**B&C: Methods and Statistics 2020/2021**

**Week #4**

**Tutorial – Cognitive Modeling**

**Note: Please complete the provided Tutorial Answer Sheet and upload it on Canvas before**

**Monday 28 Sep 2020 at 23:59.**

Go through this tutorial at your own pace. You can talk in your Zoom groups or use the chat function to help each other with setting up the software and getting the code to run. I will also be online to answer questions and help with any issues you may have with getting the software and code to work.

This tutorial **does not** require you to write any code. All the R code and commands are provided, you simply need to run them and inspect the output.

You will see a number of **Tutorial Questions** scattered throughout this tutorial. **Copy-paste your answers to these into the Tutorial Answer Sheet.**

**Introduction and software setup**

In this tutorial we will learn how to build and fit several LBA models to choice-RT data. We will be using the software package Dynamic Models of Choice (DMC; Heathcote et al., 2018), which provides a suite of functions for fitting a wide range of evidence accumulation models.

DMC is written in the R programming language, so make sure you have downloaded and installed R. R can be downloaded from: <https://mirror.lyrahosting.com/CRAN/>. Follow the installation instructions for your operating system.

The R download page should look something like this (click the correct version for your operating system):

A screenshot of a cell phone

Description automatically generated

We will also download and install R-Studio Desktop: <https://rstudio.com/products/rstudio/download/>. R-Studio simply provides a nicer ‘point-and-click’ graphical user interface that we can use to interact with R.

We will also download from Canvas the file *dmc.zip*, which contains the DMC software and several R scripts (e.g., *LBA.R, setup.R*) that we will use for this tutorial.

The contents of your unzipped *dmc* folder should look like this:

A picture containing screenshot

Description automatically generated

Open R-Studio and set your ‘working directory’ to the *dmc* folder using the *Session* menu:

A screenshot of a cell phone

Description automatically generated

Finally, we need to open and run the *setup.R* file in the *dmc* folder. This will install a number of R packages that we will use throughout the tutorial. To run the code, highlight all the code (CRTL+A) and click the ‘Run’ button on the script window:

A screenshot of a cell phone

Description automatically generated

Now we are ready to open the *LBA.R* tutorial script.

**Software preparation checklist:**

1. Install R
2. Install R-Studio
3. Download *dmc.zip* from Canvas and unzip
4. In R-Studio, make sure your working directory is set to the *dmc* folder
5. Open and run the *setup.R* file
6. Open the *LBA.R* tutorial script

**Side note on Bayesian parameter estimation**

We haven’t spoken much about how models are actually fit to data (and it is quite an involved topic in its own right!). However, we should say a little about how fitting is accomplished in DMC – in particular how the ‘best’ parameter values for a given model and data are found.

In DMC, the ‘best fitting’ parameter values are found using a Bayesian sampling algorithm (a full introduction to Bayesian methods and their advantages is beyond the scope of this course, however interested students can consult the references below to learn more). In short, the process involves taking repeated samples from parameter distributions to establish which values are most likely under a given model and data.

This method means that parameters are represented by entire *distributions* rather than single point estimates (as is common in non-Bayesian methods such as maximum likelihood estimation). Having access to entire parameter distributions is particularly useful for quantifying our confidence or uncertainty in our model, whereby the uncertainty in a parameter is directly related to the width of its *posterior distribution* (i.e., its distribution after sampling is complete). Posterior distributions are also useful for plotting parameters (e.g., as means with error bars), reporting confidence intervals, and for computing post-hoc contrasts (e.g., assessing whether a parameter changed in value between two experimental conditions).

Bayesian methods also mean that we must specify some *priors* (i.e., starting values) for each parameter in our model. These initial values simply give the sampling algorithm a place to start in the (very large!) space of all possible parameter values, and in practice the sampler will quickly move away from these to converge on the ‘true’ best parameter values. You will see some diagnostic plots and other tools to assess convergence in the tutorial code.

To learn more about Bayesian modeling start here:

Etz, A., Gronau, Q. F., Dablander, F., Edelsbrunner, P. A., & Baribault, B. (2018). How to become a Bayesian in eight easy steps: An annotated reading list. *Psychonomic Bulletin & Review*, *25*(1), 219-234.

**Background/description of task and data**

For this tutorial we will be working with a simulated dataset inspired by Forstmann et al.’s (2008) random dot motion task with a speed-accuracy trade-off manipulation.

On each trial, participants are presented with a dot motion stimulus and must decide whether the coherently moving dots are moving to the *left* or two the *right*. So we have 2 stimuli (left, right) and 2 possible responses (left, right). This would be appropriate to model using a 2-accumulator LBA model, with an accumulator for *left* responses and an accumulator for *right* responses.

|  |
| --- |
| **A drawing of a face  Description automatically generated** |
| Illustration of random dot motion stimuli. The unfilled circles move in random directions. The filled circles move coherently either to the left or to the right. Varying the proportion of random versus coherently moving dots changes how difficult it is to perceive the direction of coherent motion. |

We also included two experimental manipulations designed to selectively influence different LBA parameters:

1. Stimulus difficulty/discriminability (easy vs. hard)
2. Speed vs. accuracy instructions (speed vs. accuracy)

We used two levels of stimulus difficulty (easy, hard), which were created by varying the proportion of coherently moving dots (i.e., *easy* stimuli have mostly coherently moving dots and *hard* stimuli have mostly randomly moving dots).

Finally, before each trial, participants where cued with either *speed* or *accuracy* instructions. Speed instructions required participants to respond as quickly as possible (at the expense of accuracy). Accuracy instructions required participants to respond as accurately as possible (at the expense of speed).

In summary, we have a 2-response task performed in a 2 x 2 x 2 factorial experimental design (i.e., left/right, easy/hard, speed/accuracy). For this tutorial we assume we collected data from one participant who completed 640 trials (i.e., 80 trials per design cell).

**Tutorial Question 1:** Which LBA parameter do you expect to be affected by the stimulus discriminability manipulation? What would happen to this parameter if you increased stimulus discriminability (i.e., increased the proportion of coherently moving dots)? Why? **Copy-paste your answer into the tutorial answer sheet.**

**Tutorial Question 2:** Which LBA parameter do you expect to be affected by the speed-accuracy trade-off manipulation? What would happen to this parameter if you instructed participants to focus on responding quickly? Why? **Copy-paste your answer into the tutorial answer sheet.**

**Fitting and interpreting the LBA model in R**

In R-Studio, open the *LBA.R* script and make sure your working directory is set to the top-level *dmc* folder (as described above). The script should appear in a window, like so:

A screenshot of a cell phone

Description automatically generated

In this tutorial we will fit several variants of the LBA model to our motion detection dataset to see which parameters account for the observed effects.

The steps below will walk you through each part of the *LBA.R* script and also explain the rationale behind each step.

**Load and inspect the data**

First, we will clear the R workspace and load a number of packages and model functions:

A picture containing table, holding

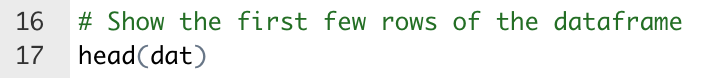
Description automatically generated

Remember: To run a piece of code in R, highlight the line you want to run and click the ‘Run’ icon (or press CTRL + ENTER).

Now we will load our data into R and check that everything looks the way it should:



Let’s run the head() command. This will show us the first few rows of our data:



The output of head() should appear in the console window and look like this:

A screenshot of a cell phone

Description automatically generated

So we have a column called ‘RT’ that has the response time on each trial in seconds. The other four columns are the factors of our experimental design, which each have two levels:

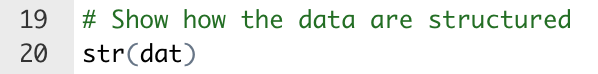
‘S’ = stimulus type (left, right)

‘D’ = stimulus difficulty (easy, hard)

‘I’ = speed/accuracy instruction (acc, spd)

‘R’ = response (L, R)

We can display this information using the str() function:



A screenshot of a cell phone

Description automatically generated

Now we will add a column (called ‘C’ for ‘correct’) that codes accuracy. This code assigns a ‘1’ for responses that match the correct stimulus (e.g., responding ‘L’ to a ‘left’ stimulus) and a ‘0’ for responses that don’t match the correct stimulus (e.g., responding ‘L’ to a ‘right’ stimulus):

A screenshot of a cell phone

Description automatically generated

We can use the head() function again to see that our new accuracy column has been created as expected:

A screenshot of a cell phone

Description automatically generated

We can now summarize the data by looking at the mean accuracy in each design cell. We pull out accuracy grouped by the speed/accuracy instruction and stimulus difficulty factors (‘I’ and ‘D’):

A screenshot of a cell phone

Description automatically generated

The output gives us a table with the mean accuracy in each cell. We can see that perceptually easy stimuli tend to elicit higher accuracy (by approximately 20%) than perceptually difficult stimuli:

A picture containing table

Description automatically generated

**Tutorial Question 3:** What is the mean accuracy for perceptually easy stimuli under speed instructions? **Copy-paste your answer into the tutorial answer sheet.**

Now we can do the same for RT. Here we compute the mean and standard deviation of RT for each cell, as well as the number of observations in each cell. Note that we also group by accuracy so that we can get separate statistics for correct and error responses:

A screenshot of a cell phone

Description automatically generated

The results are shown in the following table. We can see that easy stimuli appear to have faster RTs than hard stimuli, speed instructions appear to have faster RTs than accuracy instructions, and errors are slightly faster than correct responses:

A close up of text on a white background

Description automatically generated

**Tutorial Question 4:** What is the mean RT for correct responses to perceptually easy stimuli under accuracy instructions? **Copy-paste your answer into the tutorial answer sheet.**

As a final inspection of the behavioral data, we can look at RT distributions by plotting histograms:

A screenshot of a cell phone

Description automatically generated

After running this code, you should see the histograms displayed in the plot window. You can use the ‘export’ button on the plot window to save the plot to your computer. You can select whatever image format you prefer (e.g., .png, .pdf).

Note: You may get an error message that says “warning - figure margins too large”. This happens when your plotting window is too small. Drag the edge of the window to make it larger and re-run the plotting code and it should work.

We will now build our first LBA variant that we will then fit to our data.

**Model 1: LBA with drift rate effect**

We will first build a model that allows drift rate to vary by stimulus discriminability (easy vs. hard). Thinking in terms of evidence accumulation theory, we would expect perceptually difficult stimuli (low motion coherence) to produce lower drift rates than perceptually easy stimuli (high motion coherence). The less discriminable a stimulus is, the weaker the strength of evidence that is sampled at each time point by the perceptual system. In the model this is reflected in the accumulation or drift rate parameter.

*Step 1 - Build model and set priors*

Here we build the model in which drift varies by stimulus difficulty. We use the model.dmc() function to make a model object – the model object binds together information about the structure of our model so that we can easily pass all relevant information to the sampler in a later step. The model object contains a list of parameters and experimental factors in the model, as well as information about how responses map to stimuli and which parameters we wish to hold constant.

**A close up of a person

Description automatically generated**

Running this code creates the model object and prints the following output. It shows the parameters and constants in our model:

*A screenshot of a cell phone

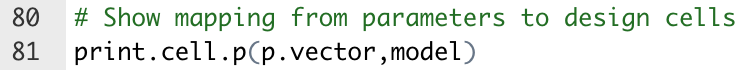
Description automatically generated*

Now we set some prior parameter values. These serve as a starting point for the sampler; during sampling these values are updated conditioned on the data to find the true ‘posterior’ parameter values.

*A screenshot of a cell phone

Description automatically generated*

This shows how each parameter maps to each design cell, which serves as a useful check whenever we are building a new model:



A screenshot of a cell phone

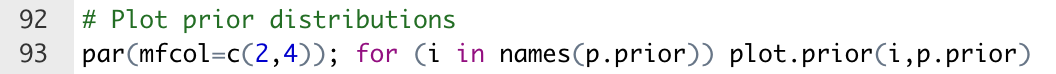
Description automatically generated

Now we create a priors object. This object bundles our mean parameter vector together with other information that describes how each parameter is *distributed*. We have eight parameters, so ‘dists’ specifies a vector of eight truncated normal distributions (“tnorm”). ‘p1’ is our mean parameter vector that we created earlier. ‘p2’ is a vector containing the variances of our prior distributions; here we use a broad value of 1 for each parameter. Finally, ‘lower’ and ‘upper’ specify the lower and upper bounds on the truncated normal distributions.

A screenshot of a cell phone

Description automatically generated

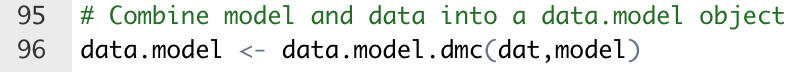
We can plot these distributions to ensure our priors cover a sufficiently broad range of parameter values:



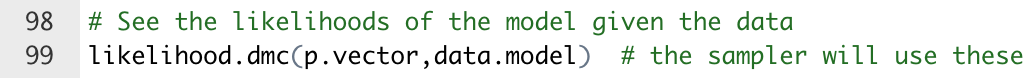
A close up of text on a white background

Description automatically generated

Now we bind the data and model together into a data.model object. This is a convenient way to keep everything related to a given model all together (and is passed to other functions in DMC):



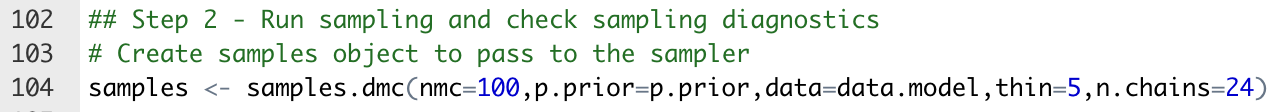
We can also print the likelihoods for our priors given the data and model. The sampler will use these to find its way through the parameter space and find the most likely values:



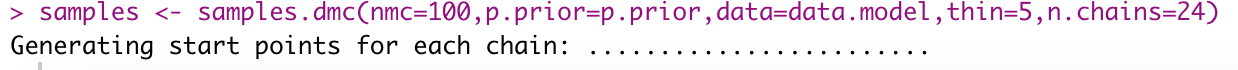
Now we are ready to run the sampling.

*Step 2 - Run sampling and check sampling diagnostics*

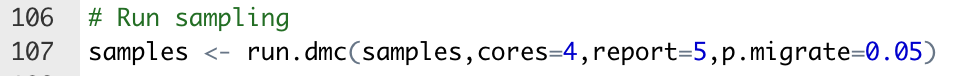
First, we create a samples object to pass to the sampler. This contains the number of samples we want (‘nmc’), our priors and data.model objects, and the number of sampling chains (a good rule of thumb is 3 times the number of parameters, so here we use 3 x 8 = 24). We also ‘thin’ the chains by 5, which means we only keep every 5th sample. This is done because the values obtained on each sampling iteration are highly correlated with the values on previous steps (this is known as *autocorrelation*). Thinning reduces this autocorrelation.



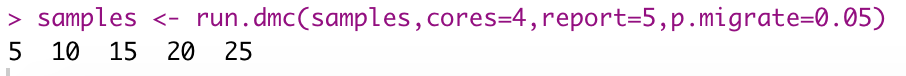
This generates start points that the sampler will use to get started:

****

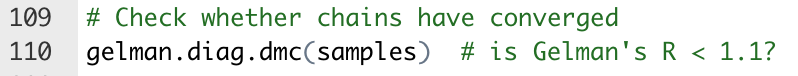
Now we can run the sampling using the run.dmc() function:

****

Sampling is computationally expensive, so we can make use of multiple processing cores. The sampler will report every fifth sampling iteration to reassure us that it is still running (it may take a minute or two). ‘p.migrate’ is an option for the sampler that helps avoid the sampler ‘getting stuck’ in local minima in the parameter space:

****

We now check whether the sampling chains have converged. Gelman’s R (aka ‘potential scale reduction factor’) measures the ratio of between- to within-chain variance. Values less than 1.1 are considered to have converged:

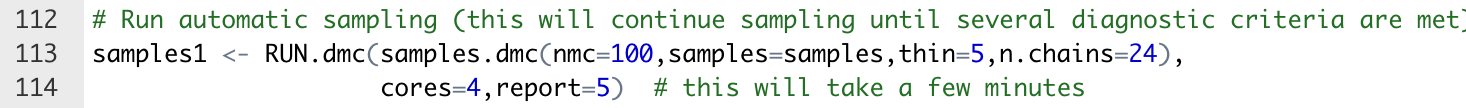
****

Clearly chains have not converged yet:

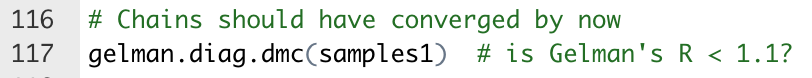
A close up of text on a white background

Description automatically generated

So we need to continue sampling for longer. We can automatically run sampling until convergence using the RUN.dmc() function. This will take a few minutes to run!



We check convergence again:

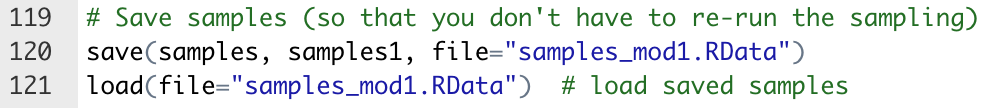


It looks good:

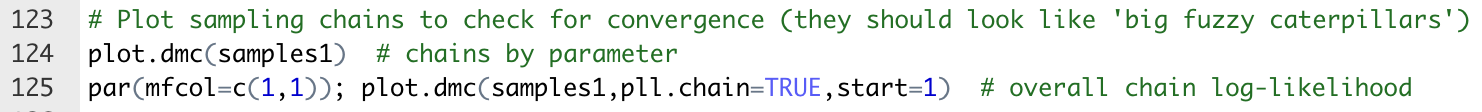
A screenshot of a cell phone

Description automatically generated

Since sampling takes so long, we will save the samples to the *dmc* folder, which will save us from having to re-run lengthy sampling steps:



We can also assess the sampling visually by plotting the sampling chains:



Well-mixed and converged chains should look like ‘big fuzzy caterpillars’:

A screenshot of a cell phone

Description automatically generated

Sampling looks reasonable, so we will next look at how well the model fits the empirical data.

*Step 3 - Plot fits*

Before we can plot how well the model fits to the data (i.e., the model’s predictions vs. the observed data), we first need to simulate the model’s predictions. The post.predict.dmc() function takes a completed samples object and repeatedly simulates the model’s predictions. Here we save off 100 simulated predictions (these are known as *posterior predictions*):

A picture containing photo, knife

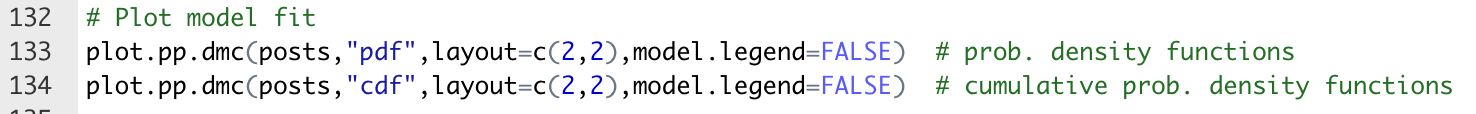
Description automatically generated

It is simulating (each dot represents 10 iterations):

A picture containing clock

Description automatically generated

Now we can plot the predictions against the data. We can plot fits in terms of probability density functions (PDF) or cumulative density functions (CDF):



Here are the fits to the CDFs:

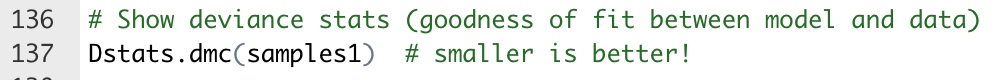
A close up of a map

Description automatically generated

For each cell of the design, these plots depict the cumulative probability of making each response by a given time (RT). As RT increases, the probability of responding increases up to an asymptote reflecting the overall response probability. The large distributions that rise steeply are the correct responses for that cell. The smaller flatter distributions are the error responses for that cell. We can see here that the fits are not very good (i.e., the black dots should lie inside the unfilled circles), suggesting that this model is too simple and is failing to capture important structure in the data. Models 2 and 3 below will explore letting different parameters to vary in order to find a better fit.

**Tutorial Question 5:** Save your plots showing the fit to each response’s cumulative density function (by either exporting as an image or taking a screenshot). Do the same for models 2 and 3 below so that you can compare their fits visually. **Copy-paste your answer into the tutorial answer sheet.**

We can also quantify the ‘goodness of fit’ using the *deviance*. The deviance provides a measure of how much the model’s predictions deviate from the empirical data and adds a penalty the more complex the model is (i.e., the more parameters you let vary).



The Dstats.dmc() function returns some statistics about deviance, including the number of parameters and variation in deviance:

A screenshot of a cell phone

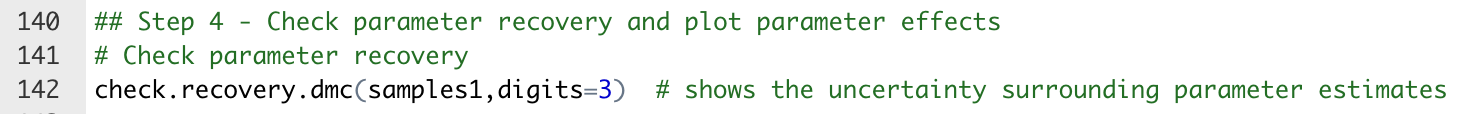
Description automatically generated

Generally, models with smaller deviance are to be preferred over models with higher deviance.

Now we will take a look at the parameter effects (i.e., which processes the model uses to explain the behavioral data).

*Step 4 – Check parameter recovery and plot parameter effects*

We are nearly ready to look at our parameter effects (i.e., how the model explains the data in terms of model parameters). First we will check parameter recovery using check.recovery.dmc():

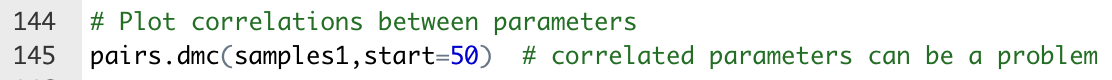


This shows the uncertainty surrounding the parameter estimates:

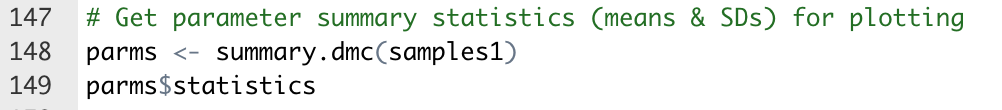
A screenshot of text

Description automatically generated

We can also plot the correlations between parameters. Correlations between parameters generally undesirable but are common and to some extent expected in evidence accumulation modelling. Correlated parameters can indicate that two parameters are trading-off in terms of explaining a certain effect.



Now we use summary.dmc() to get a statistical summary of the parameters useful for plotting:

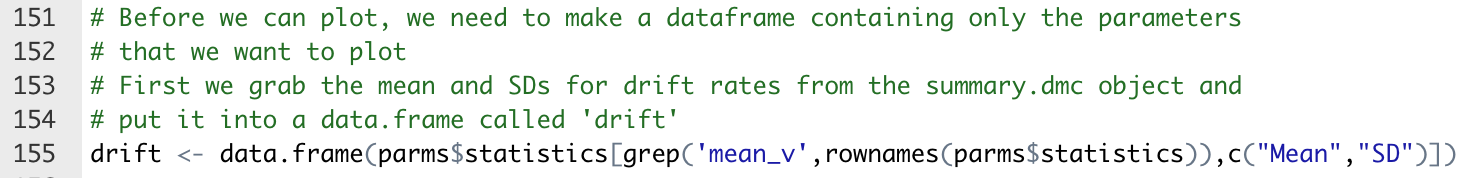


We are mainly interested in the means and standard deviations:

A screenshot of a cell phone

Description automatically generated

Before we can plot, we need to make a data frame containing only the parameters that we want to plot. The following code extracts the mean and SD columns for the drift rates only, and stores them in a data frame called ‘drift’:



Now we add a column factor called ‘D’ with levels ‘Easy’ and ‘Hard’. This represents our stimulus discriminability effect. We also create a factor called ‘C’ with levels ‘Correct’ and ‘Error’. This will let us plot correct and error drift rates separately:

A screenshot of a cell phone

Description automatically generated

So our drift rate data frame now looks like this:

A screenshot of a cell phone

Description automatically generated

Now we can build a plot using some functions from the ‘ggplot’ package. The first line in this block of code specifies which variable/factor are represented on the x-axis (stimulus difficulty), y-axis (mean parameter value), and symbol shape (correct vs. error responses). The next lines specify information about the plotting symbols, error bars, and connecting lines that make the plot look visually appealing. Finally, the y-axis limits and axis labels are specified, and the ‘classic’ theme applied. Run this block of code and see what you get:

A screenshot of a cell phone

Description automatically generated

The plot should look something like this (note: you may need to adjust the y-axis limits with ‘ylim()’ to make sure all plotting symbols appear in frame):

A picture containing object, antenna, filled, sitting

Description automatically generated

Here we can see that drift rates towards correct responses are lower when motion coherence/discriminability is low (‘hard’ difficulty) compared to when motion coherence/discriminability is high (‘easy’ difficulty).

**Tutorial Question 6:** Save your parameter plots for this model (by either exporting as an image or taking a screenshot). Do the same for models 2 and 3 below so that you can compare them visually. **Copy-paste your answer into the tutorial answer sheet.**

The remaining code applies the same four steps to fit two different versions of the LBA to the same data. ‘Model 2’ removes the drift rate effect and instead lets threshold vary by speed/accuracy instructions. ‘Model 3’ fits a model that contains both drift rate and threshold effects. Run this code and save your fit plots and parameter plots for each model. Think about which model provided the best fit to the data.

**Model 2: LBA with threshold effect**

Run the ‘Model 2’ code and save the fit and parameter plots to your Tutorial Answer Sheet.

**Model 3: LBA with drift rate and threshold effects**

Run the ‘Model 3’ code and save the fit and parameter plots to your Tutorial Answer Sheet.

If you have reached the end and would like to explore DMC further, you can look through the additional DMC tutorials found in the *tutorials* folder. These lessons cover additional DMC functions that are beyond the scope of this short introduction. Model selection, parameter recovery, advanced model diagnostics, reparametrizing models and creating new models, fitting group and hierarchical data, and fitting models to more complex/unbalanced experimental designs are all covered in the lessons. Working your way through these tutorials will give you a good practical grounding in using DMC to fit evidence accumulation models. Even if you don’t end up using such models in your own research, the hands-on experience will be valuable for when you encounter such modeling in the scientific literature.