FlexDDM: A flexible decision-diffusion Python package for the behavioral sciences

Kyle J. LaFollette (kjl113@case.edu)
Department of Psychological Sciences
Case Western Reserve University, Cleveland, OH, USA

Alessandra Puccio (agp63@case.edu)
Department of Computer and Data Sciences
Case Western Reserve University, Cleveland, OH, USA

Joy Fan (jyf6@case.edu)
Department of Computer and Data Sciences
Case Western Reserve University, Cleveland, OH, USA

Heath A. Demaree (had4@case.edu)
Department of Psychological Sciences
Case Western Reserve University, Cleveland, OH, USA

Abstract

Decision diffusion models are commonly used to explain the processes underlying decision-making. Many software options exist for cognitive scientists to fit diffusion models to data; however, they tend to lack customizability beyond existing model formulations that are already built into them, stymying new theoretical contributions. We introduce *FlexDDM*, a new Python package that requires minimal coding to develop new diffusion models. The package is equipped with four standard models of cognitive conflict tasks and a suite of fitting techniques. Our development of *FlexDDM* aims to broaden the accessibility and applicability of computational methods in cognitive science, thereby accelerating theoretical innovation and contributing to advancements in the field of behavioral sciences.

Keywords: computational modeling; decision diffusion; decision-making; Python; software

Introduction

In the dynamic realm of behavioral science, decision diffusion models (DDMs) have emerged as a pivotal tool for explaining the cognitive processes underlying decision-making (Ratcliff, 1978; Ratcliff et al., 2016). These models are based on the concept that decision-making is a process of accumulating evidence over time until a threshold is reached, which then triggers a decision. The DDM represents this process as a particle undergoing a random walk in a two-dimensional space, where the movement of the particle is influenced by the incoming evidence for or against a particular choice. This approach has been particularly useful in explaining the speed-accuracy trade-off in decision-making tasks and in modeling reaction time distributions in simple choice tasks, such as those presenting two forced-choice alternatives. By varying the parameters of the model, such as the rate of evidence accumulation and the decision threshold, researchers can predict how changes in task conditions or neural processing might affect decision-making behavior. Yet, despite their theoretical appeal and empirical successes, DDMs present notable challenges for cognitive scientists striving to forge new theoretical frontiers.

The ability to formalize and validate new theoretical models is a force for innovation in cognitive science. It ensures that models of behavior evolve in tandem with insights from the latest empirical work and theoretical directions. To implement DDMs in their research, most cognitive scientists rely on off-the-shelf fitting software (e.g. Ahn et al., 2017; Wagenmakers et al., 2007; Wiecki et al., 2013). Unfortunately, most standard software packages tend to lack the flexibility necessary to construct original models. Cognitive scientists are left with the choice to either limit themselves to testing older, albeit well-established models, or code their own modeling software from scratch. This of course, imposes a substantial barrier of entry to anyone interested in making new theoretical contributions with DDMs. Software should be both flexible and accessible, accommodating unique formulations while requiring minimal programming experience. To address these limitations in currently available software, we developed *FlexDDM*, a simulation-based model fitting package written in Python. *FlexDDM* comes with pre-written scripts for multiple leading DDM variants, and user-friendly tools for either modifying those scripts or creating new scripts from scratch with ease.

In this paper, we introduce obstacles for cognitive scientists interested in extending the DDM and *FlexDDM's* solutions. We discuss *FlexDDM's* features including optimizers, loss functions, and parallelization, and the user experience for writing diffusion models from scratch. Finally, to demonstrate a use-case for *FlexDDM* and a common practice in computational cognitive modeling, we re-analyze open experimental data from the Erikson Flanker task, identifying a best-fitting model of cognitive control and conflict processing.

Obstacles for extending the DDM

The DDM in its simplest form is typically limited to four parameters: drift rate, boundary separation, initial starting position, and non-decision time. These four parameters represent a person's average rate of evidence accumulation, response caution, choice biases, and decision-unrelated delays (such as pre-motor planning), respectively. This classical DDM is a powerful tool for cognitive scientists who seek an explanatory account for decision-making, however it does have empirical faults. Among these faults is an inability to account for fast and slow errors relative to correct response times. A number of extensions to the model have been made to account for such faults, such as between-trial variability parameters for Gaussian-distributed drift rate, and uniformly-distributed starting position and non-decision time. Together,

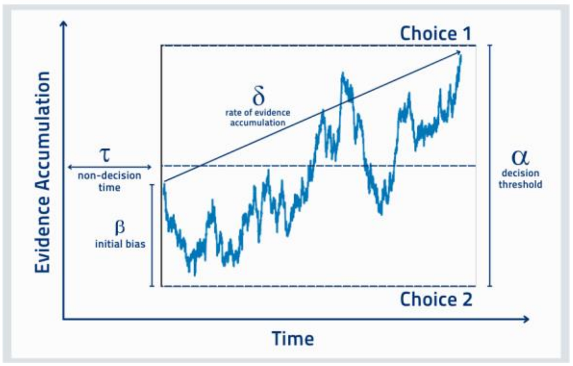

Figure 1: Schematic of the decision diffusion model. In this model, a particle diffuses through 2D-space, reflect evidence for one of two alternatives. Evidence accumulates noisily until a threshold is met, culminating in a decision and response time.

they form what is commonly referred to as the "full DDM". Despite the improvement to fit yielded by the full DDM, some authors claim that between-trial variability parameters lack theoretical motivation (Tillman et al., 2020), and the behaviors they capture can be better modeled with different formulations. Furthermore, the fit may be "too" good, disallowing falsifiability in favor of infinitely flexible models (Jones & Dzhafarov, 2014). Whether or not these extensions are worth reconsidering, I t is clear that model development benefits from theoretical constraints.

Despite the necessity to recognize theoretically motivated constraints, modelers need to be free to explore new models beyond those well-established in support of more contemporary theory. There are essentially an unlimited number of ways to conceptualize a decision diffusion process, from incorporating an urgency signal into the drift rate that builds as time passes (Ditterich, 2006) to collapsing boundaries that throw caution aside as a deadline approaches (Hawkins et al., 2015). Modifications to the standard decision diffusion model are at the forefront of new theoretical developments. Notable among these are the Dual-Stage Two-Phase (DSTP; Hübner et al., 2010) model, the Shrinking Spotlight (SSP; White et al., 2011) model, and the Diffusion Model for Conflict (DMC; Ulrich et al., 2015). These models are unified by a key characteristic: the dynamic nature of evidence accumulation over time, albeit through different mechanisms. The DSTP model posits two separate diffusion processes: one for attention selection and another for decision-making. Initially, the decision-making process gathers information from all stimuli; however, if the attention selection process terminates first, the focus shifts to solely gathering evidence from the chosen target. Conversely, the SSP model envisages a continuously shrinking "spotlight" over stimuli, zeroing in on the target until a decision is reached upon meeting an evidence threshold. The DMC model instead suggests two competing accumulation processes: a controlled process that concentrates on the target and an automatic process that inadvertently includes evidence from distractors, fluctuating over time. These three models offer diverse theoretical frameworks for understanding how information is processed in conflict tasks, such as the Erikson Flanker task (Eriksen & Eriksen, 1974).

Cognitive scientists interested in fitting these models to their own data or developing models to explore new theoretical directions will be disappointed to find that most conventional software for decision diffusion modeling tends to be rigid, constraining researchers to pre-existing formulations of diffusion models. A significant reason for this impediment arises from the reliance of most model fitting routines on finding analytical solutions with known likelihood functions. These functions measure the probability of a set of parameters given specific observed data, and are a critical ingredient for popular optimization or search methods like Maximum Likelihood Estimation or Markov Chain Monte Carlo. Unfortunately, when new formulations or seemingly benign modifications are introduced to DDMs, the resulting likelihood functions can lack closed-form analytical solutions or become so complex that they are computationally intractable (Ratcliff, 1980). These intractabilities limit theoretical advancement, as researchers are unable to explore new model dynamics due to computational constraints.

Due to the complexity of their likelihood functions, these models are omitted from the standard diffusion modeling software relying on likelihood-based fitting routines. Researchers are instead limited to more niche software specifically designed for fitting these models with simulation methods (Grange, 2016; Mackenzie & Dudschig, 2021). The process involves generating simulated data from a model and then comparing those simulated data to one's actual empirical data. A key tool in this comparison is a loss function, such as the likelihood ratio Chi-square statistic or Kullback-Leibler divergence, which serves as a measure of the discrepancy between the simulated and observed data distributions (Ratcliff & Smith, 2004). This loss function quantifies how well the model explains the observed data; a lower value indicating better fit. Optimization routines then iteratively simulate and compare, aiming to minimize loss. With each iteration, the model parameters are adjusted to reduce the discrepancy between the simulated and empirical data, until the best fitting parameter values are discovered.

To our knowledge only two packages exist for fitting customizable DDMs with likelihood-free methods, CHaRTr (Chandrasekaran & Hawkins, 2019) and PyDDM (Shinn et al., 2020). These packages offer previously unprecedented flexibility in model formulation; however, they also have a number of practical limitations. Like *FlexDDM*, CHaRTr relies on trial-wise simulation to approximate the probability distribution of response times. Simulation is slow, and so to improve efficiency CHaRTr defines models in the compiled C language which directly translates to machine code. Although having models written in C substantially speeds up simulation, it presents a notable barrier to users less familiar with the language. Conversely, PyDDM uses an alternative approach to reduce simulation time: Solving the Fokker-Planck equation (Voss & Voss, 2008). This method sidesteps

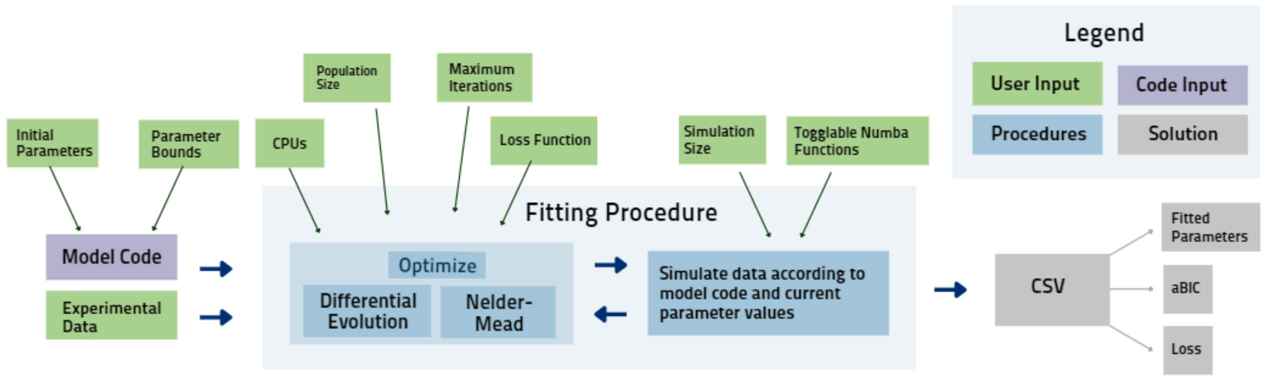

Figure 2: Workflow for the *FlexDDM* fitting routine. The user specifies a model from the model library, either provided by *FlexDDM* or a custom script. The user also provides a number of inputs as arguments to the fitting function, including settings for model parameters (initial values and boundaries), optimization (number of CPUs to parallelize across, loss function, and optimizer-specific hyperparameters), and simulation (number of simulated trials and Numba-specific togglable features). After fitting, a csv is written with fitted parameter values, loss value, and approximate BIC.

trial-wise simulation in favor of an algorithmic, numerical solution, which proves much faster than simulation. Unfortunately, these numerical solutions do not allow for certain model formulations, such as those including a between-trial variability parameter for drift rate, as is in the full DDM. Users can manipulate PyDDM's set of user-defined functions for DDM ingredients like drift rate, boundary separation, etc., but some formulations like DSTP are not possible to specify.

FlexDDM improves on existing software by operating entirely on straightforward Python. Users only need requisite knowledge of elementary Python features, like looping, and gold-standard libraries like NumPy (Harris et al., 2020). For efficiency, models are wrapped with the Numba JIT compiler (Lam et al., 2015), which translates Python models to fast machine code. Virtually any diffusion model can be formalized and simulated with *FlexDDM*. In the following section, we will discuss the features unique to *FlexDDM* that afford it greater flexibility and accessibility in comparison to alternative software.

Features of FlexDDM

At its core, *FlexDDM* allows users to understand how accurately a drift diffusion model represents their participants' latent decision-making processes through completing an iterative process that compares user-provided reaction-time data to data simulated from the chosen diffusion model. *FlexDDM* relies on optimization routines along with multiprocessing to produce accurate and efficient results. The code for this package is deployed on GitHub: https://github.com/joyfan00/FlexDDM. An illustration of the package workflow can be found in Figure 2.

Writing Models with Python

FlexDDM currently provides code for four diffusion models: a standard DDM, SSP, DMC, and DSTP models. Using the scripts for these models as templates, users are encouraged to write code for their own models. In place of abstract functions which may not be immediately transparent to the user, *FlexDDM* requires that users write out the diffusion process explicitly. We provide an example of this in Figure 3. Using two simple loops and the basic NumPy functions, the code in Figure 3 successfully simulates response time data according to the standard DDM. The outer *for* loop cycles through trials, during each of which the accumulation process propagates. Time *t* starts at non-decision time *tau*, and starting evidence at a percentage *beta* of boundary separation *alpha*. A seed is set with the numpy.random.seed() function to initialize a pseudorandom number generator, ensuring that random behaviors over the course of the trial (e.g., diffusion noise) are reproducible. The inner *while* loop increments time by *dt* and accumulates evidence proportional to the drift rate randomly sampled noise. When a threshold is met, the decision is made and both the decision and the time it took to

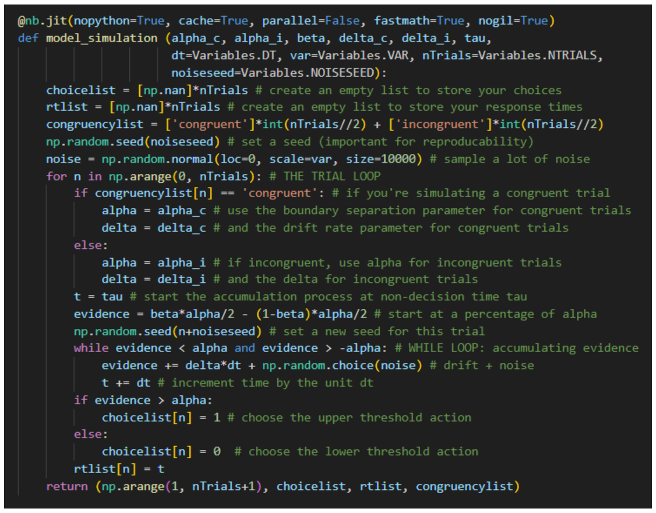
```
@nb.jit(nopython=True, cache=True, parallel=False, fastmath=True, nogil=True)
def model_simulation (alpha_c, alpha_i, beta, delta_c, delta_i, tau,
                      dt=Variables.DT, var=Variables.VAR, nTrials=Variables.NTRIALS,
                      noiseseed=Variables.NOISESEED):
    choicelist = [np.nan]*nTrials # create an empty list to store your choices
    rtlist = [np.nan]*nTrials # create an empty list to store your response times
    congruencylist = ['congruent']*int(nTrials//2) + ['incongruent']*int(nTrials//2)
    np.random.seed(noiseseed) # set a seed (important for reproducability)
    noise = np.random.normal(loc=0, scale=var, size=10000) # sample a lot of noise
    for n in np.arange(0, nTrials): # THE TRIAL LOOP
        if congruencylist[n] == 'congruent': # if you're simulating a congruent trial
            alpha = alpha_c # use the boundary separation parameter for congruent trials
            delta = delta_c # and the drift rate parameter for congruent trials
        else:
            alpha = alpha_i # if incongruent, use alpha for incongruent trials
            delta = delta_i # and the delta for incongruent trials
        t = tau # start the accumulation process at non-decision time tau
        evidence = beta*alpha/2 - (1-beta)*alpha/2 # start at a percentage of alpha
        np.random.seed(n+noiseseed) # set a new seed for this trial
        while evidence < alpha and evidence > -alpha: # WHILE LOOP: accumulating evidence
            evidence += delta*dt + np.random.choice(noise) # drift + noise
            t += dt # increment time by the unit dt
        if evidence > alpha:
            choicelist[n] = 1 # choose the upper threshold action
        else:
            choicelist[n] = 0  # choose the lower threshold action
        rtlist[n] = t
    return (np.arange(1, nTrials+1), choicelist, rtlist, congruencylist)
```

Figure 3: Example Python code for simulating a DDM.

reach the threshold is saved. This process repeats for each trial. These few lines of code can be easily replicated and modified to represent a range of models.

Compiling Models with Numba

FlexDDM uses Numba, a powerful just-in-time (JIT) compiler that translates Python code into fast machine code, significantly accelerating computational tasks, especially those involving numerical computations and array operations. By using decorators, Numba allows developers to mark functions for optimization; when these functions are called, Numba compiles them to machine code "just in time" for execution. A key feature of Numba is its "nopython" mode, which ensures that the compiled code does not rely on the Python C API. This mode guarantees maximum performance gains because it bypasses the Python interpreter entirely. Additionally, Numba supports caching, meaning that once a function is compiled, the machine code version is stored, so subsequent calls to the function do not require recompilation. This feature is particularly useful for applications that execute the same functions multiple times during their lifecycle. Numba also offers an option for "fastmath", which relaxes certain mathematical precision and ordering rules, enabling further optimizations that can lead to speedups in numerical computations. Together, these features make Numba a powerful tool for optimizing Python code, making it an attractive option for developers looking to boost the performance of computationally intensive tasks. *FlexDDM* includes toggles for each of these Numba features to improve simulation efficiency.

Optimization Routine

FlexDDM seeks to optimize a solution to an objective function G^2, otherwise known as the likelihood ratio chi-squared. G^2 provides a measure of the similarity of response time (RT) distributions between an empirical and simulated data set. RTs are grouped by trial type (congruent vs incongruent) and accuracy (correct vs incorrect) and, next, proportions of RTs falling within bins bounded by *empirical* percentiles are compared.

$$G^2 = 2\sum_i Np_i \ln\left(\frac{p_i}{\pi_i}\right) \tag{1}$$

Where p_i is the proportion of empirical RTs in bin i, π_i is the proportion of simulated RTs in bin i, and N is the number of trials. See Figure 4 for a visual representation of binned proportions being compared for G^2 calculation.

To identify the best fitting parameter values for simulating RTs which minimize the G^2 objective function, *FlexDDM* uses an iterative fitting process that starts with the use of a global optimizer known as Differential Evolution (DE; Ahmad et al., 2022). DE starts with randomly generated solutions and then creates new ones by mixing the differences of randomly selected pairs. These new solutions undergo a crossover and selection process, where they are mixed with existing solutions and the better-performing ones are kept. This cycle repeats until an optimal solution is found or a predefined condition is met. DE allows *FlexDDM* to explore the entire parameter space while avoiding local minima.

After DE, *FlexDDM* continues to refine best fitting parameters with the Nelder-Mead simplex algorithm (Wang & Shoup, 2011). This pairing combines DE's global search capability with the simplex's local optimization precision. DE effectively identifies promising areas in the search space, but may lack precision in pinpointing the exact optimum. Simplex algorithms, known for their ability to perform detailed local searches, can then refine the solution to a higher accuracy. This approach leverages DE's strength in exploring diverse solutions and the simplex's efficiency in fine-tuning, making it ideal for complex optimization problems requiring both exploration and exploitation to achieve the best result.

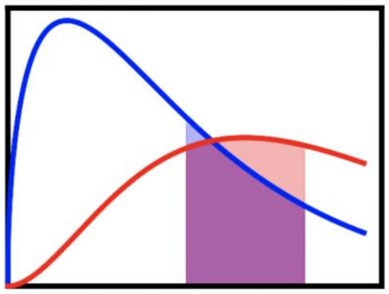

Figure 4: Illustration of empirical and simulated RT proportions being compared for G^2 calculation. A binned region is shaded, and the likelihood ratio is calculated with the proportion of RTs that fall under each curve.

Once the fitting process concludes, *FlexDDM* calculates the approximate Bayesian information criterion (aBIC). Similar to G^2, aBIC measures how well the selected model fits the empirical data. aBIC also considers the complexity of the model, making it a fairer metric for model comparison. Models with a larger number of parameters (M) will be penalized with slightly higher aBIC values, so that models can be evaluated in terms of both accuracy and parsimony.

$$aBIC = -2\sum_i Np_i \ln(\pi_i) + M\ln(N) \tag{2}$$

FlexDDM outputs a CSV file containing the discovered parameter values along with the G^2 and aBIC values. With this information, the user understands the fit and complexity of their model along with the relative influence of each parameter.

Empirical Demonstration

To demonstrate the typical use of the *FlexDDM* package, we present the following case study. Here, we model choice and response time data from an open public dataset containing the Erikson Flanker task (Hedge et al., 2022). In the Flanker task component of this experiment, participants (*N*=50; 38 F; Mean age = 20.06, SD = 2.24) were instructed to respond to the direction of a centrally presented arrow, flanked above and below by other symbols, using specific keyboard keys. A total 1,008 trials were completed, divided equally among congruent, neutral, and incongruent trial types (336 trials/type). On congruent trials, the flanking symbols were arrows pointed in same direction as the centermost arrow, whereas on incongruent trials the flanking symbols were arrows facing the opposite direction. Neutral trials used straight lines as flanking stimuli, and were excluded from our analyses. Trials remained on screen until a response was made and were interleaved with 750ms inter-trial-intervals.

In their original report, the authors of this dataset were interested in fitting the Diffusion Model for Conflict Tasks (DMC) to dissociate conflict from nonconflict-related processes (e.g., the distinction between one's processing efficiency and susceptibility to prepotent response activity). Conversely, we were interested in fitting the DMC model to these data, in addition to fitting the DSTP, SSP, and standard DDM. Each of these models purport qualitatively different predictions about response time distributions and processes for cognitive control. DSTP, SSP, and DMC assume that targets and distractors are processed sequentially, continuously, and simultaneously, respectively, whereas the standard DDM fails to provide any explanatory account for conflict (Servant et al., 2014).

Considering their divided support for opposing theories of cognitive control, it is important to compare these models' goodness-of-fit to empirical data and identify a leading hypothesis. Previous research, however, has yielded mixed results, with support found for DSTP (Servant et al., 2015), SSP (White et al., 2011), and DMC (Servant & Evans, 2020) over their competitors. To contribute to the growing body of knowledge on these models and their appropriateness for modeling conflict tasks, we used *FlexDDM* to fit the four models separately to each of the *N*=50 participants' Flanker data.

Model Fitting

The DSTP model had nine free parameters: two boundary separations αSS and αRS for the first and second diffusion phases in which a stimulus (target or flanker) and response is selected, respectively; two starting points βSS and βRS for each phase; four drift rates for the stimulus selection phase (δSS), the evidence for a response provided by the target and flankers if no stimulus is selected (δ_{tar} and δ_{fl}), and evidence for the stimulus if selected before a response is made (δRS); and one non-decision time τ.

The SSP model had six free parameters: one boundary separation α; one perceptual strength for all stimuli p; two parameters to describe the initial width of the attentional spotlight (sd_0) and its shrinking rate (sd_r); and one non-decision time τ.

The DMC model had seven free parameters: one boundary separation α; one starting point β; one drift rate for controlled processing μ_c; three parameters describing the drift rate for automatic processing, including its shape a, peak amplitude ζ, and characteristic time T; and one non-decision time τ.

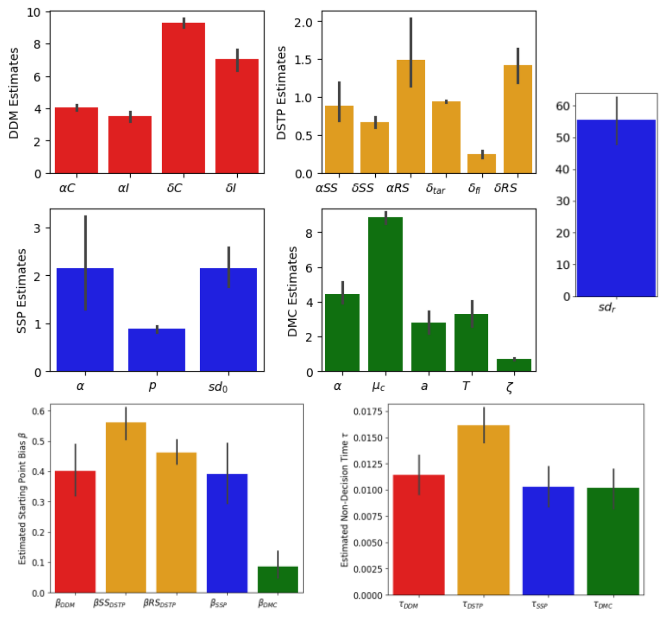

Figure 5. Best fitting parameter estimates from DDM, DSTP, SSP, and DMC models. Error bars are SE.

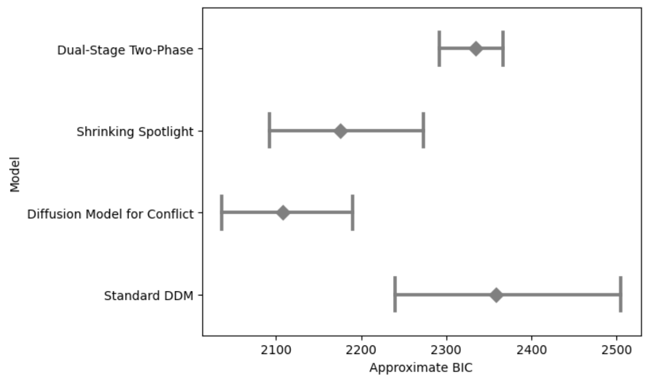

Figure 6: Average BIC across participants from DDM, DSTP, SSP, and DMC models. Error bars are 95% CI.

Finally, the DDM model had six free parameters: two boundary separations αC and αI for congruent and incongruent trials, respectively; one starting point β; two drift rates δC and δI; and one non-decision time τ.

For optimization, we opted for DE to cycle through a maximum 1,000 generations with a population multiplier of 100. 10,000 trials were simulated per calculation of G^2. Model selection was conducted with BIC, such that the best fitting model was determined by having the relatively smallest BIC value.

Results

Best fitting parameter estimates are summarized in Figure 5. BICs averaged across subjects are illustrated in Figure 6 for model comparison. The Diffusion Model for Conflict returned the smallest BIC, and therefore best accounted for the empirical data. The Shrinking Spotlight Model provided an only slightly inferior average fit, with the Dual-Stage Two Phase Model close behind. The worst fitting and most divisive model was the standard DDM.

General Discussion

In the presented paper, we address a significant challenge in cognitive science: The lack of flexible and accessible software for fitting decision diffusion models (DDMs) to data, thereby hampering theoretical innovation. By introducing *FlexDDM*, a Python-based package, we provide a solution that not only simplifies the development of new diffusion models but also includes standard models for cognitive conflict tasks alongside a suite of fitting techniques. This development is particularly notable for its aim to democratize computational methods in cognitive science, making them accessible to researchers without extensive programming experience in languages such as C. By facilitating the exploration of novel model formulations beyond those pre-built into existing software, *FlexDDM* stands to accelerate theoretical advancements in the behavioral sciences.

FlexDDM is not without its limitations. Users should be aware that, at present, *FlexDDM* is limited to non-hierarchical models, restricting the ability to capture variations across individuals or groups within a single model framework. This limitation means that the software provides only point estimates for parameters without accounting for any posterior distribution or uncertainty associated with these estimates. Such an approach contrasts with Bayesian hierarchical modeling, which can offer deeper insights into the data by considering the distribution of parameters and allowing for the estimation of individual-level effects. Furthermore, the simulation-based nature of *FlexDDM*, while offering flexibility in model construction, inherently suffers from the standard weaknesses associated with simulation methods. These include slower computational speeds compared to analytical solutions, especially as model complexity increases. The process of simulating thousands of trials to approximate the probability distribution of response times can be computationally demanding, making *FlexDDM* less efficient for large datasets or complex models without access to resources for parallelization.

Looking to the future, continued development of *FlexDDM* presents several promising avenues that could significantly enhance its utility and applicability in cognitive science research. Key among these is the integration of advanced model validation tools, which could offer users robust methodologies for assessing the fit and predictive accuracy of their custom models. In future versions of the software, we will add functions for testing model and parameter recovery. Expanding the library of template models within *FlexDDM* would also enable researchers to quickly adapt and test a wider range of theoretical frameworks without the need for extensive coding, thereby further lowering the barrier to entry. Additionally, the introduction of sophisticated plotting functions could facilitate more intuitive visualization of model behaviors, parameter effects, and fit diagnostics, making the iterative process of model refinement more accessible to users. Furthermore, incorporating options for grouping data by experimental factors beyond simple trial types would allow for a more nuanced analysis of decision-making processes, accommodating the investigation of how various cognitive and environmental factors interact to influence decision outcomes. Such enhancements would not only solidify *FlexDDM's* position as a versatile tool for cognitive modeling but also broaden its impact by enabling more detailed and comprehensive explorations of decision-making dynamics.

References

Ahmad, M. F., Isa, N. A. M., Lim, W. H., & Ang, K. M. (2022). Differential evolution: A recent review based on state-of-the-art works. *Alexandria Engineering Journal*, *61*(5), 3831–3872. https://doi.org/10.1016/j.aej.2021.09.013

Ahn, W.-Y., Haines, N., & Zhang, L. (2017). Revealing Neurocomputational Mechanisms of Reinforcement Learning and Decision-Making With the hBayesDM Package. *Computational Psychiatry*, *1*(0), 24. https://doi.org/10.1162/CPSY_a_00002

Chandrasekaran, C., & Hawkins, G. E. (2019). ChaRTr: An R toolbox for modeling choices and response times in decision-making tasks. *Journal of Neuroscience Methods*, *328*, 108432. https://doi.org/10.1016/j.jneumeth.2019.108432

Ditterich, J. (2006). Evidence for time-variant decision making. *European Journal of Neuroscience*, *24*(12), 3628–3641. https://doi.org/10.1111/j.1460-9568.2006.05221.x

Eriksen, B. A., & Eriksen, C. W. (1974). Effects of noise letters upon the identification of a target letter in a nonsearch task. *Perception & Psychophysics*, *16*(1), 143–149. https://doi.org/10.3758/BF03203267

Grange, J. A. (2016). flankr: An R package implementing computational models of attentional selectivity. *Behavior Research Methods*, *48*(2), 528–541. https://doi.org/10.3758/s13428-015-0615-y

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Van Kerkwijk, M. H., Brett, M., Haldane, A., Del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hawkins, G. E., Wagenmakers, E.-J., Ratcliff, R., & Brown, S. D. (2015). Discriminating evidence accumulation from urgency signals in speeded decision making. *Journal of Neurophysiology*, *114*(1), 40–47. https://doi.org/10.1152/jn.00088.2015

Hedge, C., Powell, G., Bompas, A., & Sumner, P. (2022). Strategy and processing speed eclipse individual differences in control ability in conflict tasks. *Journal of Experimental Psychology: Learning, Memory, and*

Cognition, *48*(10), 1448–1469. https://doi.org/10.1037/xlm0001028

Hübner, R., Steinhauser, M., & Lehle, C. (2010). A dual-stage two-phase model of selective attention. *Psychological Review*, *117*(3), 759–784. https://doi.org/10.1037/a0019471

Jones, M., & Dzhafarov, E. N. (2014). Unfalsifiability and mutual translatability of major modeling schemes for choice reaction time. *Psychological Review*, *121*(1), 1–32. https://doi.org/10.1037/a0034190

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-based Python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 1–6. https://doi.org/10.1145/2833157.2833162

Mackenzie, I. G., & Dudschig, C. (2021). DMCfun: An R package for fitting Diffusion Model of Conflict (DMC) to reaction time and error rate data. *Methods in Psychology*, *5*, 100074. https://doi.org/10.1016/j.metip.2021.100074

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

Ratcliff, R. (1980). A note on modeling accumulation of information when the rate of accumulation changes over time. *Journal of Mathematical Psychology*, *21*(2), 178–184. https://doi.org/10.1016/0022-2496(80)90006-1

Ratcliff, R., & Smith, P. L. (2004). A Comparison of Sequential Sampling Models for Two-Choice Reaction Time. *Psychological Review*, *111*(2), 333–367. https://doi.org/10.1037/0033-295X.111.2.333

Ratcliff, R., Smith, P. L., Brown, S. D., & McKoon, G. (2016). Diffusion Decision Model: Current Issues and History. *Trends in Cognitive Sciences*, *20*(4), 260–281. https://doi.org/10.1016/j.tics.2016.01.007

Servant, M., & Evans, N. J. (2020). A diffusion model analysis of the effects of aging in the Flanker Task. *Psychology and Aging*, *35*(6), 831–849. https://doi.org/10.1037/pag0000546

Servant, M., Montagnini, A., & Burle, B. (2014). Conflict tasks and the diffusion framework: Insight in model constraints based on psychological laws. *Cognitive Psychology*, *72*, 162–195. https://doi.org/10.1016/j.cogpsych.2014.03.002

Servant, M., White, C., Montagnini, A., & Burle, B. (2015). Using Covert Response Activation to Test Latent Assumptions of Formal Decision-Making Models in Humans. *Journal of Neuroscience*, *35*(28), 10371–10385. https://doi.org/10.1523/JNEUROSCI.0078-15.2015

Shinn, M., Lam, N. H., & Murray, J. D. (2020). A flexible framework for simulating and fitting generalized drift-diffusion models. *eLife*, *9*, e56938. https://doi.org/10.7554/eLife.56938

Tillman, G., Van Zandt, T., & Logan, G. D. (2020). Sequential sampling models without random between-trial variability: The racing diffusion model of speeded decision making. *Psychonomic Bulletin & Review*, *27*(5), 911–936. https://doi.org/10.3758/s13423-020-01719-6

Ulrich, R., Schröter, H., Leuthold, H., & Birngruber, T. (2015). Automatic and controlled stimulus processing in conflict tasks: Superimposed diffusion processes and delta functions. *Cognitive Psychology*, *78*, 148–174. https://doi.org/10.1016/j.cogpsych.2015.02.005

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

Wagenmakers, E.-J., Van Der Maas, H. L. J., & Grasman, R. P. P. P. (2007). An EZ-diffusion model for response time and accuracy. *Psychonomic Bulletin & Review*, *14*(1), 3–22. https://doi.org/10.3758/BF03194023

Wang, P. C., & Shoup, T. E. (2011). Parameter sensitivity study of the Nelder–Mead Simplex Method. *Advances in Engineering Software*, *42*(7), 529–533. https://doi.org/10.1016/j.advengsoft.2011.04.004

White, C. N., Ratcliff, R., & Starns, J. J. (2011). Diffusion models of the flanker task: Discrete versus gradual attentional selection. *Cognitive Psychology*, *63*(4), 210–238. https://doi.org/10.1016/j.cogpsych.2011.08.001

Wiecki, T. V., Sofer, I., & Frank, M. J. (2013). HDDM: Hierarchical Bayesian estimation of the Drift-Diffusion Model in Python. *Frontiers in Neuroinformatics*, *7*. https://doi.org/10.3389/fninf.2013.00014