Bayesian Inference Using WBDev: A Tutorial for Social Scientists

Ruud Wetzels[1], Michael D. Lee[2], and Eric–Jan Wagenmakers[1]
[1] University of Amsterdam
[2]University of California, Irvine

Correspondence concerning this article should be addressed to:
Ruud Wetzels
University of Amsterdam, Department of Psychology
Roetersstraat 15
1018 WB Amsterdam, The Netherlands
Ph: (+31) 20–525–8871
Fax: (+31) 20–639–0279
E–mail may be sent to Wetzels.Ruud@gmail.com.

Abstract

Over the last decade, the popularity of Bayesian data analysis in the empirical sciences has greatly increased. This is partly due to the availability of WinBUGS—a free and flexible statistical software package that comes with an array of predefined functions and distributions—allowing users to build complex models with ease. For many applications in the psychological sciences, however, it is highly desirable to be able to define one's own distributions and functions. This functionality is available through the WinBUGS Development Interface (WBDev). This tutorial illustrates the use of WBDev by means of concrete examples, featuring the Expectancy–Valence model for risky behavior in decision–making, and the shifted Wald distribution of response times in speeded choice.

Keywords: WinBUGS, WBDev, BlackBox, Bayesian Modeling

Introduction

Psychologists who seek quantitative models for their data face formidable challenges. Not only are data often noisy and scarce, but they may also have a hierarchical structure, they may be partly missing, they may have been obtained under an ill–defined sampling plan, and they may be contaminated by a process that is not of interest. In addition,

the models under consideration may have multiple restrictions on the parameter space, especially when there is useful prior information about the subject matter at hand.

In order to address these kinds of real–world challenges, the psychological sciences have started to use Bayesian models for the analysis of their data (e.g., Lee, 2008; Rouder & Lu, 2005; Hoijtink, Klugkist, & Boelen, 2008). In Bayesian models, existing knowledge is quantified by *prior* probability distributions and updated upon consideration of new data to yield *posterior* probability distributions. Modern approaches to Bayesian inference include Markov chain Monte Carlo sampling (MCMC; e.g., Gamerman & Lopes, 2006; Gilks, Richardson, & Spiegelhalter, 1996) a procedure that makes it easy for researchers to construct probabilistic models that respect the complexities in the data, allowing almost any probabilistic model to be evaluated against data.

One of the most influential software packages for MCMC–based Bayesian inference is known as WinBUGS (BUGS stands for Bayesian inference Using Gibbs Sampling; Cowles, 2004; Sheu & O'Curry, 1998; Lunn, Thomas, Best, & Spiegelhalter, 2000; Lunn, Spiegelhalter, Thomas, & Best, in press). WinBUGS comes equipped with an array of predefined functions (e.g., `sqrt` for square root and `sin` for sine) and distributions (e.g., the Binomial and the Normal) that allow users to combine these elementary building blocks into complex probabilistic models almost at will.

For some psychological modeling applications, however, it is highly desirable to define one's own functions and distributions. In particular, user–defined functions and distributions greatly facilitate the use of psychological process models such as the Attention Learning Covering map (ALCOVE; Kruschke, 1992), the Generalized Context Model for category learning (GCM; Nosofsky, 1986), the Expectancy–Valence model for decision–making (Busemeyer & Stout, 2002), the SIMPLE model of memory (Brown, Neath, & Chater, 2007), or the Ratcliff diffusion model of response times (Ratcliff, 1978).

The ability to implement these user–defined functions and distributions can be achieved through the use of the WinBUGS Development Interface (WBDev; Lunn, 2003), an add–on program that allows the user to hand–code functions and distributions in the programming language Component Pascal.[1] To that end, we need BlackBox, a development environment for programs such as WinBUGS, that are written in Component Pascal.

The use of WBDev brings several advantages. For instance, complicated WBDev components lead to faster computation than their counterparts programmed in straight WinBUGS code. Moreover, some models will only work properly when implemented in WBDev. Another advantage of WBDev is that it compartmentalizes the code, resulting in scripts that are easier to understand, communicate, adjust, and debug. A final advantage of WBDev is that it allows the user to program functions and distributions that are simply not available in WinBUGS, but may be central components of psychological models (Donkin, Averell, Brown, & Heathcote, in press; Vandekerckhove, Tuerlinckx, & Lee, 2009).

This tutorial aims to stimulate psychologists to use WBDev by providing four thoroughly documented examples; for both functions and distributions, we provide a simple and a more complex example. All examples are relevant to psychological research.[2] Our tutorial

[1] More information can be found at: `http://en.wikipedia.org/wiki/Component_Pascal`.

[2] There already exist two concise tutorials on how to write functions and distributions in WBDev, written by David Lunn and Chris Jackson. The examples in those tutorials require advanced programming skills and they are not directly relevant for psychologists.

is geared towards researchers who have at least some experience in computer programming, and, ideally, some experience with the WinBUGS program. Nevertheless, we have tried to keep our tutorial accessible for social scientists in general. A more gentle introduction to the WinBUGS program is provided by Ntzoufras (2009) and Lee and Wagenmakers (2009).

We start our tutorial by discussing the WBDev implementation of a simple *function* that involves the addition of variables. We then turn to the implementation of a complicated function that involves the Expectancy–Valence model (Busemeyer & Stout, 2002; Wetzels, Vandekerckhove, Tuerlinckx, & Wagenmakers, in press). Next, we discuss the WBDev implementation of a simple *distribution*, first focusing on the Binomial distribution, and then turning to the implementation of a more complicated distribution that involves the shifted Wald distribution (Heathcote, 2004; Schwarz, 2001). For all of these examples, we explain the crucial parts of the WBDev scripts and the WinBUGS code. The thoroughly commented code is available online at `www.ruudwetzels.com`. For each example, our explanation of the WBDev code is followed by application to data and the graphical analysis of the output.

Installing WBDev (Blackbox)

Before we can begin hard–coding our own functions and distributions we need to download and install three programs; WinBUGS, WBDev and BlackBox.[3] To make sure all programs function properly, they have to be installed in the order given below.

1. Install WinBUGS

WinBUGS is the core program that we will use. Download the latest version from `http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml` (WinBUGS14.exe). Install the program in the default directory `./Program Files/WinBUGS14`.[4] Make sure to register the software by obtaining the registration key and following the instructions—WinBUGS will not work without it.

2. Install WinBUGS Development Interface (WBDev)

Download WBDev from `http://www.winbugsdevelopment.org.uk/` (WBDev.exe). Unzip the executable in your WinBUGS directory `./Program Files/WinBUGS14`. Then start WinBUGS, open the"wbdev01_09_04.txt" file and follow the instructions at the top of the file. During the process, WBDev will create its own directory `/WinBUGS14/WBDev`.

3. Install BlackBox Component Builder

Blackbox can be downloaded from `http://www.oberon.ch/blackbox.html`. At the time of writing, the latest version is 1.5. Install Blackbox in the default directory: `./Program Files/BlackBox Component Builder 1.5`. Go to the WinBUGS directory and select all files (press "Ctrl–A") and copy them (press "Ctrl+C"). Next, open the BlackBox directory and paste the copied files in there (press "Ctrl+V"). Select " `Yes to all`" if asked about replacing files. Once this is done, you will be able to open BlackBox

[3] At the time of writing, all programs are available without charge.

[4] On the Windows Vista operating system is Windows Vista, install the program in the directory "c:/WinBUGS14".

and run WinBUGS from inside Blackbox. This completes installation of the software, and we can start to write our own functions and distributions.

Functions

The mathematical concept of a function expresses a dependence between variables. The basic idea is that some variables are given (the input) and with them, other variables are calculated (the output). Sometimes, complex models require many arithmetic operations to be performed on the data. Because such calculations can become computationally demanding using straight WinBUGS code, it can be convenient to use WBDev and implement these calculations as a function. The first part of this section will explain a problem without using WBDev. We then show how to use WBDev to program a simple and a more complex function.

Example 1: A Rate Problem

A binary process has two possible outcomes. It might be that something either happens or does not happen, or that something either succeeds or fails, or takes one value rather than the other. An inference that often is important for these sorts of processes concerns the underlying rate at which the process takes one value rather than the other. Inferences about the rate can be made by observing how many times the process takes each value over a number of trials.

Suppose that someone plays a simple card game and can either win or lose. We are interested in the probability that the player wins a game. To study this problem, we formalize it by assuming the player plays n games and wins k of them. These are known, or observed, data. The unknown variable of interest is the probability θ that the player wins any one specific game. Assuming the games are statistically independent (i.e., that what happened on one game does not influence the others, so that the probability of winning is the same for all of the games), the number of wins k follows a Binomial distribution, which is written as

$$k \sim \text{Binomial}\big(\theta, n\big) \tag{1}$$

and can be read "the success count k out of a total of n trials is Binomially distributed with success rate θ". In this example, we will assume a success count of 9 ($k = 9$) and a trial total of 10 ($n = 10$).

A rate problem: the model file. A so–called model file is used to implement the model into WinBUGS. The model file for inferring θ from an observed n and k looks like this:

```
model
{
      # prior on the rate parameter theta
      theta ~ dunif(0,1)

      # observed wins k out of total games n
      k ~ dbin(theta,n)
}
```

The twiddles symbol ($\sim$) means "is distributed as". Because we use a Uniform distribution between 0 and 1 as a prior on the rate parameter θ, we write `theta ~ dunif(0,1)`. This indicates that, a priori, each value of θ is equally likely. Furthermore, k is Binomially distributed with parameters θ and n (i.e., `k ~ dbin(theta,n)`). Note that `dunif` and `dbin` are two of the predefined distributions in WinBUGS. All the distributions that are predefined in WinBUGS are listed in the distributions section in the WinBUGS manual, which can be accessed by clicking the help menu and selecting User manual (or by pressing F1). The hash symbol (#) is used for comments. The lines starting with this symbol are not executed by WinBUGS.

Copy the text into an empty file and save it as "model_rateproblemfunction.txt" in the directory from where you want to work. There are now various ways in which to proceed. One way is to work from within WinBUGS; another way is to control WinBUGS by calling it from a more general purpose program. Here, we use the statistical programming language R[5] to call WinBUGS, but widely–used alternative research programming environments such as MATLAB are also available (Lee & Wagenmakers, 2009).

A rate problem: the R script. The next step is to construct an R–script to call Blackbox from R.[6] When you run the script "Rscript_RateProblemFunction.R", WinBUGS starts, the MCMC sampling is conducted, WinBUGS closes, and you return to R. The object that WinBUGS has returned to R is called "rateproblem", and this object contains all the information about the Bayesian inference for θ.

In particular, the "rateproblem" object contains a single sequence of consecutive draws from the posterior distribution of θ, a sequence that is generally known as an MCMC *chain*. Inspection of the MCMC chain is an important part of the analysis because valid inference requires that the chain has converged, meaning that the samples have been drawn from the posterior distribution.

Convergence problems can occur in a variety of situations. One source of non–convergence can be in the data. For example, there may be too little data available, or the data may not be in agreement with the model that was used. However, lack of convergence can also originate from the model itself. For example, the model may have too many parameters, or it may have parameters that are highly correlated. Another situation in which the MCMC chain might not converge is when the priors are unrealistic.

Because lack of convergence can lead to invalid results, it is important to check whether the MCMC chains have converged. One way to check this is to run multiple chains with overdispersed starting values. Despite the differing starting values, the chains should be indistinguishable from each other after enough samples have been drawn. Moreover, these chains should all have the same, constant mean. Another visual test is that when an MCMC chain has converged, each chain should look like a "fat hairy caterpillar" (see for instance Figure 4). This should happen when the subsequent draws are (relatively) independent and when a chain consists of many samples.

Apart from a visual inspection of the chains, more formal tests for convergence exist as well. One often–used method is the Gelman and Rubin (1992) $\hat{R}$ statistic that compares the between–chain variance to the within–chain variance. When the chain has converged,

[5] R is, at the time of writing, freely available from: `http://www.r-project.org/`.

[6] All the scripts can be found on the website of the first author: `http://www.ruudwetzels.com`.

$\hat{R}$ should be very close to 1. As a rule of thumb, $\hat{R}$ should be smaller than 1.1. For a more detailed discussion of convergence statistics see Cowles and Carlin (1996), Gilks et al. (1996) and Gelman and Hill (2007).[7]

Convergence problems may be remedied in various ways. One method is to try different parameterizations of the model that speed up convergence. Another method is to use transformations that bring the data and the model in correspondence. However, convergence problems can often be eliminated by two "brute force" methods. One such method is to eliminate the first m iterations of each chain (i.e., the "burn–in" period). Usually, taking $m = 1000$ does the trick. In the R–scripts that we use in this paper, we always use a burn–in of 1000 iterations. Another method is to draw more samples and use *thinning*. This means that instead of using the entire chain, we pick out for example every tenth sample (i.e., thinning=10). This reduces the dependency between subsequent MCMC draws.

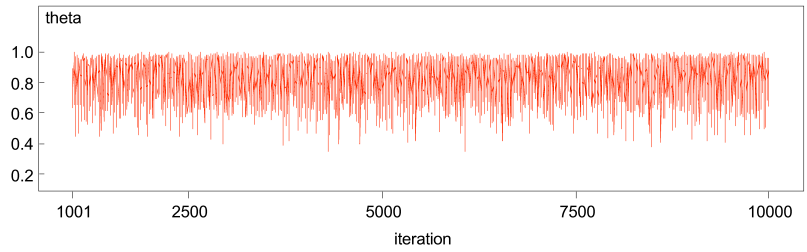

Figure 1. The MCMC chain of 9000 draws from the posterior distribution of the rate parameter θ.

For our simple Binomial example, convergence is immediate. After you run the code, WinBUGS should show an MCMC chain similar to the one shown in Figure 1.

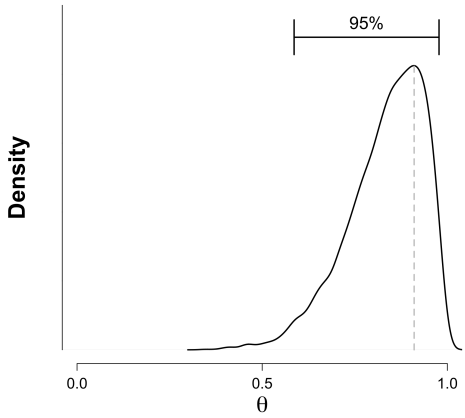

Figure 2. The posterior distribution of the rate parameter θ after observing 9 wins out of 10 games. The dashed gray line indicates the mode of the posterior distribution at $\theta = .90$. The 95% credible interval extends from .59 to .98.

We use the samples from Figure 1 to estimate the posterior distribution of θ. To

[7] Useful packages for the inspection of MCMC chains are the CODA package, `http://cran.r-project.org/web/packages/coda/` and the BOA package, `http://cran.r-project.org/web/packages/boa/`.

arrive at the posterior distribution, the samples are not plotted as a time series but as a distribution. In order to estimate the posterior distribution of θ, we applied the standard density estimator in R. Figure 2 shows that the mode of the distribution is very close to .90, just as we expected. The posterior distribution is relatively spread out over the parameter space, and the 95% credible interval extends from .59 to .98, indicating the uncertainty about the value of θ. Had we observed 900 wins out of a total of 1000 games the posterior of θ would be much more concentrated around the mode of .90, as our knowledge about the true value of θ would have greatly increased.

Example 2: ObservedPlus

In this section we examine the rate problem again, but now we change the variables. Suppose you learn that before you observed the current data, 10 games had already been played, resulting in a single win. To add this information, we design a function that adds 1 to the number of observed wins, and 10 to the number of total games. So, when we use $k = 9$ and $n = 10$ as before, we end up with

$$k_{new} = k_{old} + 1 = 9 + 1 = 10 \tag{2}$$

and

$$n_{new} = n_{old} + 10 = 10 + 10 = 20. \tag{3}$$

Hence, when we use our new function, the mode of the posterior distribution should no longer be .90 but .50 (10/20 = .50). To obtain these results, we are going to build a function called "ObservedPlus", using the template "VectorTemplate.odc". This template is located in the folder "...\$BlackBoxComponentBuilder1.5\WBdev\Mod$".

ObservedPlus: the WBDev script. The script file "ObservedPlus.odc" shows text in three colors. The parts that are colored black should not be changed. The parts in red are comments and these are not executed by Blackbox. The parts in blue are the most relevant parts of the code, because these are the parts that can be changed to create the desired function. The templates for writing the functions and distributions in this tutorial come together with the WBDev software and were written by David Lunn and Chris Jackson.

We now give a detailed explanation of the ObservedPlus WBDev function. The numbers **(*X*)** correspond to the numbers in the ObservedPlus WBDev script. For this simple example, we show some crucial parts of the WBDev scripts below.

(*1*) `MODULE WBDevObservedPlus;`

The name of the module is typed here. We have named our module ObservedPlus.

The name of the module (so the part after MODULE WBDev...) has to start with a capital letter.

(*2*) `args := "ss";`

Here you must define specific arguments about the input of the function. You can choose between scalars (s) and vectors (v). A scalar means that the input is a single number. When you want to use a variable that consists of more numbers (for example

a time series) you need a vector. This line has to correspond with the constants defined at **(*3*)**. In our example, we use two scalars, the number of successes k and the total number of observations n.

```
(*3*) in = 0; ik = 1;
```

Because of what has been defined at **(*2*)**, WBDev already knows that there should be two variables here. We name them in and ik, with in at the first spot (with number 0) and ik at the second spot (with number 1). WBDev always starts counting at 0 and not at 1.

Note that we did not name our variables n and k, but in and ik. This is because it is more insightful to use n and k later on, and it is not possible to give two or more variables the same name. Finally, note that the positions of the constants correspond to the positions of the input of the variables into the function in the model file. We will return to this issue later.

```
(*4*) n, k:  INTEGER;
```

The variables that are used in the calculations need to be defined. Both variables are defined as integers, because the Binomial distribution only allows integers as input: counts of successes and the total games that are played can only be positive integers.

```
(*5*) n := SHORT(ENTIER(func.arguments[in][0].Value()));
      k := SHORT(ENTIER(func.arguments[ik][0].Value()));
```

This code takes the input values (in and ik) and gives them a name. We defined two variables in **(*4*)**, and we are now going to use them. What the script says here is: take the input values in and ik and store them in the integer variables n and k. Because the input variables are not automatically assumed to be integers, we have to transform them and make sure the program recognizes them as integers. So, in other words, the first line says that n is the same as the first input variable of the function (see Figure 3), and the second line says that k is the same as the second input variable of the function.

```
(*6*) n:=n+10;
      k:=k+1;
      values[0] := n;
      values[1] := k;
```

This is the part of the script where we do the actual calculations. At the end of this part, we fill the output array *values* with the new n and k.

```
(*7*) END WBDevObservedPlus.
```

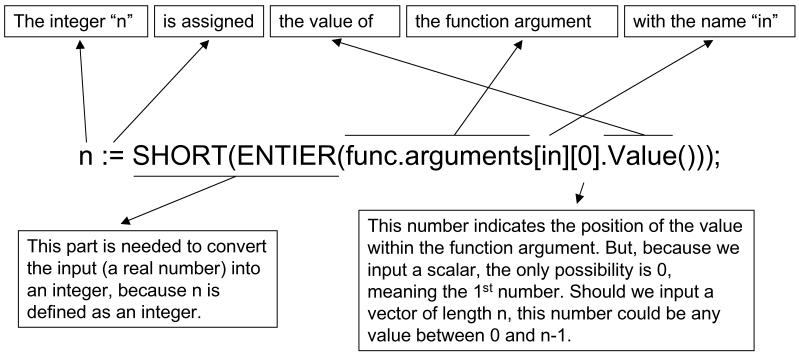

Figure 3. A detailed explanation of part **(*5*)** of “ObservedPlus.odc”.

Finally, we need to make sure that the name of the module at the end is the same as the name at the top of the file. The last line has to end with a period. Hence, the last line of the script is ”`ENDWBDevObservedPlus.`”.

Now you need to compile the function by pressing “Ctrl–k”. Syntax errors cause WBDev to return an error message. Name this file “ObservedPlus.odc” and save it in the directory “...\$BlackBoxComponentBuilder1.5\WBdev\Mod”.

We are not entirely ready to use the function yet. WBDev needs to know that there exists a function called ObservedPlus; WBDev also needs to know what the input looks like (i.e., how many inputs are there, what order are they presented, and are they scalars and vectors?), and what the output is. To accomplish this, open the file “functions.odc” in the directory “...\$BlackBoxComponentBuilder1.5\WBdev\Rsrc”. Add the line: `v<-"ObservedPlus"(s,s) "WBDevObservedPlus.Install"` and then save the file. The next time that WBDev is started, it knows that there is a function named ObservedPlus which has two scalars as input, and a vector as output. The function is now ready to be used in a model file.

ObservedPlus: the model file. In order to use the newly scripted function ObservedPlus we use a model file that is similar to the model file used in the earlier rate problem example.

```
model
{
      # Uniform prior on the rate parameter
      theta ~ dunif(0,1)

      # use the function to get the new n and the new k
      data[1:2] <- ObservedPlus(n,k)

      # define the new n and new k as variables
      newn <- data[1]
      newk <- data[2]
```

```
      # the new observed data
      newk ~ dbin(theta,newn)
}
```

We assume a Uniform prior on θ (i.e., `theta ~ dunif(0,1)`). The function ObservedPlus takes as input the total number of games n and the number of wins k. From them, the new n and new k can be calculated (i.e., `data[1:2] <- ObservedPlus(n,k)`). Note that functions require the use of the assignment operator (`<-`) instead of the twiddles symbol ($\sim$). Remember that in the WBDev function the location of in was 0 and the location of ik was 1. Because that order was used, the input has to have n first and then k.

Next, `newn` is the first number in the vector `data` and `newk` is the second (i.e., `newn <- data[1]`, `newk <- data[2]`). Remember that when scripting in WBDev, the first element has index 0, but in the model file the first element has index 1. Finally, we use our new variables to do inference on the rate parameter θ (i.e., `newk ~ dbin(theta,newn)`).

Copy the text from the model file into an empty text file and name this file "model_observedplus.txt". Copy this file to the location of the model file that was used in the rate problem example.

ObservedPlus: the R script. To run this model from R, we can use the script of the original rate problem. The only thing that needs to be changed is the name of the model file. This should now be "model_observedplus.txt". Change this name and run the R–script.

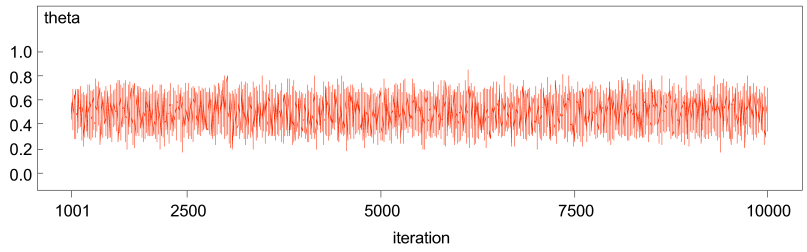

Figure 4. The MCMC chain of 9000 draws from the posterior distribution of θ, after using the function ObservedPlus.

After you run the code, WinBUGS should show an MCMC chain similar to the one shown in Figure 4. The chain looks like a fat hairy caterpillar, so for now we assume it has converged to the posterior distribution of θ.

Figure 5 shows the posterior distribution of θ. The mode of the distribution is .50, because $k_{new} = 10$ and $n_{new} = 20$. Again, because the total number of games played is fairly small, the posterior distribution of θ is relatively spread out (the 95% credible interval ranges from .30 to .70), reflecting our uncertainty about the true value of θ.

Example 3: The Expectancy–Valence Model

In the example described above, we could have used plain WinBUGS code instead of writing a script in Blackbox. But sometimes it can be very useful to write a Blackbox script

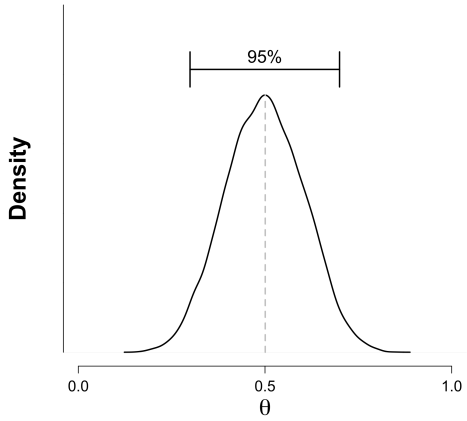

Figure 5. The posterior distribution of the rate parameter θ, after using the function ObservedPlus. The dashed gray line indicates the mode of the posterior distribution at $\theta = .50$. The 95% credible interval extends from .30 to .70.

instead of plain WinBUGS code, especially if the model under consideration is relatively complex. Implementing such a model into WBDev can speed up the computation time for inference substantially. The reason for this speed–up is that WBDev scripts are pre–compiled while the WinBUGS model files are interpreted at runtime. The present example, featuring the Exptectancy–Valence model to understand risk–seeking behavior in decision making, provides a concrete demonstration of this general point.

Suppose a psychologist wants to study decision making of clinical populations under controlled conditions. A task that is often used for this purpose is the "Iowa gambling task", developed by Bechara and Damasio (IGT; Bechara, Damasio, Damasio, & Anderson, 1994; Bechara, Damasio, Tranel, & Damasio, 1997).

In the IGT, participants have to discover, through trial and error, the difference between risky and safe decisions. In the computerized version of the IGT, the participant starts with $2000 in play money. The computer screen shows players four decks of cards (A, B, C, and D), and then they have to select a card from one of the decks. Each card is associated with either a reward or a loss. The default payoff scheme is presented in Table 1.

	Bad Decks		Good Decks	
	A	B	C	D
reward per trial	100	100	50	50
number of losses per 10 cards	5	1	5	1
loss per 10 cards	1250	1250	250	250
net profit per 10 cards	−250	−250	250	250

Table 1: Rewards and Losses in the IGT. Cards from decks A and B yield higher rewards than cards from decks C and D, but they also yield higher losses. The net profit is highest for cards from decks C and D.

At the start of the IGT, participants are told that they should maximize net profit.

During the task, they are presented with a running tally of the net profit, and the task finishes after 250 card selections.

The Expectancy–Valence (EV) model proposes that choice behavior in the IGT comes about through the interaction of three latent psychological processes. Each of these processes is vital for successful performance, typified by a gradual increase in preference for the good decks over the bad decks. First, the model assumes that the participant, after selecting a card from deck k, $k \in \{1, 2, 3, 4\}$ on trial t, calculates the resulting net profit or valence. This valence v_k is a combination of the experienced reward $W(t)$ and the experienced loss $L(t)$:

$$v_k(t) = (1 - w)W(t) + wL(t). \tag{4}$$

Thus, the first parameter of the Expectancy Valence model is w, the *attention weight* for losses relative to rewards, $w \in [0, 1]$.

On the basis of the sequence of valences v_k experienced in the past, the participant forms an expectation Ev_k of the valence for deck k. In order to learn, new valences need to update the expected valence Ev_k. If the experienced valence v_k is higher or lower than expected, Ev_k needs to be adjusted upward or downward, respectively. This intuition is captured by the equation

$$Ev_k(t + 1) = Ev_k(t) + a(v_k(t) - Ev_k(t)), \tag{5}$$

in which the *updating rate* $a \in [0, 1]$ determines the impact of recently experienced valences.

The EV model also uses a reinforcement learning method called softmax selection or Boltzmann exploration (Kaelbling, Littman, & Moore, 1996; Luce, 1959) to account for the fact that participants initially explore the decks, and only after a certain number of trials decide to always prefer the deck with the highest expected valence.

$$\Pr[S_k(t + 1)] = \frac{\exp\left(\theta(t)Ev_k\right)}{\sum_{j=1}^{4} \exp\left(\theta(t)Ev_j\right)}. \tag{6}$$

In this equation, $1/\theta(t)$ is the "temperature" at trial t and $Pr(S_k)$ is the probability of selecting a card from deck k. In the EV model, the temperature is assumed to vary with the number of observations according to

$$\theta(t) = (t/10)^c, \tag{7}$$

where c is the *response consistency* or sensitivity parameter. In fits to data, this parameter is usually constrained to the interval $[-5, 5]$. When c is positive, response consistency θ increases (i.e., the temperature $1/\theta$ decreases) with the number of observations. This means that choices will be more and more guided by the expected valences. When c is negative, choices will become more and more random as the number of card selections increases.

In sum, the EV model decomposes choice behavior in the Iowa gambling task to three components or parameters:

1. An attention weight parameter w that quantifies the weighting of losses versus rewards.

2. An updating rate parameter a that quantifies the memory for rewards and losses.

3. A response consistency parameter c that quantifies the level of exploration.

The EV model: the WBDev script. To implement the EV model as a function in WBDev it is useful to first describe what data is observed and passed on to WinBUGS. In this example, we examine the data of one participant who has completed a 250–trial IGT. Hence, the observed data are an index of which deck was chosen at each trial, and the sequence of wins and losses that the participant incurred. Please see the script–file called "EV.odc".

(*1*) We name our module EV.

(*2*) In the EV example, we use 3 scalars for the 3 parameters and 3 vectors for the wins, losses and index at each trial.

(*3*) We start with the data vectors (the order is arbitrary, but needs to correspond to the one used in the model file) and we name these constants *iwins*, *ilosses* and *iindex*. After that, the function has as input the parameters of the EV–model, *iw*, *ia* and *ic*.

(*4*) In this section we define all the variables that we need to use in our calculations. Several mathematical functions are already available in WBDev. Information about these functions can be found by right–clicking the word "Math" in the script and then by clicking "documentation".

(*5*) Here we take our input EV parameters and assign them to the variables that we defined in part **(*4*)**.

(*6*) This is the part of the script where we do the actual calculations. At the end of this part, we fill the output variable called "values", with the output of our EV–function, the probability of choice for a deck.

(*7*) Make sure that the name of the module at the end is the same as the name at the top of the file. The last line has to end with a period.

(*11*) The DrawSample(.) procedure returns a pseudo–random number from the new distribution.

Now you need to compile the function by pressing "Ctrl–k". Name this file "EV.odc" and save it in the directory "...\$BlackBoxComponentBuilder1.5\WBdev\Mod$".

We need to add this function to the function file (like in the ObservedPlus example) so that WinBUGS knows that the EV function exists, the next time it is started. Open the file"functions.odc" in the directory "*...\ BlackBox Component Builder 1.5\ WBdev\ Rsrc*". Add the line: `v <- "EV"(v,v,v,s,s,s) "WBDevEV.Install"` and then save the file. The next time that WBDev is started, it knows that there is a function named EV which has three vectors and three scalars as input, and a vector as output. The function is now ready to be used in a model file.

The EV–model: the model file. In order to use the EV–model we need to implement the graphical model in WinBUGS. The following model file is used in this example:

```
model
{
      # EV parameters are assigned prior distributions
      w ~ dunif(0,1)
      a ~ dunif(0,1)
      c ~ dunif(-5,5)

      # data from the EV function
      evprobs[1:1000] <- EV(wi[],lo[],ind[],w,a,c)

      # only use the information from the chosen deck
      # see explanation below
      for (i in 1:250)
      {
            p.EV[i,1] <- evprobs[deckA[i]]
            p.EV[i,2] <- evprobs[deckB[i]]
            p.EV[i,3] <- evprobs[deckC[i]]
            p.EV[i,4] <- evprobs[deckD[i]]
            ind[i] ~ dcat(p.EV[i,])
      }
}
```

The parameters of the model, w, a, c are assigned Uniform prior distributions. w and a are bounded between 0 and 1 and c is bounded between -5 and 5 (i.e., `w` $\sim$ `dunif(0,1), a` $\sim$ `dunif(0,1), c` $\sim$ `dunif(-5,5)`). The wins and the losses from the 250–trials are stored in the vectors *wi* and *lo*. The indices from the decks that were chosen are stored in the vector *ind*. Together with the EV parameters they are input for the EV function that calculates the probability per choice (i.e., `evprobs[1:1000] <- EV(wi[],lo[],ind[],w,a,c)`).

Note that this function calculates 1000 probabilities for a 250–trial dataset. This is because the probability for each deck is calculated, not only for the chosen deck but for all decks. So at each trial, four probabilities are calculated and for 250 trials this totals 1000 probabilities. However, we are only interested in the probability of the chosen deck.

To handle this problem, we make four vectors, `deckA`, `deckB`, `deckC` and `deckD` which are rows of length 250. Each vector contains a sequence of numbers where the number at position t is calculated by adding four to the number at position $t-1$ ($x_t = x_{t-1} + 4$). The vector `deckA` starts with number 1, `deckB` starts with number 2, `deckC` starts with number 3 and `deckD` starts with number 4. Using these vectors, we can disentangle the probabilities for each deck at each trial, `evprobs[deckA[i]]` corresponds to the probabilities of choosing deck 1 at each trial i, `evprobs[deckB[i]]` to the probabilities of choosing deck 2 at each trial i, `evprobs[deckC[i]]` to the probabilities of choosing deck 3 at each trial i and `evprobs[deckD[i]]` to the probabilities of choosing deck 4 at each trial.

Finally, we state that the choice for a deck at trial i (the observed data vector `ind`) is Categorically distributed (i.e., `ind[i]` $\sim$ `dcat(p.EV[i,])`). The Categorical distribution (also known as the multinomial distribution) is the probability distribution for the choice of a card deck. This distribution is a generalization of the Bernoulli distribution for a categorical random variable (i.e., the choice for one of the four decks at each trial of the

IGT). Copy the text from the model into an empty file and save it as "model_ev.txt" in the directory from where you want to work.

The EV model: the R script.

To run this model and to supply WinBUGS with the data, we use the R–script called "Rscript_ExpectancyValence.r". Change the working directory (in the R–script) to the directory where the model file is located on your computer. This script contains fictitious data from a person who completed a 250–trial IGT. After you run the code, WinBUGS should show three MCMC chains similar to the ones shown in Figure 6.

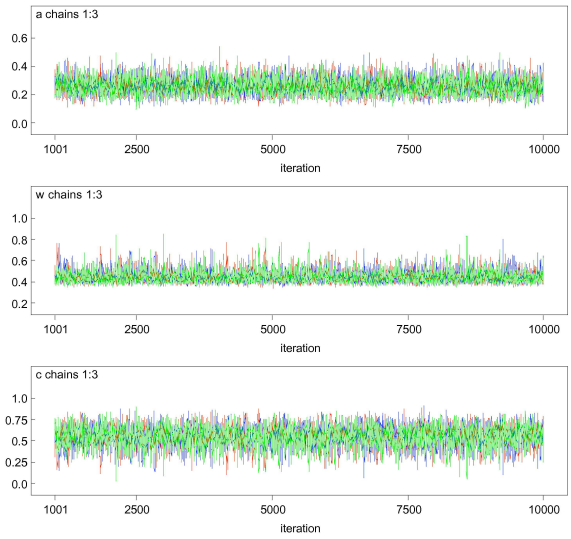

Figure 6. Three MCMC chains of 9000 draws each for the three EV parameters, the attention weight parameter w, the updating rate parameter a and response consistency parameter c.

For all the EV–parameters, the chains look like fat hairy caterpillars and hence appear to have converged to the posterior distribution. Because the EV–parameters are slightly harder to estimate than the rate parameters from the earlier examples —due to the complexity of the EV–model— we have to make sure that we are sampling from the correct posterior distribution of w, a and c.

Besides visual inspection of the MCMC chain, we now compute the often used measure of convergence, the $\hat{R}$ statistic (Gelman, Carlin, Stern, & Rubin, 2004, pp. 295–297). Specifically, we ran three chains, with different starting values, and then calculated $\hat{R}$. In this example, we observed that for each EV–parameter, the chains have converged properly ($\hat{R} \approx 1$). Note that it is important that the chains for *all* parameters have converged.

After having assured ourselves that the chains have converged we can plot the resulting posterior distributions. Figure 7 shows that the posterior mode of the attention weight parameter w is .43, the posterior mode of the update parameter a is .25 and the posterior mode of the consistency parameter c is 0.58.

On an average computer, it takes about 85 seconds to generate these chains. Had we used plain WinBUGS instead of WBDev code to compute these chains, the calculation time would have taken approximately 15 minutes. Hence, implementing the function into WBDev

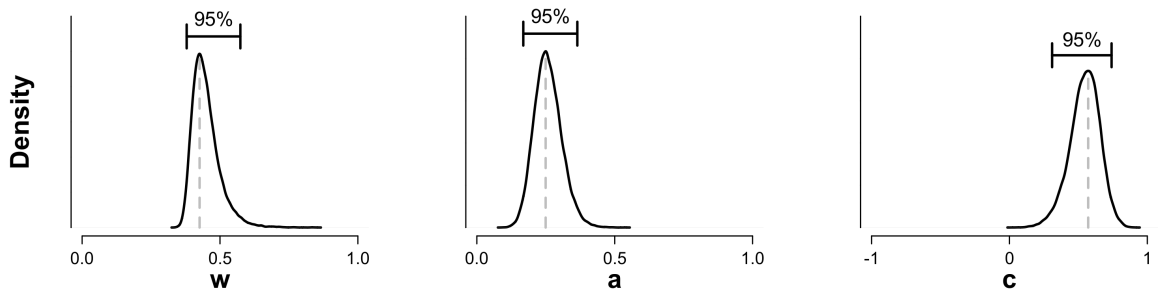

Figure 7. The posterior distributions of the three EV parameters, w, a and c. The dashed gray lines indicate the modes of the posterior distributions at $w = .43$, $a = .25$ and $c = 0.58$. The 95% credible intervals for w, a and c extend from .38 to .57, from .17 to .36 and from 0.31 to 0.74, respectively.

speeds up the analysis by a factor 10. However, this speed–up is not the only advantage of implementing functions into WBDev. Sometimes, complex models will only work properly when implemented in WBDev. Another advantage of WBDev is that it compartmentalizes the code, resulting in scripts that are easier to understand, communicate, adjust, and debug.

Distributions

Statistical distributions are invaluable in psychological research. For example, in the simple rate problem discussed earlier, we use the Binomial distribution to model our data. WinBUGS comes equipped with an array of predefined distributions, but it does not include all distributions that are potentially useful for psychological modeling. Using WBDev, researchers can augment WinBUGS to include these desired distributions.

In the next section we will explain how to write a new distribution, starting with the Binomial distribution as a simple introduction, and then considering the more complicated shifted Wald distribution.

Example 4: Binomial distribution

Obviously, the Binomial distribution is already hard–coded in WinBUGS. But, because it is a very well–known and relatively simple distribution, it serves as a useful first example.

To program a distribution in WBDev, we can use the distribution template that is already in the Blackbox directory. This file is located in the folder: "...\$BlackBoxComponentBuilder1.5\WBdev\Mod$". In order to program the distribution, we first need to write out the log likelihood function:

$$\begin{aligned}
\log\left(Pr(K=k)\right) &= \log\left(f(k;n,\theta)\right) \\
&= \log\left(\binom{n}{k}\theta^k(1-\theta)^{n-k}\right) \\
&= \log\left(\binom{n}{k} + \log(\theta^k) + \log(1-\theta)^{n-k}\right) \\
&= \log(n!) - \log(k!) - \log(n-k)! + k\log(\theta) + (n-k)\log(1-\theta).
\end{aligned} \tag{8}$$

Binomial distribution: the WBDev script. Here we describe the WDDev script for the Binomial distribution (See the file "BinomialTest.odc")

(*1*) We name our module BinomialTest.

(*2*) The parameters of the input of the Binomial distribution, *theta* and n.

(*3*) Here global variables can be declared. With global is meant that it is loaded only once, while the value of the variable may be needed many times. This part of the template does not need to be changed for this example.

(*4*) We have to declare what type of arguments are the input of the distribution. In this case these are two scalars (i.e. two single numbers), *theta* and n.

(*5*) This describes whether the distribution is discrete or continuous. When the distribution is discrete, isDiscrete should be set to TRUE. When the distribution is continuous, it should be set to FALSE. For the Binomial distribution isDiscrete is set to TRUE.

The other thing that is defined in this part of the script is if the cumulative distribution is to be provided. If so, canIntegrate should be set to TRUE. If this is set to true, an algorithm should be provided at **(*11*)**. We set canIntegrate to FALSE because we did not implement the cumulative distribution.

(*6*) This part of the code should define the natural bounds of the distribution. In our case, we take 0 as a lower bound and n as an upper bound, because k can never be larger than n.

(*7*) As the name implies, this part is the part where the full log likelihood of the distribution is defined. This is an implementation of the log likelihood as defined in Equation 8.

(*8*) Sometimes WinBUGS can ignore the normalizing constants. When that is the case, WinBUGS calls LogPropLikelihood(.). In our example, we refer back to the full log likelihood function.

(*9*) Occasionally, WinBUGS can make use of the LogPrior(.) procedure, which is proportional to the real log–prior function. In other words, this procedure omits the additive constants on the log scale. In our example, we just refer back to the full log likelihood function.

(*10*) This is the part where the cumulative distribution is defined when in part **(*7*)** canIntegrate is set to TRUE. Because we set this to FALSE, we do not define anything in this section.

(*11*) The DrawSample(.) procedure returns a pseudo–random number from the new distribution.

(*12*) The last thing that needs to be done is to make sure that the name of the module at the end is the same as the name at the top of the file. The last line has to end with a period.

Now you need to compile the function by pressing "Ctrl–k". Save this file as "BinomialTest.odc" and copy this file into the appropriate blackbox directory, "...\$BlackBoxComponentBuilder1.5\WBdev\Mod".

Open the distribution file "distributions.odc" in the directory "...\ *BlackBox Component Builder 1.5*\ *WBdev*\ *Rsrc*". Add the line `s ~ "BinomialTest"(s,s) "WBDevBinomialTest.Install"` and then save it. The next time you start Blackbox, the program will know that there exists a distribution called BinomialTest, and that the inputs are two scalars (single numbers).

Binomial distribution: the model file. To use the scripted Binomial distribution, we write a model file that is very similar to the model file used in the rate problem example. We only need to change the name of the distribution from dbin to BinomialTest.

```
model
{
      # prior on rate parameter theta
      theta~dunif(0,1)

      # observed wins k out of total games n
      k~BinomialTest(theta,n)

      # compute the posterior predictive of k
      postpred.k~BinomialTest(theta,n)
}
```

This example is essentially the same statistical problem as the first example, the rate problem. Ten games are played (i.e., $n = 10$) and nine games are won (i.e., $k = 9$). We assume a Uniform prior on θ (i.e., `theta ~ dunif(0,1)`). The observed wins k are distributed as our newly made BinomialTest with rate parameter *theta* and total games n (i.e., `k~BinomialTest(theta,n)`). With *theta* and k defined, this completes the model for BinomialTest. The Drawsample feature of the function ([**(*11*)**]), produces the posterior predictive values for k (i.e., `postpred.k~BinomialTest(theta,n)`). Save this file as "model_rateproblemdistribution.txt" and copy it to your working directory.

Binomial distribution: the R script. The last thing that we need to do is to start R and open the appropriate R–script "Rscript_RateProblemDistribution.R". Change the working directory (in the R–script) to the directory where the model file is located on your computer. After you run the code, the results should be similar to those shown in Figure 1 and Figure 2.

After having observed that data, the prediction of future data can be of interest. The so–called *posterior predictive distribution* gives the relative probability of different outcomes after the data have been observed. First, a sample is drawn from the joint posterior distribution. Next, data is generated using the posterior sample.

In this example these different outcomes can be $k = 1, 2, ..., 10$. The posterior predictive is often used for checking the assumptions of a model. If a model describes the data

well, then the posterior predictive generated under the model should resemble the observed data. Large differences between the observed data and the posterior predictive imply that the model is not suitable for the data at hand. Figure 8 shows the posterior predictive of k. The median of the posterior predictive is $k = 9$, which corresponds to the observed data.

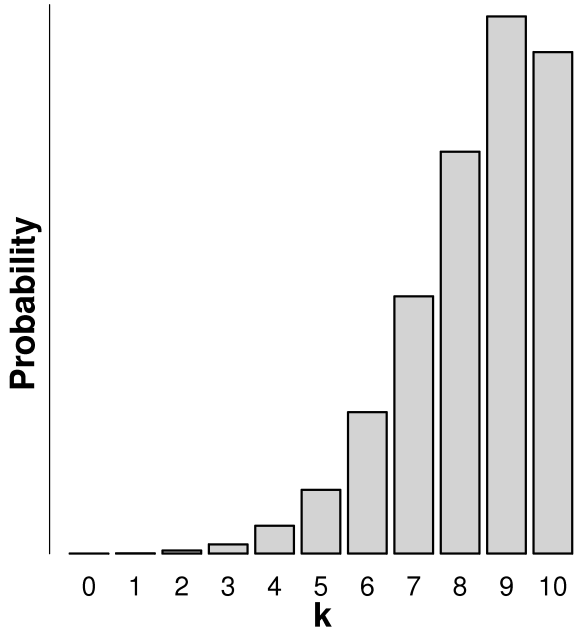

Figure 8. The posterior predictive of k, the number of wins out of 10 games. The median of the posterior predictive is $k = 9$.

Example 5: The shifted Wald distribution

Many psychological models use response times (RTs) to infer latent psychological properties and processes (Luce, 1986). One common distribution used to model RTs is the inverse Gaussian or Wald distribution (Wald, 1947). This distribution represents the density of the first passage times of a Wiener diffusion process toward a single absorbing boundary, as shown in Figure 9, using three parameters.

The parameter v reflects the drift rate of the diffusion process. The parameter a reflects the separation between the starting point of the diffusion process and the absorbing barrier. The third parameter, T_{er}, is a positive–valued parameter that shifts the entire distribution. The probability density function for this shifted Wald distribution is given by:

$$f(t|v, a, T_{er}) = \frac{a}{\sqrt{2\pi(t - T_{er})^3}} \exp\left\{ - \frac{[a - v(t - T_{er})]^2}{2(t - T_{er})} \right\}, \tag{9}$$

which is unimodel and positively skewed. Because of these qualitative properties, it is a good candidate for fitting empirical RT distributions. As an illustration, Figure 10 shows changes in the shape of the shifted Wald distribution as a result of changes in the shifted Wald parameters v, a, and T_{er}.

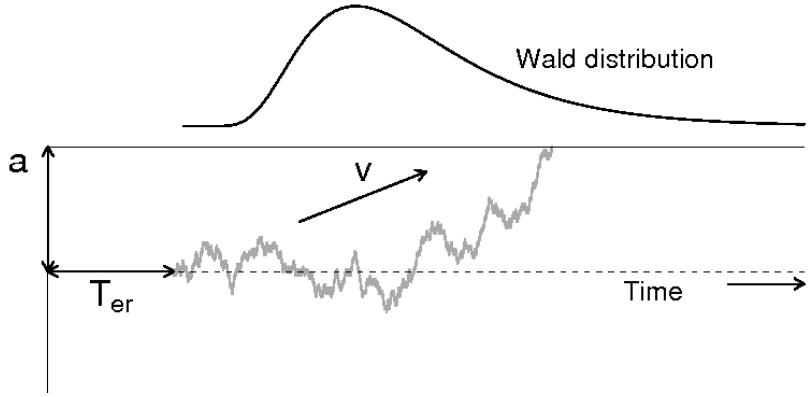

Figure 9. A diffusion process with one boundary. The shifted Wald parameter a reflects the separation between the starting point of the diffusion process and the absorbing barrier, v reflects the drift rate of the diffusion process and T_{er} is a positive–valued parameter that shifts the entire distribution.

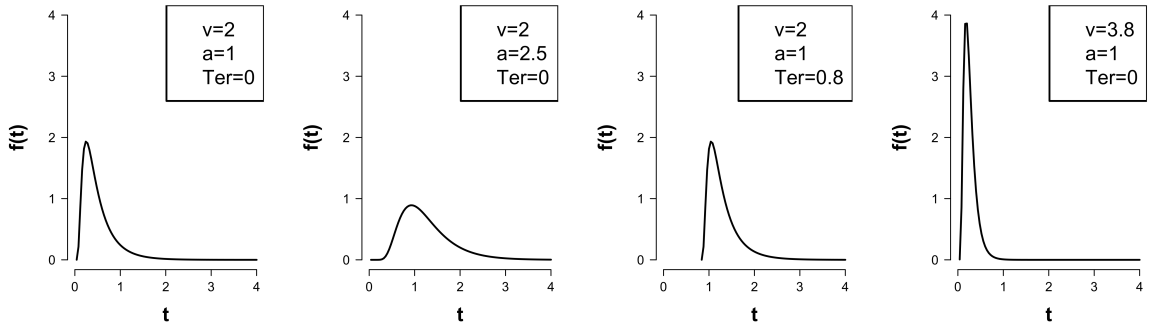

Figure 10. Changes in the shape of the shifted Wald distribution as a result of changes in the parameters v, a and T_{er}. Each panel shows the shifted Wald distribution with different combinations of parameters.

The shifted Wald parameters have a clear psychological interpretation (e.g., Heathcote, 2004; Luce, 1986; Schwarz, 2001, 2002). Participants are assumed to accumulate noisy information until a predefined threshold amount is reached and a response is initiated. Drift rate v quantifies task difficulty or subject ability, response criterion a quantifies response caution, and the shift parameter T_{er} quantifies the time needed for non–decision processes (Matzke & Wagenmakers, in press). Experimental paradigms in psychology for which it is likely that there is only a single absorbing boundary include saccadic eye movement tasks with few errors (Carpenter & Williams, 1995), go/no–go tasks (Gomez, Ratcliff, & Perea, 2007) or simple reaction time tasks (Luce, 1986, pp. 51–57). Here we show how to implement the shifted Wald distribution in WBDev.

Shifted Wald distribution: the WBDev script. Please see the supplementary materials for the WBDev code. Open Blackbox, and open the file "ShiftedWald.odc".

(*1*) We name our module ShiftedWald.

(*2*) The parameters of the distribution, which, in this case are the drift rate v, response caution a and shift T_{er}.

(*4*) We have to declare what type of arguments are the input of the distribution. In this case these are the three scalar parameters of the shifted Wald distribution.

(*6*) This part of the code should define the natural bounds of the distribution. In our case, we take Ter as a lower bound and INF (meaning $+\infty$) as an upper bound.

Now you need to compile the function by pressing "Ctrl–k". Save this file as "ShiftedWald.odc" and copy this file into the appropriate blackbox directory, "...\$BlackBoxComponentBuilder1.5\WBdev\Mod$".

Open the distribution file "distributions.odc" in the directory "...\ *BlackBox Component Builder 1.5\ WBdev\ Rsrc*". Add the line `s ~ "ShiftedWald"(s,s,s) "WBDevShiftedWald.Install"` and then save it. The next time you start Blackbox, the program will know that there exists a distribution called ShiftedWald, and that the inputs are three scalars (single numbers).

The shifted Wald distribution: the model file. Once we implemented the WBDev function in blackbox, we can use the function ShiftedWald in the model. The model file is as follows:

```
model
{
      # prior distributions for shifted Wald parameters
      # drift rate
      v ~ dunif(0,10)

      # boundary separation
      a ~ dunif(0,10)

      # Non-decision time
      Ter ~ dunif(0,1)

      # data are shifted Wald distributed
      for (i in 1:nrt)
      {
            rt[i] ~ ShiftedWald(v,a,Ter)
      }
}
```

The priors for v and a, are Uniform distributions that range from 0 to 10 (i.e., `v ~ dunif(0,10)`, `i.e., a ~ dunif(0,10)`). The prior for T_{er} is a Uniform distribution that ranges from 0 to 1 (i.e., `Ter ~ dunif(0,1)`. With the priors in place, we can use our ShiftedWald function to estimate the posterior distributions for the three model parameters v, a and T_{er} (i.e., `rt[i] ~ ShiftedWald(v,a,Ter)`). Save the lines as a text file and name it "model_shiftedwaldind.txt".

Shifted Wald distribution: the R script. Now, open the R–script "Rscript_ShiftedWald_Individual.R", for the individual analysis into an R–file and run it. Change the directory of the location of the model file and the location of your copy of Blackbox to the appropriate directories. The R–script loads a real dataset from a lexical decision task (Wagenmakers, Ratcliff, Gomez, & McKoon, 2008). Nineteen participants had to quickly decide whether a visually presented letter string was a word (e.g., table) or a nonword (e.g., drapa). We will fit the response times of correct "word" responses of the first participant to the shifted Wald distribution. The response time data can be downloaded from `www.ruudwetzels.com`. After you run the code, WinBUGS should show an MCMC chain similar to the one shown in Figure 11.

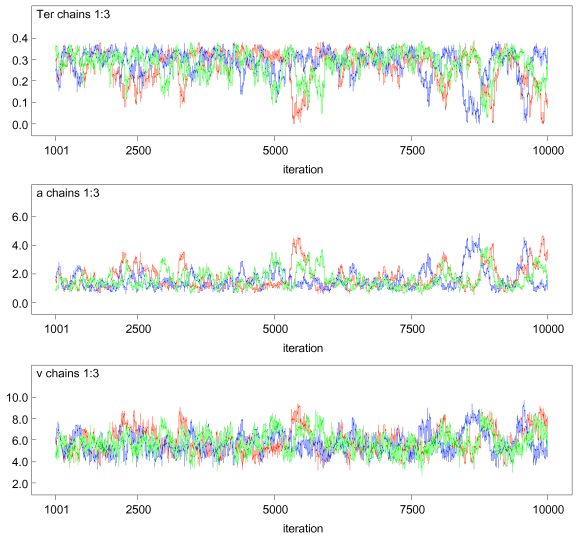

Figure 11. The MCMC chains of of the marginal posteriors of all three individual Wald parameters, v, a and Ter.

The chains do not look like fat hairy caterpillars. They seem to have a lot of freedom to move around the parameter space, so we cannot be certain that the chains have converged properly. To assess convergence more formally, we ran three chains using different starting points for each chain. Next, we calculated $\hat{R}$ to check whether the chains have converged to the same stationary distribution. For each parameter, $\hat{R}$ is smaller than 1.1, so we can tentatively assume that the chains have converged.

Figure 12 shows the posterior distribution of the three shifted Wald parameters, v, a and T_{er}. One thing that stands out is that the posterior distributions of the shifted Wald parameters are very spread out across the parameter space. The 95% credible intervals for v, a and T_{er} extend from 4.12 to 8.00, from 0.80 to 3.52 and from .09 to .36, respectively. It seems that data from only one participant are not enough to yield very accurate estimates of the shifted Wald parameters. In the following section we show how our estimates will improve when we use a hierarchical model and analyze all participants simultaneously.

Shifted Wald distribution: a hierarchical extension. In an experimental setting, the problem of few data per participant can be addressed by hierarchical modeling (Farrell &

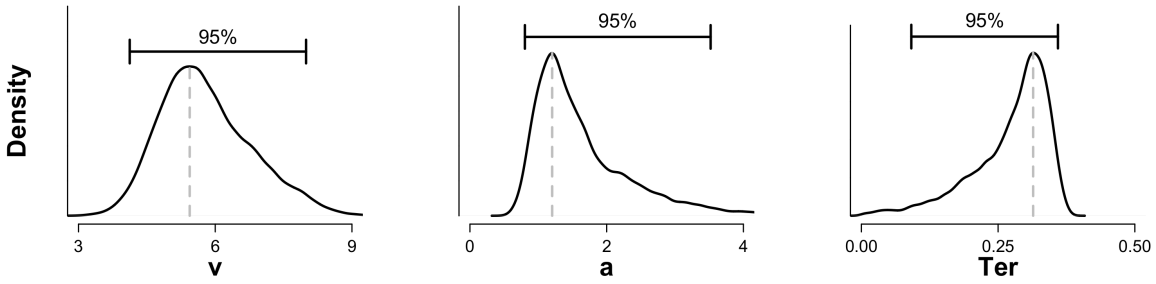

Figure 12. The posterior distribution of the three Wald parameters v, a and T_{er}. The dashed gray lines indicate the modes of the posterior distributions at $v = 5.57$, $a = 1.09$ and $T_{er} = .33$. The 95% credible intervals for v, a and T_{er} extend from 4.12 to 8.00, from 0.80 to 3.52 and from .09 to .36, respectively.

Ludwig, 2008; Gelman & Hill, 2007; Rouder, Sun, Speckman, Lu, & Zhou, 2003; Shiffrin, Lee, Wagenmakers, & Kim, 2008). In our shifted Wald example, each subject is assumed to generate their data according to the shifted Wald distribution, but with different parameter values. We extend the individual analysis and assume that the parameters for each subject are governed by a group Normal distribution. This means that all individual participants are assumed to have their shifted Wald parameters drawn from the same group distribution, allowing the data from all the partipants to be used for inference, without making the unrealistic assumption that participants are identical copies of each other.

The model file that implements the hierarchical shifted Wald analysis is shown below:

```
model
{
      # prior distributions for group means:
      v.g ~ dunif(0,10)
      a.g ~ dunif(0,10)
      Ter.g ~ dunif(0,1)

      # prior distributions for group standard deviations:
      sd.v.g ~ dunif(0,5)
      sd.a.g ~ dunif(0,5)
      sd.Ter.g ~ dunif(0,1)

      # transformation from group standard deviations to group
      # precisions (i.e., 1/var, which is what WinBUGS expects
      # as input to the dnorm distribution):
      lambda.v.g <- pow(sd.v.g,-2)
      lambda.a.g <- pow(sd.a.g,-2)
      lambda.Ter.g <- pow(sd.Ter.g,-2)

      # data come From a shifted Wald distribution
      for (i in 1:ns)          #subject loop
      {
            # individual parameters drawn from group level
            # normals censored to be positive using the
```

```
            # I(0,) command:
            v.i[i] ~ dnorm(v.g,lambda.v.g)I(0,)
            a.i[i] ~ dnorm(a.g,lambda.a.g)I(0,)
            Ter.i[i] ~ dnorm(Ter.g,lambda.Ter.g)I(0,)

            # for each participant,
            # data are shifted Wald distributed
            for (j in 1:nrt[i])
            {
                  rt[i,j] ~ ShiftedWald(v.i[i],a.i[i],Ter.i[i])
            }
      }
}
```

The hierarchical analysis of the reaction time data proceeds as follows. The prior for the group means is a Uniform distribution, ranging from 0 to 10 (i.e., `v.g` $\sim$ `dunif(0,10)`, `a.g` $\sim$ `dunif(0,10)`) or from 0 to 1 (i.e., `Ter.g` $\sim$ `dunif(0,1)`). The standard deviations are drawn from a Uniform distribution ranging from 0 to 5 (i.e., `sd.v.g` $\sim$ `dunif(0,5)`, `sd.a.g` $\sim$ `dunif(0,5)`) or from 0 to 1 (i.e., `sd.Ter.g` $\sim$ `dunif(0,5)`). Next, the standard deviations have to be transformed to precisions (i.e., `lambda.v.g <- pow(sd.v.g,-2)`, `lambda.a.g <- pow(sd.a.g,-2)`, `lambda.Ter.g <- pow(sd.Ter.g,-2)`). Then, the individual parameters $v.i$, $a.i$ and $Ter.i$ are drawn from Normal distributions with corresponding group means and group precisions (i.e., `v.i[i]`$\sim$`dnorm(v.g,lambda.v.g)I(0,)`, `a.i[i]` $\sim$ `dnorm(a.g, lambda.a.g)I(0,)`, `Ter.i[i]` $\sim$ `dnorm(Ter.g, lambda.Ter.g)I(0,)`). For each individual, the data are distributed according to a shifted Wald distribution with their own individual parameters. Save the model file as a text file and name it: "model_shiftedwaldhier.txt".

When we run this model using the R–script for the hierarchical analysis, we first focus on the group mean parameters $v.g$, $a.g$ and $T_{er}.g$. Figure 13 shows the MCMC chains from the three shifted Wald parameters. To check for convergence, we ran three chains, with all three having a different starting position, and then calculate $\hat{R}$. The chains appear to have converged, an impression that is supported by $\hat{R}$ values close to 1 ($\hat{R}$ for $Ter.g$, $a.g$ and $v.g$ is approximately 1.).

Figure 14 shows the posterior distributions of the shifted Wald group–mean parameters. The distributions indicate that there is relatively little uncertainty about the parameter values. The posterior distributions of the group–mean parameters are concentrated around their modes $v.g = 4.27$, $a.g = 0.97$, and $T_{er}.g = 0.36$. The 95% credible intervals for $v.g$, $a.g$ and $T_{er}.g$ extend from 3.80 to 4.70, from 0.85 to 1.10 and from .34 to .38, respectively.

It is informative to consider the influence of the hierarchical extension on the individual estimates for the shifted Wald parameters. Specifically, we can examine the MCMC chains for the same subject that we analyzed in the individual shifted Wald analysis, but now in the hierarchical setting.

After you run the R–script "Rscript_ShiftedWald_Hierarchical.R", for the hierarchical analysis of the shifted Wald example, WinBUGS should show three MCMC chains similar to the ones shown in Figure 15. The chains are better behaved than the chains from the

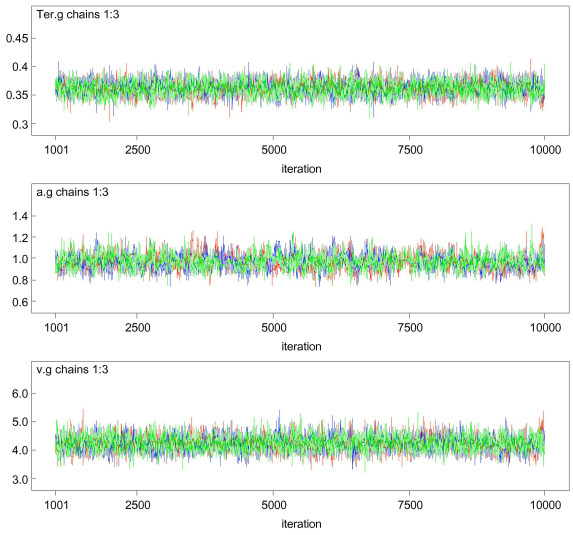

Figure 13. Three chains, consisting of 9000 MCMC draws each, from the posterior distributions of the three "group–level" shifted Wald parameters, $v.g$, $a.g$ and $T_{er}.g$.

individual analysis (Figure 11). The hierarchical extension leads to a practical improvement, through faster convergence for the computational MCMC estimation process. However, the hierarchical extension also leads to a theoretical improvement because compared to the individual analysis, the chains appear much less diffuse. This shows that the hierarchical model leads to a better understanding of the model parameters.

To underscore this point, Figure 16 shows the posterior distributions of the individual shifted Wald parameters, for both the hierarchical analysis and the individual analysis. It is clear that the posterior distributions of the shifted Wald parameters are less spread out in the hierarchical analysis than in the individual analysis. Also, the parameter estimates from the hierarchical analysis are slightly different than those from the individual analysis. More precisely, they seem to have moved towards their common group mean. This effect is

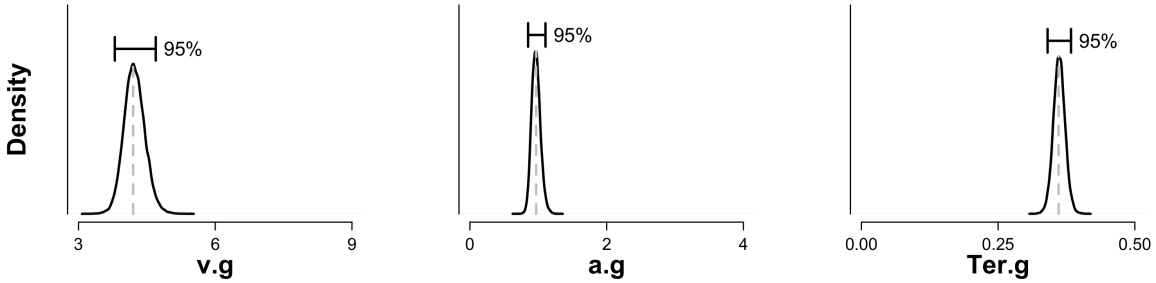

Figure 14. The posterior distribution of the three "group–level" shifted Wald parameters $v.g$, $a.g$ and $T_{er}.g$. The dashed gray lines indicate the modes of the posterior distributions at $v.g = 4.27$, $a.g = .97$ and $T_{er}.g = .36$. The 95% credible intervals for $v.g$, $a.g$ and $T_{er}.g$ extend from 3.80 to 4.70, from 0.85 to 1.10 and from .34 to .38, respectively.

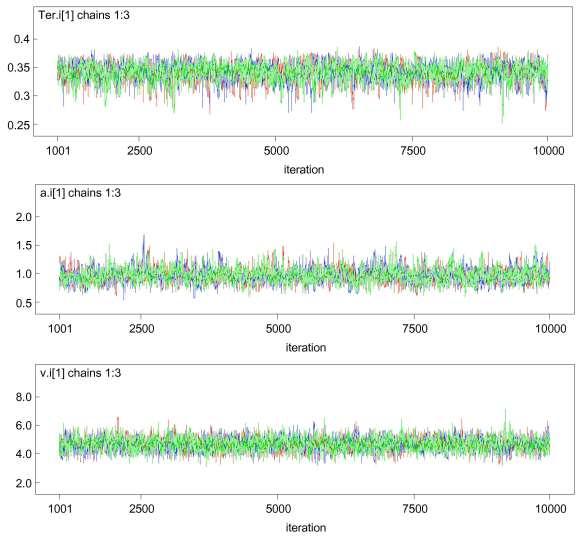

Figure 15. The MCMC chains of of the marginal posteriors of all three individual Wald parameters, v, a and Ter, analyzed using a hierarchical model.

called *shrinkage*, and is a standard and important property of hierarchical models (Gelman et al., 2004).

In sum, the WBDev implementation of the shifted Wald distribution enables researchers to infer shifted Wald parameters from reaction time data. Not only does WinBUGS allow straightforward analyses on individual data, it also makes it easy to add hierarchical structure to the model. This can greatly improve the quality of the posterior estimates, and is often a very sensible and informative way of analyzing data.

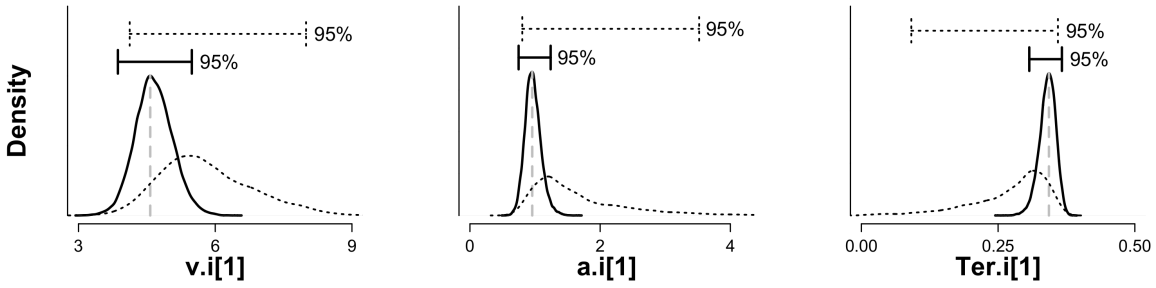

Figure 16. The posterior distribution of the three individual shifted Wald parameters $v.i$, $a.i$ and $T_{er}.i$ from the hierarchical analysis (solid lines) and the individual analysis (dotted lines). The dashed gray lines indicate the modes of the posterior distributions from the hierarchical analysis at $v.i[1] = 4.57$, $a.i[1] = 0.96$ and $T_{er}.i[1] = .34$. The 95% credible intervals in the hierarchical model for $v.i[1]$, $a.i[1]$ and $T_{er}.i[1]$ extend from 3.86 to 5.49, from 0.75 to 1.24 and from .31 to .37,respectively.

Discussion

In this paper we have shown how the WinBUGS Development Interface (WBDev) can be used to help psychological scientists model their sparse, noisy, but richly structured data. We have shown how a relatively complex function such as the Expectancy–Valence model can be incorporated in a fully Bayesian analysis of data. Furthermore, we have shown how to implement statistical distributions, such as the shifted Wald distribution, that have specific application in psychological modeling, but are not part of a standard set of statistical distributions.

The WBDev program is set up for Bayesian modeling, and is equipped with modern sampling techniques such as Markov chain Monte Carlo. These sampling techniques allow researchers to construct quantitative Bayesian models that are non–linear, highly structured, and potentially very complicated. The advantages of using WBDev together with WinBUGS are substantial. WinBUGS code can sometimes lead to slow computation and complex models might not work at all. Scripting some components of the model in WBDev can speed up the computation time considerably. Furthermore, compartmentalizing the scripts can make the model easier to understand and debug. Moreover, WinBUGS facilitates statistical communication between researchers who are interested in the same model. The most basic advantage, however, is that WBDev allows the user to program functions and distributions that are simply unavailable in WinBUGS.

Once a core psychological model is implemented in WBDev, it is straightforward to take into account variability across participants or items using a hierarchical, multi–level extension (i.e., models with random effects for subjects or items). This approach allows a researcher to model individual differences as smooth variations in parameters of a certain cognitive model, reaching an optimal compromise between the extremes of complete pooling (i.e., treating all participants as identical copies) and complete independence (i.e., treating each participant as a fully independent unit). In general, statistical models that are implemented in WinBUGS can be easily implemented to deal with the complexities that plague empirical data –such as when data are missing, the sampling plan is unclear or participants originate from different subgroups. For this reason, we believe the fully Bayesian analysis of highly structured models is likely to be a driving force behind future theoretical and empirical progress in the psychological sciences.

References

Bechara, A., Damasio, A. R., Damasio, H., & Anderson, S. (1994). Insensitivity to future consequences following damage to human prefrontal cortex. *Cognition*, *50*, 7–15.

Bechara, A., Damasio, H., Tranel, D., & Damasio, A. R. (1997). Deciding advantageously before knowing the advantageous strategy. *Science*, *275*, 1293–1295.

Brown, G., Neath, I., & Chater, N.(2007). A temporal ratio model of memory. *Psychological Review*, *114*, 539–576.

Busemeyer, J. R., & Stout, J. C. (2002). A contribution of cognitive decision models to clinical assessment: Decomposing performance on the Bechara gambling task. *Psychological Assessment*, *14*, 253–262.

Carpenter, R. H. S., & Williams, M. L. L.(1995). Neural computation of log likelihood in control of saccadic eye movements. *Nature*, *377*, 59–62.

Cowles, M. K. (2004). Review of WinBUGS 1.4. *The American Statistician*, *58*, 330–336.

Cowles, M. K., & Carlin, B. P. (1996). Markov chain Monte Carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 883–904.

Donkin, C., Averell, L., Brown, S., & Heathcote, A. (in press). Getting more from accuracy and response time data: Methods for fitting the linear ballistic accumulator model. *Behavior Research Methods*.

Farrell, S., & Ludwig, C. (2008). Bayesian and maximum likelihood estimation of hierarchical response time models. *Psychonomic Bulletin & Review*, *15*, 1209–1217.

Gamerman, D., & Lopes, H. F. (2006). *Markov chain Monte Carlo: Stochastic simulation for Bayesian inference*. Boca Raton, FL: Chapman & Hall/CRC.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2004). *Bayesian data analysis (2nd ed.)*. Boca Raton (FL): Chapman & Hall/CRC.

Gelman, A., & Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge: Cambridge University Press.

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences (with discussion). *Statistical Science*, *7*, 457–472.

Gilks, W. R., Richardson, S., & Spiegelhalter, D. J. (Eds.). (1996). *Markov chain Monte Carlo in practice*. Boca Raton (FL): Chapman & Hall/CRC.

Gomez, P., Ratcliff, R., & Perea, M. (2007). A model of the go/no–go task. *Journal of Experimental Psychology: General*, *136*, 389–413.

Heathcote, A. (2004). Fitting Wald and ex–Wald distributions to response time data: An example using functions for the S–PLUS package. *Behavior Research Methods, Instruments, & Computers*, *36*, 678–694.

Hoijtink, H., Klugkist, I., & Boelen, P. (2008). *Bayesian evaluation of informative hypotheses that are of practical value for social scientists*. New York: Springer.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kruschke, J. K. (1992). ALCOVE: An exemplar–based connectionist model of category learning. *Psychological Review*, *99*, 22–44.

Lee, M. D. (2008). Three case studies in the Bayesian analysis of cognitive models. *Psychonomic Bulletin & Review*, *15*, 1–15.

Lee, M. D., & Wagenmakers, E.-J. (2009). A course in Bayesian graphical modeling for cognitive science. Unpublished course materials, retrieved September 10, 2009, from E–J Wagenmakers' website: `http://users.fmg.uva.nl/ewagenmakers/Bayescourse/Bayesbook.pdf`.

Luce, R. D. (1959). *Individual choice behavior*. New York: Wiley.

Luce, R. D. (1986). *Response times*. New York: Oxford University Press.

Lunn, D. (2003). WinBUGS Development Interface (WBDev). *ISBA Bulletin*, *10*, 10–11.

Lunn, D., Spiegelhalter, D., Thomas, A., & Best, N. (in press). The BUGS project: Evolution, critique and future directions. *Statistics in Medicine*.

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

Matzke, D., & Wagenmakers, E.-J. (in press). Psychological interpretation of ex–Gaussian and shifted Wald parameters: A diffusion model analysis. *Psychonomic Bulletin & Review*.

Nosofsky, R. (1986). Attention, similarity, and the identification–categorization relationship. *Journal of Experimental Psychology: General*, *115*, 39–57.

Ntzoufras, I. (2009). *Bayesian modeling using WinBUGS*. Hoboken (NJ): Wiley-Blackwell.

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

Rouder, J. N., & Lu, J. (2005). An introduction to Bayesian hierarchical models with an application in the theory of signal detection. *Psychonomic Bulletin & Review*, *12*, 573–604.

Rouder, J. N., Sun, D., Speckman, P., Lu, J., & Zhou, D. (2003). A hierarchical Bayesian statistical framework for response time distributions. *Psychometrika*, *68*, 589–606.

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

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

Sheu, C.-F., & O'Curry, S. L. (1998). Simulation–based Bayesian inference using BUGS. *Behavioral Research Methods, Instruments, & Computers*, *30*, 232–237.

Shiffrin, R. M., Lee, M. D., Wagenmakers, E.-J., & Kim, W. J. (2008). A survey of model evaluation approaches with a focus on hierarchical Bayesian methods. *Cognitive Science*, *32*, 1248–1284.

Vandekerckhove, J., Tuerlinckx, F., & Lee, M. D. (2009). Hierarchical diffusion models for two–choice response time. Submitted.

Wagenmakers, E.-J., Ratcliff, R., Gomez, P., & McKoon, G. (2008). A diffusion model account of criterion shifts in the lexical decision task. *Journal of Memory and Language*, *58*, 140–159.

Wald, A. (1947). *Sequential analysis*. Wiley New York.

Wetzels, R., Vandekerckhove, J., Tuerlinckx, F., & Wagenmakers, E.-J. (in press). Bayesian parameter estimation in the Expectancy Valence model of the Iowa gambling task. *Journal of Mathematical Psychology*.