Behavior Research Methods
2009, 41 (4), 1095-1110
doi:10.3758/BRM.41.4.1095

Getting more from accuracy and response time data: Methods for fitting the linear ballistic accumulator

Chris Donkin, Lee Averell, Scott Brown, and Andrew Heathcote
University of Newcastle, Callaghan, New South Wales, Australia

Cognitive models of the decision process provide greater insight into response time and accuracy than do standard ANOVA techniques. However, such models can be mathematically and computationally difficult to apply. We provide instructions and computer code for three methods for estimating the parameters of the linear ballistic accumulator (LBA), a new and computationally tractable model of decisions between two or more choices. These methods—a Microsoft Excel worksheet, scripts for the statistical program R, and code for implementation of the LBA into the Bayesian sampling software WinBUGS—vary in their flexibility and user accessibility. We also provide scripts in R that produce a graphical summary of the data and model predictions. In a simulation study, we explored the effect of sample size on parameter recovery for each method. The materials discussed in this article may be downloaded as a supplement from http://brm.psychonomic-journals.org/content/supplemental.

Many tasks used in experimental psychology involve participants making relatively simple decisions, for which the experimenter measures the response times (RTs) and the accuracy of the responses. In many cases, the difficulty of the task is also manipulated within subjects. The resultant interaction among speed, accuracy, and difficulty is complicated and presents significant challenges for standard analysis techniques, even in the simplest case of two response alternatives. Results from an experiment conducted by Ratcliff and Rouder (1998) demonstrated the range of effects that can occur, even within data from a single participant. They also demonstrate the well-established trade-off between decision speed and accuracy, showing how participants can improve accuracy by increasing the time taken to make a decision. The complex interdependence of accuracy and RT draws into question the common practice of analyzing accuracy and RT separately—for example, by using separate ANOVAs.

For more than 30 years, mathematical psychologists have been developing cognitive models to account for the wide range of choice RT phenomena. These models use a small number of decision-process variables to account for both the accuracy of responses and the complete distribution of associated RTs. Many models exist (e.g., Brown & Heathcote, 2005, 2008; Busemeyer & Townsend, 1992; Ratcliff, 1978; Ratcliff & Rouder, 1998; Ratcliff & Tuerlinckx, 2002; Smith & Ratcliff, 2004; Van Zandt, Colonius, & Proctor, 2000; Vickers, 1970), all differing in their assumptions about the exact nature of the underlying processes. However, most share the same basic framework. They assume that, when making a decision, the participant repeatedly samples information from the environment and that this information is used as evidence for one of the potential responses. As soon as the evidence in favor of one potential response reaches a threshold, the decision process is terminated and that response is made. The time taken to make the response equals the time to accumulate the required amount of evidence, plus some time taken for nondecision processes, such as perception and the execution of a motor response. These cognitive models all provide estimates of three key parameters: the rate at which evidence for a particular response accumulates (*drift rate*), how much evidence is required before a response is made (*response threshold*), and the amount of time taken for nondecision aspects of the task (*nondecision time*). Estimations for these quantities take into account the interaction between speed and accuracy in the decision being made. This unified account can be much more informative about the decision process than independent analyses of accuracy and RT can.

The community using cognitive process models of choice RT has been growing steadily. The models have been used to describe the underlying neurology of simple decisions (e.g., Carpenter, 2004; Forstmann et al., 2008; Gold & Shadlen, 2001; Hanes & Carpenter, 1999; Mazurek, Roitman, Ditterich, & Shadlen, 2003; Ratcliff, Cherian, & Segraves, 2003; Reddi, 2001; Roitman & Shadlen, 2002; Schall, 2001; Smith & Ratcliff, 2004). They have also been used to gain insight into the cognitive processes that underlie a wide range of simple choice tasks, including elements of reading (Ratcliff, Gomez, & McKoon, 2004), recognition memory (Ratcliff, 1978), and visual discrimination (Ratcliff, 2002; Smith & Ratcliff, 2009), as well as more

C. Donkin, chris.donkin@newcastle.edu.au

complex decisions, such as purchasing a car (Busemeyer & Townsend, 1992). Ratcliff, Thapar, and McKoon (2001, 2003) used a decision model to identify which factors associated with aging were responsible for the observed slowing of older participants in simple discrimination tasks. In this application, independent analyses of accuracy and RT would not have identified these factors, because of a trade-off between speed and accuracy. Wagenmakers, van der Maas, and Grasman (2007) also suggested that variables (such as the rate of accumulation of information, or drift rate) estimated by these models be used to describe data in preference to accuracy and RT. Their approach is akin to considering intelligence in terms of an intelligence quotient rather than in terms of performance in individual tasks. Wagenmakers et al. (2007) gave a good explanation of how the analysis of data using decision models can, in a manner similar to that of psychometrics, reveal patterns in data that otherwise would have escaped the notice of the experimental psychologist.

Despite their ability to provide insight into the processes underlying decisions, the application of decision models has been limited mostly to those already within the field of cognitive modeling. This is because it has been notoriously difficult to apply the models to data, requiring complicated computer programming and mathematics to implement (Smith, 2000; Van Zandt, 2000). The models have grown more complex as the range of phenomena they can account for has grown (e.g., compare the original Ratcliff diffusion model [Ratcliff, 1978] to more recent versions in Ratcliff & Rouder, 1998, and Ratcliff & Tuerlinckx, 2002). Fortunately, there have also been attempts to reduce the complexity of choice RT models. Wagenmakers et al.'s (2007) EZ diffusion provides simple formulas for directly estimating the three key decision parameters on the basis of three easily estimated statistics. However, the model underlying the EZ diffusion approach is not comprehensive. For example, it fails to account for the latency of error responses. Such criticisms do not apply to the "complete" decision-making models, such as Ratcliff's. However, the price of this explanatory power is the great increase in mathematical and computational complexity.

Brown and Heathcote (2008) proposed the linear ballistic accumulator (LBA) model as the simplest complete model of choice RT. The LBA is simple, in the sense that Brown and Heathcote (2008) were able to derive analytically the model's probability density function (PDF), which makes efficient estimation tractable using a range of techniques. Despite its relative simplicity, the LBA can account for the same breadth of empirical two-choice RT phenomena as the Ratcliff diffusion model does. In contrast to the diffusion model, the LBA can be applied also to choices from among more than two alternatives. Even though the model is in its infancy, it has begun to be applied to experimental data sets (see, e.g., Donkin, Brown, & Heathcote, in press; Forstmann et al., 2008; Ho, Brown, & Serences, 2009). Forstmann et al. showed that an experimental manipulation of speed and accuracy emphasis produced changes in behavior and brain activity that agreed closely with appropriate parameters from the LBA.

In recent years, the options available for estimating parameters for the Ratcliff diffusion model have been increasing and have become more user-friendly. Vandekerckhove and Tuerlinckx (2007, 2008) developed the Diffusion Model Analysis Toolbox, a MATLAB program that uses methods developed by Ratcliff and Tuerlinckx (2002) to apply the Ratcliff diffusion model. Vandekerckhove, Tuerlinckx, and Lee (2009) have implemented the diffusion model into the sampling program for Bayesian inference, WinBUGS. Voss and Voss (2007, 2008) offered FastDM, stand-alone C code that also implements the Ratcliff diffusion model. Wagenmakers et al. (2007) offered three methods for obtaining EZ diffusion estimates: a spreadsheet in Excel, a Web applet, and some scripts in the statistical language R.

The present article is motivated by the observation that most previous software offerings for applying choice RT models to data have focused on Ratcliff's diffusion model. Here we provide a similar range of options for estimating the parameters of an alternative model, the LBA. We first review the details of the LBA and then describe estimation software developed for it in Microsoft Excel, R, and WinBUGS. We then describe additional software developed in R to produce a visual summary of data and model predictions. Rather than providing a comprehensive parameter estimation environment for a particular paradigm, our aim is to illustrate the three approaches in a way that allows users to flexibly extend the analysis to a range of paradigms.

Overview of the LBA Model

Consider a participant who has been presented with a string of letters and asked to decide whether the stimulus is a word or a nonword; Figure 1 shows how this decision is represented in the LBA. Each possible response ("word," "nonword") is assigned to an independent evidence accumulator. Evidence accumulation starts at a value randomly sampled (separately for each accumulator) from the interval [0, *A*], where *A* is the maximum value of the uniform start-point distribution at the beginning of each trial. The participant gathers information from the stimulus, which is then used to increment the evidence in either accumu-

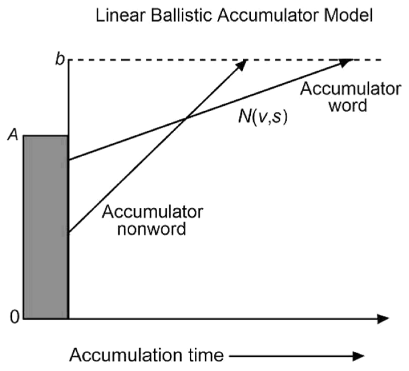


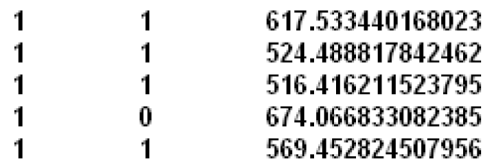
Figure 1. Graphical representation of a single decision made by the LBA model. Parameter values: value of uniform start-point distribution (*A*), upper response boundary (*b*), samples per condition (*N*), mean of between-trial variability in response time (*v*), and between-trial variability in drift rate (*s*).

lator. Brown and Heathcote (2008) made the simplifying assumption that evidence accumulation occurs linearly, at what is termed the *drift rate*. Drift rate is an indication of the quality of the stimulus: The larger the drift rate, the faster evidence accumulates. For example, because higher frequency natural-language words are easier to classify as words, a string of letters that forms a frequently used word, such as HOUSE, would likely have a higher drift rate than would the word SIEGE, since SIEGE is used less often. The drift rates for each accumulator vary from trial to trial according to a normal distribution with mean drift rates v_{W} (for words) and v_{NW} (for nonwords). For simplicity, we assume between-trial variability in drift rate, s, to be a common standard deviation for these distributions. As soon as evidence in one accumulator reaches a threshold, b, the response associated with that accumulator is made. Changing the threshold parameter changes the amount of evidence required to make a response. For example, a lower b produces less cautious responses, and an increased b produces more cautious responses. The relative values of b for different accumulators can model response bias: an a priori preference for one response over another. *Response latency* is given as the time taken for the first accumulator to reach threshold plus the time taken for nondecision aspects of the task, such as the motor response and stimulus encoding. Because nondecision time is assumed to have negligible variability, it is estimated by a single parameter, T_{er}.

One way of estimating LBA parameters from data involves the search for a set of parameters (e.g., b, A, v_{W}, v_{NW}, s, and T_{er}) that produce predictions for accuracy and RT that closely resemble the data. The resemblance between data and model is quantified by an objective function. A variety of objective functions are commonly used with RT data, including maximum-likelihood estimation (Ratcliff & Tuerlinckx, 2002), chi-squared estimation (Ratcliff & Smith, 2004), and quantile maximum products estimation (QMPE; Heathcote & Brown, 2004; Heathcote, Brown, & Mewhort, 2002). The search for a set of parameters that optimize the objective function begins with the choice of parameters at some initial values, called a *start point*. This is followed by a computer-driven search that changes parameters until a set is identified that provides a better value for the objective function than other nearby sets do. Bayesian estimation takes an alternative approach that provides a distribution of estimated parameter sets rather than a single set. Variability in the distribution quantifies uncertainty about estimation, and a measure of the distribution's central tendency, such as the mean, provides a point estimate. Our aim here is not to detail or compare these methods. Instead, we take advantage of the ready availability of efficient and general-purpose search algorithms (Solver in Excel and optim in R) and Markov chain Monte Carlo (MCMC) methods for generating Bayesian estimates (WinBUGS) to provide accessible options for estimating LBA parameters, given a set of data.

Methods for Estimating LBA Parameters From Data

We use a single set of simulated accuracy and RT data to illustrate the methods of applying the LBA model to a two-choice task. The design from which the simulated data are derived is typical of designs to which the LBA has been applied: three conditions that vary in decision difficulty (easy, medium, and hard). This set can be thought of as data from a single participant in an experiment with three within-subjects conditions. The first few lines of the simulated data are shown in Figure 2, with each line representing one choice trial. The first column codes the easy, medium, and difficult decision conditions, labeled 1–3, respectively. The second column codes the accuracy of the response made, using 0 for incorrect and 1 for correct. The third column contains the response latency of the decision (in milliseconds). We provide an R script that simulates data in the same format as our example data set (makedata.r). For this example, we sampled data from an LBA model with the following parameters: sec = 0.25, $A = 300$, $T_{\mathrm{er}} = 200$, $b = 400$, $v_{\mathrm{E}} = .9$, $v_{\mathrm{M}} = .75$, and $v_{\mathrm{H}} = .6$.



```
1        1        617.533440168023
1        1        524.488817842462
1        1        516.416211523795
1        0        674.066833082385
1        1        569.452824507956
```

Figure 2. The first five lines of data from our simulated data set. The first column shows the experimental condition (1–3), the second shows the accuracy (0 = *incorrect*; 1 = *correct*), and the third shows response time (RT, in milliseconds)

The v_{E}, v_{M}, and v_{H} parameters refer to the drift rates for correct responses. We use the traditional parameterization, which fixes drift rates for the error responses to be equal to 1 minus the drift rate for correct responses (although, see Donkin, Brown, & Heathcote, in press, for a different approach). Hence, the drift rates for incorrect responses in our example data set were .1, .25, and .4 for easy, medium, and hard conditions, respectively. To keep the example simple, we assumed that drift rate for correct (and error) responses was the same, regardless of which stimulus was presented on that particular trial. This embodies an additional assumption that drift rates are the same for both stimuli corresponding to each response (i.e., words and nonwords). In more general cases, there could be four drift rates: a correct and error drift rate for each of the two responses (e.g., *old* and *new* responses in a recognition memory task; Ratcliff, 1978). We also assume that only drift rate changes between the three difficulty conditions, with all other parameters constant. This assumption is standard in paradigms in which the stimulus factors that are used to manipulate difficulty are mixed randomly within blocks of trials. We address the more complicated cases below.

Example 1: Using Microsoft Excel

We used the file lba.xls to fit the LBA using Microsoft Excel. The Excel LBA worksheet records the data in Sheet 2 and uses the parameter values in Sheet 1 to calculate the likelihood of the data, given the present set of parameter estimates. This likelihood value is our objective function. Excel's built-in Solver function is used to find parameters, which maximizes this likelihood value.

The quality of the fit to data is shown in the histograms presented in Sheet 1.

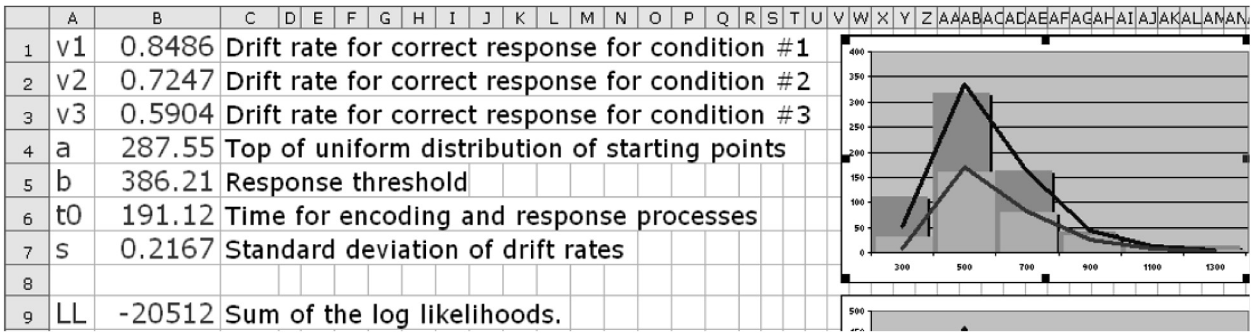
The likelihood can be thought of as a measure of the quality of the fit of the model to the data, with larger values indicating better fit. The parameters that provide the best fit to the data are those that maximize the likelihood, or, equivalently, the log likelihood. Rather than raw likelihood, we use log likelihood as the objective function, because taking logarithms avoids numerical problems associated with multiplying together many very small numbers.

In order to analyze our simulated data, we pasted the three columns of data directly from exampledata.txt into row 2, columns A–C, (hereafter referred to as Cells A2–C2) and onward in Sheet 2. Initial parameter guesses were entered into Cells B1–B7 of Sheet 1. The natural logarithm of the likelihood of the present set of parameters given the data is shown in Cell B9 of Sheet 1. The Solver function, which can be found in the Tools drop-down menu, is then applied.[1] Solver can then be used to maximize the log likelihood by going to Tools > Solver. A new application box appears, in which the user simply clicks the Solve option. Users can freely experiment with the numerous options in the Solver function. However, no such changes are required to fit the LBA to our example data. Although we do not discuss these options in detail here, we note that, by default, the Subject to the Constraints section is set up appropriately, so that Condition 1 is the easiest condition (and therefore should have the highest drift rate), that Condition 2 is the next hardest condition, and so on. A number of other sensible constraints can also be imposed, such as requiring $T_{er} > 0$ and $b > A$.

The plots in Sheet 1, an example of which is shown in Figure 3, summarize the quality of fit. To create these plots, the user must first place the RTs from their data into Column A of Sheet 3. This can be done by copying and pasting the contents of Column C of Sheet 2. There are three plots shown in Sheet 1, one for each condition. The plots show the correct and error RT histograms for the data and the LBA grouped into bins ranging from 300 to 1,500 msec, in 200-msec increments. The three histograms in the Excel sheet show data and predictions from the easy, medium, and hard conditions from top to bottom, respectively. The data are shown by the bars of the histogram: The actual spreadsheet uses color figures, in which red bars represent error responses and blue bars show correct responses. The predictions made by the LBA are shown by solid lines; again, in the actual spreadsheet, colors are used to identify correct and error responses. The predictions shown in the plots are based on parameter values given in column B of Sheet 1. Changing the parameter values by hand causes direct changes to the solid lines in the histograms. The LBA provides a good fit to the data whenever the solid lines match closely the bars of the histograms that underlie them, indicating that the RT distributions predicted by the LBA closely resemble those of the data. As soon as a good fit to the data is achieved, the user can record the parameter values reported in cells B1–B7 of Sheet 1. We do not suggest that these histograms are of publication quality; however, they do provide the user with a method for quickly checking the quality of the fit. We discuss how to use the parameter estimates to create further graphical summaries in the Using R to Create a Graphical Summary Given Parameter Values section.

Sometimes, rather than finding the optimal solution (i.e., the *global maximum*), Excel's Solver function becomes stuck in a *local maximum*, in which case, the estimated LBA parameters do not provide an accurate account of the data. This occurs when the solver finds a set of parameters—say, Solution A—that are better than other nearby sets of parameters but are still worse than the (quite different) parameters that provide the best fit to the data. Although this problem can be difficult to avoid in general, there are some measures that help address it. One method is to start the search again from a different starting point. Solution A is likely to be a global maximum if the Solver function repeatedly finds Solution A from a sufficiently wide range of starting points.

The choice of initial estimates for any method of fitting a model to data can be very important: Automatic optimization routines (like Solver) can fail badly if the initial estimates are poor. Choosing good initial estimates is something of an art that often requires experience and experimentation. This process is made relatively easy, thanks to the interactive nature of the Excel spreadsheet we have provided. The effects of changing parameters can be instantly observed by looking at the plots in Sheet 1. The initial estimates need not produce a set of solid lines that closely resemble the data; all that is necessary is that



	A	B	C
1	v1	0.8486	Drift rate for correct response for condition #1
2	v2	0.7247	Drift rate for correct response for condition #2
3	v3	0.5904	Drift rate for correct response for condition #3
4	a	287.55	Top of uniform distribution of starting points
5	b	386.21	Response threshold
6	t0	191.12	Time for encoding and response processes
7	s	0.2167	Standard deviation of drift rates
8			
9	LL	-20512	Sum of the log likelihoods.

Figure 3. Screen shot of the Excel LBA worksheet. The plot contains the data from one condition as a histogram, with bars showing correct and error responses. Solid lines show predictions of the LBA for correct and error responses, respectively. Predictions are based on parameter values given in row B.

Solver be given parameters that produce a solid line that has approximately the shape and location of the observed RT distributions. As a rough guide, because they approximate average parameter values from fits to a range of paradigms, the parameters used to generate our simulated data provide a good starting point for a parameter search for new data sets.

Given the wide range of possible data sets, it is impossible to create an Excel workbook that works "out of the box" for every case. For example, simply changing the number of observations in each condition requires the user to change certain aspects of our worksheets. However, given the flexibility and intuitive nature of data manipulation in Excel, the changes required to adapt the LBA workbook to new data sets should usually be relatively simple. For example, if the number of observations in each condition changes, the user has to update the entry in C12 of Sheet 1 and make sure columns G–O are the same length as the data that have been entered.

Example 2: Using R

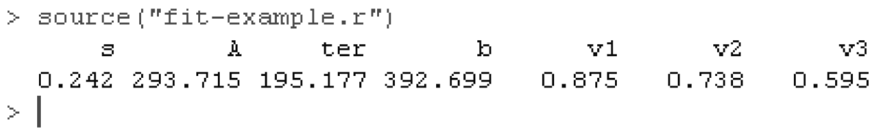
In order to estimate LBA parameters from our example data using R, the user begins by extracting the files in Rfit.zip into a directory. The R language software is free and available for Windows, Mac OS X, and Linux; it can be downloaded from the R homepage (www.r-project.org). After the executable install file is run, on-screen instructions describe the installation process. Extracting the contents of the Rfit .zip file into a folder (e.g., Desktop\Rfit) provides four files: lba-math.r, pq-lba.r, fit-example.r, and exampledata .txt. The .r files all contain R scripts that are used to estimate parameters for the LBA model from the example data contained in the .txt file. To begin the estimation process, the user should open the R software and change the working directory to the folder where the .zip file was extracted (i.e., Desktop\Rfit)—for example, by using the File > Change dir option. Typing `source("fit-example.r")` fits the LBA to the data found in the exampledata.txt file. The R prompt disappears while the software searches for a set of parameters that best fit the data. After the search is finished, the estimated parameters are printed on screen.

Using the *source* function in R is equivalent to entering each line contained in the fit-example.r script directly into the R window. Any text editor can be used to read fit-example.r file. The first command sources the pq-lba.r script, which, in turn, sources the lba-math.r script. These two scripts set up the functions that encode the basic mathematics of the LBA model. Some variables are then defined: `qps` gives the set of quantiles of the RT distribution to be used by the QMPE method of fitting the LBA, and `trim` gives the minimum and maximum values used to censor the RT data. We set `trim` to remove RTs that fall outside of the range between 180 msec and 10 sec. The R *read.table* function reads the contents of exampledata .txt file, maintaining the column structure and giving the variables `difficulty`, `correct`, and `rt`. In order to change which data file is read, the user can change the name of the first argument of the *read.table* call. However, it is necessary either to ensure that the data follow the same structure as that of the exampledata.txt file or to change other aspects of the R script appropriately.

The next section of script transforms the imported data into the format required for the QMPE fitting technique. Because QMPE is based on quantiles (i.e., order statistics, such as the median), it provides estimates more robust than does maximum-likelihood estimation in small samples (Brown & Heathcote, 2003; Heathcote et al., 2002). In the example, the estimates are based on five quantiles (.1, .3, .5, .7, and .9) for both correct and error RTs in each difficulty condition, storing them in the array `q`. The number of observations in each quantile bin is also calculated and stored in `pb`. Response accuracy and sample sizes for correct and error responses in each difficulty condition are also calculated and are stored in `p` and `n`, respectively. Finally, these variables are bundled together in a list and are stored in the variable `data`.

After the data are formatted correctly, the parameter search is called by the *fitter* function, which requires two arguments: The first is the `dat` argument, which, in our example, is the `data` list. The `maxit` argument specifies the maximum number of iterations used in searching for best-fitting parameters. Like the Excel workbook described earlier, R finds best-fitting parameters by, at each step, systematically changing parameters and keeping changes that provide a better value for the objective function. The `maxit` argument specifies how many steps are taken to find a set of parameters that best fits the data. The parameters that arise out of the *fitter* function are placed into the `pars` variable. The last few lines of the script transform the parameter values returned by the *fitter* function to those that are familiar to the reader from our explanation of the LBA. Figure 4 shows the R output after fitting the LBA to the example data set.

Unlike the Excel worksheet, the majority of the code that does the fitting in R is hidden in the pq-lba.r and math-lba.r scripts. The parameter values used to initialize the search for best-fitting parameters are produced automatically as part of the *fitter* function defined in the pq-lba.r script. These heuristics are clearly labeled in the pq-lba.r file. These estimates work in many situations but, in a few cases, are inadequate for the fitting algorithm to find good parameter estimates. Such cases reinforce the need to use the graphical summary methods to check the quality of a fit. These methods are described in the Using



```
> source("fit-example.r")
      s       A     ter       b      v1      v2      v3
  0.242 293.715 195.177 392.699   0.875   0.738   0.595
> |
```

Figure 4. Screen shot of the use of R to fit the LBA to our example data set.

R to Create a Graphical Summary Given Parameter Values section. In order to use these scripts with other data sets in which only changes in drift rate are extended across conditions, only the fit-example.r file must be edited: The `ndrifts` parameter must be set to the number of conditions in the data.

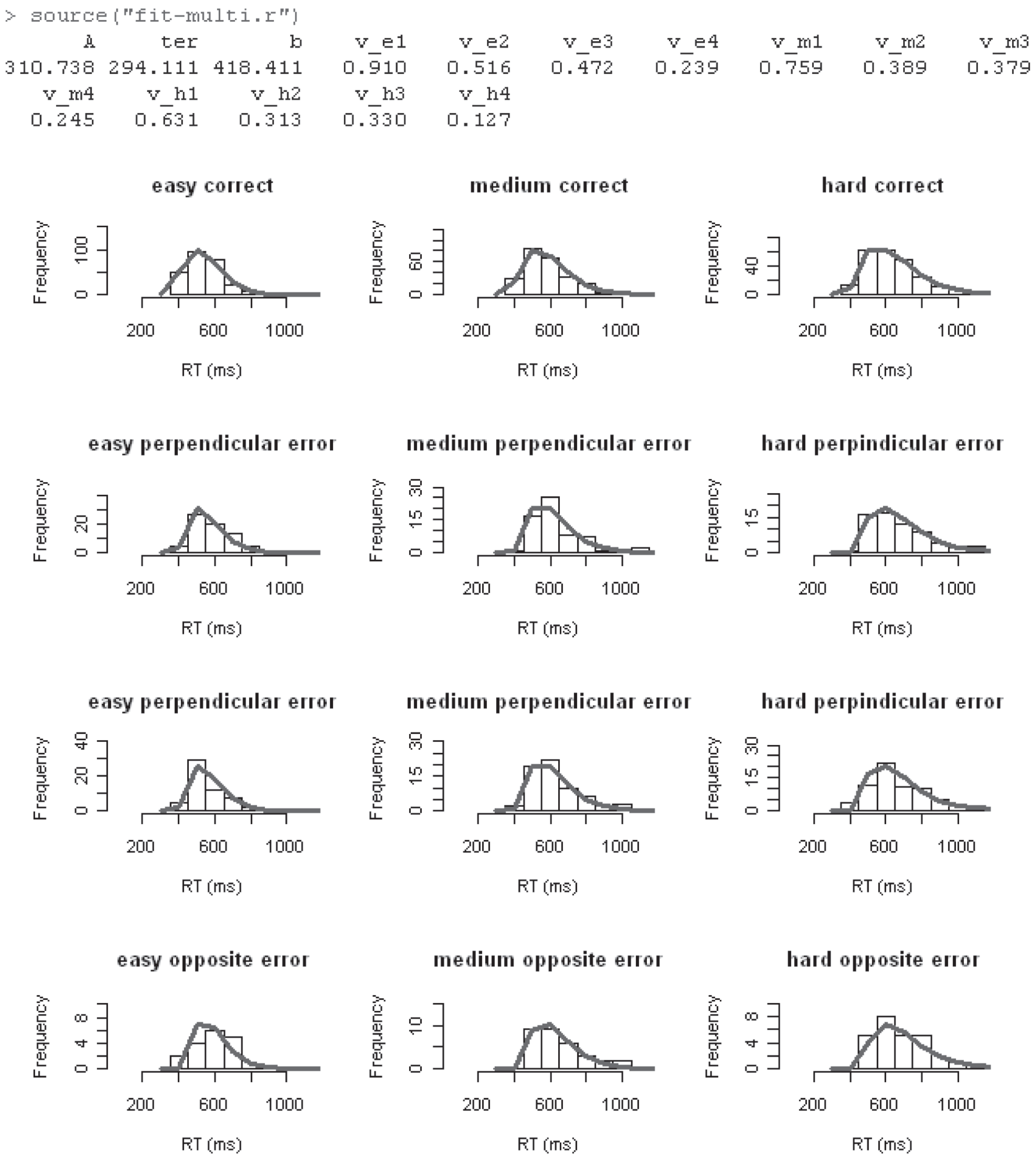
If other parameters are allowed to vary across conditions, more substantive changes are required. For example, Donkin, Heathcote, Brown, and Andrews (in press) propose that, in a lexical decision task, both drift rate and nondecision time vary with word frequency. Say we have three frequency conditions and want to estimate three values of drift rate, v, and three values of nondecision time, T_{er}. In such a situation, the *fitter* function in pq-lba.r can be updated so that starting points are generated for v_1, v_2, v_3, T_{er1}, T_{er2}, and T_{er3}. The *obj* function should then be updated to take into account these changes. Specifically, the `par` vector passed to the *obj* function is two elements longer—it used to contain s, A, T_{er}, b, v_1, v_2, and v_3, and now has s, A, T_{er1}, T_{er2}, T_{er3}, b, v_1, v_2, and v_3. The *getpreds* function expects to receive from *obj*, for each parameter of the LBA, a vector of length equal to the number of conditions (*nc*; 3, in this case). This means that, where previously we would have replicated T_{er} *nc* times (the line: `Ter = rep(par[3],nc)`), we now use three free parameters (`Ter = par[3:5]`), in the same way that we previously used three drift rate estimates (previously, `v = par[5:7]`; now, `v = par[7:9]`).

Analyzing data with more than one factor requires further changes to pq-lba.r. For example, we may have a *difficulty manipulation,* which varies across trials, and a *speed–accuracy emphasis manipulation*, which varies across blocks of trials. In this case, it is customary to fit an LBA, where v varies across difficulty conditions and where b and A vary across emphasis conditions. We begin in the same way by first generating start points for each of the parameters to be estimated within the *fitter* function. However, in the *obj* function, rather than producing a vector of length *nc* for each parameter, we must now produce a matrix with *nc* rows and two columns, one for speed emphasis parameters and one for accuracy emphasis parameters. This also means that, where the *getpreds* function had taken one element of the parameter vector (by using a loop over 1–*nc*), it now must take one element of the matrix of parameter values (using two loops, one over 1–*nc* and another over 1–2). Obviously, as the design of the data and the LBA model to be fit becomes more complex, so too does the R code needed. For users with limited programming skills, we provide code for WinBUGS that can be more easily adapted to more complicated models.

Multiple-choice data. One advantage of the LBA is its ability to model multiple-choice data (see Brown & Heathcote, 2008, for a demonstration). To illustrate, we provide code that can be used to simulate a set of data from an LBA with four accumulators, corresponding to a choice between four response alternatives. To recover the parameters, the code then fits the four-accumulator LBA model to the simulated data. The data are simulated to mimic an experiment in which a participant is presented with one of four random-dot kinematograms (RDKs), a set of pixels, of which a small proportion move coherently in one direction and that must be identified by the participant, while the others are moving randomly. The difficulty of the task is also manipulated to be easy, medium, or hard. Using this paradigm, Ho et al. (2009) fit the LBA to an experiment. The code necessary for simulating and fitting the multiple-choice data is contained in Rmultifit.zip. After the files are extracted, data can be simulated and fit using `source("fit-multi.r")`. The parameters are estimated by maximum-likelihood estimation, printed on screen, and histograms containing data and model predictions are produced. Here, we used maximum-likelihood estimation to illustrate how the R code described in the last section can be adapted for a different objective function.

The fit-LBA.r script is self-contained, in that there is no need to source the lba-math.r or pq-lba.r files, making the code required for maximum-likelihood estimation relatively simple compared with that required for QMPE. The data are simulated, and starting values are generated for the parameters we want to estimate using heuristics similar to those used in our other R code. We then define our objective function, *obj*. Since we are using maximum-likelihood estimation, our *obj* calculates the likelihood of each RT value given a set of parameters `pars`. Finally, we include code for producing histograms of observed and predicted RT distributions for each response in each difficulty condition.

For both simulating and fitting the data, we assumed that all parameters were fixed across stimuli, suggesting that participants show no bias for one particular direction of pixel flow. The simulated data used a large drift rate, corresponding to the correct response; the size of this drift rate varied for easy, medium, and difficult conditions. We reasoned that incorrect responses were more likely in the two directions perpendicular to the correct response and less likely in the direction opposite the correct response. To instantiate this, we used only two free parameters for the three error response alternatives: one value to set the fraction of the correct-response drift rate assigned to the perpendicular responses (we used .5) and one value to set the fraction assigned to the opposite response (.25 in our simulated data). We simulated the data, therefore, using 5 drift-rate parameters: 3 for the correct responses in each condition, 1 indicating the proportion of the correct drift rate for perpendicular incorrect responses, and 1 indicating the proportion of the correct drift rate required for opposite-direction incorrect responses. To fully demonstrate the method for estimating parameters from a multiple accumulator LBA model, when fitting the data, we made no assumptions about relationships across drift rates. We estimated 12 drift rate parameters: 1 for each of the four responses in the three difficulty conditions. Figure 5 shows parameter values returned by our maximum-likelihood *fitter* function. The parameters reported are close to the parameters used to simulate the data ($A = 300$, $b = 400$, $T_{er} = 300$, $v_e = .9$, $v_m = .75$, $v_h = .6$, $p_{perp} = .5$, $p_{opp} = .25$). The histograms in Figure 5 also demonstrate that the predictions from the LBA match closely the observed data. The biggest misfit is to RT distributions for opposite error responses. These responses are the most



```
> source("fit-multi.r")
      A     ter       b    v_e1    v_e2    v_e3    v_e4    v_m1    v_m2    v_m3
310.738 294.111 418.411   0.910   0.516   0.472   0.239   0.759   0.389   0.379
   v_m4    v_h1    v_h2    v_h3    v_h4
  0.245   0.631   0.313   0.330   0.127
```

Figure 5. Screen shot of the R code and resultant output used to fit data simulated from a four-accumulator LBA model. Data are represented by the bars of the histogram.

incorrect and are made least often. Therefore, RT distributions for these responses are made up of relatively few observations; hence, estimation of parameters for these responses is more erroneous.

Example 3: Using WinBUGS

Bayesian analysis in psychological research is rapidly gaining popularity for a range of reasons, such as providing estimates that perform well in predicting new data sets and account for model flexibility (Wagenmakers, Lee, Lodewyckx, & Iverson, 2008; see Raftery, 1995, and Wasserman, 2000, for general introductions). Bayesian analysis starts by assuming a *prior distribution* (i.e., distribution before new data are taken into account) of parameter estimates. It then combines the prior with the observed data to produce a *posterior distribution* of parameter estimates (i.e., estimates updated by the new data). The process of Bayesian estimation has been made relatively easy by the availability of flexible programs, such as WinBUGS (Lunn, Thomas, Best, & Spiegelhalter, 2000), which use general-purpose MCMC methods to obtain samples from the posterior distribution (see Calin & Chib, 1995). We demonstrate how WinBUGS can be used to fit the LBA to data, including instructions for compiling and running WinBUGS, as well as reviewing and saving the results.

WinBUGS makes MCMC methods available to researchers with relatively little programming and mathematical knowledge through a graphical user interface. The Appendix to this article provides instructions for installing WinBUGS and the WinBUGS Development Interface (WBDev) and BlackBox Component Builder.[2] The latter two programs are used to give WinBUGS access to the LBA PDF. The BugsLBA.zip file contains a compound document (lba.odc) that defines the LBA PDF. As described in the Appendix, a one-time installation procedure is required to enable WinBUGS to sample from the LBA posterior. After this procedure is complete, WinBUGS should always be launched from the BlackBox installation folder.

The BugsLBA.zip file also contains fitlbaexample.odc, which, in separate sections, defines the WinBUGS model and data specific to the present example. The model section specifies uniform prior distributions (`dunif`) for each LBA parameter (A, b, v, s, and T_{er}). The parameters of the uniform priors were chosen to be relatively uninformative. That is, the range of the uniform priors is chosen to span a broad range of plausible parameter values. When given a reasonable amount of data over a sufficiently broad range, the prior is not overly influential on the posterior estimates. For the A parameter, for example, the uniform prior distribution ranges from .1 to 1.0.[3]

Specification of overly broad priors can cause WinBUGS to fail, so some experimentation can be required to obtain a computationally stable, but sufficiently uninformative, prior. Relatively uninformative priors produce WinBUGS estimates that do not differ greatly from the estimates obtained from the two methods presented previously. A section containing initializing values (called *inits*) for the MCMC sampling can also be added to fitlbaexample.odc, but this is necessary only when the inits automatically generated by WinBUGS fail. Such failures are most common when priors are broad. Specifying appropriate inits can help to protect against failures of WinBUGS when broad priors are used.

As with the previous methods, we estimate three drift rates (v_1, v_2, v_3). In WinBUGS, this is done by letting v be a vector containing three priors, one for each of the three conditions. In our example code, all of the v priors are identical and relatively uninformative; however, this need not be the case: Different priors for the drift rate for each condition could be imposed if desired. The final line of the model section connects the RT data, defined as the variable t in the next (data) section, with the previously defined parameters (and their priors) via the PDF for the LBA.

The next section of fitlbaexample.odc contains a modified specification of the data contained in exampledata .txt. In order to allow WinBUGS to handle bivariate data (RT and accuracy), we follow the common practice (Vandekerckhove et al., 2009; Voss, Rothermund, & Voss, 2004; Voss & Voss, 2007, 2008): Let RT be the observed response latency for a particular response, and let t be the data given to WinBUGS. If the response is correct, then code $t = RT$; otherwise, code $t = -RT$. This enables both accuracy and RT information to be specified in a single variable. The data section also defines other variables used in the model section. For example, the number of data points, N, is defined as 3,000. The condition for each response is defined by the entries in the `cond` variable, a value of 1 for the first 1,000 RTs (i.e., Condition 1), 2 for the next 1,000 RTs (i.e., Condition 2), and so on.

The following steps can be used to compile the model and obtain posterior samples:

1. Open exampledata.odc from within WinBUGS (recall that this must be run from the BlackBox directory). After the file is opened, highlight the word "model" at the top of document and select Model > Specification; this opens the Specification Tool dialog box. From within this dialog box, select "check model," and, if all parameters are given priors, a message "model is syntactically correct" appears in the bottom left of the screen.

2. Either a single MCMC chain (default) or multiple chains may be run. In our example, we use three chains by typing "3" in the "num of chains" box. Having multiple chains helps the user check whether the MCMC chain converges to the posterior distribution.

3. Highlight the word "list" at the start of the data section and choose "load data" from the Specification Tool dialog box. A message "data loaded" appears in the bottom left of the screen. If an error occurs, it is most often due to misspecification of variables used in the model section (i.e., `N`, `nc`, and `cond` variables in our example code).

4. Select "compile" from the Specification dialog box; if everything is correct, the bottom left of the screen should display the message "model compiled."

5. Select "gen inits" to have WinBUGS generate initializing values for each of the three chains. After the initializing values have been generated, the bottom left of the screen displays the message "model initialized," indicating WinBUGS is ready to run.

Before beginning MCMC sampling, the user must indicate which posterior parameter estimates are to be saved for later analysis. This is done via the Inference > Samples menu, which brings up the Sample Monitor Tool dialog box. Steps 6 and 7 set up monitoring and run the sampling.

6. Type the variable name into the "node" section. For example, to monitor the A parameter, enter "a" into the node section (this parameter was defined as "a" in the model section). You must choose at what iteration to begin and end the monitoring. The value in `beg`, which represents the number of iterations of the MCMC chain that are discarded before monitoring, is commonly referred to as the *burn-in period*. In our examples, we used a burn-in period of 10,000 iterations. Since the MCMC chain begins with inits values that may not represent valid samples from the posterior, a burn-in period is required before the MCMC chain converges to the posterior distribution.

The `end` value represents the length of the MCMC chains; in our example, we set `end` to 21,000, which, if the chains converge, results in 11,000 samples from the posterior distribution. Larger values cause sampling to take longer to complete but provide more information about the posterior. The process is repeated for each parameter the user chooses to monitor. In our example, we monitored all seven parameters (A, b, s, T_{er}, v_E, v_M, and v_H).

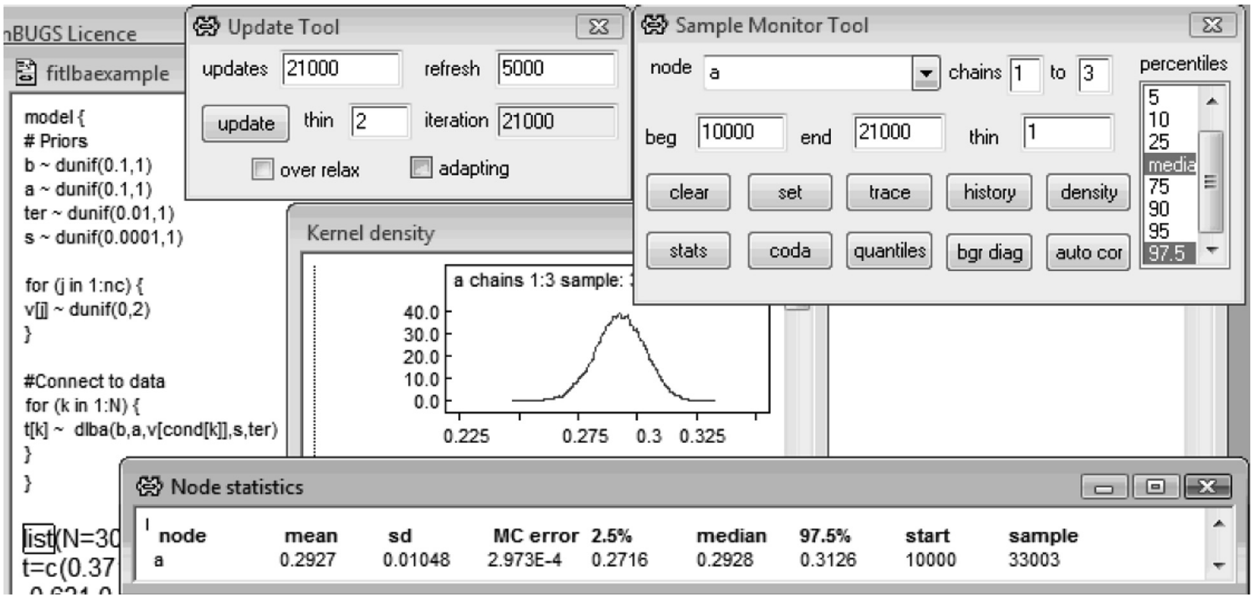
7. Select Model > Update, and the Update Tool dialog box appears. Typically, you will enter the same number in the Updates section that you did in the End section of the Sample Monitor Tool dialog box. Here, you have the option of thinning the MCMC chain. *Thinning* discards iterations in order to reduce autocorrelation among the posterior samples.[4] For example, if "thin" is set to 2 and not 1, every second iteration will be recorded, and it will take twice as long to obtain the number of iterations specified in the updates section. In our example, we set "thin" to 2 for every parameter. The "refresh" section indicates how often WinBUGS updates the screen, which indicates how many iterations of the chain have occurred. Setting a large value reduces processing time. Clicking the update button causes sampling to commence.

While WinBUGS is generating samples, the message "model updating" appears at the bottom left of the screen. This process can take a long time and is uninterruptible, so it is prudent to double-check that all parameters are being monitored and that prior specification is as desired. After WinBUGS has run through the desired number of iterations, a message "Update took *x* secs" appears in the bottom left-hand corner of the screen and results become available for analysis.

To look at the results for each parameter, return to the Sample Monitor Tool. Select the parameter of interest from the node drop-down menu. As soon as a node is selected, statistical and diagnostic options are highlighted. Among the many available choices, we will focus on the "density," "stats," and "compare" options. Figure 6 displays the "stats" and "density" outputs: node statistics and kernel density, respectively, for the *A* parameter. Clicking on the "density" option displays a density plot of the parameter of interest. This is a plot of the posterior estimates returned by WinBUGS for each iteration that was monitored. As soon as the MCMC chain has converged (i.e., when the burn-in period is large enough), this density plot approximates the marginal posterior distribution of the parameter. The quality of the approximation increases as the number of iterations or the length of the MCMC chain increases. Figure 6 shows that, in our example, where 11,000 iterations were used to generate the posterior distribution, the majority of the density plot is close to the true value of .3.

The "stats" option provides commonly used statistics, such as mean and variance, as well as quantile information, for the chosen parameter. Generally, this summary provides the information used to derive parameter estimates for the LBA. Either the median or mean of the posterior distribution can be used as a parameter estimate. When the statistic is distributed symmetrically, as in Figure 6, there is little difference between these estimates. The mode of the posterior distribution equals the maximum-likelihood estimate of the parameter (e.g., as generated by our Excel worksheet). Although WinBUGS does not directly return the mode of the distribution, the "coda" option can be used to save monitored posterior samples to a text file, which can be analyzed in another statistical package to obtain the mode.

The WinBUGS "compare" option, found in the inference drop-down menu, can be used to obtain graphical representations of credible intervals. A credible interval estimates the range, within which, given the data and the


```
model {
# Priors
b ~ dunif(0.1,1)
a ~ dunif(0.1,1)
ter ~ dunif(0.01,1)
s ~ dunif(0.0001,1)

for (j in 1:nc) {
v[j] ~ dunif(0,2)
}

#Connect to data
for (k in 1:N) {
t[k] ~  dlba(b,a,v[cond[k]],s,ter)
}
}

list(N=30
t=c(0.37
```

Update Tool: updates 21000, refresh 5000, thin 2, iteration 21000

Sample Monitor Tool: node a, chains 1 to 3, beg 10000, end 21000, thin 1

```
node   mean     sd       MC error   2.5%     median   97.5%    start   sample
a      0.2927   0.01048  2.973E-4   0.2716   0.2928   0.3126   10000   33003
```

Figure 6. Screen shot from WinBUGS. Shown are the model code, the Update Tool, the Sample Monitor Tool, and output from the density and stats options for the *A* parameter.

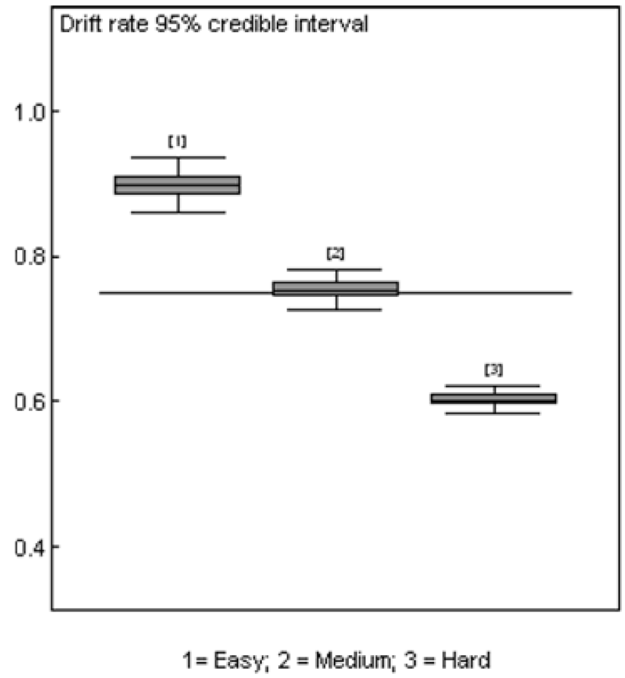


Figure 7. Box-plot representing the 95% credible regions of the drift rate for each of the three conditions, easy (1), medium (2), and hard (3). The line cutting through the center of the plot represents the median of all three conditions.

prior distribution, the true value of a parameter lies. Selecting the "compare" option causes a dialog box to appear that requires at least one variable name to be entered. Type the variable of interest into the top left dialog box, and select "box-plot." This produces a box-plot in which the whiskers represent, by default, the 95% credible interval. The whiskers correspond to the 2.5% (lower whisker) and 97.5% (upper whisker) columns in the node statistics output, because credible intervals are based on the quantiles of the posterior distribution. Figure 7 shows the credible intervals for each of the three drift rates defined in the `v` parameter; the horizontal line going from one side to the other is the group median. The plot also shows that the credible regions do not overlap, suggesting that the drift rates differ from one another.

The Sample Monitor Tool "history" option can be used to check whether the MCMC chain has converged to the posterior distribution. An example of the output produced by this option is shown in Figure 8. The vertical axis of the plot indicates the parameter estimate for *A* for the iteration of the MCMC given by the horizontal axis; collapsing this plot onto the vertical axis gives the density function shown in Figure 6. Each of the chains is represented by a different grayscale shading, and here, the three MCMC chains for the *A* parameter in our example overlap greatly. In other words, they all appear to be random samples from the same distribution throughout the entire chain. This suggests that all chains are samples from the posterior distribution of the *A* parameter. If any of the chains looked systematically different from the others, perhaps showing greater variance or a different mean, it would suggest a lack of convergence of the MCMC chains to the true posterior distribution. The Sample Monitor Tool "auto cor" option can be used to check whether further thinning is needed. It displays the correlation between parameter estimates for iterations *i* and $i - k$, for $k = 1$–50.

Changing model parameterization is very simple within WinBUGS. The user must define a new prior for each of the parameters he or she wants estimated and make a small adjustment to the call to the LBA PDF. For example, to estimate a different nondecision time, T_{er}, for each word frequency condition while fitting lexical decision data, simply augment the WinBUGS model specification to have a vector for the parameter T_{er} with a prior distribution for each of the three frequency conditions and include this extra information in the call to the LBA PDF. Specifically, to make our priors, where we would have previously used `Ter ~ dunif(0.1,1)`, instead we use a `for` loop to set `Ter[k] ~ dunif(0.1,1)` for $k = 1$–3, such as is done for drift rates. Finally, where we would have previously used `t[i] ~ dlba(b,A,v[cond[i]],s,Ter)`, we now use `Ter[cond[i]]`. To use the WinBUGS code that we provide with multiple-choice data would require a substantial change to the code for the LBA PDF, lba

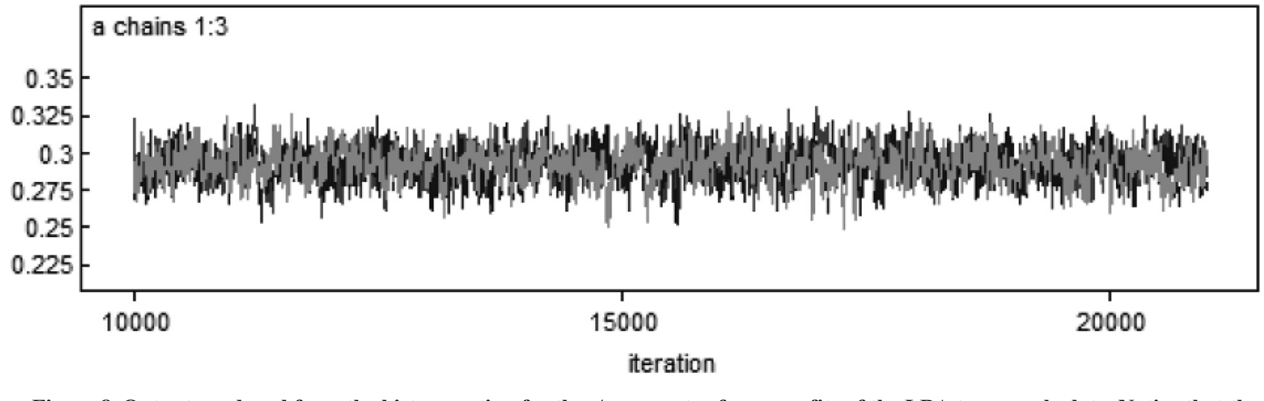


Figure 8. Output produced from the history option for the *A* parameter from our fits of the LBA to example data. Notice that the three chains (indicated by different shades of gray) greatly overlap, indicating that all chains have converged upon the same posterior distribution.

.odc, along the lines of the R code we provide for fitting multiple-choice data, but that is beyond our scope here.

Using R to Create a Graphical Summary Given Parameter Values

We also provide R code that can be used to create a graphical summary of the data and model predictions. This process is useful in determining the appropriateness of the parameter estimates returned by our various methods. The code we provide requires that the user first enter the parameters produced by one of the three methods described above (or indeed, equivalent parameters produced by another method). The user must then source the makegraphs.r file within R, which defines two functions for producing two plots: histograms similar to those described in Example 1 and a quantile probability (QP) plot. The *histplot* and *qpplot* functions provide plots that are suitable for checking parameters and can be adapted to produce figures suitable for publication (see, e.g., Maindonald, 2008).

Ensure, first of all, that the R software is installed (refer to the guide in Example 2, if this has not been done). After the installation completes, extract all of the files in the graphs.zip file into the same folder. This folder should now contain pq-lba.r, lba-math.r, makegraphs.r, and exampledata.txt. Now open R, and make sure that the folder to which the files were extracted is set as the working directory in R (again, see Example 2). The user must first enter the parameters into a vector called `pars` in the following order: s, A, T_{er}, b, v_{E}, v_{M}, v_{H}. The units for A, T_{er}, and b should be in milliseconds. This means that parameters from the WinBUGS version of the LBA, which are returned in seconds, will have to be multiplied by 1,000. Parameter values should be entered as a vector—for example, `pars = c(0.25,300,200, 400,0.9,0.75,0.6)`. The user should then type `source("makegraphs.r")`, which does two things: First, data from the exampledata.txt file are read in, and then two functions, *histplot* and *qpplot*, are defined.

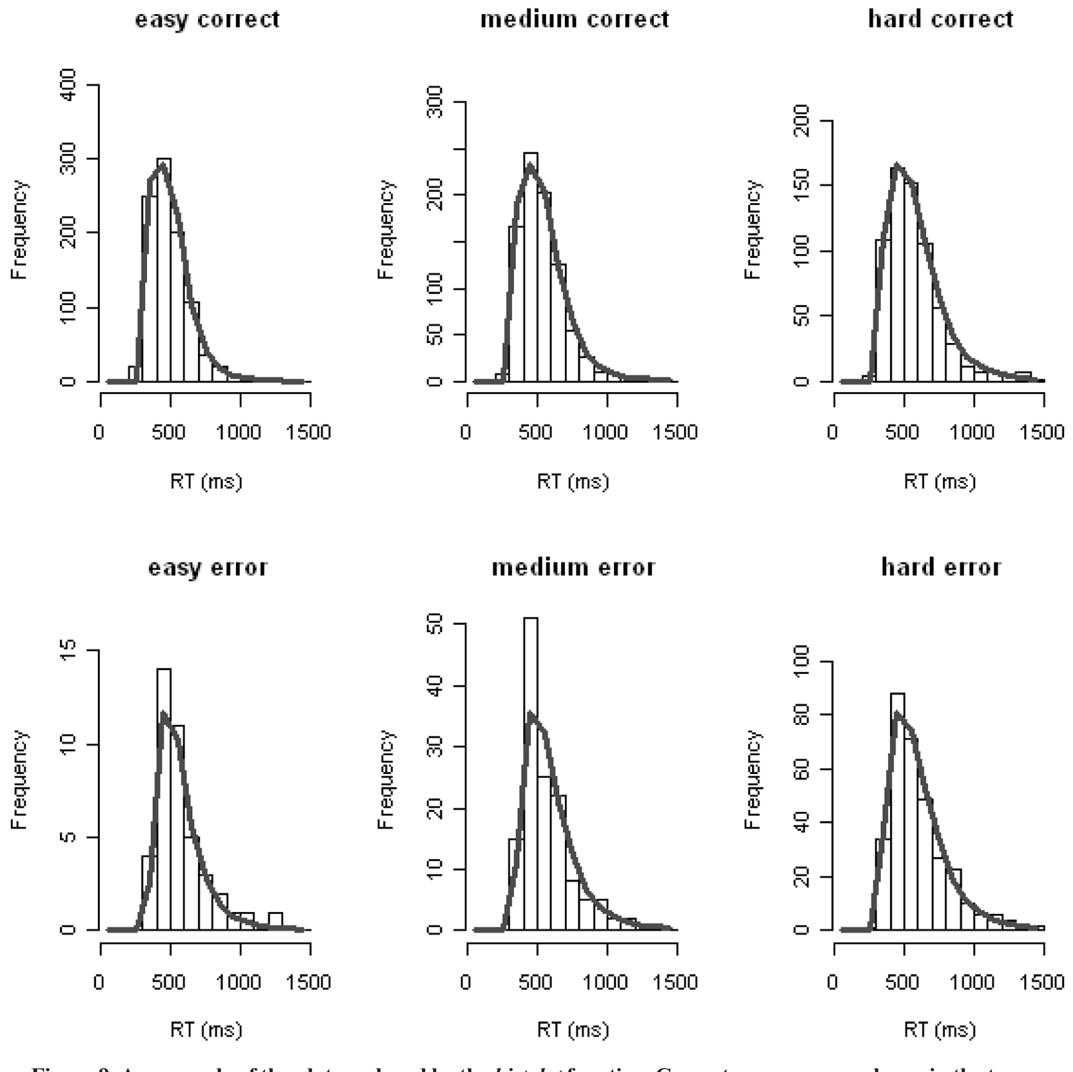


Figure 9. An example of the plot produced by the *histplot* function. Correct responses are shown in the top row; error responses are shown on the bottom row. Difficulty of the decision goes from easy, to medium, to hard from left to right.

The *histplot* function produces a plot that contains six histograms, one for error responses and one for correct responses for each difficulty level. An example of this plot is shown in Figure 9. The data are represented by the black bars of the histogram, with the predictions of the model shown by the solid line. The top row of the plot shows the correct responses, and the bottom row shows histograms for the error responses. From left to right, the order of difficulty of the conditions is easy to hard. The *histplot* function has five arguments. Two are required: `data`, which must be formatted in the way that is produced by the *read .table* function contained within the makegraphs.r script, and `pars`, which must be entered exactly as in the form given above. Three parameters are optional: `minx` and `maxx` define the smallest and largest RT values shown in the histogram, and `bindiff` defines the width (in milliseconds) of the bins of the histogram. It is essential that `bindiff` divide evenly into the difference between `minx` and `maxx`. To create the plots shown in Figure 9, we used the call `histplot(data,pars)`.

The *qpplot* function accepts four arguments. The two that are required, `dat` and `pars`, are of the same form as for the *histplot* function. The two optional arguments, `tmin` and `tmax`, define the fastest and slowest RT data points used to obtain parameter estimates. They are, by default, set at 0 and ∞, respectively, indicating that no data were censored during estimation. Figure 10 shows an example of the QP plot produced by the *qpplot* function. The QP plot gives the majority of the important information shown by the histograms but accomplishes this with one graph by taking all six histograms and summarizing them with five quantile values. The quantiles for each histogram are placed onto the one plot. This results in the accuracies for the correct and error responses for the three difficulty conditions being indicated by the horizontal position of the six dots across the QP plot. The right half of the plot (response probability above .5) shows the correct responses, and the left half (response probability below .5) gives information about the error responses. The vertical positions of the five points above each of these six accuracies refer to .1, .3, .5, .7, .9 quantiles of the RT distribution for each of the correct and error responses in the three difficulty conditions. The quantile values are the proportion of responses under which a given proportion of RTs in the distribution fall (e.g., the .5 quantile is the median). As an example, consider the bottom right point of the plot. The rightmost points of the plot refer to those decisions with the highest accuracy—in other words, the RTs from the correct responses in the easiest condition. Conversely, the leftmost points are the error responses in the easiest condition. The bottom point on the QP plot refers to the .1 quantile of the RT distribution. The .1 quantile of the RT distribution gives the value below which 10% of the RTs fall. Hence, the bottom right point of the QP plot gives the value below which 10% of the RTs for the correct responses in the easiest condition occur. To make the plot shown in Figure 10, we used the call `qpplot(data,pars)`.

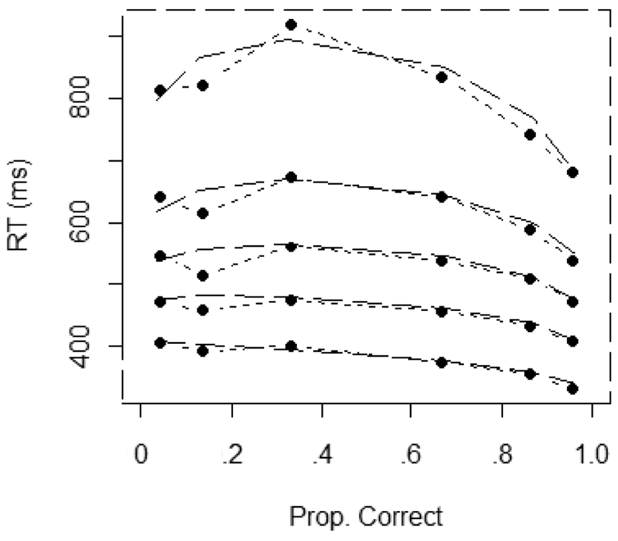


Figure 10. An example of the plot produced by the qpplot function. Proportion correct is shown on the *x*-axis, reaction time (RT, in milliseconds) shown on the *y*-axis. Data is shown by the dotted line with filled points, the LBA predictions are shown by the solid line.

The effect of sample size on parameter estimates. We have provided four methods for fitting the LBA to data, one of those specifically for fitting multiple-choice data. For the other three, we have fit the model to a set of simulated data with 1,000 observations in each of three conditions. In practice, there are often considerably fewer observations per condition. To investigate how well parameters for a two-choice task are recovered by each of our methods for a range of sample sizes, we conducted a simulation study. We simulated 10 sets of data for each of four sample sizes: $N = 50$, 100, 400, and 1,000 observations per condition. The data were simulated using the same parameter values used to generate our example data, and are shown in Table 1.

Table 1 shows the average bias and standard deviation in parameter estimates, expressed as a percentage of the respective parameter value, for each of our three methods: the Excel sheet, the R code, and WinBUGS.[5] For each method, we observe the expected pattern that the bias and standard deviation of parameter estimates increase as sample size decreases. The size and rate at which this happened varied between our methods. When sample size was only 50 observations per condition, the Excel sheet failed to recover parameters. Note, however, that when sample size increased to 100 observations per condition, the parameters were recovered reasonably well even by the Excel sheet, perhaps with the exception of s. Note also that, for the Excel sheet, although there was a reasonable reduction in both bias and standard deviation of parameter estimates when N increased from 100 to 400, the increase from 400 to 1,000 made very little difference. For R and WinBUGS, when N is only 50, the drift rate in high-accuracy condition is overestimated. This is reasonable because, with only 50 samples and high expected accuracy, there are very few error responses in this condition. Note that, for 100 samples per condition or more, there is relatively little bias in parameter recovery for any of the techniques, and the standard deviations for each of the parameters are small and decrease at a rapid rate as N grows.

Table 1
Bias and Standard Deviation (*SD*) of Parameter Estimates, As Percentages of the True Parameter Values, From Three Methods of Fitting the LBA for Four Different Values of *N*, Averaged Over 10 Data Sets

	Samples per Condition	Correct Responses						Start-Point Distribution		Upper Response Boundary		Nondecision Time		Between-Trial Drift-Rate Variability		Average Time (*t*) per Simulation
		Easy (v_E)		Medium (v_M)		Hard (v_H)		(*A*)		(*b*)		(T_{er})		(*s*)		
Method	(*N*)	Bias	*SD*	Bias	*SD*	Bias	*SD*	Bias	*SD*	Bias	*SD*	Bias	*SD*	Bias	*SD*	(sec)
Excel	50	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
R		−11.0	27.0	−2.7	12.0	−3.3	6.7	6.7	33.0	4.3	12.0	−4.0	36.0	−4	20	48
WinBUGS		−7.8	7.8	−9.3	9.3	−3.3	5.0	−4.3	16.0	−4.8	4.5	−1.3	10.0	−16	24	124
Excel	100	5.6	5.6	6.7	2.7	1.7	3.3	5.0	11.0	2.8	4.0	6.0	11.0	28	8	2
R		4.4	8.8	5.3	4.0	0.0	5.0	7.3	10.0	−1.5	7.0	12.0	15.0	12	20	48
WinBUGS		−2.2	4.4	−2.7	4.0	0.0	5.0	1.0	10.0	−1.5	6.3	2.7	6.7	−4	8	231
Excel	400	2.2	2.2	4.0	2.7	1.7	1.7	−2.3	6.0	0.5	2.5	−0.7	2.0	20	4	9
R		0.0	6.7	1.0	5.3	1.7	3.3	2.0	10.0	−0.25	2.4	1.3	14.0	4	8	48
WinBUGS		0.0	3.3	−1.0	2.7	0.0	1.7	0.0	4.3	3.0	9.5	−1.3	5.7	0	8	934
Excel	1,000	4.4	1.1	4.0	1.0	1.7	1.7	0.7	3.0	1.3	1.5	2.0	3.0	20	4	21
R		2.2	2.2	2.7	2.7	0.0	1.7	3.0	5.0	−0.25	1.8	5.0	7.0	8	8	48
WinBUGS		1.1	2.2	0.0	1.0	0.0	1.7	2.0	4.0	2.8	8.3	1.3	3.0	0	4	2,308

Note—Parameter values: $v_E = .9$; $v_M = .75$; $v_H = .6$; $A = 300$; $b = 400$ msec; $T_{er} = 200$ msec; $s = 0.25$. The final column contains the average time, *t*, taken per simulation (in seconds). Times were estimated using a single core of a Pentium quad-core Q6600 2.4-GHz processor.

Fixing parameters across conditions. When estimating model parameters, we made the assumption that only drift rate should vary across conditions. This assumption is the one usually made when the data come from an experiment where the conditions correspond to stimuli presented within subjects and that vary unpredictably from trial to trial. This is because parameters such as *b*, which determines the amount of evidence required to make a response, are thought to be under the strategic control of the participant. Ratcliff (1978) argued that these participant-determined parameters cannot be adjusted on a trial-by-trial basis, depending on which stimulus is presented. If, however, we were to fit data with conditions that varied between blocks of trials or between participants, it is reasonable to expect that parameters such as *b* could vary across these conditions. For example, if participants were instructed to respond as accurately as possible in one block of trials, given a break, and then told to respond with speed emphasis for the next block of trials, we could expect that the participants had been given enough time to adjust their cautiousness in responding by adjusting their *b* parameter.

Because we knew exactly which parameters generated the data in our simulated example, deciding which parameters should vary across conditions was straightforward. In practice, we would not necessarily know which parameters are expected to vary across conditions. Researchers should, therefore, fit a number of versions of the LBA in which we change the parameters that are allowed to vary across conditions and then select the model that provides the best account of our data. This approach is not straightforward, however, because adding parameters always gives a fit that is at least as good as the less complex model, even if the extra parameters overfit (i.e., only accommodate noise in) the data. What is required, therefore, is a measure that improves only if the extra parameters provide a genuine improvement. This is usually accomplished by penalizing a model for having extra parameters. Many such measures exist, but we focus on three easily computed options: the Akaike information criterion (AIC; Akaike, 1974), the Bayesian information criterion (BIC), and the deviance information criterion (DIC; Spiegelhalter, Best, Carlin, & van der Linde, 2002). Each measure uses deviance (−2 times the log likelihood) as its measure of misfit but applies a different complexity penalty. BIC provides the largest penalty for having more parameters: $k \log n$, where k is the number of parameters in the model and n is the number of data points. AIC applies the smallest penalty, $2k$; and DIC, which can be calculated only from Bayesian outputs, applies a penalty that is often somewhere between AIC and BIC in its severity. The DIC measure is based on an estimate of a model's effective number of parameters, *pD*, which takes account of differences in the functional form between models (see Spiegelhalter et al., 2002, for details of the calculation of *pD*). For each of these measures, the model that produces the smallest value is the one that best accounts for the data, given both goodness of fit and model complexity.

To demonstrate these model selection methods, we fit our example data, for which we know that only drift rate varied across conditions to generate the data, with two versions of the LBA: one with only drift rate varying between conditions and another where b, A, T_{er}, and v were allowed to vary across conditions. We report the results of using WinBUGS to estimate parameters here; however, when we used our R code, we found the same pattern of estimates. The deviance for the more complex model was −293.6, compared with −294 for the model in which only drift rates were allowed to vary. In other words, there was very little improvement in the quality of the fit when parameters other than drift rate were allowed to vary across conditions. After adding the various complexity penalties, all three measures of model fit were smaller for the LBA when only drift rate varied across conditions (AIC, −277.99 vs. −277.59; BIC, −238 vs. −190; DIC, −287 vs. −281). This tells us that allowing parameters other than drift rate to vary across conditions gives an increase in quality of fit that is not large enough to warrant the complexity of the extra parameters. Indeed, when we looked at the parameter values estimated in the LBA where b, A, and T_{er} were also allowed to vary, we observed almost no change across difficulty conditions. The same principles can be used to try any number of other parameter constraints, such as allowing fewer parameters to change across conditions.

DISCUSSION

We have provided four methods for fitting the LBA to data—one of those specifically for fitting multiple-choice data. Our aim was to provide the potential user of the LBA with three separate methods for implementing estimation. We (and others, e.g., Wagenmakers et al., 2007) argue that mathematical models of choice, such as the LBA, can provide an important tool for data analysis that can provide much more information about decision processes than the typical ANOVA method applied to RT and accuracy can. We have provided three methods of estimation to data to ensure that the LBA is accessible to users with a range of levels of programming and mathematical abilities. The Excel spreadsheet can be straightforwardly applied to new data that are fairly similar to those from our example data (i.e., a one within-subjects factor). Given R's flexibility and computational power, our R code can be extended to fit accuracy and RT data from almost any experimental setup. However, this requires some programming knowledge and changes, not only to the fit-example.r script, but also to the pq-lba.r code. We included the WinBUGS implementation of the LBA because, as Vandekerckhove et al. (2009) argued, it offers a highly flexible model-fitting framework that is accessible to someone with relatively little computing background. In the simple way we describe above, one can choose which parameters vary across conditions, regardless of the number of conditions or variables. We direct the reader interested in possible hierarchical extensions of the LBA, or diffusion model, to Vandekerckhove et al.'s discussion.

Our intent for this article was to provide multiple ways to apply the LBA to data, but not to compare these methods. As shown in Table 1, all methods recovered parameters quite accurately when applied to data with 100 or more observations per condition. The WinBUGS method provided parameter estimates that were generally the closest match to those used to produce the data. However, the WinBUGS method took, by far, the longest (around 4 h, with the R and Excel methods taking around 1 min). The QMPE method used in the R code is more resilient to smaller sample sizes and outlying data points than are the maximum-likelihood method used in the Excel code and the multiple-choice R code (Heathcote et al., 2002). The Bayesian framework hierarchical methods, which provide parameter estimates at the population level rather than individual participant level, offer an effective way of dealing with small samples per participant when data from a large number of participants are available.

AUTHOR NOTE

Address correspondence to C. Donkin, School of Psychology, University of Newcastle, Callaghan, NSW 2308, Australia (e-mail: chris.donkin@newcastle.edu.au).

REFERENCES

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, **19**, 716-723.

Brown, S. D., & Heathcote, A. (2003). QMLE: Fast, robust, and efficient estimation of distribution functions based on quantiles. *Behavior Research Methods, Instruments, & Computers*, **35**, 485-492.

Brown, S. D., & Heathcote, A. (2005). A ballistic model of choice response time. *Psychological Review*, **112**, 117-128.

Brown, S. D., & Heathcote, A. J. (2008). The simplest complete model of choice reaction time: Linear ballistic accumulation. *Cognitive Psychology*, **57**, 153-178.

Busemeyer, J. R., & Townsend, J. T. (1992). Fundamental derivations from decision field theory. *Mathematical Social Sciences*, **23**, 255-282.

Calin, B. P., & Chib, S. (1995) Bayesian model choice via Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society B*, **57**, 473-484.

Carpenter, R. H. S. (2004). Contrast, probability, and saccadic latency: Evidence for independence of detection and decision. *Current Biology*, **14**, 1576-1580.

Donkin, C., Brown, S. D., & Heathcote, A. (in press). The overconstraint of response time models: Rethinking the scaling problem. *Psychonomic Bulletin & Review*.

Donkin, C., Heathcote, A., Brown, S. D., & Andrews, S. (in press). Non-decision time effects in the lexical decision task. In N. A. Taatgen & H. van Rijn (Eds.), *Proceedings of the 31st Annual Conference of the Cognitive Science Society*. Austin, TX: Cognitive Science Society.

Forstmann, B. U., Dutilh, G., Brown, S. D., Neumann, J., von Cramon, D. Y., Ridderinkhof, K. R., & Wagenmakers, E.-J. (2008). Striatum and pre-SMA facilitate decision-making under time pressure. *Proceedings of the National Academy of Sciences*, **105**, 17538-17542.

Gold, J., & Shadlen, M. N. (2001). Neural computations that underlie decisions about sensory stimuli. *Trends in Cognitive Sciences*, **5**, 10-16.

Hanes, D. P., & Carpenter, R. H. S. (1999). Countermanding saccades in humans. *Vision Research*, **39**, 2777-2791.

Heathcote, A., & Brown, S. D. (2004). Reply to Speckman and Rouder: A theoretical basis for QML. *Psychonomic Bulletin & Review*, **11**, 577-578.

Heathcote, A., Brown, S. D., & Mewhort, D. J. K. (2002). Quantile maximum likelihood estimation of response time distributions. *Psychonomic Bulletin & Review*, **9**, 394-401.

Ho, T. C., Brown, S. D., & Serences, J. T. (2009). Domain general mechanisms of perceptual decision making in human cortex. *Journal of Neuroscience*, **29**, 8675-8687.

Lunn, D. J., Thomas, A., Best, N., & Spiegelhalter, D. (2000). WinBUGS—A Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics & Computing*, **10**, 325-337.

Maindonald, J. (2008). *Using R for data analysis and graphics: Introduction, examples, and commentary*. Web-based article downloaded on 8/12/08 at http://cran.r-project.org/doc/contrib/usingR.pdf.

Mazurek, M. E., Roitman, J. D., Ditterich, J., & Shadlen, M. N. (2003). A role for neural integrators in perceptual decision making. *Cerebral Cortex*, **13**, 1257-1269.

Raftery, A. (1995). Bayesian model selection in social research. In P. V. Marsden (Ed.), *Sociological methodology* (pp. 111-196). Cambridge: Blackwell.

Ratcliff, R. (1978). A theory of memory retrieval. *Psychological Review*, **85**, 59-108.

Ratcliff, R. (2002). A diffusion model account of reaction time and accuracy in a brightness discrimination task: Fitting real data and failing to fit fake but plausible data. *Psychonomic Bulletin & Review*, **9**, 278-291.

Ratcliff, R., Cherian, A., & Segraves, M. (2003). A comparison of macaque behavior and superior colliculus neuronal activity to predictions from models of two-choice decisions. *Journal of Neurophysiology*, **90**, 1392-1407.

Ratcliff, R. Gomez, P., & McKoon, G. (2004). A diffusion model account of the lexical decision task. *Psychological Review*, **111**, 159-182.

Ratcliff, R., & Rouder, J. N. (1998). Modeling response times for two-choice decisions. *Psychological Science*, **9**, 347-356.

Ratcliff, R., & Smith, P. L. (2004). A comparison of sequential sampling models for two-choice reaction time. *Psychological Review*, **111**, 333-367.

Ratcliff, R., Thapar, A., & McKoon, G. (2001). The effects of aging on reaction time in a signal detection task. *Psychology & Aging*, **16**, 323-341.

Ratcliff, R., Thapar, A., & McKoon, G. (2003). A diffusion model analysis of the effects of aging on brightness discrimination. *Perception & Psychophysics*, **65**, 523-535.

Ratcliff, R., & Tuerlinckx, F. (2002). Estimating the parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review*, **9**, 438-481.

Reddi, B. A. J. (2001). Decision making: The two stages of neuronal judgement. *Current Biology*, **11**, 603-606.

Roitman, J. D., & Shadlen, M. N. (2002). Response of neurons in the lateral intraparietal area during a combined visual discrimination reaction time task. *Journal of Neuroscience*, **22**, 9475-9489.

Schall, J. (2001). Neural basis of deciding, choosing and acting. *Nature Reviews Neuroscience*, **2**, 33-42. doi:10.1038/35049054

Smith, P. L. (2000). Stochastic dynamic models of response time and accuracy: A foundational primer. *Journal of Mathematical Psychology*, **44**, 408-463.

Smith, P. L., & Ratcliff, R. (2004). Psychology and neurobiology of simple decisions. *Trends in Neurosciences*, **27**, 161-168.

Smith, P. L., & Ratcliff, R. (2009). An integrated theory of attention and decision making in visual signal detection. *Psychological Review*, **116**, 283-317.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., & van der Linde, A. (2002). Bayesian measure of model complexity and fit. *Journal of the Royal Statistical Society B*, **64**, 583-639.

Vandekerckhove, J., & Tuerlinckx, F. (2007). Fitting the Ratcliff diffusion model to experimental data. *Psychonomic Bulletin & Review*, **14**, 1011-1026.

Vandekerckhove, J., & Tuerlinckx, F. (2008). Diffusion model analysis with MATLAB: A DMAT primer. *Behavior Research Methods*, **40**, 61-72.

Vandekerckhove, J., Tuerlinckx, F., & Lee, M. D. (2009). *Hierarchical diffusion models for two-choice response times*. Manuscript submitted for publication.

Van Zandt, T. (2000). How to fit a response time distribution. *Psychonomic Bulletin & Review*, **7**, 424-465.

Van Zandt, T., Colonius, H., & Proctor, R. W. (2000). A comparison of two response time models applied to perceptual matching. *Psychonomic Bulletin & Review*, **7**, 208-256.

Vickers, D. (1970). Evidence for an accumulator model of psychophysical discrimination. *Ergonomics*, **13**, 37-58.

Voss, A., Rothermund, K., & Voss, J. (2004). Interpreting the parameters of the diffusion model: An empirical validation. *Memory & Cognition*, **32**, 1206-1220.

Voss, A., & Voss, J. (2007). Fast-dm: A free program for efficient diffusion model analysis. *Behavior Research Methods*, **39**, 767-775.

Voss, A., & Voss, J. (2008). A fast numerical algorithm for the estimation of diffusion model parameters. *Journal of Mathematical Psychology*, **52**, 1-9.

Wagenmakers, E.-J., Lee, M. D., Lodewyckx, T., & Iverson, G. (2008). Bayesian versus frequentist inference. In H. Hoijtink, I. Klugkist, & P. A. Boelen (Eds.), *Bayesian evaluation of informative hypotheses* (pp. 181-207). New York: Springer.

Wagenmakers, E.-J., van der Maas, H. L. J., & Grasman, R. P. P. P. (2007). An EZ-diffusion model for response time and accuracy. *Psychonomic Bulletin & Review*, **14**, 3-22.

Wasserman, L. (2000). Bayesian model selection and model averaging. *Journal of Mathematical Psychology*, **44**, 92-107.

NOTES

1. If the Solver option does not appear in the Tools menu, go to Tools > Add-Ins and check the box labeled "Solver Add-in."

2. WinBUGS requires Microsoft Windows, and, although a platform-independent version, OpenBUGS, does exist, the lack of equivalent multiplatform versions of the BlackBox and WBDev software means that our implementation of the LBA into a Bayesian framework is restricted to the Windows operating system.

3. In our example, the priors for the *A*, *b*, and T_{er} parameters are defined in units of seconds. This means that RTs given to WinBUGS must also be in units of seconds. This can be done simply by dividing the RTs in the exampledata.txt file, which are in milliseconds, by 1,000.

4. MCMC chains typically are strongly autocorrelated. Autocorrelation is not a problem for parameter estimation, except that the information contributed to the estimate by each sample is reduced. However, it can be problematic in cases where the variability of samples is important (e.g., for calculation of confidence intervals on estimates).

5. We use the mean of the posterior distribution to determine bias in parameter estimates in WinBUGS. Note that we also could have used an alternate measure of central tendency, such as the median.

SUPPLEMENTAL MATERIALS

The methods for estimating LBA parameters discussed in this article—an Excel worksheet, R scripts, and WinBUGS code—may be downloaded from http://brm.psychonomic-journals.org/content/supplemental.

APPENDIX
Setting Up WinBUGS

WinBUGS can be obtained from www.mrc-bsu.cam.ac.uk/bugs/. To install WinBUGS, download the install file (WinBUGS14.exe). After the download completes, run the executable, and it will, by default, install WinBUGS to the program files folder. Note that the install directory may be different for operating systems other than Windows XP; the reader needs only to take note of their WinBUGS install directory and adjust any future folder references we make. Next, you are required to complete a short registration form that allows a registration key to be sent to the e-mail address you provide. The e-mail contains the registration key and instructions on how to register WinBUGS.

Although WinBUGS has a large number of prespecified distributions for which it can conduct a Bayesian analysis, it does not have the appropriate PDF for the LBA. We have, therefore, provided this in the BugsLBA.zip folder. Making the LBA PDF accessible to WinBUGS necessitates the use of two additional pieces of software: the BlackBox Component Builder and the WBDev. Instructions for their installation are as follows:

1. Extract the lba.odc and Distributions.odc files from the BugsLBA.zip folder.

2. Download the WBDev from www.winbugs-development.org.uk/. From the home page, navigate to the WBDev page and download the software. The contents of the .zip file should be unpacked into the WinBUGS directory. Open the .txt (wbdev_01_09_04.txt at the time of writing) file that you just extracted, and follow the instructions contained in the file to install the WBDev software.

APPENDIX (Continued)

3. Download the BlackBox Component Builder from www.oberon.ch/blackbox.html. In the present article, we refer to Version 1.5. After the download completes, run the SetupBlackBox15.exe file, which installs the BlackBox Component Builder 1.5. This adds a folder to C:\Program Files called BlackBox Component Builder 1.5.

After all of the necessary programs have been downloaded, the next step is to compile the LBA PDF into WinBUGS via BlackBox. After completing the steps below, you will be able to use WinBUGS to fit the LBA to data.

1. Open the WinBUGS directory, copy the entire contents of the WinBUGS folder, and paste them into the newly created BlackBox directory (C:\Program Files\BlackBox Component Builder 1.5\ by default in Windows XP); choose “Yes” to all the “Replace existing file?” requests.

2. Copy lba.odc to the C:\Program Files\BlackBox Component Builder 1.5\WBDev\Mod directory.

3. Open the BlackBox Component Builder program; this should now closely resemble the usual WinBUGS environment. Use File > Open to open lba.odc. Press Ctrl + K to compile lba.odc. An “ok” message should appear in the bottom left corner.

4. Put Distributions.odc into the C:\Program Files\BlackBox Component Builder 1.5\WBDev\Rsrc\ directory. Close down any BlackBox or WinBUGS windows that are still running. The next time BlackBox is run, the LBA PDF should be ready to use.

For more information on the procedure outlined above, as well as on the use of diffusion models in WinBUGS, see Vandekerckhove et al. (2009).

(Manuscript received January 4, 2009;
revision accepted for publication May 25, 2009.)