Behavior Research Methods
2008, 40 (1), 61-72
doi: 10.3758/BRM.40.1.61

Diffusion model analysis with MATLAB: A DMAT primer

Joachim Vandekerckhove and Francis Tuerlinckx
University of Leuven, Leuven, Belgium

The Ratcliff diffusion model has proved to be a useful tool in reaction time analysis. However, its use has been limited by the practical difficulty of estimating the parameters. We present a software tool, the *Diffusion Model Analysis Toolbox* (DMAT), intended to make the Ratcliff diffusion model for reaction time and accuracy data more accessible to experimental psychologists. The tool takes the form of a MATLAB toolbox and can be freely downloaded from ppw.kuleuven.be/okp/dmatoolbox. Using the program does not require a background in mathematics, nor any advanced programming experience (but familiarity with MATLAB is useful). We demonstrate the basic use of DMAT with two examples.

The Ratcliff diffusion model (RDM) has garnered significant attention in recent years as a model for the simultaneous analysis of reaction time (RT) and accuracy data. There are three main reasons for its popularity. First, it can be applied in many fields (Ratcliff, 1978, 1981, 1988, 2002; Ratcliff, Gomez, & McKoon, 2004; Ratcliff & Rouder, 1998, 2000; Ratcliff, Thapar, & McKoon, 2004; Ratcliff, Van Zandt, & McKoon, 1999; Strayer & Kramer, 1994; Thapar, Ratcliff, & McKoon, 2003). Second, it performs extraordinarily well in terms of parsimony and description of interesting patterns in RT data (see, e.g., Ratcliff, 1987; Ratcliff & Rouder, 1998). Third and finally, its main parameters have interesting process interpretations that allow for substantive insights (Voss, Rothermund, & Voss, 2004). If the RDM has one significant drawback, it is that the model is prohibitively difficult to apply in practice, to the point that methods for fitting the RDM to experimental data are a research topic in their own right (Ratcliff & Tuerlinckx, 2002; Tuerlinckx, Maris, Ratcliff, & De Boeck, 2001; Tuerlinckx, 2004; Vandekerckhove & Tuerlinckx, 2007b; Voss & Voss, in press; Wagenmakers, van der Maas, & Grasman, 2007). Only recently, are attempts being made to render the RDM more applicable in research practice (Vandekerckhove & Tuerlinckx, 2007a, 2007b; Voss & Voss, 2007). This article presents a MATLAB toolbox that is exactly such an attempt.

In the next four sections, we will (1) briefly describe the RDM, (2) repeat the basics of matrix methods in statistical modeling, (3) provide some practical information regarding a new tool, the *Diffusion Model Analysis Toolbox* (DMAT), and (4) give two didactic examples with code. For conceptual details regarding the RDM, the fitting procedures, and associated statistical treatment, the reader is referred to Vandekerckhove and Tuerlinckx (2007b).

THE RATCLIFF DIFFUSION MODEL

The basic principle behind the RDM is that of integration of noisy evidence over time. It is assumed that, in order to make a speeded choice between two options, evidence is accumulated sequentially over time. As soon as sufficient evidence toward one option or the other has gathered, the process stops and outputs a decision (*absorbing boundaries*). The accumulation process is governed by two distinct forces—namely, a tendency to drift toward either boundary (*drift rate*) and a stochastic component in the step size and direction on the decision dimension. The process itself is not assumed to be necessarily unbiased: The starting point of the process may be closer to one boundary than the other, increasing the a priori probability of one response. Figure 1 shows a graphical representation of the diffusion process.

In all, the RDM as implemented in DMAT has nine free parameters for each condition. Table 1 lists them, their notation, and their usual interpretation.

Matrix Notation and Design of Experiments

In order to impose restrictions on parameters across conditions, DMAT makes use of a matrix modeling method that is similar to the standard technique of general linear modeling (see Vandekerckhove & Tuerlinckx, 2007b, for a more detailed explanation and examples of this method). In particular, if there are c conditions, a vector $\underset{c\times 1}{\boldsymbol{\psi}}$ of a given type of parameters across conditions is assumed to be the result of the matrix product $\underset{c\times m}{\mathbf{D}} \times \underset{m\times 1}{\boldsymbol{\Theta}}$, given that $\mathbf{D}$ is a design matrix and that $\boldsymbol{\Theta}$ is a vector with free parameters that remain. Crucially, $\boldsymbol{\Theta}$ contains no more elements than does $\boldsymbol{\psi}$ ($m \leq c$), often resulting in a more parsimonious model with fewer parameters to estimate.

J. Vandekerckhove, joachim.vandekerckhove@psy.kuleuven.be

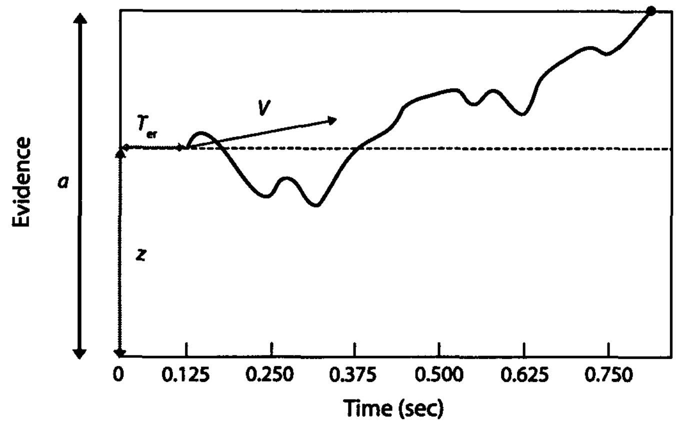

Figure 1. A graphical representation of the diffusion process. The curved line indicates the amount of evidence for the "upper" response as it evolves over time. At about 700 msec, the upper boundary is crossed and the process ends. See the text and Table 1 for more details.

Depending on how the design matrix is formulated, the restrictions change. If, for example, **D** is a column of ones, then for any value of **Θ**, the product **ψ** will be a vector with all elements equal. On the other hand, if **D** is the identity matrix, then **ψ** will be identical to **Θ**, and no restrictions will have been applied. If **D** contains a column of ones and a second column of covariate values, restrictions in the form of a linear regression are applied. The $\hat{\mathbf{\Theta}}$ vector will then contain an estimate of the intercept as its first element and the regression weight of the linear model as its second element.

Of course it is possible to construct even more complex models, such that a parameter may be made dependent on more than one covariate, linear or categorical; on interactions of the covariates; on participants; and so on. Also, different designs may be implemented for different parameters.

DMAT requires that a design matrix be formulated for each parameter of the model. In most cases, however, the design matrix will be a column of ones, indicating no variability across conditions. (This is also the default setting for DMAT models.)

THE DIFFUSION MODEL ANALYSIS TOOLBOX

Requirements

DMAT requires that you have MATLAB 7.2 (Version R2006a) or a more recent version installed. The Optimization toolbox also needs to be installed. If they are available, DMAT will make use of the Statistics and Symbolic Math toolboxes, but they are not required.

The toolbox was developed and tested on Windows and Linux platforms.

Installation

The software tool we are presenting is a MATLAB toolbox that can be freely downloaded via ppw.kuleuven.be/okp/dmatoolbox. Upon filling out a form, you will be e-mailed a link to where a ZIP archive is available. The archive contains some 300 files, 70 of which are MATLAB functions. Unpackage the file to the \toolbox folder of your MATLAB install, and then execute the included installer function from the MATLAB command window. The installer will guide you from there. If you did not unpackage the ZIP archive in the toolbox directory (e.g., because you do not have write access to it), the installer will ask you to locate it first. When the installation is finished, you can test the toolbox by calling the function test_main. Since DMAT is constantly under development, its most recent version, bug reports, and fixes can be found at the Web site above.

End User License Agreement

While the installer runs, you will be asked to read and accept an end user license agreement. Please note that, although DMAT and its source code may be downloaded

Table 1
The Nine Free Parameters of the Ratcliff Diffusion Model, As Implemented by DMAT

	Symbol	Parameter	Interpretation
Decision process	a	Boundary separation	Speed–accuracy trade-off (high a means high accuracy)
	z	Starting point	Bias for either response ($z = a/2$ is neutral)
	v	Drift rate	Amount of input information; Quality of the stimulus
Nondecision	T_{er}	Nondecision time	Sum of all other processes involved (motor RT, encoding . . .)
Intertrial variability	s_z	Intertrial range of z	Participant's variability in bias
	s_t	Intertrial range of T_{er}	Participant's variability in nondecision time
	η	Intertrial *SD* of v	Variability in stimulus quality, or variability in attention or motivation
Mixture model	π	Proportion nonoutliers	Proportion of data resulting from a diffusion process
	γ	Proportion guesses	Proportion of outliers that are guesses (and not due to delayed startup)

at no cost, it is *not* permitted to redistribute the code or derived code without the authors' consent. We welcome cooperation from third parties in further developing DMAT, but in order to maintain transparency regarding exactly which methods an end user has implemented, we want the distribution of this toolbox to remain centralized.

USE AND EXAMPLES

Two Interfaces

DMAT features both a graphical user interface (GUI) and a command interface (CI). To start the GUI, simply type "dmatgui" in the MATLAB command window. In the examples, we will focus mainly on the CI, which requires some coding. Using the GUI should be largely self-evident if the commands for the CI are known. Where it is not self-evident, we will explicitly mention how the GUI expects and handles user input.

Data Sets

For either interface, the data set should be provided in a proper format, meaning that it should be a three-column matrix in which each row indicates a trial in the experiment. Of the three columns, the first contains the condition, the second the response type (0 or 1, usually meaning incorrect and correct), and the third the RT in seconds. If the data matrix contains only two columns, all trials will be assumed to be in the same condition. If you use the CI, the data should be contained in a *double array*. For the GUI, it can be a double array in a MATLAB file (.mat) or ASCII data in a tab-delimited file (.tab or .dat), a comma-delimited file (.csv), or a space-delimited file (.txt).

General Use of the Toolbox: Command Interface

Input. The most important function in DMAT is called "multiestv4." This function accepts as its first input a data set (as a three-column double array) and as second input an optional *options structure*. A large part of using DMAT is constructing this options structure, a MATLAB variable with many fields containing different possible settings. Table 2 gives an overview of the settings (field names), their default values, and their effects. The default options structure can be obtained by calling the function multiestv4 without input arguments. Then you can edit the fields of this structure to fit your needs. The standard syntax for this is

```
opts = multiestv4();
opts.fieldname = value;
```

and then, to estimate the parameters,

```
output = multiestv4(data,opts);
```

This will return an *output structure*, which contains information about the model fit and the optimization algorithm.

Output. Like the options structure, the output structure is a MATLAB variable that has many different fields, each one containing information about the model fit or the al-

Table 2
The Fields of DMAT's Options Structure, With Default Values and Effects

Field Name	Default Value (multiestv4)	Effect/Use
DesignMatrix	`<Columns of ones>`	Parameters allowed to vary across conditions?
Display	`'off'`	How much output should DMAT give to the command window?
EWMA	`<structure>`	Provide parameters for the EWMA procedure. Defaults are $\lambda = .01$, $L = 1.5$, $s = .5$.
EstimationMethodScalar	5 `(Multinomial likelihood with fixed bin edges)`	Objective function to minimize (Multinomial likelihood or χ^2? Fixed bin edges or percentiles?).
FixedBinEdges	`[.30,.36,.42,.52,.80; .38,.47,.56,.70,1.0]`	If fixed bin edges, which values to use (in seconds, first row for corrects, second for errors).
FixedValues	`[ ]`	Provide a condition × parameter matrix with NaN for free parameters and a specific value for fixed ones.
Guess	`[ ]`	Starting position for the optimization (condition × parameter matrix). If none given, DMAT finds one.
GuessMethodScalar	1	If DMAT has to generate guess, 1 causes it to use ezdiff* and 2 uses a slight perturbation of ezdiff's output.
LongSimplexRuns	1	The number of times the long simplex run should be repeated.
MaxIter	5000	The maximum number of iterations for long simplex runs.
Name	`'No name given'`	A name for the model.
NoFitting	0	If set to 1, only construct objective function, no actual fitting.
NonparametricBootstrap	0	Number of nonparametric bootstrap iterations.
ObjectiveDecimals	7	Number of significant decimals for the objective function.
OutlierMax	`[ ]`	Maximum RT for inclusion.
OutlierMin	`[ ]`	Minimum RT for inclusion.
OutlierTreatment	`'None'`	Which outlier treatment to use.
ParameterDecimals	7	Number of significant decimals for parameters.
ParametricBootstrap	0	Number of parametric bootstrap iterations.
Percentiles	`[10,30,50,70,90; 10,30,50,70,90]`	If estimation with percentiles, which ones to use (values between 0 and 100; first row for corrects, second for errors).
ShortSimplexRuns	3	The number of times the short simplex run (200 iterations) should be repeated.
SpecificBias	`[ ]`	Per condition, value of B, where $B = z/a$. If NaN, z and a are estimated separately.

*The ezdiff function implements the EZ-diffusion algorithm, which is described in Wagenmakers et al. (2007) and in Vandekerckhove and Tuerlinckx (2007b).

Table 3
The Fields of DMAT's Output Structure, With Brief Explanations of Their Contents

Field Name	Contents
DesignVector	Point estimates of the free parameters
Df	Number of free parameters
FitInfo	Fit indices of the model
Fitvalue	Deviance of the model
Hessian	Estimate of the Hessian matrix at the minimum
Minimum	Estimate of the entire parameter set
Name	Name of the model
NonparametricBootstrapMean	If requested, nonparametric bootstrap estimate of entire parameter set
NonparametricBootstrapStdErr	If requested, nonparametric bootstrap estimate of parameters' standard errors
NonparametricBootstraps	An output structure for each nonparametric bootstrap iteration
Options	The options structure that the user provided
OutlierReport	If requested, information regarding outlier treatment
ParametricBootstrapMean	If requested, parametric bootstrap estimate of entire parameter set
ParametricBootstrapStdErr	If requested, parametric bootstrap estimate of parameters' standard errors
ParametricBootstraps	An output structure for each parametric bootstrap iteration
Simplex	Information regarding the simplex runs (convergence time, number of steps . . .)
StdErr	Estimate of the parameters' standard errors (based on the Hessian matrix)
Time	Total time needed for fitting this model
Warnings	Anything the user might need to know (e.g., if the Hessian indicates poor model fit)

gorithm. Table 3 contains information about the available fields and what they mean. Output fields can be browsed from the command window. The following syntax will display the contents of the field *fieldname*:

```
output(model).fieldname,
```

where *model* refers to the numerical index of the model you are investigating. For example, output(2).Fitvalue will return the value of the deviance function of the second model, and output(2).Df will return the number of free parameters in the model.

Further processing. If two models are nested, the difference in their deviances follows a chi-square distribution with a number of degrees of freedom equal to the difference in number of parameters, under the null hypothesis that the models are equal. Thus, the following syntax (using DMAT's chi2test function) will give the *p* value of the difference between two models:

```
>> x2 = output(1).Fitvalue-output(2).
   Fitvalue;
>> df = output(2).Df-output(1).Df;
>> p = chi2test(x2,df);
```

A convenient function in this regard is qtable, which shows different fit values of models, and also shows the *p* value of the difference in fit between each pair of consecutive models. (Note that this implies that the *p* values reported in a certain row are only correct if that model is nested in the preceding model.)

General Use of the Toolbox: Graphical User Interface

Input. In the GUI, you first need to load the data by clicking the "Browse" button and finding the data set (which you saved somewhere). When you have set all the options to your liking, the model can be added to the model queue by clicking the "Current Model" button. The model queue stores series of models that can be fitted in a batch submission. This is often advantageous, since each set of parameter estimates can be used as an initial guess of the next model, resulting in an increase in efficiency. In particular, if subsequent models are nested, then the parameter estimates of the more restrictive model will often be a good starting point for the less restrictive model. DMAT will perform the necessary linear transformations automatically.

Then you can start to define the next model and click "Current Model" again when you are finished. Click "Run" to start parameter estimation, or save the model queue with the "Save" button. Note that in the "Save As . . ." window you can choose to save the queue either in a DMAT native format (*.dmq) or as an ASCII M-file that can be viewed, edited, and run from the command line. Both *.dmq files and generated M-files can be reloaded into the GUI at a later time by clicking the "Load" button (but you have to load a data set first, and the models in the loaded queue have to be appropriate for that data set—i.e., have the same number of conditions). Note that DMAT, if able, automatically makes emergency back-ups (both of the most recent queue and of intermediate estimation results), and if something should go wrong, calling the dmatrescue function from the command interface might bring relief.

Output. As soon as the algorithm has started, an output window will replace the DMAT main window, allowing you to browse some descriptive statistics. The window will be updated as results from the queue become available. You can simply select the model and output type from two lists on the left-hand side.

Simulating Data

In the examples that follow, we will use simulated data. DMAT contains several functions that allow you to simulate data sets that are ready for use. Appendix A shows a simple sequence of commands that will produce a data set with three conditions, which differ only in drift rate. In the GUI, simply click the "Simulator" button. After you enter the number of conditions, click "Set" and then input the parameter set and the number of data points desired. Then click "Simulate and Save," select a file name and location, and click "Close." You cannot input a seed for the random number generator within the GUI.

Example 1: A Simple Design

Data set. For our first example, we will imagine an experiment with three conditions. The difference between conditions is supposed to be in the quality of a presented stimulus; hence, we are interested in the difference in drift rates. For our data set, we will use the one that results from the code in Appendix A.

Input. To analyze these data, we will attempt to fit two models. In one model, we will make the assumption that all drift rates are equal (the "reduced model" or "null model"). In the second model, we will relax that assumption. By comparing the goodness of fit of these models, we can investigate the effect of condition on drift rate (much as we would in an ANOVA). As explained in the section on matrix representations, constraining equality across conditions implies a design matrix that is a column of ones. Applying no constraints on a given parameter can be achieved by using the appropriate identity matrix as a design matrix.

Appendix B shows code for fitting these two models. Note that the only setting we have adapted is the DesignMatrix field. Usually, the rest of the default settings provided by multiestv4 are well suited.

In the GUI, you can simply click "Current Model" as soon as the data set is loaded, since the default model doesn't need to be changed for Model 1. For Model 2, the design matrix for drift rate (v) needs to be changed to an identity matrix. Click the drop-down menu next to "View/Edit design matrix" and select the "v" option. In the window that pops up, you can choose to manually input an identity matrix, or simply select "Identity" in the drop-down menu. Confirm with OK and click "Current Model" to add the second model to the queue.[1] Finally, click "Run" to start parameter estimation.

Output. We can inspect the best-fitting parameter set of the second model in the usual way:

```
>> output(2).Minimum
ans =
  Columns 1 through 4
    0.1520   0.3000   0.0610   0.1120
    0.1520   0.3000   0.0610   0.1120
    0.1520   0.3000   0.0610   0.1120
  Columns 5 through 7
    0.0150   0.1020   0.4030
    0.0150   0.1020   0.1750
    0.0150   0.1020   0.0030
```

Statistical processing. We can find the p value of the difference in deviance between the two models (since we know the number of parameters to be estimated increased by two from Model 1 to Model 2):

```
>> x2 = output(1).Fitvalue-output(2).
   Fitvalue
x2 = 945.317
>> p = chi2test(x2,2)
p = 0
```

This result indicates that there is a highly significant effect of condition on the drift rate—which is as we expected. To get a quick summary of the model fits, use the qtable function (see Appendix B for the output that would yield).

Example 2: A More Complicated Design

Data set. For our second example, we simulate data with eight conditions. Conditions 1–4 contain an accuracy instruction, and 5–8 contain a speed instruction (influencing the boundary separation). A second manipulation, within these groups, pertains to the quality of the stimulus (influencing drift rate). This time, however, the manipulation is a continuous covariate X, which takes the values 0.2, 0.8, −0.4, and 0.6. In Appendix C, we present code that will generate such data.

Input. We will construct a queue of three models, differing only in the design matrices. In the first model, we will apply no design (all parameters constrained to be equal across conditions). In the second model, the experimental design is implemented (allowing the first four conditions to have a different boundary separation from the last four, and constraining drift rate to be a linear function of the covariate X). In the third model, we allow both boundary separation and drift rate to vary freely across conditions (to test for deviation from the design). Furthermore, we will tweak some settings of the fitting algorithm. First, we no longer want to use fixed RT bins, but rather *quantile probability products* (Brown & Heathcote, 2003; Heathcote & Brown, 2004; Heathcote, Brown, & Mewhort, 2002; see also Vandekerckhove & Tuerlinckx, 2007b). To this end, we give the EstimationMethodScalar option the value 6 (see DMAT's emstable function for a table of valid values for this setting). We choose the classical percentiles 10, 30, 50, 70, and 90, and add the first and fifth percentiles to achieve a better fit of the left slope of the RT distribution. Finally, we choose not to estimate the starting point z, but rather fix it to $a/2$, since nothing in the experiment has given us cause to assume an a priori bias on behalf of the fictional participant. Appendix C shows all the code needed to provide DMAT with this input. In the GUI, the estimation method and percentiles can be set in the Advanced window, and the bias can be set via the "Bias: Set" button in the main window. The specialized design matrices have to be input manually (or pasted from an external editor).

Two things may be mentioned regarding the code in Appendix C. First, note the use of two shortcuts that are available in coding the design matrices. Supplying `'1'` (of data type "char array") for a design matrix will be interpreted as a column of ones, and supplying [ ] (the empty array) will be interpreted as an identity matrix (which are shortcuts in the sense that you don't need to figure out exactly how large these matrices ought to be.) Second, note that in order for DMAT to recognize the "specified bias" (the restriction that $z = a/2$), we need to supply a nonrestrictive design matrix for z. Otherwise, the design matrix restrictions will override the specified-bias restriction (DMAT will print a warning when this happens). The nonrestrictive design of course means the identity matrix (or its shortcut, the empty matrix).

Table 4
Output of the qtable Function in Example 2

Deviance	*d-Deviance*	*df*	*d-df*	*AICc*	*BIC*	*p*
20206.10	NaN	6	NaN	20218.12	20255.24	NaN
16118.72	4087.38	8	2	16134.75	16184.23	0.00
16101.73	16.99	20	12	16141.95	16265.51	0.15

Note—*Deviance* is the badness-of-fit measure, *d-Deviance* is the difference between consecutive models, *df* is the number of parameters in each model, *d-df* is the difference with the previous model, *AICc* is the small-sample Akaike information criterion (Hurvich & Tsai, 1989), *BIC* is the Bayesian information criterion, and *p* is the significance of the difference between consecutive models (based on a χ^2 test of *d-Deviance* with *d-df* degrees of freedom).

Output. We can see the differences between the three models by calling the qtable function. In the resulting table, shown as Table 4, we can see that Model 2 outperforms Model 1 [$\chi^2(2) = 4{,}087.38$, $p < 10^{-10}$], indicating a significant effect of the experimental design, but Model 3 does not outperform Model 2 [$\chi^2(2) = 16.99$, $p = .1501$]. Model 2 also has the lowest AIC and BIC values. Finally, Figure 2 is a screen shot of the GUI, showing drift rate as a function of condition. The error bars are in this case based on the Fisher information matrix (which is derived from the Hessian matrix). If bootstrap analyses had been requested, bootstrap confidence intervals would be shown.

SUMMARY

In this article, we have presented DMAT, an application with a low ease-of-use threshold that enables fitting and evaluation of the Ratcliff diffusion model. We briefly described the model and the basic matrix methods used in general linear modeling, which we have extended for use in diffusion models. With two didactic examples, we demonstrated the use of DMAT.

AUTHOR NOTE

This research was supported by Grants GOA/00/02, GOA/2005/04, and IUAP P5/24 from the Research Council of the University of Leu-

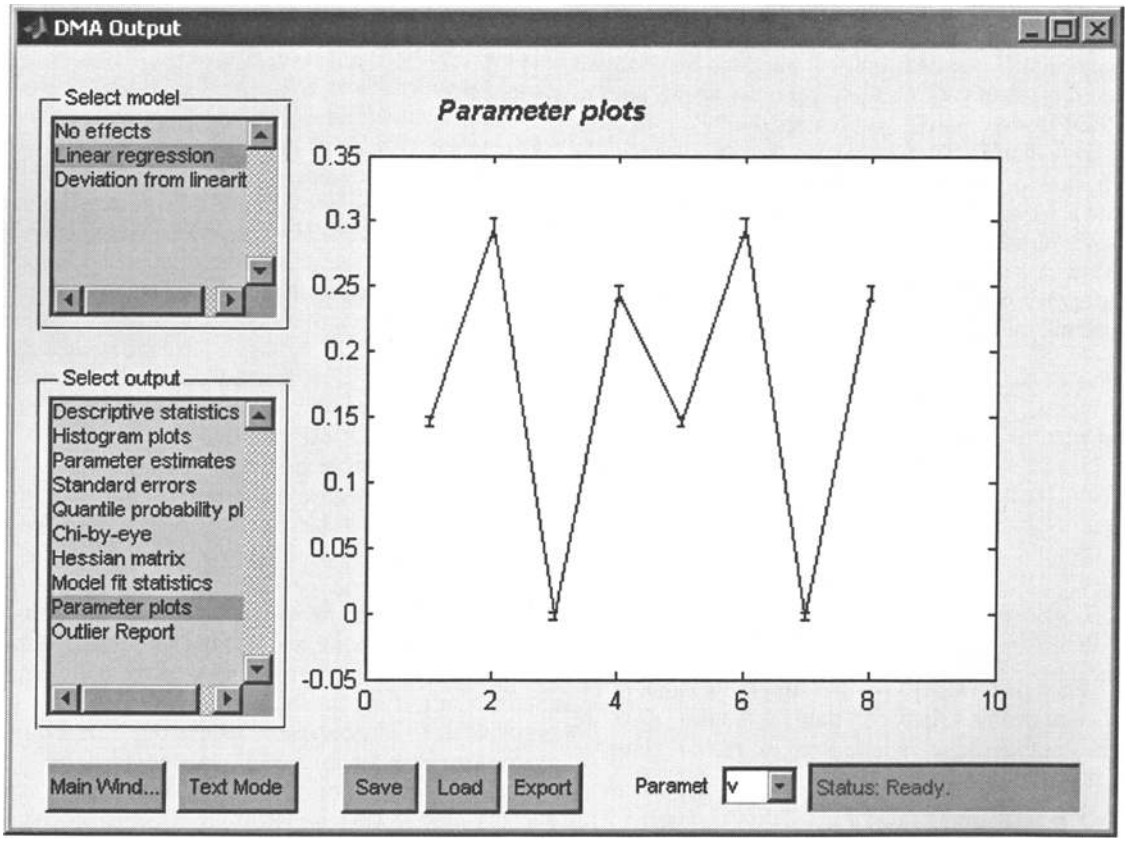

Figure 2. A screen shot from DMAT. Parameter estimation is complete, and the user can browse through the output. The graph shows estimated drift rates (and error bars of one standard error, obtained from the Hessian matrix) as a function of condition, as obtained under the model that allows deviation from the linear design.

ven. The authors thank Eric-Jan Wagenmakers and Andrew Heathcote for insightful comments on an earlier draft. Correspondence related to this article may be sent to J. Vandekerckhove, Research Group Quantitative Psychology and Personality Psychology, Department of Psychology, Tiensestraat 102, B-3000 Leuven, Belgium (e-mail: joachim .vandekerckhove@psy.kuleuven.be).

REFERENCES

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

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

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

Hurvich, C., & Tsai, C.-L. (1989). Regression and time series model selection in small samples. *Biometrika*, **76**, 297-307.

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

Ratcliff, R. (1981). A theory of order relations in perceptual matching. *Psychological Review*, **88**, 552-572.

Ratcliff, R. (1987). More on the speed and accuracy of positive and negative responses. *Psychological Review*, **94**, 277-280.

Ratcliff, R. (1988). Continuous versus discrete information processing: Modeling the accumulation of partial information. *Psychological Review*, **95**, 238-255.

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

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

Ratcliff, R., & Rouder, J. (2000). A diffusion model account of masking in letter identification. *Journal of Experimental Psychology: Human Perception & Performance*, **26**, 127-140.

Ratcliff, R., Thapar, A., & McKoon, G. (2004). A diffusion model analysis of the effects of aging on recognition memory. *Journal of Memory & Language*, **50**, 408-424.

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

Ratcliff, R., Van Zandt, T., & McKoon, G. (1999). Connectionist and diffusion models of reaction time. *Psychological Review*, **106**, 261-300.

Strayer, D., & Kramer, A. (1994). Strategies and automaticity: I. Basic findings and conceptual framework. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **20**, 318-341.

Thapar, A., Ratcliff, R., & McKoon, G. (2003). A diffusion model analysis of the effects of aging on letter discrimination. *Psychology & Aging*, **18**, 415-429.

Tuerlinckx, F. (2004). The efficient computation of the distribution function of the diffusion process. *Behavior Research Methods, Instruments, & Computers*, **36**, 702-716.

Tuerlinckx, F., Maris, E., Ratcliff, R., & De Boeck, P. (2001). A comparison of four methods for simulating the diffusion process. *Behavior Research Methods, Instruments, & Computers*, **33**, 443-456.

Vandekerckhove, J., & Tuerlinckx, F. (2007a). Diffusion Model Analysis Toolbox [Software and manual]. Retrieved from ppw .kuleuven.be/okp/dmatoolbox.

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

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. (in press). A fast numerical algorithm for the estimation of diffusion-model parameters. *Journal of Mathematical Psychology.*

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

NOTE

1. Actually, we have built some shortcuts into the DMAT GUI that make this example even easier. Load the data, click "Predefined," select "No effects" and "Effect on v only," and click "Add" and then "Run."

(Continued on next page)

APPENDIX A
Annotated Code for Generating Random Data in DMAT

(1) Define a Parameter Set From Which to Sample

Parameter sets are matrices with one row for each condition, and one column for each parameter. The parameters are a, Ter, eta, z, sz, st, v:

```
parameter_set = [.16 .30 .08 .12 .02 .10 .40
                 .16 .30 .08 .12 .02 .10 .20
                 .16 .30 .08 .12 .02 .10 .00];
```

(2) Define Simulation Constants

There are two more constants that the simulation will need. First is the number of data points in each condition, N. Note that you can choose between providing N as a scalar, like here, or as a vector with one number for each condition, for instance, N = [500 500 1000]. Second, since the simulator makes use of random number generators, a "seed" is needed. If you provide none, a seed is selected with the help of the system clock. For reproducibility, we choose a seed here:

```
N = 1000;
seed = 0;
```

(3) Simulate Data

With the multisimul function:

```
   data = multisimul(parameter_set,N,seed);
   disp(data(1:10,:))

 3.0000    1.0000    0.3788
 3.0000         0    0.5991
 3.0000    1.0000    0.4741
 1.0000    1.0000    0.4921
 2.0000    1.0000    0.6802
 3.0000    1.0000    0.4296
 3.0000         0    0.7973
 3.0000    1.0000    0.4302
 2.0000    1.0000    0.7328
 3.0000    1.0000    0.3282
```

A data set for DMAT usually has three columns, the first indicating condition, the second response, and the third RT (in seconds).

APPENDIX B
Annotated Code for Fitting Two Nested Diffusion Models in DMAT

(1) Prepare the Options Structure

Get the default options structure:

```
   options = multiestv4

options =
             DesignMatrix: {'1'  '1'  '1'  '1'  '1'  '1'  '1'}
                  Display: 'off'
                     EWMA: [1x1 struct]
 EstimationMethodScalar: 5
            FixedBinEdges: [2x5 double]
              FixedValues: []
                    Guess: []
       GuessMethodScalar: 1
         LongSimplexRuns: 1
                  MaxIter: 5000
                     Name: 'No name given'
                NoFitting: 0
 NonparametricBootstrap: 0
       ObjectiveDecimals: 7
               OutlierMax: []
               OutlierMin: []
        OutlierTreatment: 'None'
```

APPENDIX B (Continued)

```
    ParameterDecimals: 7
  ParametricBootstrap: 0
          Percentiles: [2x5 double]
     ShortSimplexRuns: 3
         SpecificBias: []
```

(2) Copy It, Because We Want to Fit More Than One Model

```
options = repmat(options,2,1);
```

(3) Prepare the Design Matrix for Each Model

In Model 1, all parameters are kept equal across conditions. In Model 2, drift rate is allowed to vary freely. To facilitate estimation and to make sure the qtable function will yield correct p values, we put the more restrictive model first and the less restrictive one second.

```
O = ones(3,1);
I = eye(3);
design_matrix1 = {O,O,O,O,O,O,O,O,O};
design_matrix2 = {O,O,O,O,O,O,I,O,O};
```

(4) Insert the Design Matrices Into the Options Structure

```
options(1).DesignMatrix = design_matrix1;
options(1).Name = 'No effect';
options(2).DesignMatrix = design_matrix2;
options(2).Name = 'Effect on v only';
```

(5) Invoke DMAT

Use the multiestv4 function to get parameter estimates:

```
   output = multiestv4(data,options);

   Guess     :   13344.70746657     (19-Dec-2006 15:59:48)
   Simplex  1:   11711.68971805     (19-Dec-2006 15:59:52)
   Simplex  2:   11684.35157724     (19-Dec-2006 15:59:56)
   Simplex  3:   11684.34775252     (19-Dec-2006 16:00:01)
   Simplex  4:   11684.34774483     (19-Dec-2006 16:00:10)
   Final loss:   11684.34774483     (19-Dec-2006 16:00:13)

   The recovered sZ parameters are suspect.
   Trying again.

   Guess     :   11730.20083810     (19-Dec-2006 16:00:13)
   Simplex  1:   11685.51405192     (19-Dec-2006 16:00:17)
   Simplex  2:   11684.35679628     (19-Dec-2006 16:00:22)
   Simplex  3:   11684.34780944     (19-Dec-2006 16:00:27)
   Simplex  4:   11684.34774486     (19-Dec-2006 16:00:37)
   Final loss:   11684.34774486     (19-Dec-2006 16:00:39)

   The last convergence point was still a suspect result.
   Returning to the best point found and giving up.

   Guess     :   11684.34774483     (19-Dec-2006 16:00:40)
   Simplex  1:   10826.07763855     (19-Dec-2006 16:00:45)
   Simplex  2:   10748.37849361     (19-Dec-2006 16:00:50)
   Simplex  3:   10740.89192211     (19-Dec-2006 16:00:54)
   Simplex  4:   10739.06920367     (19-Dec-2006 16:01:07)
   Final loss:   10739.06920367     (19-Dec-2006 16:01:10)

   The recovered sZ parameters are suspect.
   Trying again.

   Guess     :   10763.99776143     (19-Dec-2006 16:01:10)
   Simplex  1:   10739.20957907     (19-Dec-2006 16:01:15)
   Simplex  2:   10739.03689509     (19-Dec-2006 16:01:20)
```

APPENDIX B (Continued)

```
    Simplex  3:   10739.03193134      (19-Dec-2006 16:01:25)
    Simplex  4:   10739.03044311      (19-Dec-2006 16:01:39)
    Final loss:   10739.03044311      (19-Dec-2006 16:01:42)
```

(6) Process Results
Use the qtable function to get a concise summary of the model queue:

```
qtable(output)
```

```
--------------------------------------------------------------------------
   Deviance  d-Deviance    df   d-df         AICc          BIC          p
--------------------------------------------------------------------------
 11684.3477         NaN     7    NaN   11698.3758   11740.3923        NaN
 10739.0304    945.3173     9      2   10757.0786   10811.0878    0.00000
--------------------------------------------------------------------------
```

APPENDIX C
Annotated Code for Generating the Data Set and Estimating the Models Described in Example 2

Simulating the Data

(1) Define a Parameter Set From Which to Sample
We will define a more complex design for this example. We will suppose eight different conditions. Conditions 1–4 contain an accuracy instruction, and 5–8 contain a speed instruction. A second manipulation, within these groups, again pertains to the quality of the stimulus. This time, however, it is a continuous variable that takes the values .2 .8 –.4 .6. The following parameter set is roughly what we would expect from such an experiment:

```
parameter_set = [.24 .30 .08 .12 .02 .10 .15
                 .24 .30 .08 .12 .02 .10 .30
                 .24 .30 .08 .12 .02 .10 .00
                 .24 .30 .08 .12 .02 .10 .25
                 .08 .30 .08 .04 .02 .10 .15
                 .08 .30 .08 .04 .02 .10 .30
                 .08 .30 .08 .04 .02 .10 .00
                 .08 .30 .08 .04 .02 .10 .25];
```

(2) Define Simulation Constants
Let's say that the number of data points per condition wasn't equal here:

```
N = [470 440 500 450 430 460 400 450];
seed = 0;
```

(3) Simulate Data

```
    data = multisimul(parameter_set,N,seed);
    disp(data(1:10,:))

  4.0000    1.0000    0.6167
  2.0000    1.0000    0.8900
  7.0000    1.0000    0.4017
  6.0000    1.0000    0.5786
  3.0000         0    1.0389
  6.0000    1.0000    0.4326
  7.0000         0    0.3262
  1.0000    1.0000    0.4725
  3.0000         0    1.5550
  2.0000    1.0000    0.5454
```

Fitting the Model

(1) Prepare the Options Structure
Get default options structure.

```
options = multiestv4;
```

APPENDIX C (Continued)

(2) Copy It, Because We Want to Fit More Than One Model

```
options = repmat(options,3,1);
```

(3) Adapt All of the Options Structures at Once to Change Some Settings

We want to use a percentile-based method instead of a fixed-bins method. (The deal function changes all fields with a given name in an array of structures simultaneously. Note the required use of [] on the left-hand side here.)

```
[options.EstimationMethodScalar] = deal(6);
```

Setting this field to 6 indicates that we want a multinomial likelihood estimation based on percentiles (quantile probability products estimation). A table with possible values for EstimationMethodScalar can be requested with the emstable function.

Since we indicated that we want to estimate on the basis of percentiles, we need to indicate which percentiles to use. Classically, percentiles 10, 30, 50, 70, and 90 are used, but adding some smaller values increases recovery of the left slope of the RT distribution.

```
[options.Percentiles] = deal([1 2 5 10 30 50 70 90
                              1 2 5 10 30 50 70 90]);
```

We also don't want to estimate the starting point, but to fix it to exactly half of the boundary separation, in all conditions:

```
[options.SpecificBias] = deal([.5 .5 .5 .5 .5 .5 .5 .5]);
```

(4) Prepare the Design Matrix for Each Model

In Model 1, all parameters are kept equal across conditions. However, we do need to make a change in the design matrices. Since we supplied an extra restriction (namely, that all z = a/2), we need to remove the design matrix restriction to avoid a conflict. Thus:

```
options(1).DesignMatrix = {'1','1','1',[],'1','1','1','1','1'};
options(1).Name = 'No effects';
```

Note also that here we make use of two shortcuts built into the code: Supplying '1' instead of a design matrix restricts that parameter to being equal across conditions. Supplying an empty matrix ([]) allows it to vary without restriction.

In Model 2, drift rate is allowed to vary as a linear function of the covariate, and boundary separation is allowed to vary between the two manipulations.

```
v_covariate = [.2 .8 -.4 .6 .2 .8 -.4 .6]';
v_intercept = ones(8,1);
v_dm = [v_intercept v_covariate]
```

```
v_dm =
    1.0000    0.2000
    1.0000    0.8000
    1.0000   -0.4000
    1.0000    0.6000
    1.0000    0.2000
    1.0000    0.8000
    1.0000   -0.4000
    1.0000    0.6000
```

```
a_dm = [ones(4,1) zeros(4,1);zeros(4,1) ones(4,1)];
options(2).DesignMatrix = {a_dm,'1','1',[],'1','1',v_dm,'1','1'};
options(2).Name = 'Linear regression';
```

In Model 3, drift rate and boundary separation are allowed to vary freely across conditions (to check for deviations from the design).

```
options(3).DesignMatrix = {[],'1','1',[],'1','1',[],'1','1'};
options(3).Name = 'Deviation from linearity';
```

(5) Invoke DMAT

Use the multiestv4 function to get parameter estimates.

```
output = multiestv4(data,options);
```

APPENDIX C (Continued)

```
Warning: Automatically generated guess was outside parameter space. Gen-
erating new guess.

      Guess      :   21044.35257323     (19-Dec-2006 16:01:52)
      Simplex  1:    20241.99941073     (19-Dec-2006 16:02:09)
      Simplex  2:    20225.22102745     (19-Dec-2006 16:02:31)
      Simplex  3:    20225.20412897     (19-Dec-2006 16:02:54)
      Simplex  4:    20206.10358754     (19-Dec-2006 16:05:58)
      Final loss:    20206.10358754     (19-Dec-2006 16:06:21)

Warning: Hessian is not of full rank.
Warning: Hessian is not positive definite.

      Guess      :   20206.10358755     (19-Dec-2006 16:06:21)
      Simplex  1:    17656.05744417     (19-Dec-2006 16:06:46)
      Simplex  2:    17335.69652287     (19-Dec-2006 16:07:04)
      Simplex  3:    17331.95642866     (19-Dec-2006 16:07:21)
      Simplex  4:    17331.91079995     (19-Dec-2006 16:07:47)
      Final loss:    16118.72142640     (19-Dec-2006 16:08:24)

      Guess      :   16118.72142640     (19-Dec-2006 16:08:24)
      Simplex  1:    16103.99735918     (19-Dec-2006 16:08:39)
      Simplex  2:    16103.65078687     (19-Dec-2006 16:08:56)
      Simplex  3:    16103.41232557     (19-Dec-2006 16:09:12)
      Simplex  4:    16101.79752097     (19-Dec-2006 16:14:21)
      Final loss:    16101.73477408     (19-Dec-2006 16:16:26)
```

(6) Display Summary Output

```
qtable(output)
```

```
-----------------------------------------------------------------------
  Deviance  d-Deviance   df   d-df       AICc         BIC          p
-----------------------------------------------------------------------
 20206.1036        NaN    6    NaN  20218.1203  20255.2357       NaN
 16118.7214  4087.3822    8      2  16134.7526  16184.2309   0.00000
 16101.7348    16.9867   20     12  16141.9471  16265.5086   0.15010
-----------------------------------------------------------------------
```

Users should be warned that using the QPP statistic for model inference may not always be appropriate (see Vandekerckhove & Tuerlinckx, 2007b).

(Manuscript received December 21, 2006;
revision accepted for publication March 17, 2007.)