Behav Res
DOI 10.3758/s13428-015-0569-0

Efficient simulation of diffusion-based choice RT models on CPU and GPU

Stijn Verdonck · Kristof Meers · Francis Tuerlinckx

Abstract In this paper, we present software for the efficient simulation of a broad class of linear and nonlinear diffusion models for choice RT, using either CPU or graphical processing unit (GPU) technology. The software is readily accessible from the popular scripting languages MATLAB and R (both 64-bit). The speed obtained on a single high-end GPU is comparable to that of a small CPU cluster, bringing standard statistical inference of complex diffusion models to the desktop platform.

Keywords Diffusion model · First passage time · Simulation · Euler-Maruyama · GPU

Introduction

A common observable in many experimental paradigms are choice response times (RT). In such paradigms, the participant (or animal) has to choose as quickly as possible between several (most commonly, two) alternatives. In the past decades, choice RTs have also been the focus of intensive mathematical modeling in experimental psychology and neurosciences (see e.g., Stone, 1960; Laming, 1968; Link and Heath, 1975; Ratcliff, 1978; Luce, 1986). The most successful class of mathematical models for choice RTs is diffusion models (e.g., Ratcliff, 1978, Usher and McClelland, 2001). In diffusion models, noisy information is integrated across time until there is enough evidence for one of the options and then the corresponding response is executed. A great variety of diffusion models exist, differing from each other in the complexity of the information accumulation process and response criterion.

The time it takes a diffusion process to exceed a certain threshold is called the first-passage time (Karlin and Taylor 1981). The numerical methods to calculate the first-passage time densities can be divided into two classes: deterministic methods and simulation-based methods. Deterministic methods can be based on analytical work (e.g., for the constant drift diffusion model, for the constant drift diffusion model, Tuerlinckx, 2004; Navarro and Fuss, 2009; Blurton et al., 2012; Gondan et al., 2014) or more straightforward numerical integration (e.g., Smith, 2000; Diederich and Busemeyer, 2003; Voss and Voss, 2008). For high dimensionality, straightforward deterministic numerical integration becomes problematic, as the underlying matrices grow in size with a power equal to the dimensionality of the model. The main drawback of simulation-based methods (e.g., Brown et al., 2006) on the other hand, is that smooth first-passage time distributions, generally require a large number of simulations.

The goal of this paper is to present a fast, reliable, and robust software for simulating high-resolution first-passage times distributions of a broad class of diffusion processes. In particular, the diffusion models that are important for this paper are: the constant drift diffusion model (both the basic version as described in Cox and Miller (1977) as well as the elaborated version proposed by Ratcliff (1978)), the Ornstein–Uhlenbeck process (Busemeyer and Townsend, 1992, 1993), the Leaky Competing Accumulator (Usher and McClelland 2001), and the recently introduced Ising Decision Maker (IDM Verdonck and Tuerlinckx, 2014). The software seeks to maximally benefit from parallel processor architectures, especially that of current generation GPUs.

In what follows we start with a short theoretical description of the class of diffusion models for choice RTs that is

S. Verdonck (✉) · K. Meers · F. Tuerlinckx
Faculty of Psychology and Educational Sciences,
University of Leuven, Leuven, Belgium
e-mail: stijn.verdonck@ppw.kuleuven.be

Published online: 12 March 2015

compatible with the software. Next, we discuss the numerical mathematics of simulating first-passage time distributions, followed by the details of the software implementation. Subsequently, its use in MATLAB and R is explained, and then some benchmark examples are shown.

Diffusion-based choice RT models

In what follows, the general case of k choice alternatives is considered.[1] All diffusion models that fall within the scope of our software can be represented as a k-dimensional stochastic differential equation (SDE) defined for the time-dependent stochastic vector $\mathbf{y} = (y_1, ..., y_k)$ (for simplicity, we suppress the explicit time index t unless it is explicitly needed):

$$d\mathbf{y} = A(t, \mathbf{y}) \cdot dt + \mathbf{C} \cdot d\mathbf{W} \tag{1}$$

where $\mathbf{C}$ is a diagonal matrix with as diagonal elements the diffusion constants for each of the k processes and $\mathbf{W}$ is a k-dimensional Wiener process. The software is limited to drift rate vectors $A(t, \mathbf{y})$ of the following general form:

$$A(t, \mathbf{y}) = v(t) + \Gamma \cdot \mathbf{y} + \mathbf{g} \odot (\log(\mathbf{y}) - \log(1 - \mathbf{y})), \tag{2}$$

where $\odot$ is the element-wise multiplication operator. The (possibly time-varying) vector $v(t)$ represent the part of the drift rate vector independent of the current position, Γ is a k-by-k matrix with the coefficients of the self-influence and mutual influence terms and $\mathbf{g}$ the importance of the k nonlinear terms. In order to clarify the previous equation further, we also give the p-th component of the drift rate: $A_p(t, \mathbf{y}) = v_p(t) + y_p \sum_q \Gamma_{pq} y_q + g_p \left(\log(y_p) - \log(1 - y_p)\right)$. In its totality, this vector is only defined where $0 < y_p < 1$. For this reason, we only consider processes confined to the k-dimensional unit hypercube. This, however, does not restrict us from simulating linear models (with $g_p = 0$ for all p) parameterized outside of this scope. Evidently, any bounded linear diffusion process can be rescaled to another linear diffusion process that is confined to the scope of the unit hypercube and has exactly the same RT distributions (which is the reason why in the diffusion literature some researchers, e.g., Ratcliff (1978), take $c = 0.1$ for the diffusion constant while others, (Voss and Voss 2008), take $c = 1$).

Because of the log-functions in Eq. 2, for $g_p > 0$, the process is automatically bounded by the k-dimensional unit hypercube, as the drift rate goes to infinity for y_p approaching 0 and to minus infinity for y_p approaching 1. For $g_p \leq 0$, however, this is no longer the case. To keep the process from leaving the unit hypercube, at any point in time, y_p is taken to be $\min(\max(0, y_p), 1)$.[2]

For all models under consideration, the stimulus is represented by $v(t)$ (for $t \geq 0$) and drives the stochastic system's coordinates away from the evidence position at stimulus onset, $y_p(0)$. This process of evidence accumulation continues until the decision criterion (a geometrical boundary) corresponding to one of the choice alternatives is reached. This alternative is the outcome of the decision. The non-deterministic time at which the boundary crossing event occurs (first passage time) is called the decision time T. The decision time distribution (commonly conditional on the chosen response), is the quantity we wish to approximate through simulations.

For the decision criteria, the software allows for four different types of boundaries: Three different shapes for the general k-dimensional case (one boundary or choice per evidence dimension) and, exclusively for the one-dimensional case, the option to have two opposite boundaries. We will explain the boundary shape feature in detail in a later section, when we apply the simulation framework to the recent IDM model.

Usually the observed response time RT is assumed to be a sum of the decision time T and a (possibly random) non-decision or residual time T_{er}. For the users convenience, the option of defining a uniform T_{er} distribution was already included. Obviously, any user supplied T_{er} distribution can be added manually after simulating the naked decision time distribution.

The broad theoretical framework outlined in Eqs. 1 and 2, allows for the simulation of a plenitude of diffusion models with one and the same software. In the remainder of this theoretical section, we will present a number of popular models that can be simulated using the proposed software. For convenience, we will assume time homogeneous drift rates and therefore suppress the time index t in $A(t, y)$. Nevertheless, as will become clear in the software section, certain types of time-varying drift rates are supported.

In the remainder of this section, we will give an overview of four diffusion-based choice RT models. A more detailed and theoretical discussion of these models can be found in Bogacz et al. (2006) and Verdonck and Tuerlinckx (2014).

Constant drift diffusion model

The constant drift diffusion model has been introduced by Ratcliff (1978) and is the most commonly used model for

[1] At this time, the software is only compiled for k up until 6. Compiling separate function instances based on the same code for different values of k allows for k-specific loop unrolling, significantly increasing efficiency for lower values of k. Compiling extra function instances for larger k is simple and requires no substantial changes to the code. The compiled program will, however, increase in size.

[2] For numerical reasons this is approximated by $y_p = \min(\max(0.0001, y_p), 0.9999)$.

two-choice RT. It is assumed that a single univariate diffusion process integrates noisy evidence over time until a criterion is reached. The constant drift diffusion model can be framed in the general model from Eq. 1 as follows (with $k = 1$):

$$dy = \nu \cdot dt + c \cdot dW \tag{3}$$

such that $A(y) = \nu$ (hence constant drift diffusion model) and $y(0) = z$. Because of reasons of identifiability, c is set to a constant (usually, 0.1). The drift rate ν is generally taken to be different for different stimuli. The accumulation process stops if the process y hits an upper boundary a or a lower boundary 0. Each of the two boundaries correspond to a choice alternative. The starting position z is often expressed in a relative way with respect to the upper boundary: $z_r = \frac{z}{a}$. As said before, the observed choice response time is defined as: $RT = T + T_{er}$.

The basic version of the constant drift diffusion model may fail to explain some empirical phenomena (e.g., slow and fast errors). A common way to accommodate these shortcomings is to allow for trial-to-trial variation in some of the parameters of the model. For instance, for the j-th presentation of a given stimulus, the drift rate is assumed to be a draw from the normal density with an average drift rate and variance:

$$\nu_j \sim N(\nu^{avg}, \eta^2).$$

Also the starting point and the non-decision time are assumed to be sampled from distributions:

$$z_{rj} \sim U\left(z_r^{avg} - \frac{s_z}{2}, z_r^{avg} + \frac{s_z}{2}\right)$$

and

$$T_{er_j} \sim U\left(T_{er}^{avg} - \frac{s_{T_{er}}}{2}, T_{er}^{avg} + \frac{s_{T_{er}}}{2}\right).$$

Ornstein–Uhlenbeck diffusion model

The Ornstein–Uhlenbeck diffusion model (Busemeyer and Townsend, 1992, 1993) is very similar to the constant drift diffusion model. The only difference is that the drift rate changes linearly with the accumulated evidence:

$$A(y) = \nu + \gamma y,$$

with factor γ quantifying the linear dependency. The SDE is:

$$dy = (\nu + \gamma y) \cdot dt + c \cdot dW. \tag{4}$$

As in the constant drift diffusion model, the observed choice response time is defined as: $RT = T + T_{er}$. The drift rate ν, T_{er} and z_r can also be assumed random from trial to trial.

Leaky competing accumulator model

The Leaky Competing Accumulator diffusion model (LCA; Usher and McClelland, 2001) assumes k competing diffusion processes, of which the first one that crosses its corresponding threshold determines the response. In contrast to the two former models, the number of choice alternatives (and hence the number of thresholds) is equal to the number of evidence dimensions. Usually, the model is applied to two-choice RT tasks and then the number of processes is restricted to two (i.e., $k = 2$); for ease of presentation, this is also the case we will discuss here but the generalization is evident.

The drift rate vector for the two-dimensional LCA equals $A(\mathbf{y}) = \nu + \Gamma \cdot \nu$ with $\nu = (\nu_1, \nu_2)^T$ being the vector of drift rates and $\Gamma = \begin{pmatrix} \gamma & \kappa \\ \kappa & \gamma \end{pmatrix}$ the matrix with the leakage parameter γ on the diagonal and off-diagonally the cross-pool interaction parameter κ.

It is commonly assumed that the diffusion process starts at the origin (i.e., $y_p(0) = 0$ for $p = 1, 2$) but any other starting point can be considered. The process continues until one of the processes crosses its threshold before the other; these thresholds are denoted as a_1 and a_2 (both larger than zero). Because there are only thresholds in the positive quadrant, the two-dimensional process may wander off into the negative direction. To prevent this from happening, the processes are forced to remain positive: At any point in time, y_p ($p = 1, 2$) is taken to be $\max(0, y_p)$. Like for the two other models, the observed choice response time is defined as: $RT = T + T_{er}$, where T_{er} can be random from trial to trial (the remaining LCA parameters are not assumed to vary randomly from trial to trial).

The Ising Decision Maker (IDM)

The IDM has been proposed by Verdonck and Tuerlinckx (2014). In the IDM, k pools of N_p ($p = 1, ..., k$) binary stochastic neurons are assumed. Thus, each neuron can take two possible values: active (1) or inactive (0). Neurons within a pool p excite each other (represented by parameter W_p^+) and neurons between two pools p and p' inhibit each other (represented by parameter $W_{pp'}^-$). The amount of randomness of the neurons is determined by the inverse stochastic temperature β. Each pool has a common neural activation threshold Θ_p. For reasons of identifiability, β should be constrained to a fixed values (the authors take $\beta = \frac{1}{24}$).[3] The macroscopic characteristic of interest is the mean neural activity in the respective pools: $\mathbf{y} = (y_1, ..., y_k)$ where y_p is the proportion of neurons equal to one in pool p.

[3] In the IDM, there is another quasi-identifiability problem, requiring extra constraints. The authors take $\Theta_p = 52500$, for all p.

Associated with this model, there is a so-called free energy function that can be approximated as:

$$F(\mathbf{y}) = (\mathbf{B}-\Theta)^T \cdot \mathbf{y} + \mathbf{y}^T \cdot \mathbf{W} \cdot \mathbf{y} + \beta^{-1} \sum_p N_p (y_p \log(y_p) + (1-y_p)\log(1-y_p)), \quad (5)$$

with the vector $\mathbf{B}$ being the external field or input from the environment on each of the k pools. To simplify notation, we introduced a interaction weights matrix $\mathbf{W}$ that has as diagonal elements $-W_p^+$ and off-diagonal elements $\frac{1}{2}W_{pq}^-$. In Verdonck and Tuerlinckx (2014), it is shown how the time evolution of the underlying microscopic network can be reduced to an ongoing stochastic minimization process of the k-dimensional free energy function $F(\mathbf{y})$. The free energy function is stimulus dependent: before the stimulus is presented, $\mathbf{B} = \mathbf{0}$, but after stimulus presentation, the vector $\mathbf{B}$ represents the input of the stimulus on each of the k pools.

The decision criteria for the IDM suggested by Verdonck and Tuerlinckx (2014) are k box-shaped or rectangular boundaries. Their geometry is illustrated in Fig. 1 for $k = 2$, together with some additional boundary shapes that are also supported by the software. The box associated with pool p has one of its corners located on the tip of the p-th unit vector $\mathbf{e}_p$, thus coinciding with a corner of the unit hypercube. k other corners are located each on one of the neighboring unit hypercube ridges, at distances h_{pc} of the initial corner (with $c = 1, \ldots, k$ the dimension parallel to the respective ridge). This coincides with the following detection rule: Response option p is chosen, if $y_p > 1-h_{pp}$ and $y_q < 1-h_{qq}$ for all $q \neq p$. The reason for these unconventional boundaries is that the IDM, unlike the traditional diffusion models, expects the process to evolve to one of k stable states (each representing a choice alternative)

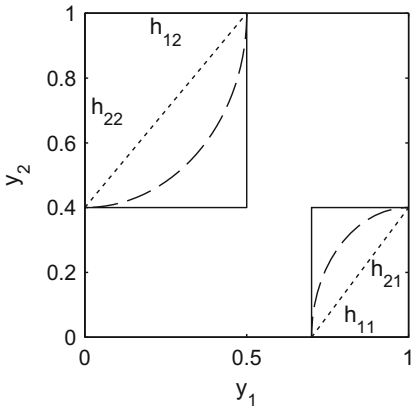


Fig. 1 IDM decision boundaries for $k = 2$. h_{pc} refers to the distance between the tip of the p-th unit vector ($p = 1, \ldots, k$) and the intersection of the decision boundary with the unit hypercube ridge parallel to dimension c. Three shapes are available: rectangular *(solid line)*, an ellipsoid *(dashed line)*, and a hyperplane *(dotted line)*

that are typically located in these particular corners of the evidence space. In Fig. 1, two other types of decision criteria with a similar parameterization are shown, a hyperplane (or triangle in two dimensions) and ellipse.

The IDM has a natural starting point distribution: In absence of a stimulus, the activation pools randomly vary according to an equilibrium distribution that is located in a low activation region (with an appropriate choice of parameters). In practice, this equilibrium is sampled by simulating a period of stimulus anticipation in which no stimulus is applied. For realistic parameters, this should be sufficient time for the process to largely forget any original position (of low activation) and can be considered a random sample of the spontaneous equilibrium distribution. An interesting byproduct of this stimulus anticipation period is the possible occurrence of premature decisions (the process spontaneously passing a decision boundary before the stimulus was presented). The longer the anticipation period, the higher the chance of a premature decisions. For a typical decision experiment, the number of premature decisions should be negligible, and model parameters that do not respect these constraints cannot be considered realistic. As usual, the observed choice response time is defined as: $RT = T + T_{er}$, where T_{er} can be assumed to vary randomly from trial to trial.

Another important aspect of the IDM is that the proposed dynamics, allows the resulting stochastic process to be either continuous (in which case it is equivalent to a multidimensional nonlinear diffusion model) or discrete (this is called coarse grained dynamics, and related to the simultaneous updating of more neurons). We will discuss these in turn.

Continuous dynamics For the continuous dynamics, the IDM is a special case of the general diffusion equation, shown in Eq. 1, with $A(\mathbf{y}) = D_p \beta \nabla F(\mathbf{y})$ and $C_{pp} = \sqrt{2D_p}$, where D_p is the pool-specific diffusion constant (see Verdonck and Tuerlinckx, 2014). Because for each pool p, the number of neurons $N_p > 0$ in Eq. 5, the drift rate $A(\mathbf{y})$ will automatically keep the system inside the unit hypercube. This can be seen from the fact that the nonlinear part in $A_p(\mathbf{y})$, or $\frac{\partial}{\partial y_p}\left(y_p \log(y_p) + (1-y_p)\log(1-y_p)\right) = \log(y_p) - \log(1-y_p)$ pulls the system away from the edges 0 and 1 for every dimension p.

Coarse-grained dynamics As an alternative account for the speed-accuracy trade-off, Verdonck and Tuerlinckx (2014) proposed to allow a coarser stochastic minimization of the free energy function in Eq. 5. For such a coarse-grained process, the free energy minimization occurs with collective step size parameter σ_p and time steps $\triangle t$. In this case, there is no corresponding diffusion equation because the process does not operate in continuous time. Although

Table 1 Overview of the models (with the name of the software wrappers between parentheses)

Drift rate	Constant	Linear	Nonlinear
unidimensional	Constant Drift Diffusion	Ornstein–Uhlenbeck	
with two criteria	Model (LDMDist)	(LDMDist)	
k dimensional		Leaky Competing	IDM (IDMDist)
with k criteria		Accumulator (LCADist)	

coarse-grained dynamics goes beyond the diffusion framework outlined at the start of the paper, the feature was added to have complete simulation capabilities for the IDM.

Summary

In Table 1 an overview is shown of the different diffusion models discussed in this paper. The models are classified according to (horizontally) linearity of the drift rate (constant, linear, nonlinear) and (vertically) the dimensionality of the evidence space (i.e., k) together with the number of criteria. Some of the combinations correspond to existing models; in those cases, we have included the model names in the table. It can be seen that the LCA and IDM are defined for k processes and they have k criteria. The constant drift diffusion and Ornstein–Uhlenbeck model are unidimensional (i.e., $k = 1$), but there are two criteria (an upper and a lower boundary). The terms between parentheses in Table 1 refer to the names of the wrapper functions that were developed to facilitate the use of the general function for the corresponding models.

Numerical aspects of the simulation of diffusion-based decision models

The stochastic differential equation from Eq. 1 together with the decision criteria, in general, do not lead to closed-form expressions for the choice probabilities and choice RT distributions. For the specific case of the one-dimensional constant drift diffusion model, a lot of analytical work has been done to reduce the computational burden (Tuerlinckx 2004; Navarro and Fuss 2009; Blurton et al. 2012; Gondan et al. 2014), but none of these approaches are readily extendible to other, more complex, diffusion models. For quite a number of models, deterministic numerical integration methods have already been implemented (see e.g., Smith, 2000; Diederich and Busemeyer, 2003; Voss and Voss, 2008). However, for problems of more than one or two dimensions, straightforward deterministic numerical integration will become very slow. Alternatively, it is possible to simulate multiple instances of the stochastic process and non-parametrically infer the theoretical RT distributions from this massive virtual experiment. In this paper, we use the Euler–Maruyama algorithm (see Kloeden and Platen, 2011) to simulate instances of the stochastic process.

Consider again the more general time inhomogeneous stochastic differential equation, as shown in Eq. 1. Suppose the process starts at time 0 at some initial position $\mathbf{y}_0$. Next, we discretize the time: $0 < \tau_1 < \tau_2 < \ldots$. Usually, we will assume equal time steps: $\tau_n = n\tau$. The state at time point $\mathbf{y}_{(n+1)\tau}$ equals:

$$\mathbf{y}_{(n+1)\tau} = \mathbf{y}_{n\tau} - A(n\tau, \mathbf{y}_{n\tau})\tau + \mathbf{C} \cdot \triangle \mathbf{W}_{(n+1)\tau}, \quad (6)$$

where the random increments $\triangle \mathbf{W}_{(n+1)\tau} = \mathbf{W}_{(n+1)\tau} - \mathbf{W}_{n\tau}$ are independently normally distributed with mean 0 and variance τ.

Because we need the first-passage time, we keep on simulating the process until a decision criterion is satisfied. This event determines both the response time RT (assuming the decision criteria is hit at time $T = N\tau$ for some N and the non-decision time is T_{er}) and the response p (assuming it was the criterion corresponding to alternative p that was reached first). This comprises the simulation of one single choice RT and has to be repeated until the desired number of simulated choice RTs is reached.

Finally, the (sometimes millions of) simulated choice RTs are binned into their choice-respective discrete time distribution (typically 3000 consecutive bins of 1 millisecond wide). Without much loss of precision, this is a much more portable way of returning the simulated data (as compared to a long vector with raw unbinned values).

Parallel implementation of the Euler–Maruyama algorithm

Simulating a single RT does not take a long time. The computational challenge lies in the large number of simulations necessary to provide sufficiently accurate approximations to the underlying RT distributions. Because the simulations are completely independent of each other, they can be run fully parallel.

Much more then CPUs, GPUs are built for massive parallel calculations. NVIDIA developed an extension of the C programming language, called CUDA, to optimally program their GPUs for custom parallel calculations.[4]

[4]There is a valid open source alternative called OpenCL that also works for AMD (previously ATI) and Intel GPUs, but for NVIDIA GPUs specifically, CUDA is the best choice.

Table 2 Settings argument structure, applicable to all wrappers

Field name	Type	Dimensions	Extra checks	Default	Description
nWalkers	integer	scalar	$1 \leq$ nWalkers	100000	number of simulated trials
seed	integer	scalar		0	seed of the random number generator (0 means clock-dependent)
dt	float	scalar	$0 \leq$ dt	0.001	time step Euler–Maruyama method
nBins	integer	scalar	$1 \leq$ nBins	3000	number of bins constituting the simulated RT probability distribution
binWidth	float	scalar	$0 \leq$ binWidth	0.001	RT difference spanned by each bin
nGPUs	integer	scalar	$0 \leq$ nGPUs	1	number of assigned GPUs (0 uses CPU)
gpuIds	integer	1 x nGPUs	$0 \leq$ gpuIds[i]	gpuIds[i] $= i - 1$	ids of assigned GPUs
loadPerGPU	float	1 x nGPUs	$0 \leq$ loadPerGPU[i]	loadPerGPU[i] $= 1$	relative load per assigned GPU

The core simulation code is exactly the same for CPU and GPU, except for the generation of normally distributed random numbers. Although the random number generator is the same (parallellized XORWOW from the CUDA library), on a CPUs it is more efficient to use the Ziggurat method (Marsaglia and Tsang 2000) to transform uniformly distributed random numbers to normally distributed ones, while on a GPU a simple Box-Muller transformation is faster.[5]

Description of the simulation software

In this section, we will give a platform-independent description of the software, which is available at http://ppw.kuleuven.be/okp/software/RTDist/. The package contains a Quick Start Guide with information on installation, testing, and some examples of use. To invoke the software from MATLAB or R, the wrapper functions and arguments explained below can be used. Obviously, the rules of the respective languages have to be respected.

All simulations are performed with a general function RTDist. However, for ease of use, three wrappers have been developed that allow the simulation of several popular models using each their own natural parameterization: LDMDist simulates the constant drift diffusion model and the Ornstein–Uhlenbeck model, LCADist simulates the LCA and IDMDist simulates the IDM. Experienced users may simulate directly from the RTDist function as this has some flexibility to specify ad hoc models that deviate from the standard models discussed earlier.

The basic wrapper functions LDMDist, LCADist and IDMDist have two main input arguments: the first argument specifies the model parameters, the second argument specifies general algorithmic and hardware settings (the latter one is common to all wrappers).[6] We start with discussing the general settings and then move to the specific wrappers.

General settings

Central to the operation of all wrappers is the settings argument. These general settings can be found in Table 2. Most entries are self-explanatory, but two fields deserve some extra attention. First, the seed of the random number generator is equal to 0 by default, and will result in a clock-dependent seed generated by the model wrapper. Any other value for seed will be used unaltered. Second, the simulated RTs are binned in order to approximate the theoretical probability distributions. The default bin width is 0.001 (equal to τ from the Euler–Maruyama algorithm; it makes no sense to have a smaller bin width than a time step τ but if this is the case, RTs will be distributed evenly across nearby bins to avoid artificial gaps).

[5] We are aware that, for both CPU and GPU, the hardware can still be used a bit more efficiently and some extra speed gain could be achieved (we expect no more than a factor 2). The implementation, however, is rather tedious so we will only consider pursuing these extra optimizations if there is a concrete demand.

[6] There is a third optional argument (called verbose) that determines whether error, warning and/or information messages are shown or not. Set to 0, RTDist does not show any messages, set to 1, RTDist shows error and warning messages only, set to 2, information about type conversions and the use of defaults is shown, as well as error and warning messages.

Checks and defaults are applied automatically upon wrapper entry but can be applied manually by running `settings=checkSettings(settings)`, assuming settings is the name of the variable used to store the general settings in.

LDMDist: A wrapper for the constant drift diffusion model and the Ornstein–Uhlenbeck model

The specific model parameters structure for the LDMWrapper (simulating the constant drift diffusion model and the Ornstein–Uhlenbeck model) is shown in Table 3. The fields can be traced back relatively simple to Eqs. 3 and 4. Special attention should be given to the last argument in Table 3 as it allows the user to specify a time-varying profile corresponding to a time-varying drift rate.

Checks and defaults are applied automatically upon wrapper entry but can be applied manually by running `LDMPars=checkLDMPars(LDMPars)`, assuming LDMPars is the name of the variable used to store the LDM model parameters in.

LCADist: A wrapper for the leaky competing accumulator model

The specific model parameter structure for the LCAWrapper (simulating the Leaky Competing Accumulator model) is shown in Table 4.

Checks and defaults are applied automatically upon wrapper entry but can be applied manually by running `LCAPars=checkLCAPars(LCAPars)`, assuming LDMPars is the name of the variable used to store the LCA model parameters in.

IDMDist: A wrapper for the Ising Decision Maker

The specific model parameter structure for the IDMWrapper (simulating the Ising Decision Maker) is shown in Table 5.

Checks and defaults are applied automatically upon wrapper entry but can be applied manually by running `IDMPars=checkIDMPars(IDMPars)`, assuming IDMPars is the name of the variable used to store the IDM model parameters in.

Table 3 Parameter argument structure for LDMDist, the wrapper to simulate linear diffusion models (constant or linear drift)

Field name	Type	Dimensions	Bounds	Default	Description
nStimuli	integer	scalar	$1 \leq$ nStimuli		number of stimuli
c	float	scalar	$0 \leq$ c	0.1	diffusion constant (SDE notation)
a	float	scalar	$0 \leq$ a ≤ 0.99		position of upper response criterion or boundary separation
zr	float	scalar	$0 \leq$ zr ≤ 1	0.5	mean relative starting position
sz	float	scalar	$0 \leq$ sz $\leq$ a $\cdot \min(1 -$ zr, zr)	0	width of uniform starting position distribution
v	float	1 x nStimuli			drift rates induced by the stimuli (constant coefficient of SDE)
eta	float	1 x nStimuli	$0 \leq$ eta[i]	0	trial-to-trial standard deviation of drift rate
gamma	float	scalar		0	linear coefficient of SDE (zero for constant drift diffusion model)
Ter	float	scalar	$0 \leq$ Ter	0	mean non-decision time
sTer	float	scalar	$0 \leq$ sTer $\leq 2 \cdot$ Ter	0	width of uniform non-decision time distribution
profile	float	scalar OR nBins x 1		1	time-varying multiplier of drift rate

Table 4 Parameter argument structure for LCADist, the wrapper to simulate the LCA (Leaky Competing Accumulator)

Field name	Type	Dimensions	Extra checks	Default	description
nDim	integer	scalar	$1 \leq$ nDims		dimensionality evidence space
nStimuli	integer	scalar	$1 \leq$ nStimuli		number of stimuli
c	float	nDim x 1	$0 \leq$ c[i]	c[i] = 0.1	diffusion constant (SDE notation)
a	float	nDim x 1	$0.0001 \leq$ a[i] ≤ 0.9999		evidence thresholds
startPos	float	nDim x 1	$0.0001 \leq$ startPos[i] ≤ 0.9999	startPos[i] = 0.0001	starting position
v	float	nDim x nStimuli			drift rates induced by the stimuli (constant coefficients of SDEs)
Gamma	float	nDim x nDim			linear coefficients of SDEs (diagonal: self, off-diagonal: mutual)
Ter	float	scalar	$0 \leq$ Ter	0	mean non-decision time
sTer	float	scalar	$0 \leq$ sTer $\leq 2 \cdot$ Ter	0	width of uniform non-decision time distribution
profile	float	scalar OR nBins x 1		1	time-varying multiplier of drift rate

Output

Table 6 shows the structure of the output returned by the model wrappers discussed in the previous section. The main components of the output are *distributions, ok, extra* and *used*.

The first component, *distributions*, contains the simulated RTs. These RTs are organized in a matrix of integer frequencies, with in the rows the time bins and in the columns the concatenation of the stimuli and the response options (so that the total number of columns corresponds to the number of response options times the number of stimuli). More specifically, the RTs of the n-th out of o response options of the m-th stimulus are binned along the $((m-1) \cdot o + n)$-th column of the matrix (hence, the response option changes fastest over columns and the stimulus slowest). For the LDMWrapper (used for the constant drift diffusion model and the OU model), the number of response options is always two, for the other wrappers, the number of response options is equal to the dimensionality of the evidence space k, in the software referred to as *nDim*. Bearing in mind the width of these bins (as provided in the settings argument, see Table 2), we can see each column as the RT histogram of a particular stimulus/response combination.. This is illustrated in Fig. 2, where the distribution output is plotted resulting from a typical model input for the IDMWrapper, as shown in Table 7.

The third component, *extra*, contains some meta-information on the simulations that were run. In this data structure, two fields are relevant for all models. First, there is the field *walkersAccountedFor*, which, if no technical problems occurred, should always be equal to the number of simulations started (*nWalkers* from Table 2). This equality is checked within each model wrapper and the second component of the output, *ok*, contains the result of this check. A second important field is *binsExceeded*, which holds the number of simulations (walkers) that are still underway at the end of the last time bin; if *binsExceeded* is too large, then a natural remedy is to increase either the number of bins, *nBins*, or their width, *binWidth* (both fields from the settings argument).

Specifically for the IDM, the user needs to monitor the field *flood:* This field refers to the number of walkers that already hit a decision criteria before the stimulus is presented; if it is too high, the period of stimulus anticipation is too long (the field *spontaneousTime* in the IDMWrapper model input structure) or the spontaneous distribution of the IDM is too broad (which implies an overall unrealistic IDM parameter set). Overlapping boundary boxes may lead to walkers that hit both decision criteria simultaneously

Table 5 Parameter argument structure for IDMDist, the wrapper to simulate the IDM (Ising Decision Maker)

Field name	Type	Dimensions	Extra checks	Default	Description
nDim	integer	scalar	$1 \leq$ nDims		number of pools (dimensionality evidence space)
nStimuli	integer	scalar	$1 \leq$ nStimuli		number of stimuli
startPos	float	nDim x 1	$0.0001 \leq$ startPos[i] ≤ 0.9999	startPos[i] $= 0.3$	starting position
beta	float	scalar	$0 \leq$ beta	$1/24$	inverse statistical temperature
N	float	nDim x 1	$0 \leq$ N[i]		number of neurons in each pool
W	float	nDim x nDim			pool interactions (diagonal: self, off-diagonal: mutual)
boxShape	integer	scalar	$0 \leq$ boxShape ≤ 2	0	shape of the detection boxes (0: box, 1: hyperplane, 2: ellipsoid)
h	float	nDim x nDim			sizes of the detection boxes
Theta	float	nDim x 1			neural activation threshold
B	float	nDim x nStimuli			external fields induced by the stimuli
Ter	float	scalar	$0 \leq$ Ter	0	mean non-decision time
sTer	float	scalar	$0 \leq$ sTer $\leq 2 \cdot$ Ter	0	width of uniform non-decision time distribution
spontaneousTime	float	scalar	$0 \leq$ spontaneousTime		period of stimulus anticipation
dynamics	integer	scalar	$0 \leq$ dynamics ≤ 1	0	continuous (0) or coarse graining (1)
D (dynamics $= 0$)	float	nDim x 1	$0 \leq D_i$		diffusion constant (Fokker-Planck sense)
sigma (dynamics $= 1$)	float	nDim x 1	$0 \leq$ deltas[i]		step size
deltat (dynamics $= 1$)	float	scalar	$0 \leq$ deltat		time step
profile	float	scalar OR nBins x 1		1	time-varying multiplier of stimulus induced external field

(the field *multiArrival*). The two *lemmings* fields contain the number of simulations that were at least once artificially truncated to the unit hypercube. For LCA simulations, this is perfectly normal behavior. For the IDM, this could indicate that either N (from the IDMWrapper model input argument) is too low in comparison to the other parameters, or simulation time accuracy dt (from the settings argument) is too high.

Table 6 Output of any of the wrapper functions

Output	Type	Dimensions	Description
(a) Output structure. In MATLAB, these fields are returned as separate variables; in R, they have to be retrieved from a single returned list object.			
distributions	integer	nBins x (nDim x nStimuli) except LDM: nBins x (2 x nStimuli)	distributions (binned simulated RTs): for every stimulus there are nDim (or 2 for LDM) distributions
ok	logical	scalar	true if all simulations have been accounted for
extra (see (b))	struct (MATLAB) list (R)	nStimuli x 1 (MATLAB) nStimuli x 6 (R)	meta-information on the simulations
used (see (c))	struct (MATLAB) list (R)	NA	parameters and settings used for the obtained distributions
(b) Structure of extra: simulation meta-data per stimulus.			
flood	integer	scalar	number of trials arriving in a boundary box before the stimulus is presented
multiArrival	integer	scalar	number of trials arriving in multiple boundary boxes at the same time (possible with overlapping boundary boxes)
binsExceeded	integer	scalar	number of trial RTs exceeding the last bin
lemmingsSpontaneous	integer	scalar	number of trials that at one point exceeding [0.0001, 0.9999] interval before stimulus presentation
lemmingsStimulus	integer	scalar	number of trials that at one point exceeding [0.0001, 0.9999] interval after stimulus presentation
walkersAccountedFor	integer	scalar	check sum that should be equal to nWalkers (from the settings argument)
(c) Structure of used: the used input arguments.			
LDMPars/LCAPars/IDMPars	struct (MATLAB) list (R)	NA	wrapper parameters used
pars	struct (MATLAB) list (R)	NA	final RTDist parameters used
settings	struct (MATLAB) list (R)	NA	settings used

The last component, *used*, lists the used parameters (both in terms of the wrappers as in terms of the underlying general simulation function RTDist). This allows the user to retrace exactly which parameters were used to obtain the current result.

Performance of the software: Accuracy, speed and benchmarks

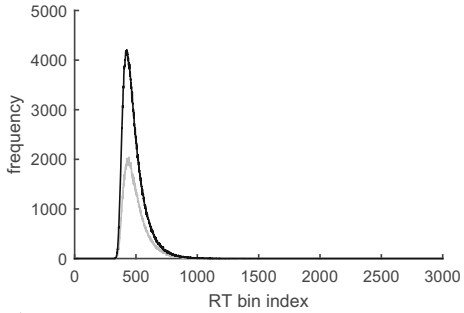
In this section we will first cross-check our software's results with the fast-dm package of Voss and Voss (2008). Fast-dm is limited to probability distribution functions of the constant drift diffusion model with trial-to-trial variability, so only the results of LDMWrapper can be (partially) checked.[7]

In a second analysis we study how (based on calculations with the IDMWrapper), the accuracy results converge for sufficient simulations (*nWalkers*) and a sufficiently small

[7]For LCAWrapper, we performed a manual check and were able to reproduce the LCA distributions (quantiles) presented in Brown et al. (2006). IDMWrapper is the first of its kind, so no comparison is possible.

	stimulus 1	
	response 1	response 2
bin 1	0	0
bin 2	0	0
...		
bin 500	2436	1379
bin 501	2357	1299
...		
bin 3000	0	0

(a) Output matrix



(b) Output distributions

Fig. 2 Distribution output resulting from the IDM model parameters shown in Table 7. The distribution output matrix, shown in panel **a**, has two columns and 3,000 rows. Each column contains the total number of simulations corresponding to a particular stimulus/response combination, distributed over 3,000 consecutive RT bins of 1 ms wide. In panel **b**, for each stimulus/response combination (column), a *solid line* plots the number of binned RTs (frequency) for increasing row index (RT bin index). The *darker line* is related to response 1 and the *lighter line* is related to response 2

values of the Euler–Maruyama time step (*dt*), and how these settings adversely affect computation time.

Finally, we provide a benchmark for a selection of CPU and GPU systems.

Accuracy: comparison to fast-dm results

To compare distributions calculated with fast-dm on the one hand and the LDMWrapper on the other hand, we use the Kolmogorov–Smirnov (KS) distance statistic (Mood 1950). Because for both techniques, the accuracy of the result depends on certain algorithmic parameters, the methods will only converge to the same result for adequate values of these parameters. It therefore makes sense to look at the difference between the distributions obtained with both techniques as a function of their respective accuracy parameters. In fast-dm, the accuracy is set by a single software-specific accuracy parameter: we will vary it from 1 to 5. In the RTDist wrappers, the settings argument contains two fields influencing the accuracy: the number of simulations (*nWalkers*) and the Euler–Maruyama time step size (*dt*). For this analysis, we take a very high number of simulations (10 million trials) and vary the Euler–Maruyama time step *dt* from 0.1 to 0.00001. Next, we create a representative comparison set by generating 500 constant drift diffusion model parameter sets (randomly sampled from uniform distributions for all parameters with reasonable lower and upper bounds), of which we calculate the model distributions with both techniques, and this for each accuracy setting. This results in 500 KS distances for each fast-dm/RTDist accuracy combination.

In Fig. 3, the median KS distance of the 500 comparisons (z-axis), is plotted against both the fast-dm accuracy parameter and the RTDist Euler–Maruyama time step *dt*. As could be expected, higher accuracies for both techniques lead to converging distributions (a KS distance of almost zero).

Based on this graph, we suggest using a Euler–Maruyama time step *dt* of at least 0.001 (which is in line with the findings of Brown et al. (2006)) and a fast-dm accuracy of at least 2.5. When using approximate intermediaries in an analysis (e.g., estimations based on approximate distributions), it is wise to at least check the convergence of this approximation for the final outcome (by comparing with higher accuracy approximations).

Using the 500 constant drift diffusion model parameter sets, and a highly accurate reference distribution of fast-dm, we can study the effect of *nWalkers* and *dt* on the simulated distribution accuracy. In Fig. 4, panel (a), is illustrated how the median KS distance between the simulated distributions and the high accuracy fast-dm reference, decreases with both an increasing number of simulations and a decreasing Euler–Maruyama time step. To give the reader an idea of the variability of this KS distance distribution, panel (b) zooms in on the most accurate *dt* value shown in panel (a)

Table 7 Typical model parameters for the IDM (see Verdonck and Tuerlinckx, 2014)

Field	Value (MATLAB notation)
nDim	2
nStimuli	1
N	[1000;1000]
W	[52500;52500]
h	[0.4,0.4;0.4,0.4]
Theta	[51450;51450]
B	[2600;2400]
D	[0.05;0.05]
dynamics	0
spontaneousTime	1
Ter	0.3

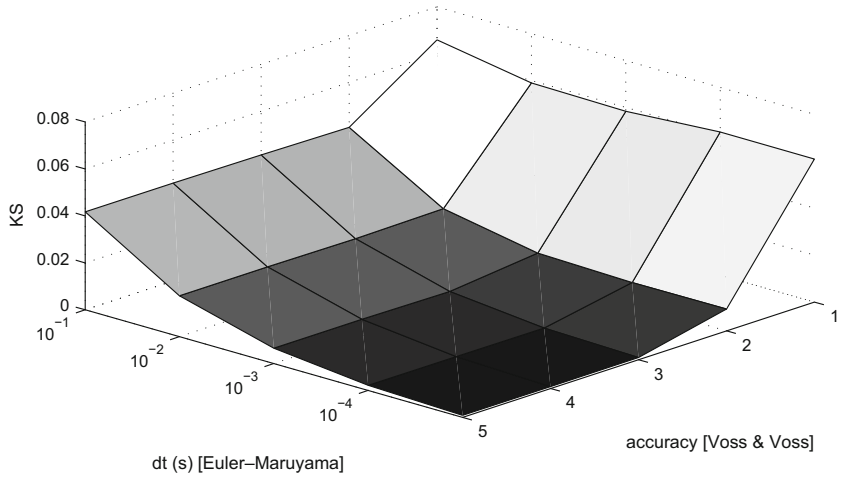


Fig. 3 The Kolmogorov–Smirnov statistic measuring the difference between the probability distribution functions generated by LDMDist (10 million trials) and fast-dm (numeric integral), based on 500 randomized parameter sets of the constant drift diffusion model. More specifically, the z-axis shows the median KS distance resulting from these parameter sets. For RTDist the Euler–Maruyama time step *dt* is varied, for fast-dm, the software-specific accuracy parameter is varied

(0.00001) and shows, apart from the median (0.5 quantile), two additional quantiles: 0.1 and 0.9.

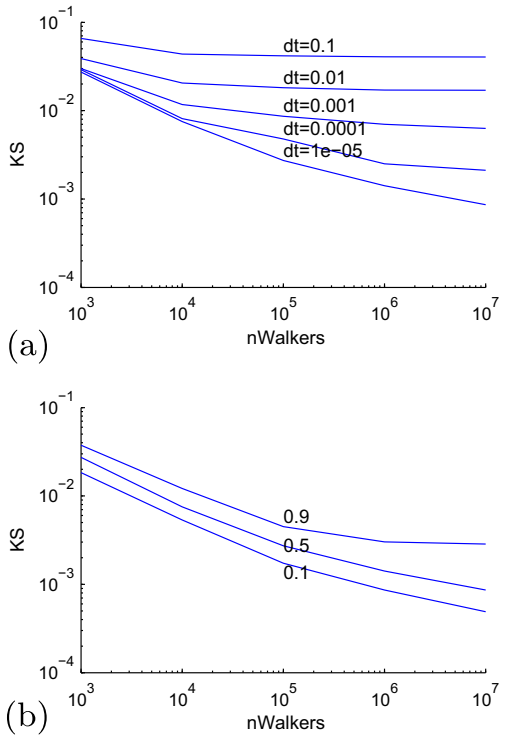


Fig. 4 The Kolmogorov–Smirnov statistic measuring difference between distributions generated by LDMDist (10 million trials) and fast-dm (numeric integral, accuracy setting 5), based on 500 randomized parameter sets of the constant drift diffusion model. Panel **a** shows the median KS distance resulting from these parameter sets versus an increasing number of simulated trials (*nWalkers*). Panel **b** shows for $dt = 0.00001$ three subsequent quantiles $(0.1, 0.5, 0.9)$ resulting from the random parameter sets, also versus an increasing number of simulated trials (*nWalkers*).

Speed: the impact of *nWalkers* and *dt*

Clearly, both a higher number of simulations and a lower Euler–Maruyama time step size will increase accuracy. Sadly, they also increase computation time. For a more comprehensive understanding of this trade off, we study the effect of both accuracy parameters on the calculation speed.

In Fig. 5, the median calculation time of the 500 random parameter sets of the constant drift diffusion model is shown for an increasing number of simulations (as run on two NVIDIA K20Xm GPUs). As expected, more accurate settings require more computation time. However, going to very low values of *nWalkers*, the (median of) the calculation time seems to become a bit slower as well. We have no solid explanation for this effect.

Figure 6 shows the impact of both *dt* and *nWalkers* on the mean calculation time of a typical parameter set of the

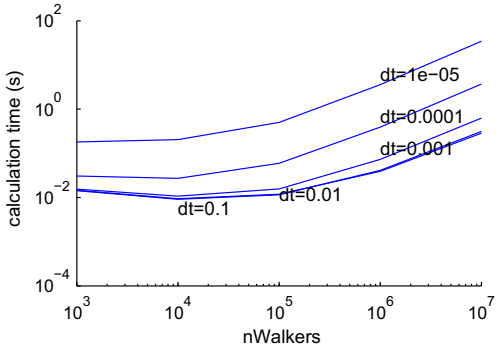


Fig. 5 The median calculation time of 500 random parameter sets of the constant drift diffusion model versus an increasing number of simulated trials (*nWalkers*)

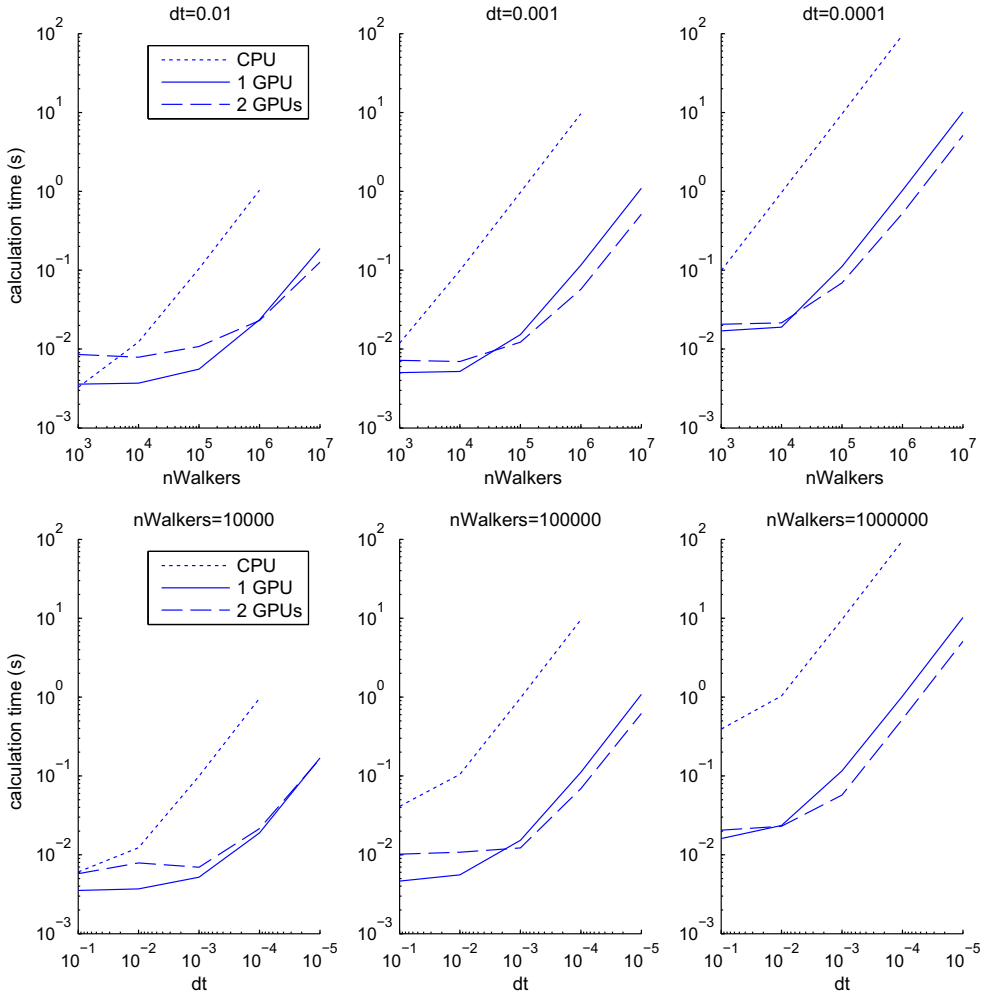


Fig. 6 Mean calculation time for 100 runs of a typical parameter set of the IDM (see Table 7) for different settings of *nWalkers* and *dt*

IDM (see Table 7), when the calculations are performed on a CPU, a GPU or two GPUs. An interesting observation is that the CPU calculation time increases linearly with *nWalkers*, already from the very start, while for the GPUs, a minimum number of simulations and is needed before they get to this linear regime. This indicates that the GPUs need a lot of "work", to deploy their full potential. The impact of decreasing *dt* on calculation time is similar, but not exactly the same, as that of increasing *nWalkers*. Decreasing *dt* makes the Euler–Maruyama algorithm longer, which masks some GPU specific overhead.

Benchmarks

In this last section, we present some CPU and GPU benchmark calculation times (IDMWrapper, typical parameter set

Table 8 Hardware configurations used for benchmarking

System name	CPU type	GPU type	OS
cluster node C	2 x 10-core Intel Xeon E5-2680 v2 @ 2.80GHz	NA	linux
cluster node G	2 x 6-core Intel Xeon E5-2630 @ 2.30 GHz	2 x NVIDIA K20Xm (sold as coprocessors)	linux
desktop	4-core Intel Xeon E5–1620 @ 3.60GHz	2 x NVIDIA GTX 780 (sold as gaming cards)	Windows
laptop	2-core Intel Core i5-3230M @ 2.60GHz	NVIDIA NVS 5200M	Windows

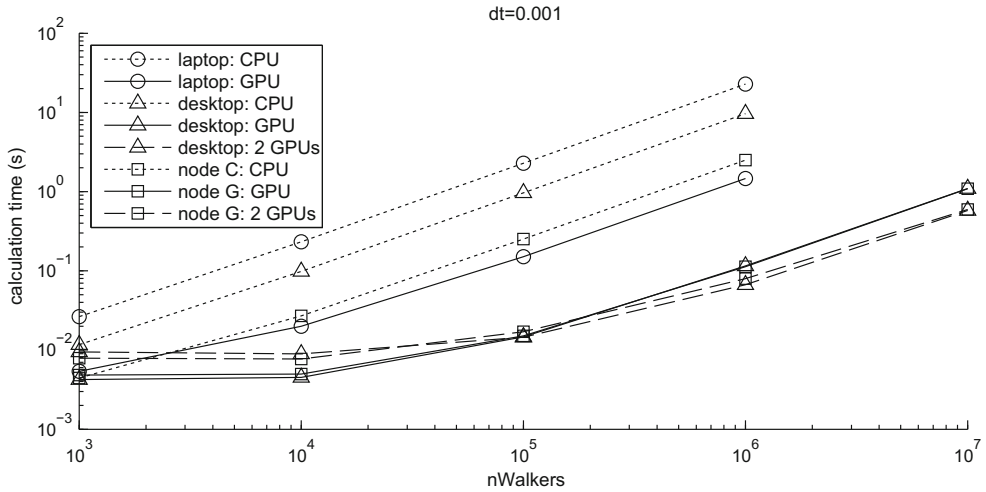


Fig. 7 Calculation times for different hardware configurations. For each configuration, the median calculation time for 100 runs of a typical parameter set of the IDM (see Table 7) is shown for increasing values of *nWalkers* ($dt = 0.001$)

shown in Table 7) based on four different hardware configurations. The configurations are specified in Table 8. In Fig. 7 median calculation times are shown for 100 runs of a typical parameter set of the IDM (see Table 7), for an increasing number of simulations. A first observation that can be made is that, for this mid to high end range of products, all GPUs are faster than any CPU (for sufficient simulations). Even the mid range laptop GPU outperforms the 20-core cluster CPU, and this already for 10,000 simulations. Also, the gaming GPUs in the desktop system perform comparable to their professional counterparts in the cluster (this is not surprising as these cards are based on the same chip). Another observation is that using two GPUs instead of one is only interesting for more than 100,000 simulations. For 10 million simulations, two GPUs are exactly twice as fast as one GPU. Finally, our desktop configuration with a single gaming GPU (at present time about 400 dollars) is, at 100,000 simulations, 20 times faster then a full 20-core CPU cluster node.

Conclusions

In this paper, we presented software developed for efficiently simulating a broad class of linear and nonlinear diffusion models for choice RT using either CPU or (multiple) NVIDIA CUDA GPU(s). The core code is written in C/CUDA, but four models were explicitly translated in wrapper functions for R and MATLAB: the constant drift diffusion model, the Ornstein–Uhlenbeck model, the Leaky Competing Accumulator and the Ising Decision Maker. On the level of the constant drift diffusion model, the results of RTDist were cross-checked with the existing fast-dm package (Voss and Voss 2008). If the desired number of simulations is high enough (10,000 or more for normal use on a high end GPU), GPUs greatly outperform CPUs.

We want to emphasize that the software presented in this paper only simulates choice RT distributions. It can immediately be used to investigate properties and predictions of a wide variety of models, but in order to estimate the parameters of one of those models, the software needs to be plugged into additional code. This additional code will depend on the chosen method of statistical inference (e.g., maximum likelihood estimation, method of moments, Bayesian inference). We leave it up to the user to make this choice and provide the necessary code.

Hence, with a small investment in a CUDA compatible high end GPU, a standard estimation approach to complex diffusion models (for which millions of simulations are necessary), formerly only feasible on expensive CPU clusters, is now possible on the desktop platform.

References

Blurton, S.P., Kesselmeier, M., & Gondan, M. (2012). Fast and accurate calculations for cumulative first-passage time distributions in Wiener diffusion models. *Journal of Mathematical Psychology*, *56*(6), 470–475. doi:10.1016/j.jmp.2012.09.002

Bogacz, R., Brown, E., Moehlis, J., Holmes, P., & Cohen. J.D. (2006). The physics of optimal decision making: A formal analysis of models of performance in two-alternative forced-choice tasks. *Psychological Review*, *113*(4), 700–765. doi:10.1037/0033-295X.113.4.700

Brown, S.D., Ratcliff, R., & Smith, P.L. (2006). Evaluating methods for approximating stochastic differential equations. *Journal of Mathematical Psychology*, *50*(4), 402–410. doi:10.1016/j.jmp.2006.03.004

Busemeyer, J.R., & Townsend, J.T. (1992). Fundamental derivations from decision field theory. *Mathematical Social Sciences*, *23*(3), 255–282. doi:10.1016/0165-4896(92)90043-5

Busemeyer, J.R., & Townsend, J.T. (1993). Decision field theory: A dynamic-cognitive approach to decision making in an uncertain environment. *Psychological Review*, *100*(3), 432–459. doi:10.1037/0033-295X.100.3.432

Cox, D.R., & Miller, H.D. (1977). *The theory of stochastic processes*. Taylor & Francis.

Diederich, A., & Busemeyer, J.R. (2003). Simple matrix methods for analyzing diffusion models of choice probability, choice response time, and simple response time. *Journal of Mathematical Psychology*, *47*(3), 304–322. doi:10.1016/S0022-2496(03)00003-8

Gondan, M., Blurton, S.P., & Kesselmeier, M. (2014). Even faster and even more accurate first-passage time densities and distributions for the Wiener diffusion model. *Journal of Mathematical Psychology*, *60*, 20–22. doi:10.1016/j.jmp.2014.05.002

Karlin, S., & Taylor, H.M. (1981). *A second course in stochastic processes*. Academic Press.

Kloeden, P.E., & Platen, E. (2011). *Numerical solution of stochastic differential equations*. Springer.

Laming, D.R.J. (1968). *Information theory of choice-reaction times*. Oxford England: Academic Press.

Link, S.W., & Heath, R.A. (1975). A sequential theory of psychological discrimination. *Psychometrika*, *40*(1), 77–105. doi:10.1007/BF02291481

Luce, R.D. (1986). *Response times: Their role in inferring elementary mental organization*. New York: Oxford University Press.

Marsaglia, G., & Tsang, W.W. (2000). The ziggurat method for generating random variables. *Journal of Statistical Software*, *5*(8), 1–7.

Mood, A.M. (1950). *Introduction to the theory of statistics*. New York: McGraw-Hill.

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

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

Smith, P.L. (2000). Stochastic dynamic models of response time and accuracy: A foundational primer. *Journal of Mathematical Psychology*, *44*(3), 408–463. doi:10.1006/jmps.1999.1260

Stone, M. (1960). Models for choice-reaction time. *Psychometrika*, *25*(3), 251–260. doi:10.1007/BF02289729

Tuerlinckx, F. (2004). The efficient computation of the cumulative distribution and probability density functions in the diffusion model. *Behavior Research Methods*, *36*(4), 702–716.

Usher, M., & McClelland, J.L. (2001). The time course of perceptual choice: The leaky, competing accumulator model. *Psychological Review*, *108*(3), 550–592. doi:10.1037/0033-295X.108.3.550

Verdonck, S., & Tuerlinckx, F. (2014). The Ising Decision Maker: A binary stochastic network for choice response time. *Psychological Review*, *121*(3), 422–462. doi:10.1037/a0037012

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