A flexible framework for simulating and fitting generalized drift-diffusion models

Maxwell Shinn¹*, Norman H. Lam¹†, John D. Murray¹

¹Yale University, New Haven, CT

Abstract  The drift-diffusion model (DDM) is an important decision-making model in cognitive neuroscience. However, innovations in model form have been limited by methodological challenges. Here, we introduce the generalized drift-diffusion model (GDDM) framework for building, simulating, and fitting DDM extensions, and provide a software package which implements the framework. The GDDM framework augments traditional DDM parameters through arbitrary user-defined functions. Models are simulated numerically by directly solving the Fokker-Planck equation using efficient numerical methods, yielding a 100-fold or greater speedup over standard methodology. This speed allows GDDMs to be fit to data using maximum likelihood on the full response time (RT) distribution. We show that a GDDM fit with our framework explains a classic open dataset with better accuracy and fewer parameters than several DDMs implemented using the latest methodology. Overall, our framework will allow for decision-making model innovation and novel experimental designs.

Introduction  The drift-diffusion model (DDM) is an important model in cognitive neuroscience and is fundamental to our understanding of decision-making (Ratcliff, 1978; Bogacz et al., 2006). The DDM explains both choice and response time (RT) behavior across a wide range of tasks and species, and has proven to be an essential tool for studying the neural mechanisms of decision-making (Gold and Shadlen, 2007; Forstmann et al., 2016; Ratcliff et al., 2016). As a sequential sampling model for two-alternative forced-choice tasks, it assumes the subject obtains a continuous stream of evidence for each alternative throughout the course of an experimental trial. Evidence consists of some true underlying signal (drift) in addition to neural and environmental noise (diffusion). According to the DDM, once a decision variable tracking the difference in evidence between the two alternatives is sufficiently large, i.e. when the decision variable reaches some fixed upper or lower bound, a choice is made at the moment of bound crossing, thereby modeling for the decision both choice and RT. In its simplest form, the DDM includes three parameters: a drift rate, representing evidence; a non-decision time, representing efferent and afferent delay; and either a diffusion constant or bound height to represent noise (Ratcliff, 1978). These models can be fit to choice and response time data.

In order to test new hypotheses about decision-making processes, there is great interest in developing extensions to the DDM. For example, the three-parameter DDM is known to be sub-optimal with respect to maximizing reward rate in standard block-design tasks (Malhotra et al., 2017; Evans and Hawkins, 2019). Therefore, it has been hypothesized that decision-making is governed by an urgency signal which increases the probability of response over the course of a trial, implemented as either time-varying evidence gain (Cisek et al., 2009; Thura et al., 2012; Murphy...
et al., 2016; Ditterich, 2006b) or collapsing integration bounds (Churchland et al., 2008; Hawkins et al., 2015b; Ditterich, 2006b). Additionally, the DDM may be extended in order to model more diverse experimental paradigms. For example, evidence which changes in magnitude throughout a single trial violates the DDM’s assumption that drift rate is fixed for a single trial. Modeling such tasks requires DDM extensions to support time-varying drift rate. We will show that these and most other extensions can be encapsulated within a single generalized DDM (GDDM). The GDDM provides a common mathematical language for discussing DDM extensions, and as we will show, supports a fast and flexible framework for simulating models and fitting them to data.

There are three main methods for simulating the GDDM. First, analytical solutions are fast to simulate, but only exist for special cases of the GDDM. While efficient, most GDDMs do not permit analytical solutions, and thus analytical solutions are only common for the DDM. Several packages exist for analytically simulating DDMs and a limited number of extensions (Wiecki et al., 2013; Wagenmakers et al., 2007, 2008; Grasman et al., 2009; Wagenmakers et al., 2008; Vandekerckhove and Tuerlinckx, 2008; Voss and Voss, 2007; Drugowitsch, 2016; Tavares et al., 2017; Millner et al., 2018; Srivastava et al., 2017). A second method, trial-wise trajectory simulation, fully supports GDDMs for flexibility in model design (Chandrasekaran and Hawkins, 2019). However, this method is inefficient, and shows limited precision and computational efficiency. We will show that this in practice impacts the ability to fit data to RT distributions. To our knowledge, the only software package which currently exists for simulating GDDMs uses trial-wise trajectory simulation (Chandrasekaran and Hawkins, 2019).

A third and better method is distribution propagation, which is achieved by solving the Fokker-Planck equation. Distribution propagation allows GDDMs to be simulated with better performance and lower error than trial-wise trajectory simulation. However, it is difficult to implement distribution propagation, which has prevented its widespread use in neuroscience. Currently, GDDMs without known analytical solutions are best implemented using home-grown code (e.g. Zylberberg et al., 2016; Hawkins et al., 2015b; Brunton et al., 2013), which may not utilize the latest methodology or could cause misleading results (Ratcliff and Tuerlinckx, 2002; Ditterich, 2006a). Progress is limited by the ability to simulate and fit extensions to the DDM. A common open-source software package for efficiently simulating GDDMs would help promote code sharing and reuse, and reduce the amount of time which must be spent implementing diffusion models. Researchers need a general, highly-efficient toolbox for building, simulating, and fitting extended DDMs.

Here, we formally define and evaluate the GDDM framework for simulating and fitting extensions to the DDM, and introduce the PyDDM\(^1\) software package as an implementation of this framework. We show that GDDMs capture experimental data from a simple perceptual decision-making task in a classic open dataset better than alternative models. Then, we show the execution time vs error tradeoff induced by different numerical methods, and demonstrate that GDDMs may be simulated with short runtime and low error. In particular, GDDMs are simulated using the Crank-Nicolson and backward Euler methods to numerically solve the Fokker-Planck equation, which we will show to be highly efficient compared to alternative methods, and which permits fitting with continuous maximum likelihood. The GDDM together with its associated fitting methodology comprise the GDDM framework. Finally, we compare our implementation of the GDDM framework to other software packages for diffusion modeling. In addition to being the only package to support the GDDM framework, we show PyDDM has a number of technical advantages, such as near-perfect parallel efficiency, a graphical user interface for exploring model forms, and a software verification system for ensuring reliable results. We hope that the GDDM framework will encourage innovative experimental designs and lower the barrier for experimenting with new model mechanisms.

\(^1\)https://github.com/murraylab/PyDDM
Results

GDDM as a generalization of the DDM

The classic DDM is a three parameter model which specifies fixed values for drift, non-decision time, and either bound height or noise level (Figure 1). It assumes that evidence is constant throughout the trial, and that the integration process does not directly depend on time. This form is mathematically accessible, and has been used to model choice and RT data (Ratcliff, 1978; Ratcliff et al., 2016).

In order to expand our knowledge of decision-making mechanisms, there is interest in using the DDM to model a variety of experimental paradigms. To accommodate different paradigms, the DDM itself must be extended. One key example is the case where a choice bias is induced experimentally via a change in prior probability or reward magnitude. The DDM has been extended to accommodate an experimentally-induced bias through either a shift in the starting position of the integrator (Edwards, 1965; Laming, 1968; Ratcliff, 1985) or a constant drift bias (Mulder et al., 2012; Ratcliff, 1985; Ashby, 1983).

There is a set of influential yet insufficient extensions to the DDM called the “full DDM”. This set of extensions was developed to explain two specific discrepancies between the DDM and data (Anderson, 1960; Laming, 1968; Blurton et al., 2017). First, experimental data exhibited a difference in mean RT between correct and error trials which could not be captured by the classic DDM, so two parameters for trial-by-trial variability were introduced to explain this difference: uniformly-distributed initial conditions (Laming, 1968) and Gaussian-distributed drift rate (Ratcliff, 1978; Ratcliff and Rouder, 1998). Second, the classic DDM also had a sharper rise time than experimental data, so uniformly-distributed trial-by-trial variability in non-decision time was introduced to capture this effect. Compared to the classic DDM, the “full DDM” improves the fit to data, but does not expand the range of experiments which may be performed. There is also no way to extend the “full DDM” to extensions other than those described above. A more versatile framework is needed.

The GDDM framework provides a flexible means for developing extensions to the DDM (Figure 1) which can accommodate a wide range of experimental designs and hypotheses about decision-making. The GDDM allows for extensions in five key aspects. First, the drift rate and noise level may depend on time. This allows a range of new experimental paradigms to be modeled using the DDM, such as doubly-stochastic evidence (Zylberberg et al., 2016), pulsed evidence (Huk, 2005), discrete evidence (Cisek et al., 2009; Brunton et al., 2013; Pinto et al., 2018), or changing evidence (Shinn et al., 2019). Additionally, the time dependence of drift rate and noise allows a time-variant urgency signal which modulates the gain of evidence (Cisek et al., 2009; Thura et al., 2012; Murphy et al., 2016). Second, drift rate and noise level may depend on position. This allows for model features such as unstable (Wong, 2006; Roxin and Ledberg, 2008; Erlich et al., 2015) and leaky (Wong, 2006; Ossmy et al., 2013; Farashahi et al., 2018) integration. Leaky integration with a short time constant is functionally similar to the urgency-gated model with a low pass filter (Hawkins et al., 2015a; Thura and Cisek, 2014). Likewise, position-dependent variance may allow for multiplicative noise (Freyer et al., 2012). Third, the bound may change over time, which allows collapsing bounds according to an arbitrary function (Hawkins et al., 2015b; Evans and Hawkins, 2019), such as for an urgency signal. Fourth, models are parameterizable by an arbitrary dependence on fittable parameters or task conditions. These parameters may be shared between two related experimental tasks, such as an experimental and control task. It also allows alternative implementations of side bias (Hanks et al., 2011) and condition-dependent nonlinearities (Shinn et al., 2019). Fifth, single-trial parameter variability is supported within the GDDM framework for both starting point and non-decision time.

Mathematically, these concepts are expressed using five functions, corresponding to the drift rate, noise, bound height, starting position, and post-simulation modifications to the distribution (Equation 1). Collectively, the GDDM framework allows complicated models to be expressed in a consistent manner, simplifying the way which we may think about complicated DDM extensions.
Figure 1. Generalized DDM. The DDM has fixed, limited parameters, and requires a stream of constant evidence. The “full DDM” expands on the DDM by allowing uniformly-distributed starting position or non-decision time and Gaussian-distributed drift rate. The GDDM is a generalization of the DDM framework which allows arbitrary distributions for starting position and non-decision time, as well as arbitrary functions for drift rate and collapsing bounds. It also permits experimental paradigms with dynamic evidence. Several examples of potential GDDM mechanisms are shown.
GDDM improves fit to empirical data

To demonstrate the flexibility of GDDMs (Equation 1), we show that a GDDM fits a freely available dataset both more accurately and with fewer parameters than the DDM. Roitman and Shadlen (2002) presented two rhesus monkeys with a random dot motion task in which dots moved randomly but with varying strengths of coherent motion to the left or right (Figure Supplement 1). Motion coherence was randomly selected to be one of six conditions ranging from 0% to 51.2%. The monkeys were required to determine the direction of coherent motion and respond via saccade to flanking targets. Response time was recorded, which we fit with the standard DDM, “full DDM”, and GDDM.

We compared the GDDM to the “full DDM” and to three different implementations of the DDM which differ in simulation and fitting methodology. A full comparison of these three packages—PyDDM, HDDM, and EZ-Diffusion—as well other common packages is provided in the section “Software package for GDDM fitting”. The DDMs included terms for drift, non-decision time, and either bound height or noise magnitude. PyDDM assumed drift rate was a linear combination of some baseline drift rate and coherence (Equation 14), whereas HDDM fit separate drift rates for each coherence condition (Equation 16), and EZ-Diffusion fit all parameters fully independently for each coherence condition (Equation 17). We also fit a “full DDM” model, which included trial-by-trial variability in the drift rate, non-decision time, and starting position (Equation 15). Finally, we fit a GDDM, which consisted of a typical DDM with no trial-by-trial variability in the parameters, but utilizing leaky integration and an exponentially collapsing bound (Equation 13). The two different monkeys were fit using separate parameters for all models.

The fit of each model to the data is summarized in Figure 2 and Figure Supplement 2. First we examine the fit of the model quantitatively. Bayesian information criterion (BIC) is a common way of quantifying model fit while penalizing for free parameters. According to BIC, the 5 parameter GDDM fit better than all other models (Figure 2a). To determine whether this improved fit was due to the penalty BIC places on the number of parameters, we compared the models directly with likelihood. Again, we found that the GDDM outperformed models with many more parameters (Figure 2b). Finally, we tested whether this improved fit is due to the fact that PyDDM and HDDM use continuous maximum likelihood as an objective function, whereas EZ-Diffusion does not. We found that the mean squared error (MSE) of the GDDM was also lower than all other models (Figure 2c). These findings collectively provide strong evidence that the GDDM outperforms the DDM and the “full DDM”.

Additionally, we assessed the qualitative fit of the models to the data. All five models provided a good fit to empirical psychometric (Figure 2d and Figure Supplement 2d) and chronometric (Figure 2e and Figure Supplement 2e) functions. However, models varied substantially in their ability to fit the qualitative properties of the RT distribution. The standard DDMs fit with HDDM and PyDDM predicted a faster rise for the RT distribution at all coherence levels, and hence underestimated the mode of the distribution (Figure 2f and Figure Supplement 2f). The DDM fit with EZ-Diffusion overestimated the rise rate but also overestimated the non-decision time, creating a more accurate mode but an incorrect shape (Figure 2f and Figure Supplement 2f). The “full DDM” performed better than the DDMs, and managed to capture the general shape of the RT distribution at high coherences (Figure 2f and Figure Supplement 2f), in addition to low coherences for one monkey (Figure Supplement 2). However, the GDDM performed better and with fewer parameters than the “full DDM”, accurately predicting the rise rate, non-decision time, and overall distribution shape for all models under all conditions (Figure 2f, Figure Supplement 2f). These quantitative and qualitative comparisons demonstrate that the flexibility afforded by the GDDM framework has important practical implications for understanding decision-making.

Methods for simulating the DDM and GDDM

There are three broad classes of methods for simulating the GDDM. First, analytical expressions can be derived using stochastic calculus and sequential analysis (Wald, 1945; Anderson, 1960). An-
Figure 2. Fit of DDM, “full DDM”, and GDDM to Roitman and Shadlen (2002). Five models (see methods) were fit with three different software packages to data from monkey N. Model fits are compared according to BIC (a), negative log likelihood (b), and mean squared error (MSE) (c). The psychometric (d) and chronometric (e) functions represent data as dots and models as lines. The probability density function (PDF) (f) of each model is shown for 12.8% coherence, 6.4% coherence, 3.2% coherence, and 0% coherence. The data RT histogram is shown in gray, and black dots show the smoothed kernel density estimate probability distribution derived from the data (Scott, 2015). Columns of the figure and the color of the line denote different models.
Figure 3. GDDM simulating and fitting. (a) A schematic of trial-wise trajectory simulation methods. In a single trial, a subject’s decision process can be summarized by a stochastic decision variable (DV); simulating this single-trial decision variable many times results in a histogram approximation to the RT distribution. (b) A schematic of solving the Fokker-Planck equation. The full probability distribution of decision variables is solved at each time point, providing a numerical solutions with higher accuracy and reduced execution time.

Analytical expressions were favored historically because they can be computed quickly by humans or computers. However, apart from the simplest form of the DDM and a few particular classes of extensions, analytical solutions often do not exist. While analytical expressions offer maximum performance and accuracy, deriving new analytical expressions is not a practical way to rapidly implement and test new GDDMs.

The second method, trial-wise trajectory simulation, fully supports GDDM simulation. This method simulates the decision variable trajectories of many individual trials to sample RTs without producing a probability distribution (Figure 3a). Previous work has used trial-wise trajectories to simulate DDMs with no known analytical solution (e.g. Hawkins et al., 2015b). However, this flexibility comes at the cost of performance. As we show in section “Execution time vs error tradeoff”, trial-wise trajectory simulation is very slow and provides a coarse-grained representation of the model. Since this method does not produce a probability distribution, it is difficult or impossible to use efficient and robust methods such as continuous maximum likelihood for fitting data.

By contrast, the third method uses the Fokker-Planck equation to numerically compute the first passage time distribution. The Fokker-Planck equation is a mathematical expression which re-formulates the trial-by-trial diffusion process: rather than simulating single-trial decision variable trajectories, this method simulates all possible paths at once by propagating the distribution of single-trial decision variable trajectories over time (Figure 3b). Unlike analytical solutions, numerical solutions from the Fokker-Planck equation can be computed for all GDDMs. They also generate a probability density function for the RT distribution instead of a sample, allowing fitting with continuous likelihood methods. Several algorithms may be used to solve the DDM or GDDM using the Fokker-Planck equation, each with different advantages and disadvantages.

We consider three such algorithms: forward Euler, backward Euler, and Crank-Nicolson (see Appendix). These algorithms differ in how they propagate the probability distribution over time: forward Euler iteratively approximates the probability distribution of trajectory position at each timestep using the distribution at the previous timestep, while backward Euler and Crank-Nicolson solve a system of linear equations to compute the probability distribution. The three algorithms also differ in their conceptual and technical difficulty to implement, with forward Euler being easier than backward Euler or Crank-Nicolson. In section “Execution time vs error tradeoff”, we will show that the backward Euler and Crank-Nicolson algorithms are faster and provide higher accuracy than the forward Euler algorithm and trial-wise trajectory simulation.

Execution time vs error tradeoff
Next, we evaluated the effectiveness of different simulation methodologies. In general, when simulating models, two key methodological considerations are the precision with which the simulations are performed (accuracy), and the amount of time it takes the simulation to run on a computer (ex-
execution time). Accurate simulations are important to ensure the simulation faithfully represents
the model, and fast simulations are important to increase the feasibility of fitting models to data.
In general, for a given algorithm, there is a tradeoff between execution time and the accuracy of
the solution obtained. One algorithm may be more efficient than another algorithm if it produces
similar accuracy in a shorter amount of time, or better accuracy in the same amount of time.

We directly evaluate the execution time vs error tradeoff in trial-wise trajectory simulation and
in solving the Fokker-Planck equation. To do so, we consider a standard DDM with constant drift,
diffusion, and bounds (Equation 18). While this is not itself a GDDM, such a model has an analytical
solution which can be used as ground truth for computing the error in numerical simulations.

First, we test trial-wise trajectory simulation. We perform a parameter sweep across timesteps
Δt and sample sizes N. We simulate using the Euler-Maruyama method, which is standard in the
DDM literature (e.g. Hawkins et al., 2015a;b; Evans et al., 2017; Harty et al., 2017; Zhou et al.,
2009; Atiya et al., 2019; Simen et al., 2011; Nguyen et al., 2019; Lewis et al., 2014; Verdonck et al.,
2015) and for simple models provides identical truncation error but better execution time than
more sophisticated methods such as stochastic fourth-order Runge-Kutta (Gard, 1988; Brown et al.,
2006). To give the method the benefit of the doubt, for each simulation, we tried smoothing with
all mean filters of size 1 to 500 points and chose the filter with the lowest error, even though
such smoothing would not be possible for this method in practice due to the lack of analytical
ground truth for most GDDMs. Execution time and error are shown separately for a range of
numerical parameters (Figure 4a,b). As a proxy for the execution time vs error tradeoff, we also
show the product of execution time and mean squared error for the parameter sweep (Figure
Supplement 1).

In general, trial-wise trajectory simulation provides a suboptimal execution time vs error trade-
off. This tradeoff is maximized for highly imprecise models, i.e. those with a large timestep and a
small number of trajectories simulated. To examine the execution times and accuracies which lead
to this tradeoff, we fix Δt = 0.01 and plot the execution time and accuracy across many different
values of sample size N (Figure 4c). We see that as execution time increases, accuracy does not
increase by more than one order of magnitude. Examples of these traces are provided in Figure
Supplement 2.

Next, we examine methods for solving the Fokker-Planck equation. We consider three different
algorithms: forward Euler, backward Euler, and Crank-Nicolson (Table 1). The execution time vs
accuracy tradeoff of these three algorithms are governed by two parameters: the timestep Δt, and
the granularity of decision variable discretization Δx. Decreasing either of these two parameters
slows down the simulation but increases accuracy. These three algorithms operate in a distinct but
related manner. Forward Euler is the easiest algorithm to implement, however it is only numerically
stable for values of Δt which are very small relative to Δx. Forward Euler is an approximation in
the limit as Δt → 0, and thus cannot be run for large Δt step sizes relative to Δx (see Methods).
Backward Euler solves this problem by providing a numerically stable solution across all values of
Δt and Δx. While it provides a similar execution time and accuracy as forward Euler given Δt and
Δx, the fact that it is numerically stable across all values of these parameters allows more efficient
values of Δt and Δx to be chosen. Crank-Nicolson further improves the accuracy and execution
time of model simulation (Table 1). Like backward Euler, Crank-Nicolson does not have theoretical
constraints on Δx or Δt, but unlike backward Euler, it may experience numerical instability for
models with time-variant integration bounds. In practice, for each of these methods, the required
maximum step size is constrained by how strongly varying the drift and noise are in space and
time, where more varying drift would require a smaller step size. Convergence of the results can
be evaluated by checking results with smaller step sizes.

To evaluate the execution time vs error tradeoff, we perform a parameter sweep across values
of Δt and Δx for these three algorithms and again compute the product of execution time and mean
squared error as a proxy for execution time vs error tradeoff (Figure 4a,b, Figure Supplement 1).
Overall, the execution time vs error tradeoff is 1-6 orders of magnitude better for these algorithms.
Figure 4. Execution time vs error tradeoff. A DDM with constant drift, noise, and bounds was simulated at different spatial ($\Delta x$) and temporal ($\Delta t$) resolutions. Execution time (a) and MSE (b) are shown across values of $\Delta x$ and $\Delta t$. Darker colors indicate a more favorable execution time vs error tradeoff. Gray color indicates numerical instability. (c) Relationships between execution time and error which approximately maximize this tradeoff are shown. Parametric expressions were empirically determined to approximately maximize accuracy vs error tradeoff (see Figure Supplement 1), and are given by $\Delta t = 0.43\Delta x^2$ for forward Euler and $\Delta t = \Delta x$ for trial-wise trajectory simulation, backward Euler, and Crank-Nicolson. The lower left indicates low error and fast execution time whereas the upper right indicates slow execution time and high error.

**Figure 4–Figure supplement 1.** Execution time vs MSE tradeoff across parameters.

**Figure 4–Figure supplement 2.** Example simulations with different numerical parameters.

**Table 1.** Fokker-Planck simulation methods, in order of priority within PyDDM.

| Method                      | Time accuracy | Grid accuracy | Requirements                                           |
|-----------------------------|---------------|---------------|--------------------------------------------------------|
| Analytical                  | Exact         | Exact         | Time- and position-independent drift rate, diffusion constant, and bounds (or linearly collapsing bounds); centered point source initial condition |
| Crank-Nicolson              | $O(\Delta t^2)$ | $O(\Delta x^2)$ | Time independent bounds                                |
| Backward Euler              | $O(\Delta t)$  | $O(\Delta x^2)$ | None (fallback for all models)                        |
| Forward Euler               | $O(\Delta t)$  | $O(\Delta x^2)$ | Numerics $\Delta t/\Delta x^2 < 1/\sigma^2$ (not used by PyDDM) |
| Trial-wise trajectories     | N/A           | N/A           | No overlays except non-decision time (not used by PyDDM) |
than for trial-wise trajectory simulation. We find that, for backward Euler and Crank-Nicolson, the
equation time vs error tradeoff in the model shown in Figure Supplement 1 is maximized approx-
imately when $\Delta t = \Delta x$. Forward Euler is unstable in this parameterization, with the best execution
time vs error tradeoff when it is closest to its stability condition. We examine the execution time
and error for parameterizations which satisfy these constraints, and compare them to trajectory
simulation (Figure 4c). We see that Crank-Nicolson is strictly better than other methods; for a given
error, Crank-Nicolson offers a faster execution time, and for a given execution time, Crank-Nicolson
offers lower error. However, since Crank-Nicolson may not be appropriate for models with time-
variant boundaries, the second best method is also of interest. We see that, for a given execution
time, backward Euler is better than forward Euler by two orders of magnitude, and better than
trial-wise trajectory simulations by up to five orders of magnitude. Therefore, solving the Fokker-
Planck equation using Crank-Nicolson and backward Euler represents a key advantage over the
forward Euler and trial-wise trajectory simulation methods.

Software package for GDDM fitting
Overview of our software package
We developed a software package to encapsulate the methodological framework we have pre-
sented. Our package, PyDDM, allows the user to build models, simulate models, and fit them to
data using the efficient GDDM framework methodology described previously. The structure and
workflow of the package were designed to mirror the GDDM framework (Figure Supplement 1). Py-
DDM selects the most efficient approach to solve a GDDM by automatically determining whether
the model can be solved analytically and whether it uses time-varying bounds (Table 1). PyDDM is
also built to emphasize the modularity of the GDDM framework. Components of models may be
readily reused in other models, and to promote this, we additionally provide an online database of
models (the “PyDDM Cookbook”\(^2\)) to promote code sharing and replication. Users are encouraged
to submit their models to this database.

Furthermore, PyDDM is designed to accommodate the arbitrary flexibility of GDDMs. For reli-
able parameters estimation, it supports differential evolution (Storn and Price, 1997) as a global op-
timization method in addition to local search methods such as the Nelder-Mead simplex algorithm
(Nelder and Mead, 1965). Fits are performed using continuous maximum likelihood on the full
probability distribution in order to use all available data to estimate parameters. While continuous
maximum likelihood can be overly sensitive to outliers or “contaminants” (Ratcliff and Tuerlinckx,
2002), PyDDM allows fitting with an arbitrary parameterizable mixture model to reduce their im-
pact. Because some models can be difficult to understand conceptually, PyDDM also includes a
GUI for visualizing the impact of different model parameters (Figure Supplement 2), analogous to
a GDDM-compatible version of the tool by Alexandrowicz (2018).

Comparison to other software packages
Besides PyDDM, several software packages are available to simulate DDMs, and one package to
simulate GDDMs. We compare these in detail in Table 2, and below emphasize a few key points of
comparison.

The choice of software package depends on the model to be fit. As demonstrated in Figure 2 and
noted previously (Ratcliff, 2008), fitting a diffusion model to a small number of summary statistics,
as in EZ-Diffusion, leads to poor qualitative and quantitative model fit to the RT distribution. Better
options for fitting the DDM and “full DDM” are available, such as fast-dm and HDDM. However,
these packages are limited in model form, and make it difficult to extend beyond the “full DDM”.
As we have shown previously (Figure 2), the GDDM is desirable for obtaining good model fits with
few parameters.

\(^2\)https://pyddm.readthedocs.io/en/latest/cookbook/index.html
Table 2. Comparison of existing DDM packages. PyDDM is compared to HDDM (Wiecki et al., 2013), EZ-Diffusion (Wagenmakers et al., 2007, 2008), CHaRTr (Chandrasekaran and Hawkins, 2019), Diffusion Model Analysis Toolbox (DMAT) (Grasman et al., 2009; Wagenmakers et al., 2008; Vandekerckhove and Tuerlinckx, 2008), and fast-dm (Voss and Voss, 2007; Voss et al., 2015). Red indicates limited flexibility, yellow indicates moderate flexibility, and green indicates maximal flexibility. For the solver, these colors indicate minimal, moderate, and maximal efficiency.

|                      | PyDDM          | HDDM            | EZ-Diffusion    | CHaRTr          | DMAT            | fast-dm         |
|----------------------|----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| **Language**         | Python3        | Python2/3       | Matlab, R, Javascript, or Excel | Requires both R and C | Matlab          | Command line    |
| **Solver**           | Fokker-Planck, analytical | Analytical numerical hybrid | None | None | Analytical numerical hybrid | Analytical numerical hybrid |
| **Task parameters**  |                |                 |                 |                 |                 |                 |
| Time dependence of drift/noise | Any function | Constant | Constant | Any function | Constant | Constant |
| Position dependence of drift/noise | Any function | Constant | Constant | Any function | Constant | Constant |
| Bounds               | Any function   | Constant | Constant | Any function | Constant | Constant |
| Parameter dependence on task conditions | Any relationship for any parameter | Regression model | Categorical | Categorical | Linear | Categorical |
| **Across-trial variability** |               |                 |                 |                 |                 |                 |
| Across-trial drift variability | None | Normal distribution | None | Any distribution | Normal distribution | Normal distribution |
| Across-trial starting point variability | Any distribution | Uniform distribution | None | Any distribution | Uniform distribution | Uniform distribution |
| Across-trial non-decision variability | Any distribution | Uniform distribution | None | Any distribution | Uniform distribution | Uniform distribution |
| **Model simulation and fitting** |               |                 |                 |                 |                 |                 |
| Fitting methods      | Any numerical (default: differential evolution) | MCMC | Analytical | Any numerical | Nelder-Mead | Nelder-Mead |
| Objective function   | Any function (default: likelihood) | Likelihood | Mean/stdev RT and P(correct) | Any sampled (e.g. quantile maximum likelihood) | Quantile maximum likelihood or chi-squared | Likelihood, chi-squared, Kolmogorov-Smirnov |
| Mixture model        | Any distribution(s) | Uniform | None (extendable) | None | Uniform and undecided guesses | Uniform |
The major highlight of PyDDM compared to other packages is that it allows GDDMs to be simulated, and is thus compatible with time- and position-dependent drift rate and noise, time-varying bounds, starting point variability, and non-decision time variability. Besides PyDDM, the only other package to our knowledge capable of simulating GDDMs is the new R package CHaRTr (Chandrasekaran and Hawkins, 2019). This package is a major innovation compared to previous approaches, as it allows flexible model forms to be tested. However, CHaRTr utilizes trial-wise trajectory simulations for simulating models rather than distribution propagation via the Fokker-Planck equation. As shown previously (Figure 4), trial-wise trajectory simulation is especially inefficient, reducing the accuracy of the model simulations and increasing execution time by several orders of magnitude. In practice, this makes it infeasible to fit data to more flexible models. For the user to define new models, CHaRTr requires defining models in C and modifying the CHaRTr source code. By contrast, PyDDM facilitates the specification and implementation of models by defining them in user scripts written in pure Python.

Parallel performance
In addition to PyDDM’s efficient algorithms for solving GDDM models, PyDDM also includes the ability to further speed up simulations by solving models in parallel. Any model can be converted from single-threaded to multi-threaded with a single function call. PyDDM solves models for different conditions on multiple CPUs via the “pathos” library (McKerns et al., 2012). It uses a so-called “embarrassingly parallel” algorithm, which turns the most essential parts of the simulation into independent tasks and distributes them equally across CPUs. Such parallelization can be enabled in any PyDDM script with one line of code.

In a perfectly efficient parallelized environment, a simulation utilizing $N$ CPUs should be $N$ times faster than a simulation utilizing one CPU. In practice, this is rare due to communication latency and other overhead. We evaluate the speedup gained by solving an example DDM (Equation 19) on more than one CPU (Figure 5). As expected, the execution time decreases when more CPUs are used to perform the simulations. However, this speedup is strikingly linear (Figure 5a); for up to 20 CPUs, PyDDM achieves close to 100% parallel efficiency (Figure 5b). This means that PyDDM is able to effectively utilize multiple processors to decrease execution time. In practice, the authors routinely fit 15 parameter GDDMs on 8 CPUs with differential evolution for global parameter optimization in under 5 hours.

Discussion
The GDDM framework provides a consistent description of model extensions, and allows researchers to efficiently simulate and fit a wide variety of extensions to the DDM. Using an open dataset, we
demonstrated that GDDMs are able to fit data more effectively and with fewer parameters than existing methods. Models are simulated and fit numerically using multiple fast and accurate methods for solving the Fokker-Planck equation. We built the PyDDM software package to implement the GDDM framework. PyDDM makes it easy to utilize the efficiency of this framework, while providing additional technical advantages over other packages.

The major benefit of our GDDM framework is the flexibility and ease with which the model may be extended. The GDDM generalizes the DDM to support a wide variety of task paradigms and cognitive mechanisms. Any parameter in the standard DDM is allowed to be an arbitrary function of time and/or position when applicable. Mechanisms such as leaky or unstable integration, collapsing bounds, time-dependent drift rates, and complex non-decision time and starting point distributions all fall within the GDDM framework and can be readily simulated in PyDDM. GDDMs may be built modularly, eliminating the need to reimplement a separate model for each new mechanism. This modularity also helps promote code reuse through our online database of PyDDM model components.

GDDMs may be simulated by efficiently solving the Fokker-Planck equation. In our tests, trial-wise trajectory simulation performed poorly because large increases in the number of trials did not result in as large of increases in accuracy. While the smoothing procedure we applied improved accuracy, trial-wise trajectory simulation was unable to obtain low error for reasonable execution times. The Crank-Nicolson and backward Euler algorithms performed up to five orders of magnitude better than trial-wise trajectory simulation. This performance increase is substantial because the improvement leads to a qualitative difference in the types of models which may be fit to data. While inefficient methods are sufficient for simulating a single instance of the model, fitting models with many parameters to data requires thousands of simulations or more. By providing efficient algorithms for solving this broad class of models, our framework makes it possible to fit parameters to data for models which were previously prohibitively time consuming. In this way, it expands the range of potential cognitive mechanisms which can be tested and experimental paradigms which can be modeled.

In order to test such claims about (potentially complex) cognitive mechanisms, it is important that models are recoverable. That is, given a set of models, simulated data from each model should be best fit by the model and parameters that simulated it (Wiecki et al., 2013). This explicitly tests against model and parameter degeneracy, a known weakness of the “full DDM” (van Ravenzwaaij and Oberauer, 2009; Boehm et al., 2018). Individual GDDMs must be tested for model and parameter recovery before claims can be made about model mechanisms. PyDDM makes it straightforward to test whether a given GDDM has parameters which are recoverable, and whether any two given models can be distinguished from one another. This is facilitated because synthetic RT data can be sampled from a the calculated probability distributions of the GDDM without the need to run trial-wise trajectory simulations.

While the GDDM framework supports individual trial variability in non-decision time and starting position according to any arbitrary distribution, it does not support drift rate variability. As a result, the GDDM framework does not support the entire “full DDM”. Nevertheless, we have shown in Figure 2 that drift rate variability is not necessary to provide a good model fit (Lerche and Voss, 2016). Drift rate variability was initially introduced to support finite accuracy of responses (Ratcliff, 1978) and asymmetry in mean RT for correct and error trials (Ratcliff and Rouder, 1998). However, the GDDM framework is broad enough to support an infinite number of extensions which create the same RT features which originally motivated inclusion of drift rate variability. Indeed, in Figure 2, our 5-parameter GDDM provided a better fit to the asymmetry between correct and error RT distributions than the 11-parameter “full DDM”.

PyDDM also has the potential to support other applications of interest in the field. First, it could be used as a simulation engine for fitting hierarchical models of GDDM parameters, in which each subject’s individual parameters are drawn from a group distribution. The most critical component to hierarchical model fitting is an efficient method for simulating model likelihood. Since PyDDM
provides such a method, hierarchical fitting of GDDMs could be readily implemented as an extension to PyDDM. Second, PyDDM may be extended to support multi-dimensional diffusion, enabling it to represent interacting race models (Purcell et al., 2010; Usher and McClelland, 2001; Bogacz et al., 2006; Tsetsos et al., 2012). The Fokker-Planck equation can be solved in higher dimensions, in principle enabling similar highly efficient methods to be used for these models as well. The GDDM framework would require only minor adaptations to be compatible with multi-dimensional diffusion.

In summary, we show that the GDDM framework is an flexible and efficient way to explore new extensions on the DDM, allowing new possibilities in task design. Our package, PyDDM, offers a convenient and modular method for implementing this framework.

Methods
GDDM description
The GDDM is described as the following differential equation for decision variable $x$:

$$\frac{dx}{dt} = \mu(x,t,...)dt + \sigma(x,t,...)dW$$

(1)

where “...” represents task conditions and fitting parameters. Initial conditions are drawn from the distribution $x \sim X_0(\ldots)$. The process terminates when $|x(t,\ldots)| \geq B(t,\ldots)$. This can be parameterized by five functions:

- $\mu(x,t,...)$ – The instantaneous drift rate, as a function of the decision variable position, time, and task parameters. In PyDDM, it is represented by the “Drift” class and defaults to $\mu(x,t,...) = 0$.

- $\sigma(x,t,...)$ – The instantaneous noise, as a function of the decision variable position, time, and task parameters. In PyDDM, it is represented by the “Noise” class and defaults to $\sigma(x,t,...) = 1$.

- $X_0(\ldots)$ – The probability density function describing the initial position of the decision variable, as a function of task parameters. It must hold that $\sum x_i X_0(x_i,\ldots) = 1$. In PyDDM, it is represented by the “IC” class and defaults to a Kronecker delta function centered at $x = 0$.

- $B(t,...)$ – The boundaries of integration, as a function of time and task parameters. In PyDDM, it is represented by the “Bound” class and defaults to $B(t,...) = 1$.

- $p_c^*(t,...)$ and $p_e^*(t,...)$ – Post-processing performed on the simulated first passage time distributions of correct and error RT distributions $p_c(t)$ and $p_e(t)$, respectively. In PyDDM, these functions are represented by the “Overlay” class and default to $p_c^*(t,...) = p_c(t)$ and $p_e^*(t,...) = p_e(t)$.

In practice, “Overlay” is used to implement non-decision times and mixture models for fitting contaminant responses.

The organization of the code in PyDDM mirrors this structure (Figure Supplement 1).

Fokker-Planck formalism of the GDDM
PyDDM solves the GDDM using the Fokker-Planck equation, a partial differential equation describing the probability density $\rho(x,t,...)$ of the decision variable to be at position $x$ at time $t$, with initial condition $X_0(\ldots)$:

$$\frac{\partial}{\partial t} \rho(x,t,...) = -\frac{\partial}{\partial x} (\mu(x,t,...)\rho(x,t,...)) + \frac{\partial^2}{\partial x^2} \left( \frac{\sigma^2(x,t,...)}{2} \rho(x,t,...) \right).$$

(2)

By definition, the GDDM has absorbing boundary conditions ($\rho(x,t,...) = 0$ for all $x \leq -B(t,...)$ and $x \geq B(t,...)$), and any mass of the probability density crossing the boundaries are treated as committed, irreversible decisions.

The Fokker-Planck equation permits numerical solutions of the GDDM. The numerical algorithm is described for the GDDM with fixed bounds in this section, and the algorithm for the GDDM with time-varying bounds is described in the next section. The system is first discretized with space...
and time steps \( \Delta x \) and \( \Delta t \). Without loss of generality, we assume \( \Delta x \) divides \( 2B(t, \ldots) \) and \( \Delta t \) divides the simulated duration. The probability density solution of Equation 2, \( \rho(x, t, \ldots) \), can then be approximated as the probability distribution function at spatial grids \( \{-B(t, \ldots) + \Delta x, -B(t, \ldots) + 2\Delta x, \ldots, 0, \ldots, B(t, \ldots) - \Delta x, B(t, \ldots) - 2\Delta x, \ldots\} \) and temporal grids \( \{n\Delta t\} \) for any non-negative integer \( n \).

Due to absorbing bounds, the two spatial grids one step outward at \( \pm B(t, \ldots) \) have 0 density, and are not considered explicitly. The choice of \( \Delta t \) is constrained by the stability criteria and accuracy vs execution time tradeoff (see Figure 4).

Define \( P^m_i \) as the probability distribution at the \( i \)th space-grid and the \( m \)th time-grid. For clarity, here and below \( P^m_i \) denotes the total probability inside a grid of lengths \( \Delta x \) and \( \Delta t \), and its center represented by \((i,m)\). As such, \( P^m_i \) is unitless, in contrast to \( \rho(x,t) \) which has a unit of the inverse of the spatial dimension. Probability distribution functions can also be viewed as a discrete proxy of the continuous probability density (i.e. approximating \( \rho(x,t) \) at the corresponding space and time grid), which will then have a unit of the inverse of the spatial dimension. Both views are mathematically equivalent (and just off by a \( \Delta x \) factor to the whole equation), but the former definition was used for simplicity and interpretability.

The Fokker-Planck equations can be discretized in several distinct ways, including the forward Euler method, the backward Euler method, and the Crank-Nicolson method. In the forward Euler method, Equation 2 is approximated as:

\[
\frac{P^n_j - P^{n-1}_j}{\Delta t} = -\frac{1}{2\Delta x} \left( (\mu P)_{j+1}^{n-1} - (\mu P)_{j-1}^{n-1} \right) + \frac{\Delta t}{\Delta x^2} \left( (DP)_{j+1}^{n-1} + (DP)_{j-1}^{n-1} - 2(DP)_{j}^{n-1} \right)
\]

where \( D = \sigma^2/2 \) is the diffusion coefficient. Here and below, dependencies of space, time, and other variables are omitted for simplicity. Terms in parenthesis with subscripts and superscripts are evaluated at the subscripted space grid and superscripted time grid, in the same manner as \( P^n_j \). The spatial derivatives at the right side of Equation 2 are evaluated at the previous time step \( n-1 \), such that only one term of the equation is at the current time step \( n \), generated from the temporal derivative at the left side of Equation 2. As such, Equation 3 for each \( j \) fully determines \( P^n_j \) from the probability distributions at the previous time-step:

\[
P^n_j = P^{n-1}_j - \frac{\Delta t}{2\Delta x} \left( (\mu P)_{j+1}^{n-1} - (\mu P)_{j-1}^{n-1} \right) + \frac{\Delta t}{\Delta x^2} \left( (DP)_{j+1}^{n-1} + (DP)_{j-1}^{n-1} - 2(DP)_{j}^{n-1} \right)
\]

It is important to note that the forward Euler method, Equation 4, is an approximation method. Since the probability distribution is iteratively propagated from early to later times, any errors of the solution accumulate over the iterative process. Moreover, the forward Euler method is prone to instability. Assuming small \( \Delta x \) and \( \Delta t \) (i.e. much less than 1), \( \frac{\Delta t}{\Delta x} \) is much larger than \( \frac{\Delta t}{2\Delta x} \) and thus the third term on the right side of Equation 4 should in general be much larger than the second term. If \( \Delta x \) and \( \Delta t \) are such that the third term is large for any \( n \) and \( j \), then \( P^n_j \) will not be properly computed in Equation 4, and such errors will propagate to the whole space over time. The forward Euler method thus has a stability criteria:

\[
D \frac{\Delta t}{\Delta x^2} < \frac{1}{2}
\]

Equation 5 poses a strict constraint on the step-sizes: while \( \Delta x \) and \( \Delta t \) should both be small to minimize the error due to discretization, decreasing \( \Delta x \) demands a much stronger decrease in \( \Delta t \), resulting in a much longer execution time of the method. This can be seen in our simulations shown in Figure 4.

The backward Euler and Crank-Nicolson methods largely circumvent the aforementioned problems. In the backward Euler method, the right side of Equation 2 is expressed at the current time step \( n \):

\[
\frac{P^n_j - P^{n-1}_j}{\Delta t} = -\frac{1}{2\Delta x} \left( (\mu P)_{j+1}^{n} - (\mu P)_{j-1}^{n} \right) + \frac{\Delta t}{\Delta x^2} \left( (DP)_{j+1}^{n} + (DP)_{j-1}^{n} - 2(DP)_{j}^{n} \right)
\]
In the Crank-Nicolson method, half of the right side of *Equation 2* is expressed at the current time step \( n \), and half at the previous time step \( n - 1 \):

\[
\frac{P^n_j - P^{n-1}_j}{\Delta t} = 0.5 \left( -\left( \frac{\mu P^{n-1}_{j+1} + \mu P^{n-1}_{j-1}}{2\Delta x} + \frac{(DP)^{n-1}_{j+1} + (DP)^{n-1}_{j-1} - 2(DP)^{n-1}_{j}}{\Delta x^2} \right) \right) + 0.5 \left( -\left( \frac{\mu P^n_{j+1} + \mu P^n_{j-1}}{2\Delta x} + \frac{(DP)^n_{j+1} + (DP)^n_{j-1} - 2(DP)^n_{j}}{\Delta x^2} \right) \right).
\]  

(7)

With multiple terms at the current time-step, *Equation 6* and *Equation 7* cannot be directly solved. Instead, the equations across all spatial grids can be summarized in matrix form:

\[
(1 + (1 - a)\Delta t)\vec{P}^n = (1 - a\Delta t)\vec{P}^{n-1}
\]

(8)

for \( \vec{P}^n = \begin{pmatrix} p^n_{x-B+\Delta x} \\ \vdots \\ p^n_{x=B-\Delta x} \end{pmatrix} \), \( M = \begin{pmatrix} 2\nu & -\nu + \chi/2 & 0 & 0 & 0 \\ -\nu - \chi/2 & 2\nu & -\nu + \chi/2 & 0 & 0 \\ 0 & -\nu - \chi/2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & \ddots & \ddots & -\nu + \chi/2 \\ 0 & 0 & 0 & -\nu - \chi/2 & 2\nu \end{pmatrix} \).

Here, \( \nu = D\Delta t/\Delta x^2 \) and \( \chi = \mu\Delta t/\Delta x \). In this formulation, \( a = 1 \) for the forward Euler method, \( a = 0 \) for the backward Euler method, and \( a = 0.5 \) for the Crank-Nicolson method. Since \( M \) is a tridiagonal matrix, it can be inverted from the left side to the right side of the equation in an efficient manner. Practically, this allows the backward Euler and Crank-Nicolson method to rapidly generate the discretized probability distribution function inside the boundaries at each time-step.

Finally, the probability of decision formation at each time step is computed as the outward flux from the outermost grids \( \pm B \pm \Delta x \) (to the hypothetical grids at \( \pm B \)):

\[
\Delta p^n_{\text{correct}} = (-\nu - \chi/2)p^n_{x-B-\Delta x}|_{s=x-B-\Delta x} \\
\Delta p^n_{\text{incorrect}} = (-\nu + \chi/2)p^n_{x-B-\Delta x}|_{s=x-B-\Delta x}.
\]

(9)

This provides the correct and incorrect reaction time distributions, as well as the total correct \((\sum \Delta p^n_{\text{correct}})\) and incorrect \((\sum \Delta p^n_{\text{incorrect}})\) probability.

Unlike the forward-Euler method which iteratively propagates the solution and the approximation error in a feedforward manner, the backward Euler and Crank-Nicolson methods solve for the matrix equation of the probability distribution evolution, and are less susceptible to error being propagated. In addition, the two methods do not have any stability criteria which constrain the step size. Comparing the two methods, the backward Euler method has larger truncation error \((O(\Delta x, \Delta t^2))\) than the Crank-Nicolson method \((O(\Delta x^2, \Delta t^2))\) (Table 1), while the Crank-Nicolson method generally has twice the execution time of the backward Euler method, assuming the matrix construction is the slowest step of the algorithms. Crucially, the solution from the Crank-Nicolson method tends to be susceptible to oscillations over iterations, especially with varying bounds (see below). PyDDM automatically chooses the best solver for any given model (Table 1).
Fokker-Planck formalism with time-varying bounds

The previous section considered the Fokker-Planck equation of the GDDM assuming fixed bounds. The formalism can be adapted to accommodate time-varying bounds. This is more generally studied as a free boundary problem, with a typical example including the Stefan problem (Meyer, 1978; Li, 1997).

In particular, consider upper and lower bounds $B_1 = B(n\Delta t, \ldots)$ and $-B_1 = -B(n\Delta t, \ldots)$ at an arbitrary time $t = n\Delta t$. If $B_1$ is exactly on a grid point, the previous results apply. Alternatively, if $B_1$ is in between 2 grid points $(x_j, x_{j+1})$, the solution is approximated by a linearly weighted summation of the solutions in two grid systems: an inner grid system $\{x_j | k = -j, \ldots, j\}$ and an outer grid system $\{x_j | k = -j, \ldots, j + 1\}$.

Define $\tilde{P}_n^{\text{in/out}}$ to be the probability distribution function at time $t = n\Delta t$ over the inner and outer grid systems, respectively. In the backward Euler method ($\alpha = 0$ in Equation 8), $\tilde{P}_n^{\text{in/out}}$ can be computed by propagating from the actual probability distribution function at the previous step $\tilde{P}_n^{\text{in/out}}$:

$$\tilde{P}_n^{\text{in/out}}(\mathbf{l} + M_{\text{in/out}}) = \tilde{P}_{n-1}^{\text{in/out}},$$

where $M_{\text{in/out}}$ are as in Equation 8, but defined over the domain of inner/outer grids respectively.

Furthermore, define

$$w_{\text{out}} = \frac{B_{n} - x_j}{\Delta x}$$
$$w_{\text{in}} = \frac{x_{j+1} - B_{n}}{\Delta x}$$

as the weights of the linear approximation of the solution to that of the outer and inner grid systems. The resulting probability distribution function and the probabilities of decision formation at step $n$ are:

$$\tilde{P}_n = w_{\text{out}} \tilde{P}_n^{\text{out}} + w_{\text{in}} \tilde{P}_n^{\text{in}}$$
$$\Delta P_n^{\text{correct}} = w_{\text{out}} (-\nu + \chi/2)_{x \in [x_j, x_{j+1}]} \tilde{P}_n^{\text{out}} + w_{\text{in}} (-\nu + \chi/2)_{x \in [x_j, x_{j+1}]} \tilde{P}_n^{\text{in}}$$
$$\Delta P_n^{\text{incorrect}} = w_{\text{out}} (-\nu - \chi/2)_{x \in [x_j, x_{j+1}]} \tilde{P}_n^{\text{out}} + w_{\text{in}} (-\nu - \chi/2)_{x \in [x_j, x_{j+1}]} \tilde{P}_n^{\text{in}}$$

In the literature of the DDM and neuroscience, collapsing bounds are the most common form of time-varying bounds considered. As bounds collapse, it is possible that the outermost grids at the previous step (with non-zero probability densities) become out-of-bound at the current step. Based on the side they become out of bound, such probabilities are added to the correct and incorrect probabilities respectively (and multiplied by $w_{\text{out/in}}$ for each of the outer/inner grid system). Nevertheless, increasing bounds are also supported. The algorithm for the Crank-Nicolson scheme of time-varying bounds often generates oscillatory solutions, and is thus not described here or implemented in PyDDM.

**Compared models**

The 5 parameter GDDM fit by PyDDM is parameterized by

$$\mu(x, t, \ldots) = \mu_0 C - \varepsilon x$$
$$\sigma(x, t, \ldots) = 1$$
$$X_0(x, \ldots) = \begin{cases} 1, & x = 0 \\ 0, & \text{otherwise} \end{cases}$$
$$B(t, \ldots) = B_0 e^{-\gamma t}$$

where fittable parameters are drift rate $\mu_0$, leak constant $\varepsilon$, initial bound height $B_0$, bound collapse rate $\gamma$, and non-decision time $t_{\text{nd}}$. 

17 of 26
The three-parameter DDM fit by PyDDM is a subset of the GDDM framework parameterized by

\[ \mu(x, t, \ldots) = \mu_0 C \]
\[ \sigma(x, t, \ldots) = 1 \]
\[ X_0(x, \ldots) = \begin{cases} 1, & x = 0 \\ 0, & \text{otherwise} \end{cases} \]
\[ B(t, \ldots) = B_0 \]
\[ p^*_i(t) = 0.98 p_i(t - t_{nd}) + 0.01 e^{-(t - t_{nd})} \]  

where \( C \) is coherence and \( \mu_0, B_0, \) and \( t_{nd} \).

The 11 parameter “full DDM” fit by HDDM does not fall into the GDDM framework because the trial-wise variability in drift rate \( \mu(x, t, \ldots) \) is a random variable. Thus, the “full DDM” is parameterized by

\[ \mu(x, t, \ldots) \sim N(\mu_j, s^2_v) \]
\[ \sigma(x, t, \ldots) = 1 \]
\[ X_0(x, \ldots) = \begin{cases} 1, & x = 0 \\ \frac{1}{2s_x + 1}, & \text{otherwise} \end{cases} \]
\[ \begin{align*} s_x < x < s_z, & \text{ otherwise} \\ B(t, \ldots) = B_0 \end{align*} \]
\[ p^*_i(t) = \frac{1}{2s_x + 1} \sum_{k \in [-s_t, s_t]} p_i(t - t_{nd} - k) \]

where \( N \) is a normal distribution, and \( \mu_0, B_0, \) and \( t_{nd} \) are six independent drift rates \( \mu_j \) for each coherence level \( j \), non-decision time \( t_{nd} \), and bounds \( B_0 \) as well as variability in drift rate \( s_v \), starting position \( s_x \), and non-decision time \( s_t \).

The 8 parameter DDM fit by HDDM is a subset of the GDDM framework parameterized by

\[ \mu(x, t, \ldots) = \mu_j \]
\[ \sigma(x, t, \ldots) = 1 \]
\[ X_0(x, \ldots) = \begin{cases} 1, & x = 0 \\ 0, & \text{otherwise} \end{cases} \]
\[ B(t, \ldots) = B_0 \]
\[ p^*_i(t) = p_i(t - t_{nd}) \]

where \( \mu_0, t_{nd}, \) and six independent drift rates \( \mu_j \) one for each coherence level \( j \).

The 18 parameter DDM fit by EZ-Diffusion is a subset of the GDDM framework parameterized by

\[ \mu(x, t, \ldots) = \mu_j \]
\[ \sigma(x, t, \ldots) = 1 \]
\[ X_0(x, \ldots) = \begin{cases} 1, & x = 0 \\ 0, & \text{otherwise} \end{cases} \]
\[ B(t, \ldots) = B_C \]
\[ p^*_i(t) = p_i(t - t_{nd}) \]
where fittable parameters are independent drift rates $\mu_j$, bounds $B_j$, and non-decision times $t'_{nd}$ for each of the six coherence level $j$.

All simulations were fit separately for each monkey.

**Execution time and accuracy analysis**

To analyze the execution time vs error tradeoff, we benchmarked a DDM which fits into the GDDM framework, defined by:

$$
\mu(x, t, \ldots) = 2 \\
\sigma(x, t, \ldots) = 1.5 \\
X_0(x, \ldots) = \begin{cases} 
1, & x = 0 \\
0, & \text{otherwise} 
\end{cases} \\
B(t, \ldots) = 1 \\
p^*_i(t) = p_i(t)
$$

Performance was measured on a Lenovo T480 computer with an Intel M540 i7-8650U 4.2GHz CPU with hyperthreading disabled. Verification and parallelization were disabled for these simulations.

In order to examine combinations of $\Delta t$ and either $\Delta x$ or $N$ which simultaneously minimize both execution time and error, we plot the product of the MSE and execution time in seconds. This product should be interpreted as a heuristic which encompasses both execution time and accuracy rather than a fundamentally important quantity in and of itself.

**Model for parallel performance**

To analyze PyDDM’s parallel performance, we benchmarked a DDM which fits into the GDDM framework, defined by:

$$
\mu(x, t, \ldots) = 2C \\
\sigma(x, t, \ldots) = 1 \\
X_0(x, \ldots) = \begin{cases} 
1, & x = 0 \\
0, & \text{otherwise} 
\end{cases} \\
B(t, \ldots) = 1 \\
p^*_i(t) = p_i(t)
$$

where the model was simulated using backward Euler for 64 coherence values $C$ uniformly spaced from 0 to 0.63. Because PyDDM parallelizes functions by utilizing an embarrassingly parallel approach over experimental conditions, we included many such conditions to enable benchmarks which reflect the simulations instead of the specifics of the model. Simulations were performed on a high performance computing cluster using a single node with 28 Xeon E5-2660v4 cores. Verification was disabled for these simulations.

**Validity of results**

It is important that simulation results are valid and reliable. Results could be inaccurate either due to bugs in user-specified models or in PyDDM itself. Due to the ability to easily construct user-defined models, PyDDM takes this issue very seriously. In addition to utilizing standard software engineering practices such as unit and integration testing with continuous integration (Kent Beck, 2004), PyDDM is the first neuroscience software to our knowledge which incorporates software verification methods for ensuring program correctness (Ghezzi et al., 2002). It does so using a library...
which checks for invalid inputs and outputs within the code, and also checks for invalid inputs and outputs in user-defined models which do not explicitly incorporate the library (Shinn, 2019). Invalid inputs and outputs include anything which invalidates the assumptions of the GDDM, such as initial conditions outside of the bounds, noise levels less than zero, or probability distributions which do not sum to 1. Such checking helps ensure that results generated by PyDDM are not due to a configuration or programming mistake, which is especially important because user-specified models logic can quickly become complicated for intricate experimental designs. This, combined with the ease of reproducibility and code reuse afforded by open source software in general, means that high validity and reliability are key advantages of PyDDM.

Acknowledgments

We thank Robert Yang for his contribution to the analytic solver, and Daeyeol Lee and Hyojung Seo for helpful discussions.
Implementing and fitting models in PyDDM

Briefly, we describe the process of building and fitting a model in PyDDM. For more detailed information, see the quickstart guide in the documentation at https://pyddm.readthedocs.io/en/latest/quickstart.html. A familiarity with Python and with Python classes is assumed.

PyDDM is based on a modular architecture, organized around the principles of the GDDM. There are five components, specified using Python classes, which correspond to the five functions listed in section “GDDM description”:

- Drift: the drift rate. May depend on time, decision variable position, task conditions, and fittable parameters. Defaults to 0.
- Noise: the standard deviation of diffusion process. May depend on time, decision variable position, fittable parameters, and task conditions. Defaults to 1.
- Bound: The integration bounds (+B and -B). May depend on time, fittable parameters, and task conditions. Defaults to 1.
- IC: The initial conditions at time t=0, in terms of a probability density function over decision variable position. May depend on fittable parameters or task conditions. Defaults to a probability density function of a Kronecker delta function centered at $x = 0$.
- Overlay: Post-simulation modifications to the histogram, for instance to create a mixture model or add a non-decision time. May depend on fittable parameters or task conditions. Defaults to an identity function.

These model components do not specify a single value for these parameters, but rather specify a function through which these values can be computed. Each may take any number of parameters and utilize any number of task variables. Most models will not implement all of these components from scratch, but will use the default component or a built-in component for most of them. For example, the GDDM in Figure 2 (Equation 13) used the default “Noise” and “IC” components, built-in Bounds and Overlay components, and a customized “Drift” component. A full list of built-in components, as well as a library of example components, can be found in the online documentation.

Each of these model components may depend on parameters; for example, the “Bound” component requires the parameters $B_0$ and $\tau$. These parameters must be specified when the class for the model component is instantiated, for example, “BoundCollapsingExponential” is the class for exponential collapsing bounds used in the GDDM in Figure 2 (Equation 13). It can be instantiated as

```python
BoundCollapsingExponential(B0=2, tau=3)
```

for $B_0 = 2$ and $\tau = 3$.

Of course, we would often like to fit these parameters to data, and thus, we cannot specify their exact values when instantiating a component. For these situations, we use an instance of a “Fittable” class as a placeholder. Minimum and maximum values must be specified for bounded fittable algorithms, such as differential evolution. For example, we could fit both of these parameters using

```python
BoundCollapsingExponential(B0=Fittable(minval=0.5, maxval=3),
                           tau=Fittable(minval=.1, maxval=10)))
```

Components which are not built-in to PyDDM may be specified by defining a class which gives four pieces of information: the component’s name, the parameters it depends on, the
task conditions it depends on, and a function to compute its value. For example, here is
“DriftCoherenceLeak”, the “Drift” class used in our GDDM in Figure 2 (Equation 13):

```python
class DriftCoherenceLeak(ddm.models.Drift):
    name = "Leaky drift depends linearly on coherence"
    # Parameters we want to include in the model
    required_parameters = ["driftcoh", "leak"]
    # Task parameters/conditions. Should be the same name as in the sample.
    required_conditions = ["coh"]
    # The get_drift function is used to compute the instantaneous value of drift.
    def get_drift(self, x, conditions, **kwargs):
        return self.driftcoh * conditions["coh"] + self.leak * x
```

Once we have determined all of the components of our model, we may tie them together
using an instance of the “Model” class. When creating an instance of the “Model” class,
we must also define our timestep “dx”, grid spacing “dt”, and simulation duration “T_dur”.
Shown below is the GDDM in Figure 2 (Equation 13):

```python
m = Model(name=’Roitman−Shadlen GDDM’,
    drift=DriftCoherenceLeak(driftcoh=Fittable(minval=0, maxval=20),
                            leak=Fittable(minval=-10, maxval=10)),
    # Do not specify noise=... since we want the default
    # Do not specify IC=... since we want the default
    bound=BoundCollapsingExponential(B=Fittable(minval=0.5, maxval=3),
                                      tau=Fittable(minval=.0001, maxval=5)),
    # OverlayChain strings together multiple overlays.
    overlay=OverlayChain(overlays=[
        OverlayNonDecision(nondectime=Fittable(minval=0, maxval=.4)),
        OverlayPoissonMixture(pmixturecoef=.02, rate=1)
    ],
    dx=.001, dt=.001, T_dur=2)
```

If the model “m” includes at least one component with a “Fittable” instance, we must
first fit the model to data by calling the “fit_adjust_model” function, passing our model and
dataset as arguments. By default, it fits using differential evolution (Storn and Price, 1997),
however Nelder-Mead (Nelder and Mead, 1965), BFGS (Nocedal and Wright, 2006), basin
hopping (Wales and Doye, 1997), and hill climbing (Luke, 2013) are also available. Fitting a
model in parallel using n cores of a CPU is as simple as calling the function “set_N_cpus(n)”
before fitting the model. Once we load our sample “samp” using one of a number of conve-
nience functions for loading data, we may fit the model:

```python
set_N_cpus(4)
fit_adjust_model(m, sample=samp)
```

After fitting the model to data, the “Fittable” instances are automatically replaced by
their fitted values, so the model can be directly simulated using “m.solve()”. If any of the
model components depend on task conditions, these must be passed as a dictionary to
the solver; for example, since the “DriftCoherenceLeak” component depends on “coh” (the
trial’s coherence), we must instead call

```python
m.solve(conditions={"coh": 0.064})
```

to simulate with a coherence of 0.064. Alternatively, the function “solve_partial_conditions”
can simulate across many task conditions simultaneously. Both the functions “m.solve” and
“solve_partial_conditions” will return an instance of the “Solution” class, which has methods for obtaining the correct and error RT distributions as well as useful summary statistics.

References
Alexandrowicz, R. W. (2018). The diffusion model visualizer: an interactive tool to understand the diffusion model parameters. Psychological Research.

Anderson, T. W. (1960). A modification of the sequential probability ratio test to reduce the sample size. The Annals of Mathematical Statistics, 31(1):165–197.

Ashby, F. (1983). A biased random walk model for two choice reaction times. Journal of Mathematical Psychology, 27(3):277–297.

Atiya, N. A. A., Zgonnikov, A., O’Hora, D., and Wong-Lin, K. (2019). Changes-of-mind in the absence of new post-decision evidence. Biorxiv.

Blurton, S. P., Kesselmeier, M., and Gondan, M. (2017). The first-passage time distribution for the diffusion model with variable drift. Journal of Mathematical Psychology, 76:7–12.

Boehm, U., Annis, J., Frank, M. J., Hawkins, G. E., Heathcote, A., Kellen, D., Krypotos, A.-M., Lerche, V., Logan, G. D., Palmeri, T. J., van Ravenzwaaij, D., Servant, M., Singmann, H., Starns, J. J., Voss, A., Wiecki, T. V., Matzke, D., and Wagenmakers, E.-J. (2018). Estimating across-trial variability parameters of the diffusion decision model: Expert advice and recommendations. Journal of Mathematical Psychology, 87:46–75.

Bogacz, R., Brown, E., Moehlis, J., Holmes, P., and 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.

Brown, S. D., Ratcliff, R., and Smith, P. L. (2006). Evaluating methods for approximating stochastic differential equations. Journal of Mathematical Psychology, 50(4):402–410.

Brunton, B. W., Botvinick, M. M., and Brody, C. D. (2013). Rats and humans can optimally accumulate evidence for decision-making. Science, 340(6128):95–98.

Chandrasekaran, C. and Hawkins, G. E. (2019). CHaRTr: An r toolbox for modeling choices and response times in decision-making tasks. Biorxiv.

Churchland, A. K., Kiani, R., and Shadlen, M. N. (2008). Decision-making with multiple alternatives. Nature Neuroscience, 11(6):693–702.

Cisek, P., Puskas, G. A., and El-Murr, S. (2009). Decisions in changing conditions: The urgency-gating model. Journal of Neuroscience, 29(37):11560–11571.

Ditterich, J. (2006a). Evidence for time-variant decision making. European Journal of Neuroscience, 24(12):3628–3641.

Ditterich, J. (2006b). Stochastic models of decisions about motion direction: Behavior and physiology. Neural Networks, 19(8):981–1012.

Drugowitsch, J. (2016). Fast and accurate monte carlo sampling of first-passage times from wiener diffusion models. Scientific Reports, 6(1).

Edwards, W. (1965). Optimal strategies for seeking information: Models for statistics, choice reaction times, and human information processing. Journal of Mathematical Psychology, 2(2):312–329.

Erlich, J. C., Brunton, B. W., Duan, C. A., Hanks, T. D., and Brody, C. D. (2015). Distinct effects of prefrontal and parietal cortex inactivations on an accumulation of evidence task in the rat. eLife, 4.

Evans, N. J. and Hawkins, G. E. (2019). When humans behave like monkeys: Feedback delays and extensive practice increase the efficiency of speeded decisions. Cognition, 184:11–18.

Evans, N. J., Hawkins, G. E., Boehm, U., Wagenmakers, E.-J., and Brown, S. D. (2017). The computations that support simple decision-making: A comparison between the diffusion and urgency-gating models. Scientific Reports, 7(1).
Farashahi, S., Ting, C.-C., Kao, C.-H., Wu, S.-W., and Soltani, A. (2018). Dynamic combination of sensory and reward information under time pressure. *PLOS Computational Biology*, 14(3):e1006070.

Forstmann, B., Ratcliff, R., and Wagenmakers, E.-J. (2016). Sequential sampling models in cognitive neuroscience: Advantages, applications, and extensions. *Annual Review of Psychology*, 67(1):641–666.

Freyer, F., Roberts, J. A., Ritter, P., and Breakspear, M. (2012). A canonical model of multistability and scale-invariance in biological systems. *PloS Computational Biology*, 8(8):e1002634.

Gard, T. C. (1988). *Introduction to Stochastic Differential Equations. Monographs and Textbooks in pure and applied mathematics*. Dekker, Inc.

Ghezzi, C., Jazayeri, M., and Mandrioli, D. (2002). *Fundamentals of Software Engineering*. Pearson.

Gold, J. I. and Shadlen, M. N. (2007). The neural basis of decision making. *Annual Review of Neuroscience*, 30(1):535–574.

Grasman, R. P., Wagenmakers, E.-J., and van der Maas, H. L. (2009). On the mean and variance of response times under the diffusion model with an application to parameter estimation. *Journal of Mathematical Psychology*, 53(2):55–68.

Hanks, T. D., Mazurek, M. E., Kiani, R., Hopp, E., and Shadlen, M. N. (2011). Elapsed decision time affects the weighting of prior probability in a perceptual decision task. *Journal of Neuroscience*, 31(17):6339–6352.

Harty, S., Murphy, P. R., Robertson, I. H., and O’Connell, R. G. (2017). Parsing the neural signatures of reduced error detection in older age. *NeuroImage*, 161:43–55.

Hawkins, G. E., Forstmann, B. U., Wagenmakers, E.-J., Ratcliff, R., and Brown, S. D. (2015a). Revisiting the evidence for collapsing boundaries and urgency signals in perceptual decision-making. *Journal of Neuroscience*, 35(6):2476–2484.

Hawkins, G. E., Wagenmakers, E.-J., Ratcliff, R., and Brown, S. D. (2015b). Discriminating evidence accumulation from urgency signals in speeded decision making. *Journal of Neurophysiology*, 114(1):40–47.

Huk, A. C. (2005). Neural activity in macaque parietal cortex reflects temporal integration of visual motion signals during perceptual decision making. *Journal of Neuroscience*, 25(45):10420–10436.

Kent Beck, C. A. (2004). *Extreme Programming Explained*. Addison Wesley.

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

Lerche, V. and Voss, A. (2016). Model complexity in diffusion modeling: Benefits of making the model more parsimonious. *Frontiers in Psychology*, 7.

Lewis, M., Fedor, A., Öllinger, M., Szathmáry, E., and Fernando, C. (2014). Modelling reaction times in non-linear classification tasks. In *From Animals to Animats 13*, pages 53–64. Springer International Publishing.

Li, Z. (1997). Immersed interface methods for moving interface problems. *Numerical Algorithms*, 14(4):269–293.

Luke, S. (2013). *Essentials of Metaheuristics*. Lulu.

Malhotra, G., Leslie, D. S., Ludwig, C. J. H., and Bogacz, R. (2017). Time-varying decision boundaries: insights from optimality analysis. *Psychonomic Bulletin & Review*, 25(3):971–996.

McKerns, M. M., Strand, L., Sullivan, T., Fang, A., and Aivazis, M. A. G. (2012). Building a framework for predictive science. *Proceedings of the 10th Python in Science Conference*.

Meyer, G. H. (1978). The numerical solution of Stefan problems with front-tracking and smoothing methods. *Applied Mathematics and Computation*, 4(4):283–306.

Millner, A. J., Gershman, S. J., Nock, M. K., and den Ouden, H. E. M. (2018). Pavlovian control of escape and avoidance. *Journal of Cognitive Neuroscience*, 30(10):1379–1390.

Mulder, M. J., Wagenmakers, E.-J., Ratcliff, R., Boekel, W., and Forstmann, B. U. (2012). Bias in the brain: A diffusion model analysis of prior probability and potential payoff. *Journal of Neuroscience*, 32(7):2335–2343.

Murphy, P. R., Boonstra, E., and Nieuwenhuis, S. (2016). Global gain modulation generates time-dependent urgency during perceptual choice in humans. *Nature Communications*, 7:13526.
Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

Nguyen, K. P., Josić, K., and Kilpatrick, Z. P. (2019). Optimizing sequential decisions in the drift-diffusion model. *Journal of Mathematical Psychology*, 88:32–47.

Nocedal, J. and Wright, S. J. (2006). Numerical optimization.

Ossmy, O., Moran, R., Pfeffer, T., Tsetsos, K., Usher, M., and Donner, T. H. (2013). The timescale of perceptual evidence integration can be adapted to the environment. *Current Biology*, 23(11):981–986.

Pinto, L., Koay, S. A., Engelhard, B., Yoon, A. M., Deverett, B., Thiberge, S. Y., Witten, I. B., Tank, D. W., and Brody, C. D. (2018). An accumulation-of-evidence task using visual pulses for mice navigating in virtual reality. *Frontiers in Behavioral Neuroscience*, 12.

Purcell, B. A., Heitz, R. P., Cohen, J. Y., Schall, J. D., Logan, G. D., and Palmeri, T. J. (2010). Neurally constrained modeling of perceptual decision making. *Psychological Review*, 117(4):1113–1143.

Ratcliff, R. (1978). A theory of memory retrieval. *Psychological Review*, 85(2):59–108.

Ratcliff, R. (1985). Theoretical interpretations of the speed and accuracy of positive and negative responses. *Psychological Review*, 92:212–225.

Ratcliff, R. (2008). The EZ diffusion method: Too EZ? *Psychonomic Bulletin & Review*, 15(6):1218–1228.

Ratcliff, R. and Rouder, J. N. (1998). Modeling response times for two-choice decisions. *Psychological Science*, 9(5):347–356.

Ratcliff, R., Smith, P. L., Brown, S. D., and McKoon, G. (2016). Diffusion decision model: Current issues and history. *Trends in Cognitive Sciences*, 20(4):260–281.

Roitman, J. D. and Shadlen, M. N. (2002). Response of neurons in the lateral intraparietal area during a combined visual discrimination reaction time task. *The Journal of Neuroscience*, 22(21):9475–9489.

Roxin, A. and Ledberg, A. (2008). Neurobiological models of two-choice decision making can be reduced to a one-dimensional nonlinear diffusion equation. *PLoS Computational Biology*, 4(3):e1000046.

Scott (2015). *Multivariate Density Estimation*. John Wiley & Sons.

Shinn, M. (2019). Refinement type contracts for verification of scientific investigative software. In *Proceedings of the 11th Working Conference on Verified Software: Theories, Tools, and Experiments*.

Shinn, M., Ehrlich, D., Lee, D., Murray, J. D., and Seo, H. (2019). Confluence of timing and reward biases in perceptual decision-making dynamics.

Simen, P., Balci, F., deSouza, L., Cohen, J. D., and Holmes, P. (2011). A model of interval timing by neural integration. *Journal of Neuroscience*, 31(25):9238–9253.

Srivastava, V., Feng, S. F., Cohen, J. D., Leonard, N. E., and Shenhav, A. (2017). A martingale analysis of first passage times of time-dependent wiener diffusion models. *Journal of Mathematical Psychology*, 77:94–110.

Storn, R. and Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

Tavares, G., Perona, P., and Rangel, A. (2017). The attentional drift diffusion model of simple perceptual decision-making. *Frontiers in Neuroscience*, 11.

Thura, D., Beauregard-Racine, J., Fradet, C.-W., and Cisek, P. (2012). Decision making by urgency gating: theory and experimental support. *Journal of Neurophysiology*, 108(11):2912–2930.

Thura, D. and Cisek, P. (2014). Deliberation and commitment in the premotor and primary motor cortex during dynamic decision making. *Neuron*, 81(6):1401–1416.

Tsetsos, K., Chater, N., and Usher, M. (2012). Salience driven value integration explains decision biases and preference reversal. *Proceedings of the National Academy of Sciences*, 109(24):9659–9664.
Usher, M. and McClelland, J. L. (2001). The time course of perceptual choice: The leaky, competing accumulator model. *Psychological Review*, 108(3):550–592.

van Ravenzwaaij, D. and Oberauer, K. (2009). How to use the diffusion model: Parameter recovery of three methods: EZ, fast-dm, and DMAT. *Journal of Mathematical Psychology*, 53(6):463–473.

Vandekerckhove, J. and Tuerlinckx, F. (2008). Diffusion model analysis with MATLAB: A DMAT primer. *Behavior Research Methods*, 40(1):61–72.

Verdonck, S., Meers, K., and Tuerlinckx, F. (2015). Efficient simulation of diffusion-based choice RT models on CPU and GPU. *Behavior Research Methods*, 48(1):13–27.

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

Wagenmakers, E.-J., Maas, H. L. J. V. D., and Grasman, R. P. P. P. (2007). An EZ-diffusion model for response time and accuracy. *Psychonomic Bulletin & Review*, 14(1):3–22.

Wagenmakers, E.-J., van der Maas, H. L. J., Dolan, C. V., and Grasman, R. P. P. P. (2008). EZ does it! extensions of the EZ-diffusion model. *Psychonomic Bulletin & Review*, 15(6):1229–1235.

Wald, A. (1945). Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186.

Wales, D. J. and Doye, J. P. K. (1997). Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116.

Wiecki, T. V., Sofer, I., and Frank, M. J. (2013). HDDM: Hierarchical bayesian estimation of the drift-diffusion model in python. *Frontiers in Neuroinformatics*, 7.

Wong, K.-F. (2006). A recurrent network mechanism of time integration in perceptual decisions. *Journal of Neuroscience*, 26(4):1314–1328.

Zhou, X., Wong-Lin, K., and Philip, H. (2009). Time-varying perturbations can distinguish among integrate-to-threshold models for perceptual decision making in reaction time tasks. *Neural Computation*, 21(8):2336–2362.

Zylberberg, A., Fetsch, C. R., and Shadlen, M. N. (2016). The influence of evidence volatility on choice, reaction time and confidence in a perceptual decision. *eLife*, 5.
Figure 2-Figure supplement 1. Random dot motion task. Two rhesus monkeys performed a perceptual decision-making task. Visual stimuli consisted of randomly placed dots on a screen which each moved independently to the left or right. The monkey was required to determine the majority direction of the dots. Task difficulty was varied by changing the fraction of dots which moved coherently in a single direction. Used with permission from Roitman and Shadlen (2002).
Figure 2–Figure supplement 2. Fit of DDM and GDDM in monkey B. Format similar to Figure 2.
Figure 4–Figure supplement 1. Execution time vs MSE tradeoff across parameters. As a proxy for the execution time vs error tradeoff, mean squared error (MSE) multiplied by execution time in seconds is shown for several values of $\Delta x$ and $\Delta t$ for each method. The lower left corner is the most accurate, and the upper right corner is the fastest. Darker colors indicate more favorable values.

Figure 4–Figure supplement 2. Example simulations with different numerical parameters. (a) Example traces are shown for all four methods. These simulations are compared to the analytic solution, shown as a dotted black line. (b) The execution time is plotted against the accuracy. Lines from Figure 4c are plotted in light colors for comparison. Note that not all points fall on these lines because the points don't necessarily satisfy the constraints on $\Delta x$ and $\Delta t$ from Figure 4c and Figure Supplement 1.
**Figure 5—Figure supplement 1. Organization of PyDDM’s source code.** The PyDDM source code is organized in a way which mirrors the GDDM framework. Each model is built by defining five components, analogous to those described in “GDDM description”. Each of these components may have any number of fittable parameters, and these parameters may be shared across model components. They may also depend on task parameters, which vary trial to trial. There are three central functions which may be performed on a model: it may be fitted to data, it can be simulated for a specific trial type (i.e. with a given set of task parameters), or it may visualized using the GUI.

**Figure 5—Figure supplement 2. Model GUI.** The PyDDM GUI allows the user to interactively change the parameters of their model (right pane) and see how these changes are reflected in the RT distribution. This makes it possible to gain an intuition for how various parameters and interactions among parameters impact the shape of the RT distribution. The GUI may be used with or without experimental data. When experimental data is provided the model’s simulated RT distribution is overlaid on the empirical RT histogram. The data may be shown all at once, or subset by experimental condition (left pane). Shown here are the data from Roitman and Shadlen (2002) and the 5-parameter GDDM from Figure 2.