it may be easier to learn about the parameters by looking at the examples in the file rather than reading this section of the documentation.

5.1.1 Data File

The data input file should be given a name with the extension ".DAT". As shown in `Exampl1.DAT`, the basic data format includes two lines per data set. Scores on the first line are the observations from one condition, arbitrarily referred to as the "control" condition, listed in any order and separated by spaces. Scores on the second line are the observations from the other condition, referred to as "experimental". Observations in both conditions can be any types of numerical values.

Tabs or spaces can be used to separate the numbers in the data file. The spacing is not significant to the program as long as different numbers are separated by at least one space or tab, but of course it is easier for a person to read if the numbers line up in columns.

As shown in `Exampl2.DAT`, the program can also process several data sets in a given run (these will be referred to as different "CASEs"). If multiple pairs of lines are included in the data file, MixTest will process each pair separately and then perform an aggregation across pairs, as is described in section 5.2.3. To input multiple cases, the same pair of lines simply repeats for each new case. By default, it is assumed that there are just two lines per case in the input file. In Exampl2.Dat, however, a blank line has been inserted between cases to improve readability. This departure from the default is specified with the "DataSpacers" command in the RSP file (see below).

5.1.2 Control File

The control file is used to specify various optional aspects of the data analysis (or simulation), and it should be given the extension ".RSP". RSP stands for "response", because these files convey the responses that you would give if you were controlling the program with a more traditional on-screen interface.

The control files (*.RSP) convey all of the desired option settings to MixTest. There are many options, and a complete list of them is given in section 5.3. In this section, I will just try to take you through a simple example, shown in Table 1, to get a basic idea of how to set up the program.

Table 1: Example option specification lines used in the file `Exampl1.RSP`.

```
Title Exampl1 Test Run
ControlFit Normal(0,1)
ExptlFitMix Normal(2,1)
StartingP 0.5
ExptlFitUni Normal(1,2)
AddExptlFitUni Normal(0,1)
AddExptlFitUni Normal(2,1)
```

Here is a detailed explanation of the lines in Table 1:

- The first line simply gives the program a title to be shown at the top of the output. This can be omitted.
- The second line ("ControlFit") indicates that the data from the control condition should be fit to a normal distribution, with 0 and 1 as the starting values for the mean and standard deviation of this distribution. This is the starting point for the f_c distribution, although the parameters will be adjusted to maximize the likelihood of the observations.
- The third line ("ExptlFitMix") indicates that, for the experimental condition, the effect-present observations should be fit to a normal distribution with 2 and 1 as the starting values for the mean and standard deviation. This is the starting point for the f_{em} distribution. Together, the second and third lines imply that the full set of the observations in the experimental condition should be fit to a mixture of two normals, with one normal starting at mean 0 and the other starting at mean 2.
- The fourth line indicates that the search should starting with a value of 0.5 for the probability that the effect is present (i.e., P).
- The final three lines determine the starting points for fitting the version of the model with a uniform effect. The "ExptlFitUni" line gives one of the search starting points. This line must be present and must appear before any "AddExptlFitUni" lines.
- The "AddExptlFitUni" lines are optional. These give additional starting points, so for this example a total of three starting points are specified. The program takes the best fit that it finds, but the option of multiple starting points is useful to try to combat the problem of stopping the search in a local minimum. Note that it is completely arbitrary in what order the different starting points are specified (here, the three different normals) although it is necessary for the "ExptlFitUni" line to appear first.

Some explanation of the concept of "starting values" may be useful. Starting values should be your ballpark guesses as to what the true values of the parameters might be. For example, if you are assuming normal distributions, then you should pick starting values for its μ and σ that would be realistic for the type of data you actually have. It doesn't matter much exactly which starting values you choose. The maximum-likelihood parameter search routine will always try to adjust the parameters from the starting values that you give it, in order to find parameters that give an even better fit to your data. So, for example, it probably won't matter whether you start with $\mu = 100$ and $\sigma = 20$ or $\mu = 95$ and $\sigma = 18$; these are similar enough that the parameter search routine fill find the same maximum-likelihood point from either one of them. But the parameter search routine can get stuck if your initial guesses are really far off the mark, so it is pretty important to have reasonably good starting values. For example, if you started with $\mu = 0$ and $\sigma = 1$ where something like the actual values were more like 100 and 20, then there is a good chance that the parameter search routine would not converge on the best values. If you are totally unsure of the starting values, you should try many different ones and see which ones give the best fits.

Quite a few different analysis options can be controlled from the RSP control file, and these are described in detail in section 5.3. First, however, I describe the program's output.

5.2 Output File Format

Tables 2 and 3 show an example output file with the analysis for an input data file with four cases. The case-by-case part of the output file is too wide to fit on a single page, so it is here broken up into two parts (left half in Table 2, right half in Table 3) for readability. See file `Exampl2.txt` to see both halves on the same lines.

5.2.1 Case-by-Case Output

For each case, the first output (column 2) is a classification of the effect as either "Uni" or "Mix" for that case. This is simply an easy-to-read summary of the analysis for that case. The next six columns provide descriptive information about the data in the control and experimental conditions, respectively: i.e., the sample sizes (N), observed mean (Mn), and observed standard deviation (SD).

"MixP" is the maximum likelihood estimate of P for each case.

"ChiSq" is the chi-square value of the likelihood ratio test.

Continuing on the right side of the page, "p" is the significance level of the observed ChiSq value, judged against a chi-square distribution with one degree of freedom. (The mixture probability is the one extra parameter in the mixture model, corresponding to the one df of the chi-square test.)

Most of the remaining columns on the right side of the page show the estimated values of the parameters for each condition (control, experimental) and each model (uniform, mixture). In this example the assumed distributions were normal, so Parm1 is the mean of the distribution and Parm2 is its standard deviation. In other examples, the distribution might have different parameters (e.g., rate of an exponential).

To elaborate a little bit on the meanings of these parameter estimates, those for the experimental condition correspond to the estimates for the distributions f_{eu} and f_{em} within the uniform and mixture models, respectively. The estimates for the control distribution are estimates for f_c, but note that these estimates differ across the two models. In the uniform model, the estimates of f_c are influenced only by the scores in the control condition. In the mixture model, the estimates of f_c are influenced by the scores in the experimental condition as well.

The right-most two columns in the table are the -Ln(Likelihood) values corresponding to the best-fitting pure and mixed models, respectively. Each of these is the negative of the natural log of the likelihood of the data, under the particular model. Note that smaller values indicate higher likelihood (i.e., better fit to the model), because of the minus sign. If you analyze the same data set in several different runs (e.g., with different starting parameter values), you should use as the "final" parameter estimates the run that gives the smallest -Ln(L) value across all of the different runs for that data set.

5.2.2 Results Summary

The results summary section summarizes the results across all cases. If you only analyze a single case, this section will not be produced.

Table 2: Example data analysis output shown in file `Exampl2.txt`. The case-by-case lines in the file are too long to fit on a single page, and these are the left halves of the lines. The right halves are in Table 3.

```
Title:
  Exampl2 Test Run



Case   Class     N_C     N_E    Mn_C    SD_C    Mn_E    SD_E    MixP   ChiSq
   1     Mix      20      20   0.100   1.369   2.404   4.690 0.15000  63.649
   2     Mix      20      20   0.120   1.374   2.424   4.673 0.15001  64.482
   3     Mix      20      20   0.085   1.376   2.389   4.697 0.15001  63.665
   4     Mix      20      20   0.100   1.369  17.404  41.219 0.14984 522.308

Results Summary:
  N good cases:      4
  N bad parm ests:   0
  N bad effects:     0
  Pr(ChiSqObs sig .10):    1.00000
  Pr(ChiSqObs sig .05):    1.00000
  Pr(ChiSqObs sig .01):    1.00000
  ChiSqObs      Mn, SD:  178.526 229.188
  Mixture P     Mn, SD:    0.150   0.000
  Control Distribution:    Mean      SD     Min     Max
    Uni Parm 1:           0.101   0.014   0.085   0.120
    Uni Parm 2:           1.338   0.004   1.335   1.342
    Mix Parm 1:           0.300   0.016   0.282   0.320
    Mix Parm 2:           1.253   0.008   1.241   1.259
  Exptl Distribution:
    Uni Parm 1:           1.938   0.936   0.534   2.424
    Uni Parm 2:           3.710   1.716   1.135   4.578
    Mix Parm 1:          38.010  50.000  13.010 113.010
    Mix Parm 2:           0.015   0.000   0.015   0.015

Aggregated Results:
  Total ChiSqObs(4) = 714.104  p = 0.0000

Fitting Conditions:
  Starting Control RV:    Normal(0,1)
  Starting Exptl Uni RV:  Normal(1,2)
                     and: Normal(0,1)
                     and: Normal(2,1)
  Starting Exptl Mix RV:  Normal(2,1)
  Starting P:             0.5000
```

Table 3: Example data analysis output shown in file `Exampl2.txt`. The case-by-case lines in the file are too long to fit on a single page, and these are the right halves of the lines. The left halves are in Table 2. Note that the -Ln(Likelihood) values "-Ln(L)" are only written if WriteMaxLikelihood is specified.

...	Cntrl	Cntrl	Cntrl	Cntrl	Exptl	Exptl	Exptl	Exptl		
...	Uni	Uni	Mix	Mix	Uni	Uni	Mix	Mix	Uni	Mix
Case... p	Parm1	Parm2	Parm1	Parm2	Parm1	Parm2	Parm1	Parm2	-Ln(L)	-Ln(L)
1...0.00000	0.100	1.335	0.298	1.255	2.404	4.571	13.010	0.015	92.928	61.103
2...0.00000	0.120	1.340	0.320	1.241	2.424	4.554	13.010	0.015	92.927	60.686
3...0.00000	0.085	1.342	0.282	1.259	2.389	4.578	13.010	0.015	93.058	61.226
4...0.00000	0.100	1.335	0.298	1.255	0.534	1.135	113.010	0.015	322.258	61.104

The first three lines of the summary are simply counts. The number of good cases is generally the number of cases analyzed, although it is possible to exclude cases based on wild parameter estimates or inappropriate effect sizes, as is described later. If any cases are excluded for these reasons, they are counted in the next two N lines.

The next three lines show the proportion of cases in which the observed chi-square value was significant at the indicate p-level, .1, .05, or .01. This provides one way of describing the overall results across a number of cases (e.g., "75% yielded significant evidence for a mixture model").

The "ChiSqObs" line summarizes the mean and standard deviation, across cases, of the ChiSq values.

The "Mixture P" line summarizes the mean and standard deviation, across cases, of the estimated P values.

The remaining lines summarize the parameter estimates that were shown on the right-hand side of the case-by-case analysis, showing the mean, standard deviation, minimum, and maximum (across cases) for each parameter.

5.2.3 Aggregated Results

The total chi-square value is simply the sum of the observed chi-square values across all cases. Assuming that the cases are independent, this sum should have a chi-square distribution with degrees of freedom equal to the number of cases. The associated p value is thus computed by looking up the computed total relative to that chi-square distribution. If this p is less than the chosen significance level (e.g., $p < .05$), then the data as a whole (i.e., aggregating across cases) provide evidence for a mixture effect.

5.2.4 Fitting Conditions

This section just reiterates the input parameters controlling the conditions of the analysis—i.e., the specifications used in fitting the uniform and mixture models.

5.3 Data Analysis Options

Numerous aspects of MixTest's behavior can be controlled by the user via a set of analysis options. This section lists analysis options that are needed for data analysis, and section 6.1 lists options that are needed mainly or exclusively for running simulations.[1] In addition, section 7 lists advanced options that may be used in either mode. Within each section, I have attempted to list the options approximately in order from most-frequently to least-frequently used.

All analysis options are specified by including one or more lines in the RSP file. An option is specified by typing its name as the first word on a line of the input RSP file (preceding blank space on the line is ignored). If the option requires further information, that information is typed as successive words on the same line (i.e., separated by one or more blank spaces) or, in a few cases to be described, on successive lines.

[1] The division into analysis versus simulation options is somewhat arbitrary, because some options apply in both cases. So, if you think an option listed in the analysis section should apply to simulation, or vice versa, you are probably right. Try it and see what happens.

Within an RSP file, the asterisk character (*) is used to indicate comment material: anything following an asterisk on the same line is ignored.

Here are some general comments about the parameters:

- Most parameters are optional and have reasonable defaults.
- The parameters can appear in (almost) any order.
- No parameters are case-sensitive.
- With some crucial exceptions, one parameter is specified on each line of the input file.
- 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.

5.3.1 DataFile

This option is used to specify the name of the input data file. It should be followed on the same line by a file name, like `DataFile MyInFile1`.

5.3.2 Setting the Output File Name

MixTest tries to generate a unique and reasonable file name for its output file(s). It does this by using the name of the first `Rsp` files that is read or the first goto label that is referenced. For example, if you invoke the program with

```
MixTest @Test1(Goto1)
```

then by default the output files will be called `Goto1.*`. Or, if you invoke it with

```
MixTest @Test1
```

then by default the output files will be called `Test1.*`.

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.

By default, MixTest will overwrite an existing output file of the same name when run in data analysis mode, but it will append to an existing output file of the same name when run in simulation mode. To override these defaults, you can specify the option `AppendFile` to make it append in data analysis mode or the option `Overwrite` to make it overwrite in simulation mode.

5.3.3 DataSpacers

This option is used to indicate that the input data file has a blank line at the end of each pair of data lines to improve readability. If you don't include blank lines in the data file, simply omit this option.

5.3.4 ControlFit, ExptlFitUni, ExptlFitMix

These three commands are used to specify the probability distributions to which the data are fit. As illustrated in the file `exampl1.RSP`, each of these options is followed by a distribution name to use in fitting the uniform and mixture models. Starting parameter values are also specified for each distribution. These are the starting points for the simplex routine that searches for the maximum-likelihood parameter values, and it is at least useful—and possibly essential—for these parameter values to be close to the true values.

The "ControlFit" distribution is the distribution to be used in fitting the data from the control condition. The same distribution is used for both the uniform model and the mixture model, although the parameter values are estimated separately for the two models.

The "ExptlFitUni" distribution is the distribution to be used in fitting the experimental condition within the uniform model.

The "ExptlFitMix" distribution is the distribution to be used in fitting the effect-present trials in the experimental condition within the mixture model. The unaffected trials within this model are of course fit to the ControlFit distribution.

The complete list of probability distributions that can be specified for fitting within MixTest, along with their parameters, can be found in section 8. The distribution options include not only dozens of standard distributions but also various transformations of these distributions (e.g., linear, log, power) and other distributions formed by taking convolutions, mixtures, order statistics, and so on, as derived from CUPID (Miller, 1998). In fact, MixTest is itself merely an extension of CUPID for this particular type of data analysis. The same distributions can also be used for generating data when the program is used for simulations.

In most applications that I can think of, I would expect the same distributional family (e.g., normal) to be used for all three types of fits (i.e., control, experimental within uniform model, effect-present experimental within mixture model). The program does not require this, however. It does require that the total number of parameters in the mixture model be one more than the number of parameters in the uniform model, however.

Section 7.2 describes further options allowing constrained fitting of distributions. As covered in that section, it is possible to have MixTest compute maximum-likelihood fits holding constant certain distributional parameters or restricting them to integer values.

5.3.5 StartingP

This option is used to specify the starting value (i.e., at the start of the maximum-likelihood search) for the mixture probability within the mixture model. For example, `StartingP 0.35` indicates that the search should start with a mixture model in which the effect is present 35% of the time. Of course the final value at the end of the search is whatever value maximizes likelihood. To prevent division by zero, you may not use a StartingP value of 0 or 1. The program checks for these cases and automatically adjusts them to starting values of nearly zero and nearly one, respectively.

5.3.6 AddExptlFitUni

A well-known problem with iterative parameter search routines, like the simplex, is that they can get stuck at local minima—parameter combinations that are best within a certain region of the parameter space but not best overall (see section 5.4 for further discussion of this problem). One way to combat that problem is to try starting the parameter search from different points in the parameter space, hoping that it will find the real overall minimum from at least one of the starting points.

Using the AddExptlFitUni option, MixTest lets you specify additional starting points for the search for optimal parameter values of the uniform model. If you do specify additional starting points, MixTest will carry out the search separately for each starting point, and it will of course use the best overall result (i.e., parameter values yielding maximum likelihood).

The AddExptlFitUni option must be specified *after* the ExptlFitUni option. For example, it would be correct to specify the sequence

```
ExptlFitUni Normal(0,1)
```

```
AddExptlFitUni Normal(0.2,1)
```

but it would not be correct to specify these in the reverse order.

5.3.7 Excluding Data Sets

When processing multiple sets of data (e.g., multiple experimental participants) in one run, it may be appropriate to exclude some sets of data from the analysis (and, especially, from the aggregation of results as described below). Three methods are providing for doing this.

One possibility is to exclude data sets in which the difference between the experimental and control means is too small to be of interest or too large to be realistic. The former could be done by specifying something like `MinEffect 10`, and the latter could be done with `MaxEffect 100`. These options just specify the minimum and maximum differences in the means (experimental minus control) needed for a data set to be included in the overall tabulation. For example, if there is a zero difference between the experimental and control averages within a given data set, it makes perfect sense to exclude this data set from the overall tabulation, because it cannot possibly provide any information about the nature of the difference between these two conditions.

Another possibility is to exclude data sets in which the estimated parameter values are out of reasonable bounds. It can happen with the simplex parameter search method (or any other search method) that occasionally an unusual dataset is obtained where the maximum likelihood parameter lie in some absurd region of the parameter space, and it may be reasonable to exclude such datasets from consideration. This can be done with a specification like

```
ParmBound 1 1 1 0 9999
```

"ParmBound" indicates that this specification defines an acceptable region for a parameter. Following that, the five numbers have the following meanings:

1. This number is either "1" to indicate a parameter estimate for the control condition or "2" to indicate a parameter estimate for the experimental condition.
2. This number is either "1" to indicate a parameter estimate for the uniform model or "2" to indicate a parameter estimate for the mixture model.
3. This number is 1, 2, 3, ... to indicate which parameter (i.e., 1st, 2nd, 3rd, ...) of the selected distribution (e.g., control, uniform) is being constrained.
4. This number is the smallest acceptable value for the parameter.
5. This number is the largest acceptable value for the parameter.

As an example, Table 4 shows a set of ParmBound specifications that might be used when the control and experimental distributions are normal, to include only iterations where the estimated means are between -100 and +100 and the estimated sigmas are between 0 and 500.

5.3.8 Title

This option is just used to generate a title that is included like a header line at the beginning of the summary output file.

5.3.9 MissingValue

If your input data includes missing scores, you can instruct MixTest to skip over those by using this option to set a numeric value indicating a missing score. By default, the value -999999.9999 is assumed to be missing. You can change it to any real number via a command like `MissingValue 999.99`.

Table 4: Example of specifications to restrict acceptable parameter values when each distribution is normal with mean, sigma.

```
ParmBound 1 1 1 -100 100  * Control, uniform, mean must be between -100 and +100
ParmBound 1 1 2 0 500     * Control, uniform, sigma must be between 0 and +500
ParmBound 1 2 1 -100 100  * Control, mixture, mean must be between -100 and +100
ParmBound 1 2 2 0 500     * Control, mixture, sigma must be between 0 and +500
ParmBound 2 1 1 -100 100  * Exptl, uniform, mean must be between -100 and +100
ParmBound 2 1 2 0 500     * Exptl, uniform, sigma must be between 0 and +500
ParmBound 2 2 1 -100 100  * Exptl, mixture, mean must be between -100 and +100
ParmBound 2 2 2 0 500     * Exptl, mixture, sigma must be between 0 and +500
```

5.4 Search Modes

MixTest 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 MixTest 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 MixTest 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. MixTest 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.

MixTest 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.

5.4.1 Specifying the Grid Search Mode

To use the grid search mode, you need to include a set of specifications like those shown in Table 5. Here is an explanation of the lines in that table:

Table 5: 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 control condition with the uniform model. Alternatively, you can use "2" here to specify grid (or random–see below) searches for the experimental condition with the uniform model, or "3" to indicate that you want grid or random searches for the mixture model. If you want to specify all three types of searches to be grid or random searches, you need to include three blocks of the sort shown in the table—one for each search type.

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, MixTest 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 MixTest 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 5 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 the control condition in the uniform model, and suppose the ControlFit distribution is Normal(0,1). Then the first parameter line corresponds to the normal mean, and the second parameter corresponds to the normal standard deviation. Similarly if you specify a search for the experimental condition in the uniform model. The trickiest case is the one where you specify a search for the mixture model. In that case, the first parameter is the mixture probability. This is then followed by all of the parameters for the control distribution, and all of the parameters for the effect-present experimental distribution come last.

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, MixTest will attempt as many as $100^4 = 100,000,000$ searches. That's going to take some time!

5.4.2 Specifying the Random Search Mode

As is shown in Table 6, the specifications for a random search are quite similar to those of a grid search, so I will just describe the differences.

Table 6: 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.

5.4.3 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 indicate 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.

5.4.4 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.

5.5 Parameter Penalties

In some situations, it may be desirable to restrict the range of one or more of the parameters being estimated by the search process. This can be done with the "ParmPenalty" command. The basic format of this command is:

```
ParmPenalty SearchType ParameterNumber Smaller/Larger Boundary PenaltyFactor
```

For example, an example command might look like this:

```
ParmPenalty 1 3 smaller 49.33 1000
```

Here is an explanation of these control parameters, in order:

SearchType This indicates which search the parameter penalty should be applied to. The integer "1" on that same line indicates that you want to specify this grid search for the control condition with the uniform model. Alternatively, you can use "2" here to specify grid (or random–see below) searches for the experimental condition with the uniform model, or "3" to indicate that you want grid or random searches for the mixture model. If you want to specify all three types of searches to be grid or random searches, you need to include three blocks of the sort shown in the table—one for each search type.

ParameterNumber This indicates which parameter, from first to last, is being constrained. In the example command, the third parameter is being constrained, as indicated by the "3".

Smaller/Larger This indicates whether the penalty should be applied when the estimate gets smaller or larger than a certain boundary value. In the example command, the penalty is applied whenever the third parameter falls below 49.33. Presumably, there is good reason to suppose that the parameter should be larger than that value.

Boundary This indicates the boundary point below which or above which the penalty should be applied. In this example, the boundary is 49.33.

PenaltyFactor This is a multiplier that determines the size of the penalty for parameter values beyond the boundary. The larger this multiplier, the more severe is the penalty for an out-of-bounds parameter value. With a large multiplier like the example value of 1000, the parameter will virtually never be estimated beyond the boundary; with a much smaller multiplier, like 0.01, however, the parameter might well be estimated beyond the boundary. So, you can incorporate weaker or stronger constraints by adjustin this multiplier value. Trial and error will be needed to find an appropriate multiplier, though.

5.5.1 Specifying Constraints on the Parameters of the Mixture Model

In some cases, it may also be desirable to further constrain the parameters of the mixture model in terms of overall means and variances. For example, with some datasets the search procedure may determine that the effect-present distribution component of the best-fitting mixture model has zero variance, and this may be an unacceptable model on theoretical grounds. To address such problems, MixTest provides some rudimentary facilities for constraining the parameters of the mixture model. You will only need to use these facilities if you are getting unreasonable estimated distributions.[2]

Specifically, you can use the `ForceEP` option to specify a constraint between the parameters of the control distribution and the effect-present distribution within the mixture model. An example is

[2]It is not really necessary to understand why the estimated effect-present distribution sometimes has zero variance, but I will attempt to explain it anyway: Within the mixture model, the likelihood of each data point in the experimental condition is the weighted sum of its likelihood in the control distribution and its likelihood in the effect-present distribution, weighted by the mixture probability. For any given experimental condition data point (say, x_q), then, the likelihood under the mixture model can be made arbitrarily large by picking an effect-present distribution whose mean is the same as that data point and whose variance is arbitrarily small. Given that the likelihood of the whole data set is just the product of the likelihoods of the individual data points, this means that it may also be possible to make the overall likelihood (i.e., of the whole data set) arbitrarily large by increasing the likelihood of x_q. Whether or not this is really possible depends on how fast the likelihoods of the other data points decrease as the likelihood of x_q increases, which depends on the full data set, but it is sometimes possible, so effect-present distributions are sometimes estimated to have zero variance.

```
ForceEP Mean Larger 0.9
```

which indicates that MixTest should only consider mixture distributions in which the mean of the effect-present distribution is larger than 0.9 times the mean of the control distribution. Another example is

```
ForceEP Variance Smaller 2.0
```

which indicates that MixTest should only consider mixture distributions in which the variance of the effect-present distribution is smaller than 2.0 times the variance of the control distribution.

In general, the `ForceEP` option takes three parameters: (a) Mean or Variance, depending on which parameter is to be constrained; (b) Larger or Smaller, depending on the desired value of the effect-present distribution (c) a multiplier for the value of the control distribution, relative to which the parameter of the effect-present distribution is judged. The selected parameter (i.e., mean or variance) of the effect-present distribution is then forced to satisfy the desired relation (larger or smaller) relative to the product of the multiplier and the control distribution's parameter.

You can specify up to a maximum of four `ForceEP` options to constrain both the mean and variance of the effect-present distribution to be both larger than a certain fraction of the control value and smaller than some other fraction of it, as in this example:

```
ForceEP Mean Smaller 2.0     * Effect present mean < 2.0*control mean
ForceEP Mean Larger 0.5      * Effect present mean > 0.5*control mean
ForceEP Variance Smaller 2.0 * Effect present variance < 2.0*control variance
ForceEP Variance Larger 0.1  * Effect present variance > 0.1*control variance
```

It is also worth noting that the mixture model may not necessarily be at least as good as the uniform model when constraints are used. Although the uniform model is mathematically a special case of the mixture model *in general*, it is not a special case when mixture parameters are constrained (because uniform model parameters are unconstrained). Thus, if you use constraints, you may find some cases where the mixture model does not fit as well as the uniform model.

6 Simulation Mode

MixTest can also be used to generate simulated data of the sort that it analyzes, and you might want to do this to test the statistical properties of the likelihood ratio test under conditions similar to those you are studying. In particular, researchers in the planning phase of an experiment may wish to run computer simulations of a proposed design to estimate its statistical power, biases, etc.

For each iteration in simulation mode, the program (a) generates a set of data (experimental and control conditions), and (b) performs the maximum likelihood test for that data set. Tabulating the results across many iterations, you can study the statistical properties of the test (e.g., its power) under known conditions of interest to you.

6.1 Additional Control Options for Simulation

The simulation parameters are set with a control *.RSP file with the same basic format as the RSP files used to control data analysis. This section describes options that are used mainly or exclusively for simulation. Note that:

- the data fitting options described previously are relevant for simulation too, because in simulation mode the program also has to fit the uniform and mixture models to the generated data.
- the "DataFile" option is not used, because the data are generated by the simulation process.
- examples are given in the files `SimExampl*`.

6.1.1 ControlGen, ExptlGen,TrueP

These three options allow you to specify the underlying distributions from which the data are randomly generated.

`ControlGen Normal(10,2)` specifies that the data for the control condition should be random samples from a normal distribution with a mean of 10 and a standard deviation of 2. You can specify here any distribution and set of parameters known to CUPID.

`ExptlGen Normal(12,3)` specifies that the data for the *effect-present* observations in the experimental condition should be generated from a normal distribution with $\mu = 12$ and $\sigma = 3$.

Finally, `TrueP 0.88` specifies that the effect should be present with a true probability of 0.88. That is, in the long run 88% of the observations in the experimental condition come from the effect-present distribution specified by `ExptlGen`, whereas the remaining 12% come from the control distribution specified by `ControlGen`.

Beware: These options are involved in one of the rare cases in which the order of specifying options is important. When `ControlGen, ExptlGen`, or `TrueP` is specified, MixTest sets the corresponding fitting parameters (i.e., `ControlFit, ExptlFit`, and `StartingP`) to matching values. Therefore, if you want mismatching values, you have to specify the fitting option *after* the generating option.

6.1.2 SampleSizes

This option is used to specify the number of observations to generate for each condition (i.e., experimental and control). For equal sample sizes (say 150), you can simply specify the number with `SampleSizes 150`. If you want different sample sizes in the two conditions, specify each one individually using `SampleSizeC 150` and `SampleSizeE 50`, where C and E denote the control and experimental conditions, respectively.

6.1.3 NIterations

This option is used to specify the number of iterations of the simulation, with a command such as `NIterations 3000`. Each iteration involves generating the control and experimental condition data sets, and then performing the likelihood ratio test on them.

6.1.4 Controlling Simulation Output

By default, in simulation mode MixTest writes two output files (see section 6.2 for details). One file contains the results of each iteration in a tab-delimited format, and the other file contains a plain-text summary of the results computed after all iterations are completed. There are some options to modify this behavior.

The option `NoTabulate` indicates that MixTest should not compute the plain-text summary file, presumably because you want to summarize the results on your own.

The option `TabulateOnly` tells MixTest not to compute any further simulations but only to tabulate the results in the existing tab-delimited file.

The option `SaveIterationResults` tells MixTest to write a summary of the results of each iteration to the tab-delimited output file (this is also the default), and the option `NoSaveIterationResults` tell it not to do so.

The option `WriteSimulatedObservations` tells MixTest to write the simulated data of each iteration (i.e., observations in the control and experimental conditions) to the tab-delimited output file. By default, these data are not written. This option has no effect if `NoSaveIterationResults` has been specified.

The option `WriteMaxLikelihood` tells MixTest to write the maximum likelihood values associated with the estimated parameters to the tab-delimited output file. By default, these values are not written.

6.1.5 Excluding Iterations

Sometimes it is desirable to exclude unrealistic iterations from the overall tabulation. The methods available for doing this in simulation mode involve MinEffect, MaxEffect, and ParmBound, exactly as in data analysis mode (see section 5.3.7).

6.1.6 Aggregation Analyses

In addition to investigating the behavior of the maximum likelihood test one dataset at a time, it may also be of interest to investigate the properties of the test when aggregated across a number of datasets. For example, a psychologist might plan to compare two conditions separately for each of 20 experimental subjects (with, for example, 100 data points per condition per subject), and wonder about the power of the planned mixture test aggregating across 20 subjects.

The option `NumNCasesToAggregate` is used to tell MixTest how many likelihood ratio tests are to be aggregated. It is a little tricky, because several different numbers can be examined in the same run. For example,

```
NumNSubsToAggregate 3
 10 20 40
```

tells MixTest to summarize the properties of the test aggregated over 10 individual tests, over 20 tests, and over 40 tests. The initial "3" on the same line as `NumNSubsToAggregate` indicates that three numbers will follow on the next line, and a separate aggregation analysis is computed for each of these numbers.

In the simulation, the aggregation analysis works by randomly selecting (without replacement) the appropriate number of individual simulated tests, and treating those tests as one set to be aggregated. The option `NAggregationIterations` controls how many such random sets are selected. This is really fast, so use lots (the default is 10,000).

6.1.7 Controlling the Seed of the Random Number Generator

By default, the program starts its random number generator with a randomly generated seed each time it is run. This section describes options provided for controlling the seed of the random number generator.

```
SEED SAVE START filename
```

saves the starting value random number seed into the specified file. This starting value can be retrieved for another program run so that the subsequent run can be carried out with the identical sequence of random numbers, if desired.

```
SEED SAVE END filename
```

also saves the random number seed into the specified file. With this option, however, saving is done at the end of the program. If the seed is then retrieved in another program run, the subsequent run will be carried out starting from the ending point of the previous sequence of random numbers, eliminating the possibility of overlap in the random sequences.

```
SEED START filename
```

causes the program to start by reading the seed from the indicated file (which should previously have been written with the SEED SAVE START or SEED SAVE END option already described).

Examples:

```
* With this command, successive runs will give identical random numbers:
* Seedfile should already have been written by using a SEED SAVE
*  command in a previous run of MixTest.
SEED START seedfile
* With these 2 commands, successive runs will ‘‘continue on’’ the random number
* generator, guaranteeing no repetition of the random number generator
* until it reaches the end of its period.
SEED START seedfile
SEED SAVE END seedfile
```

By default, the random number seed is also saved into the `*.dat` file for the first iteration of each simulation run. This default can be turned off with the option `NoSaveIterationSeeds`.

Optionally, the additional command

```
SaveEverySeed
```

causes the program to save out the seed at the beginning of each iteration of the simulation. This can be used to allow restarting of the program at a specific seed value in order to get right to a problematic sample—e.g., one that gives unsatisfactory parameter estimates.

6.2 Simulation Output

By default, simulations produce two output files. One, called `*.dat`, is a report of the results for each iteration of the simulation. The other, called `*.txt`, is a summary of the results. Options described later allow you to suppress production of either of these output files.

6.2.1 The Output DAT File

This file is a tab-delimited file with one line per iteration of the simulation. The values in the first seven columns of the file are as follows:

1. The mean of the observations in the control condition.
2. The standard deviation of the observations in the control condition.
3. The mean of the observations in the experimental condition.
4. The standard deviation of the observations in the experimental condition.
5. The estimated mixture probability.
6. The observed value of the chi-square likelihood ratio test statistic.
7. The significance level of the chi-square likelihood ratio test statistic.

The values in the next set of columns—usually all of the remaining columns—are parameter estimates. These are written out in the same order as their summaries produced in connection with data analysis (see Table 2). That is, all of the parameter estimates for the control condition come first, and all of the parameter estimates for the experimental condition come after. Within each of these two, the parameter estimates for the uniform model come first, and the parameter estimates for the mixture model come second. And for each distribution (e.g., control, uniform), the parameters are written in the order in which they appear in the distribution name (e.g., μ then σ for a normal distribution).

Finally, if the option `SaveIterationSeeds` has been specified (see below), an extra set of columns is written after the aforementioned columns for the first iteration of each run of the program. This set of lines is a representation of the starting point of the random number generator, which can be useful for debugging.

6.2.2 The Output Summary TXT File

Most of the simulation summary output file is analogous to the data analysis summary output file described in section 5.2.2, with cases of course corresponding to iterations of the simulation.

An additional component of the simulation summary output is a brief summary of the "Simulation Conditions", which is basically just a recap of the model used to generate the data.

If aggregations are requested, another additional component of the simulation summary output file looks like Table 7. The first column shows the number of simulated cases over which the results are aggregated, and the three probability columns to the right show the probability of a significant aggregated result, with significance at the .05, .01, or .10 level, as indicated.

Table 7: Aggregation output from a simulation summary file.

```
Aggregation Results:
NCasesToAggregate   Pr(Sig .05)   Pr(Sig .01)   Pr(Sig .10)
             10:       0.99280       0.98860       0.99390
             20:       0.99993       0.99983       0.99997
             40:       1.00000       1.00000       1.00000
```

7 Miscellaneous Advanced Options

This section describes some additional advanced options that may be included in *.RSP files, both for data analysis mode and for simulation mode.

7.1 AppendName

This option is used to append a certain string to the current name of the output file. For example, if you first specify the option `OutFile Version1`, and later specify the option `AppendName PartA`, the output files would be named "Version1PartA.*". You can use AppendName repeatedly to build up a file name from multiple components. This is useful mainly when you want to combine the same components in different ways within a set of simulation runs.

7.2 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. MixTest 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,

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

indicates that the parameter search routine should fit the control condition 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 the distribution name with any or all of ControlFit, ExptlFitUni, AddExptlFitUni, and ExptlFitMix.

7.3 FieldWidth and DecPlace

These two separate options are used to control the format in which numerical values are written out. For example:

```
FieldWidth 14 * Allow 14 spaces for each number.
```

```
Decplace 8 * Give 8 decimal places for each number.
```

The defaults are 7 and 3, respectively.

7.4 Controlling the Accuracy of Numerical Integration and Inversion

One limitation of MixTest (as well as all of the other CUPID-related programs–see section 10) is that all distributions are represented numerically, with finite limits. MixTest'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 MixTest is using by running CUPID and using the functions `minimum` and `maximum`). In addition, MixTest sometimes has to change bounds of naturally bounded distributions in order to avoid numerical errors. MixTest'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 MixTest 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 MixTest 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.

7.5 Checking MixTest

Although it is very general, MixTest 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 MixTest 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.

7.6 Comment Character

The comment character (asterisk, by default), can be changed to any ASCII character. For example:

```
COMMENT ! * Change takes place starting with next line.
```

would change the comment character to an exclamation point.

Table 8: `Exampl3.RSP` file illustrating the use of labels and `goto`.

```
Analysis
DataFile Exampl1
ControlFit Normal(0,1)
ExptlFitUni Normal(1,2)
ExptlFitMix Normal(2,1)
StartingP 0.5
Goto Common

Simulation
ControlGen Normal(0,1)
ExptlGen Normal(2,1)
TrueP 0.5
NIterations 50
Goto Common

* The following options are used for both analysis and simulation.
Common          * This is the label
FieldWidth 8
DecPlaces 5
```

7.7 Flow of Control

MixTest normally read its control parameters line by line through the RSP file. Two commands will alter this behavior, and these are useful in preparing a single control file to carry out several different analyses or simulations in different runs.

END If this command is encountered, processing of the input control file stops.

GOTO label If this command is encountered, processing of the input control file skips to a line on which the indicated label appears, and continues on from the next line following that.

As an example, one might prepare the following file `TwoDists.RSP` for use with both data analysis and simulation: generate simulated data from both a normal and a uniform distribution:

7.8 Start Simplex with Estimates from Moments

Although the simplex procedure ordinarily attempts only to find maximum-likelihood parameter estimates, as are needed for the likelihood ratio test, MixTest provides an option to start each parameter search by looking first for parameter values that provide moments matching those of the to-be-fit distribution—i.e., to start with estimation by the method of moments. For some distributions, moment-based estimation is much faster than likelihood-based estimation, and starting with parameter estimates yielding the right moments can sometimes quickly get the simplex search procedure into the vicinity of the best parameter estimates, thus speeding the overall search.

To instruct MixTest to start with moment-based estimation for each estimation process, use a command like this:

```
StartMomentEstimates 2
```

The number (2, here) indicates the number of moments for the estimation process to match. For example, 1 signals that the mean should be matched, 2 signals that the mean and variance should be matched, 3 signals that the mean, variance, and third central moment should be matched, etc.

As far as I know, only trial and error will tell whether moment-based estimation will be helpful in any particular situation.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The CDF of the Weibull is therefore

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

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

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.

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

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

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

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

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

rather than the equivalent

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

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

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

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

MixTest 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 MixTest, 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

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

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

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

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

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

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

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

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

.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

because

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

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

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

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

Because distributions are constructed recursively, it is legal within MixTest 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 MixTest, but let me know if you find such a case!

8.5 Approximation Distributions

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

8.5.1 The automatic approximation

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

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

is automatically expanded into

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

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

8.5.2 Bin-based approximations

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

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

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

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

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

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

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

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

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

Details of construction.

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

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

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

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

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

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

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

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

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

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 $p = \text{Adjust}/N$; if $p = 1$, set $p = 1 - \text{Adjust}/N$. "Adjust" is a parameter between 0 and 1, specified by the user, to indicate how the extreme data values should be treated.
2. Find Z such that $p = \Pr(z \leq Z)$, where z is a random variable having the standard normal distribution.

APrime(NSignalTrials,PrHit,NNoiseTrials,PrFalseAlarm) This is the distribution of the sample A' computed from an experiment with NSignalTrials signal trials each having the specified true probability

of a hit, and NNoiseTrials noise trials each having the specified true probability of a false alarm. Specifically, A' is the distribution-free estimate of the area under the ROC curve computed using Equations 2 and 9 of Aaronson and Watts (1987).

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

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

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

9 Release History

- Version 1.02, Jan 2006, included some new distributions.
- Version 1.01, Dec 2005, included the ParmPenalty feature.
- Version 1.0 was released in August 2005.
- Beta version 0.91 was released in March 2005 with added search modes.
- Beta version 0.9 was released in January 2005.

10 Related Programs

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

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

RandGen Generates random values from a specified probability distribution.

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

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

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

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, Angelo M. Mineo provided crucial information about the general error distribution, and Angelo Mazza provided information about the Neyman Type A distribution. Thanks also to Roman Krejci, from whom I got the pascal code for the Mersenne Twister. Others who have helped, directly or indirectly, include Timo Salmi, Duncan Murdoch, and Ellen Hertz. Special credit goes to Alann Lopes, who showed me how object-oriented programming could be useful in the first place.

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.

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

Miller, J. O. (2006). A likelihood ratio test for mixture effects. *Behavior Research Methods,* in 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

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