Beyond Drift Diffusion Models: Fitting a Broad Class of Decision and Reinforcement Learning Models with HDDM

Alexander Fengler[1], Krishn Bera[1], Mads L. Pedersen[1,2], and Michael J. Frank[1]

Abstract

■ Computational modeling has become a central aspect of research in the cognitive neurosciences. As the field matures, it is increasingly important to move beyond standard models to quantitatively assess models with richer dynamics that may better reflect underlying cognitive and neural processes. For example, sequential sampling models (SSMs) are a general class of models of decision-making intended to capture processes jointly giving rise to RT distributions and choice data in *n*-alternative choice paradigms. A number of model variations are of theoretical interest, but empirical data analysis has historically been tied to a small subset for which likelihood functions are analytically tractable. Advances in methods designed for likelihood-free inference have recently made it computationally feasible to consider a much larger spectrum of SSMs. In addition, recent work has motivated the combination of SSMs with reinforcement learning models, which had historically been considered in separate literatures. Here, we provide a significant addition to the widely used HDDM Python toolbox and include a tutorial for how users can easily fit and assess a (user-extensible) wide variety of SSMs and how they can be combined with reinforcement learning models. The extension comes batteries included, including model visualization tools, posterior predictive checks, and ability to link trial-wise neural signals with model parameters via hierarchical Bayesian regression. ■

INTRODUCTION

The drift diffusion model (DDM, also called diffusion decision model or Ratcliff diffusion model; Ratcliff, Smith, Brown, & McKoon, 2016; Ratcliff, 1978) and, more generally, the framework of sequential sampling models (SSMs; Heathcote, Matzke, & Heathcote, 2022; Tillman, Van Zandt, & Logan, 2020; Voss, Lerche, Mertens, & Voss, 2019; Ratcliff et al., 2016; Hawkins, Forstmann, Wagenmakers, Ratcliff, & Brown, 2015) have become a mainstay of the cognitive scientist's model arsenal in the last two decades (Lawlor et al., 2020; Wieschen, Voss, & Radev, 2020; Van Zandt, Colonius, & Proctor, 2000).

SSMs are used to model neurocognitive processes that jointly give rise to choice and RT data in a multitude of domains, spanning from perceptual discrimination to memory retrieval to preference-based choice (Smith, Ratcliff, & Sewell, 2014; Krajbich, Lu, Camerer, & Rangel, 2012; Krajbich & Rangel, 2011; Ratcliff, Thapar, & McKoon, 2006; Ratcliff, 1978) across species (Doi, Fan, Gold, & Ding, 2020; Yartsev, Hanks, Yoon, & Brody, 2018; Gold & Shadlen, 2007). Moreover, researchers are often interested in the underlying neural dynamics that give rise to such choice processes. As such, many studies include additional measurements such as EEG, fMRI, or eye-tracking signals as covariates, which act as latent variables and connect to model parameters (e.g., via a regression model) to drive trial-specific parameter valuations (Doi et al., 2020; Yartsev et al., 2018; Frank et al., 2015; Forstmann et al., 2010; Rangel, Camerer, & Montague, 2008). See Figure 1 for an illustration of the DDM and some canonical experimental paradigms.

The widespread interest and continuous use of SSMs across the research community has spurred the development of several software packages targeting the estimation of such models (Fontanesi, 2022; Heathcote et al., 2019; Vandekerckhove & Tuerlinckx, 2008). For a hierarchical Bayesian approach to parameter estimation, the HDDM toolbox in Python (Wiecki, Sofer, & Frank, 2013; available at github.com/hddm-devs/hddm) is widely used and the backbone of hundreds of studies published in peer-reviewed journals.

HDDM allows users to conveniently specify and estimate DDM parameters for a wide range of experimental designs, including the incorporation of trial-by-trial covariates via regression models targeting specific parameters.

This article is part of a Special Focus entitled Integrating Theory and Data: Using Computational Models to Understand Neuroimaging Data; deriving from a symposium at the 2020 Annual Meeting of the Cognitive Neuroscience Society.
[1]Brown University, [2]University of Oslo

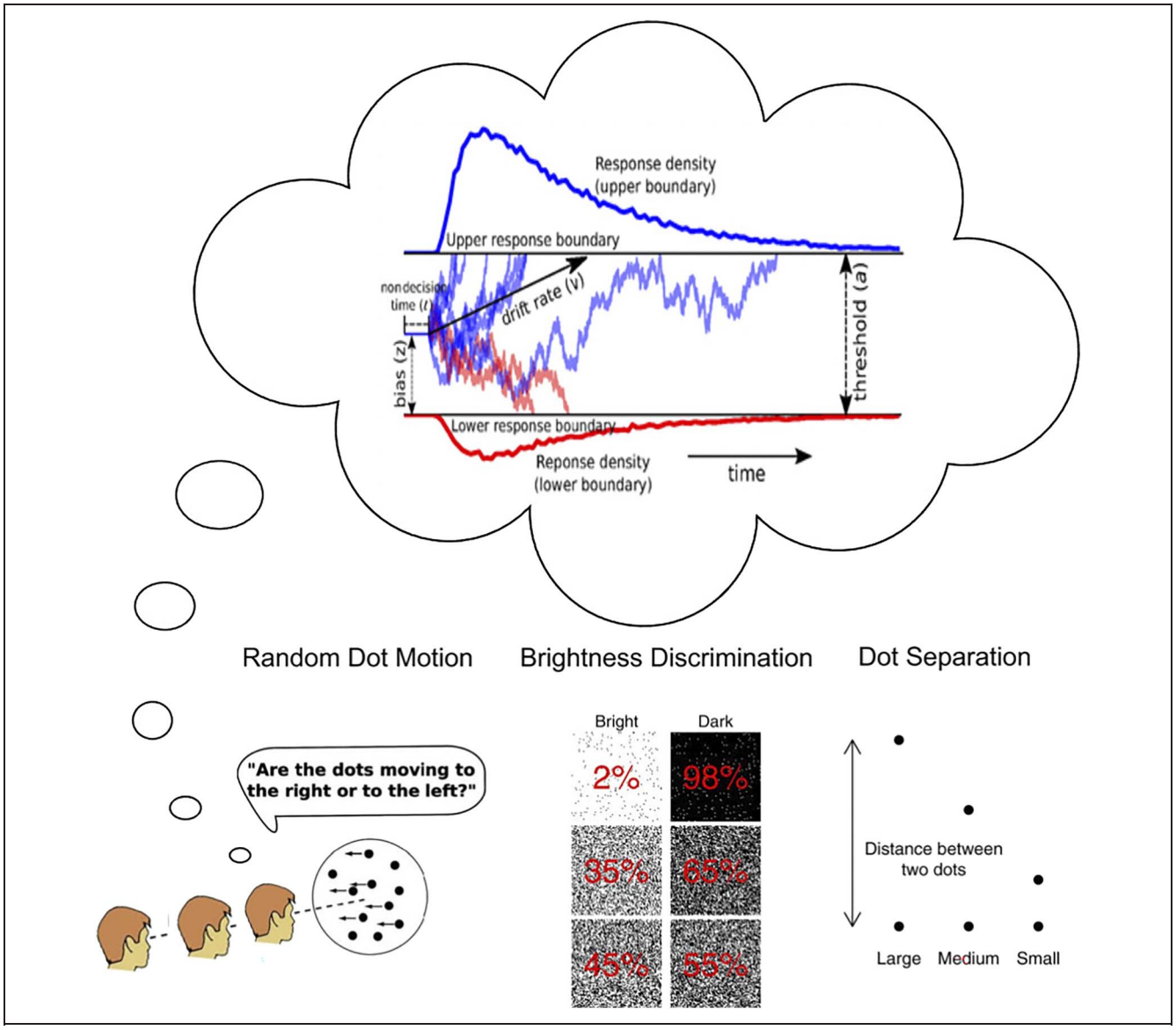


Figure 1. DDM and some example applications.

As an example, one may use this framework to estimate whether trial-by-trial drift rates in a DDM covary with neural activity in a given region (and/or temporal dynamic), pupil dilation, or eye gaze position. Moreover, by using hierarchical Bayesian estimation, HDDM optimizes the inference about such parameters at the individual subject and group levels.

Nevertheless, until now, HDDM and other such toolboxes have been largely limited to fitting the two-alternative choice DDM (albeit allowing for the full DDM with intertrial parameter variability). The widespread interest in SSMs has however also spurred theoretical and empirical investigations into various alternative model variants. Notable examples are, among others, race models with more than two decision options, the leaky competing accumulator model (Usher & McClelland, 2001), SSMs with dynamic decision boundaries (Trueblood, Heathcote, Evans, & Holmes, 2021; Ratcliff & Frank, 2012; Cisek, Puskas, & El-Murr, 2009), and more recently, SSMs based on Levy flights rather than basic Gaussian diffusions (Wieschen et al., 2020). Moreover, as mentioned earlier, SSMs naturally extend to *n*-choice paradigms.

A similar state of affairs is observed for another class of cognitive models that aim to simultaneously model the dynamics of a feedback-based learning across trials as well as the within-trial decision process. One way to achieve this is by replacing the choice rule in a reinforcement learning (RL) process, in itself an important theoretical framework in the study of learning behavior across trials (Collins & Shenhav, 2022; Eckstein, Wilbrecht, & Collins, 2021; McDougle & Collins, 2021;

Dowd, Frank, Collins, Gold, & Barch, 2016) with cognitive process models such as SSMs. This forms a powerful combination of modeling frameworks. Although recent studies moved into this direction (Fontanesi, 2022; Pedersen & Frank, 2020; Fontanesi, Gluth, Spektor, & Rieskamp, 2019; Turner, 2019; Pedersen, Frank, & Biele, 2017), they have again been limited to an application of the basic DDM.

Despite the great interest in these classes of models, tractable inference and, therefore, widespread adoption of such models has been hampered by the lack of easy-to-compute likelihood functions (including essentially all of the examples provided above). In particular, although many interesting models are straightforward to simulate, often researchers want to go the other way: from the observed data to infer the most likely parameters. For all but the simplest models, such likelihood functions are analytically intractable, and hence previous approaches required computationally costly simulations and/or lacked flexibility in applying such methods to different scenarios (Boehm, Cox, Gantner, & Stevenson, 2021; Shinn, Lam, & Murray, 2020; Palestro, Sederberg, Osth, Van Zandt, & Turner, 2019; Turner & Van Zandt, 2018; Turner & Sederberg, 2014). We recently developed a novel approach using artificial neural networks that can, given sufficient training data, approximate likelihoods for a large class of SSM variants, thereby amortizing the cost and enabling rapid, efficient, and flexible inference (Fengler, Govindarajan, Chen, & Frank, 2021). We dubbed such networks LANs, for "likelihood approximation networks."

The core idea behind computation amortization is to run an expensive process only once, so that the fruits of this labor can later be reused and shared with the rest of the community. Profiting from the computational labor incurred in other research groups enables researchers to consider a larger bank of generative models and to sharpen conclusions that may be drawn from their experimental data. The benefit is threefold. Experimenters will be able to adjudicate between a rising number of competing models (theoretical accounts) and capture richer dynamics informed by neural activity, and at the same time, new models proposed by theoreticians can find wider adoption and be tested against data much sooner.

Just as streamlining the analysis of simple SSMs (via, e.g., the HDDM toolbox and others) allowed an increase in adoption, streamlining the production and inference pipeline for amortized likelihoods, we hope, will drive the embrace of SSM variations in the modeling and experimental community by making a much larger class of models ready to be fit to experimental data.

Here, we develop an extension to the widely used HDDM toolbox, which generalizes it to allow for flexible simulation and estimation of a large class of SSMs by reusing amortized likelihood functions.

Specifically, this extension incorporates the following:

- LAN-based (Fengler et al., 2021) likelihoods for a variety of SSMs (batteries included)
- LAN-driven extension of the RL–DDM capabilities, which allows RL learning rules to be applied to all included SSMs
- New plots that focus on visual communications of results across models
- An easy interface for users to import and incorporate their own models and likelihoods into HDDM

This article is formulated as a tutorial to support application of the HDDM LAN extension for data analysis problems involving SSMs.

The rest of the article is organized as follows. We start by providing some basic overview of the capabilities of HDDM. We then give a brief overview of LANs (Fengler et al., 2021). The main part of this article constitutes a tutorial with a detailed introduction on how to use these new features in HDDM. We conclude by embedding the new features into a broader agenda and, finally, mention limitations and preview future developments.

HDDM: THE BASICS

The HDDM Python package (Wiecki et al., 2013) was designed to make hierarchical Bayesian inference for DDMs simple for end users with some programming experience in Python. The toolbox has been widely used for this purpose by the research community, and the feature set evolves to accommodate new use cases. This section serves as a minimal introduction to HDDM to render the present tutorial self-contained. To get a deeper introduction to HDDM itself, please refer to the original article (Wiecki et al., 2013), an extension article specifically concerning RL capabilities (Pedersen & Frank, 2020), and the documentation of the package. Here, we concern ourselves with a very basic workflow that uses the HDDM package for inference.

Data

HDDM expects a data set, provided as a `pandas DataFrame` (McKinney, 2010) with three basic columns: a `"subj_idx"` column that identifies the subject, a `"response"` column that specifies the choice taken in a given trial (usually coded as 1 for "correct" choices and 0 for "incorrect" choices), and an `"rt"` column that stores the trial-wise RTs (in seconds). Other columns can be added, for example, to be used as covariates (task condition or additional measurements such as trial-wise neural data). Here, we take the example of a data set that is provided with the HDDM package. Codeblock 1 shows how to load this data set into a Python interpreter, as shown below:

```
cav_data = hddm.load_csv(hddm.__path__[0] + '/examples/cavanagh_theta_nn.csv')
```

```
      subj_idx stim     rt  response     theta  dbs conf
0            0   LL  1.210       1.0  0.656275    1   HC
1            0   WL  1.630       1.0 -0.327889    1   LC
2            0   WW  1.030       1.0 -0.480285    1   HC
3            0   WL  2.770       1.0  1.927427    1   LC
4            0   WW  1.140       0.0 -0.213236    1   HC
...        ...  ...    ...       ...       ...  ...  ...
3983        13   LL  1.450       0.0 -1.237166    0   HC
3984        13   WL  0.711       1.0 -0.377450    0   LC
3985        13   WL  0.784       1.0 -0.694194    0   LC
3986        13   LL  2.350       0.0 -0.546536    0   HC
3987        13   WW  1.250       1.0  0.752388    0   HC

[3988 rows x 7 columns]
```

Codeblock 1. Loading package-included data.

HDDM Model

Once we have our data in the format expected by HDDM, we can now specify an HDDM model. We focus on a simple example here: a basic hierarchical model that estimates separate drift rates (v) as a function of task condition, denoted by the "stim" column, and moreover estimates the starting point bias z. (Boundary separation, otherwise known as decision threshold a and nondecision time t, is also estimated by default.)

This model assumes that the subject-level z, a, and t parameters are each drawn from the respective group distributions, the parameters of which are also inferred. The v parameters derive from separate group distributions for each value of "stim". Details about the choices of group priors and hyperparameters can be found in the original toolbox paper (Wiecki et al., 2013). Codeblocks 2 and 3 show how to construct and sample from such a model.

Sample and Analyze

With the HDDM model defined, the goal is to fit this model to a given data set. In a Bayesian context, this implies obtaining a posterior distribution over model parameters. For completeness, we note that such posterior distributions are defined via Bayes' rule,

$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

where D is our data, θ is our set of parameters, $p(D|\theta)$ defines the likelihood (analytic in the case of the standard HDDM class) of our data set under the model, and $p(\theta)$ defines our initial prior over the parameters. HDDM uses the probabilistic programming toolbox PyMC (Patil, Huard, & Fonnesbeck, 2010) to generate samples from the posterior distribution via Markov chain Monte Carlo (MCMC; specifically, using coordinate-wise slice samples

```
basic_hddm_model = hddm.HDDM(cav_data, include=['z'], depends_on={'v': ['stim']})
```

Codeblock 2. Initializing HDDM model.

```
basic_hddm_model.sample(1000, burn=500)
```

Codeblock 3. Sampling from a basic HDDM model.

[Neal, 1995]). To generate samples from the posterior, we simply type.

HDDM then provides access to a variety of tools to analyze the posterior and generate quantities of interest, including the following:

1. Chain summaries: To get a quick glance at mean posterior estimates (and their uncertainty) for parameters.
2. Trace plots and the Gelman–Rubin statistic (Brooks & Gelman, 1998): To understand issues with chain convergence (i.e., whether one can trust that the estimates are truly drawn from the posterior).
3. The deviance information criterion (Spiegelhalter, Best, Carlin, & Van der Linde, 2014): As a score to be used for purposes of model comparison (with caution).
4. Posterior predictive plots: To check for the absolute fit of a given model to data (potentially as a function of task condition, etc.).

The HDDM LAN extension maintains this basic HDDM workflow, which we hope facilitates seamless transition for current users of HDDM. After some brief explanations concerning approximate likelihoods, which form the spine of the extension, we will expose the added capabilities in detail.

APPROXIMATE LIKELIHOODS

Approximate Bayesian inference is an active area of research. Indeed, the last decade has seen a multitude of proposals for new algorithms, many of which rely in one way or another on popular deep learning techniques (Tejero-Cantero et al., 2020; Greenberg, Nonnenmacher, & Macke, 2019; Lueckmann, Bassetto, Karaletsos, & Macke, 2019; Papamakarios, Nalisnick, Rezende, Mohamed, & Lakshminarayanan, 2019; Papamakarios, Sterratt, & Murray, 2019; Gutmann, Dutta, Kaski, & Corander, 2018; Papamakarios & Murray, 2016). Relevant to our goals here are algorithms that can estimate trial-by-trial likelihoods for a given model. The main idea is to replace the "likelihood" term in Bayes' rule with an approximation $\hat{p}(\mathbf{D}|\theta)$, which can be evaluated via a forward pass through a simple neural network. Once the networks are trained, these "amortized" likelihoods can then be used as a plug-in (replacing the analytical likelihood function) to run approximate inference. Having access to approximate likelihoods, the user will now be able to apply HDDM to a broad variety of SSMs.

The HDDM extension described here is based on a specific likelihood amortization algorithm, which we dubbed LANs (Fengler et al., 2021). Details regarding this LAN approach, including methods, parameter recovery studies, and thorough tests, can be found in Fengler and colleagues (2021). Note that, in principle, our extension supports the integration of any approximate (or exact) likelihood, in the context of a now simple interface for adding models to HDDM. The scope remains limited only insofar as HDDM remains specialized toward choice/RT modeling. Figure 2 provides some visual intuition regarding the ideas behind LANs.

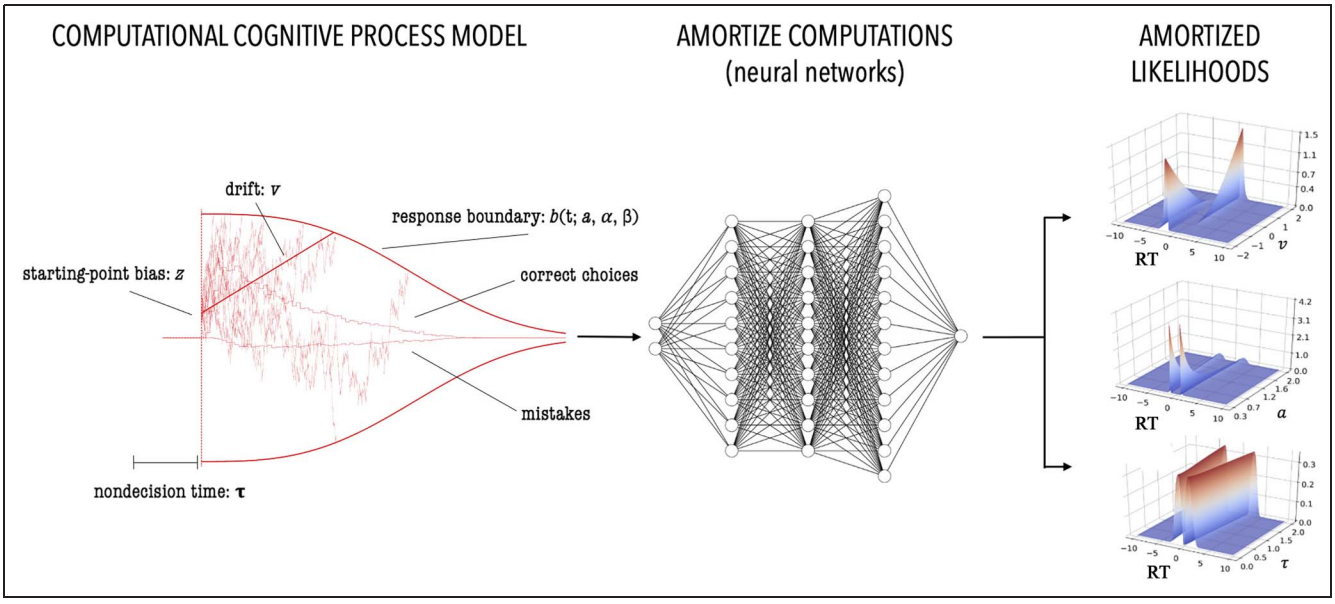


Figure 2. Depiction of the general idea behind LANs. We use a simulator of a likelihood-free cognitive process model to generate training data. These training data are then used to train a neural network, which predicts the log-likelihood for a given feature vector consisting of model parameters, as well as a particular choice and RT. This neural network then acts as a stand-in for a likelihood function facilitating approximate Bayesian inference. Crucially, these networks are then fully and flexibly reusable for inference on data derived from any experimental design.

HDDM EXTENSION: STEP BY STEP

A Central Database for Models: `hddm.model_config`

To accommodate the multitude of new models, HDDM > 0.9 now uses a model specification dictionary to extract data about a given model that is relevant for inference. The `model_config` module contains a central dictionary with which the user can interrogate to inspect models that are currently supplied with HDDM. Codeblock 4 shows how to list the models included by name. For each model, we have a specification dictionary. Codeblock 5 provides an example for the simple DDM.

We focus on the most important aspects of this dictionary (more options are available). Under `"params"`, the parameter names for the given model are listed. `"params_trans"` specifies if the sampler should transform the parameter at the given position (transforming parameters can be helpful for convergence, especially if the parameter space is strongly constrained a priori, e.g., between 0 and 1). The order follows the list supplied under `"params"`. `"param_bounds"` lists the parameter-wise lower and upper bounds of parameters that the sampler can explore. This is important in the context of LAN-based likelihoods, which are only valid in the range of parameters that were observed during training. We trained the LANs included in HDDM on a broad range of parameters (spanning quite a large range of sensible data, you can inspect the training bounds in the `hddm.model_config.model_config` dictionary under the `param_bounds` key). However, it cannot be guaranteed that these were broad enough for any given empirical data set. If the provided LANs are deemed inappropriate for a given data set (e.g., if parameter estimates hit the bounds upon fitting), it is always possible to retrain on an even broader range of parameters. Ruling out convergence issues, however, should be the first order of business in such cases.

HDDM uses the inverse logistic (or logit) transformation for the sampler to operate on an unconstrained parameter space. For a parameter θ and parameter bounds $[a,b]$, this transformation takes θ from a value in $[a,b]$ to a value x in $(-\infty, \infty)$ via

$$x = \ln\left(\frac{\theta - a}{b - \theta}\right)$$

A given SSM usually has a `"decision boundary"`, which is supplied as a function that can be evaluated over time points $(t_0,\ldots,t_n)$ given boundary parameters (supplied implicitly via `"params"`). The values representing each choice are reported as a `list` under `"choices"`. A note of caution: If a user wants to estimate a new model that is not currently in HDDM, a new LAN (or generally likelihood) has to be created for it to be added to the `model_config` dictionary. Simply changing a setting in an existing `model_config` dictionary will not work. Under the `"hddm_include"` key, a list holds a working default for the `include` argument expected from the `HDDM` classes. Finally, `"params_default"` specify the parameter values that are fixed ("not fit") by HDDM, and `"params_std_upper"` specify upper bounds on group-level standard deviations for each parameter (optional, but this can help constrain the parameter ranges proposed by the sampler, making it more efficient).

These `model_config` dictionaries provide a scaffolding for model specification, which is applied

```
hddm.model_config.model_config.keys()
```

Codeblock 4. `model_config`—list available models.

```
hddm.model_config.model_config["ddm"]  =
    {
    "params": ["v", "a", "z", "t"],
    "params_trans": [0, 0, 1, 0],
    "param_bounds": [[-3.0, 0.3, 0.1, 1e-3], [3.0, 2.5, 0.9, 2.0]],
    "boundary": hddm.simulators.bf.constant,
    "hddm_include": ["z"],
    "choices": [-1, 1],
    "params_default": [0.0, 1.0, 0.5, 1e-3],
    "params_std_upper": [1.5, 1.0, None, 1.0],
    }
```

Codeblock 5. DDM specifications in `model_config`.

throughout all of the new functionalities discussed in the next sections.

Batteries Included: `hddm.simulators` and `hddm.network_inspectors`

The new `HDDMnn` (where nn = neural network), `HDDMnnRegressor`, and `HDDMnnStimCoding` classes have access to a (growing) stock of supplied SSMs, including rapid compiled (Behnel et al., 2010) simulators, and rapid likelihood evaluation via LANs (Fengler et al., 2021) and their implementation in PyTorch (Paszke et al., 2019). We will discuss how to fit these models to data in the next section. Here, we describe how one can access the low-level simulators and LANs directly, in case one wants to adopt them for custom purposes. We also show how to assess the degree to which the LAN approximates the true (empirical) likelihood for a given model. Users who only want to apply existing SSMs in HDDM to fit data can skip to the next section. As described in the previous section, the user can check which models are currently available by using the `model_config` dictionary. Figure 3 provides some pictorial examples. For a given model, a docstring includes some information (and possible warnings) about usage. As an example, let us pick the "angle" model, which is an SSM that allows for the decision boundary to decline linearly across time with some estimated angle (note that, although other aspects of the model are standard DDM, even in this case, the likelihood is analytically intractable). Nevertheless, we previously observed that inference using LANs yields good parameter recovery, as per Fengler and colleagues (2021).

Codeblock 6 illustrates such a docstring. Codeblock 7 shows how we can simulate synthetic data from this

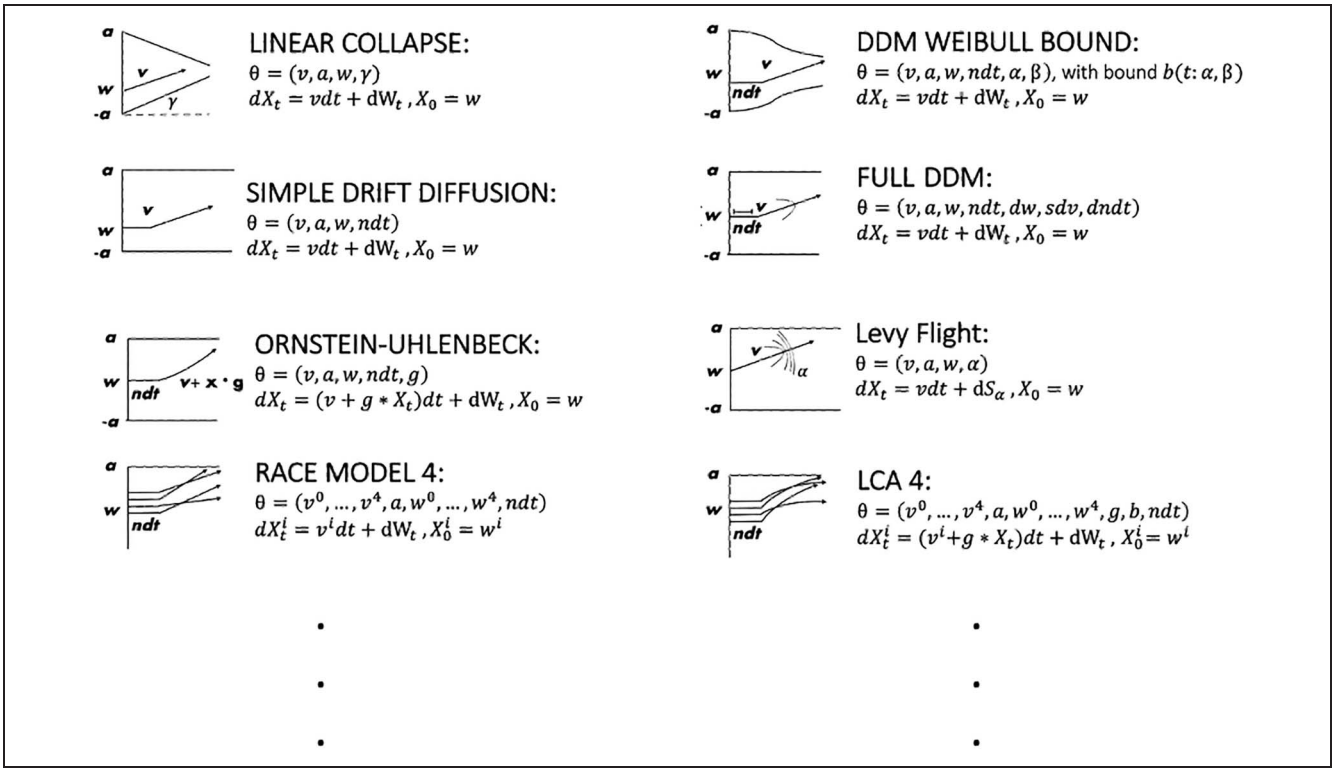


Figure 3. Graphical examples for some of the sequential sampling included in HDDM.

```
print(hddm.model_config.model_config['angle']['doc'])
```

```
Model formulation is described in the documentation under LAN Extension.
Meant for use with the extension.
```

Codeblock 6. `model_config` docstring for the `angle` model.

```
from hddm.simulators import simulator
from hddm.model_config import model_config
out = simulator(model='angle', theta=model_config['angle']['params_default'],
                n_samples=100, delta_t=0.001)
```

Codeblock 7. Using the `simulator` simulator for generating synthetic data.

model. Following the code, the variable `out` is now a three-tuple. The first element contains an array of RTs, the second contains an array of choices, and finally, the third element returns a dictionary of metadata concerning the simulation run. Next, we can access the LAN corresponding to our angle model directly by typing the code in Codeblock 8. We can utilize the `get_torch_mlp` function, which is defined in the `network_inspectors` submodule.

The `lan_angle` object defined in Codeblock 8 is in fact a method that defines the forward pass through a given LAN. It expects as input a matrix where each row defines a parameter vector suitable for the SSM of choice (here angle, so we need a value for each of the parameters `["v", "a", "z", "t", "theta"]`, which can be found in our `model_config` dictionary). Two elements are then added: an RT and a choice at which we would like to evaluate our likelihood. Codeblock 9 provides a full example.

To facilitate a simple sanity check, we provide the `kde_vs_lan_likelihoods` plot, which can be accessed from the `network_inspectors` submodule. This plot lets the user compare LAN likelihoods against empirical likelihoods from simulator data for a given matrix of parameter vectors (Fengler et al., 2021). The empirical likelihoods are defined via kernel density estimators (KDEs) (Silverman, 1986). We show an example in Codeblock 10. Figure 4 shows the output.

Fitting Data Using `HDDMnn`, `HDDMnnRegressor`, and `HDDMnnStimCoding` Classes

Using the `HDDMnn`, `HDDMnnRegressor`, and `HDDMnnStimCoding` classes, we can follow the general workflow established by the basic `HDDM` package to perform Bayesian inference. In this section, we will fit the angle model to the example data set provided with the HDDM package. Codeblock 11 shows us how to load the corresponding data set, after which we can set up our HDDM model and draw 1000 MCMC samples using the code in Codeblock 12.

We note a few differences between a call to construct an `HDDMnn` class and a standard `HDDM` class. First is the supply of the `model` argument specifying which SSM to fit (requires that this model is already available in HDDM; see above). Second is the inclusion of model-specific parameters under the `include` argument. The workflow is otherwise equivalent, a fact that is conserved for the `HDDMnnRegressor` and `HDDMnnStimCoding` classes. A third difference concerns the choice of argument defaults. The `HDDMnn` class uses noninformative priors, instead of the informative priors derived from the literature that form the default for the basic `HDDM` class. Because, as per our earlier discussions, variants of SSMs are historically rarely fit to experimental data, we cannot easily derive reasonable informative priors from the literature and therefore choose to remain agnostic in our beliefs about the parameters underlying a given data set. If the research community starts fitting SSM variants to experimental data, this state of affairs may evolve through collective learning. At this point, we caution the user to however not use these new models blindly. We strongly encourage conducting appropriate parameter recovery studies, specific to the experimental data set under consideration. We refer to the section on Inference Validation Tools below, for how HDDM might help in this procedure.

New Visualization Plots: `hddm.plotting`

On the basis of our model fit from the previous section, we illustrate a few new informative plots, which are now included in HDDM. We can generally distinguish between two types of plots: plots that use the traces only (to display posterior parameter estimates) and plots that make use of the model simulators (to display how well the model

```
from hddm.network_inspectors import get_torch_mlp
lan_angle = get_torch_mlp(model='angle')
```

Codeblock 8. Loading a torch network from the package.

```
# Make some random parameter set
from hddm.simulators import make_parameter_vectors_nn

parameter_df = make_parameter_vectors_nn(model='angle',
                        param_dict=None, n_parameter_vectors=1)

parameter_matrix = np.tile(np.squeeze(parameter_df.values), (200, 1))

# Initialize network input
network_input = np.zeros((parameter_matrix.shape[0], parameter_matrix.shape[1] + 2))

# Note the + 2 on the right we append the parameter vectors with
# reaction times (+1 columns) and choices (+1 columns)

# Add reaction times
network_input[:, -2] = np.linspace(0, 3, parameter_matrix.shape[0])

# Add choices
network_input[:, -1] = np.repeat(np.random.choice([-1, 1]), parameter_matrix.shape[0])

# Note: The networks expects float32 inputs
network_input = network_input.astype(np.float32)

# Show example output
print('Some network outputs')
print(lan_angle(network_input)[:10]) # printing the first 10 outputs
print('Shape')
print(lan_angle(network_input).shape) # original shape of output
```

```
Some network outputs
[[-2.9323568]
 [ 2.078088 ]
 [ 0.4104141]
 [-0.5943402]
 [-1.1136726]
 [-1.6901499]
 [-2.3512228]
 [-3.080151 ]
 [-3.8215086]
 [-4.4257374]]
Shape
(200, 1)
```

Codeblock 9. Check forward pass of supplied `angle` network.

can reproduce empirical data given posterior parameters). The first such plot is produced by the `plot_caterpillar` function, which presents an approximate posterior 99% highest density interval (specifically, we show the 1%–99% range in the cumulative distribution function of the posterior), for each parameter. Codeblock 13 shows us how to invoke this function, and Figure 5 illustrates the resulting plot.

The second such plot is the a posterior pair plot, called via the `plot_posterior_pair` function. This plot shows the pairwise posterior distribution, subject by subject (and, if provided, condition by condition). Codeblock 14 illustrates how to call this function, and Figure 6 exemplifies the resulting output.

A last very useful plot addition is what we call the "model plot," an extension to the standard posterior, predictive

```
from hddm.network_inspectors import kde_vs_lan_likelihoods

# Make a set of parameter vectors
parameter_df = make_parameter_vectors_nn(model=model, param_dict=None,
                        n_parameter_vectors=6)

# Generate plot
kde_vs_lan_likelihoods(parameter_df=parameter_df,  model=model,
                       n_samples=1000,  n_reps=10,  font_scale=1.25)
```

Codeblock 10. Example usage of the `kde_vs_lan_likelihood()` function to compare LAN likelihoods to empirical kernel-density estimates. KDE = kernel density estimator.

plot, which can be used to visualize the impact of the parameter posteriors on decision dynamics. For example, if one is estimating a linearly collapsing bound, instead of only interpreting the posterior angle parameter, one can see how that translates to the evolving decision bound over time in tandem with the estimating drift rate and so forth. It is an extension of the `plot_posterior_predictive` function. This function operates by manipulating `matplotlib axes` objects, via a supplied axes manipulator. The novel axis manipulator in the example shown in Codeblock 15 is the `_plot_func_model` function. Figure 7 shows the resulting plot.

We use this moment to illustrate how the angle model in fact outperforms the DDM on this example data set. For this purpose, we take an example subject from Figure 7 and contrast the posterior predictive of the angle model with the posterior predictive of the DDM side by side in Figure 8. We clearly see that the DDM model has trouble capturing the leading edge and the tail behavior of the RT distributions simultaneously, whereas the angle model

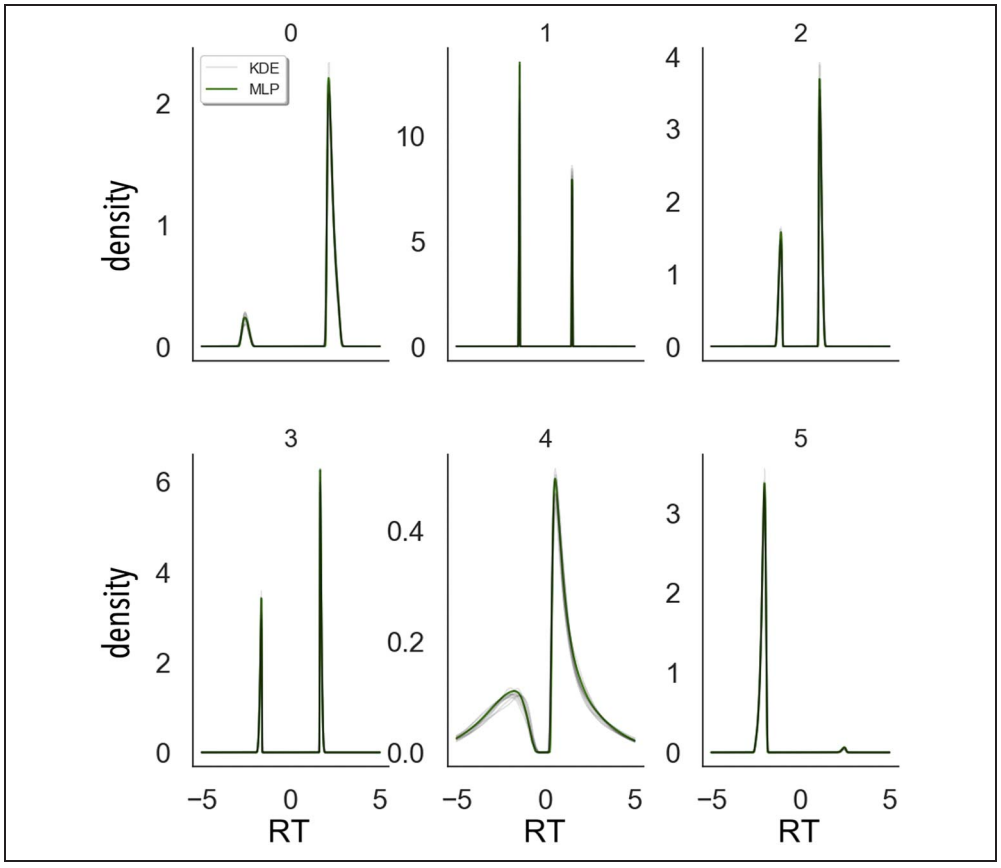


Figure 4. Example of a `kde_vs_lan_likelihoods` plot. If the green (deterministic) and gray (stochastic) lines overlap, then the approximate likelihood (MLP = multilayered perceptron, the neural network that provides our LAN) is a good fit to the actual likelihood. KDE = kernel density estimator.

```
cav_data = hddm.load_csv(hddm.__path__[0] + '/examples/cavanagh_theta_nn.csv')
```

```
      subj_idx stim     rt  response     theta  dbs conf
0            0   LL  1.210       1.0  0.656275    1   HC
1            0   WL  1.630       1.0 -0.327889    1   LC
2            0   WW  1.030       1.0 -0.480285    1   HC
3            0   WL  2.770       1.0  1.927427    1   LC
4            0   WW  1.140       0.0 -0.213236    1   HC
...        ...  ...    ...       ...       ...  ...  ...
3983        13   LL  1.450       0.0 -1.237166    0   HC
3984        13   WL  0.711       1.0 -0.377450    0   LC
3985        13   WL  0.784       1.0 -0.694194    0   LC
3986        13   LL  2.350       0.0 -0.546536    0   HC
3987        13   WW  1.250       1.0  0.752388    0   HC

[3988 rows x 7 columns]
```

Codeblock 11. Loading package-supplied **Cavanagh** data set.

```
hddmnn_model_cav = hddm.HDDMnn(cav_data, model='angle', include=['z', 'theta'],
                               is_group_model = True)
hddmnn_model_cav.sample(1000, burn=500)
```

```
 [-----------------100%-----------------]
 1001 of 1000 complete in 365.3 sec
```

Codeblock 12. Sampling from an HDDMnn model.

strikes a much better balance. Although this example does not present a fully rigorous model comparison (deviance information criterion scores, for example, however bear out the same conclusion) exercise, it provides a hint at the benefits one may expect from utilizing an expanded model space.

Inference Validation Tools: `simulator_h_c()`

Validating that a model is identifiable on simulated data is an important aspect of a trustworthy inference procedure (Tran, Van Maanen, Heathcote, & Matzke, 2021; Evans, Trueblood, & Holmes, 2020; Wilson & Collins, 2019; Holmes & Trueblood, 2018). We have two layers of uncertainty in this regard. First, LANs are approximate likelihoods. A model that is otherwise identifiable could in principle lose this property when using LANs to estimate its parameters from a data set, should the LAN not have been trained adequately. Second, a given model can inherently be unidentifiable for a given data set and/or theoretical commitments (regardless of whether its likelihood is analytic or approximate). As a simple example, consider an experimental data set that does not include enough samples to identify the parameter of a model of interest with any degree of accuracy. Slightly more involved, the posterior could tend to be multimodal, a problem for MCMC samplers that can lead to faulty inference. Although increasing the number of

```
from hddm.plotting import plot_caterpillar
plot_caterpillar(hddm_model=hddmnn_model_cav, figsize=(8, 8), columns=3)
```

Codeblock 13. Example usage of the `caterpillar_plot()` function.

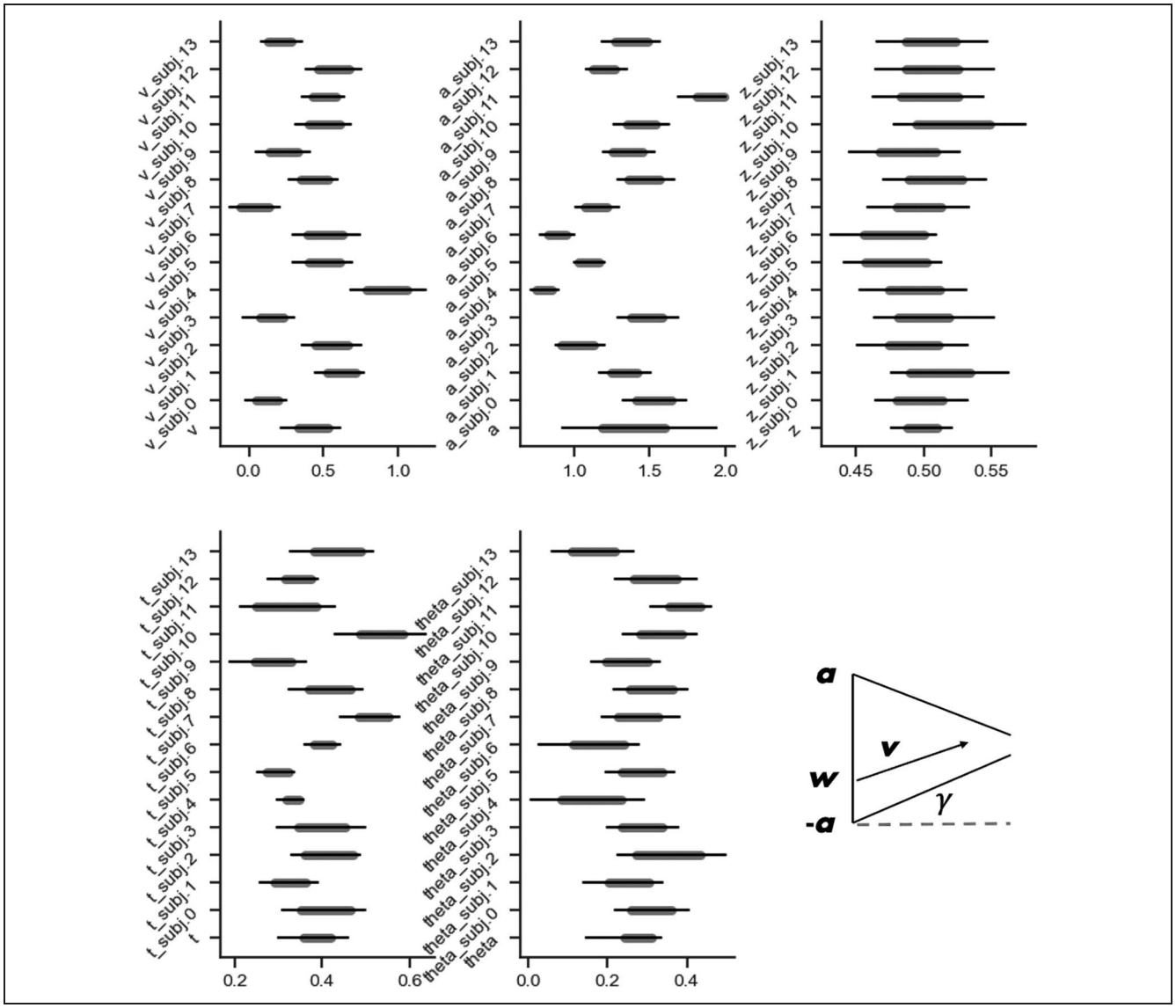


Figure 5. Example of a `caterpillar_plot`. The plot, split by model parameters, shows the 99% (line ends) and 95% (gray band ends) highest density intervals of the posterior for each parameter. Multiple styling options exist.

trials in an experiment and/or increasing the number of participants can help remedy this situation, this is not a guarantee. Apart from the size and structure of the empirical data set, our modeling commitments play an important role for identifiability too. As an example, we might have experimental data from a random dot motion task, and we are interested in modeling the choices and RTs of participating subjects with our angle model. A reasonable assumption is that the v parameter (a rough proxy for processing speed) differs depending on the difficulty of the trial. However, the parameters t and a may not depend on the difficulty because we do not have a good a priori theoretical reason to suspect that the nondecision time (t) and the initial the boundary separation a (the degree of evidence expected to take a decision) will differ across experimental conditions. These

```
from hddm.plotting import plot_posterior_pair
plot_posterior_pair(hddmnn_model_cav, samples=500, figsize=(6, 6))
```

Codeblock 14. Example concerning usage of the `plot_posterior_pair()` function.

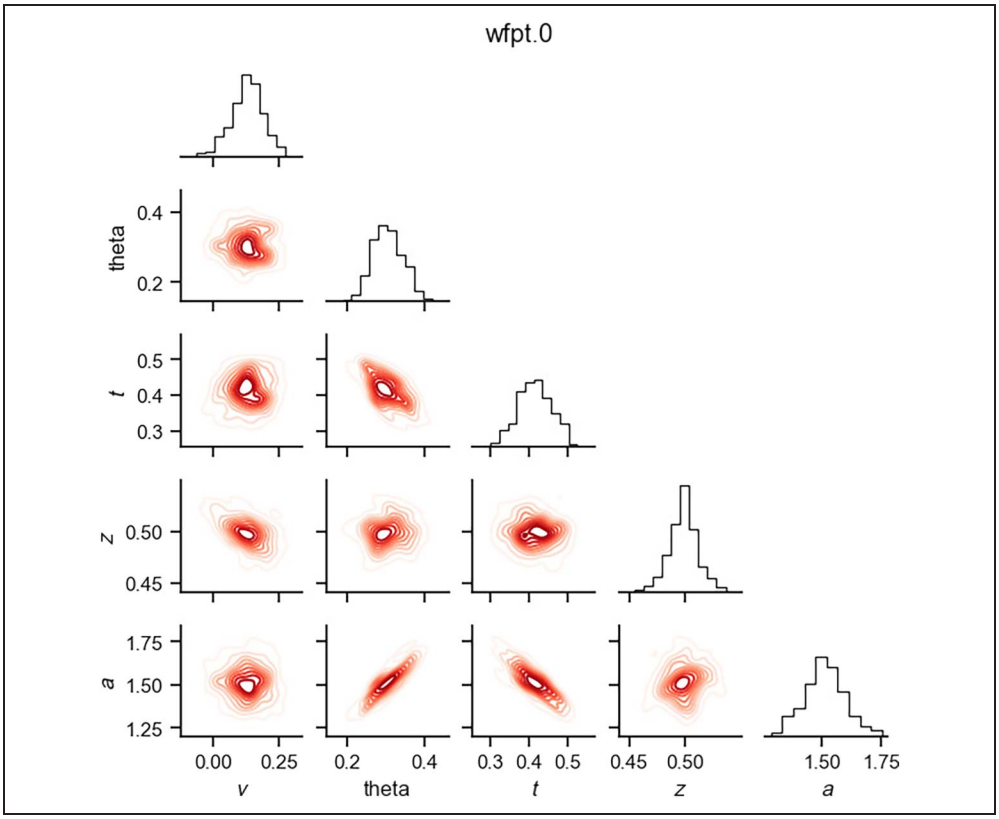


Figure 6. Example of a `posterior_pair_plot` in the context of parameter recovery. The plot is organized per stochastic node (here, grouped by the `"subj_idx"` column where in this example `"subj_idx" = "0"`). The diagonal shows the marginal posterior of a given parameter as a histogram. The elements below the diagonal show pairwise posteriors via (approximate) level curves. These plots are especially useful to identify parameter collinearities, which indicate parameter-trade-offs and can hint at issues with identifiability. This example shows how the `theta` (boundary collapse) and `a` (boundary separation) parameters as well as the `t` (nondecision time) and `a` parameters trade-off in the posterior. We refer to Fengler and colleagues (2021) for parameter recovery results using the underlying angle SSM. We note that such parameter trade-offs and attached identifiability issues not only derive from a given likelihood model but are also affected by the data and parameter structure as task design and modeling choices.

commitments are embedded in the model itself (they are assumptions on the data-generating process imposed by the modeler) and determine jointly with an experimental data set whether inference can be successful. For a modeler, it is therefore of paramount importance to check whether their chosen combinations of theoretical commitments and experimental data set jointly lead to an inference procedure that is accurate. Because the space of models incorporated into HDDM has been significantly expanded with the LAN extension, we provide a few tools to help facilitate parameter recovery studies, which are relevant to real experimental data analysis and plan to supplement these tools even further in the future.

First, we provide the `simulator_h_c` function, in the `hddm_dataset_generators` submodule. The function is quite flexible; however, we will showcase a particularly relevant use case. Taking our `cav_data` data set loaded previously, we would like to generate data from our angle model in such a way that we encode assumptions about our model into the generated data set. In the example below, we assume that the *v* and *theta*

```
from hddm.plotting import plot_posterior_predictive

plot_posterior_predictive(model = hddmnn_model_cav, columns=3,   figsize=(10, 12),
                          groupby=['subj_idx'], value_range=np.arange(0.0, 3, 0.1),
                          plot_func=hddm.plotting._plot_func_model,
                          **{'alpha': 0.01, 'ylim': 3, 'samples': 200, 'legend_fontsize': 7.,
                             'legend_location': 'upper left',
                             'add_posterior_uncertainty_model': True,
                             'add_posterior_uncertainty_rts': False,
                             'subplots_adjust': {'top': 0.94, 'hspace': 0.35, 'wspace': 0.3}
                          })
```

Codeblock 15. Example usage of the `plot_posterior_predictive()` function.

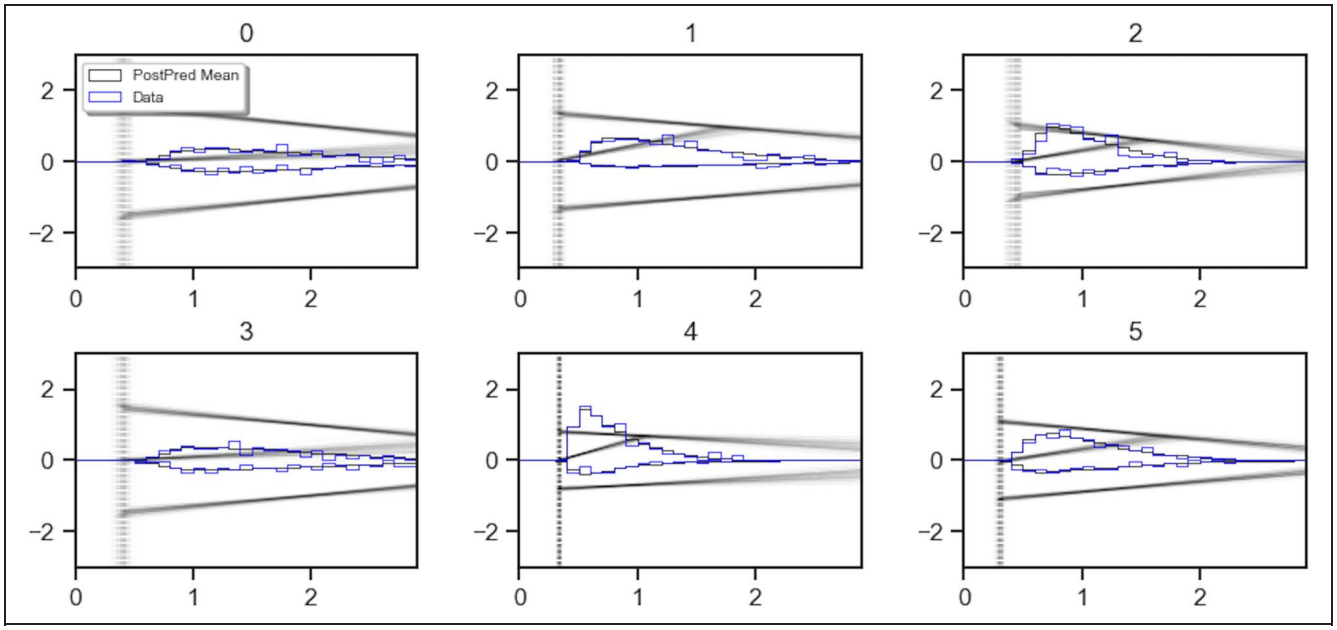


Figure 7. Example of a `model_plot`. This plot shows the underlying data in blue, with choices and RTs presented as histograms (positive y axis for Choice Option 1, negative y axis for Choice Option 0 or −1). The black histograms show the RTs and choices under the parameters corresponding to the posterior mean. In addition, the plot shows a graphical depiction of the model corresponding to parameters drawn from the posterior distribution in black. Various options exist to add and drop elements from this plot; the provided example corresponds to what we consider the most useful settings for purposes of illustration. Note that, in the interest of space, we only illustrate the first six subjects here. PostPred = posterior predictive.

parameters vary as a function of the `"stim"` column. For each value of `"stim"`, a group-level μ and σ (defining the mean and standard deviation of a group-level normal distribution) are generated, and subject-level parameters are sampled from this group distribution. This mirrors exactly the modeling assumptions when specifying an HDDM model with the `depends_on` argument set to {`"v"`: `"stim"`, `"theta"`: `"stim"`}.

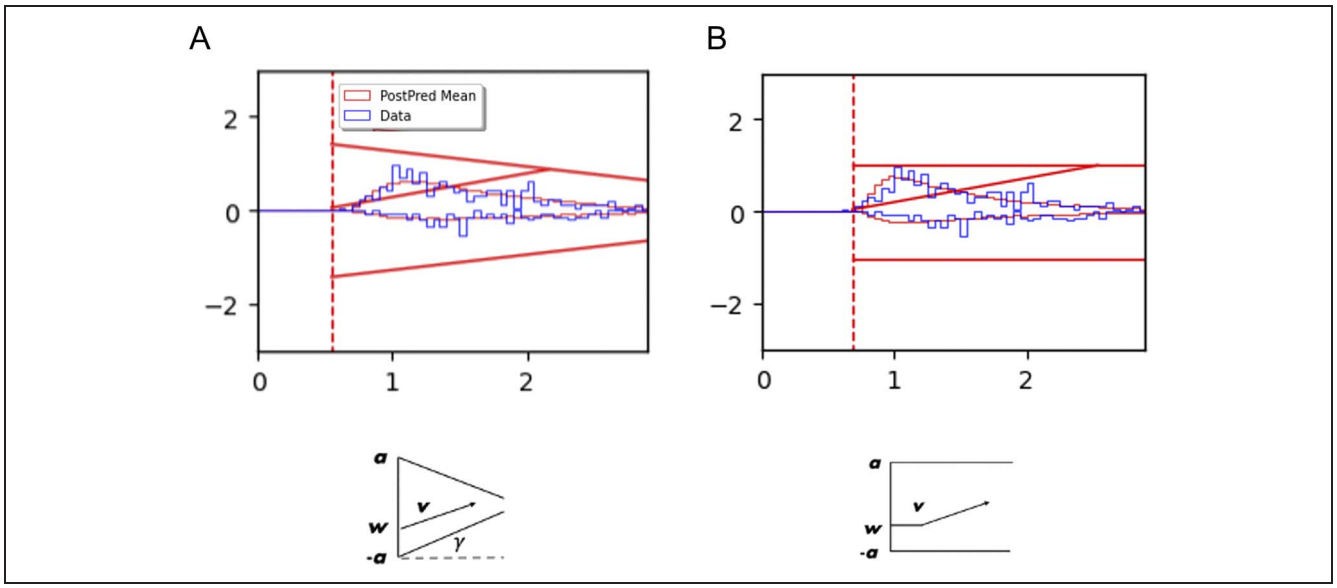


Figure 8. Contrasting the posterior predictive of the angle and DDM model on an example subject. A shows the angle model, and (B) shows the DDM. Although the fits are not dramatically better for this data set (in our experience, more extreme differences can be seen in other cases), the angle model shows two characteristic differences to the DDM model fit. First, it better captures the graceful initial increase in density for short RTs. Second, it captures the slower decrease in density for longer RTs, as compared to the DDM, for which the RT density falls off quicker than is apparent in the data. Both of these effects are directly produced by allowing a collapsing bound, instead of the DDM's static, parallel bounds. PostPred = posterior predictive.

```
# Generate some data
from hddm.simulators.hddm_dataset_generators import simulator_h_c
sim_data, parameter_dict = simulator_h_c(data=cav_data, model='angle',
                                         p_outlier=0.00,
                                         depends_on={'v': ['stim'], 'theta': ['stim']},
                                         regression_models=None,
                                         regression_covariates=None,
                                         group_only_regressors=False,
                                         group_only=None, fixed_at_default=None)

sim_data
```

```
      subj_idx stim        rt  response     theta  dbs conf         v  \
0            0   LL  2.020890       1.0  0.442532    1   HC -0.474451
1            0   WL  2.075889       1.0  0.844691    1   LC -0.865643
2            0   WW  2.119889       1.0  0.661660    1   HC  0.752663
3            0   WL  1.804893       0.0  0.844691    1   LC -0.865643
4            0   WW  2.410885       1.0  0.661660    1   HC  0.752663
...        ...  ...       ...       ...       ...  ...  ...       ...
3983        13   LL  2.874057       1.0  0.402371    0   HC -0.473813
3984        13   WL  2.169051       0.0  0.972350    0   LC -1.001207
3985        13   WL  1.798055       0.0  0.972350    0   LC -1.001207
3986        13   LL  1.709054       1.0  0.402371    0   HC -0.473813
3987        13   WW  2.115052       1.0  0.911009    0   HC  0.824063

             a         z         t
0     1.402356  0.577363  1.468893
1     1.402356  0.577363  1.468893
2     1.402356  0.577363  1.468893
3     1.402356  0.577363  1.468893
4     1.402356  0.577363  1.468893
...        ...       ...       ...
3983  1.283326  0.616165  1.618054
3984  1.283326  0.616165  1.618054
3985  1.283326  0.616165  1.618054
3986  1.283326  0.616165  1.618054
3987  1.283326  0.616165  1.618054

[3988 rows x 11 columns]
```

Codeblock 16. Using the `simulator_h_c()` function.

Codeblock 16 provides an example on how to call this function. The `simulator_h_c` function returns the respective data set (here `sim_data`) exchanging values in the previous RT and response columns with simulation data. Trial-by-trial parameters are attached to the data frame as well. The `parameter_dict` dictionary contains all the parameters of the respective hierarchical model that was used to generate the synthetic data. This parameter dictionary follows the parameter naming conventions of HDDM exactly. We can fit this data using the `HDDMnn` class as illustrated in Codeblock 17.

The plots defined in the previous section allow us to specify a `parameter_recovery_mode`, which we can utilize to check how well our estimation worked on our synthetic data set. Codeblocks 18, 19, and 20 and Figures 9, 10, and 11 show, respectively, code and plot examples.

Note how both the `plot_posterior_pair` function and the `plot_posterior_predictive` function take the `parameter_recovery_mode` argument to add a ground truth to the visualization automatically (the ground truth is expected to be included in the data set attached to the HDDM model itself). The `plot_caterpillar` function needs a `ground_truth_parameter_dict` argument to add the ground-truth parameters. The `simulator_h_c` function provides such

```
hddmnn_model_sim = hddm.HDDMnn(sim_data, model='angle',
                               is_group_model=True, p_outlier=0.00,
                               include=['v', 'a', 't', 'z', 'theta'],
                               depends_on = {'v': ['stim'], 'theta': ['stim']},
                              )
hddmnn_model_sim.sample(1000, burn=500)
```

```
 [-----------------100%-----------------]
 1001 of 1000 complete in 1436.2 sec
```

Codeblock 17. Fitting an `HDDMnn` model to synthetic data.

```
from hddm.plotting import plot_caterpillar
plot_caterpillar(hddm_model=hddmnn_model_sim,
                 ground_truth_parameter_dict=parameter_dict,
                 figsize=(10, 15), y_tick_size=6, columns=3)
```

Codeblock 18. `caterpillar plot` for fit to simulated data.

```
from hddm.plotting import plot_posterior_predictive
plot_posterior_predictive(model = hddmnn_model_sim, columns = 3, figsize = (10, 12),
                          groupby = ['subj_idx'], value_range = np.arange(0.0, 3, 0.1),
                          plot_func = hddm.plotting._plot_func_model,
                          parameter_recovery_mode = True,
                          **{'alpha': 0.01, 'ylim': 3,
                          'add_model': True, 'samples': 200,
                          'legend_fontsize': 7., 'legend_location': 'upper left',
                          'add_posterior_uncertainty_rts': False,
                          'add_posterior_uncertainty_model': True,
                          'add_posterior_mean_model': True,
                          'add_posterior_mean_rts': True,
                          'subplots_adjust': {'top': 0.94, 'hspace': 0.35, 'wspace': 0.3}
                          })
```

Codeblock 19. `model_plot` for fit to simulated data.

```
from hddm.plotting import plot_posterior_pair
posterior_pair_plot(hddmnn_model_sim, parameter_recovery_mode = True,
                    samples=500, figsize=(6, 6))
```

Codeblock 20. `posterior_pair_plot` for fit to simulated data.

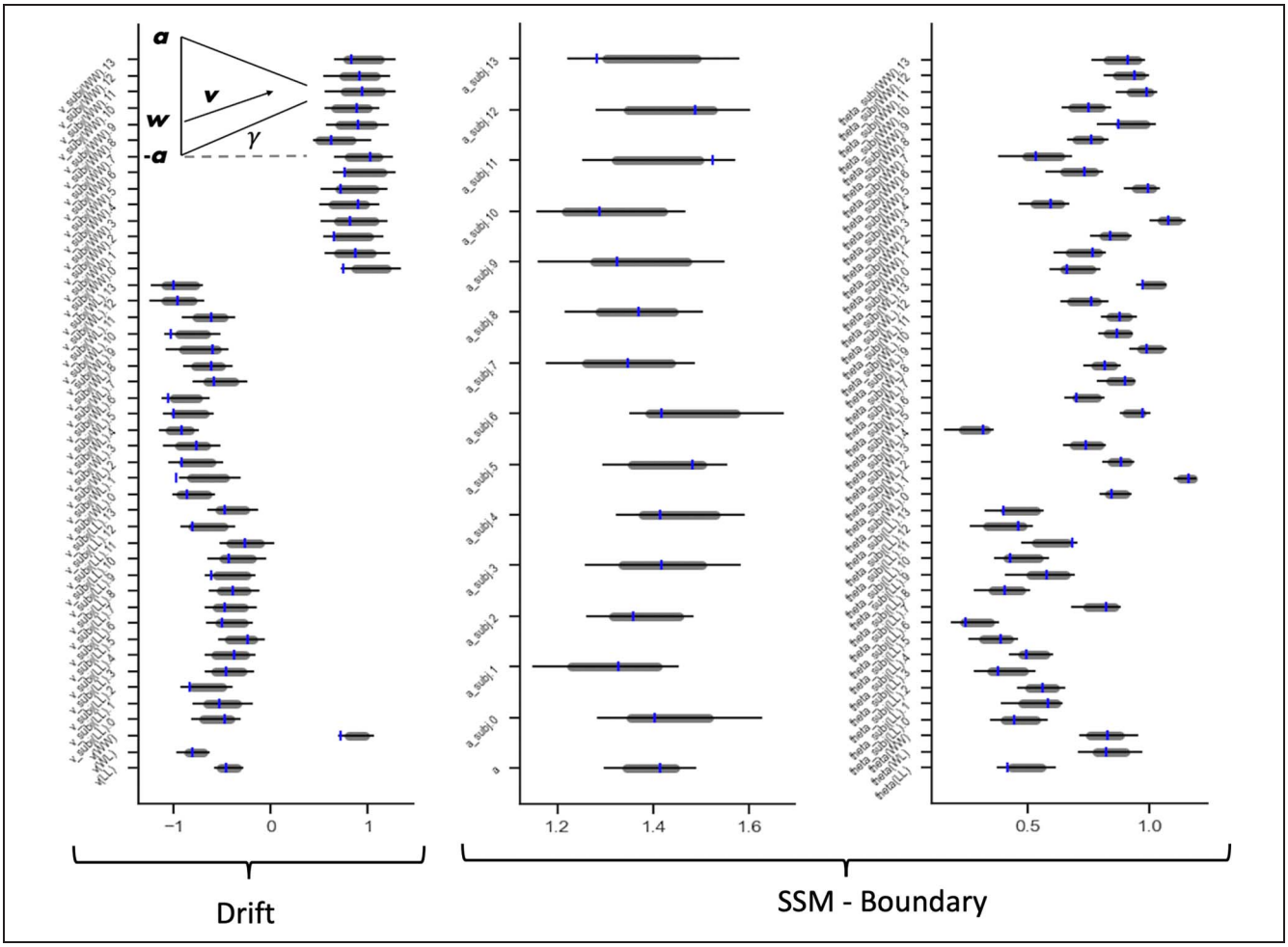


Figure 9. Example of a `caterpillar_plot`. The plot is split by model parameter kind, showing parameter-wise, the 99% (line ends) and 95% (gray band ends) highest density intervals (HDIs) of the posterior. In the context of parameter recovery studies, the user can provide ground-truth parameters to the plot, which will be shown as blue tick marks on top of the HDIs. Multiple styling options exist. Note, in the interest of space, we show only three of the five basic parameters here: [`"v"`, `"a"`, `"theta"`] of the underlying model (leaving out [`"z"`, `"t"`]).

a compatible dictionary of ground-truth parameters. Using the set of tools in this section, we hope that HDDM conveniently facilitates application-relevant parameter recovery studies.

Adding to the Bank of SSMs: User-Supplied Custom Models

The new models immediately available for use with HDDM are just the beginning. HDDM allows users to define their own models via adjusting the `model_config` and the provision of custom likelihood functions. The goal of this functionality is twofold. First, we aim to make HDDM maximally flexible for advanced users, cutting down red tape to allow creative usage. Second, we hope to motivate users to follow through with a two-step process of model integration. Step 1 involves easy testing of new likelihoods through HDDM, however, with somewhat limited auxiliary functionality (one can generate plots based on the posterior traces, but other plots will not work because of the lack of a simulator). Step 2 involves sharing the model likelihood and a suitable simulator with the community to allow full integration with HDDM as well as other similar toolboxes that operate across programming languages and probabilistic programming frameworks. In future work, we hope to flesh out a pipeline that allows users to follow a simple sequence of steps to full integration of their custom models with HDDM. Here, we show how to complete Step 1, defining an `HDDMnn` model with a custom likelihood to allow fitting a new model through HDDM. See the section on Future Work for some guidance on producing your own LAN, or contact the authors.

We start with configuring the `model_config` dictionary. We add a `"custom"` key and assign the specifics of our new model. For illustration purposes, we will add the angle model to HDDM (although it is already provided with the LAN extension). In addition, we need to define a basic likelihood function that takes in a vector (or

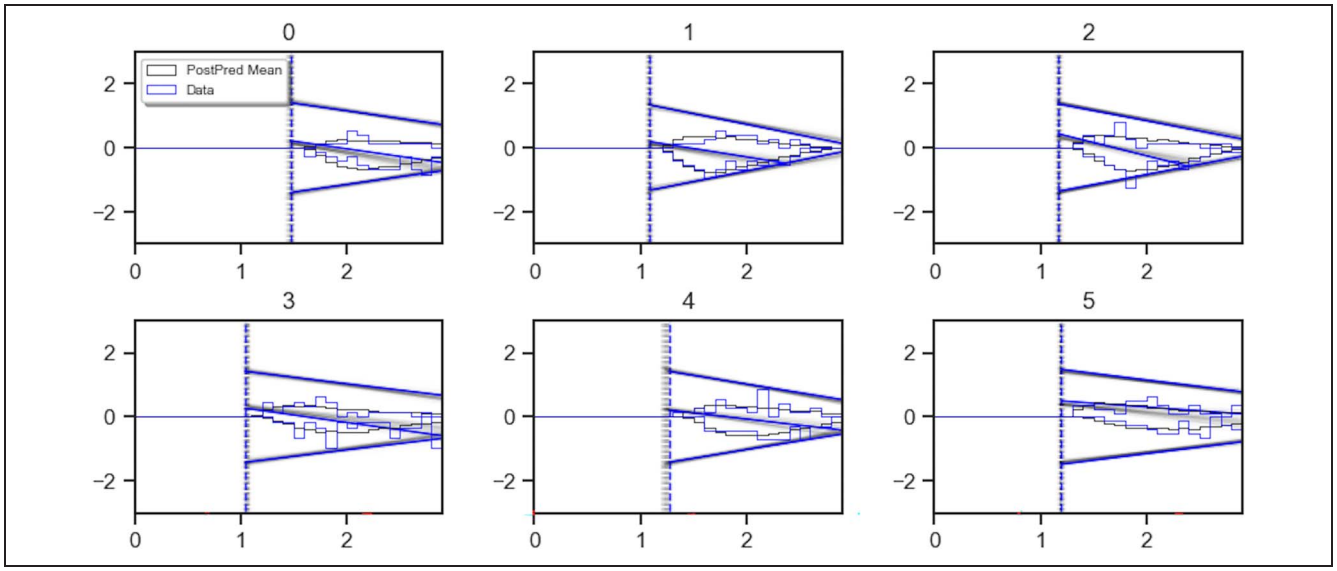
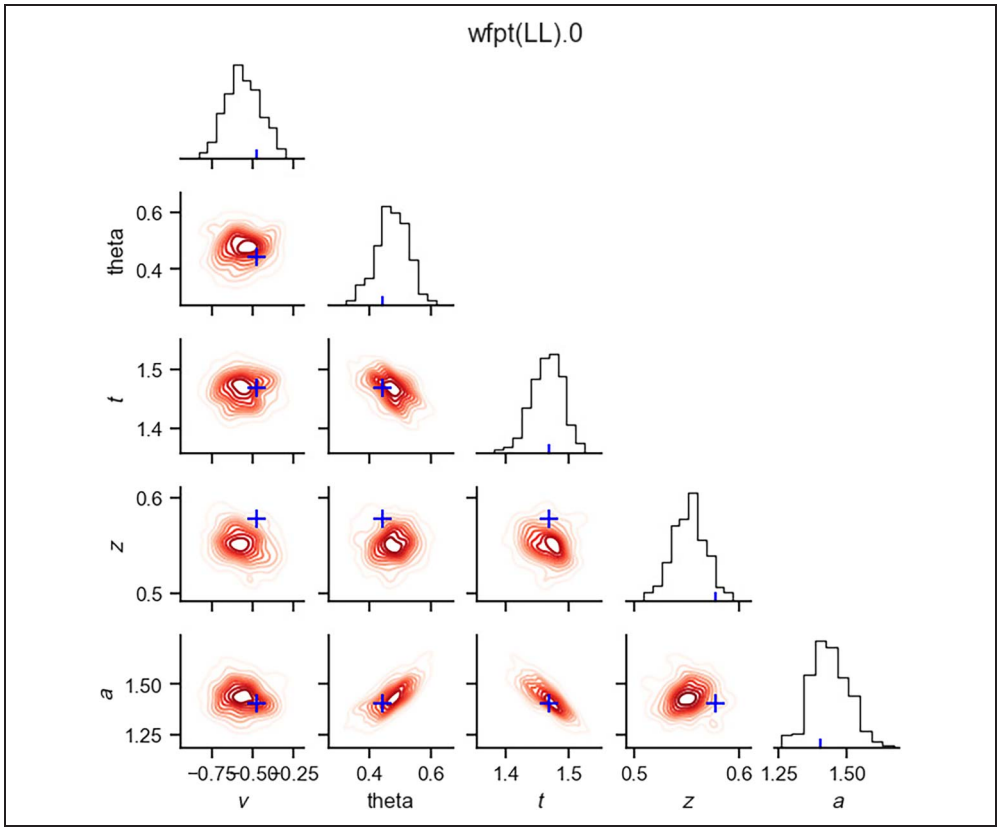


Figure 10. Example of a `model_plot`. This plot shows the underlying data in blue, with choices and RTs presented as a histogram (positive *y* axis for Choice Option 1, negative *y* axis for Choice Option 0 or −1). The black histograms show the RTs and choices under the parameters corresponding to the posterior mean. In addition, the plot shows a graphical depiction of the model corresponding to parameters drawn from the posterior distribution in black, as well as such a depiction for the ground-truth parameters in blue, in case these were provided (e.g., if one is performing recovery from simulated data). Inclusion of the ground-truth parameters distinguishes the present display from Figure 7. Various options exist to add and drop elements from this plot; the provided example corresponds to what we consider the most useful settings for purposes of illustration. Note that, in the interest of space, we only illustrate the first six subjects here. PostPred = posterior predictive.

Figure 11. Example of a `posterior_pair_plot` in the context of parameter recovery. The plot is organized per stochastic node (here, grouped by the `"stim"` and `"subj_idx"` columns where in this example (`"stim"` = `"LL"`, `"subj_idx"` = `"0"`). The diagonal shows the marginal posterior of a given parameter as a histogram, adding the ground-truth parameter as a blue tick mark. The elements below the diagonal show pairwise posteriors via (approximate) level curves and add the respective ground truths as a blue cross.



```
from hddm.torch.mlp_inference_class import load_torch_mlp
# Define our custom model config
my_model_config = {
            "params": ["v", "a", "z", "t", "theta"],
            "params_trans": [0, 0, 1, 0, 0],
            "params_std_upper": [1.5, 1.0, None, 1.0, 1.0],
            "param_bounds": [[-3.0, 0.3, 0.1, 1e-3, 0.0], [3.0, 2.5, 0.9, 2.0, 1.1]],
            "boundary":  hddm.simulators.bf.constant,
            "params_default": [0.0, 1.0, 0.5, 1e-3],
            "hddm_include": ["z"],
            "choices": [-1, 1],
            "slice_widths": {"v": 1.5, "v_std": 1, "a": 1, "a_std": 1,
                             "z": 0.1, "z_trans": 0.2, "t": 0.01, "t_std": 0.15},
            }
# Load our custom network (here we load one supplied by HDDM)
custom_network = load_torch_mlp(model='angle')
# Define HDDM model
hddm_model_custom = hddm.HDDMnn(data=data, include=["z", "theta"],
                                model='custom', model_config=my_model_config,
                                network=custom_network)
# Sample from the HDDM model
hddm_model_custom.sample(1000, burn=500)
```

```
 [-----------------100%-----------------]
 1001 of 1000 complete in 365.3 sec
```

Codeblock 21. Construct and fit an HDDMnn model using a custom likelihood.

matrix/2d `numpy array`) of parameters, ordered according to the list in the `"params"` key above. As an example, we load our LAN for the angle model (as supplied by HDDM) as if it is a custom network. Finally, we can fit our newly defined custom model. Codeblock 21 illustrates the whole process.

Note that the only difference to a normal call to the `hddm.HDDMnn` class is supplying appropriate model specifications for our custom likelihood. We supply the `model` argument as `"custom"` alongside our own configuration dictionary to the `model_config` argument. In addition, we explicitly pass to the `network` argument, our `custom_network` defining the likelihood.

Moreover, we note that the supply of custom networks opens up multiple degrees of freedom to explore improved likelihood approximations. As an example, users may utilize LANs trained on the log-RT distributions instead of the original RT distributions of an SSM.

Combining SSMs with RL

Although the previous sections focused on employing SSMs in modeling stationary environments, a host of commonly applied experimental task paradigms involve some form of learning that results from the agent's interactions with the environment. Although SSMs can be used to model the decision processes, we need additional machinery to capture the learning dynamics that arise while participants perform such tasks. RL (Sutton & Barto, 2018) is one computational framework that can allow us to account for such learning processes. In RL, researchers typically assume a simple *softmax* choice rule, informed by some "utility" (or "goodness") measure of taking a particular action in a given state. Mathematically, the choice probabilities are expressed as

$$p_{\text{action},i}(t) = \frac{e^{q_{\text{action},i}(t)}}{\sum_j e^{q_{\text{action},j}(t)}}$$

Although RL models can account for learning dynamics in basic choice behavior, the choice functions commonly employed (e.g., softmax) cannot capture the RT. To combine the strengths of SSMs and RL models, recent studies have used the DDM to jointly model choice and RT distributions during learning (Pedersen & Frank, 2020; Fontanesi et al., 2019; Pedersen et al., 2017). Such an approach allows researchers to study not only the across-trial dynamics of learning but also the within-trial dynamics of choice processes, using a single model. The main idea behind these models is to allow an RL process to drive the trial-by-trial parameters of an SSM (such as the basic

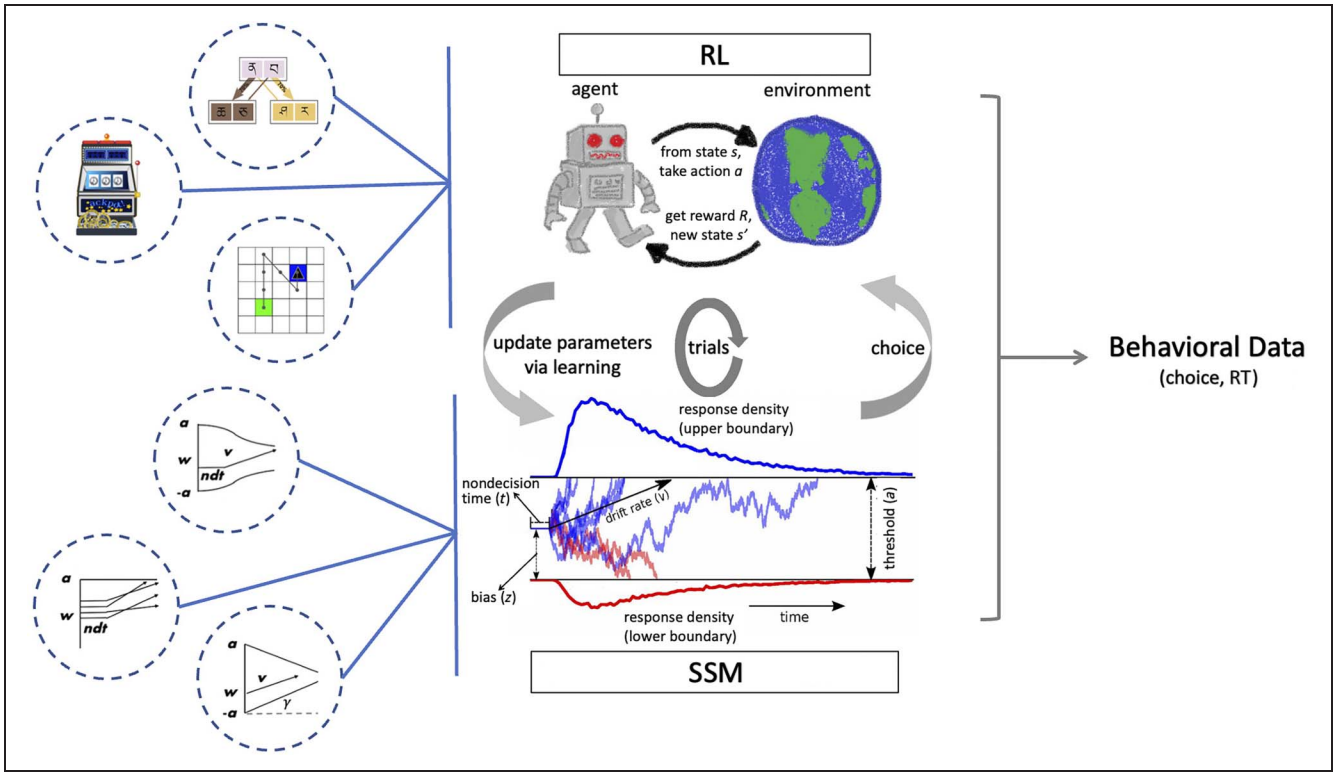


Figure 12. RL–SSM: combining RL and SSMs.

DDM), which in turn is used to jointly capture RT and choice behavior for a given trial. This can be applied in complex tasks that involve learning from feedback (see Figure 12). This results in a much more broadly applicable class of models and naturally lends itself for use in computational modeling of numerous cognitive tasks where the learning process informs the decision-making process. Indeed, a recent study showed that the joint modeling of choice and RT data can improve parameter identifiability of RL models, by providing additional information about choice dynamics (Ballard & McClure, 2019). However, to date, such models have been limited by the form of the decision model. Many RL tasks involve more than two responses, making the DDM inapplicable. Similarly, the assumption of a fixed threshold may not be valid. For example, during the early learning phase, the differences in q-values, and hence drift rates, will be close to zero, and there is little value in accumulating evidence. A standard DDM model would predict that such choices are associated with very-long-tail RT distributions. A more appropriate assumption would be that learners use a collapsing bound so that, when no evidence is present, the decision process can terminate.

Utilizing the power of LANs, we can further generalize the RL–DDM framework to include a much broader class of SSMs as the decision-making process. The rest of this section provides some details and code examples for these new RL-SSMs.

Test Bed

We test our method on a synthetic data set of the two-armed bandit task with binary outcomes. However, our approach can be generalized to any n-armed bandit task given a pretrained LAN that outputs likelihoods for the corresponding n-choice decision process (e.g., race models). The model employed a simple delta learning rule (Rescorla, 1972) to update the action values:

$$q_{\text{action},i}(t+1) = q_{\text{action},i}(t) + \alpha^* \left[r(t) - q_{\text{action},i}(t) \right]$$

where $q_{\text{action}}(t)$ denotes expected reward (q-value) for the chosen action at time t, $r(t)$ denotes reward obtained at time t, and α (referred to as rl_alpha in the result plots) denotes the learning rate. The trial-by-trial drift rate depends on the expected reward value learned by the RL rule. The drift rate is therefore a function of q-value updates and is computed by the following linking function:

$$v(t) = \left[q_{\text{action},1}(t) - q_{\text{action},2}(t) \right] * s$$

where s is a scaling factor of the difference in q-values. In other words, the scalar s is the drift rate when the difference between the q-values of both the actions is exactly 1 (note that we refer to the scalar s as v in the corresponding figure). We show an example parameter recovery plot for this Rescorla–Wagner learning model connected to an SSM with collapsing bound in Figure 13.

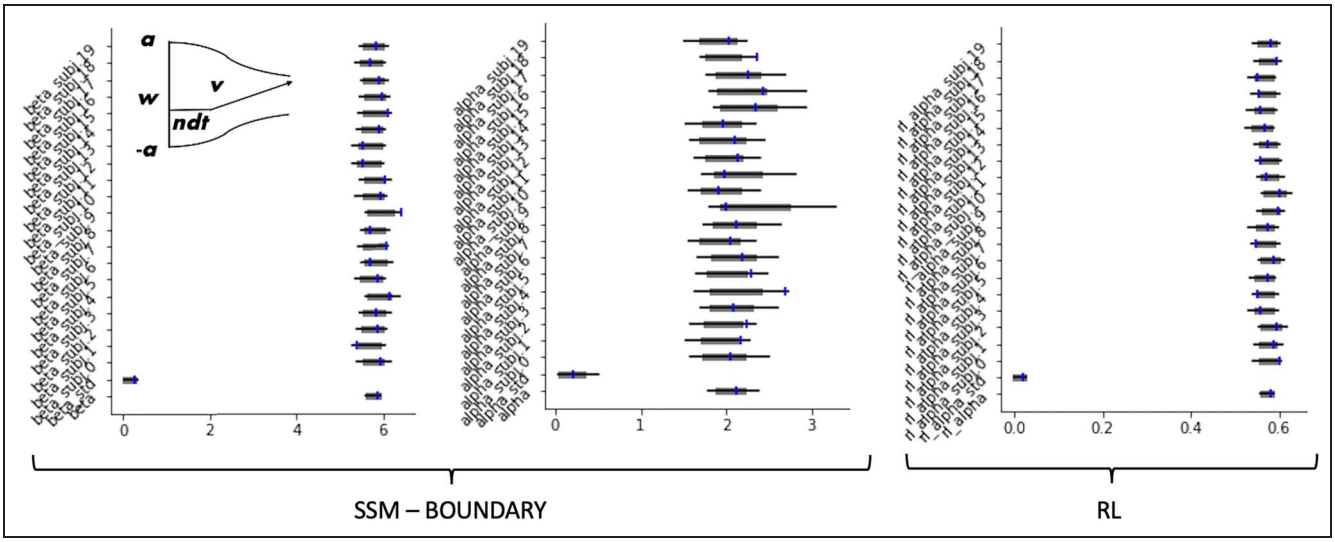


Figure 13. Parameter recovery on a sample synthetic data set using the RL + Weibull model. Posterior distributions for subject-level and group-level parameters are shown using caterpillar plots. The thick black lines correspond to 5th–95th percentiles; thin black lines correspond to 1st–99th percentiles. The blue tick marks show the ground-truth values of respective parameters. Note that, in the interest of space, we show only a subset of the parameters of the model—the two boundary parameters alpha and beta and the RL rate rl_alpha.

Model Definitions for RL with `model_config_RL`

Just like the `model_config`, the new HDDM version includes `model_config_rl`, which is the central database for the RL models used in the RLSSM settings. Below is an example for simple Rescorla–Wagner updates (Rescorla, 1972), a basic RL rule. The learning rate (referred to as rl_alpha in Figure 13 to avoid nomenclature conflicts with the alpha parameter in some SSMs) is the only parameter in the update rule. We do not transform this parameter (`"params_trans"` is set to 0) and specify the parameter bounds for the sampler as [0, 1]. Note that, for hierarchical sampling, the learning rate parameter α is transformed internally in the package. Therefore, the output trace for the learning rate parameter must be transformed by an inverse-logit function,

$$\frac{1}{1+e^{-\alpha}}$$

to get the learning rate values back in range [0, 1]. Codeblock 22 shows us an example of such a `model_config_rl` dictionary.

Analyzing Instrumental Learning Data: The `HDDMnnRL` Class

Running `HDDMnnRL` presents only a few slight adjustments compared to the other `HDDM` classes. First, the data frame containing the experimental data should be properly formatted. For every subject in each condition, the trials must be sorted in ascending order to ensure proper RL updates. The column `split_by` identifies each row with a specific task condition (as integer). The `feedback` column gives the reward feedback on the current trial, and `q_init` denotes the initial q values for the model. The rest of the data columns are the same as in other `HDDM` classes. Codeblock 23 provides an example.

We can fit the data loaded in Codeblock 23 using the `HDDMnnRL` class. We showcase such a fit using the Weibull model in conjunction with the classic Rescorla–Wagner learning rule (Rescorla, 1972). The `HDDMnnRL` class definition (shown in Codeblock 24) takes a few additional arguments compared to the `HDDMnn` class: `"rl_rule"` specifies the RL update rule to be used, and `non_centered` flag denotes if the RL parameters should be reparameterized to avoid troublesome

```
hddm.model_config_rl.model_config_rl["RWupdate"]  =
    {
        "doc": "Rescorla-Wagner update rule.",
        "params": ["rl_alpha"],
        "params_trans": [0],
        "params_std_upper": [10],
        "param_bounds": [[0.0], [1.0]],
        "params_default": [0.5],
    }
```

Codeblock 22. `model_config` definition for RL–SSM models.

```
import pandas as pd
data = pd.read_csv(hddm.__path__[0] + '/examples/demo_HDDMnnRL/rlssm_data.csv')
```

```
      response  rt         feedback   subj_idx split_by trial    q_init
0        0.0    2.729579   0.0        0        0        1        0.5
1        1.0    3.090593   1.0        0        0        2        0.5
2        1.0    3.892617   1.0        0        0        3        0.5
3        1.0    2.429583   1.0        0        0        4        0.5
4        1.0    2.566581   1.0        0        0        5        0.5
...      ...    ...        ...        ...      ...      ...      ...
29995    1.0    3.381547   1.0        19       2        496      0.5
29996    1.0    3.324544   0.0        19       2        497      0.5
29997    1.0    3.132535   0.0        19       2        498      0.5
29998    0.0    3.206539   0.0        19       2        499      0.5
29999    1.0    5.009474   0.0        19       2        500      0.5

[30000 rows ×7 columns]
```

Codeblock 23. Reading in RL–SSM example data.

```
rlssm_model = hddm.HDDMnnRL(data, model="weibull", rl_rule="RWupdate",
                            non_centered=True, p_outlier=0.0,
                            include=["z", "alpha", "beta", "rl_alpha"],
                           )
rlssm_model.sample(3000, burn=1500)
```

Codeblock 24. Constructing and sampling from an HDDMnnRL model.

sampling from the neck of the funnel of probability densities (Betancourt & Girolami, 2013; Papaspiliopoulos, Roberts, & Sköld, 2007). Figure 13 shows a caterpillar plot to verify the LAN-based parameter recovery on a sample RL–SSM model.

Neural Regressors for RL–SSM with the `HDDMnnRLRegressor` Class

The new `HDDMnnRLRegressor` class is aimed at capturing even richer (learning or choice) dynamics informed by neural activity, just like the `HDDMnnRegressor` class described above for basic SSMs. The extension works the same as the bespoke `HDDMnnRegressor` class, except that the model is now informed by an RL process to account for the across-trial dynamics of learning. The method allows estimation of the parameters (coefficients and intercepts) linking the neural activity in a given region and time point to the RL–SSM parameters.

The usage of `HDDMnnRLRegressor` class is the same as `HDDMnnRL` class except that our data frame will now have additional column(s) for neural (or other, e.g., EEG, pupil dilation) trial-by-trial covariates. Just as with the `HDDMnnRegressor` class, the model definition will also include specifying regression formulas that link covariates to model parameters. For example, if the boundary threshold parameter a is dependent on some neural measure *neural_reg*, Codeblock 25 shows us how to specify a corresponding `HDDMnnRLRegressor` model.

```
rlssm_reg_model = hddm.HDDMnnRLRegressor(data, 'a ~ 1 + neural_reg', model="weibull",
                                         rl_rule="RWupdate", p_outlier=0.0
                                         include=["z", "alpha", "beta", "rl_alpha"],
                                        )
rlssm_reg_model.sample(3000, burn=1500)
```

Codeblock 25. Constructing and sampling from an HDDMnnRLRegressor model.

Finally, it is important to note that we are continually adding new functionalities to the `HDDMnnRL` and `HDDMnnRLRegressor` classes. Given the state of active development for these classes, we suggest that the users refer to the HDDM documentation for any updates to the usage syntax or other changes.

More Resources

The original HDDM (Wiecki et al., 2013) article as well as the original HDDMrl article (Pedersen & Frank, 2020) are good resources on the basics of HDDM. The documentation provides examples for many complex use cases, including a long tutorial specifically designed to illustrate the `HDDMnn` classes and another tutorial specifically designed to showcase the `HDDMnnRL` classes. Through the HDDM user group, an active community of HDDM users, one can find support on many problems and use cases that may not come up in the official documentation or published work.

Concluding Thoughts

We hope this tutorial can help kick-start a more widespread application of SSMs in the analysis of experimental choice and RT data. We consider the initial implementation with focus on LANs (Fengler et al., 2021) as a starting point, which allows a significant generalization of the model space that can be considered by experimenters. The ultimate goal, however, is to lead toward community engagement, providing an easy interface for the addition of custom models as a start, which could greatly expand the space of models accessible to research groups across the world. We elaborate on a few possible directions for advancements in the next section.

LIMITATIONS AND FUTURE WORK

The presented extension to HDDM greatly expands the capabilities of a tried-and-tested Python toolbox, popular in the cognitive modeling sphere. However, using HDDM as the vehicle of choice, limitations endemic to the toolbox design remain and warrant a look ahead. First, HDDM is based on PyMC2 (Patil et al., 2010), a probabilistic modeling framework that has since been superseded by its successor PyMC3 (Salvatier, Wiecki, & Fonnesbeck, 2016; PyMC 4.0, a rebranded PyMC has just been released too). Because PyMC2 is not an evolving toolbox, HDDM is currently bound to fairly basic MCMC algorithms, specifically a coordinate-wise slice sampler (Neal, 2003). Although we have confirmed adequate posterior sampling and estimation using our LANs, estimation may be rendered more efficient if one were to leverage more recent MCMC algorithms such as Hamiltonian Monte Carlo (Hoffman & Gelman, 2014). Moreover, new libraries have emerged that act as independent functionality providers for other probabilistic programming frameworks, for example, the ArViz (Kumar, Carroll, Hartikainen, & Martin, 2019) python library that provides a wide array of capabilities from posterior visualizations to the computation of model comparison metrics such as the Widely Applicable Information Criterion (Watanabe, 2013). Custom scripts can be used currently to deploy ArViz within HDDM. We are moreover working on a successor to HDDM (we dub it HSSM), which will be built on top of one or more of these modern probabilistic programming libraries. Second, we realize that a major bottleneck in the wider adoption of LANs (and other likelihood approximators) lies in the supply of amortizers. Although our extension comes batteries included, we focused on supplying a few SSM variants of proven interest in the literature, as well as some that we used for our or laboratory-adjacent research. It is not HDDM but user-friendly training pipelines for amortizers that we believe to spur the quantum leap in activity in this space. Although we are working on the supply of such a pipeline for LANs (Fengler et al., 2021), our hope is that the community will provide many alternatives. Third, we caution against uninformed use of approximate likelihoods. Before basing results of empirical studies on inference performed with LANs or other approximate likelihoods (e.g., user supplied), it is essential to test for the quality of inference that may be expected. Inference can be unreliable in manifold ways (Talts, Betancourt, Simpson, Vehtari, & Gelman, 2018; Gelman & Rubin, 1992; Geweke, 1992). Parameter recovery studies and calibration tests, for example, simulation-based calibration (Talts et al., 2018), should form the backbone of trust in reported analysis on empirical (experimental) data sets. To help the application of a universal standard of rigor, we are working on a set of guidelines, such as a suggested battery of tests to pass, before given user-supplied likelihoods should be made available to the public. Other interesting work in this sphere is emerging (Hermans, Delaunoy, Rozet, Wehenkel, & Louppe, 2021; Lueckmann, Boelts, Greenberg, Goncalves, & Macke, 2021).

Acknowledgments

This work was funded by National Institute of Mental Health grants P50 MH 119467-01 and R01 MH084840-08A1 and additionally supported by the Brainstorm Program at the Robert J. and Nancy D. Carney Institute for Brain Science. We thank Lakshmi N. Govindarajan for useful tips concerning code quality. We also thank Thomas Wiecki for reviewing the additions to the HDDM package.

Reprint requests should be sent to Alexander Fengler, Department of Cognitive, Linguistic and Psychological Sciences, Brown University, 198 Lippitt St., Apt. 3, Providence, RI 02906-1618, or via e-mail: alexander_fengler@brown.edu.

Data Availability

Data for this article are available from https://github.com/AlexanderFengler/hddmnn_tutorial_paper.

Author Contributions

Krishn Bera: Conceptualization; Data curation; Methodology; Software; Visualization; Writing—Original draft;

Writing—Review & editing. Mads L. Pedersen: Software; Writing—Review & editing. Michael J. Frank: Conceptualization; Funding acquisition; Methodology; Supervision; Writing—Review & editing.

Diversity in Citation Practices

Retrospective analysis of the citations in every article published in this journal from 2010 to 2021 reveals a persistent pattern of gender imbalance: Although the proportions of authorship teams (categorized by estimated gender identification of first author/last author) publishing in the *Journal of Cognitive Neuroscience* (*JoCN*) during this period were M(an)/M = .407, W(oman)/M = .32, M/W = .115, and W/W = .159, the comparable proportions for the articles that these authorship teams cited were M/M = .549, W/M = .257, M/W = .109, and W/W = .085 (Postle and Fulvio, *JoCN*, 34:1, pp. 1–3). Consequently, *JoCN* encourages all authors to consider gender balance explicitly when selecting which articles to cite and gives them the opportunity to report their article's gender citation balance. The authors of this article report its proportions of citations by gender category to be as follows: M/M = .692, W/M = .135, M/W = .135, and W/W = .038.

REFERENCES

Ballard, I. C., & McClure, S. M. (2019). Joint modeling of reaction times and choice improves parameter identifiability in reinforcement learning models. *Journal of Neuroscience Methods*, *317*, 37–44. https://doi.org/10.1016/j.jneumeth.2019.01.006, PubMed: 30664916

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2010). Cython: The best of both worlds. *Computing in Science & Engineering*, *13*, 31–39. https://doi.org/10.1109/MCSE.2010.118

Betancourt, M. J., & Girolami, M. (2013). Hamiltonian Monte Carlo for hierarchical models. *arXiv:1312.0906*. https://doi.org/10.48550/arxiv.1312.0906

Boehm, U., Cox, S., Gantner, G., & Stevenson, R. (2021). Fast solutions for the first-passage distribution of diffusion models with space–time-dependent drift functions and time-dependent boundaries. *Journal of Mathematical Psychology*, *105*, 102613. https://doi.org/10.1016/j.jmp.2021.102613

Brooks, S. P., & Gelman, A. (1998). General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, *7*, 434–455. https://doi.org/10.1080/10618600.1998.10474787

Cisek, P., Puskas, G. A., & El-Murr, S. (2009). Decisions in changing conditions: The urgency-gating model. *Journal of Neuroscience*, *29*, 11560–11571. https://doi.org/10.1523/JNEUROSCI.1844-09.2009, PubMed: 19759303

Collins, A. G., & Shenhav, A. (2022). Advances in modeling learning and decision-making in neuroscience. *Neuropsychopharmacology*, *47*, 104–118. https://doi.org/10.1038/s41386-021-01126-y, PubMed: 34453117

Doi, T., Fan, Y., Gold, J. I., & Ding, L. (2020). The caudate nucleus contributes causally to decisions that balance reward and uncertain visual information. *eLife*, *9*, e56694. https://doi.org/10.7554/eLife.56694, PubMed: 32568068

Dowd, E. C., Frank, M. J., Collins, A., Gold, J. M., & Barch, D. M. (2016). Probabilistic reinforcement learning in patients with schizophrenia: Relationships to anhedonia and avolition. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, *1*, 460–473. https://doi.org/10.1016/j.bpsc.2016.05.005, PubMed: 27833939

Eckstein, M. K., Wilbrecht, L., & Collins, A. G. (2021). What do reinforcement learning models measure? Interpreting model parameters in cognition and neuroscience. *Current Opinion in Behavioral Sciences*, *41*, 128–137. https://doi.org/10.1016/j.cobeha.2021.06.004, PubMed: 34984213

Evans, N. J., Trueblood, J. S., & Holmes, W. R. (2020). A parameter recovery assessment of time-variant models of decision-making. *Behavior Research Methods*, *52*, 193–206. https://doi.org/10.3758/s13428-019-01218-0, PubMed: 30924107

Fengler, A., Govindarajan, L. N., Chen, T., & Frank, M. J. (2021). Likelihood approximation networks (LANs) for fast inference of simulation models in cognitive neuroscience. *eLife*, *10*, e65074. https://doi.org/10.7554/eLife.65074, PubMed: 33821788

Fontanesi, L. (2022). *Rlssm (Version 0.1.1)*. https://zenodo.org/record/4562217.

Fontanesi, L., Gluth, S., Spektor, M. S., & Rieskamp, J. (2019). A reinforcement learning diffusion decision model for value-based decisions. *Psychonomic Bulletin & Review*, *26*, 1099–1121. https://doi.org/10.3758/s13423-018-1554-2, PubMed: 30924057

Forstmann, B. U., Anwander, A., Schäfer, A., Neumann, J., Brown, S., Wagenmakers, E.-J., et al. (2010). Cortico-striatal connections predict control over speed and accuracy in perceptual decision making. *Proceedings of the National Academy of Sciences, U.S.A.*, *107*, 15916–15920. https://doi.org/10.1073/pnas.1004932107, PubMed: 20733082

Frank, M. J., Gagne, C., Nyhus, E., Masters, S., Wiecki, T. V., Cavanagh, J. F., et al. (2015). fMRI and EEG predictors of dynamic decision parameters during human reinforcement learning. *Journal of Neuroscience*, *35*, 485–494. https://doi.org/10.1523/JNEUROSCI.2036-14.2015, PubMed: 25589744

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, *7*, 457–472. https://doi.org/10.1214/ss/1177011136

Geweke, J. (1992). Evaluating the accuracy of sampling-based approaches to the calculations of posterior moments. *Bayesian Statistics*, *4*, 641–649. https://doi.org/10.21034/sr.148

Gold, J. I., & Shadlen, M. N. (2007). The neural basis of decision making. *Annual Review of Neuroscience*, *30*, 535–574. https://doi.org/10.1146/annurev.neuro.29.051605.113038, PubMed: 17600525

Greenberg, D., Nonnenmacher, M., & Macke, J. (2019). Automatic posterior transformation for likelihood-free inference. *Proceedings of the 36th International Conference on Machine Learning, PMLR*, *97*, 2404–2414.

Gutmann, M. U., Dutta, R., Kaski, S., & Corander, J. (2018). Likelihood-free inference via classification. *Statistics and Computing*, *28*, 411–425. https://doi.org/10.1007/s11222-017-9738-6, PubMed: 31997856

Hawkins, G. E., Forstmann, B. U., Wagenmakers, E.-J., Ratcliff, R., & Brown, S. D. (2015). Revisiting the evidence for collapsing boundaries and urgency signals in perceptual decision-making. *Journal of Neuroscience*, *35*, 2476–2484. https://doi.org/10.1523/JNEUROSCI.2410-14.2015, PubMed: 25673842

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

Heathcote, A., Matzke, D., & Heathcote, A. (2022). Winner takes all! What are race models, and why and how should psychologists use them? *Current Directions in Psychological Science*.

Hermans, J., Delaunoy, A., Rozet, F., Wehenkel, A., & Louppe, G. (2021). Averting a crisis in simulation-based inference. *arXiv:2110.06581*. https://doi.org/10.48550/arXiv.2110.06581

Hoffman, M. D., & Gelman, A. (2014). The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, *15*, 1593–1623. https://jmlr.org/papers/v15/hoffman14a.html

Holmes, W. R., & Trueblood, J. S. (2018). Bayesian analysis of the piecewise diffusion decision model. *Behavior Research Methods*, *50*, 730–743. https://doi.org/10.3758/s13428-017-0901-y, PubMed: 28597236

Krajbich, I., Lu, D., Camerer, C., & Rangel, A. (2012). The attentional drift-diffusion model extends to simple purchasing decisions. *Frontiers in Psychology*, *3*, 193. https://doi.org/10.3389/fpsyg.2012.00193, PubMed: 22707945

Krajbich, I., & Rangel, A. (2011). Multialternative drift-diffusion model predicts the relationship between visual fixations and choice in value-based decisions. *Proceedings of the National Academy of Sciences, U.S.A.*, *108*, 13852–13857. https://doi.org/10.1073/pnas.1101328108, PubMed: 21808009

Kumar, R., Carroll, C., Hartikainen, A., & Martin, O. (2019). Arviz a unified library for exploratory analysis of Bayesian models in python. *Journal of Open Source Software*, *4*, 1143. https://doi.org/10.21105/joss.01143

Lawlor, V. M., Webb, C. A., Wiecki, T. V., Frank, M. J., Trivedi, M., Pizzagalli, D. A., et al. (2020). Dissecting the impact of depression on decision-making. *Psychological Medicine*, *50*, 1613–1622. https://doi.org/10.1017/S0033291719001570, PubMed: 31280757

Lueckmann, J.-M., Bassetto, G., Karaletsos, T., & Macke, J. H. (2019). Likelihood-free inference with emulator networks. *Proceedings of The 1st Symposium on Advances in Approximate Bayesian Inference, PMLR*, *96*, 32–53.

Lueckmann, J.-M., Boelts, J., Greenberg, D., Goncalves, P., & Macke, J. (2021). Benchmarking simulation-based inference. Paper presented at the International Conference on Artificial Intelligence and Statistics (pp. 343–351). https://proceedings.mlr.press/v130/lueckmann21a.html

McDougle, S. D., & Collins, A. G. (2021). Modeling the influence of working memory, reinforcement, and action uncertainty on reaction time and choice during instrumental learning. *Psychonomic Bulletin & Review*, *28*, 20–39. https://doi.org/10.3758/s13423-020-01774-z, PubMed: 32710256

McKinney, W. (2010). Data structures for statistical computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 51–56). https://doi.org/10.25080/majora-92bf1922-00a

Neal, R. M. (1995). *Bayesian learning for neural networks* [Doctoral dissertation]. University of Toronto.

Neal, R. M. (2003). Slice sampling. *Annals of Statistics*, *31*, 705–741. https://doi.org/10.1214/aos/1056562461

Palestro, J. J., Sederberg, P. B., Osth, A. F., Van Zandt, T., & Turner, B. M. (2019). *Likelihood-free methods for cognitive science*. Berlin: Springer. https://doi.org/10.1007/978-3-319-72425-6

Papamakarios, G., & Murray, I. (2016). Fast ε-free inference of simulation models with Bayesian conditional density estimation. In *Advances in neural information processing systems* (pp. 1028–1036).

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2019). Normalizing flows for probabilistic modeling and inference. *arXiv:1912.02762*. https://doi.org/10.48550/arXiv.1912.02762

Papamakarios, G., Sterratt, D., & Murray, I. (2019). Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics* (pp. 837–848). https://proceedings.mlr.press/v89/papamakarios19a.html

Papaspiliopoulos, O., Roberts, G. O., & Sköld, M. (2007). A general framework for the parametrization of hierarchical models. *Statistical Science*, *22*, 59–73. https://doi.org/10.1214/088342307000000014

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32, pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Patil, A., Huard, D., & Fonnesbeck, C. J. (2010). PyMC: Bayesian stochastic modelling in Python. *Journal of Statistical Software*, *35*, 1–81. https://doi.org/10.18637/jss.v035.i04, PubMed: 21603108

Pedersen, M. L., & Frank, M. J. (2020). Simultaneous hierarchical Bayesian parameter estimation for reinforcement learning and drift diffusion models: A tutorial and links to neural data. *Computational Brain & Behavior*, *3*, 458–471. https://doi.org/10.1007/s42113-020-00084-w, PubMed: 35128308

Pedersen, M. L., Frank, M. J., & Biele, G. (2017). The drift diffusion model as the choice rule in reinforcement learning. *Psychonomic Bulletin & Review*, *24*, 1234–1251. https://doi.org/10.3758/s13423-016-1199-y, PubMed: 27966103

Rangel, A., Camerer, C., & Montague, P. R. (2008). A framework for studying the neurobiology of value-based decision making. *Nature Reviews Neuroscience*, *9*, 545–556. https://doi.org/10.1038/nrn2357, PubMed: 18545266

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

Ratcliff, R., & Frank, M. J. (2012). Reinforcement-based decision making in corticostriatal circuits: Mutual constraints by neurocomputational and diffusion models. *Neural Computation*, *24*, 1186–1229. https://doi.org/10.1162/neco_a_00270, PubMed: 22295983

Ratcliff, R., Smith, P. L., Brown, S. D., & McKoon, G. (2016). Diffusion decision model: Current issues and history. *Trends in Cognitive Sciences*, *20*, 260–281. https://doi.org/10.1016/j.tics.2016.01.007, PubMed: 26952739

Ratcliff, R., Thapar, A., & McKoon, G. (2006). Aging, practice, and perceptual tasks: A diffusion model analysis. *Psychology and Aging*, *21*, 353–371. https://doi.org/10.1037/0882-7974.21.2.353, PubMed: 16768580

Rescorla, R. A. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black & W. F. Prokasy (Eds.), *Classical conditioning II: Current research and theory* (pp. 64–99). New York: Appleton Century Crofts.

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, *2*, e55. https://doi.org/10.7717/peerj-cs.55

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, PubMed: 32749218

Silverman, B. W. (1986). *Density estimation for statistics and data analysis* (Vol. 26). CRC Press. https://doi.org/10.1201/9781315140919

Smith, P. L., Ratcliff, R., & Sewell, D. K. (2014). Modeling perceptual discrimination in dynamic noise: Time-changed diffusion and release from inhibition. *Journal of Mathematical Psychology*, *59*, 95–113. https://doi.org/10.1016/j.jmp.2013.05.007

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., & Van der Linde, A. (2014). The deviance information criterion: 12 years on. *Journal of the Royal Statistical Society, Series B: Statistical Methodology*, *76*, 485–493. https://doi.org/10.1111/rssb.12062

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Talts, S., Betancourt, M., Simpson, D., Vehtari, A., & Gelman, A. (2018). Validating Bayesian inference algorithms with simulation-based calibration. *arXiv:1804.06788*. https://doi.org/10.48550/arXiv.1804.06788

Tejero-Cantero, A., Boelts, J., Deistler, M., Lueckmann, J.-M., Durkan, C., Gonçalves, P. J., et al. (2020). Sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, *5*, 2505. https://doi.org/10.21105/joss.02505

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*, 911–936. https://doi.org/10.3758/s13423-020-01719-6, PubMed: 32424622

Tran, N.-H., Van Maanen, L., Heathcote, A., & Matzke, D. (2021). Systematic parameter reviews in cognitive modeling: Towards a robust and cumulative characterization of psychological processes in the diffusion decision model. *Frontiers in Psychology*, *11*, 608287. https://doi.org/10.3389/fpsyg.2020.608287, PubMed: 33584443

Trueblood, J. S., Heathcote, A., Evans, N. J., & Holmes, W. R. (2021). Urgency, leakage, and the relative nature of information processing in decision-making. *Psychological Review*, *128*, 160–186. https://doi.org/10.1037/rev0000255, PubMed: 32852976

Turner, B. M. (2019). Toward a common representational framework for adaptation. *Psychological Review*, *126*, 660–692. https://doi.org/10.1037/rev0000148, PubMed: 30973248

Turner, B. M., & Sederberg, P. B. (2014). A generalized, likelihood-free method for posterior estimation. *Psychonomic Bulletin & Review*, *21*, 227–250. https://doi.org/10.3758/s13423-013-0530-0, PubMed: 24258272

Turner, B. M., & Van Zandt, T. (2018). Approximating Bayesian inference through model simulation. *Trends in Cognitive Sciences*, *22*, 826–840. https://doi.org/10.1016/j.tics.2018.06.003, PubMed: 30093313

Usher, M., & McClelland, J. L. (2001). The time course of perceptual choice: The leaky, competing accumulator model. *Psychological Review*, *108*, 550–592. https://doi.org/10.1037/0033-295x.108.3.550, PubMed: 11488378

Van Zandt, T., Colonius, H., & Proctor, R. W. (2000). A comparison of two response time models applied to perceptual matching. *Psychonomic Bulletin & Review*, *7*, 208–256. https://doi.org/10.3758/BF03212980, PubMed: 10909132

Vandekerckhove, J., & Tuerlinckx, F. (2008). Diffusion model analysis with Matlab: A DMAT primer. *Behavior Research Methods*, *40*, 61–72. https://doi.org/10.3758/brm.40.1.61, PubMed: 18411528

Voss, A., Lerche, V., Mertens, U., & Voss, J. (2019). Sequential sampling models with variable boundaries and non-normal noise: A comparison of six models. *Psychonomic Bulletin & Review*, *26*, 813–832. https://doi.org/10.3758/s13423-018-1560-4, PubMed: 30652240

Watanabe, S. (2013). A widely applicable Bayesian information criterion. *Journal of Machine Learning Research*, *14*, 867–897

Wiecki, T. V., Sofer, I., & Frank, M. J. (2013). HDDM: Hierarchical Bayesian estimation of the drift-diffusion model in python. *Frontiers in Neuroinformatics*, *7*, 14. https://doi.org/10.3389/fninf.2013.00014, PubMed: 23935581

Wieschen, E. M., Voss, A., & Radev, S. (2020). Jumping to conclusion? A Lévy flight model of decision making. *Quantitative Methods for Psychology*, *16*, 120–132. https://doi.org/10.20982/tqmp.16.2.p120

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. *eLife*, *8*, e49547. https://doi.org/10.7554/eLife.49547, PubMed: 31769410

Yartsev, M. M., Hanks, T. D., Yoon, A. M., & Brody, C. D. (2018). Causal contribution and dynamical encoding in the striatum during evidence accumulation. *eLife*, *7*, e34929. https://doi.org/10.7554/elife.34929, PubMed: 30141773