Fitting computational models to data: A tutorial

Timothy Ballard, Hector Palada, & Andrew Neal
The University of Queensland

Citation: Ballard, T., Palada, H., Neal, A. (2023). Fitting computational models to data: A tutorial. In J. B. Vancouver, M. Wang, J. M. Weinhardt (Eds.), *Computational Modeling for Industrial-Organizational Psychologists* (pp 255-296). Taylor Francis.

Abstract

Computational modeling is being increasingly advocated as a method for developing and testing theory in industrial and organizational psychology. This chapter provides a tutorial on fitting computational models to data, which is an important step in testing a model. The aim is to demonstrate the steps required to code up a model, estimate its parameters, quantify model-data correspondence, and compare alternative models. We begin by describing why model fitting is important. We then walk the reader through each step using an existing computational model– the multiple-goal pursuit model–as an example. We end by discussing how the approach outlined in this chapter can be extended to more complex research questions. This tutorial is aimed at the reader who has some conceptual familiarity with computational modeling and is looking to implement a model of their own. The tutorial is conducted using the R statistical programming language, although we have aimed to make it accessible to readers with little to no knowledge of R. All the data and code required to conduct the analyses demonstrated in this chapter (in both R and MATLAB) are can be found at https://osf.io/eq6tg/.

One of the goals of Industrial-Organizational (I/O) psychology is to understand the laws or principles that govern behavioral and organizational phenomena. What makes this process challenging, however, is that these laws and principles are not directly observable (Farrell & Lewandowsky, 2018; Heathcote, Brown, & Wagenmakers, 2015; Myung, 2003). For example, the concept of skill acquisition is often used to explain why task performance improves with practice (Kanfer & Ackerman, 1989). Yet the processes underlying skill acquisition are not observable. It is impossible to directly quantify the rate at which skill is acquired or the level of skill a person has at a given point in time. These things must be inferred based on patterns of behavior or performance.

In order to develop and test explanations for how unobserved processes operate, I/O psychologists rely on models. Although conceptual (e.g., verbal or pictorial) models have traditionally dominated the literature, computational models are becoming more widely used. A computational model is a representation of a theory that is expressed mathematically or using formal logic (see Ballard, Palada, Griffin, & Neal, 2019). Computational models produce precise, quantitative predictions regarding the pattern of behavior that should be observed given the assumptions of the

theory. This ability to generate such precise predictions strengthens the link between the theory and the hypotheses implied by the theory, which ultimately allows for a more direct test of the theory (Oberauer & Lewandowsky, 2019).

To illustrate the importance of precision for theory testing, consider the following example inspired by Meehl (1978). Suppose three meteorologists with competing theories of atmospheric dynamics put their theories to the test by attempting to predict next month's rainfall in Australia. The first theory predicts that it will, indeed, rain at some point next month in Australia, and this prediction turns out to be supported. The second theory predicts that it will rain more next month than it did this month, and this too is eventually supported. The third theory predicts that next month Australia will receive an average rainfall of between 100 and 120 mm, which also ends up being correct. Which theory do you think is the most credible? 1

This example illustrates that theories should not be evaluated simply on the basis of whether or not its predictions are supported. To do so would lead to the erroneous conclusion that all three of the theories above are equally supported by the evidence. The researcher also needs to consider to what extent support for a theory's predictions constitutes evidence for the theory itself. The first two theories in the example above make predictions that are quite broad, and there are likely many different theoretical explanations that would lead to the same predictions. Thus, support for these predictions can only ever provide relatively weak evidence for the theory. The third prediction, however, is more precise and is less likely to happen by chance or overlap with the predictions of another theory. This precision, therefore, means that we should be more persuaded of the theory's credibility when its predictions are supported.

The precise nature of computational models' predictions makes theories easier to test. This precision facilitates the comparison of competing explanations by clarifying the differences in their predictions. It also makes it easier to identify explanations that fail to account for observable phenomena. Ultimately, this precision allows the researcher to better understand the implications of the data for the theory.

The purpose of this chapter is demonstrate how to use empirical data to test a computational models' predictions. Previous chapters in this volume have addressed the issue of how to develop computational models and generate predictions (CITE REFS WITHIN VOLUME). Here, we focus on the issue of testing models against data. In the same way as in a structural equation modeling analysis, computational models need to be *fit* to the data in order to be tested. That is, one usually needs to estimate certain model parameters from data and quantify the alignment between the data and the model predictions. However, unlike structural equation modeling, computational models usually cannot be implemented in statistical packages such as *Mplus*, *SAS*, or *Stata*. Thus, fitting computational models to data often requires a few extra steps beyond coding the model itself. In the next section, we describe the model fitting process and further discuss its importance for theory testing. The rest of the paper provides a hands-on tutorial which demonstrates the model fitting process using a computational model of multiple-goal pursuit. The tutorial concludes with an example results section, which demonstrates how the results from the analysis we present in our tutorial may be written up for publication.

What is Model Fitting and Why is it Important?

Fitting a model to data refers to the process of adjusting model parameter values in such a way as to maximize the correspondence between the data and the predictions of the model. Farrell and Lewandowsky (2018) describe this process by comparing the model parameters to tuning knobs on

an analogue radio. As the listener turns the knob, the radio picks up different stations. In the same way, changing the values of model parameters alters the predictions the model makes. Consider a simple example of a univariate regression model, which has two parameters: an intercept and a slope. Each parameter has a unique effect on the model predictions. Increasing the value of the slope parameter increases the strength of the predicted relationship between the predictor and outcome variable. Increasing the intercept value increases the overall predicted level of the outcome variable.

To illustrate why the model fitting process is important for testing theory, consider the more familiar case of structural equation modeling. As with analysis based on computational models, structural equation modeling usually involves testing a series of alternative models in order to build support for a set of theoretical assumptions by ruling out alternative explanations. However, suppose a researcher decided to skip the model-fitting step in such an analysis. That is, they simply chose parameter values that they believed to be reasonable and evaluated model-data alignment, for example, via measures such as the comparative fit index (CFI), Tucker-Lewis index (TLI), or the root mean squared error of approximation (RMSEA) based on output of the model under those parameter values. Under this approach, how confident would you be in the conclusion that Model 1 provides a better account that of the data than Model 2?

The issue with this approach is that lack of model-data correspondence is ambiguous. Suppose Model 2 is found to be a poor approximation of the data (e.g., the CFI and TLI have relatively low values and the RMSEA is relatively high). This could suggest that Model 2 provides a poor explanation of the process being investigated and should be ruled out in favor of Model 1. However, the poor model-data correspondence under Model 2 could simply be the result of the researcher's choice of parameter values. It may be that there is another set of parameters under which Model 2 provides a better approximation of the data than Model 1.

Fitting the model to the data allows the researcher to identify the parameters that produce the output that most closely aligns with the data. This gives the model the best possible chance of accounting for a set of empirical observations. When this is the case, lack of model-data correspondence is less ambiguous. If the researcher knows for certain that there is no combination of parameter values under which the model more accurately characterizes the data, they can conduct a fair assessment of the model's adequacy. For example, Ballard, Vancouver, and Neal (2018) fit several versions of a computational model of multiple-goal pursuit to data from an experiment in which participants pursued two goals with different deadlines. They demonstrated that an earlier version of the model could not account for the tendency to prioritize the goal with the shorter deadline. They interpreted this finding as evidence against this version of the model. This conclusion would not have been possible had the models not been fit to the data, because failure of the model to accurately characterize the data would be ambiguous. Such a failure could be due to the inadequacy of the model itself or to the inadequacy of the parameter values used to generate the model predictions.

Model fitting also facilitates the comparison of alternative models. Modellers will typically evaluate the evidence for a hypothesized model by comparing it to a set of theoretically plausible alternative models (Heathcote et al., 2015). The evidence in favor of a model is much stronger when the researcher demonstrates that a hypothesized model not only adequately characterizes the data, but also does a better job of accounting for the data than competing models. The challenge however is that it can be difficult to determine what is meant by "a better job of accounting for the data". Typically, the researcher will wish to establish that the model adheres to the principle of

parsimony, that is, that the model provides the simplest possible explanation that accounts for the pattern of observed behavior. Ideally, this involves demonstrating that simplifying the model further by removing certain assumptions undermines the model's ability to capture essential features of the data, as well as demonstrating that an increase in the complexity of the model provides little to no increase in the alignment between model and data. Model fitting allows the researcher to quantify both the correspondence between the model and the data and the complexity of the model, which together help the researcher assess the model's parsimony. For example, the AIC (Akaike, 1973) and BIC (Schwarz, 1978) are commonly used metrics for quantifying the ability of the model to capture that data that account for both model-data correspondence and model complexity (as measured by the number of estimated parameters). Such indices help the researcher to select the model that provides the best trade-off between model-data correspondence and complexity, provided that model is theoretically plausible.

Model fitting is also important because it allows parameters to be *estimated* from the data. This allows the researcher to obtain useful information about the process under investigation. In the case of a regression analysis, for example, the researcher would interpret the estimate of the slope parameter to make inferences about the relationship between the predictor and outcome variable. In the same way, estimates of computational model parameters can be used to make inferences about latent components of the process that the model purportedly represents. For example, Zhou, Wang, and Zhang (2019)'s computational model of leadership goal striving contained a parameter that reflected the extent to which the leader prioritizes working on their own tasks as opposed to acting to support their subordinates. After fitting their model to experimental data, they found the estimated value of this parameter to be relatively low, indicating that leaders in their study tended to prioritize their subordinates' tasks.

Parameter estimates are often interpreted by comparing them across individuals or experimental conditions. For example, Vancouver, Weinhardt, and Schmidt (2010) found individual differences in the estimates of a parameter that represents sensitivity to time and deadlines. Between-person variation in this parameter accounted for individual differences in resource allocation patterns over time. This type of comparison is analogous examining the bi-variate correlation between a pair of variables, with the main difference being that one of the variables is estimated by the model rather than directly observed. Ballard, Yeo, Loft, Vancouver, and Neal (2016) found that a similar parameter in their model was affected by goal type and concluded that people were less sensitive to deadlines when pursuing avoidance goals compared with approach goals. This type of comparison involves a comparison of means (which conventionally might be conducted using a t-test or ANOVA), though the inferences are best on estimated variables rather than observed ones.

In the next section, we begin our tutorial by describing the model that we use as an example to illustrate the steps required to fit a model to data. The model we use is the multiple-goal pursuit model (MGPM; Ballard, Vancouver, & Neal, 2018; Ballard, Yeo, Loft, et al., 2016; Vancouver, Weinhardt, & Schmidt, 2010; Vancouver, Weinhardt, & Vigo, 2014), which attempts to explain how people make prioritization decisions when managing competing goals. In the section that follows, we show how to translate the model from a set of equations to computer code. In subsequent sections, we demonstrate how to use this code to fit the model to empirical data, and how the result from this analysis can be written up for publication. We conclude by discussing how this approach can be extended to develop and test more complex models.

The approach we demonstrate uses maximum likelihood methods for parameter estimation, which is a frequentest approach to model fitting. Although parameter estimation can also be done in

using the Bayesian framework (e.g., Ballard, Palada, et al., 2019; Ballard, Vancouver, & Neal, 2018), we decided to use maximum likelihood methods in this tutorial for two reasons. First, we wanted to make this tutorial as accessible as possible. We expect that most readers will be more familiar with frequentest methods than Bayesian ones and did not want these readers to have to learn Bayesian methods in order to make use of the content on model fitting. Second, many of the principles of maximum likelihood parameter estimation are directly relevant to Bayesian parameter estimation. This tutorial should therefore provide a useful foundation for readers who eventually wish to use Bayesian methods. We elaborate on some of the useful features of the Bayesian approach in the "Extensions" section.

The Multiple-Goal Pursuit Model

Multiple-goal pursuit is the process by which people manage competing demands on their on their time and other resources as they strive to achieve desired outcomes and to avoid undesired outcomes (Ballard, Yeo, Neal, & Farrell, 2016; Neal, Ballard, & Vancouver, 2017; Schmidt & DeShon, 2007; Schmidt & Dolis, 2009; Schmidt, Dolis, & Tolli, 2009). Multiple-goal pursuit is challenging because allocating time and effort toward one goal will often undermine our capacity to make progress on other goals. As a result, we must make decisions about which goal to attend to at a given point in time and repeatedly reevaluate these decisions as priorities change. The multiple-goal pursuit model (MGPM; Ballard, Vancouver, & Neal, 2018; Ballard, Yeo, Loft, et al., 2016; Vancouver, Weinhardt, & Schmidt, 2010; Vancouver et al., 2014) is a computational model that has resulted from over a decade of research into the process by which people make prioritization decisions. The core assumption of the MGPM is that people make prioritization decisions by weighing up the need to act on the goal, which Vancouver, Weinhardt, and Schmidt (2010) refer to as *valence*, and the perceived likelihood of achieving the goal, which is referred to as *expectancy*. Valence and expectancy combine to form a perception regarding the *expected utility* of acting on the goal, which represents the degree to which the individual is motivated to prioritize the goal at that point in time.

As the multiple-goal pursuit model is a computational model, it is articulated in the form of mathematical equations. These equations describe the relationships between theoretical constructs proposed by the model and ultimately the pattern of observed behavior that should emerge if the model's assumptions are correct (the model's predictions). There are two types of constructs that are operationalized in the equations. The first type are variables that are directly observable (e.g., the time available before a deadline). The values of these variables can usually be ascertained based on the environment and are akin to independent variables in a regression equation. The second type are parameters. Parameters usually reflect constructs that are not directly observable and therefore must be inferred based on patterns of observable behavior. For example, it would be difficult to directly measure the extent to which people are sensitive to differences between the time available and the time required to reach the goal (referred to as time sensitivity). However, we can make inferences about this construct by examining the prioritization decisions that people make when faced with different deadlines. Parameters are referred to as *free* when their values are not known beforehand but rather estimated from the data (akin to regression coefficients). Parameters are referred to as *fixed* when their values are defined in advance.

Here we briefly describe the equations that form the model, which we translate to computer code in the next section. At the time of writing, the version of the MGPM introduced by Ballard, Vancouver, and Neal (2018) represents the latest and most general version of the model. We therefore focus on this version of the model in the tutorial. The valence of acting on goal i at time t

(denoted $V_k(t)$) can be defined formally as follows[1]:

$$V_i(t) = \kappa_i \cdot \frac{TR_i(t)}{TA_i(t)}, \tag{1}$$

where $TR_i(t)$ represents the time required to reach the goal and $TA_i(t)$ represents the time available before the deadline. The time required is the product of the *distance* to goal *i* at time *t* (denoted $d_i(t)$) and a one's belief regarding the time needed to reduce the distance to goal *i* by a single unit (referred to as the *expected lag*). The time available and distance to the goal are generally assumed to be observable variables. κ_i is a gain parameter that reflects the importance of the goal. In most applications of the MGPM, κ_i has been treated as a fixed parameter with its value set to one. However, κ_i may be useful as a free parameter in settings where one wishes to examine the extent to which goals differ in importance.

The expectancy of achieving goal *i* at time *t* (denoted $E_i(t)$) can be defined as follows:

$$E_i(t) = \frac{1}{1 + \exp[-\gamma \cdot (TA_i(t) - TR_i(t))]}. \tag{2}$$

According to Equation 2, expectancy is a function of the difference between the time available and the perceived time required to reach the goal. If the time available and time required are equal, expectancy will be 0.5. Expectancy will be greater than 0.5 when the time available exceeds the time required, and less than 0.5 when the time required exceeds the time available. The γ parameter represents *time sensitivity* and determines how strongly expectancy is affected by the difference between the time available and time required. When $\gamma = 0$, the person is completely insensitive to the difference between the time available and time required, and expectancy is always 0.5. As γ increases, expectancy becomes more sensitive to this difference. Time sensitivity is usually treated as a free parameter.

The expected utility of acting on a goal is influenced by the product of valence and expectancy. However, Ballard, Vancouver, and Neal (2018) showed that expected utility can also be subject to temporal discounting (e.g., Ainslie & Haslam, 1992; Steel & König, 2006). Under this assumption, people discount goals that have deadlines that are relatively far away, treating them as less important than they would if the deadline were nearer. The expected utility of prioritizing goal *i* at time *t* is therefore a function of valence, expectancy, and the time to the deadline, which can be defined as follows:

$$U_i(t) = \frac{V_i(t) \cdot E_i(t)}{1 + \Gamma TA_i(t)}, \tag{3}$$

where Γ is a parameter that refers to the *discount rate*, which represents the extent to which future deadlines are discounted. When $\Gamma = 0$, no discounting occurs. In this case, deadlines have no influence on expected utility (over and above their influence on valence and expectancy). As Γ increases, discounting becomes stronger. Like time sensitivity, discount rate is usually treated as a free parameter.

The MGPM assumes that the actions that one may take in order to progress toward a goal can have uncertain consequences. The attractiveness of prioritizing a goal at a given point in time depends on the particular consequence being considered. When considering the possibility that good

[1]The MGPM addresses both approach and avoidance goals. For simplicity, however, we limit our discussion to approach goal pursuit. The model of avoidance goal pursuit has been presented by Ballard, Yeo, Loft, et al. (2016)

progress can be made, prioritizing the goal will be attractive. On the other hand, when considering the possibility that little or no progress would be made, prioritizing the goal is less attractive. The attractiveness of an action *j* at time *t* is referred to its *momentary attractiveness* (denoted $A_j(t)$) and is defined as follows:

$$A_j(t) = \sum\nolimits_i U_i(t) \cdot Q_{ij}(t), \tag{4}$$

where $Q_{ij}(t)$ refers to *quality*, which represents the impact of the consequence that is being considered at time *t* on goal progress. Specifically, $Q_{ij}(t)$ represents the consequence for goal *i* that would result from action *j*. Quality fluctuates over time as people consider different consequences of each action. $Q_{ij}(t)$ is positive when the person anticipates that an action will have a beneficial effect on goal progress, with higher values indicating more beneficial effects. $Q_{ij}(t)$ is negative when the person anticipates that an action will have a detrimental impact on goal progress. Quality is most often treated as an observable variable that is defined based on the environment. The momentary attractiveness of action *j* is the sum of the product of quality and expected utility for all goals that are affected by that action.

According to the MGPM, the preference for each action evolves over time according to a sequential sampling process, as the person considers different possible consequences of the actions they can take. In our example, we consider a scenario where the person is striving for two goals and therefore has two possible actions: prioritize Goal 1 or prioritize Goal 2. In this case, the change in preference over time can be described by the following equation:

$$P(t) = P(t-1) + [A_1(t) - A_2(t)], \tag{5}$$

where $A_1(t)$ and $A_2(t)$ represent the momentary attractiveness of prioritizing Goals 1 and 2 respectively. According to this equation, preference is measured on a bipolar continuum with positive values indicating a preference for prioritizing Goal 1 and negative values indicating a preference for prioritizing Goal 2. The change in preference at time *t* is determined by the the difference in the momentary attractiveness between the two actions being considered. The model assumes that the person continues to consider different consequences until preference for one action breaches a threshold, at which point that action is selected.

Because the quality of action j with respect to goal i fluctuates over time as different consequences are considered, preference accumulation is stochastic. Thus, we cannot predict with certainty which action will be selected. We can only predict the *probability* of each action being chosen. In the scenario described above where there are two actions being considered, the probability of selecting Action 1 can be calculated as follows:

$$p_1(t) = L\bigg(2 \cdot \frac{E[A_{diff}(t)]}{\sqrt{(Var[A_{diff}(t)]}} \cdot \theta\bigg), \tag{6}$$

,

where L represents the standard cumulative logistic distribution function, $f(x) = 1/[1+\exp(-x)]$. The probability of choosing Action 2 is simply $1 - p_1(t)$. The θ parameter represents the threshold that determines the preference strength required before an action is selected (usually treated as a free parameter). High thresholds mean that a stronger preference is required before an action is selected. $E[A_{diff}(t)]$ and $Var[A_{diff}(t)]$ refer to the mean and variance of the difference in momentary

attractiveness between the two actions at any given point in time. These quantities can be calculated from the mean and variance of the momentary attractiveness of each action:

$$E[A_{diff}(t)] = E[A_1(t)] - E[A_2(t)], \tag{7}$$

$$Var[A_{diff}(t)] = Var[A_1(t)] + Var[A_2(t)]. \tag{8}$$

In the above equations, $E[A_i(t)]$ and $Var[A_i(t)]$ refer to the mean and variance of the momentary attractiveness of Action i respectively. The mean momentary attractiveness represents the average attractiveness that would be expected across the different consequences considered. The variance in the momentary attractiveness reflects the extent to which attractiveness deviates from the mean as different consequences are considered. These quantities can be calculated based on the quality of the relevant action with respect to each goal and the utility of the goal:

$$E[A_j(t)] = \sum\nolimits_i U_i(t) \cdot E[Q_{ij}], \tag{9}$$

$$Var[A_j(t)] = \sum\nolimits_i U_i(t)^2 \cdot Var[Q_{ij}], \tag{10}$$

where $E[Q_{ij}]$ and $Var[Q_{ij}]$ represent the mean and variance of the quality of action *j* with respect to goal *i*.

Translating the Model to Code

Once the model has been specified in equation form, the next step is to translate the equations into computer code. The researcher needs to create several functions in order in construct the code that will be used for fitting the model. In this section and the ones that follow, we explain these functions in detail. The first function that must be created is one that takes as inputs the parameter values and the data required to generate predictions and returns the model predictions. In the case of the MGPM, the predictions come in the form of a probability of prioritizing each goal. In this example, the parameter values required to generate the model's prediction are the values of the three free parameters–time sensitivity, discount rate, and threshold–and the data required are variables such as the time available for and distance to each goal. Figure 1 shows an illustration of how this function works. The name of the function is `MGPM_predictions` and the function has two input arguments: 1) the parameter values, and 2) a dataset containing the variables required to generate model predictions. In the subsection below, we describe the structure of each of these input arguments. We then explain how the function transforms the inputs into a prediction regarding goal prioritization.

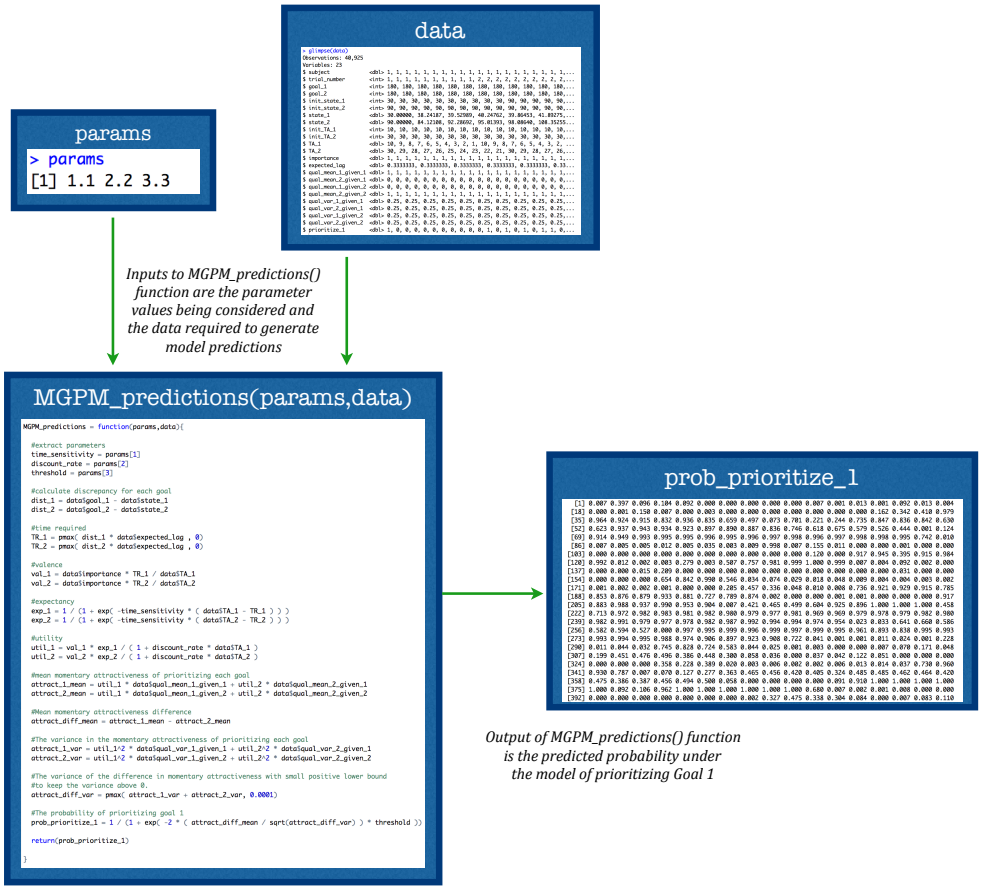

Figure 1. Structure of the `MGPM_predictions` function. The `params` argument contains the parameter values being considered. The `data` argument is a dataset containing the values of the variables required to generate model predictions. The `MGPM_predictions` function returns the predicted probability of prioritizing Goal 1 under the parameters values in `params` for every observation in `data`.

The Input Arguments

The first input argument is a vector called `params` which contains the values of the model parameters that are being used to generate model predictions. When the model is fit to the data, these parameter values will be repeatedly adjusted until the combination of parameter values is found that maximizes the correspondence between the data and the predictions that are made by the model under those parameter values. This is how the parameter values are estimated from the data. In this model, there are three parameters being estimated: time sensitivity (γ), discount rate (Γ), and threshold (θ). We therefore assume the `params` object is a vector with three elements (one for each estimated parameter). The value of the time sensitivity parameter being considered at that point occupies the first element in the vector. The discount rate parameter occupies the second element and the threshold occupies the third. In Figure 1, the values being considered are 1.1, 2.2, and 3.3 respectively. However, these values will be updated repeatedly during the parameter estimation

process as different combinations of parameters are tested.

The second input argument is a data frame (an R object containing a dataset) called `data` which contains the variables needed to generate model predictions. The data is structured in the long format, where each row represents an observation, and each column represents a variable. The variables in the dataset are assumed to be known quantities, as opposed to values that are estimated by the model. In this way, they are akin to predictors in a regression analysis. The dataset we use in this tutorial is from Experiment 1 in Ballard, Vancouver, and Neal (2018). In this experiment, 48 participants played a computerized farming game, which was broken down into a series of trials. In each trial, participants had to manage two crops. Their objective was to ensure that the height of each crop exceeded a target height at the end of the trial. Each time step in the game represented one week in the growing season. Participants facilitated the growth of the crops by irrigating them. However, only one crop could be irrigated in each week. Thus, in each week participants had to choose which crop to prioritize.

In this experiment, the distance to the goals and the deadlines were manipulated by varying the height of the crop at the start of the growing season and the number of weeks in the growing season for one of the two crops (this crop was referred to as the experimental crop). These properties were held constant for the other crop (the fixed crop). The growing season for the experimental crop varied across five levels (10, 20, 30, 40, and 50 weeks). The starting height of the experimental crop was varied across five levels (30, 60, 90, 120, and 150 cm). The target height for each crop was always 180 cm, so the five starting heights corresponded to distances of 150, 120, 90, 60, and 30 cm respectively. The fixed crop always had a deadline of 30 weeks and a starting height of 90 cm (which corresponded to a distance of 90 cm from the goal). This resulted in a 5 (initial time available: 10, 20, 30, 40, or 50 weeks) x 5 (initial distance to goal: 30, 60, 90, 120, or 150 cm) within-participants design, with each participant completing each of the 25 unique combinations twice.

The output below shows a summary of the dataset. As can be seen, there are 40,925 observations in total. Each observation in the dataset represents one week in the growing season. The `subject` and `trial_number` variables identify the participant and the trial (each participant performed 50 trials). The `goal` variables correspond to the target height. The suffixes `_1` and `_2` represent the experimental and fixed crops respectively. The `init_state` variables represent the height of the crop at the start of the trial, whereas the `state` variables represent the height of the crop before each observation was made. The `init_TA` variables represent the length of the growing season for each crop, whereas the `TA` variables represent the number of weeks remaining in the growing season before each observation was made. As can be seen, the `TA` variables decrease by 1 with each successive observation within a trial.

```
Observations: 40,925
Variables: 23
$ subject             <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ trial_number        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2...
$ goal_1              <int> 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, ...
$ goal_2              <int> 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, ...
$ init_state_1        <int> 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 90, 90, 90, 90,...
$ init_state_2        <int> 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90,...
$ state_1             <dbl> 30.00000, 38.24187, 39.52989, 40.24762, 39.86453, 41.89...
$ state_2             <dbl> 90.00000, 84.12108, 92.28692, 95.01393, 98.08640, 108.3...
$ init_TA_1           <int> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,...
$ init_TA_2           <int> 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30,...
$ TA_1                <dbl> 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3,...
$ TA_2                <dbl> 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 30, 29, 28, 27,...
$ importance          <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
```

```
$ expected_lag        <dbl> 0.3333333, 0.3333333, 0.3333333, 0.3333333, 0.3333333, ...
$ qual_mean_1_given_1 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ qual_mean_2_given_1 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
$ qual_mean_1_given_2 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
$ qual_mean_2_given_2 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ qual_var_1_given_1  <dbl> 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0...
$ qual_var_2_given_1  <dbl> 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0...
$ qual_var_1_given_2  <dbl> 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0...
$ qual_var_2_given_2  <dbl> 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0...
$ prioritize_1        <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1...
```

The `importance` variable contains the value of κ. We assume that the importance of each goal is fixed to a known value which is the same for the two goals. Importance is therefore a fixed parameter in this application of the MGPM. The value of the `importance` variable is one for every observation. The `expected_lag` variable also takes on the same value for every observation. We set the expected lag to one third because, in this experiment, it would take one third of a week on average to reduce the distance to each goal by one cm.

In this experiment, the impact of the chosen action on the progress for each goal depends on both the action and goal in question. Thus, there are four different qualities that need to be considered. The quality variables are labelled such that the first number corresponds to the goal that is being affected and the second number corresponds to the goal being prioritized. The `qual_mean` variables represent the mean quality for a given goal resulting from a particular prioritization decision ($E[Qij]$ in Equation 9). `qual_mean_1_given_1` therefore refers to the expected impact on the progress for Goal 1 when Goal 1 is prioritized. `qual_mean_2_given_1` refers to the expected impact on the progress for Goal 2 when Goal 1 is prioritized. `qual_mean_1_given_2` refers to the expected impact on the progress for Goal 1 when Goal 2 is prioritized. As can be seen by the values of the `qual_mean` variables, the expected impact on progress for the prioritized goal is set to one whereas the expected impact for the non-prioritized goal is set to zero. This reflects the notion that the irrigated crop was likely to grow in that week, whereas the non-irrigated crop was equally likely to increase versus decrease in height. The `qual_var` variables represent the variance in quality. ($Var[Qij]$ in Equation 10). The variance in the quality for each goal was always 0.25, regardless of which crop was prioritized. The `prioritize_1` variable is a binary variable representing whether or not the participant prioritized Goal 1 in that week of the growing season (1 if yes, 0 if no).

From Inputs to Model Predictions

The code below shows the function, called `MGPM_predictions`, which runs the model. By "run the model", we mean that this function converts the parameters and data into predictions regarding the prioritization of each goal. The first operation the function performs (lines 4-6) is to extract the values of the model parameters that are being tested from the `params` object. The `params[1]`, `params[2]`, and `params[3]` statements extract the first, second, and third element of the `params` vector respectively. The value contained in each element is then stored in its own object. The `time_sensitivity`, `discount rate`, and `threshold` objects will therefore contain the value of the time sensitivity, discount rate, and threshold parameters respectively[2].

```
1  MGPM_predictions = function(params,data){
2
3    #extract parameters
```

[2]Note that this operation is not required for model fitting, but it enhances the readability of the code. Throughout this tutorial, we have prioritized code readability over efficiency in order to facilitate understanding.

```
4    time_sensitivity = params[1]
5    discount_rate = params[2]
6    threshold = params[3]
7
8    #calculate distance to each goal
9    dist_1 = data$goal_1 - data$state_1
10   dist_2 = data$goal_2 - data$state_2
11
12   #time required
13   TR_1 = pmax( dist_1 * data$expected_lag, 0 )
14   TR_2 = pmax( dist_2 * data$expected_lag, 0 )
15
16   #valence
17   val_1 = data$importance * TR_1 / data$TA_1
18   val_2 = data$importance * TR_2 / data$TA_2
19
20   #expectancy
21   exp_1 = 1 / (1 + exp( -time_sensitivity * ( data$TA_1 - TR_1 ) ) )
22   exp_2 = 1 / (1 + exp( -time_sensitivity * ( data$TA_2 - TR_2 ) ) )
23
24   #utility
25   util_1 = val_1 * exp_1 / ( 1 + discount_rate * data$TA_1 )
26   util_2 = val_2 * exp_2 / ( 1 + discount_rate * data$TA_2 )
27
28   #mean momentary attractiveness of prioritizing each goal
29   attract_1_mean = util_1 * data$qual_mean_1_given_1 + util_2 * data$qual_mean_2_given_1
30   attract_2_mean = util_1 * data$qual_mean_1_given_2 + util_2 * data$qual_mean_2_given_2
31
32   #mean difference in momentary attractiveness
33   attract_diff_mean = attract_1_mean - attract_2_mean
34
35   #variance in the momentary attractiveness of prioritizing each goal.
36   attract_1_var = util_1^2 * data$qual_var_1_given_1 + util_2^2 * data$qual_var_2_given_1
37   attract_2_var = util_1^2 * data$qual_var_1_given_2 + util_2^2 * data$qual_var_2_given_2
38
39   #variance of the difference in momentary attractiveness.
40   attract_diff_var = pmax( attract_1_var + attract_2_var, 0.0001);
41
42   #probability of prioritizing Goal 1
43   prob_prioritize_1 = 1 / (1 + exp( -2 * ( attract_diff_mean / sqrt(attract_diff_var) ) * threshold ))
44
45   return(prob_prioritize_1)
46 }
```

The function then calculates a series of new variables according to the equations described above. On lines 9 and 10, the distance to each goal is computed by subtracting the current state with respect to that goal from the goal itself. Because `data` is a data frame object, accessing the variables contained within requires the `data$` prefix. For example, the variable representing Goal 1 would be accessed with `data$goal_1`.

On lines 13 and 14, the time required to reach achieve goal is calculated based on the distance to each goal and the expected lag. Note that time required is calculated in such a way that it is constrained to have a minimum value of 0. This constraint is implemented via the `pmax` function. This function works by, for each observation, comparing the product of distance and expected lag to a value of 0 and returning the maximum of the two values. The result is that `TR_1` and `TR_2` will never take on values less than 0. This constraint is needed because time cannot take on a negative value. When the person has surpassed the goal, no more time is needed to reach it, regardless of the extent to which the goal has been exceeded.

Lines 17 and 18 compute the valence of each goal according to Equation 1. Lines 21 and

22 compute the expectancy of each goal according to Equation 2. Lines 25 and 26 compute the expected utility of each goal according to Equation 3. Lines 29 and 30 compute the mean attractiveness of prioritizing each goal according to Equation 9. Line 33 computes the difference in the mean momentary attractiveness between the two actions based on Equation 7.

Lines 36 and 37 compute the variance in the momentary attractiveness of prioritizing each goal according to Equation 10. Line 40 then computes the variance in the difference in momentary attractiveness between the two actions according to Equation 8. When the variance in the difference in momentary attractiveness is 0, the ratio of the mean difference in momentary attractiveness to the variance of the difference is undefined. We therefore set this variance to have a lower bound of a small, positive number (in this case, 0.00001). This constraint is implemented via the same `pmax` function that was used to compute the time required, except here the lower bound is set to 0.00001 instead of 0. This constraint prevents the variance in the momentary attractiveness difference from taking on a value of 0. This constraint is needed because a variance of 0 means that the predicted probability of prioritizing Goal 1, which is computed on Line 43 according to Equation 6, would be undefined. An undefined prediction would cause an error in the code used to fit the model. Restricting the variance from taking on a value of 0 eliminates the opportunity for this error to emerge. Such constraints are often necessary when dealing with equations that may have values of 0 in their denominators. This is not problematic theoretically because there is virtually no difference in the predictions made by the model when the variance is 0.0001 compared to when the variance is even closer to 0. Changes in values at this scale have no real effect on the model's predictions. Thus, setting the lower bound of the variance to 0.0001 does not affect our ability to test the theory.

Defining the Cost Function

Model fitting involves generating predictions under different parameter values and examining the alignment between the data and model under each value considered. To do this, the modeller must first define a *cost* or *discrepancy* function that quantifies the model-data correspondence. The cost function is defined in such a way that higher values indicate poorer alignment between the model and data. In other words, the greater the output of the cost function, the larger the discrepancy between the model predictions and the empirical observations. The goal of model fitting, therefore, is to identify the parameter values that minimize the value of the cost function. The choice of cost function will depend on the nature of the data, the model, and the research question being investigated. One approach to defining a cost function is the least-squares method (e.g., Vancouver, Weinhardt, & Schmidt, 2010). Using this approach, a modeller working with continuous data might define the cost function as the square root of the mean squared difference between each observation and the model prediction that corresponds to that observation (commonly referred to as the root mean squared deviation or RMSD). A commonly-used analogue of the RMSD that is appropriate for categorical data is the χ^2 (or G^2) index (see Farrell & Lewandowsky, 2018).

Another approach to defining the cost function is maximum likelihood (e.g., Ballard, Yeo, Loft, et al., 2016). The maximum likelihood approach differs to the least-squares in that the goal is to identify the parameters that are most likely given the data, as opposed to the parameters that yield the smallest mean squared difference between the model and the data. This difference might seem subtle, but it has some important implications. A major advantage of the maximum likelihood approach is that the output of the cost function can be used to conduct quantitative model comparisons. Under maximum-likelihood estimation, the cost function defines the probability of the data having been observed given the model predictions that are generated under the parameter values

being considered. The goal is to find the parameters under which the data are most likely to have been observed. These are the parameter values that are most likely given the data.

We use the maximum likelihood approach to define the cost function for our example implementation of the MGPM. As the reader will see, the maximum likelihood approach makes it easy to transform the value of the cost function into an index that quantifies the ability of the model to parsimoniously account for the data, which can be used to compare the model to other theoretically plausible alternatives. The binary nature of the prioritization data makes the maximum likelihood approach straightforward to implement. Because the prioritization variable to which the model is being fit is binary (1 if Goal 1, 0 if Goal 2), the data can be modeled using a Bernoulli distribution (which is the same approach used in logistic regression). Recall that the model generates predictions regarding the probability that Goal 1 will be prioritized. This prediction can be converted into a likelihood by simply reading out the model's predicted probability of that decision having been observed. This can be done using the following rule: If Goal 1 is prioritized, the likelihood of that observation is equal to the predicted probability of prioritizing Goal 1 under the model. If Goal 2 is prioritized, the likelihood is equal to one minus the predicted probability of prioritizing Goal 1 under the model. For example, if the model predicts that the Goal 1 has a 0.75 probability of being prioritized, the likelihood of an observation in which Goal 1 is prioritized is 0.75 whereas the likelihood of an observation in which Goal 2 is prioritized is 0.25.

In order to implement the cost function, we need to construct a second function that computes the likelihood of the observed prioritization decisions under the parameter values in question. Figure 2 shows a diagram of how this second function works. The function is called `MGPM_likelihood` and it has the same two inputs as the `MGPM_predictions` function (the parameter values and the data). The `MGPM_likelihood` passes the information regarding the parameter values and the data to the `MGPM_predictions` function, which returns the predicted probability of prioritizing Goal 1 for each observation in the dataset. The `MGPM_likelihood` function uses this information to determine the likelihood of the data.

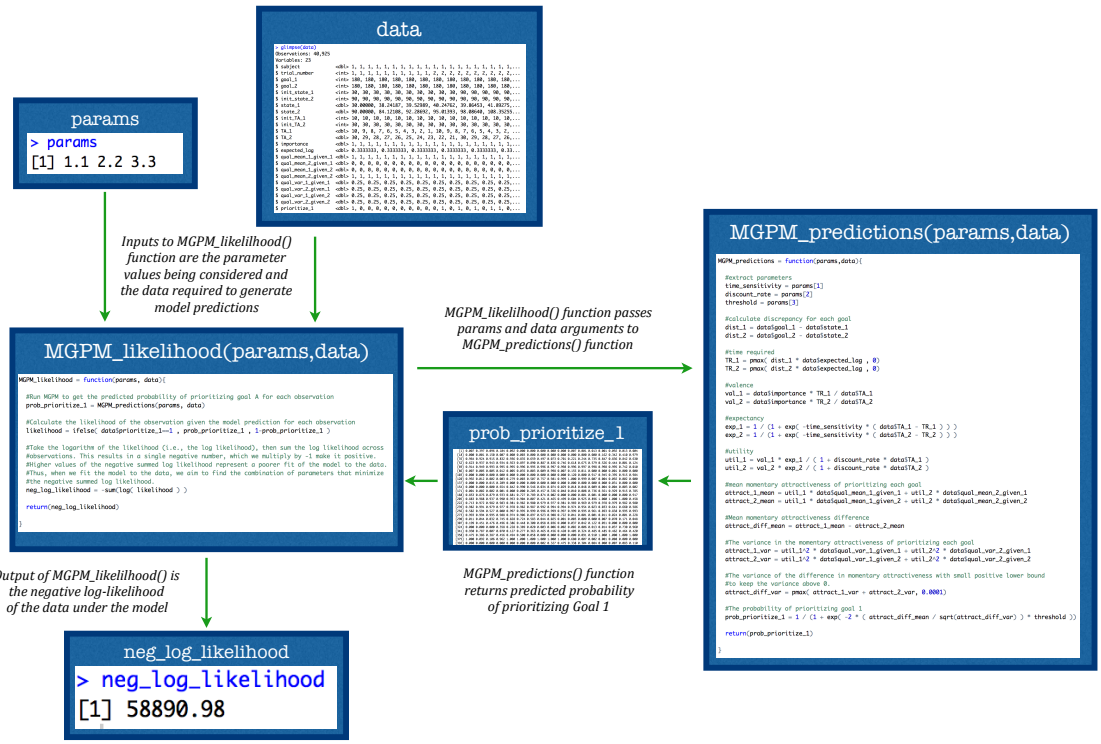

Figure 2. Structure of the `MGPM_likelihood` function. The `params` argument contains the parameter values being considered. The `data` argument is a dataset containing the values of the variables required to generate model predictions. The `MGPM_likelihood` function passes these arguments to the `MGPM_predictions` function, which returns the predicted probability of prioritizing Goal 1. This prediction is used to compute the negative log-likeliood, which is returned by the `MGPM_likelihood` function.

The code below shows the `MGPM_likelihood` function. This function performs three operations. On line 4, the `MGPM_predictions` function is run to generate the model predictions, which are stored in the `prob_prioritize_1` object. The likelihood of each observation is calculated on line 7. The likelihood is computed using the `ifelse` function, which performs a logical test on each element of the first input argument and returns the corresponding element of the second argument if the condition is met and the corresponding element of the third argument otherwise. Here, the logical test is whether Goal 1 is prioritized. The first argument in the `ifelse` is a logical test, which returns `TRUE` for observations where Goal 1 where prioritized and `FALSE` when Goal 2 was prioritized. For observations where Goal 1 is prioritized, the likelihood is equal to `prob_prioritize_1`, which is predicted probability of prioritizing Goal 1. For observations where Goal 2 is prioritized, the likelihood is equal to `1-prob_prioritize_1`, which is the predicted probability of prioritizing Goal 2. The likelihood of each observation is stored in the `likelihood` object

```
1  MGPM_likelihood = function(params, data){
```

```
2
3   #Run MGPM to get the predicted probability of prioritizing goal 1 for each observation
4   prob_prioritize_1 = MGPM_predictions(params, data)
5
6   #Calculate the likelihood of the observation given the model prediction
7   likelihood = ifelse( data$prioritize_1==1,
8                        prob_prioritize_1,
9                        1-prob_prioritize_1 )
10
11  #Take the logarithm of the likelihood (i.e., the log-likelihood),
12  #then sum the log-likelihood across observations and
13  #multiply by -1 to convert negative number to positive.
14  neg_log_likelihood = -sum( log( likelihood ) )
15
16  return(neg_log_likelihood)
17 }
```

There is one more operation that needs to be performed in order to calculate the cost function. In order for the algorithm that estimates the parameters to work, the cost function must represent the discrepancy between the model and the data as a single number. This means that we need to aggregate the values stored in the `likelihood` variable, which reflect the likelihood of each individual observation, to create a global index reflecting the likelihood of the data as a whole. Conceptually, this is straightforward. According to probability theory, the joint probability of observing a set of independent data points is equal to the product of the probabilities of observing each individual data point. Thus, the likelihood of the data as a whole can be calculated by multiplying the predicted probabilities of each observation across the entire dataset.

In practice, however, multiplying probabilities across an entire set of observations results in extremely small numbers that are often outside the range of values that can be represented by most computers. It is therefore common practice to compute the likelihood on a logarithmic scale. As can be seen on line 14 of the code above, we transform the likelihood of each individual observation to the logarithmic scale by taking the log of the `likelihood` variable. We then sum the log-likelihoods across observations (which is equivalent to multiplying the raw likelihoods) to create a single index representing the logged likelihood of the data as a whole having been observed. The sum of the log-likelihoods will be a negative number, with lower values representing data that are less likely under the model (i.e., poorer model-data correspondence). We therefore reverse the sign of the sum of the log-likelihoods so that the output of the cost function a positive number where higher values represent less likely data. Note that the log transformation is *monotonic*, which means that it does not change the relationship between the parameters and the output of the cost function. In other words, the combination of parameters that produces the lowest logged value of the cost function will be the same combination the produces the lowest raw value.

When modeling data that are continuous, rather than binary, a different likelihood function is needed. For normally distributed data, the likelihood can be computed by evaluating the probability density function of the normal distribution at the observed value (see Ballard, Palada, Griffin, & Neal, 2019; Neal, Gee, Ballard, Vancouver, & Yeo, under review). Most statistical or mathematical programming languages contain functions that automate this computation (e.g., the `dnorm` function in R or the `normpdf` function in MATLAB). For continuous data that are not normally distributed, such as highly skewed data, a different functional form may be more appropriate. For example, researchers seeking to model response times, which tend to be positively skewed, may elect to compute the likelihood using a log-normal, gamma, or ex-gaussian distribution.

Minimizing the Cost Function

Fitting the model to the data involves evaluating the cost function for different combinations of parameters with the goal of finding the combination that minimizes the cost function. Minimizing the cost function is necessary because it allows the researcher to identify the parameter values that provide the best possible correspondence between the model predictions and the data (as measured, in this case, by the likelihood of the model and its parameters given the data). As described in the introduction to this chapter, when model-data correspondence is maximized, the researcher can attribute any discrepancy between model and the data to the inadequacy of the model itself (as opposed to inadequacy in the parameter values used to generate model predictions). This makes theories easier to evaluate.

To minimize the cost function, the researcher must first specify the free parameters that are to be adjusted when attempting to bring the model predictions in line with the data. Revisiting the metaphor of the analogue radio, free parameters are the tuning knobs that can be changed in order to pick up a better signal. This process of adjusting the values of free parameters until a combination is found that minimizes the cost function enables free parameters to be estimated from the data. As illustrated above, our implementation of the MGPM has three free parameters: time sensitivity, discount rate, and threshold.

The number of free parameters bears upon the parsimony of the model. In general, as the number of free parameters increases, the model becomes more flexible. This can make the model difficult to falsify because it increases the range of data patterns for which the model can account. Increasing model flexibility also increases the risk that the model provides such a close fit to one dataset that it fails to adequately capture new observations (this problem is referred to as *overfitting* or *fitting to noise*). This limits the model's generality. Thus, there is a tradeoff between giving the model the best chance of accounting for the data (by maximizing the number of free parameters) and maintaining a parsimonious model (by limiting the number of free parameters). We return to this issue in the *Model Comparison* section.

Parameters that are held constant instead of being estimated are commonly referred to as *fixed* parameters. Fixing parameters helps to keep the number of free parameters to a minimum. The researcher may choose to fix parameters that have values that are known a priori or are not of theoretical interest. For example, Zhou, Vancouver, and Wang's (2018) computational model of leadership goal striving included parameters that controlled the rate at which leaders and subordinates could make progress toward the goal. These parameters were not focal to the hypotheses being tested and were therefore fixed for the purposes of parsimony. In our implementation of the MGPM, we fix the expected lag and quality parameters because these parameters have known values that are determined by the experimental task. We also fix the goal importance parameter because the goals that participants pursue in this experiment are identical in value and therefore would not be expected to differ in importance.

The process of testing different free parameter values is conceptually simple, but often practically challenging. The simplest way to do this is to conduct an exhaustive search of the parameter space. Consider an example where a model has a single free parameter. In this case, the modeller might simply test a series of parameter values that span the range of values that the parameter could plausibly take on. If the modeller expects the parameter value to fall between 0 and 10, they might test all values along this interval increments of 0.01. This would require testing 1001 possible parameter values, which would take only a few seconds on a most computers.

It is often the case, however, that a model will have more than one free parameter. In this scenario, an exhaustive search requires testing every possible *combination* of parameter values (a technique known as "grid search"). This approach becomes less feasible as the number of free parameter combinations that need to be tested increases. For example, if a second parameter with the same range of possible values was added to the model above, over 1 million unique combinations of parameter values would need to be tested. If a third parameter was added, the number of possible combinations would be over 1 billion! Such an analysis would be far too computationally intensive to be practical. A grid search is therefore only feasible when the number of unique combinations of parameter values is modest.

The range of possible parameter values and the increments between parameter values tested also influence the number of combinations of parameters that need to be assessed. If a grid search is used, the range of each parameter should be selected so that it covers the full space of plausible parameter values. The upper and lower bounds on the range are typically guided by theory. For example, it might not make theoretical sense for certain parameters to take on negative values or there may be certain parameter values beyond which further increases (or decreases) in the parameter value have negligible effects on the model's predictions. In the latter case, parameter values above (or below) the identified value carry the same theoretical interpretation, so values beyond this point may not need to be considered.

The increments in parameter values should be chosen so that they are as small as possible without rendering the grid search infeasible. Smaller increments are better because they allow for more fine-grained parameter estimates. In general, the increments should be small enough that transitioning between consecutive parameter values results in only a minor change in the model's predictions. If this is not the case, the researcher runs the risk of "missing" parameter values that result in far superior model-data correspondence. This possibility can make a discrepancy between the data and the model's predictions under the estimated parameter values difficult to interpret for reasons outlined earlier in this chapter.

Gee, Neal, and Vancouver (2018) used a grid search to estimate two parameters from their computational model of goal revision. However, the plausible ranges of these parameters were narrow (one parameter was bounded between 0 and 1 and the other between -3 and 3). Furthermore, the increment between parameter values considered was relatively course (0.1). As a result, only 671 unique combinations of parameter values needed to be tested. In most cases where more than one parameter needs to be estimated, however, a grid search will be too computationally intensive to estimate parameters efficiently.

Fortunately, there are far more efficient ways to estimate parameters than by exhaustively searching the parameter space. Modelers will typically make use of algorithms that find the values that minimize the cost function without having to test all possible combinations of parameters. Broadly, these algorithms work by first evaluating the cost function under some fixed set of initial parameter values. The algorithms then repeatedly modify the parameter values in such a way that the value of the cost function decreases with each iteration, until some stopping criterion is met (Farrell & Lewandowsky, 2018). This stopping criterion may involve stopping after a certain number of iterations or when the change in the cost function between iterations drops below some threshold value. Most programs have default stopping criteria that are sufficient in the majority of cases. Once the algorithm ceases, the resulting parameter values constitute the best estimates.

The code below implements an algorithm that estimates the parameters by finding the combination of parameters that minimize the cost function. The function that runs the algorithm is a

built-in function in R called `optim` (line 5). This function has three arguments. The first argument, `par`, is a vector of initial values for the free parameters. This vector, called `starting_values` is assigned on line 2. The vector has three elements, one for each free parameter in our model. The time sensitivity, discount rate, and threshold parameters occupy the first, second, and third elements respectively.

```
1 #specify initial parameter values
2 starting_values = c(1,1,1)
3
4 #run parameter estimation algorithm
5 result = optim(par = starting_values,
6                fn = MGPM_likelihood,
7                data = data)
```

The second argument, `fn`, is the name of the cost function that is be minimized. The third argument, `data`, is the data required to evaluate the cost function. The `optim` function requires the function to be minimized to have two input arguments. The first argument to the function being minimized must be the vector of parameters that are adjusted in order to minimize the function. The second argument to the function being minimized is the data required to evaluate the function. The `optim` function also assumes that the function being minimized returns a single numeric value. In our case, the function to be minimized is the `MGPM_likelihood` function. We've structured the `MGPM_likelihood` function so that it meets the requirements of the `optim` function. The first argument to the `MGPM_likelihood` function contains the values of the three parameters that are updated as the model is fit to the data. The second argument `MGPM_likelihood` function contains the data needed to generate the model predictions.

Figure 3 shows how the `optim` function works. The function repeatedly evaluates the `MGPM_likelihood` function, each time, adjusting the values for the `params` argument and examining the corresponding change in negative log-likelihood. This process continues until the algorithm implemented by the `optim` function reaches a point where no further reductions in the negative log-likelihood can be achieved. This process is referred to as function *optimization* (hence the name "optim").

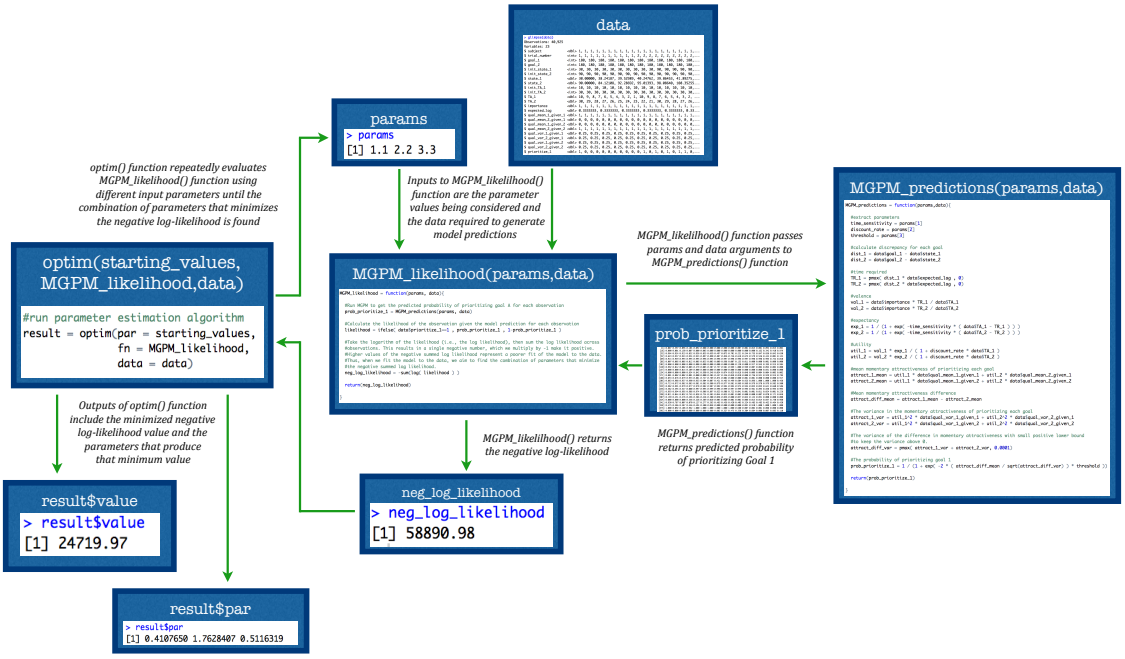

Figure 3. Structure of the `optim` function. The input arguments are the starting parameter values, the name of the function to be minimized, and the dataset containing the variables required to generate model predictions. `optim` repeatedly evaluates the `MGPM_likelihood` function until the combination of parameters is found that minimizes the function output.

The code below shows the output from the `optim` function, which is stored in the `result` object. As can be seen, the `optim` function returns a list with five elements. The `par` element contains the parameter values that minimize the function, or in other words, the parameter estimates. As can be seen, the estimates for the time sensitivity, discount rate, and threshold parameters were approximately 0.41, 1.76, and 0.51 respectively. The `value` element contains the minimum value of the function. In our implementation, this is the negative log-likelihood of the data under the estimated parameter values. The `counts` element contains information regarding how many times the function needed to be evaluated before the minimum value was found. The `convergence` element confirms that the algorithm converged, with zero indicating successful convergence. Finally, the `message` element contains any other information that was returned by the `optim` function.

```
$par
[1] 0.4107650 1.7628407 0.5116319

$value
[1] 24719.97

$counts
function gradient
     154       NA

$convergence
[1] 0

$message
NULL
```

At this point, the reader may be wondering how much the results of this analysis depend on the starting values for the three parameters. This is an important question to consider, because in some cases, changing the starting values will result in different parameter estimates. This happens because the relationship between the parameters and the output of the cost function can be complex. An an example, Figure 4 shows the relationship between the time sensitivity parameter and the value of the cost function (assuming the discount rate and threshold parameters are fixed to their estimated values). As can be seen, the value of the cost function is high when time sensitivity is 0 and then rapidly decreases before reaching its minimum around 0.4. As time sensitivity increases beyond this value, the output of the cost function begins to increase. This increase is rapid at first, but then levels off as time sensitivity reaches 8 or so.

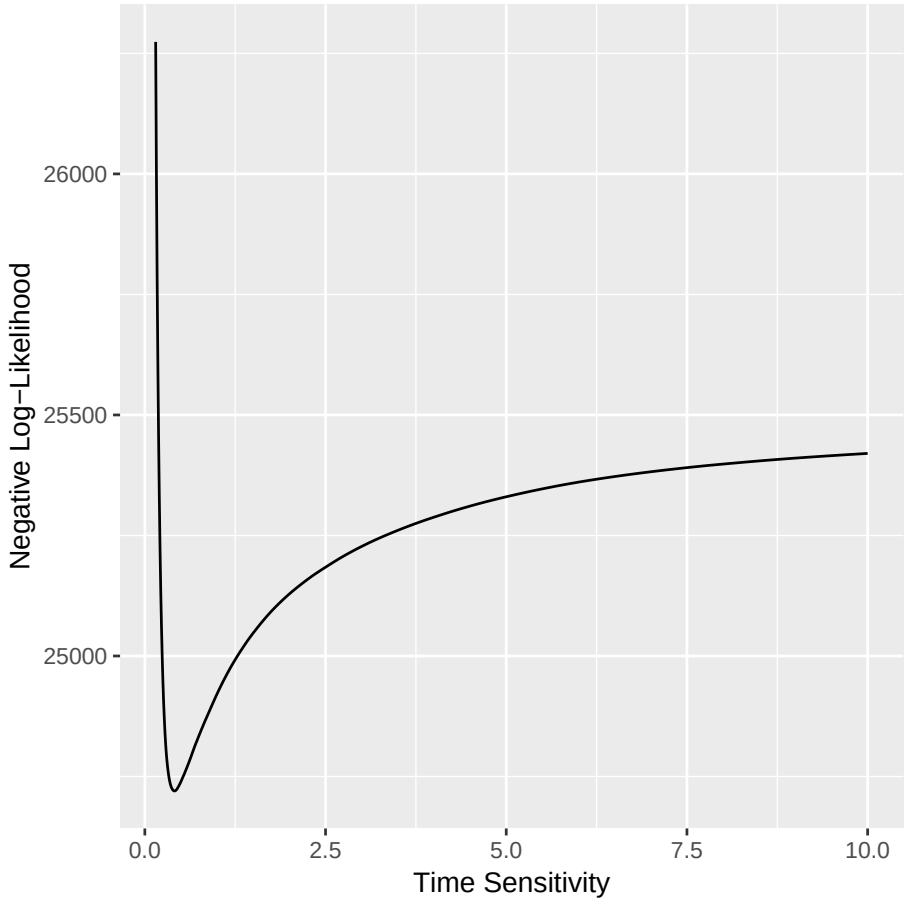

Figure 4. The relationship between time sensitivity and the negative log-likelihood of the data under the model. This figure was constructed by fixing the discount rate and threshold to their estimated values (1.76 and 0.51 respectively) and examining the result of the `MGPM_likelihood` function for a range of time sensitivity values.

Recall that the starting value for the time sensitivity parameter was one. At this value, any small change in the parameter value will have a relatively pronounced effect on the output of the cost function. Small increases will amplify the output of the cost function, whereas small decreases will reduce the output. Because of this, it's easy for the algorithm to identify the changes in the

parameter that are necessary to move towards the function's minimum. However, suppose we used a starting value of 10 for the time sensitivity parameter. In this region, changes in time sensitivity have little effect on the output of the cost function (as can be seen by the relatively flat nature of the function around that value). Such cases can lead the algorithm to prematurely "conclude" that the minimum of the function has been found because it cannot find any neighbouring values that produce meaningful decreases in the output of the function. Indeed, this is precisely what happens if we change the starting values for our three parameters to 10. When working with more complex models, it is also possible for more than one "valley" to exist in the parameter space. In such cases, the parameter estimates are especially sensitive to the starting values that are chosen because there are multiple points at which all neighboring parameter values produce an increase in the output of the cost function. The parameter estimates will therefore depend on which "valley" is closest to the starting values. This issue is referred to as the *problem of local minima*.

The parameters that produce the closest correspondence between model and data are those for which the value of the cost function is lowest. The lowest point of the cost function is referred to as the *global minimum*. As illustrated above, finding the global minimum may be challenging because many algorithms can converge on parameter values that produce the lowest value of the cost function among parameter values in the immediate vicinity, but may not represent the lowest values possible (such values are referred to as *local minima*). There are several steps the modeller can take to ensure that the parameter estimates returned by the algorithm reflect the true minimum value of the cost function (referred to as the *global* minimum). It is generally advisable to run the analysis using different starting points and examining the output under each set. It is not uncommon for different sets of starting values to yield different parameter estimates. However, typically the different sets of parameter estimates will produce different "minimum" values for the cost function. For example, when we run the algorithm using starting values of 10 for all three parameters, the resulting parameter estimates are 16.49, 1.72, and 0.42 for time sensitivity, discount rate, and threshold respectively. However, the value of the cost function under this combination of parameter values is 25327.92, which is higher than the value under the original parameter estimates. Thus, we know that the parameter values that were returned when we set the starting values to 10 do not represent the best estimates because there is another combination of values under which the model and data are more closely aligned.

One useful way to systematically test different starting values is to use a grid search method. We noted above that an advantage of the grid search is that it can systematically test every combination of parameters, but that it has the limitation of only being practical when the number of combinations is modest. However, one can combine the virtues of the grid search with the those of the optimization algorithm by using a grid search to systematically vary the starting values that are used for each run of the algorithm. For example, we might construct a set of all combinations of starting values in which each of our three parameters takes on values of 0, 5, or 10 (alternatively, we could randomly sample the sets of starting values). We would then run the optimization algorithm once for each set of starting values and record the minimum value of the cost function returned each time. We would then identify the starting values that produced the minimum value that was lowest. When the algorithm is run using these starting values, the parameter estimates returned would constitute the best estimates, and the minimum value identified would represent our best guess at the global minimum of the cost function.

Model Comparison

Demonstrating that the data correspond with the model's predictions is an important first step in testing a computational model. We can infer on the basis of model-data correspondence that the model is a possible explanation for the empirical phenomena under investigation. However, we cannot rule the rule out the possibility that there are other viable mechanisms. In other words, model predictions that map accurately onto the data provide evidence that the assumptions that form the model are sufficient to account for the empirical phenomenon. However, they do not speak to the necessity of these assumptions. We therefore encourage modellers to go beyond simply showing that their model can account for the data by comparing their model against other plausible models.

As with more commonly used methodology such as structural equation modelling, computational model comparison generally involves comparing a series of models on the basis their ability to parsimoniously account for the data in order determine which of a set of models provides the best explanation. The comparison may be between a hypothesized model and a set of variants that represent competing explanations or between a series of models that are equally plausible theoretically. There are several criteria on which the quality of the model's explanation is evaluated (Myung & Pitt, 1997). The first criterion is the alignment between the model predictions and the data, which in structural equation modelling might be quantified using indices such as the CFI, TLI, RMSEA, or the value of the log likelihood function under the estimated parameter values. A second criterion is parsimony, which in structural equation modelling, as well as many applications of computational models, is often measured as the number of free parameters in the model. All else being equal, a simpler model (i.e., a model with fewer free parameters) should be preferred. This is because a simpler model can account for a narrower range of possible patterns of data. Thus, the simpler the model, the more convincing it is when the model aligns with the data. A third criterion is generality, which refers to the extent to which the model can be applied across experimental tasks or settings. A final criterion is plausibility, which reflects the reasonableness of the model's assumptions and their compatibility with established findings.

Of these four criteria, model-data alignment and parsimony are the most readily quantifiable, and are therefore the two primary dimensions considered in most model comparisons. Together, these two dimensions determine the degree to which the model satisfies Occam's razor, which implies that models should be as simple as possible while still being able to account for the data. The trade-off between fit and parsimony can be quantified in a number of ways. One method is the likelihood ratio test, which examines whether the addition of a free parameter produces enough of an increase in the log-likelihood (i.e., model-data alignment) to warrant the added complexity. However, this test only applies to comparisons between nested models. Two more generally applicable indices are the Akaike information criterion (AIC Akaike, 1973) and the Bayesian information criterion (BIC Schwarz, 1978), which combine assessments of fit and parsimony into a single value. These indices can be applied to nested or non-nested models (we elaborate on this issue in the "Extensions" section). They are interpreted such that lower indices indicate a better balance between fit and parsimony, and therefore a better explanation.

In this tutorial, we use the BIC to compare different versions of the MGPM. The BIC is calculated according to the following equation:

$$BIC = -2 \cdot ln(L) + ln(n) \cdot k. \tag{11}$$

The first term in Equation 11 assesses the model-data alignment, as measured by the log-likelihood

under the estimated parameters ($ln(L)$). As can be seen, the better the model-data alignment (the lower the log-likelihood), the lower the BIC. The second term in Equation 11 penalizes the complexity of the model. As can be seen, the BIC increases with the number of free parameters (k) and the log of the number of observations (n). This means that the model with more free parameters will incur a greater penalty for complexity, with the difference between penalties being more pronounced when there are more observations.

The code below calculates the BIC for the MGPM. On line 1, the number of observations is obtained by using the `nrow` function which counts the number of rows in the `data` object. On line 2, the number of free parameters is obtained by using the `length` function to count the number of estimated parameters stored in the `result` object. On line 3, the log-likelihood under the estimated parameters is obtained by taking the negative of the minimum cost function value returned by the `optim` function. Recall that the output of the cost function was the negative log-likelihood of the data under the model. Here, we simply convert that value back to the positive log-likelihood (which, confusingly, is a negative number) by taking the negative. On line 5, the BIC is calculated according to Equation 11.

```
1  n = nrow(data)          #number of observations
2  k = length(result$par)  #number of estimated parameters
3  lnL = -result$value     #log-likelihood of the model under estimated parameters
4
5  BIC = - 2*lnL + log(n)*k
```

We compare the implementation of the MGPM described above to two alternative models. The first alternative tests the necessity of the assumption that the expected utility of a goal is subject to temporal discounting. We do this by comparing the version of the MGPM introduced above to a simpler model in which the discount rate parameter is fixed to 0. This constraint turns off the temporal discounting mechanism within the model. This model has only two free parameters: time sensitivity and threshold. The second alternative model tests the sufficiency of the assumption that the perceived time required to progress toward the goal by a single unit is equal to the actual time required in the experimental task. We do this by testing a more complex alternative in which the expected lag parameter is treated as a free parameter. This model has four free parameters: time sensitivity, discount rate, threshold, and expected lag.

Figure 5 shows the predictions of each model superimposed against the choices observed. As can be seen, in the condition where the experimental goal started off a short or very short distance from the goal (30-60 cm), longer deadlines resulted in a weaker tendency to prioritize the experimental goal. In the condition where the experimental goal started off a moderate distance from the goal (90 cm), there was a non-monotonic effect of deadline, such that the tendency to prioritize the experimental goal was highest when the deadline was moderate. When the experimental goal started off farther from the goal (120-150 cm), longer deadlines resulted in a stronger tendency to prioritize the experimental goal. The model predictions suggest the 3- and 4-parameter versions of the model both do a reasonable job of capturing these trends, whereas the alignment between the data and the 2-parameter model is poorer.

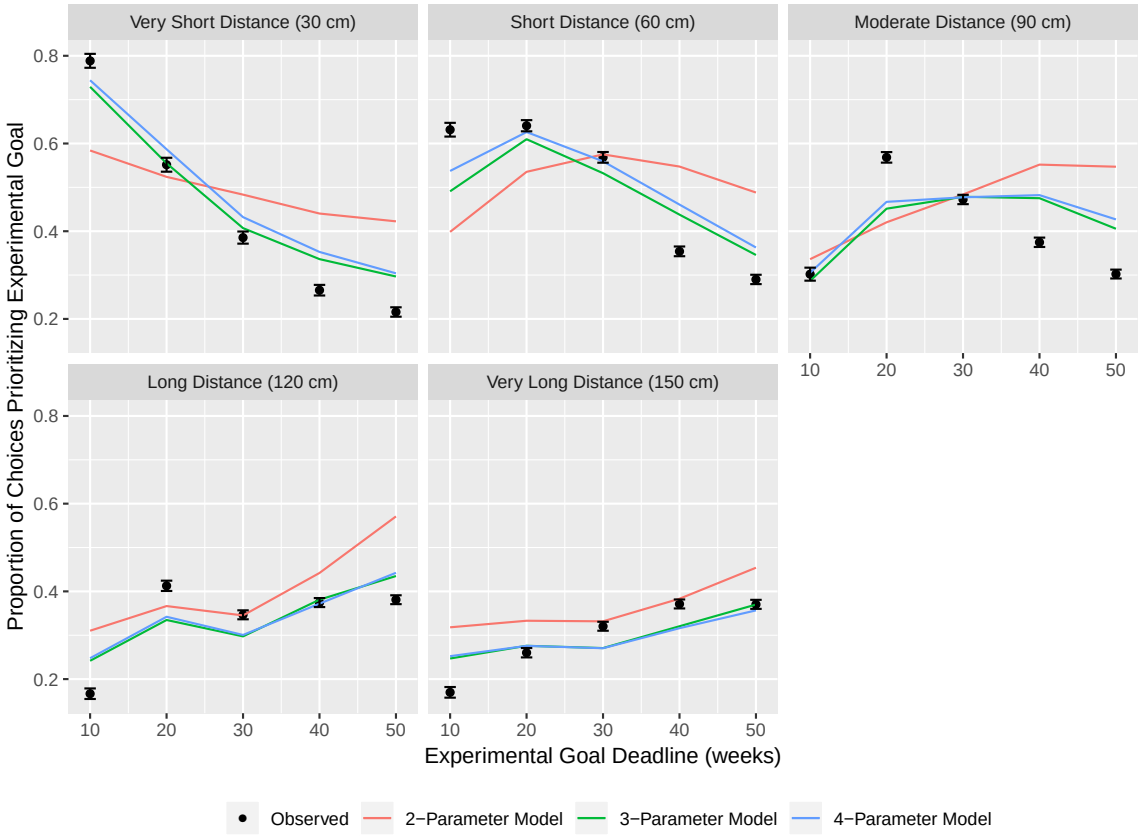

Figure 5. The fit of the three models superimposed over the data observed in the experiment. The black dots represent the observed proportion of decision prioritizing Goal 1 in each experimental condition (with error bars representing the standard error of the mean). The lines represent the average probability predicted by the models.

At this point, we can rule out the 2-parameter model based on its inferior ability to account for the data. The question to be answered is whether the increase in model-data alignment that is obtained from transitioning from the 3-parameter model to the 4-parameter model is worth the added complexity. The BICs suggest that the answer to this question is yes. The BIC for the three parameter model is 49472, whereas the the BIC for the four parameter model is 49328 (the BIC for the 2-parameter model is 51727). On the basis of this result, one could argue that the 4-parameter model provides a better trade-off between fit and parsimony. On the other hand, the two models do not make qualitatively different predictions, and it appears that the increase in model-data alignment achieved by the extra parameter is rather modest. This argument might be counted as a strike against the 4-parameter model.

One other question that is worth considering is to what extent the estimate of expected lag in the 4-parameter model deviates from the value to which this parameter was fixed in the 3-parameter model. Recall that expected lag had been fixed to a value of 0.33, because this represents the average amount of time taken to reduce the distance to each goal by one unit in the task. The estimated value of this parameter in the 4-parameter model is 0.35. This estimate suggests that people may slightly overestimate the time required to progress toward the goal. However, is the difference between the

perceived and actual lags different enough to warrant treating expected lag as a free parameter? The point we are attempting to illustrate is that there are several considerations that need to be made when comparing models. Although quantitative model comparison metrics are informative, any decisions that are made need to be informed by the broader theoretical context.

It is important to note that the approach described above is not the only method for comparing models. One alternative approach is cross-validation. Cross validation involves fitting models to a subset of the data and then evaluating the models on the basis of their ability to predict the remaining data. For example, Kennedy and McComb (2014) estimated the parameters of their model of team process shifts based on 75% of the data and then tested their model's ability to account for the remaining data by generating predictions using their parameter estimates. Cross validation handles the issue of parsimony in a different way to the approach described above. Rather than imposing an explicit penalty for model complexity, as is the case with information criteria such as the BIC, cross validation takes advantage of the tendency for overly complex or flexible models to mistake noise in the data for systematic variance (i.e., overfitting). When this happens, the model's predictions are less able to generalize to new data even if they provide a better fit to the data to which the model was initially fit. As such, cross-validation manages the trade-off between model-data correspondence between and complexity by directly assessing the generality of the model's predictions. The Bayesian framework also offers alternative approaches model comparison, which we address in the "Extensions" section.

Parameter Recovery

The final issue we wish to discuss pertains to the reliability of the parameter estimates. As discussed in the opening section, modellers will often wish to interpret the parameter estimates to make inferences about components of the process being investigated. For example, one may wish to examine whether a particular sub-process differs across individuals or experimental conditions (e.g., Ballard, Yeo, Loft, et al., 2016; Gee et al., 2018). However, the fact that the optimization algorithm has identified the"best" parameter estimates does not mean those parameter estimates are practically meaningful. It is possible that there are several different combinations of parameters that produce virtually identical predictions. In this scenario, the parameter estimates would be unreliable because there might be many different combinations of parameters that generate predictions that align with the data.

If we wish to obtain reliable parameter estimates, each combination of parameters must produce unique predictions. When this is the case, the parameters are said to be *identifiable*, because there will only ever be a single combination of parameters that best explains a given pattern of data. This means that parameters estimated based on the data can be interpreted meaningfully. One way to assess the reliability of a model's parameter estimates is to conduct a parameter recovery analysis. A parameter recovery analysis involves simulating data under known parameter values, fitting the model to the simulated data, and then examining how closely the parameter estimates match the values that were used to generate the data. Parameters are said to be *recovered* when the estimates match the data-generating values. Good parameter recovery is an indication that the parameters can be reliably estimated. Importantly, parameter recovery depends not only on the model but also on the dataset to which the model is being fit. For example, larger sample sizes produce more reliable parameter estimates because there is more information to constrain the estimate. Thus, it is important that the parameter recovery analysis be conducted using a simulated dataset that is identical to the empirical dataset to which the model will eventually be fit. That is, the simulated data should

contain the same number of observations as the empirical data, and the observed variables that are needed to generate the predictions (e.g., time available, goal level, etc) should have the same values.

Figure 6 shows the results of a parameter recovery analysis that was conducted on the 3-parameter version of the MGPM. This analysis was conducted by randomly sampling 100 combinations of the time sensitivity, discount rate, and threshold parameters. For each combination of parameters, we ran the model to determine the probability of prioritizing Goal 1 for each observation in the dataset. We then repeated the following procedure 100 times: 1) Simulate each choice based on the probabilities predicted by the model; 2) Fit the model to the simulated choices and; 3) record the parameter estimates. The figure is constructed such that the data-generating parameter values are shown on the x-axis. The recovered values are shown on the y-axis, with the points representing the mean of the recovered values across the 100 repetitions and the error bars representing the 2.5 and 97.5 percentiles of the recovered values.

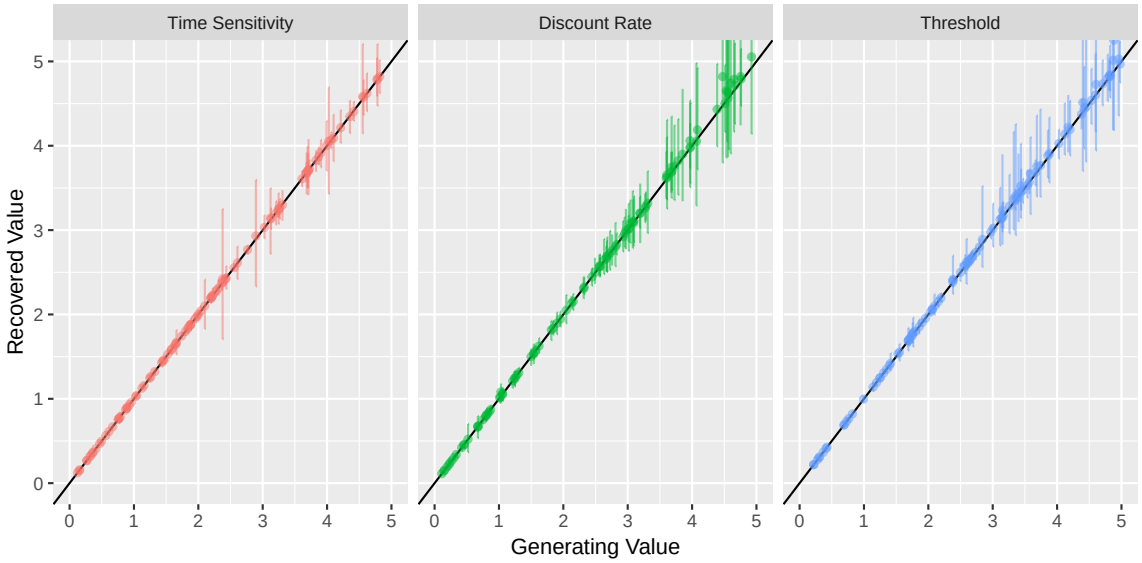

Figure 6. Results of the parameter recovery analysis conducted on the 3-parameter version of the MGPM. Parameters were randomly sampled independently from a uniform distribution on the interval [0.1,5].

The diagonal line represents perfect correspondence between data-generating and recovered parameter values. A discrepancy between the mean recovered value and data-generating value indicates bias in the parameter estimate. Points that fall above the diagonal indicate that the model systematically over estimates that parameter. Points that fall below the diagonal indicate that the parameter is underestimated by the model. The figure does not suggest that there is systematic bias in any of the parameter estimates. The width of the error bars reflects the precision in the estimate. As can be seen, the parameter estimates become less precise as data-generating parameter value increases. This is to be expected, because changes in the parameter value have less of an impact on the model predictions when the parameter value is larger. For example, the model predictions change more drastically when transitioning between time sensitivity values of 0.4 and 0.5 than when transitioning between values of 4.4 and 4.5 (this can be seen by the curvature of the log-likelihood function in Figure 4). Overall, the results of this analysis suggest very good parameter recovery,

which means the modeller can trust the parameter estimates obtained when the model is fit to this dataset.

Example Results Section

Model fitting was conducted in R (R Core Team, 2018) using a maximum likelihood approach. Our hypothesized model contained three free estimated parameters: time sensitivity (γ), discount rate (Γ), and the threshold (θ). A single set of parameters was estimated for the entire sample, meaning that each parameter was assumed to take on the same value for all participants. Because the two goals in this experiment were equally important, we fixed the gain parameter (κ_i) for each goal to one so that this parameter would be equal for the two goals. We fixed the expected lag parameter (α) to $\frac{1}{3}$ because this is the average amount of time needed to reduce the distance to each goal by one unit in the experiment. To be consistent with the experimental environment, the quality (Q_{ij}) of an action's consequences was defined as a normally distributed, random variable. The quality distribution had a mean of one for the goal that was prioritized and zero for the goal that was not prioritized, with both distributions having a variance of 0.25. The reliability of the parameters estimated by this model was confirmed via a parameter recovery analysis (see Figure 6).

The predicted probability of Goal 1 being prioritized was calculated according to Equations 1-10. As the prioritization decision was a binary variable (1 = Goal 1 prioritized, 0 = Goal 2 prioritized), the likelihood of the observed prioritization decision under the model was assumed to be Bernoulli distributed. The parameters were estimated using the optimization algorithm implemented by the *optim* function in R, which minimized the negative summed log likelihood of the data under the model. In order to ensure our parameter estimates were robust, we ran the optimization algorithm multiple times using different starting values for each run and selecting the parameter estimates that produced the lowest minimized negative summed log likelihood across all runs.

In order to assess the assumptions of our hypothesized model, we compared the hypothesized model to two alternatives. The purpose of the first alternative was to test the necessity of the assumption that the expected utility of prioritizing a goal is subject to temporal discounting. In this model, the discount rate parameter was fixed to zero, which eliminates the effect of temporal discounting. The first alternative therefore had two free parameters (time sensitivity and threshold). The purpose of the second alternative was to test the sufficiently of the assumption that people have an accurate perception of the time required to reduce the distance to each goal by one unit. In this model, the expected lag parameter was treated as a free parameter and estimated from the data. The second alternative therefore had four free parameters. The alternative models were otherwise identical to the hypothesized model. These models were fit using the same protocol described above. The ability of the each model to adequately characterize the data was quantified using the BIC (Schwarz, 1978), which is interpreted such that the model with the lowest value is said to provide the best trade-off between model-data correspondence and parsimony.

The predictions of the three models given their estimated parameters are presented in Figure 5. As can be seen the hypothesized and first alternative models (shown by the green and blue lines respectively) are able to accurately reproduce the empirical trends in the data. The second alternative model performs noticeably worse. The BICs were 49472, 49328, and 51727 for the hypothesized, first alternative, and second alternative models respectively. The BICs suggest that the first alternative model provides a better explanation of the data than the hypothesized model. This result suggests that the assumption that participants' perception of the time required to reduce the distance to each goal by one unit (i.e., expected lag) is equal to the objective time required in the

experiment may not be sufficient for adequately characterizing the empirical trends. However, the difference in BICs between the hypothesized model and first alternative is relatively small and the predictions of the two models shown in Figure 5 are very similar. This suggests that any advantage that comes from treating expected lag as a free parameter in the first alternative model is very small. The fact that the second alternative is inferior to the other two models suggests that the assumption that the expected utility of a goal is subject to temporal discounting is necessary to account for the empirical trends.

As the four-parameter model provided the best explanation of the data according to the BIC, we report the parameter estimates from this model. The values of the time sensitivity, discount rate, threshold, and expected lag parameters were 0.34, 1.47, 0.51, and 0.35 respectively. Note that the estimate of the expected lag parameter (0.35) differs only slightly from the value to which this parameter was fixed in the hypothesized model (0.33). This reinforces the conclusion that the first alternative offers only a slight improvement over the hypothesized model. The values of the time sensitivity, discount rate, and threshold parameters were comparable to their estimated values in previous research (see Ballard, Vancouver, & Neal, 2018; Ballard, Yeo, Loft, et al., 2016)[3].

Extensions

The method we have presented in the preceding sections is a general approach that can be used to test theories of many different organizational phenomena. Although we have demonstrated this approach using a model of multiple-goal pursuit, it can just as easily be applied to understand phenomena such as leadership (e.g., Vancouver, Wang, & Li, 2018), fatigue (Walsh, Gunzelmann, & Van Dongen, 2017, e.g.,), socialization (e.g., Vancouver, Tamanini, & Yoder, 2010), time pressure (Palada, Neal, Tay, & Heathcote, 2018), scheduling (e.g., Hannah & Neal, 2014), stress and coping (e.g., Vancouver & Weinhardt, 2012), among others.

The method presented above can extended in several ways to help researchers answer more complex research questions. One extension involves estimating unique parameters for different individuals or experimental conditions. In this tutorial, we assumed that every participant was characterized by the same set of parameters. However, there is often reason to estimate a unique set of parameters for each participant. For example, the researcher may wish to examine whether a certain trait variable explains variance in model parameters. In this case, the model fitting procedure described above needs to be conducted for each participant separately.

Modellers might also wish to examine whether certain parameters vary across experimental conditions (e.g., Ballard, Sewell, Cosgrove, & Neal, 2019). To do this, the researcher would fit the model separately to the data from each condition. Note, however, that fitting to individual participants or conditions will mean that there is less data to inform each parameter estimate. This can result in less precise parameter estimates. So it is important that a parameter recovery analysis is conducted to examine the reliability of parameters estimated using smaller subsets of the dataset.

[3] In cases where separate parameters are estimated for each individual, the reader is encouraged to report on the distribution of the estimated parameters (e.g., the mean and SD of each parameter. In such cases, it can also be informative to compare the models at the level of the participant, for example, by reporting the number of participants whose data were best explained by each model

Other Types of Models

In the example model comparison used above, the three models were identical in structure, but differed in terms of which parameters were treated as free parameters and estimated from the data and which parameters were fixed. In such cases where one can transition between models by freeing or fixing a single parameter, the models are referred to as nested. It is important to note, however, that the approach we demonstrated for comparing models also applies to non-nested models that have different functional forms. For example, we could use this same approach to test an alternative variant of the MGPM in which expected utility is proportional to the sum of valence and expectancy as opposed to their product. This approach also generalizes to models that include logical if-then statements, rather than just mathematical equations.

The models used in our tutorial contain likelihood functions that can be derived analytically. In other words, the likelihood of the data given the model's prediction can be calculated based on a set of equations. This is not the case for all models. For example, some models might have random components (i.e., noise) with effects that cannot be expressed in equation form. For example, Grand (2017) examined the effects of stereotype threat on organizational performance using a computational model in which employee turnover was governed by a random process. Models might also involve the dynamic interaction between different agents (i.e., agent-based models) with downstream consequences that cannot be predicted analytically. For example, Tarakci, Greer, and Groenen (2016) used this type of model to understand the effects of power disparity on group performance. (Mäs, Flache, Takács, & Jehn, 2013) developed such a model to examine the role of team diversity on opinion polarization. These types of models need to be simulated in order to generate predictions.

In theory, these so-called "simulation" models can also be analyzed using the approach described above. However, doing so is often extremely computationally intensive. For example, models with random components typically need to be simulated thousands of times in order to generate reliable predictions that are not contaminated by noise. This means that the time required to fit such a model to data may be thousands of times longer than a similar model that can be analytically derived. Models with multiple interacting agents also typically have longer run times because the number of operations required to generate predictions usually scales with the number of agents in the model. Such models can take much longer to generate predictions even if they do not contain random components. Due to these practical challenges, simulation models are often not fit to data in the way that we demonstrated above. However, recent advances in computing combined with the development of more efficient parameter estimation algorithms has meant that fitting simulation models to data is becoming easier (Evans, 2019; Holmes, 2015).

Bayesian Parameter Estimation

We have limited our focus to maximum likelihood methods, which are a frequentest approach to parameter estimation. An alternative to this approach is Bayesian parameter estimation. Bayesian parameter estimation has become widely-used as a method for fitting computational models to data in the cognition literature, and is starting to be used by researchers in industrial and organizational (see Ballard, Farrell, & Neal, 2018; Ballard, Palada, et al., 2019; Ballard, Vancouver, & Neal, 2018; Neal, A., Gee, P., Ballard, T., Vancouver, J. B., Yeo, 2019). The Bayesian approach differs to the maximum likelihood approach in some important ways (Kruschke, Aguinis, & Joo, 2012). First, although the Bayesian approach involves evaluating parameters based on their likelihood given the

data, this approach also incorporates prior beliefs regarding the plausibility of different parameter values into the analysis. The potential to incorporate prior beliefs allows the researcher to take into account results from previous studies, theoretical assumptions regarding the parameter value, or information about the plausible parameter values for other participants. The latter makes it easy to implement hierarchical models which capture variation in parameters between individuals while simultaneously estimating parameters at the group (or condition) level.

Second, whereas the maximum likelihood approach uses optimization methods to find the single best set of parameter estimates given the data, the Bayesian framework quantifies the uncertainty in a parameter estimate. This is done by estimating the *posterior distribution* on each parameter, which represents the range of plausible values for that parameter given the data and the researcher's prior beliefs. By taking into account uncertainty, the posterior distribution makes it easy to determine whether differences in parameter values (e.g., between conditions or individuals) are meaningful. Overlapping posterior distributions indicate that the researcher can plausibly conclude that there is no difference in the parameters. Posterior distributions that do not overlap suggest a reliable difference between parameter values.

Another point of difference between the maximum likelihood and Bayesian approaches to parameter estimation is how they conceptualize model complexity. As discussed above, maximum likelihood approaches usually operationalize complexity as the number of free parameters in the model. The Bayesian approach takes the number of parameters into account, but also considers two other dimensions of complexity: 1) the functional form of the model and 2) the range of possible values that the model parameters can take on (Lee & Wagenmakers, 2013; Myung & Pitt, 1997). This more general approach to quantifying model complexity can facilitate the comparison of models that may differ in their flexibility for reasons other than a difference in the number of free parameters estimated by each model.

Conclusion

We hope that the tutorial presented in this chapter provides a useful foundation for researchers wishing to develop and test their own computational models. We believe that computational modeling encourages cumulative scientific progress because it enables a more direct mapping between theory and data, which makes theories easier to test, compare, refine, extend, and reject. Fitting models to data is a central part of that process. We are therefore hopeful that the material presented here will help researchers to develop stronger theories that ultimately move the field forward.

Further Reading

Busemeyer, J. & Diederich, A. (2010). *Cognitive Modeling*. Sage, New York, NY.

Farrell, S. & Lewandowsky, S. (2018). *Computational Modeling of Cognition and Behavior*. Cambridge University Press, Cambridge, UK.

Kruschke, J. (2010). *Doing Bayesian Data Analysis: A tutorial with R, JAGS, and Stan*. Academic Press, New York, NY.

Lee, M., & Wagenmakers, E. (2013). *Bayesian cognitive modeling: A practical course*. Cambridge University Press, New York, NY.

Myung, I. J. (2003). Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology, 47*, 90-100.

Glossary

Cost function: A mathematical function that quantifies that model-data correspondence (or perhaps more accurately, the lack of model-data correspondence) in such a way that higher values represent poorer correspondence between the data and predictions of the model.
Fixed parameter: A parameter that is specified before the analysis to a known value.
Free parameter: A parameter with a value that is unknown before the analysis and that is estimated from data.
Global minimum: A combination of parameter values for which the value of the cost function is lowest value possible for the model.
Likelihood: The probability of the data having been observed under the model and the model's parameter values.
Local minimum: A combination of parameter values for which the value of the cost function is lower than all combinations involving similar parameter values, but that does not produce the lowest value possible for the model.
Model-data correspondence: The extent to which the predictions of a model align with or match the data (also referred to as model-data alignment, model-data fit, or goodness of fit).
Model fitting: The process of adjusting model parameter values in such a way as to maximize the correspondence between the data and the predictions of the model (also referred to as fitting a model to data).
Optimization: A method of identifying the parameter values that maximize model-data correspondence that uses an algorithm to iteratively adjust the parameter values based on the value of the cost function produced at each step,
Parameter: A variable that is required by the model to generate predictions. Parameters can be fixed to predefined values or estimated from data.

References

Ainslie, G., & Haslam, N. (1992). Hyperbolic discounting. In G. Loewenstein & J. Elster (Eds.), *Choice over time* (pp. 57–92). New York: Russell Sage Foundation.

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In B. N. Petrov & F. Caski (Eds.), *Proceedings of the second international symposium on information theory* (pp. 267–281). Budapest: Akademiai Kiado. doi: 10.1007/978-1-4612-1694-0_15

Ballard, T., Farrell, S., & Neal, A. (2018). Quantifying the psychological value of goal achievement. *Psychonomic Bulletin and Review*, *25*, 1184–1192. doi: 10.3758/s13423-017-1329-1

Ballard, T., Palada, H., Griffin, M., & Neal, A. (2019). An integrated approach to testing dynamic theory: Using computational models to connect theory, model, and data. *Organizational Research Methods*, 1–34. doi: 10.1177/1094428119881209

Ballard, T., Sewell, D. K., Cosgrove, D., & Neal, A. (2019). Information processing under reward versus under punishment. *Psychological Science*, *30*, 757–764. doi: 10.1177/0956797619835462

Ballard, T., Vancouver, J., & Neal, A. (2018). On the Pursuit of Multiple Goals With Different Deadlines. *Journal of Applied Psychology*. doi: 10.1037/apl0000304

Ballard, T., Yeo, G., Loft, S., Vancouver, J. B., & Neal, A. (2016). An integrative formal model of motivation and decision making: The MGPM. *Journal of Applied Psychology*, *101*, 1240–1265. doi: 10.1037/apl0000121

Ballard, T., Yeo, G., Neal, A., & Farrell, S. (2016). Departures from optimality when pursuing multiple approach or avoidance goals. *Journal of Applied Psychology*, *101*, 1056–1066. doi: 10.1037/apl0000082

Evans, N. J. (2019). A method, framework, and tutorial for efficiently simulating models of decision-making. *Behavior Research Methods*, *51*, 2390–2404. doi: 10.3758/s13428-019-01219-z

Farrell, S., & Lewandowsky, S. (2018). *Computational Modeling of Cognition and Behavior*. Cambridge: Cambridge University Press.

Gee, P., Neal, A., & Vancouver, B. (2018). A formal model of goal revision in approach and avoidance contexts. *Organizational Behavior and Human Decision Processes*, *146*, 51–61. doi: 10.1016/j.obhdp.2018.03.002

Grand, J. A. (2017). An examination of stereotype threat effects on knowledge acquisition in an exploratory learning paradigm. *Journal of Applied Psychology*, *102*, 115–150.

Hannah, S. D., & Neal, A. (2014). On-the-fly scheduling as a manifestation of partial-order planning and dynamic task values. *Human Factors*, *56*, 1093–1112. doi: 10.1177/0018720814525629

Heathcote, A., Brown, S. D., & Wagenmakers, E.-J. (2015). An Introduction to good practices in cognitive modeling. In B. U. Forstmann & E. J. Wagenmakers (Eds.), *An introduction to model-based cognitive neuroscience* (pp. 25–48). New York, US: Springer.

Holmes, W. R. (2015). A practical guide to the Probability Density Approximation (PDA) with improved implementation and error characterization. *Journal of Mathematical Psychology*, *68-69*, 13–24. doi: 10.1016/j.jmp.2015.08.006

Kanfer, R., & Ackerman, P. L. (1989). Motivation and cognitive abilities: An integrative/aptitude treatment interaction approach to skill acquisition. *Journal of Applied Psychology*, *74*, 657–690. doi: 10.1037//0021-9010.74.4.657

Kennedy, D. M., & McComb, S. A. (2014). When teams shift among processes: Insights from simulation and optimization. *Journal of Applied Psychology*, *99*, 784–815. doi: 10.1037/a0037339

Kruschke, J. K., Aguinis, H., & Joo, H. (2012). The time has come: Bayesian methods for data analysis in the organizational sciences. *Organizational Research Methods*, *15*, 722–752. doi: 10.1177/1094428112457829

Lee, M. D., & Wagenmakers, E.-J. (2013). *Bayesian cognitive modeling: A practical course*. New York, NY: Cambridge University Press.

Mäs, M., Flache, A., Takács, K., & Jehn, K. A. (2013). In the short term we divide, in the long term we unite: Demographic crisscrossing and the effects of faultlines on subgroup polarization. *Organization Science*, *24*(3), 716–736. doi: 10.1287/orsc.1120.0767

Meehl, P. E. (1978). Theoretical risks and tabular asterisks: Sir Karl, Sir Ronald, and the slow progress of soft psychology. *Journal of Consulting and Clinical Psychology*, *46*, 806–834. doi: 10.1136/jnnp.65.4.554

Myung, I. J. (2003). Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, *47*(1), 90–100. doi: 10.1016/S0022-2496(02)00028-7

Myung, I. J., & Pitt, M. A. (1997). Applying Occam's razor in modeling cognition: A Bayesian approach. *Psychonomic Bulletin & Review*, *4*, 79–95.

Neal, A., Ballard, T., & Vancouver, J. B. (2017). Dynamic self-regulation and multiple-goal pursuit. *Annual Review of Organizational Psychology and Organizational Behavior*, *4*, 410–423. doi: https://doi.org/10.1146/annurev-orgpsych-032516-113156

Neal, A., Gee, P., Ballard, T., Vancouver, J. B., Yeo, G. (2019). The Dynamics of Affect During Approach and Avoidance Goal Pursuit. *Manuscript under review*.

Oberauer, K., & Lewandowsky, S. (2019). Addressing the theory crisis in psychology. *Psychonomic Bulletin & Review*, 1596–1618. doi: 10.3758/s13423-019-01645-2

Palada, H., Neal, A., Tay, R., & Heathcote, A. (2018). Understanding the causes of adapting, and failing to adapt, to time pressure in a complex multistimulus environment. *Journal of Experimental Psychology: Applied*, *24*, 380–399. doi: 10.1037/xap0000176

R Core Team. (2018). *R: A language and environment for statistical computing.* Vienna, Austria: R Founda-

tion for Statistical Computing.

Schmidt, A. M., & DeShon, R. P. (2007). What to do? The effects of discrepancies, incentives, and time on dynamic goal prioritization. *Journal of Applied Psychology*, *92*, 928–941. doi: 10.1037/0021-9010.92.4.928

Schmidt, A. M., & Dolis, C. M. (2009). Something's got to give: The effects of dual-goal difficulty, goal progress, and expectancies on resource allocation. *Journal of Applied Psychology*, *94*, 678–691. doi: 10.1037/a0014945

Schmidt, A. M., Dolis, C. M., & Tolli, A. P. (2009). A matter of time: Individual differences, contextual dynamics, and goal progress effects on multiple-goal self-regulation. *Journal of Applied Psychology*, *94*, 692–709. doi: 10.1037/a0015012

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*, 461–464.

Steel, P., & König, C. J. (2006). Integrating theories of motivation. *Academy of Management Review*, *31*, 889–913.

Tarakci, M., Greer, L. L., & Groenen, P. J. F. (2016). When does power disparity help or hurt group performance? *Journal of Applied Psychology*, *101*, 415–429. doi: 10.1037/apl0000056

Vancouver, J. B., Tamanini, K. B., & Yoder, R. J. (2010). Using dynamic computational models to reconnect theory and research: Socialization by the proactive newcomer as example. *Journal of Management*, *36*, 764–793. doi: 0.1177/0149206308321550

Vancouver, J. B., Wang, M., & Li, X. (2018). Translating informal theories into formal theories: The case of the dynamic computational model of the integrated model of work motivation. *Organizational Research Methods*, 1–37. doi: 10.1177/1094428118780308

Vancouver, J. B., & Weinhardt, J. M. (2012). Modeling the mind and the milieu: Computational modeling for micro-level organizational researchers. *Organizational Research Methods*, *15*, 602–623. doi: 10.1177/1094428112449655

Vancouver, J. B., Weinhardt, J. M., & Schmidt, A. M. (2010). A formal, computational theory of multiple-goal pursuit: Integrating goal-choice and goal-striving processes. *Journal of Applied Psychology*, *95*, 985–1008. doi: 10.1037/a0020628

Vancouver, J. B., Weinhardt, J. M., & Vigo, R. (2014). Change one can believe in: Adding learning to computational models of self-regulation. *Organizational Behavior and Human Decision Processes*, *124*, 56–74. doi: 10.1016/j.obhdp.2013.12.002

Walsh, M. M., Gunzelmann, G., & Van Dongen, H. P. A. (2017). Computational cognitive modeling of the temporal dynamics of fatigue from sleep loss. *Psychonomic Bulletin and Review*, *24*, 1784–1807. doi: 10.3758/s13423-017-1243-6

Zhou, L., Wang, M., & Vancouver, J. B. (2018). A formal model of leadership goal striving: Development of core process mechanisms and extensions to action team context. *Journal of Applied Psychology*. doi: 10.1037/apl0000370

Zhou, L., Wang, M., & Zhang, Z. (2019). Intensive longitudinal data analyses with dynamic structural equation modeling. *Organizational Research Methods*, 1–32. doi: 10.1177/1094428119833164