Article:
Lin, Y-S orcid.org/0000-0002-2454-6601 and Strickland, L (2020) Evidence accumulation models with R: A practical guide to hierarchical Bayesian methods. The Quantitative Methods for Psychology, 16 (2). pp. 133-153. ISSN 2292-1354

10.20982/tqmp.16.2.p133

2020 ■ Vol. 16 ■ no. 2

Evidence accumulation models with R: A practical guide to hierarchical Bayesian methods

Yi-Shin Lin [a] and Luke Strickland [b]

[a] University of Leeds
[b] Curtin University

Abstract ■ Evidence accumulation models are a useful tool to allow researchers to investigate the latent cognitive variables that underlie response time and response accuracy. However, applying evidence accumulation models can be difficult because they lack easily computable forms. Numerical methods are required to determine the parameters of evidence accumulation that best correspond to the fitted data. When applied to complex cognitive models, such numerical methods can require substantial computational power which can lead to infeasibly long compute times. In this paper, we provide efficient, practical software and a step-by-step guide to fit evidence accumulation models with Bayesian methods. The software, written in C++, is provided in an R package: 'ggdmc'. The software incorporates three important ingredients of Bayesian computation, (1) the likelihood functions of two common response time models, (2) the Markov chain Monte Carlo (MCMC) algorithm (3) a population-based MCMC sampling method. The software has gone through stringent checks to be hosted on the Comprehensive R Archive Network (CRAN) and is free to download. We illustrate its basic use and an example of fitting complex hierarchical Wiener diffusion models to four shooting-decision data sets.

Keywords ■ Population-based Markov Chain Monte Carlo, Bayesian cognitive modeling, hierarchical cognitive models.. **Tools** ■ R, ggdmc, C++.

✉ yishinlin001@gmail.com

doi 10.20982/tqmp.16.2.p133

Acting Editor ■ Cheng-Ta Yang (Department of Psychology, National Cheng Kung University)

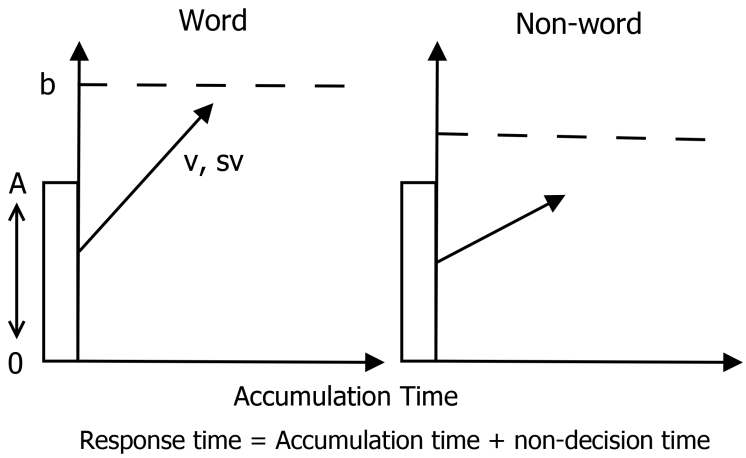
Reviewers ■ Two anonymous reviewers

Introduction

Mathematical models of cognition are essential tools for understanding how observable human behaviors are driven by latent cognitive variables (Farrell & Lewandowsky, 2018). By providing a formal, completely specified process account of how the cognitive system produces behavior, mathematical models can provide insights beyond those offered by conventional statistics. For example, evidence accumulation models (Forstmann & Wagenmakers, 2016) can decompose observed response choice and response time (RT) data into latent psychological variables such as information processing speed and decision thresholds. Over the past decades, such models have proved successful in accounting for a range of human behaviors, and increasingly also their neural underpinnings (Forstmann, Brown, Dutilh, Neumann, & Wagenmakers, 2010; Nunez, Vandekerckhove, & Srinivasan, 2017; Ratcliff, Philiastides, & Sajda, 2009; Ratcliff, Sederberg, Smith, & Childers, 2016). In this paper, we introduce an R package, `ggdmc` that allows applied researchers efficiently and accurately estimate the parameters of evidence accumulation models. The `ggdmc` package provides support to specify models in the context of complex factorial designs. This makes it ideal for psychology research, where such factorial designs are common. Furthermore, `ggdmc` provides an efficient Bayesian sampler that will allow users to fit evidence accumulation models on their own personal computers, without the need for extensive computing resources. In addition to introducing `ggdmc`, we walk the reader step by step through multiple examples of evidence accumulation modeling.

Evidence accumulation models assume that when humans are presented a stimulus, they accumulate evidence towards the possible decisions they could make about that stimulus towards threshold, and the threshold reached

Figure 1 ■ An illustration of the linear ballistic accumulator model. Upon stimulus presentation, evidence accumulates towards each possible decision in parallel, and the first accumulator to reach threshold determines the observed response. Total RT is the time for the first accumulator to reach threshold plus non-decision time. A and b stand for the upper bound of the start-point noise and the response threshold. v and sv stand for the accumulation rate and its standard deviation. Figure adapted from Strickland, Loft, Remington, and Heathcote (2018).



determines the observed decision. For example, the linear ballistic accumulator (LBA) model (Brown & Heathcote, 2008) assumes that evidence for each possible decision (e.g. word, & non-word) accrues towards its respective threshold in a separate, independent, accumulator. We depict the LBA model, as it would apply to a lexical decision task in which participants decide whether items are 'words' or 'non-words' (Figure 1). When the stimulus is presented, each accumulator begins with a preliminary level of evidence drawn from the uniform distribution $U[0, \mathrm{A}]$. As the stimulus is processed, evidence to each decision accrues linearly towards threshold (b) at an accumulation rate (v). The observed decision is determined by the first accumulator to reach response threshold. Thresholds control strategic decision processes, such as the speed accuracy tradeoff (by raising thresholds, individuals can trade speed for accuracy). Accumulation rates, which are drawn from a normal distribution (mean v, & standard deviation s_v), measure the quality and quantity of human information processing. Total RT in the LBA is determined by combining total decision time (time for the first accumulator to reach threshold), with an estimated non-decision time parameter, included to capture phenomena such as stimulus encoding and motor responding.

Another commonly used evidence accumulation model is the diffusion decision model (Ratcliff, 1978; Ratcliff & McKoon, 2008), depicted in Figure 2 and Table 1. The diffusion model is similar to the LBA model in that it assumes that evidence accumulation begins at a starting point (z, drawn from a uniform distribution) and proceeds towards threshold (a or 0), with the threshold reached determining the decision. It differs from the LBA model in that it assumes a single relative evidence accumulation process, in which evidence towards one decision (e.g. evidence moving towards the 'word' threshold in Figure 2) is evidence against competing decisions. It also differs from the LBA model in that evidence accumulation proceeds noisily, with an average direction given by a drift rate (v). This drift rate is drawn from a normal distribution (with standard deviation s_v). One useful simplified version of the diffusion model is the Wiener diffusion model, in which the variabilities of the drift rate and the starting point are 0. The Wiener diffusion model is useful because it takes enormous amount of computational time to calculate the DDM likelihoods with the variability parameters. When they are not main research interests and evidence indicates they influence little on data, the Wiener diffusion model becomes a simple and effective model for fitting the evidence accumulation process.

Often, when fitting evidence accumulation models, the goal is to accurately estimate the parameters that best account for observed data. For example, a researcher may wish to know whether their experimental manipulation slowed the rate of information processing (reduced evidence accumulation rates) or increased response caution (increased thresholds). However, accurately estimat-

Table 1 ■ The parameters of the diffusion decision model.

	Symbol	Parameter
Decision process	a	Boundary separation
	z	Starting point
	v	mean drift rate
Non-decision	t_0	Non-decision time
	d	The differences in the non-decision time between upper and lower boundaries
Inter-trial variability	s_z	Inter-trial range of z
	s_v	Inter-trial standard deviation of the drift rate
	s_{t0}	Inter-trial range of t_0

ing the parameters of evidence accumulation can be both computationally demanding (Holmes & Trueblood, 2018; Holmes, Trueblood, & Heathcote, 2016; Miletic, Turner, Forstmann, & van Maanen, 2017; Turner, Dennis, & Van Zandt, 2013), and technically challenging. Best practice is to fit *hierarchical* evidence-accumulation models, which estimate separate parameters for each individual participant but constrain these estimates with a population level distribution (Boehm, Marsman, Matzke, & Wagenmakers, 2018). To facilitate parameter estimation, the `ggdmc` package provides a suite of functions designed to check model adequacy, as well as optimized C++ code for fast computation. Before discussing the package in more depth, we provide the reader a brief background to methods of parameter estimation. We begin by introducing the standard method of model parameter estimation, maximum likelihood estimation (MLE) (Myung, 2003). We then discuss Bayesian estimation, which is the focus of `ggdmc` because it provides a flexible and coherent inferential framework.

Maximum Likelihood Estimation

MLE finds the parameter values, θ, that make the data most likely under a given model (Fisher, 1920). It typically maximizes the log-likelihood function, $\ln L(\theta|x)$ by calculating the first and second derivatives of the function, to estimate the θ giving maxima. However, this analytic method is typically not feasible for evidence accumulation models. Instead, MLE often relies on optimization routines, which numerically find the minimum of a function of n parameters by first applying negation to the function and then iteratively making sensible guesses for the parameters.

We now exemplify how MLE would proceed with numerical methods fitting a Wiener diffusion model to a simple data set. θ is a vector containing four parameter values: the boundary separation, the drift rate, the starting point and the non-decision time. The data, x, are stored in a matrix with two columns, one for the continuous dependent variable, RTs, and the other for the discrete variable, response choices. Each row of the matrix is an observation. To calculate the likelihood of one observation x_i, $p(x_i|\theta)$, we enter the parameter values, the RT and the observed response choice into the probability density function of the Wiener diffusion model. To get the likelihood for the entire data set, the likelihoods of all data points are multiplied together (or the log likelihoods summed). MLE would proceed by iteratively searching for the parameters that maximize this value.

Searching for the MLE estimates numerically can be problematic in evidence accumulation models. For one, it relies on the assumption that the MLE estimates exist and is unique, which is often difficult, if not impossible, to verify in high-dimensional models. Further, parameter searches are prone to getting stuck in ranges that give local, but not global, maximum likelihoods. A more general issue is that MLE does not easily yield estimates of uncertainty in parameter values, which is important for sound statistical inference. To surmount these difficulties, `ggdmc` implements Bayesian estimation. Although there are frequentist workarounds to the previously mentioned problems, the Bayesian solution is unique in that it provides a coherent framework for the modeler to include prior information in their cognitive model. The `ggdmc` package allows flexible Bayesian specifications over factorial experimental designs, a feature that is missing from many existing tools (Ahn, Haines, & Zhang, 2017; Annis, Miller, & Palmeri, 2017; Wiecki, Sofer, & Frank, 2013).

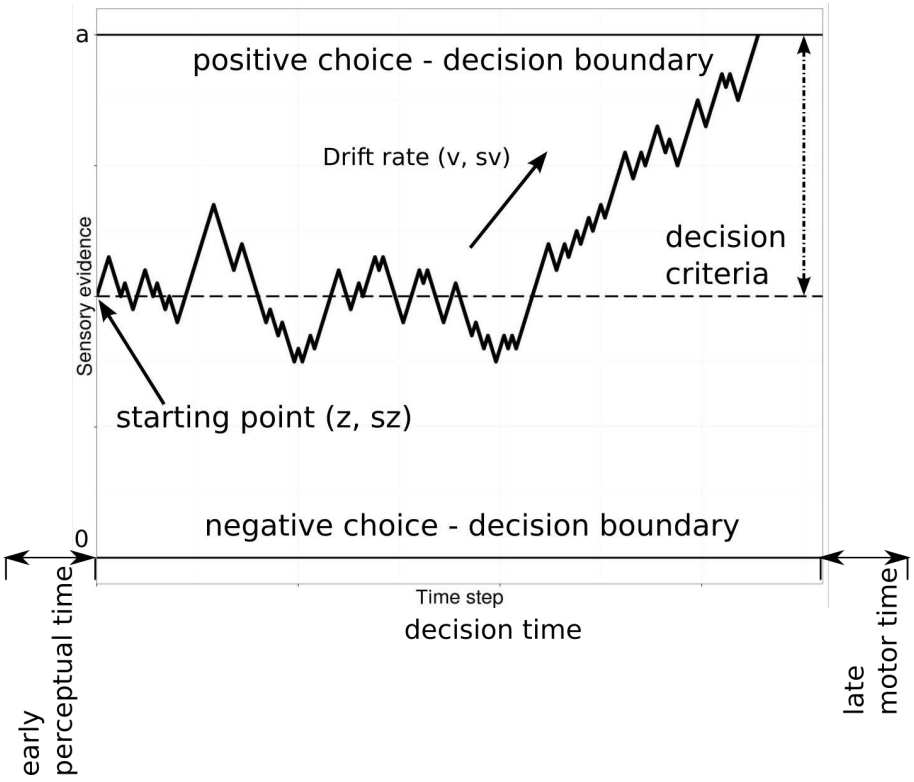
Bayesian Estimation

Whereas MLE finds parameter values, θ, by maximizing the data likelihoods, Bayesian estimation aims to obtain the probability distribution of θ given the data. The foundation of Bayesian estimation is the classic Bayes' theorem (Bayes, Price, & Canton, 1763),

$$p(\theta|x) = \frac{p\,(x|\theta)\,p\,(\theta)}{p\,(x)}. \tag{1}$$

The notation $p(\theta|x)$ stands for the probability of θ given the data (x), referred to as the posterior probability. Bayes' theorem states that the posterior probability of θ equals the product of the likelihood, $p\,(x|\theta)$, and the prior probability,

Figure 2 ■ An illustration of the diffusion decision model. The between-trial variabilities for the starting point, sz, and the drift rate, s_v, are grouped together with the main parameters, v and z. A positive choice (a) corresponds to, for instance a word stimulus with a word response. A negative choice (0) corresponds to a word stimulus with a non-word response. Non-decision time = early perceptual time + late motor time. RT = Non-decision time + decision time. The within-trial variability of the drift rate, the stochastic feature of the model, is illustrated by the wiggled line.



$p(\theta)$ divided by the marginal likelihood of the data, $p(x)$. The marginal likelihood is an expectation of the data probability over the entire range of possible model parameters,

$$p(x) = E(p(x|\theta)) = \int p(x|\theta)\, p(\theta)\, d\theta.$$

Although the posterior distribution can sometimes be derived analytically, this is typically not the case with evidence accumulation models. Fortunately, Bayes' theorem, when applied together with a particular stochastic process, a Markov chain, becomes a powerful tool to sample random quantities from the posterior distribution. Markov Chain Monte Carlo (MCMC) is a stochastic process that can learn the statistical characteristics of model parameters by sampling random quantities from the target distribution function. MCMC methods allow estimating the posterior distribution because they visit parameter space with frequencies proportional to the posterior density of interest.

One prominent MCMC method is the Metropolis-Hastings algorithm (Hastings, 1970). At each step, the algorithm proposes a new set of model parameters and calculates its posterior likelihood, which is compared to the posterior likelihood calculated based on existing parameters. With the likelihood comparison, the algorithm yields an acceptance ratio. Since the marginal likelihood is independent of the proposed model parameters, the acceptance ratio can be calculated without it, resulting in equation two.

$$r = \frac{p(x|\theta_{t+1})\, p(\theta_{t+1})}{p(x|\theta_t)\, p(\theta_t)}. \tag{2}$$

If the ratio is greater than 1, it indicates θ_{t+1} is more likely than θ_t and the sample of θ_{t+1} is guaranteed to be accepted. If the ratio is not greater than 1, the sample may be accepted with some probability.

MCMC sampling is a natural choice to fit hierarchical evidence accumulation models. However, this method can be slow because it requires many calls to the model likelihoods. Although significant advances have been made to increase the computational speed of evidence accumula-

tion model likelihoods (Brown & Heathcote, 2008; Navarro & Fuss, 2009; Voss, Rothermund, & Voss, 2004; Voss & Voss, 2008; Wagenmakers, Maas, Dolan, & Grasman, 2008), they still cost more time than standard statistical models. Such computational challenges have made Bayesian evidence accumulation modeling unobtainable for researchers lack of high-performance computational resources. In addition, psychological experiments utilize factorial designs, resulting in data with multiple participants and conditions. Specifying evidence accumulation models for complex factorial designs requires substantial skills with specialized Bayesian tools (Carpenter et al., 2017; Lunn, Spiegelhalter, Thomas, & Best, 2009; Plummer, 2003), posing another challenge to the adoption of Bayesian evidence accumulation modeling.

To facilitate working with Bayesian evidence accumulation models, we present a new, highly efficient R package, `ggdmc`, that incorporates a user-friendly interface for entering various factorial designs (Heathcote et al., 2019; Vandekerckhove & Tuerlinckx, 2007). Below we provide a general overview of the steps for researchers to apply `ggdmc`. We then provide step-by-step examples applying both the diffusion decision model and the LBA model with `ggdmc`. Finally, we illustrate the utility of `ggdmc` with a real-world example, fitting hierarchical Wiener diffusion models to the data of four shooting decision studies. The hierarchical diffusion model assumes the subject-level model parameters themselves are random variables drawn from distributions. This sets it apart from the conventional diffusion model analysis where one fits the model to the data of individual participant and summarizes the data across participants. The studies examined how racial stereotyping can affect shooting decisions (Pleskac, Cesario, & Johnson, 2018). We conclude by discussing some more complex cognitive models that have been implemented in `ggdmc`, alternative solutions of hierarchical Bayesian evidence accumulation modeling, and future package development.

ggdmc package

System Requirements

The `ggdmc` package was developed in the R environment (R Core Team, 2018) and is released in the forms of source code and compiled binaries. Compiling the former requires `Rtools` on Windows or `Xcode` on macOS. To set up an R environment, one can obtain the R base binaries as well as Rtools freely at the Comprehensive R Archive Network (CRAN), https://cran.r-project.org/ under the terms of GNU General Public License. The CRAN has a dedicated web page, *R for Mac OS X*, detailing how to set up development tools in macOS. Although the package was originally developed on Linux platforms, it has been tested on a wide range of different platforms. `ggdmc` requires six supporting R packages: `Rcpp`, `RcppArmadillo`, `coda`, `data.table`, `matrixStats` and `ggplot2`. The first two bridge the internal C++ code in `ggdmc` with the R interface. The third provides many post-hoc checking tools for Bayesian estimation. The fourth and fifth are computationally efficient R packages for operating upon square data structures and matrices. The last is the well-known package of grammar of graphics (Wickham, 2016), providing aesthetic plotting functions to facilitate visual checks of posterior samples .

Installation

The software can be downloaded freely at the CRAN permanent address, https://cran.r-project.org/web/packages/ggdmc/index.html. There are several methods to install an R package. Since `ggdmc` adheres to the CRAN policy, all installation methods of R packages apply to `ggdmc`. One convenient way is via the CRAN infrastructure by entering, `install.packages(ggdmc)`. For other methods, we provide a README file in the package (see also the package Github, https://github.com/yxlin/ggdmc).

An Overview of Bayesian Analysis

We briefly summarize the steps for data analysis that we follow below. The first step is loading and formatting data to the data frame in R language. The next step is to specify the model and associate the model parameters with the experimental design (see more on this in the *Model Specification* section). After this one must run the MCMC sampling. This requires specifying the numbers of samples to obtain, chains to run and the degree of thinning. A wise choice of these MCMC specifics requires experience with Bayesian analysis. Thus far there is not an established set of general principles that can guide these choices when fitting hierarchical Bayesian evidence-accumulation models. Upon acquiring the modeling results, one must proceed to diagnose the posterior distributions to determine whether sampling has been successful. This is crucial to warrant a reliable and unbiased inference. The last analytic step is to review whether the model does provide reasonably close fit to the data (i.e., retrodiction), and perhaps whether its posterior predictions can cover unseen data. The user can then proceed to use the posterior distributions of the model parameters to make inference regarding to their research questions.

Example 1: Fitting One Subject Data

Below, we give an overview of the process of fitting Bayesian evidence accumulation models step by step: specifying models, setting up prior distributions, drawing posterior samples, diagnosing whether posterior sampling

Listing 1 ■ Generating random data for Example 1.

```
p.vector <- c(a = 1, v = 1.2, z = .38, sz = .25, sv = .2, t0 = .15)
ntrial   <- 20

require(ggdmc)
model <- BuildModel(
  p.map     = list(a = "1", v = "1",  z = "1", d = "1",
                   sz = "1", sv = "1",  t0 = "1", st0 = "1"),
  match.map = list(M =list (s1 = "r1", s2 = "r2")),
  factors   = list(S = c ("s1", "s2")),
  responses = c("r1", "r2"),
  constants = c(st0 = 0, d = 0),
  type      = "rd")

## Parameter vector names are: ( see attr (," p.vector ") )
## [1] "a"  "v"  "z"  "sz" "sv" "t0"
##
## Constants are (see attr (," constants ") ):
## st0  d
## 0    0
## Model type = rd

dat <- simulate(model, nsim = ntrial, ps = p.vector)
```

was successful, and summarizing posterior samples. We follow by demonstrating the steps to estimate the parameters of diffusion model for a simulated data set. The example concludes by discussing diffusion model "parameter recovery", which refers to how well a model can recover its true data-generating parameters, and by emphasizing the importance of running parameter recovery studies.

Methods

Modeling Steps

The first and perhaps most daunting step in evidence accumulation modeling is to specify a likelihood function. Deriving entirely novel model likelihoods is beyond the scope of this tutorial. However, `ggdmc` provides the likelihoods of two common, useful, accumulation models: the LBA and the diffusion decision model. The second step is to specify an experimental design, a process which is often laborious with Bayesian model fitting software. Our package includes a convenient interface for specifying different designs, which we discuss later when walking through examples. The next critical step in Bayesian estimation is to specify initial beliefs about the values of each model parameter, often referred to as the prior. Practically speaking, this requires specifying a probability distribution corresponding to uncertainty about each model parameter. In our examples we provide prior distributions with minimal information, but that includes sensible constraints. For example, we are certain before data are observed that the non-decision times in RT models must be 0 or above, and therefore the prior for non-decision time is bounded to be above 0. After specifying the likelihood function, the prior distributions and the factorial design, the fourth step is to draw posterior samples. Often, posterior samples are drawn by multiple Markov chains. Two critical choices we must make are (1) how many posterior samples to draw and (2) how many Markov chains to launch. After this, we must diagnose whether sampling has been successful, by evaluating whether the Markov chains are stationary, mixed, and converged (Gelman et al., 2013). If sampling is not successful, the sampling procedure may need to be altered. For example in fitting the hierarchical DDM, we increased thinning to 8 to increase the number of posterior samples obtained without overburdening memory. We judged it necessary to obtain more posterior samples because the boundary separation parameter correlates with the starting point parameter in the DDM. Often, several possible candidate models are sampled and compared, and the model that provides the most accurate and parsimonious fit to the observed data is retained. The last step is typically to summarize the parameter values of the chosen model. This is key to answering many research questions, for instance, are accumulation rates higher in one condition than another?

Table 2 ■ `BuildModel` arguments.

Name	Symbol	Interpretation
Parameter map	`p.map`	The association between model parameters with experimental factors
Match map	`match.map`	The association between responses and stimuli.
Factors	`factors`	The factor levels
Responses	`responses`	Response types.
Constants	`constants`	The option fixes model parameters at a constant value.

Simulating data

Data should be loaded into an appropriately formatted R `data.frame`, with each observation on a row and each variable on a column. The data must have four key variables, `s`, `S`, `R`, and `RT`, storing subject, stimulus, response, and response time. The first three columns are categorical and the last is continuous. To add additional experimental factors, the user may enter other factors between the `S` and `R` column.

This example creates synthetic data using `ggdmc`'s `simulate` funciton, and then walk through model-fitting to that as if it was the 'real' data set. The code below specifies a 'true' parameter vector for the simulated data and specifies the number of observations. Listing 1, lines 1 and 2, shows the instructions.

First and foremost, we must load `ggdmc` to access its R functions. The simulation function takes three arguments, the model object, which will be discussed in the next subsection, `nsim`, standing for the number of simulated trials observed per experimental design cell, and `ps` for (true) parameter(s).

Model Specification

The function `BuildModel` is critical to mapping model parameters to experimental designs. An example using `BuildModel` is shown in Listing 1, lines 4 to 12.

Table 2 lists the interpretations of the five arguments to `BuildModel`, which uses the R data type, `list` to group the parameters (see Table 1 for the parameters in `rd` type). The function serves two main purposes. One is to specify a factorial design. In the example above, parameters are specified not to vary with any factors. This is achieved in the `p.map` argument by associating all parameters with a `1` character. To vary a parameter over a factor, the `1` character should be replaced with the factor name. If one wants to add varying t_0 over the factor `cond` in the running example here, one enters `t0 = "cond"`. To vary a parameter over multiple variables, a vector should be supplied. For example, to vary t_0 over both `cond` and `trialtype` factors, by entering `t0 = c("cond", "trialtype")`. The other purpose of `BuildModel` is to specify a model likelihood. In this example, `BuildModel` specifies the likelihood function of diffusion model by entering `"rd"` (Ratcliff & McKoon, 2008) to the `type` argument.

`BuildModel` will print a parameter vector, `p.vector`, that shows the association between the model parameters and the experimental design, seen in Listing 1, lines 14 to 20. For example, following the previous multi-factor example where t_0 varies by `cond` and `trialtype`, the `p.vector` consists of nine parameters: `a`, `v`, `z`, `sz`, `sv`, `t0.cond1.type1`, `t0.cond2.type1`, `t0.cond1.type2`, and `t0.cond2.type2`. Note that the order of `p.vector` is important for the subsequent step in which we associate the model object with prior distributions.

```
Parameter vector names are: ( see attr(,
    "p.vector") )
[1] "a"  "v"  "z"  "sz" "sv" "t0"

Constants are (see attr(,"constants") ):
st0   d
0     0

Model type = rd
```

Next, the function `BuildPrior` sets up the association between the model parameters and their prior distributions, as is done in Listing 2. Table 3 lists the five main arguments in `BuildPrior`.

To visually check prior distributions, the prior object created by `BuildPrior` works with two base R functions, `print` and `plot` (note for this the `ggdmc` package must be loaded). Figure 3 shows the result when executing the `plot(p.prior)` command.

Sampling

Calling `BuildDMI` links the model with the data, creating a data model instance (i.e., DMI), `dmi <- BuildDMI(dat, model)`. After specifying this object, we are ready to draw posterior samples. Before sampling, we must decide the number of chains, the number of samples and the method of setting initial values for each Markov chain. By default, `ggdmc` chooses the number of

Listing 2 ■ Setting the association between the model parameters and the prior distribution for Example 1.

```
p.prior  <- BuildPrior(
  dists  = c(rep("tnorm", 2), "beta", "beta", "tnorm", "beta"),
  p1     = c(a = 1, v = 0, z = 1, sz = 1, sv = 1, t0 = 1),
  p2     = c(a = 1, v = 2, z = 1, sz = 1, sv = 1, t0 = 1),
  lower  = c(0, -5, NA, NA, 0, NA),
  upper  = c(5,  5, NA, NA, 5, NA))
```

Table 3 ■ Prior options.

Name	Symbol	Interpretation
Distributions	`dists`	The form of the prior distributions
Parameter 1	`p1`	The location parameters
Parameter 2	`p2`	The scale parameters
Lower bound	`lower`	The lower boundaries
Upper bound	`upper`	The upper boundaries

chains by multiplying the number of parameters by three, following Ter Braak's (2006) heuristics. Often, we save posterior samples at a regular interval rather than saving every interval (i.e., `thin`) to reduce memory requirements. The number of samples, together with the length of thin, are difficult to decide a priori so must be verified after sampling.

We initialize Markov chains by entering, `fit <- StartNewsamples(dmi, p.prior)`. The two arguments are the data model instance and the prior distributions. By default, the function attempts to locate the parameter space of the target distribution by using a mixture of crossover and migration operators (Hu & Tsui, 2010; Turner, Sederberg, Brown, & Steyvers, 2013). Next, we simply draw samples starting from the object created before by entering, `fit <- run(fit)`. The default `run` function draws 500 new samples and discards the 200 samples drawn previously. We chose the default setting of drawing 500 samples after 200 burn-in samples because our experience of fitting evidence-accumulation models shows the DE-MCMC sampler usually starts to draw valid samples in the parameter space reliably after 200 iterations in the type of simulation study shown here. The user can change the default behavior by entering, for example, `fit <- run(fit, nmc = 1000)` to draw 1000 samples or entering `add = TRUE` to keep the samples.

Results

Diagnoses

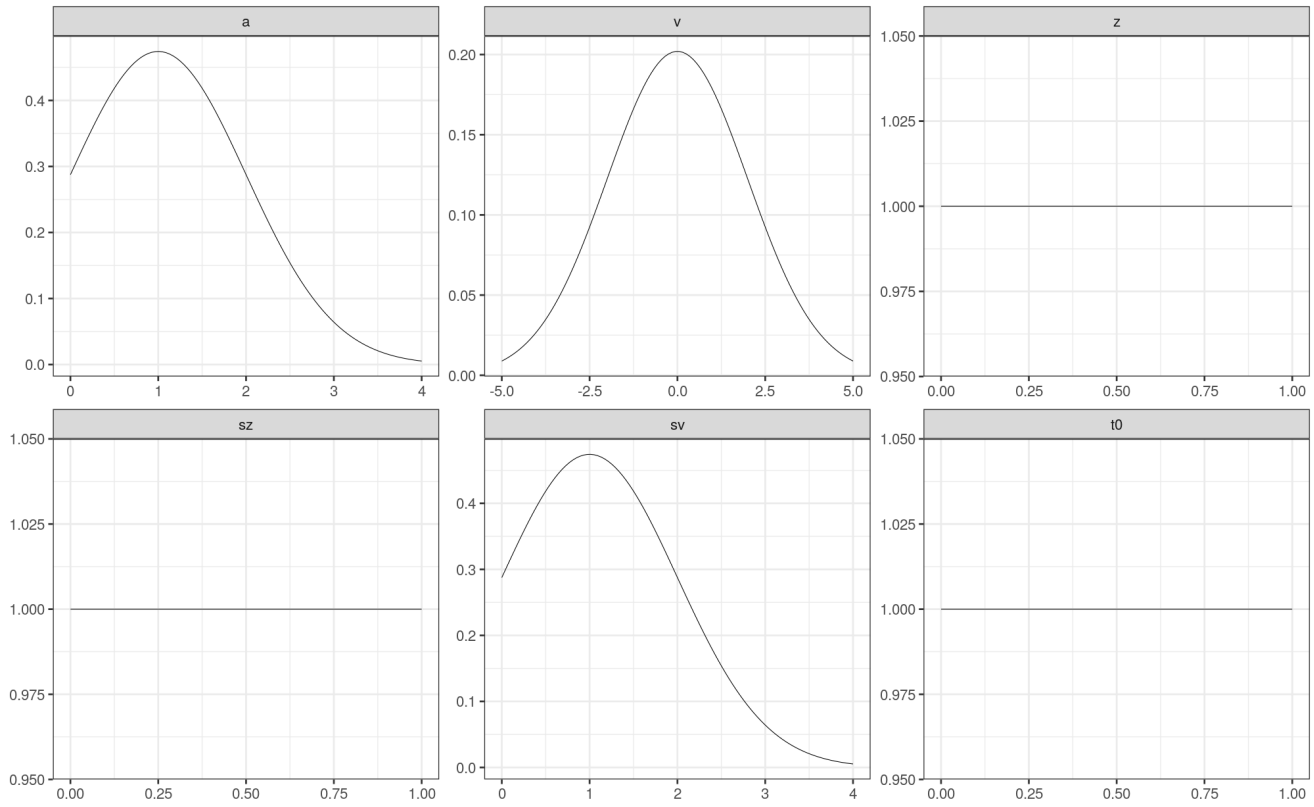
It is crucial to check (1) whether the MCMC process was successfully drawing samples from the target distribution and (2) whether the number of samples is sufficient. Two main methods for the diagnoses are to check the trace and probability density plots visually and to calculate diagnosis statistics. The upper panel in Figure 4 shows the trace plots of the first and the second sample objects. The upper left panel shows the MCMC chains moving from their start point towards what looks like the target distribution, reaching there around the 150th sample. The right panel shows the Markov chains are well-mixed. The lower panel in Figure 4 shows the posterior distributions of each parameter. Potential scale reduction factor (PSRF) diagnoses whether Markov chains are converged by calculating the ratio of the within-chain variance to the between-chain variance. When the ratio deviates drastically from 1, PSRF suggests a failure to converge. It is argued that Markov chains mix well when their PSRF is less than 1.1 (Brooks & Gelman, 1998).

```
gelman(fit)
Potential scale reduction factors:
   Point est. Upper C.I.
a        1.04       1.06
v        1.03       1.04
z        1.02       1.03
sz       1.04       1.06
sv       1.05       1.08
t0       1.05       1.08

Multivariate psrf: 1.07
```

Once successful posterior sampling is completed, we should examine whether the model is able to fit to the observed data with the sampled parameters. This can be done by posterior-predictive simulation, which uses the posterior samples to simulate data and compare them against the real data (recall that in this example, the 'real' data was simulated earlier). Figure 5 illustrates probabil-

Figure 3 ■ An example of prior distributions.



ity density plots from 100 posterior simulations against the 'real' fitted data. It shows that the model accounts for the correct RT distributions in the s1 and s2 condition satisfactorily. It is slightly off in fitting the error RT distribution in the s2 condition, and more substantially off in fitting errors in the s1 condition. This model mis-fit likely occurs due to the small number of error responses in the s1 condition.

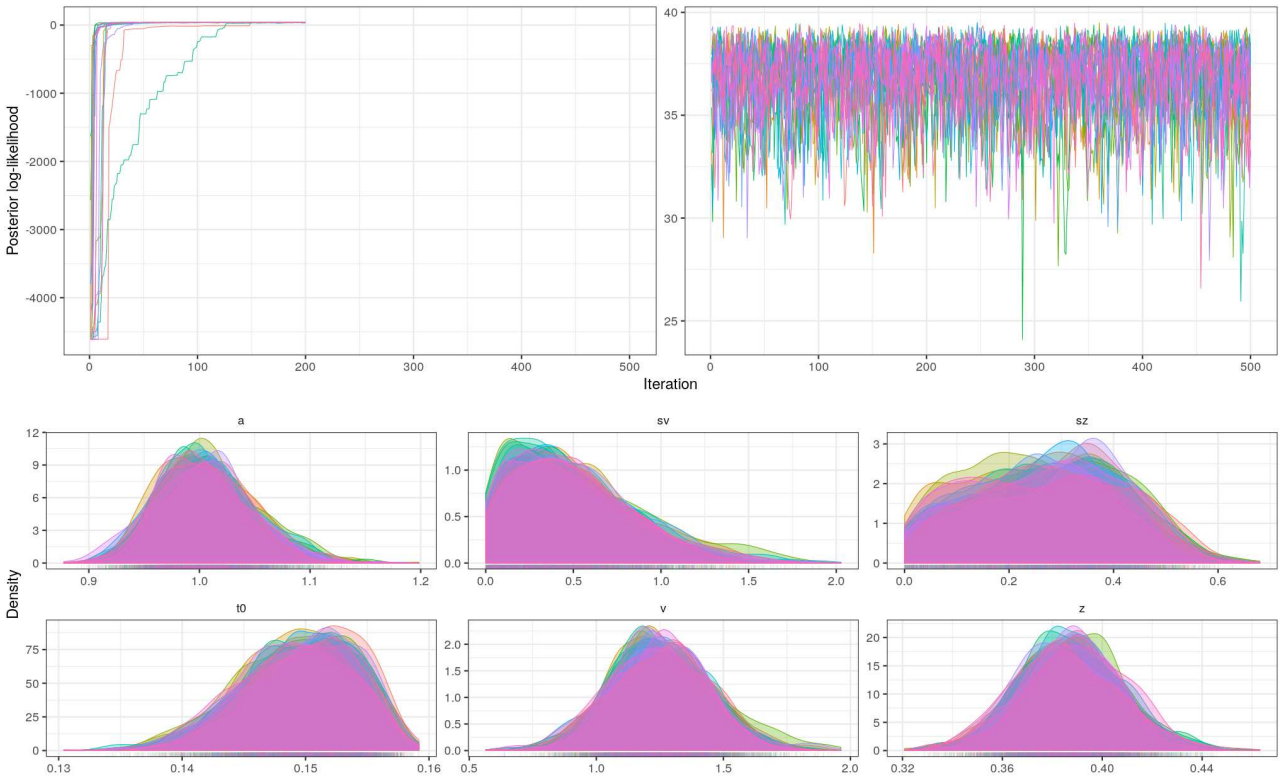
We now evaluate how successful the described model fitting procedures were in recovering the 'true' parameters generating the synthetic data. This can be done conveniently by executing the, `summary` function (Table 4). We call the function with the argument, `recovery` set to `TRUE` to check how closely the model recovered the true parameters. As previously found, the diffusion parameters corresponding to drift rate variability are difficult to estimate (Voss & Voss, 2007). Besides this, parameter recovery was satisfactory. Before analyzing real data, it is critical to perform parameter recovery studies like we have described to assure that the estimation properties of the model and data are adequate for the chosen research question.

```
est <- summary(sam, recovery = TRUE,
        ps = p.vector, verbose = TRUE)
                  a    sv   sz   t0   v    z
True              1.00 0.20 0.25 0.15 1.20 0.38
2.5% Estimate     0.93 0.02 0.02 0.14 0.89 0.35
50% Estimate      1.00 0.43 0.27 0.15 1.25 0.39
97.5% Estimate    1.09 1.30 0.52 0.16 1.64 0.43
Median-True       0.00 0.23 0.02 0.00 0.05 0.01
```

Example 2: Fitting hierarchical models

Having performed a parameter recovery study on a simple data set, we now proceed to discuss how `ggdmc` could apply to a more realistic example. We analyze the data of Pleskac et al.'s (2018) shooting decision studies. We examine these data for two reasons. Firstly, we aim to demonstrate that hierarchical evidence accumulation modeling can apply to a validated paradigm in social psychology. Secondly, we aim to test the robustness of hierarchical modeling on data sets with per-condition observations as

Figure 4 ■ Trace and probability density plots. Note the difference in the y axes. Different colors represent samples from different chains.



small as eight. This will cast light on the utility of hierarchical modeling when the number of observations is smaller than recommended practice for modeling individual subjects (Ratcliff & Tuerlinckx, 2002; Voss, Voss, & Klauer, 2010). This example also illustrates how quantitative models may inform issues of social importance.

Methods

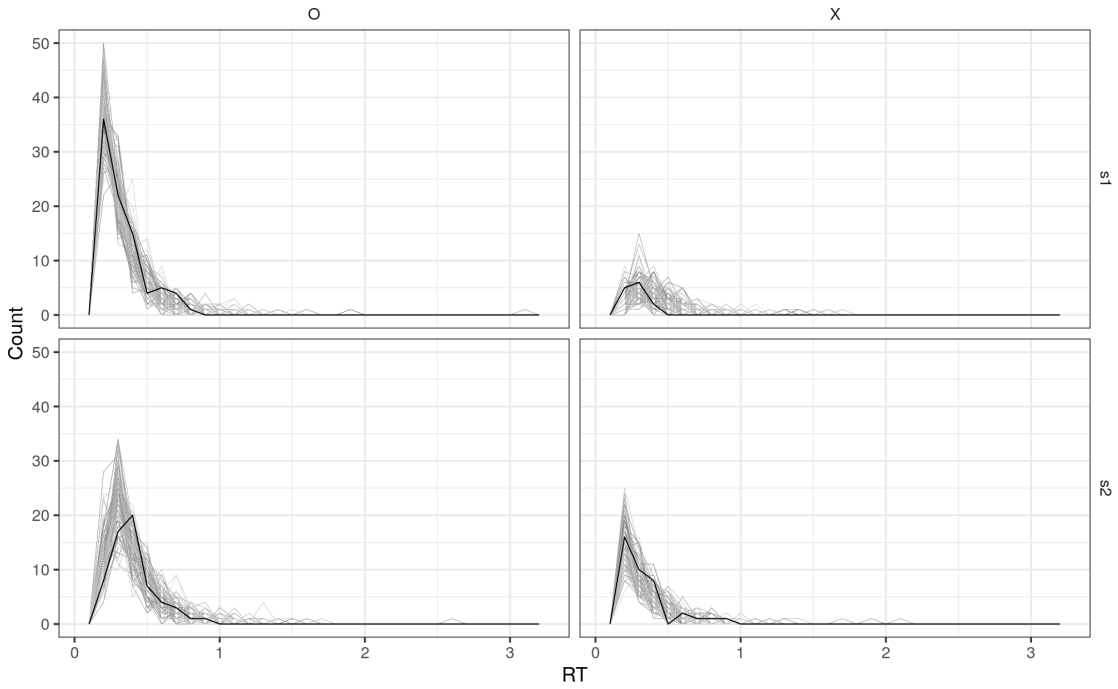
The Paradigm

The shooting-decision paradigm investigates shooting decisions in a first-person shooter task (FPST) (Correll, Park, Judd, & Wittenbrink, 2002). The computerized task displays pictures of a target holding a gun or another object. Participants submit a response to choose whether to shoot or not shoot at the target. They are asked to shoot if the target holds a gun, and otherwise not to shoot. This paradigm can be used to test the influence of many factors on shooting decisions, including dangerous vs. safe neighborhoods, clear vs. blurred gun images, and the role of the race of the target. Previous analyses with signal detection theory indicate that black targets resulted in lower decision thresholds than white targets (Correll et al., 2002; Correll, Wittenbrink, Park, Judd, & Goyle, 2011). However, signal detection theory accounts only for response choices, and not RTs. Recently, Pleskac et al. (2018) applied the hierarchical Wiener diffusion model to perform a more comprehensive analysis of shooting decisions. Their study yielded three major results: (1) that boundary separation to shoot at black targets is larger than to shoot at white targets; (2) that drift rate is higher towards armed black than armed white targets; and (3) that non-decision times are longer for non-gun targets than gun targets.

Data sets

The data sets were downloaded from the OSF site, https://osf.io/9qku5/ (Pleskac et al., 2018). Four different FPST studies were examined (56, 116, 38, & 108 participants). All tasks examined included safe neighborhoods and clear object views. The minimum numbers of observation per design cell in each data set were 19, 8, 13, and 40. Such trial numbers, although less than desirable, are typical of many

Figure 5 ■ The post-predictive probability density plots. O and X stand for the correct and incorrect responses. The two stimulus conditions are presented separately on the s1 and s2 rows (right y axis). Each grey line draws one posterior predictions based on the parameter estimates. The black line draws the data.



applied psychology experiments.

Model

The Wiener diffusion model can be derived from the full diffusion model by fixing the between-trial variability of the drift rate and the starting point at zero. The model has four free parameters, a, v, z, and t_0 (Table 1). The experimental design includes two factors, race of the target (black vs. white) and stimulus type (gun vs. non-gun). There are two response choices (shoot vs. don't shoot). We demonstrate how the model is specified with the `BuildModel` function in the code of Listing 3, lines 1 to 8. We denote the experimental factors as uppercase, RACE for black vs. white target, and S for gun vs. non-gun targets. Following Pleskac et al. (2018), we allow threshold parameters (z and a) to vary across decisions to respond to black and white targets. Note that conventionally, z and a (and equivalent threshold parameters in other models) are assumed not to vary over trial-level manipulations. However, Pleskac et al. (2018) make arguments for breaking this convention in this instance. Perhaps most notably, varying z and a by trial type is necessary to compare the evidence accumulation modeling to previous signal detection theory analyses. The drift rate (v) can vary by race and stimulus type, as can

Table 4 ■ The arguments of summary function.

Name	Symbol	Interpretation
Posterior samples	`object`	An R list storing posterior samples
Boolean switch for a recovery study	`recovery`	A Boolean switch for summarizing either a recovery or an estimation study
True parameter	`ps`	The true parameter vector for summarizing a recovery study
Boolean switch for information printing	`verbose`	A Boolean switch for printing either short or long form of information

non-decision time. Thus, there are twelve parameters: two boundary separations, four drift rates, two starting points, and four non-decision times. The parameters are reported by `BuildModel` and stored in the `p.vector` attribute of the model object.

After specifying the base model, we must set up one set of base-level prior distributions and one set of hyper-level distributions. The latter endows the model with a hierarchical structure, assuming participants are drawn from a population. The hierarchical structure of model matches that of the data, allowing one to directly estimate the population-level parameters. In the following example, we will assume that model parameters follow a normal distribution in the population and attempt to estimate the mean and standard deviation of those distributions. It is important to understand that the parameters defining the population-level distributions (often referred to as *hyper-parameters*) are also estimated in a Bayesian manner, and thus are each associated with their own prior and posterior distributions. Below we go through the process of setting up priors and hyper-priors in `ggdmc`. The code sets up some convenient variables which will be used for setting values for both base-level and hyper-level prior distributions. As an aside, note that the prior values we opted for here do not bear special significance, but only selected based on the parameter recovery study mimicking the numbers of participants and observations in the original study 1. In certain occasions, it may be beneficial to choose priors based on previous research and / or theory. To determine the extent of prior influence over subsequent results, it can be important to check whether the posterior distributions substantially change when prior values are varied. The relevant instructions are shown in Listing 3, lines 17 to 22.

To minimize possible human errors of entering the parameter names, we use a convenient function, `GetPNames` to retrieve the parameter names and the parameter number. The next lines specify values that we will use to define the hyper-level prior distributions. The `pop.loc` and `pop.scale` will be used to the mean and the standard deviation of a normal distribution, respectively. The last two lines associate the means and standard deviations with the parameter names. We select these values based on parameter-recovery studies resembling to the data sets. As in the single participant example, we use `BuildPrior` to set up the base-level prior distributions. See Listing 3, lines 24 to 29.

Next, we set up the hyper-level prior distributions by building one set for the location priors and the other for the scale priors (Listing 3, lines 31 to 42). In this case the priors for the population mean values are identical to the priors for individual's participant values. The prior for the population standard deviations follow a uniform distribution with a lower bound of 0 and an upper bound of 2. By default, the `beta` type prior sets a lower bound at 0 when no values are specified.

The line of code below binds the base-level and the hyper-level priors together as one object, which will informs the software to fit a hierarchical model:

```
priors <- list(pprior=p.prior,
               location=mu.prior,
               scale=sigma.prior)
```

In summary, the hierarchical model estimates 24 hyper-level parameters, twelve for the location parameter and twelve for the scale parameters in each of the four data sets. In addition, the model estimates twelve individual-level parameters for each participant.

Sampling

The sampling step is the same as the previous example. In the case of empirical data, we first load the preprocessed and appropriately formatted data. Then we associate the data with the model, building a data model instance.

```
load("data/study1.rda")
dmi <- BuildDMI(data.frame(d), model)
```

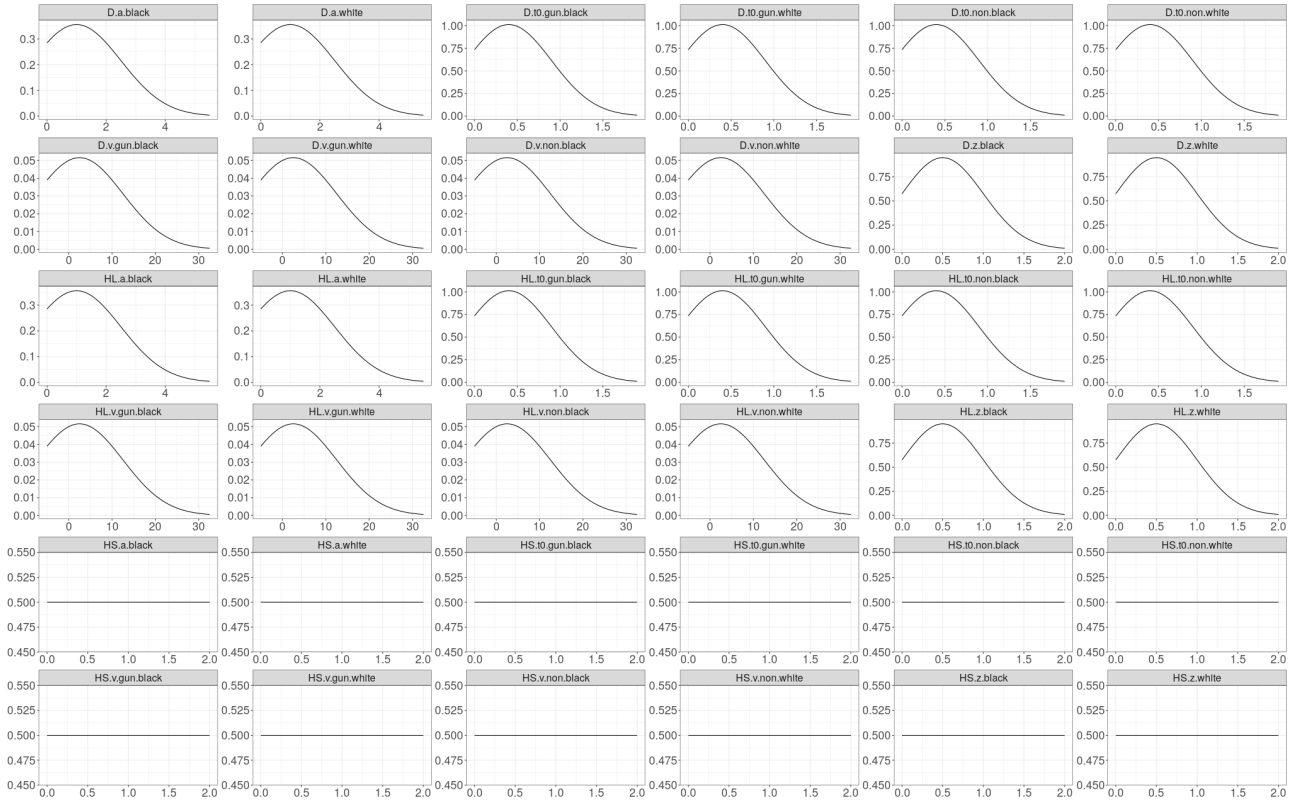
After building the data model instance, we initialize samples as before, `fit <- StartNewsamples(dmi, priors, thin=8)`. We set the argument `thin` to eight to store samples every 8th step. The function, `StartNewsamples` recognizes when the prior object carries both the hyper-level and base-level priors and proceeds with hierarchical modeling. By default, the function initializes and draws 200 samples. In this case, we discard the initial 200 samples as a *burn-in* period (in which the sampler moved from the start points to the target density). We then redraw another 500 samples, `fit<- run(fit, thin=8)`.

Results

Diagnoses

It can be laborious to diagnose the sampling of high-dimensional hierarchical models. Here we use the function, `hgelman`, to calculate PSRFs of the hyper parameters and the parameters of individual participants. The `hyper` label indicates the averaged PSRF across the hyper parameters and the rest are the averaged PSRFs for the parameters for each individual participant (the labels are names for individual participants).

Figure 6 ■ Prior distributions. The base-level priors are shown under the strip titled, `D.a.black`, `D.a.white`, ..., `D.z.white`. `D` stands for the data-level priors. The location and the scale parameters of the hyper-level priors are represented by the HL and the HS, respectively. The x axis shows the value of the parameters and the y axis shows the probability densities.




```
hats <- hgelman(hsam, verbose = TRUE)
## hyper   55   47    1   30   54   40   12   11   38    9
##   1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
##     7   45   39   14   56   22   17   16    5   25   44
##   1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
##    49   52   42   36   41    4   51   29   35   53    2
##   1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
##    24   48   18   37   27   10    8   21   46   34   23
##   1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
##     6   50   32   20   28   15   19   31   13   26    3
##   1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
##    43   33
##   1.0  1.1
```

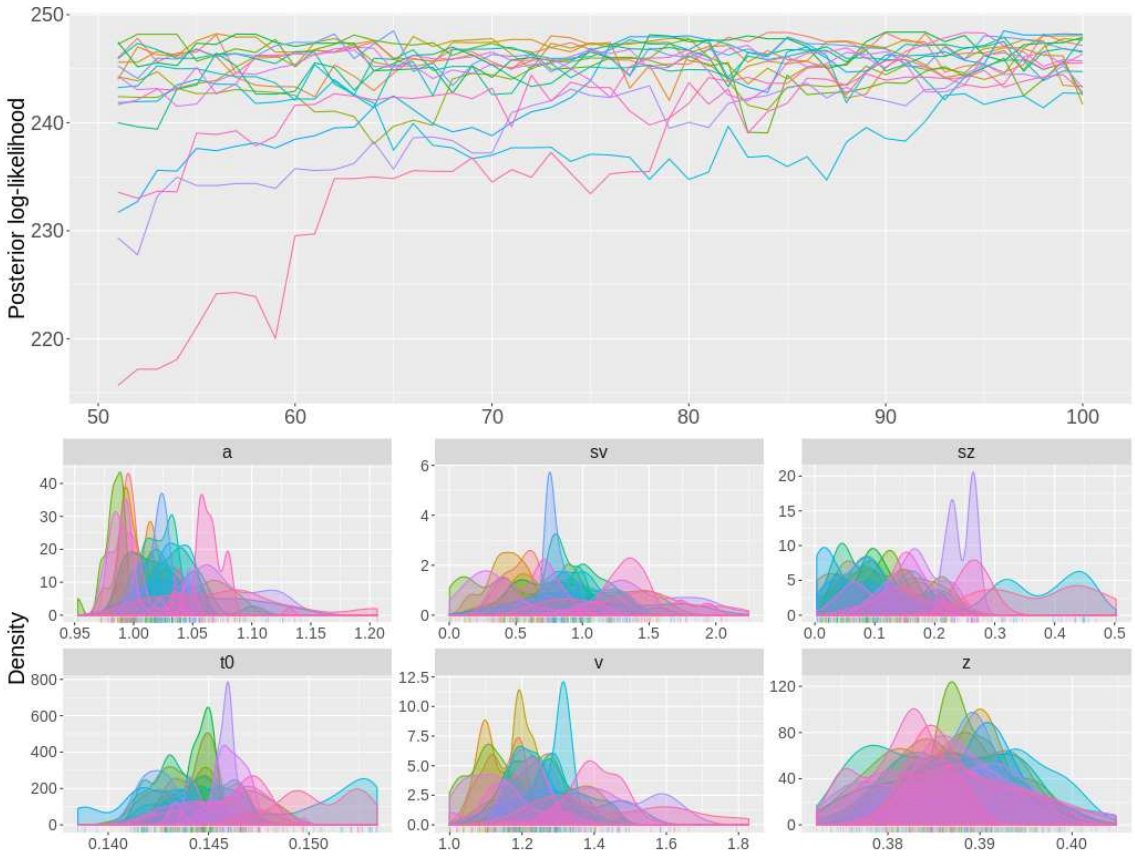
We looked for the values greater than 1.1, suggesting that some chains probably did not converge. One could also perform visual checks of the trace and probability density plots. To illustrate how Markov chains can look when they are not converged, we show early sampling stages for a participant from the study with unmixed trace and probability density plots (Figure 7), as well as the converged result (Figure 8).

We applied the sampling procedure above to all of Pleskac et al.'s (2018) data sets. Below we report the results of the behavioral and the model-based analyses.

Behavioral and Model-based Analyses

Firstly, we examine the behavioral patterns of the averaged RT and error rate data, summarized in Figure 9. Unsurprisingly, the data summaries are consistent with Pleskac et al. (2018). Then we report summaries of the hyper-parameter distributions in Table 6 and 7, and present a visual comparison in Figure 10. Our results are squarely in line with Pleskac et al. (2018). We found drift rate differences depending on the race of the target in study 1 and 4. Drift rates towards shooting black targets with guns were higher than towards gun-holding white targets, suggesting an effect of racial stereotyping on information processing. There was also a credible difference

Figure 7 ■ An example showing chains not converged.



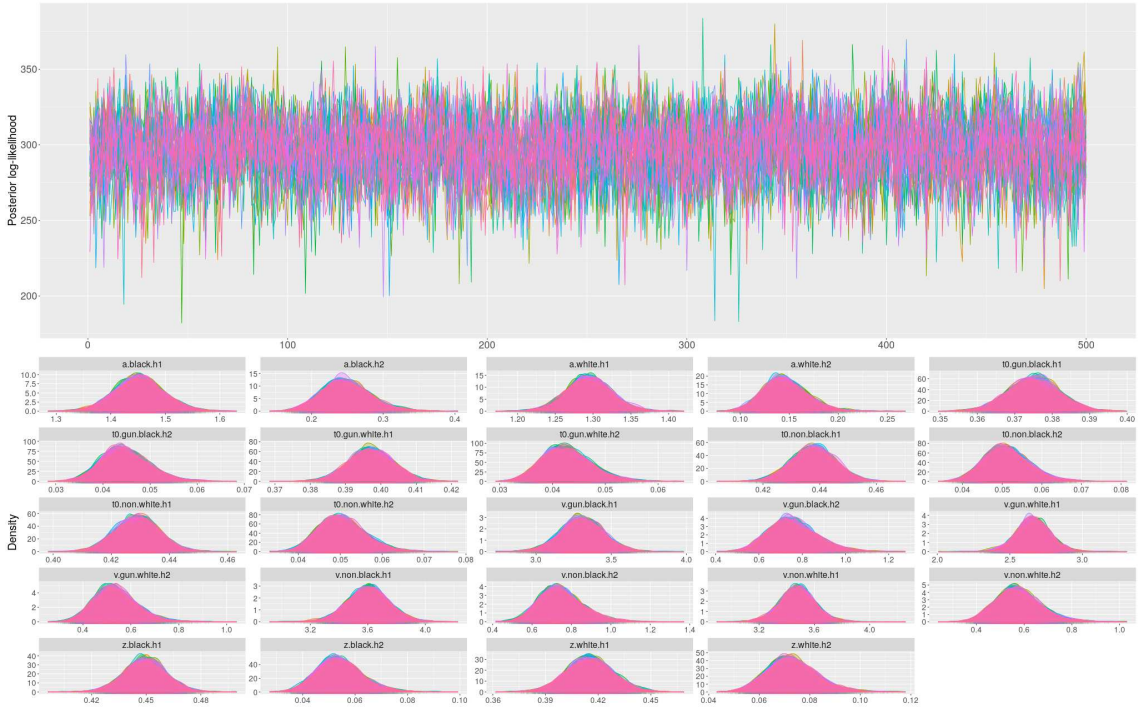
in boundary separation by race of the target in study 1 and 2, with the criterion to shoot at black targets being larger than to shoot at white targets. Pleskac et al. (2018) argued this suggested that people are aware of the racial bias in information processing towards black targets; and therefore, adjusting their decision boundary, attempting to compensate for such bias. Non-gun objects resulted in longer non-decision times than gun objects in study 1 and 2. Pleskac et al. (2018) speculated that this might occur due to differences in stimulus encoding, as the non-gun objects are more variable in terms of their anticipated shape and size than the gun objects. Also replicating Pleskac and colleagues' composite analysis, we found that starting points were not affected by the race of the target.

Discussion

In this paper, we provided a hands-on tutorial for using the `ggdmc` R package to fit hierarchical Bayesian evidence accumulation models. Until recently, Bayesian accumulation modeling was a specialized skill used only by experts, either assisted by high performance computers or by specific Bayesian programming languages. The potential applications of evidence accumulation modeling are vast. We demonstrated this by using `ggdmc` to estimate the latent cognitive processes underlying human decision making in a simulation of a safety-critical scenario.

The `ggdmc` package is ready to assist applied researchers to exploit the advantages of hierarchical evidence accumulation modeling. Our software is built upon an earlier suite of R functions, Dynamic Models of Choice (Heathcote et al., 2019). We have provided a convenient interface to incorporate experimental designs. This is especially crucial for psychological researchers, because psychology has a long tradition of using novel experimental paradigms, such as first-person shooter and self-tagging tasks (Correll et al., 2002; Sui, He, & Humphreys, 2012), and factorial designs to tackle complex questions. Extending previous work, we have greatly eased computational difficulties with Bayesian modeling by coding an efficient, intelligent sampling algorithm, population-based MCMC (pM-

Figure 8 ■ Posterior trace and probability density plots.



CMC) in C++, which incorporates DE-MC (Ter Braak, 2006; Turner, Sederberg, et al., 2013) and a genetic algorithm (Hu & Tsui, 2005; Tanese, 1989). By implementing a design interface with efficient Bayesian MCMC in the R programming language, `ggdmc` enables psychology researchers to efficiently model data sets of interest to them.

The reader should note that although we have focused on using `ggdmc` to apply the LBA and diffusion models, it potentially has a much wider application for implementing novel cognitive models. For example, we have recently fitted the piecewise LBA model (Holmes et al., 2016) with `ggdmc` to explore the method of using massive parallel computation in likelihood simulations. This example shows `ggdmc` can accommodate cognitive models without analytic likelihood functions (Holmes, 2015; Lin, Heathcote, & Holmes, 2019; Turner & Sederberg, 2012). When this combined with approximate Bayesian computation (Beaumont, Zhang, & Balding, 2002), it can empower researchers to estimate model parameters with only process descriptions in hand.

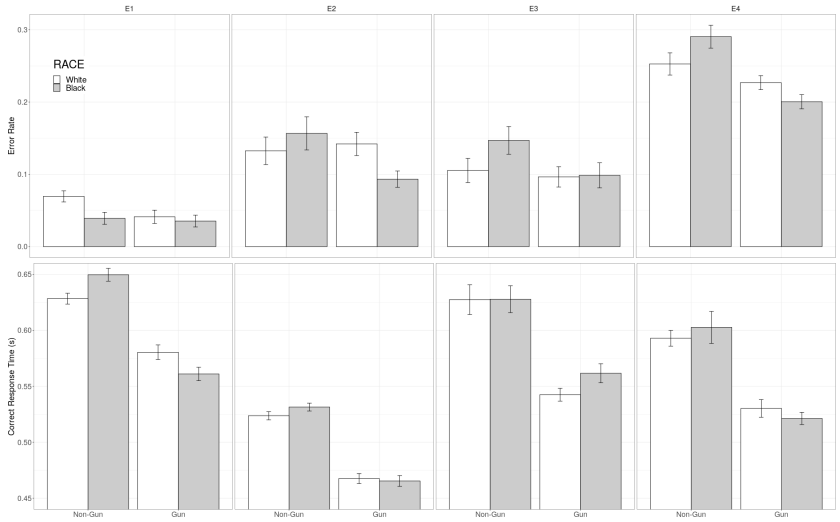
Alternative Solutions

Other software for hierarchical Bayesian modeling includes Bayesian programming languages, most notably,

Table 5 ■ The key arguments of the plot function.

Name	Symbol	Interpretation
Posterior samples	`x`	The posterior samples from model fits
Hyper parameters	`hyper`	A Boolean switch to draw hyper parameters (TRUE) or the parameters of individual participants (FALSE)
Density	`den`	A Boolean switch to draw the probability density plots (TRUE) or the trace plots (FALSE)
Posterior log-likelihood	`pll`	A Boolean switch to draw the trace plot of the posterior log-likelihoods (TRUE) or that of the marginal log-likelihoods (FALSE)

Figure 9 ■ Error rates and correct response times for the four studies reported in Pleskac, Cesario, and Johnson (2018). E1, E2, E3, and E4 represent experiment 1, 2, 3 and 4.



BUGS (Lunn et al., 2009), JAGS (Plummer, 2003), and Stan (Carpenter et al., 2017). However, they are designed for standard statistical models, limiting their applications on certain cognitive models (Lee & Wagenmakers, 2015). Recent efforts to overcome such limitations have created useful tools built on top of these specialized Bayesian languages (Ahn et al., 2017; Annis et al., 2017; Cox & Criss, 2017; Pleskac et al., 2018). However, such efforts do not provide convenient frameworks for flexibly specifying complex factorial designs. This renders it necessary to hand-code the specialized Bayesian languages to accommodate different factorial designs. Assigning various model parameters to experimental design cells can be complex and laborious for many realistic examples. A Python-based tool (Wiecki et al., 2013) has been offered as an alternative to specialized Bayesian languages. However, this package comes ready for the user with only the Wiener diffusion model, and substantial Python expertise is required to adapt it to fit the full diffusion model or LBA. In summary, `ggdmc`'s advantage to experimental and applied researchers over alternatives arises from its combination of in-built likelihood functions for popular accumulation models, framework for flexibility of accommodating model variants and factorial designs, and efficient sampler written in C++.

Limitations

Like other Bayesian software that can handle high-dimensional models (Carpenter et al., 2017), to attain efficient and accurate computation, `ggdmc` must code the likelihood functions in the C++ language. Thus, at present, implementing the likelihood functions of new cognitive models outside of those we have provided requires C++ programming knowledge. However, we do strive to continue to add new available likelihood functions to `ggdmc`. Although `ggdmc` is designed to be a generic tool, it is unlikely to resolve all modeling problems. Managing Bayesian posterior sampling requires expertise and judgement, particularly when fitting new models. For fitting new models, we recommend investigating the correlation structure between model parameter estimates, experimenting with differing burn-in lengths and posterior draws to identify target distributions, and running parameter recovery studies to determine whether trial numbers are sufficient to identify parameters values.

Authors' note

The authors are grateful for the constructive comments from Andrew Heathcote and his time for testing early version of the software. We also thank Angus Reynold for his suggestion regarding the preparation of the manuscript and Dr. Pleskac's permission to use their data. The stable version of software can be downloaded from https://cran.r-project.org/web/packages/ggdmc/index.html. The latest updates of the software can be found on GitHub, https://github.com/yxlin/ggdmc.

Table 6 ■ The estimations of the location parameter estimation in Study 1 (E1) and Study 2 (E2).

	E1			E2		
	2.5%	50%	97.5%	2.5%	50%	97.5%
Black a	1.37	1.45	1.54	0.96	1.00	1.05
White a	1.24	1.29	1.35	0.88	0.92	0.96
Gun, black v	3.03	3.30	3.58	2.13	2.44	2.75
Gun, white v	2.44	2.66	2.88	1.88	2.21	2.55
Non-gun black v	3.32	3.60	3.88	2.44	2.88	3.32
Non-gun white v	3.24	3.48	3.72	2.39	2.76	3.13
Black z	0.43	0.45	0.47	0.39	0.42	0.45
White z	0.39	0.41	0.44	0.42	0.45	0.47
Gun, black t0	0.36	0.38	0.39	0.31	0.33	0.34
Non-gun, black t0	0.43	0.44	0.45	0.33	0.34	0.35
Gun, white t0	0.39	0.40	0.41	0.35	0.36	0.38
Non-gun, black t0	0.41	0.43	0.44	0.35	0.37	0.38

Table 7 ■ The estimations of the location parameter estimation in Study 3 (E3) and Study 4 (E4).

	E1			E2		
	2.5%	50%	97.5%	2.5%	50%	97.5%
Black a	1.07	1.13	1.19	0.90	0.93	0.96
White a	1.05	1.10	1.16	0.96	0.99	1.02
Gun, black v	1.95	2.33	2.72	1.32	1.47	1.62
Gun, white v	2.21	2.56	2.90	1.05	1.20	1.34
Non-gun black v	1.92	2.30	2.68	1.13	1.32	1.52
Non-gun white v	2.14	2.54	2.94	1.41	1.61	1.81
Black z	0.41	0.43	0.46	0.43	0.44	0.46
White z	0.43	0.46	0.50	0.42	0.43	0.45
Gun, black t0	0.37	0.39	0.40	0.34	0.35	0.37
Non-gun, black t0	0.37	0.38	0.40	0.34	0.36	0.37
Gun, white t0	0.37	0.39	0.42	0.33	0.34	0.36
Non-gun, black t0	0.38	0.41	0.44	0.34	0.36	0.37

References

Ahn, W.-Y., Haines, N., & Zhang, L. (2017). Revealing neurocomputational mechanisms of reinforcement learning and decision-making with the hbayesdm package. *Computational Psychiatry*, *1*, 24–57. doi:10.1162/CPSY_a_00002

Annis, J., Miller, B. J., & Palmeri, T. J. (2017). Bayesian inference with stan: A tutorial on adding custom distributions. *Behavior Research Methods*, *49*(3), 863–886. doi:10.3758/s13428-016-0746-9

Bayes, T., Price, R., & Canton, J. (1763). *An essay towards solving a problem in the doctrine of chances*. London: Philosophical transactions of the Royal Society.

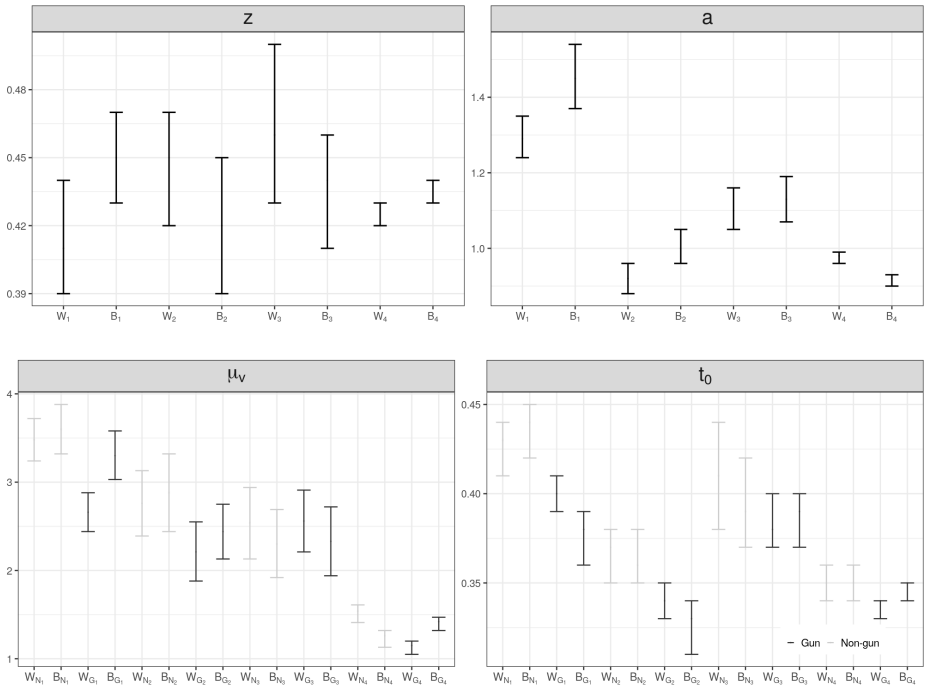
Beaumont, M. A., Zhang, W., & Balding, D. J. (2002). Approximate Bayesian computation in population genetics. *Genetics*, *162*(4), 2025–2035.

Boehm, U., Marsman, M., Matzke, D., & Wagenmakers, E.-J. (2018). On the importance of avoiding shortcuts in applying cognitive models to hierarchical data. *Behavior Research Methods*, *50*(4), 1614–1631. doi:10.3758/s13428-018-1054-3

Brooks, S. P., & Gelman, A. (1998). General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, *7*(4), 434–455. doi:10.2307/1390675

Brown, S., & Heathcote, A. (2008). The simplest complete model of choice response time: Linear ballistic accumulation. *Cognitive Psychology*, *57*(3), 153–178. doi:10.1016/j.cogpsych.2007.12.002

Figure 10 ■ The group-level parameter estimates of the Wiener diffusion model for each condition. The gray and dark lines are to highlight non-gun vs. gun objects (see the legend at the lower right corner). μv and t0 stands for drift rate and non-decision time. The error bars show 95% Bayesian credible intervals calculated from the posterior distributions.



Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., … Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*, 1–19. doi:10.18637/jss.v076.i01

Correll, J., Park, B., Judd, C. M., & Wittenbrink, B. (2002). The police officer's dilemma: Using ethnicity to disambiguate potentially threatening individuals. *Journal of Personality and Social Psychology*, *83*(6), 1314–1328. doi:10.1037/0022-3514.83.6.1314

Correll, J., Wittenbrink, B., Park, B., Judd, C. M., & Goyle, A. (2011). Dangerous enough: Moderating racial bias with contextual threat cues. *Journal of Experimental Social Psychology*, *47*(1), 184–189. doi:10.1016/j.jesp.2010.08.017

Cox, G. E., & Criss, A. H. (2017). Parallel interactive retrieval of item and associative information from event memory. *Cognitive Psychology*, *97*, 31–61. doi:10.1016/j.cogpsych.2017.05.004

Farrell, S., & Lewandowsky, S. (2018). *Computational modeling of cognition and behavior*. Cambridge, MA: Cambridge University Press.

Fisher, R. (1920). A mathematical examination of the methods of determining the accuracy of an observation etc monthly notices roy. *Monthly Notices of the Royal Astronomical Society*, *80*, 758–770.

Forstmann, B. U., Brown, S., Dutilh, G., Neumann, J., & Wagenmakers, E.-J. (2010). The neural substrate of prior information in perceptual decision making: A model-based analysis. *Frontiers in Human Neuroscience*, *4*, 4–44. doi:10.3389/fnhum.2010.00040

Forstmann, B. U., & Wagenmakers, E.-J. (2016). *Introduction to model-based cognitive neuroscience*. New York: Springer.

Gelman, A., Stern, H. S., Carlin, J. B., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis*. Raton Choco: Chapman and Hall/CRC.

Hastings, W. K. (1970). Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, *57*(1), 97–109.

Heathcote, A., Lin, Y.-S., Reynolds, A., Strickland, L., Gretton, M., & Matzke, D. (2019). Dynamic models of choice. *Behavior Research Methods*, *51*, 961–985. doi:10.3758/s13428-018-1067-y

Holmes, W. R. (2015). A practical guide to the probability density approximation (pda) with improved implementation and error characterization. *Journal of Mathematical Psychology*, *69*, 13–24. doi:10.1016/j.jmp.2015.08.006

Holmes, W. R., & Trueblood, J. S. (2018). Bayesian analysis of the piecewise diffusion decision model. *Behavior Research Methods*, *50*(2), 730–743. doi:10.3758/s13428-017-0901-y

Holmes, W. R., Trueblood, J. S., & Heathcote, A. (2016). A new framework for modeling decisions about changing information: The piecewise linear ballistic accumulator model. *Cognitive Psychology*, *85*, 1–29. doi:10.1016/j.cogpsych.2015.11.002

Hu, B., & Tsui, K.-W. (2005). *Distributed evolutionary monte carlo with applications to bayesian analysis* (tech. rep. No. 1112). Retrieved from https://pdfs.semanticscholar.org/9bb5/69efc61bb77739c446561ca2ace155cd8168.pdf

Hu, B., & Tsui, K.-W. (2010). Distributed evolutionary monte carlo for Bayesian computing. *Computational Statistics & Data Analysis*, *54*(3), 688–697. doi:10.1016/j.csda.2008.10.025

Lee, M. D., & Wagenmakers, E.-J. (2015). *Bayesian cognitive modeling: A practical course*. Cambridge: Cambridge University Press.

Lin, Y.-S., Heathcote, A., & Holmes, W. R. (2019). *Parallel probability density approximation*. doi:10.3758/s13428-018-1153-1

Lunn, D. J., Spiegelhalter, D., Thomas, A., & Best, N. (2009). The bugs project: Evolution, critique and future directions. *Statistics in Medicine*, *28*(25), 3049–3067. doi:10.1002/sim.3680

Miletic, S., Turner, B. M., Forstmann, B. U., & van Maanen, L. (2017). Parameter recovery for the leaky competing accumulator model. *Journal of Mathematical Psychology*, *76*, 25–50. doi:10.1016/j.jmp.2016.12.001

Myung, I. J. (2003). Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, *47*(1), 90–100. doi:10.1016/S0022-2496(02)00028-7

Navarro, D. J., & Fuss, I. G. (2009). Fast and accurate calculations for first-passage times in wiener diffusion models. *Journal of Mathematical Psychology*, *53*(4), 222–230. doi:10.1016/j.jmp.2009.02.003

Nunez, M. D., Vandekerckhove, J., & Srinivasan, R. (2017). How attention influences perceptual decision making: Single-trial eeg correlates of drift-diffusion model parameters. *Journal of Mathematical Psychology*, *76*, 117–130. doi:10.1016/j.jmp.2016.03.003

Pleskac, T. J., Cesario, J., & Johnson, D. J. (2018). How race affects evidence accumulation during the decision to shoot. *Psychonomic Bulletin & Review*, *25*(4), 1301–1330. doi:10.3758/s13423-017-1369-6

Plummer, M. (2003). Jags: A program for analysis of Bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing* (pp. 124–125). Vienna.

R Core Team. (2018). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. Vienna, Austria. Retrieved from https://www.R-project.org/

Ratcliff, R. (1978). A theory of memory retrieval. *Psychological Review*, *85*(2), 59–108. doi:10.1037/0033-295X.85.2.59

Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: Theory and data for two-choice decision tasks. *Neural Computation*, *20*(4), 873–922. doi:10.1162/neco.2008.12-06-420

Ratcliff, R., Philiastides, M. G., & Sajda, P. (2009). Quality of evidence for perceptual decision making is indexed by trial-to-trial variability of the eeg. *Proceedings of the National Academy of Sciences*, *106*(16), 6539–6544. doi:10.1073/pnas.0812589106

Ratcliff, R., Sederberg, P. B., Smith, T. A., & Childers, R. (2016). A single trial analysis of eeg in recognition memory: Tracking the neural correlates of memory strength. *Neuropsychologia*, *93*, 128–141. doi:10.1016/j.neuropsychologia.2016.09.026

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*(3), 438–481. doi:10.3758/BF03196302

Strickland, L., Loft, S., Remington, R. W., & Heathcote, A. (2018). Racing to remember: A theory of decision control in event-based prospective memory. *Psychological Review*, *125*, 851–887.

Sui, J., He, X., & Humphreys, G. W. (2012). Perceptual effects of social salience: Evidence from self-prioritization effects on perceptual matching. *Journal of Experimental Psychology: Human Perception and Performance*, *38*(5), 1105–1120. doi:10.1037/a0029792

Tanese, R. (1989). Distributed genetic algorithms. In *International conference on genetic algorithms*, Washington, DC. Retrieved from https://www.semanticscholar.org/paper/Distributed-Genetic-Algorithms-Tanese/c6316a32ad8cce67fb4b64ffc3e4b3354c8ec431

Ter Braak, C. J. F. (2006). A Markov chain monte carlo version of the genetic algorithm differential evolution: Easy Bayesian computing for real parameter spaces. *Statistics and Computing*, *16*(3), 239–249. doi:10.1007/s11222-006-8769-1

Turner, B. M., Dennis, S., & Van Zandt, T. (2013). Likelihood-free Bayesian analysis of memory models. *Psychological Review*, *120*(3), 667–678. doi:10.1037/a0032458

Turner, B. M., & Sederberg, P. B. (2012). Approximate Bayesian computation with differential evolution. *Journal of Mathematical Psychology*, *56*(5), 375–385. doi:10.1016/j.jmp.2012.06.004

Turner, B. M., Sederberg, P. B., Brown, S. D., & Steyvers, M. (2013). A method for efficiently sampling from distributions with correlated dimensions. *Psychological Methods*, *18*(3), 368–384. doi:10.1037/a0032222

Vandekerckhove, J., & Tuerlinckx, F. (2007). Fitting the ratcliff diffusion model to experimental data. *Psychonomic Bulletin & Review*, *14*(6), 1011–1026. doi:10.3758/BF03193087

Voss, A., Rothermund, K., & Voss, J. (2004). Interpreting the parameters of the diffusion model: An empirical validation. *Memory & Cognition*, *32*(7), 1206–1220. doi:10.3758/BF03196893

Voss, A., & Voss, J. (2007). Fast-dm: A free program for efficient diffusion model analysis. *Behavior Research Methods*, *39*(4), 767–775. doi:10.3758/BF03192967

Voss, A., & Voss, J. (2008). A fast numerical algorithm for the estimation of diffusion model parameters. *Journal of Mathematical Psychology*, *52*(1), 1–9. doi:10.1016/j.jmp.2007.09.005

Voss, A., Voss, J., & Klauer, K. C. (2010). Separating response-execution bias from decision bias: Arguments for an additional parameter in ratcliff's diffusion model. *British Journal of Mathematical and Statistical Psychology*, *63*(3), 539–555. doi:10.1348/000711009X477581

Wagenmakers, E.-J., Maas, H. L. J. v. d., Dolan, C. V., & Grasman, R. P. P. P. (2008). Ez does it! extensions of the ez-diffusion model. *Psychonomic Bulletin & Review*, *15*(6), 1229–1235. doi:10.3758/PBR.15.6.1229

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. Retrieved from https://ggplot2.tidyverse.org

Wiecki, T. V., Sofer, I., & Frank, M. J. (2013). Hddm: Hierarchical Bayesian estimation of the drift-diffusion model in python. *Frontiers in Neuroinformatics*, *7*, 1–19. doi:10.3389/fninf.2013.00014

Citation

Lin, Y.-S., & Strickland, L. (2020). Evidence accumulation models with R: A practical guide to hierarchical bayesian methods. *The Quantitative Methods for Psychology*, *16*(2), 133–153. doi:10.20982/tqmp.16.2.p133

Received: 12/09/2018 ~ Accepted: 11/08/2019

Listing 3 follows.

Listing 3 ■ Setting the association between the model parameters and the prior distribution for Example 2.

```
model <- BuildModel(
   p.map      = list(a   = "RACE", v = c("S", "RACE"), z = "RACE", d = "1",
                     sz  = "1",    sv = "1", t0 = c("S", "RACE"), st0 = "1"),
   match.map = list(M = list(G = "shoot", N = "not")),
   factors    = list(S = c("gun", "non"), RACE = c("white", "black")),
   constants = c(st0 = 0, d = 0, sv = 0, sz = 0),
   responses = c("shoot", "not"),
   type       = "rd")
## Parameter vector names are: ( see attr ("p.vector") )
## [1] "a.black"      "a.white"      "v.gun.black"  "v.non.black"  "v.gun.white"  "v.non.white"
## [7] "z.black"      "z.white"  "t0.gun.black"  "t0.non.black"  "t0.gun.white"  "t0.non.white"
## Constants are (see attr ("constants") ):
##    st0    d   sz   sv
##      0    0    0    0
## Model type = rd

npar      <- length(GetPNames(model))
pnames    <- GetPNames(model)
pop.loc   <- c(1, 1, 2.5, 2.5, 2.5, 2.5, .50, .50, .4, .4, .4, .4)
pop.scale <- c(.15, .15, 1, 1, 1, 1, .05, .05, .05, .05, .05, .05)
names(pop.loc)   <- pnames
names(pop.scale) <- pnames

p.prior <- BuildPrior(
  dists = rep("tnorm", npar),
  p1    = c(1, 1, 2.5, 2.5, 2.5, 2.5, .50, .50, .4, .4, .4, .4),
  p2    = c(1.5, 1.5, 10, 10, 10, 10, .5, .5, .5, .5, .5, .5),
  lower = c(rep(0, 2), rep(-5, 4), rep(0, 6)),
  upper = c(rep(10, 2), rep(NA, 4), rep(5, 6)))

mu.prior <- BuildPrior(
  dists = rep("tnorm", npar),
  p1    = pop.mean,
  p2    = pop.scale*10,
  lower = c(rep(0,  2), rep(-5, 4), rep(0, 6)),
  upper = c(rep(10, 2), rep(NA, 4), rep(5, 6)))

sigma.prior <- BuildPrior(
  dists = rep("beta", npar),
  p1    = rep(1, npar),
  p2    = rep(1, npar),
  upper = rep(2, npar))
```