Cognitivemodels: An R Package for Formal Cognitive Modeling

Jana B. Jarecki (jana.jarecki@unibas.ch)
Center for Economic Psychology, University of Basel
Missionsstrasse 64A, 4055 Basel, Switzerland

Florian I. Seitz (florian.seitz@unibas.ch)
Center for Economic Psychology, University of Basel
Missionsstrasse 64A, 4055 Basel, Switzerland

Abstract

We introduce cognitivemodels—a free software package for formal cognitive modeling in the statistical programming environment R. The package offers novice modelers a collection of models and offers experienced modelers a back-end for model development. This paper introduces the syntax of the package by example. The models in the software package include, for instance, the generalized context model for categorization (Nosofsky, 1986), cumulative prospect theory for risky choice (Tversky & Kahneman, 1992), and a Bayesian probability learning model. The package allows modelers to estimate model parameters and to constrain parameters by box constraints and equality constraints; it also allows to select choice rules such as soft maximum, epsilon greedy, or Luce's rule. It further offers modelers a selection of goodness of fit measures such as a binomial or normal log likelihood and mean-squared error, and a selection of 22 numeric optimization routines for parameter estimation. We believe this software package may facilitate the usage and testing of formal cognitive theories and may increase the robustness of cognitive modeling.

Keywords: cognitive modeling; model building; model testing; tutorial; R-statistics; software; robustness of code

Formal models of cognition enjoy an increasing popularity in cognitive science, for instance, to describe categorization (e.g., Nosofsky, 1986), judgments (e.g., Juslin, Olsson, & Olsson, 2003), and risky choice (e.g., Tversky & Kahneman, 1992). Over 100 such models have been developed in the past decades (Jarecki, Tan, & Jenny, 2020). Formalizing psychological theories can facilitate theory development and scientific progress. Recent recommendations for improving psychological science have not only emphasized replicable empirical effects, but also called for an increase in formal explanations of cognitive capacities (see e.g. Guest & Martin, 2020; Navarro, 2019; Van Rooij & Baggio, 2020). However, such cognitive models can only be fruitful if they are implemented in a robust manner (Lee, Chriss, & Vandekerckhove, 2019). One aspect of robustness is code reproducibility, which refers to the ability of a third party to reobtain a result by executing the original code (Benureau & Rougier, 2018; Wilson et al., 2017). Such robustness can be achieved by implementing cognitive models in a software package, such as the one presented here. The code in this software package is robust, because the package includes automatic tests of the modeling functions for invalid inputs and consistency (so-called unit tests, see the section Advantages of Cognitivemodels below). We believe that this package may make modeling more broadly accessible and may support the efforts towards increased formalization of psychological theories.

The **cognitivemodels** package is a library for the statistical programming environment R (R Core Team, 2019). It offers tools to estimate free model parameters, impose parameter constraints, make model predictions, and calculate the goodness of fit of models to data. The functions in the package have a consistent syntax across models which this paper introduces by example. Table 1 lists the models that the package currently provides to model discrete responses, that is choices. The package also offers models for continuous responses such as judgments which are not listed in the table. Table 1 shows that all models have a similar set of arguments (see the column "Arguments to function call"). These arguments will be detailed below in the section Setting up a Generalized Context Model.

Because the **cognitivemodels** package provides a collection of models from multiple domains in one library (see Table 1) it differs from existing modeling packages in R which contain specific types of models. Examples of such specific packages are pt for cumulative prospect theory, Speekenbrink's mcplr for multicue probability learning, or MPTinR for multinomial processing trees (Singmann & Kellen, 2013). **Cognitivemodels** provides a toolbox for model application

Table 1: Models in cognitivemodels

		Arguments to function call			
Model	Function call	formula	data	fix	choicerule
Generalized context model (Nosofsky, 1986)	gcm()	y ~ x1+...+xn	data	list(x1="xn", xn=.5)	string
Exemplar-based judgment (Juslin, Olsson & Olsson, 2003)	ebm_j()	y ~ x1+...+xn	data	list(x1="fn", xn=.5)	string
Cumulative prospect theory (Tversky & Kahneman, 1992)	cpt_d()	y ~ x1+px+x2 | y1+py+y2	data	list(alpha="beta", beta=.8)	string
Shortfall (Andraszewicz, 2014)	shortfall_d()	y ~ x1+px+x2 | y1+py+y2	data	list(delta="beta", beta=.5)	string
Risk-sensitivity model (Houston & McNamara, 1988)	hm1988()	y ~ x1+px+x2 | y1+py+y2	data	-	string
Bayesian learning model (Griffiths & Yuille, 2008)	bayes_beta_d()	y ~ x1+x2	data	list(priors = c(0.1, 2))	string
Power utility (Wakker, 2003)	utility_pow_d()	y ~ x1 | x2	data	list(rn="rp", rp=.5)	string
Soft-max choice rule (Sutton & Barto, 1998)	softmax()	y ~ x1 | x2	data	list(tau=.1)	string
Epsilon-greedy choice rule	epsilon()	y ~ x1 | x2	data	list(eps=.2)	string
Baseline	baseline_mean_d()	y ~ .	data	-	-
Baseline	baseline_const_d()	y ~ .	data	-	-

Note. Baseline models are stimulus-agnostic models that are often included in cognitive model comparisons. The models that have function call ending in _d are models for discrete response data such as choices. The package offers versions of these models for continuous response data such as judgments (not listed), which have the same function call except that the call ends in _c instead of _d.

and model development (the toolbox for model development targets experienced modelers, it will not be outlined in this paper).

The **cognitivemodels** package v0.0.9 implements computational models of cognition, it does not implement cognitive architectures such as ACT-R. It estimates free model parameters with numeric optimization or constrained numeric optimization such as maximum likelihood. Bayesian parameter estimation (e.g., Scheibehenne & Pachur, 2014) may be added in the future.

Getting Started

Throughout this paper we use R v3.6.3 (2020-02-29), the Rcpp package (v1.0.4.6), the latest matlib package (v0.9.4). The package **cognitivemodels** 0.0.9 is installed by running:

```
# install.packages(devtools)
library(devtools)
install_github(
  "janajarecki/cognitivemodels")
```

The package is loaded by running:

```
library(cognitivemodels)
```

The Generalized Context Model in the Package cognitivemodels

We will introduce the syntax of **cognitivemodels** by modeling categorization data with the generalized context model (GCM, Nosofsky, 1986, 2011). The generalized context model is a formal model of classification which assumes that people infer the category membership of a new stimulus based on how similar the stimulus is to previously-experienced category members. The stimulus is predicted to belong most probably to the category to whose members it is most similar. Formally, the model computes the psychological similarity between two stimuli i and and j based on the distance between the features of the stimuli. The similarity is given by: $s_{ij} = \exp\left(-\lambda \cdot \left[\sum_f w_f (x_{fi} - x_{fj})^r\right]^{q/r}\right)$, where x_{fi} and x_{fj} are the values of feature f of stimuli i and j, respectively. The similarity function has four free parameters highlighted in red: w_f is interpreted as the relative attention to feature f and constrained by $\sum_f w_f = 1$ and $0 \leq w_f \leq 1$, λ governs the sensitivity towards small differences between stimuli, q governs the relation between distance and psychological similarity, and r is the norm of the distance metric with $r \geq 1$; $r = 1$ produces a city-block metric and $r = 2$ produces the Euclidean metric. The model finally computes the evidence that stimulus i belongs to a category "1" as the sum of the similarities to previously encountered members of category "1" relative to the similarity to all previously encountered stimuli: $\Pr(C = 1, i) = \frac{b_1 \sum_{n=1}^{N_1} s_{in,C=1}}{\sum_C b_c \sum_{n=1}^{N_c} s_{in,C=c}}$, where $s_{in,C=1}$ is the similarity between stimulus i and the n^{th} member of category "1". The last free parameter b_1 is interpreted as a bias towards category "1", with $\sum_C b_c = 1$ and $0 \leq b_c \leq 1$.

Setting up a Generalized Context Model

We fit the model to data from a supervised categorization experiment in which participants learned to categorize lines into two categories by receiving feedback about the true category (Nosofsky, 1989). The lines were characterized by two features namely their size and their tilting angle. Because the paper reports aggregated data, we reconstructed the raw data which is available by `data(nosofsky1989long)`. We one condition from this data called "size". The syntax below loads the data and sets up the model, it is explained below the code.

```
# Use the 'size' condition in the data
data(nosofsky1989long)
DT <- nosofsky1989long
DT <- DT[DT$condition=="size", ]
D  <- DT[!is.na(DT$true_cat), ]

# Fit the model to the data D
model <- gcm(
  formula = response ~ angle + size,
  class = ~ true_cat,
  data = D,
  choicerule = "none")
```

The function `gcm()` fits the generalized context model and needs four arguments (see also the help file: `?gcm`). The arguments `formula` and `class` indicate the columns in the data to be modeled (in our data: "response", "angle", "size", and "true_cat"). The left side of the argument `formula` specifies the column that contains participants' trial-by-trial categorizations, in our example this column is called "response". The right side of `formula` specifies the column names of the stimulus features—here "angle" and "size"—separated by a plus sign.[1] The argument `class` specifies the column name in the data that holds the category feedback, in our example this column is called "true_cat". The `gcm()` model automatically names each attention weight parameters (w_f) after the column name of the corresponding stimulus feature. In our model the attention weight parameters are therefore called "angle" and "size", referring to the attention allocated to the angle and size feature, respectively. If the feature columns in the data were called "x1" and "x2" the corresponding formula would be `response ~ x1 + x2` and the attention weight parameters would be called "x1" and "x2". The argument `data` specifies the data which must be a data frame with the variables that are modeled in the columns and with one choice trial in each row. The argument `choicerule` specifies which choice rule or action selection rule, if any, the model uses to map continuous model predictions to discrete responses. The currently available choice rules are "argmax", "epsilon", "luce", and "softmax" (see `cm_choicerules()` for the allowed values). We set `choicerule = "none"` to not use a choice rule. The fitted generalized context model can be viewed by calling the object in R that holds the model, in our example this is `model`.

[1] While in categorization tasks the input in a given trial is generally one single stimulus, different tasks exist where multiple stimuli are presented simultaneously (e.g., when deciding between two monetary gambles called gamble `x`, consisting of outcomes `x1` with probability `px` and outcome `x2` else, and gamble `y`, consisting of outcomes `y1` with probability `py` and outcome `y2` else). In this case, the stimuli are separated from each other by a pipe `|` (e.g., the formula for predicting a participant's gamble choice `r` between the aforementioned gambles `x` and `y` is `~, x1 + px + x2 | y1 + py + y2`, see also Table 1).

Estimation of Model Parameters

If a model has free parameters, the **cognitivemodels** package estimates any free parameters of the model by default. The parameter estimation uses a numeric optimization method that searches the parameter space to optimize the goodness of fit between the predictions of the model and the observations in the data given possible parameter constraints. Our example code above estimates all the parameters of the generalized context model using maximum likelihood with a binomial probability density function. The resulting estimates for the free parameters can be viewed by `coef(model)`.

Different models in the **cognitivemodels** package (Table 1) have different parameter spaces, that is the names and ranges of the free parameters are model-specific. The parameters of any model are documented in the corresponding help file in the section Model Parameters (e.g., `?gcm` for the generalized context model). The lower and upper limits of the parameters in the different models are set internally and are based on parameter ranges and estimates in the literature; and in our example they are based on Nosofsky (1989). Modelers can change the parameter bounds as outlined below in the section Advanced Options. The parameter space of a model in **cognitivemodels** can

be printed using the method `parspace()`. For example, `parspace("gcm")` prints the parameter space of the generalized context model. Given a model has been stored as `model`, `parspace(model)` prints the parameter space of this very model. Furthermore, the method `constraints(model)` shows the parameter constraints of the stored model.

```
parspace(model)
constraints(model)
```

The parameter space of the generalized context model is as follows: each attention weight parameter ranges from 0.001 to 1, λ ranges from 0.001 to 10, r, and q each range from 1 to 2 and the bias parameter b_0 and b_1 each range from 0 to 1. The constraints show that both the attention weights and the bias parameters need to sum up to 1.

Parameter constraints. The following examples show how to fix model parameters, rather than estimating them, and how to implement parameter constraints. To fix or constrain parameters an argument called `fix` is needed when setting up a model. The value of the argument `fix` must be a named list containing the names of the parameters to fix and their respective values. The parameters that are not listed in `fix` will be estimated. For instance, to set the parameters r and q equal to 1 and estimate the remaining parameters we add the argument `fix = list(r = 1, q = 1)` to the call to `gcm()` as shown below. If the model is stored as `model`, `coef(model)` prints the free parameter estimates and `summary(model)` prints all parameter estimates.

```
model <- gcm(
  formula = response ~ angle + size,
  class = ~true_cat,
  data = D,
  fix = list(r = 1, q = 1),
  choicerule = "none")
```

As a further illustration of this logic consider a generalized context model that divides attention equally between the features "angle" and "size". This requires setting the attention weight parameters to 0.50, and is implemented by adding `fix = list(angle = 0.50, size = 0.50)` to the call to `gcm()`. To force the model to attend 99% to the feature "angle", the syntax is `fix = list(angle = 0.99, size = 0.01)`. Note, that in the generalized context model, the names of the attention weight parameters match the right side of the argument `formula`. If the argument `fix` fixes all model parameters no parameters are estimated, such as in `fix = list(r = 2, q = 2, angle = 0.5, size = 0.5, lambda = 1.60, b0 = 0.5, b1 = 0.5)`.

Cognitivemodels also allows the specification of equality constraints. To constrain, for instance, the value of the parameter r to be equal to the value of the parameter q we use `fix = list(r = "q")`. Then the parameter q is estimated and r is set equal to q. This equality constraint is implemented in the code below:

```
model <- gcm(
  formula = response ~ angle + size,
  class = ~true_cat,
  data = D,
  fix = list(r = "q"),
  choicerule = "none")
```

Equality constraints and fixed parameters can also be combined. For instance, the argument `fix = list(angle = 0.5, r = "q")` sets the attention weight for the feature "angle" to 0.50 and constrains $r = q$.

Models without parameter estimation. The package offers two possibilities to use cognitive models that contain free parameters without the estimation of the free parameters. The first method consists in fixing all model parameters to a numeric value using the `fix` argument, as outlined above. This is useful for simulating model behavior in an experimental design from a model with parameter values of interest. In this case the argument `formula` needs only a left-hand side. The second method to estimate no parameters consists in an argument `options = list(fit = FALSE)`. This is useful for testing toy models. In this case, a model is constructed with model-specific default parameter values. The default parameter values are listed in a column called "start" of the parameter space of a model (e.g., see `parspace("gcm")`). Because for the general context model, there are no universal default parameter values, the parameter values in this case correspond to the mean of the parameter ranges. The code below fixes all parameter values of the generalized context model to the estimated parameter values from Nosofsky (1989) (Table 5, row 1), and estimates no parameters.

```
model <- gcm(formula = response ~
    angle + size, class = ~true_cat,
    data = D, fix = list(angle = 0.1,
        size = 0.9, lambda = 1.6,
        r = 2, q = 2, b0 = 0.5,
        b1 = 0.5), choicerule = "none")
```

Generating Predictions

Given a cognitive model stored as `model`, the method `predict(model)` returns predictions from the model given its parameters. It makes predictions for the data used to set up the model. In our example `predict(model)` makes predictions for the data `D` that we used to fit the model. An optional argument `newdata` can be supplied to `predict()` to make predictions for new stimuli using the parameters of the model without newly estimating parameters. The new data needs to have the same format and column names as the data that was used to set up the model. Using the model from the last code block with parameters fixed to the parameter estimates in Nosofsky (1989), the below code predicts the categorization for all 16 stimuli in the "size" condition using the `newdata` argument.

```
newD <- DT[!duplicated(DT$stim_id), ]
newD <- newD[order(newD$stim_id), ]
predict(model, newdata = newD)
```

The predictions match the predictions in Nosofsky (1989) (Figure 5, "size" condition).

Goodness of Fit and Model Comparisons

The **cognitivemodels** package offers the following goodness of model fit measures for each model: log likelihood, the Bayesian information criterion (BIC, Schwarz, 1978), Akaike's information criterion (AIC, Kass & Raftery, 1995; Wagenmakers & Farrell, 2004) including the finite-sample corrected AICc (see Wagenmakers & Farrell, 2004), and the mean-squared error (MSE). The following code returns the respective goodness of fit measures.

```
logLik(model)
BIC(model)
AIC(model)
MSE(model)
```

To compare models, the `anova()` method can be used to render ANOVA-style tables. If one model is supplied as argument to `anova()`, the function returns an error summary. If multiple models are supplied to `anova()`, the function returns a model comparison table. The model comparison table includes the relative evidence strength measured by Akaike weights (Wagenmakers & Farrell, 2004) as well as a χ^2-test of the log likelihoods of the two models given these belong to the same class (e.g., two generalized context models will be compared by χ^2, but not a Bayesian model and a generalized context model). The example code below compares a generalized context model 1 that has the parameter constraints $r = 1, q = 1$ to a model 2 that has the parameter constraints $r = 2, q = 2$.

```
model1 <- gcm(
  formula = response ~ angle + size,
  class = ~true_cat,
  data = D,
  fix = list(r = 1, q = 1),
  choicerule = "none")

model2 <- gcm(
  formula = response ~ angle + size,
  class = ~true_cat,
  data = D,
  fix = list(r = 2, q = 2),
  choicerule = "none")

anova(model1, model2)
```

Advanced Options

The next section details advanced options for modelers. For each modeling function in the package (Table 1) there is an optional argument `options` to change the modeling procedure. The value of `options` is a list in which each element sets one option, the help file found under `?cm_options` shows all available options, some of which we will detail next.

Modelers can change the lower and upper bounds of the free parameters in a model by using the options `lb` and `ub`. For instance, in the `gcm()` model we can change the bounds of the parameter λ to range from a lower bound of 0 to an upper bound of 20 by setting `options = list(lb = c(lambda = 0), ub = c(lambda = 20))`. If only `lb` is set, only the lower parameter bound is changed, and if only `ub` is set, only the upper parameter bound is changed.

Modelers can change the goodness of fit measure that the model optimizes during parameter estimation. The syntax is `list(fit_measure = ...)` where `...` can be `"loglikelihood"` or `"mse"` (mean-squared error). The log likelihood assumes a binomial probability density function when modeling discrete responses and a normal density function when modeling continuous responses. In the latter case the model assumes the responses to follow a normal distribution around each prediction (as mean) with a constant standard deviation that is estimated as an additional free parameter called `"sigma"`.

Modelers can change the algorithm that solves the parameter optimization problem. The option is set by `list(solver = ...)`. Currently, 22 different optimization solvers are available, which can be viewed by running `cm_solvers()`. The available solvers consist of all solvers in the R optimization infrastructure ROI, which include solvers such as global optimization by differential evolution (`solver = "Deoptim`), nonlinear optimization routines (`"nloptr"`), or optimx (`"optimx"`), and others (see their webpage for available solvers). Also available are a general nonlinear optimization using the augmented Lagrange multiplier method (`solver = "solnp"`) and a simple grid search (`solver = "grid"`). Lastly, setting `solver = c("grid", "solnp")` will first perform a coarse grid search and use the best solutions from this grid as starting values for repeated optimization with the solver (`"solnp"` in this example, but each available solver can be applied for this second step).

Advantages of Cognitivemodels

The **cognitivemodels** package is one way to facilitate the usage of formalized theories and to achieve robust cognitive modeling, which has been called for in recent meta-scientific proposals (e.g. Guest & Martin, 2020; Benureau & Rougier, 2018; Lee et al., 2019; Navarro, 2019; Van Rooij & Baggio, 2020; Wilson et al., 2017). Besides robustness of code, further advantages of the package are programming efficiency and flexibility.

Robustness. The models implemented in the **cognitivemodels** package are robust, because the package contains automated error checks and allows for manual error checks. By automated error checks we mean that there are unit tests implemented for the cognitive models in the package. Unit tests are automatic tests of parts of a program. Our unit tests test whether the model predictions are correct across a range of model parameters and input data. They also test if the default parameter estimation method replicates previously-obtained parameter values given published data. This is a safeguard against introducing programming errors during code development. By human error checks we mean that because the **cognitivemodels** package is open source, users of the package can report bugs through https://github.com/JanaJarecki/cognitivemodels/issues.

Programming Efficiency. The **cognitivemodels** package offers modelers a tool to fit cognitive models with minimal programming effort, because it uses a standardized syntax across the different cognitive models that are implemented in the package. The package requires minimal additional code to constrain parameters, make predictions, and compare models. The syntax of our package is similar to the syntax of the standard ANOVA or regression commands in R (e.g., the `predict()`, `anova()`, `logLik()`, or `coef()` methods). We believe, syntax standardization leads to efficiency gains when implementing models. Further, if standard models need not be implemented anew this saves implementation time.

Flexibility. The syntax for cognitive models in **cognitivemodels** allows modelers to adjust the modeling procedure to their own needs, to set parameter constraints, and to compare models. The package offers a general-purpose model development back-end in the R6 class which experienced modelers can use to implement further cognitive models in the package. This feature is not documented here, because of space limitations.

Summary

We have introduced **cognitivemodels**, the first R package to provide a robust, unified interface for formal modeling in cognitive science. Formalizing theories is seen as a crucial step to overcome the replication crisis in psychology. The **cognitivemodels** package may facilitate this step through its standardized and flexible syntax adapted to the needs of both novice and experienced modelers. We have exemplified the syntax of **cognitivemodels** with the generalized context model by applying basic and advanced modeling functionalities including model fitting with different types of parameter constraints and model testing with various

goodness of fit measures.

References

‘‘‘

Andraszewicz, S. (2014). *Quntitative [i.e. Quantitative] analysis of risky decision making in economic environments* (PhD thesis).

Benureau, F. C. Y., & Rougier, N. P. (2018). Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. *Frontiers in Neuroinformatics*, *11*(January), 1–8. http://doi.org/10.3389/fninf.2017.00069

Griffiths, T. L., Kemp, C., & Tenenbaum, J. B. (2008). Bayesian models of cognition. In R. Sun (Ed.), *The cambridge handbook of computational psychology* (pp. 59–100). Cambridge, UK: Cambridge University Press.

Guest, O., & Martin, A. E. (2020). *How Computational Modeling Can Force Theory Building in Psychological Science*.

Houston, A. I., & McNamara, J. M. (1988). A framework for the functional analysis of behaviour. *Behavioural and Brain Science*, *11*, 117–163. http://doi.org/10.1017/S0140525X00053061

Jarecki, J. B., Tan, J. H., & Jenny, M. A. (2020). A Framework for Building Cognitive Process Models. *Psychonomic Bulletin & Review*. http://doi.org/10.3758/s13423-020-01747-2

Juslin, P., Olsson, H., & Olsson, A.-C. (2003). Exemplar effects in categorization and multiple-cue judgment. *Journal of Experimental Psychology: General*, *132*(1), 133–156. http://doi.org/10.1037/0096-3445.132.1.133

Kass, R. E., & Raftery, A. E. (1995). Bayes Factors. *Journal of the American Statistical Association*, *90*(430), 773–795. http://doi.org/10.1080/01621459.1995.10476572

Lee, M. D., Chriss, A. H., & Vandekerckhove, J. (2019). Robust Modeling in Cognitive Science. *Computational Brain & Behavior*, *2*, 141–153. http://doi.org/10.1007/s42113-019-00029-y

Navarro, D. J. (2019). Between the Devil and the Deep Blue Sea: Tensions Between Scientific Judgement and Statistical Model Selection. *Computational Brain & Behavior*, *2*, 28–34. http://doi.org/10.1007/s42113-018-0019-z

Nosofsky, R. M. (1986). Attention, similarity, and the identification-categorization relationship. *Journal of Experimental Psychology: General*, *115*(1), 39–57. http://doi.org/10.1037/0096-3445.115.1.39

Nosofsky, R. M. (1989). Further tests of an exemplar-similarity approach to relating identification and categorization. *Perception & Psychophysics*, *45*(4), 279–290. http://doi.org/10.3758/BF03204942

Nosofsky, R. M. (2011). The Generalized Context Model: An Exemplar Model of Classification. In E. M. Pothos & A. J. Wills (Eds.), *Formal approaches in categorization* (pp. 18–39). Cambridge, UK: Cambridge University Press.

R Core Team. (2019). R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from http://www.r-project.org/

Scheibehenne, B., & Pachur, T. (2014). Using Bayesian hierarchical parameter estimation to assess the generalizability of cognitive models of choice. *Psychonomic Bulletin & Review*, 391–407. http://doi.org/10.3758/s13423-014-0684-4

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464. http://doi.org/10.1214/aos/1176344136

Singmann, H., & Kellen, D. (2013). MPTinR: Analysis of multinomial processing tree models in R. *Behavior Research Methods*, *45*(2), 560–75. http://doi.org/10.3758/s13428-012-0259-0

Tversky, A., & Kahneman, D. (1992). Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and Uncertainty*, *5*(4), 297–323. http://doi.org/10.1007/BF00122574

Van Rooij, I., & Baggio, G. (2020). Theory before the test: How to build high-verisimilitude explanatory theories in psychological science. Retrieved from https://psyarxiv.com/7qbpr/

Wagenmakers, E.-j., & Farrell, S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin & Review*, *11*(1), 192–196. http://doi.org/10.3758/BF03206482

Wakker, P. P. (2008). Explaining the characteristics of the power (CRRA) utility family. *Health Economics*, *17*(12), 1329–1344. http://doi.org/10.1002/hec.1331

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, *13*(6), 1–20. http://doi.org/10.1371/journal.pcbi.1005510