ABSTRACT

Title of Dissertation:	ESTIMATING THE Q-DIFFUSION MODEL PARAMETERS BY APPROXIMATE BAYESIAN COMPUTATION
	Chen Tian, Doctor of Philosophy, 2023
Dissertation directed by:	Associate Professor, Yang Liu, Human Development and Quantitative Methodology

The Q-diffusion model is a cognitive process model that considers decision making as an unobservable information accumulation process. Both item and person parameters decide the trace line of the cognitive process, which further decides observed response and response time. Because the likelihood function for the Q-diffusion model is intractable, standard parameter estimation techniques such as the maximum likelihood estimation is difficult to apply. This project applies Approximate Bayesian Computation (ABC) to estimate parameters of the Q-diffusion model. Different from standard Markov chain Monte Carlo samplers that require pointwise evaluation of the likelihood function, ABC builds upon a program for data generation and a metric on the data space to gauge the similarity between imputed and observed data. This project aims to compare the performance of two criteria for gauging the similarity or distance. The limited-information criterion measures the distance in suitable summary statistics (i.e., variances, covariances, and means) between imputed and observed data. The enhanced limited information criterion additionally considers the dependencies among persons' responses and

response times. Bias, rooted mean squared error, and coverage of credible intervals were reported. Results show that when using posterior median as the point estimate, by jointly considering a person's responses and response time, the enhanced criterion yielded less biased estimation on population scale of person power and slightly better item parameters. This SMC-ABC algorithm informs researchers about key data features that should be captured when determining the stopping rule for the algorithm.

ESTIMATING THE Q-DIFFUSION MODEL PARAMETERS BY APPROXIMATE BAYESIAN COMPUTATION

by

Chen Tian

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Associate Professor Yang Liu, Chair
Professor Gregory R. Hancock
Professor Peter M. Steiner
Associate Professor Tracy M. Sweet
Associate Professor Xin He

Dedication

For the coauthor of my life.

Acknowledgements

I want to thank my advisor, Dr. Yang Liu, for all you did during my five years of graduate study. I appreciate and remember your unreserved support every time I face challenges, from upsetting failed job interviews to tough research difficulties. I appreciate your serious and responsible attitude towards our weekly meetings and thoughtful guidance, which lasted for five years as if it were one day. I feel so lucky to have spent five years of my 20s working with you. You have always directed me towards a bright future, with unwavering support and guidance.

Thank Dr. Peter Steiner, for your support during summer and your invaluable guidance on conducting projects.

Thank Dr. Alia Lancaster and the Academic Technology Experience team at the Division of Information Technology, for your support during my last year at UMD.

Thank members in Maryland Assessment Research Center, for supporting me in my early graduate student years.

Thank my committee members, for your supports and suggestions.

Thank many of QMMS members. I remember your helping hands.

Thank HPC at UMD, who supports me on finishing the computation.

Thank my parents and sister, who always stand by me and firmly support me.

Thank Hongyu Yang. Being with you makes me feel alive.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In addition to responses, response times contain important information that can help improve measurement accuracy, explain test taking behaviors, and inform test development. As collateral information, response times can help improve both the accuracy and bias of item response theory (IRT) parameter estimates (e.g., van der Linden et al., 2010). Relatively short or long response times can help diagnose aberrant test-taking behaviors such as rapid guessing and low motivation (e.g., Wang & Xu, 2015). Response times also inform test developers about the amount of time required by each item. This enables test developers to make appropriate testing accommodations for disabled students (van der Linden, 2007) and adaptively select items that give maximum information per time unit (e.g., Choe et al., 2018).

Measurement models and process models both consider responses and response times jointly. For measurement models, Molenaar et al. (2015a) proposed a Bivariate Generalized Linear Item Response Theory (B-GLIRT) modeling framework. B-GLIRT models are latent variable models that link responses and response times by cross-relations. The cross-relations take various forms, and many popular models such as van der Linden's (2007) hierarchical model can be expressed as a special case. B-GLIRT models focus on assessing individual differences rather than modeling the underlying cognitive processes that produced individual differences. Though B-GLIRT models may represent the structure of data well, the interpretation of latent variables plainly describes individual differences without a detailed picture of individual characteristics that play roles in cognitive processes.

In contrast, process models are rooted in cognitive theory and believe the joint distribution of responses and response times depends on the nature of response processes. Different process models have different applicable scenarios and assumptions. The diffusion model (Ratcliff, 1978) is for binary decision making and assumes the cognitive process is a noisy information accumulation process. An example is to decide whether to drive left or right round a car in front. Diffusion model assumes that the noisy information accumulation process involves a rapid matching of immediate surroundings and stored knowledge in memory (Ratcliff et al., 2016). Once the accumulated information reaches a boundary or limit, a corresponding decision is made. Alternative to diffusion models, there are various accumulator models that assume a race between different information accumulation processes (i.e., accumulators). Each accumulator has its boundary, and the first accumulator reaches its bound determines the response choice and response time. Different accumulator models have different assumptions about the properties of each accumulator and how accumulator relates to response choices (e.g., Tuerlinckx & De Boeck, 2005; Ranger & Kuhn, 2014, Rouder et al., 2015; Tillman et al., 2020).

This study focuses on the Q-diffusion model (van der Maas et al. 2011), which is a process IRT model that incorporates the underlying cognitive process (i.e., information accumulation process) when respondents are deciding which answer is correct. The underlying process is assumed to be a Wiener process with some starting point, drift rate, volatility, and two absorbing boundaries (Figure 1). When a respondent starts problem solving, the process starts between two boundaries. The sign of drift rate decides which boundary this process is generally heading to. The

upper boundary is for the correct answer, and the lower boundary is for incorrect answer(s). The extent of wiggling is defined by the volatility. When accumulated information reaches a limit (i.e., the process hits a boundary), the respondent will stop and make corresponding response. What we observe is the final response (i.e., boundary) and the response time. Q-diffusion model is appropriate when the tasks required by the measurement instrument are simple. That is, a task depends on a single ability and no other abilities. An example is the mental rotation problem. It simply depends on respondents' spatial imagination, without involving any reading or calculation ability. Respondents extract the features from the targeted object; when a respondent believe he/she knows enough about the object, he/she will make a choice from given options.

There are two types of parameters in a Q-diffusion model: drift rate and boundary separation. The drift rate is related to the quality of extracted information. In a two-choice Q-diffusion model, the drift rate is decomposed using the *quotient* of person drift rate and item drift rate (van der Maas et al., 2011). The person drift rate may be conceptualized as person ability, as a higher ability is related to higher quality of information. The item drift rate may be conceptualized as item difficulty. The boundary separation is related to the amount of needed information, and it is the *quotient* of person boundary separation and item boundary separation (van der Maas et al., 2011). The person boundary separation parameter may be conceptualized as person carefulness, as the more careful one person is, the more information one wants to accumulate before making a confident decision. The item boundary separation parameter may be conceptualized as item time pressure.

1.1 Statement of the Problem

Despite interesting interpretation of Q-diffusion models, the underlying cognitive processes are complicated and unobservable. The joint density of response and response time involves infinite sums (see Equations 1 and 2), which makes likelihood-based estimation techniques difficult. Section 2.2.2 discussed the computation issues when evaluating the likelihood function. In Bayesian inferences, some previous efforts are discussed in Section 2.5, but few literatures applied likelihood-free Bayesian methods.

1.2 Purpose and Significance of the Study

We aim to sample from the posterior distribution without directly evaluating the likelihood function. Our algorithm is based on Approximate Bayesian Computation (ABC). In standard Bayesian techniques, the posterior involves the likelihood of a parameter value given the observed data (Equation 10). In contrast, ABC examines if the simulated data from a plausible parameter value is *close* to the observed data (Equation 11). For defining "closeness", we combine ABC with Sequential Monte Carlo (SMC) and gradually reduce the tolerance of closeness (Algorithm 3, adapted SMC-ABC algorithm). To achieve sufficient movement before each time of reducing tolerance, Algorithm 3 uses local random-walk Metropolis moves and determines the number of moves adaptively by monitoring the autocorrelations.

We aim to compare the performance of standard limited information and enhanced limited information criteria when evaluating closeness between observed

and imputed data. We are interested in exploring key data features that inform the estimation of model parameters.

This application is an exploration on applying likelihood-free Bayesian method on high-dimensional problems. The results or algorithm itself may inspire other researchers when handling intractable likelihood functions. Our results inform on the choice of summary statistics in designing an ABC algorithm.

1.3 Overview of the Chapters

Sections 2.1-2.5 focuses on interpretation and estimation issues about diffusion models, especially the diffusion measurement model that includes latent variables. Section 2.1 introduces the basic diffusion process and some variant models that have applications in social sciences. Section 2.2 summarizes the statistical and computational issues about diffusion processes, which are frequently mentioned in the literature. Sections 2.3-2.4 review the implied distributions of responses and response times and connect them to IRT models for better interpretations. Section 2.5 reviews current estimation methods and critiques on their limitations.

Following estimation issues in diffusion models, Sections 2.6-2.7 review possible estimation methods that avoid evaluating the likelihood function. Sections 2.6-2.7 are the foundation of our algorithm. Section 2.6 introduces ABC and some simple ABC algorithms. Equations 10 and 11 express the core idea of ABC. Simple ABC algorithms are naïve for our high-dimensional problem. Therefore, Section 2.7 discusses the SMC method that can be combined with ABC and yield efficient estimation algorithms. Algorithms 1 and 2 are the foundation of our algorithm (i.e., algorithm 3).

Chapter 3 introduces our algorithm. Algorithm 3 is a frequently referred skeleton, and Section 3.1 introduces all detailed settings. Section 3.2 introduces the simulation design and evaluation criteria. Chapter 4 presents simulation results on point estimates and estimated posteriors. Implications, possible extensions, and limitations of the present work are discussed in Chapter 5.

Chapter 2: Literature Review

2.1 Diffusion Process and its Applications in Social Sciences

Diffusion process is the mathematical foundation for diffusion measurement models and diffusion decision models. Wiener process, a type of diffusion process, is frequently used in psychometrics and experimental psychology to model responses and response times in decision-making tasks. Diffusion decision models are developed to model the cognitive processes (e.g., memory retrieval, lexical decision, etc.) of individuals to inform human cognitive mechanisms in experimental psychology. On the other hand, individual differences in the decision making process can be explicitly modeled by diffusion measurement models. This section provides an overview of these three concepts, their definitions, and their applications in different fields.

2.1.1 Diffusion Process

Diffusion processes are Markov processes in which only continuous changes of state occur over time (Cox, 1970, p. 203). An example of diffusion process is the movement of particles suspended in a fluid. If we plot the displacements of a particle in a given direction against time, we obtain a diffusion process with erratic path. This process is continuous because particles only move to neighboring positions. This is also a Markov process, because the probability of a particle moving to some position only depends on the current position, regardless of previous trace of this particle.

The above example is also a Wiener process, which is a frequently used diffusion process. Wiener process is a continuous limit of the simple random walk where one-step transitions are permitted only to the nearest neighboring states (Cox,

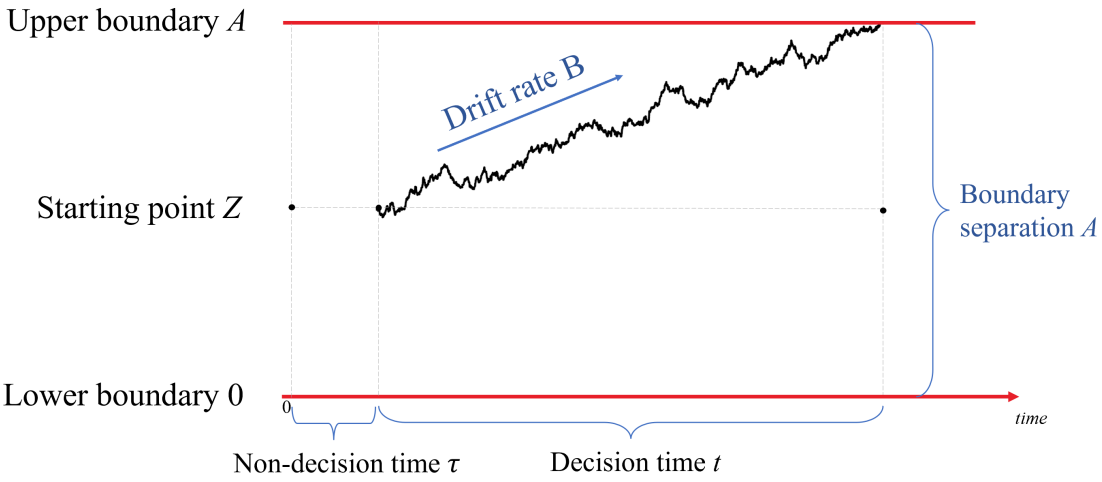
1970, p. 205). The Wiener process can be represented by a continuous random variable $\mathbb{X}(t)(t \geq 0)$, denoting the position of the process in the state space at time *t*. There are three components deciding $\mathbb{X}(t)$: starting point *Z*, drift rate *B*, and volatility *V*. For a time-homogeneous Wiener process where *B* and *V* are constant over time, we have $\mathbb{X}(t) = Z + Bt + VS(t)$, where $S(t)$ is the standardized Wiener process whose starting point is 0, drift rate is 0, and volatility is 1. $\mathbb{X}(t)$ has the property that for any $0 < s < t$, $\mathbb{X}(t) - \mathbb{X}(s) \sim N((t-s)B, V^2(t-s))$. In words, from time *s* to time *t*, the increment in the position of the process has a normal distribution with mean equal to drift rate times the time interval and standard deviation equal to volatility times the square root of time increment. This property shows that the drift rate *B* is the mean increment rate. A positive drift rate means that $\mathbb{X}(t)$ tends to increase as *t* increases. The volatility decides how large a step size could be. The larger the volatility, the more erratic a path is.

The above-mentioned time-homogeneous Wiener process have some variants for applications in various social sciences. Some consider drift rate or volatility as functions of time. For example, in econometrics, the price of assets changes over time and can be considered as a diffusion process. The price does not always fluctuate at a constant volatility, and a volatility changing across time has been reported in many studies about stock, currency, and commodity (Tayler, 1994). The time heterogeneous Wiener process is not the focus of this review, because it is less relevant to applications in psychometrics. A more frequently used process in psychometric literature is the Wiener process with two absorbing boundaries (Figure 1). The starting point is between the two boundaries. Once the process reaches an absorbing

boundary, it stops. Literatures in psychometrics and experiment psychology refer Wiener processes with two boundaries as *diffusion models*. *Diffusion decision models* or *drift diffusion models* are developed for experiment psychology, and *diffusion measurement models* are developed for psychometrics. The following describes these two types of models in details.

Figure 1

An Unbiased Wiener Process with Two Absorbing Boundaries



2.1.2 Diffusion Decision Models

The *diffusion decision model* (DDM) was developed in experimental psychology for modeling responses and response time from fast binary decision tasks (Ratcliff, 1978). Using DDM for binary decision tasks can help researchers represent experiment data and answer questions in different domains of cognitive psychology such as memory retrieval, perception processes, lexical decision, etc. DDM assumes that individuals continuously track their momentary preference for one response option relative to the other (Ranger et al., 2016). Once the preference towards one option is accumulated enough, a choice is made, and the individual stops thinking.

The trace of momentary preference can be appropriately modeled by a Wiener process with two absorbing boundaries.

The interpretations of diffusion model parameters make sense in experimental psychology. Drift rate is the quality of information extracted from the stimulus. For example, in memory retrieval tasks, the drift rate depends on the relatedness between stimulus and memory. Drift rate is interpreted as the mean rate of information accumulation towards a boundary (Ratcliff & Tuerlinckx, 2002). A smaller absolute drift rate leads to slower responses. Boundary separation reflects the amount of information needed for decision making. Boundary separation is influenced by the experiment instructions that ask participants to focus more on speed or accuracy. A larger boundary is related to higher accuracy and slower response. Volatility describes if the process wanders across a wide range. A larger volatility means the process is more likely to reach the non-expected boundary by chance. Volatility is usually fixed for the purpose of model identification. DDM further considers a *residual time*, or *non-decision time*, which is necessary for participants to encode the presented stimulus and execute a response. Residual time is not related to the information accumulation process. The total response time is residual time plus the decision time (Figure 1).

Experimental psychology studies do not decompose drift rate and boundary separation into person latent traits and trial/item characteristics. They are interested in collective human cognitive mechanisms rather than individual differences in latent traits. They are also not interested in specific trial but interested in if groups of trials

under different conditions stimulated different cognitive mechanisms. DDM is not appropriate for measurement purposes.

2.1.3 Diffusion Measurement Models

The *diffusion measurement models* were developed from the DDM. Similar to DDM, diffusion measurement models assume that to make a response, the process of information accumulation is noisy and continuous. The drift rate is related to the quality of information extraction, and boundary separation is related to speed-accuracy tradeoff. Different from DDM, diffusion measurement models decompose drift rate and boundary separation into item and person characteristics. For measuring bipolar traits such as extraversion, Tuerlinckx et al. (2005) proposed the D-diffusion model that uses a difference function for decomposition. For measuring abilities such as the ability to do spatial imagination, van der Maas et al. (2011) proposed the Q-diffusion model that uses a quotient function for decomposition. D-diffusion and Q-diffusion models are different in the way they describe how item and person characteristics interact to determine drift rate, boundary separation, thus the cognitive process. The choice of difference or quotient function is related to the interpretation of parameters. Empirical examples for diffusion measurement models are included in the later sections for D- and Q-diffusion models.

This review focuses on diffusion measurement models. The following discusses the distributions of responses, response times, interpretations of parameters, and estimation issues.

2.2 Joint Distribution of Responses and Response Times

The joint probability density of responses and response times involves infinite series. Therefore, exact evaluation of the joint density is impossible. Various computational techniques have been proposed to numerically approximate the density and generate Monte Carlo samples that approximately follow the joint density.

2.2.1 Joint Distribution

This section reviews the joint probability density function (*p.d.f.*) for Wiener processes with two absorbing boundaries. This density function applies to both diffusion decision models and diffusion measurement models, except that the diffusion measurement models further decompose drift rate B and boundary separation A (e.g., Tuerlinckx & De Boeck, 2005, Equation 3; Molenaar et al., 2015, Equation 1). There are two forms for the joint distribution, derived separately using different methods (e.g., the method of images, Fourier series solutions, see Cox, 1977, *p*. 222, Equations 78 and 81). Both forms involve infinite sums. One form is less frequently used in psychometric literatures. Its expression is given in problem 22 in Feller (1968, Chap 14):

$$f_{T,Y}(T=t, Y=0|A,B,Z,V) = \frac{1}{\sqrt{2\pi V^2 t^3}} \exp\left(-\frac{B^2 t + 2ZB}{2V^2}\right) \times \sum_{k=-\infty}^{\infty} (Z+2kA) \exp\left(-\frac{(Z+2kA)^2}{2tV^2}\right). \quad (1)$$

The other form is more frequently used in psychometric literatures. The expression is given in Equation 81 in Cox (1977, Chap 5):

$$f_{T,Y}(T=t,Y=0|A,B,Z,V)=\frac{\pi V^2}{A^2}\exp\left(-\frac{ZB}{V^2}-\frac{B^2}{2V^2}t\right)\times \sum_{k=1}^{\infty}k\sin\left(\frac{\pi kZ}{A}\right)\exp\left(-\frac{\pi^2V^2k^2}{2A^2}t\right). \quad (2)$$

In both forms, B is the drift rate, A is the boundary separation, Z is the starting point, and V is the volatility. T is the time spent on the diffusion process. Y has values 0 and 1, representing which boundary the process hits. For hitting the upper boundary (i.e., $Y=1$), we can obtain the *p.d.f.* by replacing B by -B and Z by (A - Z) in Equations 1 and 2. This operation interchanges the role of two boundaries by using a flipped diffusion process. To write Equations 1 and 2 in a general way like $f_{T,Y}(T=t,Y=y|A,B,Z,V)$, we can replace the $-ZB$ in exponential terms by $(Ay-Z)B$, and replace Z in the infinite sums by $(Ay-2yZ+Z)$ (Boehm et al., 2021). Note that in Equations 1 and 2, multiplying A, B, Z, and V by a constant c will result in the same *p.d.f.*. It means that for model identification, one parameter should be restricted to an arbitrary value. Most previous literatures fixed $V=1$ for easier interpretation (e.g., Tuerlinckx & De Boeck, 2005; Navarro, Fuss, 2009).

2.2.2 Computation Issues and Solutions

Both forms of the joint *p.d.f.* involves infinite sums, which makes it difficult to evaluate the density function value. Truncation and approximation are needed for calculating the sum of infinite number of terms. A standard way is to compare each term to the previous sum and terminate when the term is too small to change the sum. For the first form expressed in Equation 1, if t is small, the terms in the infinite sum are close to 0 and the approximation converges fast. If t is large, the expression becomes unstable (Van Zandt et al., 2000). For the second form in Equation 2, if t is

large, the terms in the infinite sum are close to 0 and the approximation converges fast. If t is small, due to the sine function, the value of term oscillates and is not monotonically decreasing as k increases.

Though using a single form of joint *p.d.f.* is doable for evaluating function values, the convergence is slow. Ratcliff and Tuerlinckx (2002) only used the second form (Equation 2) for approximating the cumulative density function (*c.d.f.*). As stated before, the sine function in the infinite sum makes the value of term fluctuates a lot. A term can be close to 0 and barely change the sum, but its next term may be higher than a tolerance and change the sum a lot. To avoid terminating the approximation incorrectly, instead of examining a single term, Ratcliff and Tuerlinckx (2002) examined if two *successive* terms in the *c.d.f.* were both less than 10^{-29} times the current value of the sum. Their method is intuitively reasonable, but the convergence is slow for small t values. For $t = 0.001$, this method requires 227 terms (Navarro and Fuss, 2009). Besides, it does not explicitly specify the acceptable degree of approximation error.

Van Zandt et al. (2000) and Navarro and Fuss (2009) took advantage of both forms and developed more efficient methods. As stated before, Equation 1 converges fast for small t, and Equation 2 converges fast for large t. Given a t value, the algorithm can use whichever form is superior. To determine if a time value is small or large, unlike van Zandt et al. (2000) who used a heuristic method, Navarro and Fuss (2009) analytically define what time value is "small", considering the specified acceptable degree of approximation error. This analytical method minimizes the number of terms evaluated. Navarro and Fuss (2009) found that even for extremely

stringent error tolerances, this method requires no more than 8 terms to approximate the *p.d.f.*. Navarro and Fuss's (2009) method was adopted by Monlenaar et al. (2015b) for developing the R package *diffirt* for fitting diffusion models.

2.3 Distribution of Response

The probability of hitting the upper boundary is given by (e.g., Tuerlinckx et al., 2001):

$$P(Y=1)=\frac{\exp\left(-\frac{2ZB}{V^2}\right)-1}{\exp\left(-\frac{2AB}{V^2}\right)-1}. \quad (3)$$

Since the process will either hit the upper or the lower boundary, we have $P(Y=0)=1-P(Y=1)$. We can also obtain $P(Y=0)$ from Equation 3 by replacing *B* by *-B* and *Z* by (*A-Z*). If the diffusion process is unbiased (i.e., $Z=A/2$), Equation 3 is further reduced to:

$$P(Y=1)=\frac{1}{1+\exp\left(-\frac{AB}{V^2}\right)}, \quad (4)$$

which is similar to a 2-parameter logistic (2PL) model. The following discusses unbiased D-diffusion and Q-diffusion models and their relation to the 2PL model. For simplicity and model identification, V was set to 1 for Equations

(*5*, 6, and 7.

2.3.1 D-diffusion Model for Bipolar Traits

Tuerlinckx and De Boeck (2005) proposed the D-diffusion model for measuring bipolar traits such as personality and attitude. D-diffusion models assume

both persons and items can be ordered on a continuum with two ends (Tuerlinckx et al., 2016). For example, neuroticism is a continuum from introvert to extravert. A question measuring neuroticism may ask "Do you enjoy talking with others" and give two response options "Yes" and "No". An item has its location on the continuum (i.e., threshold). If a person's trait level is close to the item's threshold, this person will hesitate between two options because he/she has a comparable preference to selecting either one.

D-diffusion model decomposes the drift rate *B* as a simple difference between person trait level *b* and item threshold β (i.e., $B = b - \beta$). Originally, boundary separation *A* was not decomposed (Tuerlinckx & DeBoeck, 2005). Recently, Tuerlinckx et al. (2016) borrowed the idea of Q-diffusion model and decomposed boundary separation using the ratio of person caution a and item time pressure α (i.e., $A = a/\alpha$). Fixing $V = 1$ in Equation 4, we have $P(Y = 1) = 1/(1 + \exp(-(a/\alpha)(b - \beta)))$, which is exactly a 2PL model. The boundary separation is the 2PL discrimination, the person trait level is the 2PL latent trait level, and the item threshold is the 2PL difficulty.

It is intuitive to decompose the drift rate as $b - \beta$, interpret person traits level *b* as a 2PL latent trait level, and interpret item threshold β as the 2PL item difficulty. For drift rate, in diffusion processes, the sign of drift rate decides which answer (i.e., yes/no) or boundary this person is heading to. Using $b - \beta$ as the drift rate, the sign of drift rate is defined as if a person has a higher trait level than the item "expects" (i.e., item threshold).

Decomposing the boundary separation as a/α and interpreting the ratio as 2PL discrimination also make sense. In diffusion processes, a larger separation is related to more accumulated information and less misclassification (Tuerlinckx & De Boeck, 2005). Using a/α as boundary separation, higher person caution a or lower item time pressure α defines larger separation, which means more accumulated information before decision making and less misclassification. As for interpreting boundary separation as 2PL item discrimination, note that a larger 2PL item discrimination is also related to less misclassification, due to increased $|P(\vartheta_2) - P(\vartheta_1)|$ for $\vartheta_1 < item\ difficulty < \vartheta_2$, where ϑ is the latent ability in 2PL models.

Though the D-diffusion model provides interesting predictions for items measuring bipolar traits, it is less suitable for items measuring ability (e.g., van der Maas et al., 2011; Tuerlinckx et al., 2016). First, D-diffusion model predicts that the response time is longest when $b - \beta = 0$ (i.e., zero drift rate). Persons with $b \ll \beta$ are as fast as persons with $b \gg \beta$. It is plausible for measuring bipolar traits but not for measuring ability (van der Maas et al., 2011). Second, the D-diffusion model predicts that persons with $b < \beta$ are more likely to hit the lower boundary as time pressure decreases (i.e., larger boundary separation). This is not true in ability tests, because giving more time to less able examinees is not supposed to result in higher probability of answering incorrectly.

2.3.2 Q-diffusion Model for Abilities

Q-diffusion model was proposed by van der Maas et al. (2011) for ability tests. They reconsider the concept of ability and believes that any task measuring an

ability requires some of the ability. For example, a task measuring the distance one person walks require a person to walk a positive distance. Therefore, ability has a natural zero point and always take non-negative values. Any individual who possesses this ability can carry out this task given sufficient time. Q-diffusion model is appropriate for measuring simple abilities. The task should only depend on a single ability and no other abilities. One example is the mental rotation problem (van der Maas et al., 2011), in which respondents need to decide which option is a rotated representation of the target object. This task only depends on the ability of spatial imagination. Another example is the chess puzzle (e.g., the Amsterdam Chess Test; van der Maas & Wagenmakers, 2005), in which chess players solve puzzles by selecting the best move given a certain configuration.

Based on the idea of positive ability, van der Maas et al. (2011) decomposed drift rate using the ratio between person pure ability level b and item pure difficulty level β. The item pure difficulty level β is on the same scale as b and also takes positive numbers. Fixing $V = 1$ in Equation 4, the probability that a person "hit" the upper boundary and answer correctly to an item becomes $P(Y = 1) = 1/(1 + \exp(-(a/\alpha)(b/\beta))$), which is applicable to two-choice items. van der Maas et al. (2011) further generalized the model to *O*-option items. Using Bock's nominal model, the probability of answering an *O*-option item correctly is given by:

$$P(Y = 1) = \frac{1}{1 + \exp\left(-\frac{a}{\alpha}\frac{b}{\beta} + \ln(O - 1)\right)}, \quad (5)$$

which can be written in the form of a diffusion model defined in Equation 4:

$$P(Y=1)=\frac{1}{1+\exp\left(-\frac{a}{\alpha}\left(\frac{b}{\beta}-\frac{\alpha}{a}\ln(O-1)\right)\right)}, \quad (6)$$

and rewritten in a 2PL form:

$$logit\ P(Y=1)=\frac{1}{\alpha\beta}(ab-\alpha\beta\ln(O-1)). \quad (7)$$

From Equation 6, the new drift rate for an O-option item is $b/\beta-(\alpha/a)\ln(O-1)$, and the new boundary separation is a/α. For two-choice items, the drift rate is always positive. The larger the person pure ability b compared to item pure difficulty β, the higher the drift rate and $P(Y=1)$. For items with more than two options, the drift rate may be negative. The less options one item has, the higher the drift rate. With $O>2$, fixing O, b, and β, the lower the time pressure α or the higher the person caution a, the higher the drift rate and $P(Y=1)$. This decomposition of drift rate intuitively makes sense. For the decomposition of boundary separation, Q-diffusion model shares the same interpretation as the recently proposed D-diffusion model described above.

From Equation 7, ab represents person latent ability in 2PL models, $\alpha\beta\ln(O-1)$ represents the 2PL item difficulty, and $1/\alpha\beta$ represents the 2PL item discrimination. The Q-diffusion model connects person caution and pure ability to the person latent ability in 2PL models, which is in line with De Boeck & Jeon's (2019) statement that the latent ability in 2PL is an "effective" ability for an unknown speed-accuracy tradeoff. The Q-diffusion model connects item time pressure, item pure difficulty, and the number of options to the 2PL difficulty which is "effective" difficulty. The Q-diffusion model connects item time pressure and item pure difficulty to 2PL discrimination. The higher the item time pressure or pure difficulty,

the lower the discrimination, or in other words, the smaller $P(Y = 1)$ for persons with $ab > \alpha\beta \ln(O - 1)$. Note that all persons have $ab > \alpha\beta \ln(O - 1) = 0$ for two-option items, and the majority have $ab > \alpha\beta \ln(O - 1) > 0$ for *O*-option items.

2.4 Distribution of Response Time

The time for human decision-making is a random variable described by the *first passage time* for Wiener processes. It is the time taken for $\mathbb{X}(t)$ to reach an absorbing boundary. The conditional distribution of first passage time is $f_{T,Y}(T = t|Y = 0, A, B, Z, V) = f_{T,Y}(T = t, Y = 0|A, B, Z, V)/P(Y = 0)$, where $f_{T,Y}(T = t, Y = 0|A, B, Z, V)$ is defined in Equations 1 and 2, and $P(Y = 0)$ is defined in Equation 3 (e.g., Tuerlinckx et al., 2001). To obtain the marginal distribution of first passage time, we can use the law of total probability and have $f_{T,Y}(T = t|A, B, Z, V) = f_{T,Y}(T = t|Y = 0, A, B, Z, V) + f_{T,Y}(T = t|Y = 1, A, B, Z, V)$. The marginal distribution is rarely used in psychometrics because *Y* is observable and easy to condition on.

Most psychometric literature assume the Wiener process is unbiased (i.e., $Z = A/2$, the starting point is in equidistant from two boundaries) and simplified the distributions of conditional first passage time. For an unbiased Wiener process, the conditional distribution of first passage time is the same for correct and incorrect responses (i.e., $f_{T,Y}(T = t|Y = 0, A, B, Z, V) = f_{T,Y}(T = t|Y = 1, A, B, Z, V)$, e.g., Tuerlinckx et al., 2001). Furthermore, given $Z = A/2$, the sine term reduces to $sin\ (\pi k/2)$ and only takes three values. The simplified conditional distribution is always smaller than an exponential distribution, which allows researchers to use

rejection sampling and sample first passage times without simulating the whole path (Tuerlinckx et al., 2001, *p*. 446-447).

For unbiased Wiener processes, Wagenmakers et al. (2005) derived the closed form expressions for the expectation and variance of first passage time. These expressions greatly simplify calculations and analytically describe how drift rate and boundary separation determine the distribution of response time. The expressions are:

$$E(T) = \begin{cases} \dfrac{A^2}{4V^2}, & B \to 0 \\ \left(\dfrac{A}{2B}\right)\dfrac{1-\exp\left(-\dfrac{AB}{V^2}\right)}{1+\exp\left(-\dfrac{AB}{V^2}\right)}, & otherwise \end{cases} \tag{8}$$

$$var(T) = \begin{cases} \dfrac{A^4}{24V^4}, & v \to 0 \\ \left(\dfrac{A}{2B}\right)\left(\dfrac{V^2}{B^2}\right)\dfrac{-\dfrac{2AB}{V^2}\exp\left(-\dfrac{AB}{V^2}\right)-\exp\left(-\dfrac{2AB}{V^2}\right)+1}{\left(1+\exp\left(-\dfrac{AB}{V^2}\right)\right)^2}, & otherwise \end{cases} \tag{9}$$

There are also closed-form solutions for first-passage time for specific classes of time-varying processes (e.g., Broderick et al., 2010), but are less frequently used in psychometric literature.

2.5 Parameter Estimation for Diffusion Models

This section reviews methods for estimating diffusion models. Section 2.5.1 introduces how to estimate an EZ diffusion model, which is a simplified version of general *diffusion model*. Section 2.5.2 briefly discusses how to estimate *diffusion decision models*. Section 2.5.3 is the focus of this section and discusses methods for estimating *diffusion measurement models*.

2.5.1 EZ Diffusion Model

The EZ diffusion model proposed by Wangenmakers et al. (2007) considerably simplifies the estimation of diffusion models by adding three constraints. First, unlike the diffusion decision model, EZ diffusion model does not allow a person to have different non-decision time, starting point, or drift rate on different trials/items. Second, unlike diffusion measurement models, EZ diffusion model does not decompose drift rate and boundary separation into person and item characteristics. Third, EZ diffusion model assumes all Wiener processes are unbiased (i.e., $Z = A/2$). Though the EZ diffusion model does not account for the observed data in detail, it reduces the number of parameters and popularizes diffusion models by simplifying the model estimation. From the mean response time, variance response time, and the probability of answering correctly, EZ diffusion model is able to analytically determine drift rate, boundary separation, and non-decision time, using Equation 4, Equation 8, Equation 9, and the fact that the total response time is a sum of non-decision time and first passage time. Van Ravenzwaaij et al. (2017) fitted the EZ diffusion model from data generated from the full diffusion decision model and found that the parsimonious EZ diffusion model sometimes is better at detecting experimental effects than the true model. The estimates from the EZ diffusion model may also serve as starting values for more complex diffusion models (e.g., Molenaar et al., 2015b).

2.5.2 Diffusion Decision Model

Ratcliff's (1978) diffusion decision model assumes that for different trials/items, a person has normally distributed drift rate *B*, uniformly distributed non-

decision time τ, and uniformly distributed starting points *Z*. So, the model estimates seven parameters: the individual means and variances for *B*, τ, and *Z*, plus an invariant boundary separation *A*. Common methods for estimating diffusion decision models include the maximum likelihood (ML) method, the chi-square method, and the weighted least square (WLS) method (e.g., Ratcliff & Tuerlinckx, 2002; Ratcliff & Childers, 2015). ML method calculates the multiplication of joint *p.d.f.* (Equation 2) across all trials for a person. The multiplication is then integrated over the distributions of drift rate, starting point, and non-decision time, which yields the likelihood function to be maximized. The chi-square method minimizes the difference between observed and model implied quantiles for response times conditioning on correct and incorrect responses. WLS method minimizes the differences between observed and model implied accuracy values plus the sum of weighted squared differences between observed and model-implied quantiles for response times. The weights are larger for quantile points for which variability was smaller. The full-information method, ML, is the best in terms of smallest bias and standard errors (Ratcliff & Tuerlinckx, 2002). The limited-information method, chi-square and WLS, lost some information by aggregating response time data, but they are more robust to assumption violations. For example, the chi-square method is robust to moderate changes in across-trial distributions of parameter values (Ratcliff, 2013). WLS is robust to outlying response times (Ratcliff & Tuerlinckx, 2002).

2.5.3 Diffusion Measurement Model

Though multiple methods were developed for fitting diffusion decision models, those methods are not appropriate for measurement purposes. Diffusion

measurement models do not assume trials/items under the same condition are equal and are interested in item effects. Besides, diffusion measurement models care more about the individual differences in person characteristics than common human cognitive processes. The following reviews estimation methods specifically designed for diffusion measurement models.

Full Information Methods

The marginal maximum likelihood (MML) method for diffusion measurement models replaces the drift rate B and boundary separation A in Equation 2 with person and item parameters (Molenaar et al., 2015b). The person parameters are treated as nuisance parameters, and they are integrated out of the likelihood equation based on assumptions of their distributions. Lognormal distributions may be assumed for person boundary separation. For person drift rate, D-diffusion model assumes normal distribution, and Q-diffusion model assumes log-normal distribution. The lognormal distribution is chosen for positive random variables mainly for convenience.

Though MML method is the gold standard, it is computationally challenging and not robust to outlying time values. The joint *p.d.f.* involves infinite sum and the marginal likelihood involves multiple integrals over person parameters. Numerically integrating the density function over several parameters introduces the curse of dimensionality. Even adaptive quadrature does not reduce the computational burden much (Ranger et al., 2020). Ranger et al. (2016) stated that, with 1000 persons, MML estimator with 20 quadrature points requires 8 hours for the D-diffusion model and 6 hours for the Q-diffusion model on an Intel Core 2 Duo processor. For the robustness, Jordan (2020) found that a single fast response time can dominate the likelihood and

produce MML estimates close to 0. Ranger and Kuhn (2018) mentioned that categorizing response time into intervals may make ML estimator more robust. However, this method loses some information and is inefficient.

One way to avoid the intractable functions in MML is to use the Bayesian method (Vandekerckhove et al., 2011; van der Maas et al., 2011). One can use uninformative uniform priors for item parameters and standard lognormal priors for person parameters. However, the Bayesian method is also time-consuming due to burn-in and thinning of Markov chains.

Limited Information Methods

Limited information methods can avoid determining the intractable joint *p.d.f.*. For example, the weighted least square (WLS) method uses the covariance matrices of responses and response times and minimizes the difference between model-implied and observed covariance matrices (Ranger et al., 2016). The expression for model implied covariances were derived from Equations 8 and 9, which assume unbiased diffusion processes. The weight matrix may use the standard errors of estimated covariances. WLS is consistent and asymptotically normally distributed (Ranger & Kuhn, 2018). However, WLS also face the problem of contaminated data with outlying response times. Yuan & Bentler (1998) mentioned that a robust WLS can use a robust estimator of the covariance matrix. Ranger and Kuhn (2018) proposed modifications of WLS to fit contaminated data. The modified WLS ignores the variances of response times in the objective function, because those variances are inflated by the contaminants. Besides, based on specific assumptions of the data contamination model, further refinements are possible (e.g., Ranger & Kuhn,

2018). WLS is less efficient than the MML estimator due to limited information (Ranger et al., 2016). To increase efficiency and robustness, Ranger et al. (2020) proposed minimum distance estimator that further considers quantiles of response times in addition to expectations and covariances. However, this method was only developed for D-diffusion models, where the drift rates and log boundary separations are linear functions of the latent traits.

The limited information methods avoid the intractable joint *p.d.f.*, but it brings new problems. The covariance matrix is not a sufficient statistic, and recovering it is only a small part of the whole story. Therefore, limited-information estimators are not fully efficient. Besides, future studies are needed to explain how large the information loss is compared to MML. Is the lost information important? It is unexplored if the limited information methods are robust or just numb to outlying response times due to missed information.

2.6 Approximate Bayesian Computation

Given the difficulty of evaluating the likelihood function in Q-diffusion models, Approximate Bayesian Computation (ABC) is an option for parameter estimation. This section introduces the fundaments of ABC and example algorithms.

2.6.1 What is Approximate Bayesian Computation

ABC is a "likelihood-free" approach for Bayesian inference: It does not require pointwise evaluation of the likelihood function and thus is suitable when the likelihood function is intractable or expensive to evaluate. Instead of obtaining an estimate of the likelihood-based posterior distributions $\pi(\Theta|x^{obs})$, ABC aims to

approximate the posterior distribution $\pi(\Theta|\rho(x^{obs}, X) < \epsilon)$, where X represents the simulated data, $\rho(x^{obs}, X)$ measures the distance or discrepancy between observed and simulated data, and ϵ is a pre-specified tolerance. Stated differently, a parameter value generated from the prior distribution is retained as a draw from the approximate posterior only if it produces imputed data that are close to the observed data. With carefully chosen distance function ρ and small tolerance level ϵ, $\pi(\Theta|\rho(x^{obs}, X) < \epsilon)$ approximates $\pi(\Theta|x^{\text{obs}})$ (e.g., Beaumont, 2010; Pritchard et al., 1999; Tuner and Van Zandt, 2012). ABC is suitable for complicated models whose likelihood functions are not easily calculated analytically. For example, in biological sciences about population genetics, Pritchard et al. (1999) applied ABC to estimate parameters in their model of human demographic history. In our case of estimating parameters in diffusion models, by using ABC, we can avoid calculating the likelihood in Equations 1 and 2, but to simulate artificial data sets and compare them with the observed data.

To compute the posterior distribution $\pi\left(\Theta\middle|x^{\text{obs}}\right)$, we need to compute the likelihood of the observed data and supply a proper[1] prior distribution for Θ. Let $f_{X|\Theta}(x^{obs}|\Theta)$ be the likelihood of Θ given observed data x^{obs}, and let $\pi(\Theta)$ be a proper prior distribution of Θ. By the Bayes formula again, we have:

$$\pi(\Theta|x^{obs}) = \frac{f_{X|\Theta}(x^{obs}|\Theta)\,\pi(\Theta)}{\int f_{X|\Theta}(x^{obs}|\Theta)\,\pi(\Theta)\,d\Theta} \propto f_{X|\Theta}(x^{obs}|\Theta)\pi(\Theta). \tag{10}$$

[1] For example, for a non-negative parameter, a proper prior has zero density on negative parameter values.

To compute or estimate the approximate posterior distribution $\pi(\Theta|\rho(x^{obs}, X) < \epsilon)$, besides the prior, we need the probability of simulating a "close-to-observed" data from some parameter value, $P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)$. Using Bayes' theorem, we have the following:

$$\begin{aligned}\pi(\Theta|\rho(x^{obs}, X) < \epsilon) &= \frac{P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)\pi(\Theta)}{\int P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)\pi(\Theta)\, d\Theta} \\ &\propto P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)\pi(\Theta).\end{aligned} \tag{11}$$

Equation 11 can be viewed as an approximation of Equation 10, if ϵ is small enough and $\rho(\cdot)$ is suitable (e.g., Beaumont, 2010; Marin et al., 2011; Pritchard et al., 1999). It is because the probability of simulating a close-enough data X from Θ is approximately proportional to the probability of simulating x^{obs} from Θ. More formally,

$$\begin{aligned}f_{X|\Theta}(x^{obs}|\Theta) &= \lim_{\Delta x \to 0^+} \frac{Prob_{X|\Theta}(x^{obs} - \Delta x \le X \le x^{obs} + \Delta x|\Theta)}{2\Delta x} \\ &\overset{suitable\ \rho}{=} \lim_{\epsilon \to 0^+} \frac{P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)}{vol\{\rho(x^{obs}, X) < \epsilon|\Theta\}} \\ &\overset{small\ \epsilon}{\approx} \frac{P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)}{vol\{\rho(x^{obs}, X) < \epsilon|\Theta\}} \\ &\propto P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta),\end{aligned}$$

in which $vol\{\rho(x^{obs}, X) < \epsilon|\Theta\}$ is the volume where the data simulated from Θ is close to the observed data.

Equations 10 and 11 are expressed in terms of the raw observed and/or simulated data. However, for computational convenience, the raw data x^{obs} and X

can be replaced by sufficient statistics[2] $S(x^{obs})$ and $S(X)$. A sufficient statistic should contain no less information than the raw data for estimating interested parameters. For example, in independent and identically distributed Bernoulli trials, if we are interested in the probability of success, the total number of successes is a sufficient statistic. For the purpose of estimating success rate, we do not need to know in which trials we succeed and in which trials we failed. Knowing a total number of successes is sufficient. For formal definition for sufficient statistics of a parameter, please refer Fisher-Neyman Factorization Theorem (Rice, 2006, p. 306) and the Equation 4 in Turner and Van Zandt (2012).

Though ABC does not require evaluating the likelihood function, it requires defining a good distance function ρ and a small tolerance level ϵ to measure "closeness". ρ can be the l^p distance in the space of summary statistics ($0 < p \leq \infty$, e.g., Beaumont et al., 2002; Turner and Van Zandt, 2012; Pritchard et al., 1999). According to Turner and Van Zandt (2012), generally speaking, the best distance functions incorporate all data points in x^{obs} and X, except that the model parameters reflect a limit on the range of data points. For example, the sum of all points may be better than the interquartile range. For the tolerance level ϵ, though a smaller tolerance level means closer, it may result in higher rejection rates, especially when the prior is quite different from the posterior. This idea will be shown in the following ABC rejection algorithm. If where we sample from (i.e., prior) gives us a lot of parameters values that cannot simulate close-enough data, the ABC rejection

[2] Other summary statistics may also be used in ABC, not only sufficient statistics (e.g., Turner and Van Zandt, 2012).

algorithm is inefficient. An adaptive schedule of tolerance value may be used, together with the sequential Monte Carlo algorithm.

2.6.2 Example ABC Algorithms

The simplest ABC algorithm is the ABC rejection algorithm (e.g., Beaumont et al., 2010; Pritchard et al., 1999; Turner and Van Zandt, 2012). To obtain a sample of size *n* from the approximated posterior $\pi(\Theta|\rho(x^{obs}, X) < \epsilon)$ (Equation 11), repeat the following steps until *n* draws of Θ have been accepted:

1. Draw a parameter value from its prior: $\Theta \sim \pi(\Theta)$.
2. Simulate data from Θ: $X \sim f_{X|\Theta}(X|\Theta)$.
3. Reject Θ if $\rho(x^{obs}, X) \geq \epsilon$.

Intuitively, the ABC algorithm is in line with the meaning of $\pi(\Theta|\rho(x^{obs}, X) < \epsilon)$: the distribution of Θ given that its simulated data is close to the observed data. Analytically, Marin et al. (2011) mentioned that the ABC-rejection algorithm is taking draws from the following joint distribution:

$$\pi_\epsilon(\Theta, X = x|x^{obs}) = \frac{\pi(\Theta) f_{X|\Theta}(X = x|\Theta) \mathbb{I}(\rho(x, x^{obs}) < \epsilon)}{\int \pi(\Theta) f_{X|\Theta}(X = x|\Theta) \mathbb{I}(\rho(x, x^{obs}) < \epsilon)\, d\Theta dx}.$$

In the numerator, the three terms correspond to step 1, 2, and 3 in the ABC-rejection algorithm. After we have draws of (Θ, X), we discard X and thus have the marginal distribution of Θ:

$$\int \pi_\epsilon(\Theta, X|x^{obs})\, dx \propto \pi(\Theta) \int f_{X|\Theta}(X = x|\Theta) \mathbb{I}(\rho(x, x^{obs}) < \epsilon)\, dx$$

$$= \pi(\Theta) P_{X|\Theta}(\rho(X, x^{obs}) < \epsilon|\Theta),$$

which is proportional to the right-hand-side of Equation 11. As illustrated, the ABC-rejection algorithm yields (marginal) draw from $\pi(\Theta|\rho(x^{obs}, X) < \epsilon)$.

ABC can also be used with Markov chain Monte Carlo (MCMC) sampling (e.g., Bortot et al., 2007; Marjoram et al., 2003; Turner and Van Zandt, 2012). A popular MCMC sampler is the Metropolis-Hastings (MH) algorithm. Starting from some initial value Θ, we propose a new value Θ^* based on Θ (i.e., $\Theta^* \sim q(\Theta, \cdot)$) and accept the new value with some probability. This procedure is repeated until we obtain a chain of values that we assume are a sample from the posterior distribution. Without ABC, the acceptance probability involves likelihood-based posterior (Equation 10) and the proposal distribution q:

$$\begin{aligned} acceptance\ rate &= min\left(1, \frac{\pi(\Theta^*|x^{obs})q(\Theta^*, \Theta)}{\pi(\Theta|x^{obs})q(\Theta, \Theta^*)}\right) \\ &= min\left(1, \frac{\pi(\Theta^*)f_{X|\Theta}(x^{obs}|\Theta^*)q(\Theta^*, \Theta)}{\pi(\Theta)f_{X|\Theta}(x^{obs}|\Theta)q(\Theta, \Theta^*)}\right) \end{aligned}$$

When embedding ABC with the MH sampler, the acceptance probability involves the approximated posterior (Equation 11) and the proposal distribution q:

$$\begin{aligned} acceptance\ rate &= \min\left(1, \frac{\pi(\Theta^*|\rho(X^*, x^{obs}) < \epsilon)q(\Theta^*, \Theta)}{\pi(\Theta|\rho(X, x^{obs}) < \epsilon)q(\Theta, \Theta^*)}\right) \\ &= \min\left(1, \frac{\pi(\Theta^*)P_{X|\Theta}(\rho(X^*, x^{obs}) < \epsilon|\Theta^*)q(\Theta^*, \Theta)}{\pi(\Theta)P_{X|\Theta}(\rho(X, x^{obs}) < \epsilon|\Theta)q(\Theta, \Theta^*)}\right) \end{aligned} \quad (12)$$

The numerator or denominator in the ratio of $\frac{P_{X|\Theta}(\rho(X^*, x^{obs}) < \epsilon|\Theta^*)}{P_{X|\Theta}(\rho(X, x^{obs}) < \epsilon|\Theta)}$ can be approximated by simulating one data set and examine if it is close to the observed data. The resulted acceptance ratio is given by:

$$acceptance\ rate = \begin{cases} min\left(1, \frac{\pi(\Theta^*)q(\Theta^*,\Theta)}{\pi(\Theta)q(\Theta,\Theta^*)}\right), if\ \rho(X^*, x^{obs}) < \epsilon \\ 0 \qquad\qquad\qquad\qquad\qquad\quad , if\ \rho(X^*, x^{obs}) \geq \epsilon \end{cases} \tag{13}$$

Note that with one simulated data, $P_{X|\Theta}(\rho(X, x^{obs}) < \epsilon|\Theta) = 1$, otherwise the original Θ value was already rejected and had no change to yield new proposals.

Though straightforward, the ABC-MCMC algorithm is more likely to get stuck in low-probability regions of the posterior than the regular MCMC algorithm (Turner and Van Zandt, 2012): That is, the proposed values are rarely accepted. This is because that to have a non-zero acceptance rate, the proposed value must yield close-enough data (see the first condition of Equation 13). Instead of using Equation 13, the proportion $\frac{P_{X|\Theta}(\rho(X^*, x^{obs}) < \epsilon|\Theta^*)}{P_{X|\Theta}(\rho(X, x^{obs}) < \epsilon|\Theta)}$ can be better estimated by simulating multiple data sets. With *M* simulated data sets, $\frac{\sum_{m=1}^{M} \mathbb{I}(\rho(x^{obs}, X_m^*) < \epsilon)}{\sum_{m=1}^{M} \mathbb{I}(\rho(x^{obs}, X_m) < \epsilon)}$ approximates the proportion better and leads to a pseudo marginal MCMC sampler.

2.7 SMC-ABC

To improve the efficiency of ABC, Sequential Monte Carlo (SMC) sampling can be used. SMC builds upon importance sampling[3]: It constructs a sequence of importance distributions bridging the target truncated distribution, which is difficult to sample from, and the prior distribution, which is easy to sample from. At each time of SMC, importance sampling is used such that the particles sampled last time

[3] Importance sampling is a Monte Carlo method that involves drawing samples from a proposal distribution that is easier to sample from, but may not be similar to the target distribution of interest. The samples are then weighted based on how representative they are of the target distribution. The importance weights are used to estimate the expectation or integral of a function with respect to the target distribution.

estimate the intermediate posterior distribution this time. The initial set of particles comes from the prior, and all particles have equal weights. Later on, during iterations, tighten the precision of approximation (i.e., decreasing the value of tolerance ϵ), perturb the particles, and update the importance weights of particles based on how they conform to the stricter tolerance. The procedures of reassigning importance weights and perturbing iterate until the tolerance ϵ falls below some threshold. The final set of particles can be seen as a sample from the approximate posterior distribution. In case that during iteration, the number of alive particles (i.e., particles with non-zero weights) is smaller than some threshold, the set of particles will be resampled according to their weights (dead particles have no chance of being resampled). ABC plays a role when we assign weights to particles. Intuitively, particles that produces close-enough data should receive higher weights. ABC also helps determine if we should accept a perturbed particle value, by examining if perturbed particle produces better simulated data than unperturbed value.

This section discusses the generic SMC algorithm and how to embed ABC with SMC algorithm. This section is the base of the algorithm used in this study (i.e., algorithm 3).

The Sequential Monte Carlo (SMC) samplers are particle-filtering algorithms that can track a sequence of probability distributions $\mathbb{P}_t(d\Theta)$ for $t = 0, 1, 2, 3, \dots, T$. At each time t, the distribution $\mathbb{P}_t(d\Theta)$ can be approximated by a collection of N random samples termed *particles*. The sequence of distributions is chosen by the user. In the application of ABC, the approximated posterior distribution is our final target, which is hard to directly sample from. With the SMC sampler, we can start from a

distribution $\mathbb{P}_0(d\Theta)$ that is easy to sample from (e.g., the prior distribution), and then sequentially move the particles such that they finally can be considered as a sample from the final target distribution $\mathbb{P}_T(d\Theta)$ (e.g., the approximated posterior distribution). For the purpose of estimating one target distribution, the intermediate distributions are not of interest.

2.7.1 Specify the Sequence of Probability Distributions

The sequence of probability distributions is defined on a common measurable space $(\boldsymbol{\Theta}, \mathcal{B}(\boldsymbol{\Theta}))$ (e.g., Del Moral et al., 2006; Chopin & Papaspiliopoulos, 2020). Borrowing the definition and notation of Chopin and Papaspiliopoulos (2020, Chapter 17), the probability distributions are of the form:

$$\mathbb{P}_t(d\Theta) = \frac{\gamma_t(\Theta)}{L_t} \nu(d\Theta) \tag{14}$$

where $\nu(d\Theta)$ is the Lebesgue measure on the space Θ, $\gamma_t(\Theta)$ is an unnormalized density, and L_t is defined as $\int_{\boldsymbol{\Theta}} \gamma_t(\Theta)\, \nu(d\Theta) > 0$.

The expression of approximate posterior distribution (Equation 11, reprinted below) is in the format of Equation 14:

$$\pi(\Theta|\rho(x^{obs}, X) < \epsilon) = \frac{P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)\pi(\Theta)}{\int P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon|\Theta)\pi(\Theta)\, d\Theta}.$$

Equation 11 is the final target distribution (i.e., $\mathbb{P}_T(d\Theta)$) with a sufficiently small value ϵ (i.e., ϵ_T). To get to the final target distribution from an easy-to-sample distribution (e.g., prior), we can artificially construct intermediate distributions by giving a sequence of decreasing ϵ_t (i.e., $\infty \geq \epsilon_0 > \epsilon_1 > \cdots > \epsilon_T = \epsilon$). We specify intermediate distributions by comparing Equations 11 and 14. $\nu(d\Theta)$ in Equation 11

is the Lebesgue measure. $\gamma_t(\Theta)$ in Equation 11 corresponds to the product of prior and the likelihood of Θ given that the simulated data X is close to the observed data x^{obs}. The "closeness" can be defined as $\rho(X, x^{obs}) < \epsilon_t$ that depends on t. L_t in Equation 11 corresponds to the marginal probability that simulated data X and observed data x^{obs} are "close" in terms of $\rho(X, x^{obs}) < \epsilon_t$[4], integrating across all possible Θ values. As t increases and ϵ_t decreases, we get stricter with the definition of closeness, and thus move closer to the approximated posterior distribution (Equation 11).

2.7.2. Transition between Sequential Distributions

Given that we can define a sequence of distributions by decreasing ϵ_t, it is natural to ask (1) What is the relationship between the distributions at time t and $t-1$, and (2) How can we weight the particles from time $t-1$ to t such that they are considered as samples from $\mathbb{P}_{t-1}(d\Theta)$ to $\mathbb{P}_t(d\Theta)$, respectively. As for the relationship between $\mathbb{P}_{t-1}(d\Theta)$ and $\mathbb{P}_t(d\Theta)$, from Equation 14, given a particle value Θ_0, its probability/density at time $t-1$ and its probability/density at time t has the following relationship (Chopin & Papaspiliopoulos, 2020, p. 330):

$$\mathbb{P}_t(d\Theta_0) = \frac{L_{t-1}}{L_t} \frac{\gamma_t(\Theta_0)}{\gamma_{t-1}(\Theta_0)} \mathbb{P}_{t-1}(d\Theta_0). \tag{15}$$

On the right-hand-side of Equation 15, the first ratio, L_{t-1}/L_t, does not depend on the particle value Θ_0. In other words, from time $t-1$ to time t, all *N* particles have the same value on L_{t-1}/L_t. Therefore, we have:

[4] Note that given a distance function, the definition of "closeness" (i.e., ϵ_t) depends on t.

$$\mathbb{P}_t(d\Theta_0) \propto \frac{\gamma_t(\Theta_0)}{\gamma_{t-1}(\Theta_0)} \mathbb{P}_{t-1}(d\Theta_0) \tag{16}$$

where the common normalizing constant L_{t-1}/L_t was omitted. Equation 16 enables us to implement sequential importance sampling (SIS)[5]: it indicates how we are going to reassign the weight for a particle from time $t-1$ to time t. Specifically, we have $W_t^{(n)} \propto W_{t-1}^{(n)} \frac{\mathbb{P}_t(d\Theta^{(n)})}{\mathbb{P}_{t-1}(d\Theta^{(n)})}$, where $W_t^{(n)}$ is the weight at time t for the *n*th particle. Note that both numerator and denominator have $\Theta^{(n)}$ that does not vary with time t. In other words, from time $t-1$ to time t, the particle $\Theta^{(n)}$ was not moved.

The above strategy of sequentially reassigning weights to the *same* set of particles is usually too naïve for problems of interest (Chopin & Papaspiliopoulos, 2020). For example, suppose we have 1000 particles sampled from $\mathbb{P}_0(d\Theta)$, and after *T* times of reassigning weights, only one particle has non-zero weight and can be considered as a sample from $\mathbb{P}_T(d\Theta)$. In this case, the estimated target distribution concentrates on just one value, which is likely to be an inaccurate estimation. Such piked estimated posterior happens especially when $\mathbb{P}_0(d\Theta)$ and $\mathbb{P}_T(d\Theta)$ differ a lot. To better explore the state space and improve the estimates of target distributions, we can propose new particle values based on old values in each iteration/time.

Del Moral et al. (2006) describes how to obtain the incremental importance weight with moved particles. The particles can be moved using any local move, including MCMC moves. The authors proposed an auxiliary variable technique and introduced artificial backward Markov kernels with density $\mathbb{L}_{t-1}(\Theta_t^{(n)}, \Theta_{t-1}^{(n)})$. To get

[5] The reason why we use SIS rather than a simple IS is that when the target distribution is a non-standard high dimensional distribution, it is hard to select an importance distribution that is similar to the target distribution (Del Moral et al., 2006).

incremental weights, they then perform importance sampling between the *joint* target distribution and the *joint* importance distribution. With the auxiliary backward kernels, the importance weight for *n*th particle is given by:

$$W_t^{(n)}\left(\Theta_{1:t}^{(n)}\right) = \frac{\mathbb{P}_t\left(d\Theta_{1:t}^{(n)}\right)}{\eta_t(\Theta_{1:t})} = \frac{\mathbb{P}_t\left(d\Theta_t^{(n)}\right)\prod_{k=1}^{t-1}\mathbb{L}_k(\Theta_{k+1}^{(n)},\Theta_k^{(n)})}{\eta_{t-1}(\Theta_{1:t-1})\mathbb{K}_t(\Theta_{t-1}^{(n)},\Theta_t^{(n)})},$$

where $\eta_t(\Theta_{1:t})$ is the joint importance distribution from which we sample/propose particles. $\mathbb{K}_t$ is the transition kernel at time t. Similarly, we have a similar expression for $W_{t-1}^{(n)}\left(\Theta_{1:t-1}^{(n)}\right)$. The incremental weight is the ratio of $W_t^{(n)}\left(\Theta_{1:t}^{(n)}\right)$ and $W_{t-1}^{(n)}\left(\Theta_{1:t-1}^{(n)}\right)$. Therefore, we have importance weight given by

$$W_t^{(n)} \propto W_{t-1}^{(n)}\frac{\mathbb{P}_t\left(d\Theta_t^{(n)}\right)}{\mathbb{P}_{t-1}\left(d\Theta_{t-1}^{(n)}\right)}\frac{\mathbb{L}_{t-1}\left(\Theta_t^{(n)},\Theta_{t-1}^{(n)}\right)}{\mathbb{K}_t\left(\Theta_{t-1}^{(n)},\Theta_t^{(n)}\right)}. \tag{17}$$

Equation 17 is the general formula for weight update. By carefully choosing the backward and transition kernels, Equation 17 can be simplified.

In practice, to obtain good performance, rather than choosing arbitrarily, we should optimize the backward kernels $\{\mathbb{L}_{t-1}\}$ with respect to the transition kernels $\{\mathbb{K}_t\}$. According to the proposition 1 in Del Moral et al. (2006), the optimal $\{\mathbb{L}_{t-1}\}$ yields unnormalized weights that have the minimal variance over all choices of backward Kernels. However, the optimal kernels are typically impossible in practice, so Del Moral et al. (2006, section 3.3.2) discussed some approximations to the optimal $\{\mathbb{L}_{t-1}\}$. One approximation reduces Equation 17 to the following:

$$W_t^{(n)} \propto W_{t-1}^{(n)}\frac{\mathbb{P}_t\left(d\Theta_t^{(n)}\right)}{\int_{\Theta_{t-1}\in\boldsymbol{\Theta}}\mathbb{P}_{t-1}\left(d\Theta_{t-1}^{(n)}\right)\mathbb{K}_t\left(\Theta_{t-1}^{(n)},\mathrm{d}\Theta_t^{(n)}\right)}. \tag{18}$$

However, Del Moral et al. (2012) shows that in the context of ABC, Equation 18 is computationally complex. They proposed an alternative approach that is easier to compute and perform similarly if $\mathbb{P}_t(d\Theta)$ does not differ significantly from $\mathbb{P}_{t-1}(d\Theta)$ (i.e., in SMC-ABC, $\epsilon_{t-1} - \epsilon_t$ is not too large). This method requires using a transition Markov kernel $\mathbb{K}_t(\Theta_{t-1}, \mathrm{d}\Theta_t)$ leaves invariant $\mathbb{P}_t(d\Theta_\mathrm{t})$. That is,

$$\mathbb{P}_t(d\Theta_\mathrm{t}) = \int_{\Theta_{t-1}\in\boldsymbol{\Theta}} \mathbb{P}_t(d\Theta_{t-1})\mathbb{K}_t(\Theta_{t-1}, \mathrm{d}\Theta_t). \tag{19}$$

With $\mathbb{K}_t$ as an MCMC kernel and $\mathbb{L}_{t-1}$ as the reversal backward kernel, the alternative approach reduces Equation 17 to the following:

$$W_t^{(n)} \propto W_{t-1}^{(n)} \frac{\mathbb{P}_t\left(d\Theta_{t-1}^{(n)}\right)}{\mathbb{P}_{t-1}\left(d\Theta_{t-1}^{(n)}\right)} \propto W_{t-1}^{(n)} \frac{\sum_{m=1}^{M} \mathbb{I}\left\{\rho\left(X_{t-1,m}^{(n)}, x^{obs}\right) < \epsilon_t\right\}}{\sum_{m=1}^{M} \mathbb{I}\left\{\rho\left(X_{t-1,m}^{(n)}, x^{obs}\right) < \epsilon_{t-1}\right\}}, \tag{20}$$

where *M* is the total number of data simulated from *one* particle, and $X_{t-1,m}^{(n)}$ is the *m*th data simulated from particle $\Theta_{t-1}^{(n)}$. For computational convenience, the raw data x^{obs} and $X_{t-1,m}^{(n)}$ can be replaced by their summary statistics. Note that Equation 18 uses the moved/perturbed particle value $\Theta_t^{(\mathrm{n})}$, but Equation 20 uses the unperturbed particle value $\Theta_{t-1}^{(\mathrm{n})}$. The fraction in the right-hand-side of Equation 20 means that, for the *n*th particle, after we get stricter with the definition of closeness (i.e., $\epsilon_{t-1} \rightarrow \epsilon_t$) but before perturbing, how many simulated data of this unperturbed particle remained to be "close" to the observed data, in proportion. Equation 20 was used in this study, with raw data replaced by their summary statistics.

2.7.3. SMC Samplers

This sections first introduces a generic SMC sampler, then introduce how SMC can be utilized to perform ABC. Differences between the generic SMC and the SMC-ABC sampler are discussed.

Generic SMC sampler

Chopin and Papaspiliopoulos (2020, Chapter 17.1) introduce a generic SMC sampler (Algorithm 1). This generic SMC sampler assumes that (1) we are able to compute the ratios $\gamma_t(\Theta_0)/\gamma_{t-1}(\Theta_0)$, and (2) we are able to construct a Markov kernel $\mathbb{K}_t(\Theta_{t-1}, \mathrm{d}\Theta_t)$ that leaves *invariant* $\mathbb{P}_{t-1}(d\Theta_t)$.

Algorithm 1: *Generic SMC Sampler*

Operations involving index n must be performed for $n = 1, \ldots, N$.

I. Initial iteration ($t = 0$)

1 $\Theta_0^{(n)} \sim \pi(d\Theta)$ # Sample

2 $w_0^{(n)} \leftarrow \gamma_0(\Theta_0^{(n)})$ # Obtained unnormalized weights

3 $W_0^{(n)} = w_0^{(n)} / \sum_{n=1}^{N} w_0^{(n)}$ # Normalize weights

II. Later iterations ($1 \leq t \leq T$)

for t in 1:T

4 **if** $ESS(W_{t-1}^{1:N}) < ESS_{min}$, **then**:

5 $A_t^{1:N} \leftarrow resample(W_{t-1}^{1:N})$ # Get indices of resampled particles

6 $\widehat{w}_{t-1}^{(n)} \leftarrow 1$ # Reassign equal weights

else

7 $A_t^{(n)} \leftarrow n$ # Keep the original indices

8 $\widehat{w}_{t-1}^{(n)} \leftarrow w_{t-1}^{(n)}$ # Keep the original weights

9 $\Theta_t^{(n)} \sim \mathbb{K}_t(\Theta_{t-1}^{A_t^{(n)}}, d\Theta_t)$ # Propose/perturb[6]

10 $w_t^{(n)} \leftarrow \widehat{w}_{t-1}^{(n)} \gamma_t(\Theta_t^{(n)}) / \gamma_{t-1}(\Theta_t^{(n)})$ # Update weights

11 $W_t^{(n)} = w_t^{(n)} / \sum_{n=1}^{N} w_t^{(n)}$ # Normalize weights

Note. Adapted from *An introduction to sequential Monte Carlo* (p. 331), by N. Chopin, and O. Papaspiliopoulos, 2020, Springer Nature. Copyright 2020 by Springer Nature Switzerland AG. Adapted with permission from Springer Nature.

[6] The acceptance/rejection is a part of the transition kernel $\mathbb{K}_t$. See Equation 7.2 in Robert et al. (1999), p. 272.

There are several things worth noting in algorithm 1. First, in ABC, with our specification of the sequence of approximate posterior distributions (Section 2.7.1), $\pi(d\Theta)$ in line 1 is the prior distribution. In line 2, $\gamma_0(\Theta) = P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon_0|\Theta)$. If we choose $\epsilon_0 = \infty$, then $\gamma_0(\Theta)$ is a constant, and all particles sampled from the prior have equal weights. In line 10, $\gamma_t(\Theta) = P_{X|\Theta}(\rho(x^{obs}, X) < \epsilon_t|\Theta)$. Algorithm 1 suppose the sequence of ϵ_t is pre-determined, but in a general case of ABC, ϵ_t can be determined adaptively. Algorithm 2 for ABC-SMC sampler explains how to adaptively choose ϵ_t in detail. In short, we choose a stricter tolerance ϵ_t such that a *fixed* proportion of particles is retained at each iteration. The removed particles produce simulated data that are no longer "sufficiently close" to the observed data, after update the tolerance.

Second, in line 4, ESS represents the Effective Sample Size, which counts the weighted number of alive particles. Indeed, we have (Liu 2001, p. 35-36):

$$ESS\left(\left\{W_t^{(n)}\right\}\right) = \left(\sum_{n=1}^{N}\left(W_t^{(n)}\right)^2\right)^{-1}. \tag{21}$$

We resample particles with weights once the ESS is smaller than some pre-decided threshold ESS_{min}. $ESS_{min}$is the minimal weighted number of particles we want for estimating the target posterior distribution.

Third, in line 9 of algorithm 1, the ABC-MCMC scheme (Section 2.6.2) and Equation 13 can be used. Majoram et al. (2003) slightly modified the ABC-MCMC scheme by considering more than one simulated data (i.e., $M > 1$) and use pseudo-marginal MCMC. The modified acceptance rate is:

$$acceptance\ rate = \min\left(1, \frac{\sum_{m=1}^{M} \mathbb{I}(\rho(x^{obs}, X_m^*) < \epsilon_t)}{\sum_{m=1}^{M} \mathbb{I}(\rho(x^{obs}, X_m) < \epsilon_t)} \frac{\pi(\Theta^*) q_t(\Theta^*, \Theta)}{\pi(\Theta) q_t(\Theta, \Theta^*)}\right), \quad (22)$$

where $q_t(\Theta^*, \Theta)$ is the density of proposing Θ from Θ^* at time t, and $\pi(\cdot)$ is the prior distribution. X_m is the mth data simulated from the unperturbed Θ, and the X_m^* is the mth data simulated from proposed Θ^*. The term $\frac{\sum_{m=1}^{M} \mathbb{I}(\rho(x^{obs}, X_m^*) < \epsilon_t)}{\sum_{m=1}^{M} \mathbb{I}(\rho(x^{obs}, X_m) < \epsilon_t)}$ in Equation 22 better approximates $\frac{P_{X|\Theta}(\rho(X^*, x^{obs}) < \epsilon | \Theta^*)}{P_{X|\Theta}(\rho(X, x^{obs}) < \epsilon | \Theta)}$ in Equation 12 than Equation 13. Comparing Equation 13 and 22, when quantifying "whether the proposed particle value produces simulated data closer to the observed data than the original value", Equation 13 uses just two values (i.e., 0 and 1) to approximate, but Equation 22 uses a continuous ratio between 0 and 1. The larger M is, the finer the approximation. Equation 22 helps construct a pseudo-marginal MCMC. Note that the proposal distribution q_t at different times may be different. The parameters of $q_t(\Theta, \cdot)$ can be determined adaptively based on the previous estimation of $\mathbb{P}_{t-1}$ (e.g., Andrieu & Johanes, 2008; Beaumont et al., 2009; Del Moral, et al., 2012). For example, if we use random walk as the proposal distribution at time t, the means can be the old particle values $\Theta_{t-1}^{(n)}$, and the (co)variances may be determined based on the variance of all weighted particles at time $t-1$.

Fourth, the incremental weight in line 10 is from Equation 18, which is the optimal choice. This incremental weight is based on the perturbed particle value. In the context of ABC, for easier computation, we can use Equation 20 in place, which used the unperturbed particle value.

ABC-SMC Sampler

Embedding ABC into the generic SMC sampler and relaxing some assumptions, the ABC-SMC sampler is described in Algorithm 2 (summarized from Del Moral et al., 2012 and Chopin & O. Papaspiliopoulos, 2020). Similar to the generic SMC algorithm, it requires inputting a resampling threshold ESS_{min}, and the number of particles N. Different from the generic SMC sampler, we have the following specifications.

First, algorithm 1 requires a pre-defined sequence of tolerance, $\epsilon_0, \dots, \epsilon_T$, such that we can compute the ratios $\frac{\gamma_t(\Theta_0)}{\gamma_{t-1}(\Theta_0)}$. In contrast, algorithm 2 only requires ϵ_0 (line 3, $\epsilon_0 = \infty$) and ϵ_T (line 6), and decides intermediate tolerance values (i.e., $\epsilon_1, \dots, \epsilon_{T-1}$) on-the-fly (line 7-11). ϵ_T describes the *sufficiently* close distance between observed and simulated data. A good ϵ_T indicates that the approximated posterior distribution (Equation 11) is close to the likelihood-based posterior distribution (Equation 10). The intermediate values are determined by shrinking ESS by s each time (i.e., $ESS_t = s \times ESS_{t-1}$, line 7). At each time/iteration, we find a smaller tolerance value such that a small proportion (i.e., $(1 - s) \times 100\%$) of particles have their weights decreased from positive to zero. The shrinkage rate s should be close to 1 (e.g., 0.8, 0.9), such that $\mathbb{P}_t(d\Theta)$ does not differ from $\mathbb{P}_{t-1}(d\Theta)$ a lot, which is the prerequisite for using Equation 20 (line 8) in algorithm 2. Both the shrinkage rate and ϵ_T affect the number of iterations *T*. *T* is not predetermined.

Second, line 22-26 in algorithm 2 expands line 9 in algorithm 1. In algorithm 2, we use a convenient Gaussian random walk as the proposal distribution, and we

determine its parameters on-the-fly (line 22). The proposal will be accepted with some rate (line 25-26).

Third, algorithm 1 updates particle weights after perturbing particles (line 10 in algorithm 1). In contrast, algorithm 2 updates particle weights before perturbing particles (line 8 in algorithm 2). The is because that algorithm 1 uses the optimal solution expressed in Equation 18, while algorithm 2 uses the solution expressed in Equation 20 for computational simplicity.

Algorithm 2: *ABC-SMC Sampler*

`Operations involving index` n `must be performed for` $n = 1, \ldots, N$ `unless specified.`

I. Initial iteration ($t = 0$)

$\Theta_0^{(n)} \sim \pi(d\Theta)$ # Sample from prior

$X_{1:M}^{(n)} \sim f(\cdot\,|\Theta_0^{(n)})$ # Simulate *M* data from a particle

$W_0^{(n)} = 1/N$ # Assign equal weights

II. Later iterations ($1 \le t \le T$)

$t = 1$

while True:

0. Break loop if $\epsilon_{t-1} = \epsilon_T$, where ϵ_T is predefined.

1. Determine ϵ_t by solving[7] $ESS\left(\left\{W_t^{(n)}\right\}, \epsilon_t\right) = s \times ESS\left(\left\{W_{t-1}^{(n)}\right\}, \epsilon_{t-1}\right)$.

$$W_t^{(n)} \propto W_{t-1}^{(n)} \frac{\sum_{m=1}^{M} \mathbb{I}\left\{\rho\left(X_{t-1,m}^{(n)}, x^{obs}\right) < \epsilon_t\right\}}{\sum_{m=1}^{M} \mathbb{I}\left\{\rho\left(X_{t-1,m}^{(n)}, x^{obs}\right) < \epsilon_{t-1}\right\}}$$

$$ESS\left(\left\{W_t^{(n)}\right\}\right) = \left(\sum_{n=1}^{N} \left(W_t^{(n)}\right)^2\right)^{-1}$$

if $\epsilon_t < \epsilon_T$: $\epsilon_t \leftarrow \epsilon_T$.

Record ϵ_t and normalized weights $W_t^{(n)}$.

2. Resample *N* particles if needed.

if $ESS(W_t^{1:N}) < ESS_{min}$:

  $A_t^{1:N} \leftarrow resample(W_t^{1:N})$ # Get indices of resampled particles

  $W_t^{(n)} \leftarrow 1/N$ # Reassign equal weights

else

  $A_t^{(n)} \leftarrow n$ # Keep the original indices

  $W_t^{(n)} \leftarrow W_t^{(n)}$ # Keep the original normalized weights

$\Theta_{t-1}^{(n)} \leftarrow \Theta_{t-1}^{A_t^{(n)}}$; $X_{t-1,1:M}^{(n)} \leftarrow X_{t-1,1:M}^{A_t^{(n)}}$ # Denote abusively[8]

3. Propose new particle values for alive particles and accept with some rate.

for *n*th particle that satisfies $W_t^{(n)} > 0$:

  $\sum_t = 2 \times cov(\Theta_{t-1}^{1:N})$ # Specify covariance[9] of $q_t(\Theta_{t-1}^{(n)}, \cdot)$

  $\Theta_t^{(n)*} \sim q_t(\Theta_{t-1}^{(n)}, d\Theta_t)$ # Propose/perturb based on old values

  $X_{t,1:M}^{(n)*} \sim f\left(\cdot \,\middle|\, \Theta_t^{(n)*}\right)$ # Simulate *M* data from proposed value

  $R = \min\left(1, \frac{\sum_{m=1}^{M} \mathbb{I}\left(\rho\left(x^{obs}, X_{t,m}^{(n)*}\right) < \epsilon_t\right)}{\sum_{m=1}^{M} \mathbb{I}\left(\rho\left(x^{obs}, X_{t-1,m}^{(n)}\right) < \epsilon_t\right)} \frac{\pi\left(\Theta_t^{(n)*}\right) q_t\left(\Theta_t^{(n)*}, \Theta_{t-1}^{(n)}\right)}{\pi\left(\Theta_{t-1}^{(n)}\right) q_t\left(\Theta_{t-1}^{(n)}, \Theta_t^{(n)*}\right)}\right)$ # Acceptance rate

  $u \sim unif(0,1)$; **if** $u < R$: $\Theta_t^{(n)} \leftarrow \Theta_t^{(n)*}$; **else** $\Theta_t^{(n)} \leftarrow \Theta_{t-1}^{(n)}$. # Accept or not

for *n*th particle that satisfies $W_t^{(n)} = 0$:

  $\Theta_t^{(n)} \leftarrow \Theta_{t-1}^{(n)}$; $X_{t,1:M}^{(n)} \leftarrow X_{t-1,1:M}^{(n)}$

4. Proceed

$t = t + 1$

---

[7] Given ϵ_{t-1} and $\forall \epsilon_t > 0$, we can use bisection method to find ϵ_t.

[8] If not resampled, this step means doing nothing, due to $A_t^{(n)} \leftarrow n$.

[9] Here we use multivariate normal distribution as the proposal distribution. The constant 2 is heuristic.

As illustrated throughout this section, SMC-ABC has advantages over ABC-MCMC on helping us better estimate the approximate posterior distribution. We alleviate the problem of high rejection rates caused by directly applying a small tolerance ϵ. An adaptive schedule enables us to decompose the problem of estimating posteriors into a series of simpler sub-problems. Therefore, we can approach the posterior gradually from a prior that may be quite different from the posterior. The next section introduces the algorithm for estimating Q-diffusion models in detail.

Chapter 3: Methods

3.1 ABC-SMC Sampler for Q-diffusion Model

When the number of interested parameters is large, algorithm 2 (SMC-ABC) has a problem of low acceptance rate. Therefore, an adapted version (algorithm 3, adapted SMC-ABC) will be used. In algorithm 2, all parameters need to be perturbed together (line 23), such that we can simulate data and calculate the acceptance rate (line 25). However, with many interested parameters, the probability of perturbing all parameters in right directions is low, which results in low acceptance rate and rare movement of particles. If we do not sufficiently move particles, once the effective sample size drops below the threshold, resampling is just duplicating good particles. Such duplication without movement may result in an inaccurate posterior that pikes only around several good particle values. Algorithm 3 differs from algorithm 2 in section #3 (line 18-29). Algorithm 3 keeps perturbing particles until the final set of particles differs from the unperturbed particles to some extent (i.e., sufficient movement, line 28). Algorithm 3 also changed the way we decide parameters for the proposal distribution (line 22), to increase the chance of perturbing all parameters in right directions. Specifically, we decide the covariance matrix of proposal distribution by using the covariance of *30 % neighboring* alive particles.

The idea of perturbing multiple times come from Buchholz et al. (2021). They suggest continuing perturbing until a large fraction (e.g., 90%) of the product of the first order autocorrelation drops below a threshold (e.g., $r = 0.1$). At time t, we originally have *N* particles (i.e., rows). Some particles have zero weight and are considered dead. The particle is multidimensional, and one particle is a vector

containing all interested parameters (e.g., $3 \times J + 2$ parameters). We perturb a particle by proposing a new vector (e.g., $3 \times J + 2$ new parameter values) and accept the proposal with some rate. After perturbing once, we calculate the column-wise autocorrelation between two matrices (excluding dead particles) and have $3 \times J + 2$ correlation coefficients. One coefficient describes how close the particles are before and after perturbing, in one dimension. We keep perturbing and calculating the element-wise product of autocorrelations (i.e., autocorrelation 1 × autocorrelation 2 × ……, for each of the $3 \times J + 2$ dimensions), until 90% of the product of autocorrelations are below some threshold. The rest part of Section 3.1 goes over algorithm 3 in detail for Q-diffusion models. Table 1 lists the tuning aspects of algorithm 3 and the choices in this study. Figure 2 is an illustration of the SMC-ABC algorithm.

Algorithm 3: *Adapted ABC-SMC Sampler for Q-diffusion Model*

`Operations involving index n must be performed for n = 1,…,N unless specified.`

I. Initial iteration ($t = 0$)

$\boldsymbol{\Lambda}_0^{(n)} \sim \pi(d\boldsymbol{\Lambda})$ # Sample from prior

$X_{1:M}^{(n)} \sim f(\cdot\,|\boldsymbol{\Lambda}_0^{(n)})$ # Simulate M data from a particle

$W_0^{(n)} = 1/N$ # Assign equal weights

II. Later iterations ($1 \le t \le T$)

$t = 1$

while True:

 # 0. Break loop if $\boldsymbol{\epsilon_{t-1}} = \boldsymbol{\epsilon_T}$, where $\boldsymbol{\epsilon_T}$ is predefined.

 # 1. Determine $\boldsymbol{\epsilon_t}$ by solving $ESS\left(\left\{W_t^{(n)}\right\}, \boldsymbol{\epsilon_t}\right) = s \times ESS\left(\left\{W_{t-1}^{(n)}\right\}, \boldsymbol{\epsilon_{t-1}}\right)$.

if $\boldsymbol{\epsilon_t} < \boldsymbol{\epsilon_T}$: $\boldsymbol{\epsilon_t} \leftarrow \boldsymbol{\epsilon_T}$.

Record $\boldsymbol{\epsilon_t}$ and normalized weights $W_t^{(n)}$.

 # 2. Resample N particles if needed.

if $ESS(W_t^{1:N}) < ESS_{min}$:

  $A_t^{1:N} \leftarrow resample(W_t^{1:N})$ # Get indices of resampled particles

  $W_t^{(n)} \leftarrow 1/N$ # Reassign equal weights

else

  $A_t^{(n)} \leftarrow n$ # Keep the original indices

  $W_t^{(n)} \leftarrow W_t^{(n)}$ # Keep the original normalized weights

$\boldsymbol{\Lambda}_{t-1}^{(n)} \leftarrow \boldsymbol{\Lambda}_{t-1}^{A_t^{(n)}}$; $X_{t-1,1:M}^{(n)} \leftarrow X_{t-1,1:M}^{A_t^{(n)}}$ # Denote abusively

 # 3. Repeatedly proposing new particle values for alive particles and accept with some rate, until movement is sufficient.

$k = 1$; $\boldsymbol{\lambda}_{k-1}^{(n)} \leftarrow \boldsymbol{\Lambda}_{t-1}^{(n)}$; $x_{k-1,1:M}^{(n)} \leftarrow X_{t-1,1:M}^{(n)}$ # Create temporary storage

while True:

  for nth particle that satisfies $W_t^{(n)} > 0$:

    $\sum_{t,k}^{(n)} = 0.1 \times cov(\boldsymbol{\lambda}_{k-1}^{neighbor\_ind_{(n)}})$ # Specify covariance of $q_{t,k}^{(n)}(\lambda_{k-1}^{(n)}, \cdot)$

    $\boldsymbol{\lambda}_k^{(n)*} \sim q_{t,k}^{(n)}(\boldsymbol{\lambda}_{k-1}^{(n)}, d\boldsymbol{\lambda}_k)$ # Propose/perturb based on old values

    $x_{k,1:M}^{(n)*} \sim f(\cdot\,|\boldsymbol{\lambda}_k^{(n)*})$ # Simulate M data from proposed value

    $R = \min\left(1, \frac{\sum_{m=1}^{M} \mathbb{I}\left(\rho\left(x^{obs}, x_{k,m}^{(n)*}\right) < \epsilon_t\right)}{\sum_{m=1}^{M} \mathbb{I}\left(\rho\left(x^{obs}, x_{k-1,m}^{(n)}\right) < \epsilon_t\right)} \frac{\pi\left(\boldsymbol{\lambda}_k^{(n)*}\right) q_t\left(\boldsymbol{\lambda}_k^{(n)*}, \boldsymbol{\lambda}_{k-1}^{(n)}\right)}{\pi\left(\boldsymbol{\lambda}_{k-1}^{(n)}\right) q_t\left(\boldsymbol{\lambda}_{k-1}^{(n)}, \boldsymbol{\lambda}_k^{(n)*}\right)}\right)$ # Acceptance rate

    $u \sim unif(0,1)$; **if** $u < R$: $\boldsymbol{\lambda}_k^{(n)} \leftarrow \boldsymbol{\lambda}_k^{(n)*}$; **else** $\boldsymbol{\lambda}_k^{(n)} \leftarrow \boldsymbol{\lambda}_{k-1}^{(n)}$. # Accept or not

  $\boldsymbol{r}_k = column\_wise\_corr(\boldsymbol{\lambda}_{k-1}^{alive\_ind}, \boldsymbol{\lambda}_k^{alive\_ind})$ # Column-wise correlation[10]

  if ≥ 90% elements[11] of $\prod_{\mathbb{k}=1}^{k-1} \boldsymbol{r}_{\mathbb{k}} < r$: **break; else:** $k = k + 1$.

$\boldsymbol{\Lambda}_t^{(n)} \leftarrow \boldsymbol{\lambda}_k^{(n)}$; $X_{t,1:M}^{(n)} \leftarrow x_{k,1:M}^{(n)}$ # Update particle values at time t

for nth particle that satisfies $W_t^{(n)} = 0$:

  $\boldsymbol{\Lambda}_t^{(n)} \leftarrow \boldsymbol{\Lambda}_{t-1}^{(n)}$; $X_{t,1:M}^{(n)} \leftarrow X_{t-1,1:M}^{(n)}$

 # 4. Proceed

$t = t + 1$

Notes. (1) The covariance matrix is calculated based on the neighbors of the nth particle. $neighbor\_ind_{(n)}$ specifies the indices of those neighbors. (2) $alive\_ind$ is the indices of alive particles in the current iteration.

[10] For example, if $\boldsymbol{\lambda}_{k-1}^{1:N}$ has 32 columns, then the vector $\boldsymbol{r}_k$ has 32 elements.

[11] $\prod_{\mathbb{k}=1}^{k-1} \boldsymbol{r}_{\mathbb{k}}$ is the element-wise product.

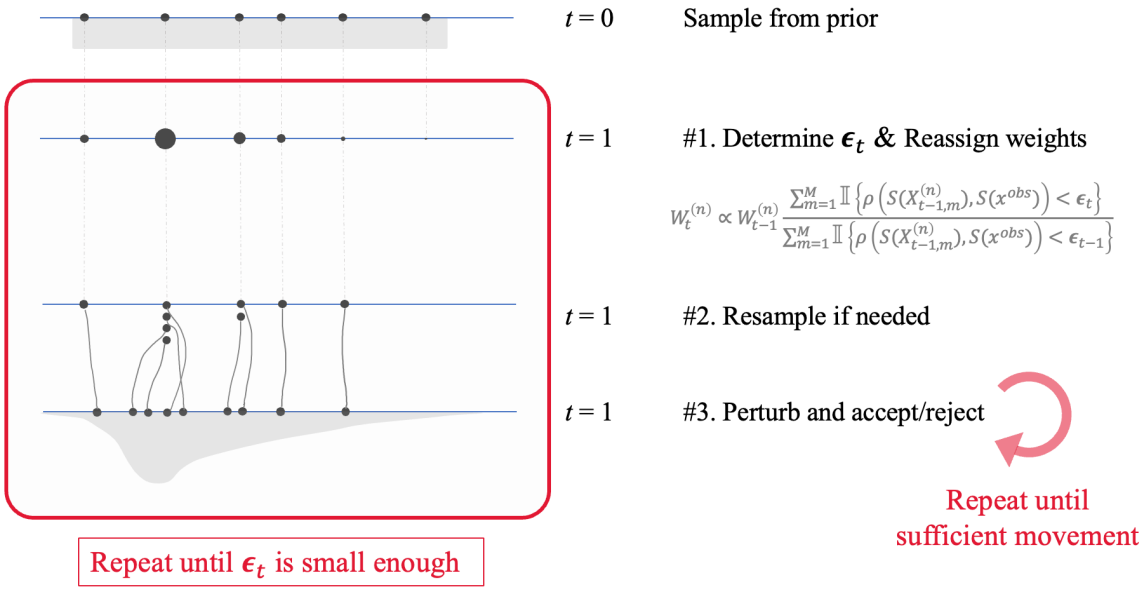
Table 1

Tuning Aspects of Algorithm 3 and the Choices in this Study

Tuning Aspects	Choices in this Study
Number of simulated data for one particle (M, line 2)	10
Total number of particles (N, line 3)	600
Shrinkage rate (s, line 7)	0.8
Minimum effective sample size (ESS_{min}, line 11)	280
Multiplier for deciding the covariance matrix of proposal (line 22)	0.1
Percentage of neighboring alive particles (line 22)	30%
Threshold for the product of autocorrelation[12] (r, line 28)	0.843
Percentage of interested parameters that have a product of autocorrelation below the threshold (line 28)	90%

Figure 2

An Illustration of the Adapted ABC-SMC Algorithm (Algorithm 3)



[12] The threshold for the product of autocorrelation is 0.843 (i.e., $r = 0.843$), which is lenient. The purpose of keeping perturbing particles is to avoid insufficient movement or high correlation between two times of resampling. We aimed to achieve a correlation no greater than 0.5 between resampling. With a shrinkage rate of 0.8 and a minimum effective sample size of 280, we have three[12] or four chances of perturbing particles between resampling ($600 \times 0.8^4 = 245.8 < 280$; $600 \times 0.8^3 = 307.2 > 280$). Therefore, we aim to achieve a correlation of $\sqrt[4]{0.5} = 0.843$ or smaller after each time/iteration. In one iteration, we keep perturbing until 90% elements have the product of autocorrelation smaller than 0.843.

Note. Figure was redrawn from "Multivariate and Multiscale Data Assimilation in Terrestrial Systems: A Review," by Montzka, C., Pauwels, V. R., Franssen, H. J. H., Han, X., and Vereecken, H., 2012, *Sensors, 12*(12), p. 16299. Copyright 2012 by Sensors (14248220).

3.1.1 Before Starting

Simulate Observed Data

Q-diffusion model produces responses and response times. We denote the observed by $x^{obs} = \left(\mathbb{Y}^{\text{obs}}, \mathbb{T}^{\text{obs}}\right)$, where $\mathbb{Y}^{\text{obs}}$ is the observed response data, and $\mathbb{T}^{\text{obs}}$ is the observed response time data. Both $\mathbb{Y}^{\text{obs}}$ and $\mathbb{T}^{\text{obs}}$ are of size $I \times J$, where I is the total number of persons, and J is the total number of items. In this study, we use $I = 200$ persons and $J = 5$ items.

True Parameters. The data generating parameters follow the settings in previous studies. The true item drift rates and boundary separation come from exponential uniform distributions, and the true person drift rates and boundaries come from lognormal distributions (e.g., Molenaar et al., 2015; Jordan, 2020). To resembles real-life scenarios, we constrain item time pressure to be equal. This is because that in a test where questions are of similar format, the time pressure on each item is similar. Molenaar et al. (2015) also shows that when fitting the Q-diffusion model to a 10-item mental rotation test, the model constraining equal item time pressure fits the data best. Table 2 summarizes the true model parameters used in this study. Note that the person drift rate b is non-negative, which is consistent with the Q-diffusion model assumption of positive ability. The number of options was set to 2 (i.e., $O_j = 2$, for $j = 1, \dots, J$).

Table 2

True Model Parameters for Generating Observed Data[13]

	α	β_j	τ_j
Item 1	0.607	1.517	0.592
Item 2	0.607	2.055	0.686
Item 3	0.607	1.000	0.846
Item 4	0.607	1.353	0.897
Item 5	0.607	1.158	1.039
Population scale on person caution (σ_a)	0.3		
Population scale on person caution (σ_b)	0.3		

Data Generation Mechanism. Responses of 200 examinees were directly generated using the marginal probability $P(Y = 1|A, B) = 1/(1 + e^{-AB})$, where the boundary separation is given by $\mathrm{A} = a/\alpha$, and the drift rate is $B = b/\beta$. Response times include a residual part (e.g., time for click mouse and made a response) and a non-residual part (i.e., decision time). The non-residual response times were generated from its conditional density, $f_{T,Y}(T = t|Y = 0)$ or $f_{T,Y}(t|Y = 1)$, using rejection sampling (Tuerlinckx et al., 2001). Rejection sampling does not involve simulating a sample path as Figure 1 shows. Instead, the idea is to directly simulate the decision time that it would take for the process to travel a fixed distance from its starting point. For more details about rejection sampling, please refer to Tuerlinckx et al. (2001, p. 446). However, the rejection sampling is not efficient if boundary separation A and/or drift rate B is large (see Figure 1 and Equation 12 in Tuerlinckx et al., 2001, p. 447.). When the acceptance rate is below 1/1000, we simulate decision time from a sample path. When the boundary separation A is large and the drift rate B

[13] The common item time pressure, α, is the median of distribution $exp\big(unif(-1,0)\big)$. The item drift rates, β_j, were randomly generated from $exp\big(unif(0,1)\big)$. The residual times, τ_j, were randomly generated from $unif(0.5,1,5)$ (Molenaar et al., 2015). According to σ_a and σ_b, the true nuisance person parameters, a and b, were randomly generated from $lognormal(0, 0.3^2)$.

is close to 0, the path may travel a long time to hit a boundary. To prevent endlessly simulating such paths, we cropped decision time at 100, if by that time the process still has not hit a boundary. This limit is quite large compared with the simulated decision times. Data were simulated using Python 3.8 (Van Rossum & Drake, 2009). In one observed response data of size 5×200, the passing rates ranges from 0.65 to 0.85. The mean response time ranges from 1.25 to 1.72 time units[14].

Parameters of Interest

The Q-diffusion model with constrained item time pressure has $2 \times J + 3$ model parameters (i.e., $\log(\alpha), \log(\boldsymbol{\beta}), \boldsymbol{\tau}, \log(\sigma_a), \log(\sigma_b)$). α represents the common item boundary separation (i.e., item time pressure). $\boldsymbol{\beta}$ represents item drift rates (i.e., pure item difficulties) for J items. $\boldsymbol{\tau}$ represents the item residual times for J items. σ_a represents the population scale in person boundary separation (i.e., caution), and σ_b represents the population scale in person drift rate (i.e., power). All parameters are nonnegative. For convenience, we log transformed α, $\boldsymbol{\beta}$, σ_a, and σ_b. We treat all parameters together as a particle $\boldsymbol{\Lambda} = [\log(\alpha), \log(\boldsymbol{\beta}^T), \boldsymbol{\tau}^T, \log(\sigma_a), \log(\sigma_b)]^T$.

The model with parameters $\log(\alpha), \log(\boldsymbol{\beta}), \boldsymbol{\tau}, \log(\sigma_a), \log(\sigma_b)$ is identified, as long as the volatility V and means of log person caution and power are fixed (see

[14] Fixing volatility at 1, the definition of time unit affects the scales on which we put drift rate and boundary separation. For example, the following three scenarios yielded equivalent diffusion processes: (1) time unit is second, $V = 1$, $B = 2$, $A = 2$; (2) time unit is minute, $V = \sqrt{60}$, $B = 2 \times 60$, $A = 2 \times 1$; (3) time unit is minute, $V = \frac{\sqrt{60}}{\sqrt{60}} = 1$, $B = 2 \times 60/\sqrt{60}$, $A = 2/\sqrt{60}$. (2) is an intermediate scenario connecting (1) and (3) that have the same $V = 1$. (1) and (2) are equivalent due to the property that for any $0 < s < t$, $\mathbb{X}(t) - \mathbb{X}(s) \sim N((t-s)B, V^2(t-s))$. Specifically, $\mathbb{X}(t) - \mathbb{X}(t-60) \sim N(60B, 60V^2)$. (2) and (3) are equivalent because multiplying A, B, V, and Z remain the joint density (Equations 1 and 2) unchanged. Note that Z is a function of A (e.g., $Z = A/2$ for unbiased processes).

Equations 1 and 2). In this study, consistent with Molenaar et al. (2015b), we fixed volatility as 1, and fixed the means of log person caution and power as 0.

Distance Functions

We use Equation 23 to calculate the distance between two matrices $\mathbb{A}$ and $\mathbb{B}$.

$$\rho(\mathbb{A},\mathbb{B}) = \sqrt{\frac{\sum_{k=1}^{K}\sum_{l=1}^{L}(\mathbb{A}_{kl}-\mathbb{B}_{kl})^2}{K\times L}} \tag{23}$$

where $\mathbb{A}$ and $\mathbb{B}$ have K rows and L columns.

This study explores the performance of two criteria: the limited information criterion and the enhanced limited information criterion. The limited information distance between simulated and observed data is a vector containing three elements: the distance between (co)variances of response, the distance between (co)variances of log response time, and the distance between item-wise mean log response time. Specifically, they are $\rho(vech(cov(\mathbb{Y})), vech(cov\ (\mathbb{Y}^{\text{obs}})))$, $\rho(vech(cov\ (\log\mathbb{T})), vech(cov(\log\mathbb{T}^{\text{obs}})))$, and $\rho(\mathrm{C}\cdot\log\mathbb{T}, \mathrm{C}\cdot\log\mathbb{T}^{\text{obs}})$. $vech(\cdot)$ is the half-vectorization of a symmetric matrix (e.g., $vech(cov(\cdot))$ is the lower triangle elements of a covariance matrix, *including* diagonal elements). $C = [\frac{1}{I}, \frac{1}{I}, \ldots, \frac{1}{I}]$ is a $1\times I$ matrix, and for example, $\mathrm{C}\cdot\mathbb{T}$ contains the column means of $\mathbb{T}$. We did not use the column means of response data, because it includes redundant information in addition to the covariance matrices of response data[15].

The enhanced limited-information criterion further considers the row-wise relationship between response and response time matrix (i.e., the relationship between

[15] For Bernoulli distribution $Bernoulli(p)$, we have mean p, and variance $p(1-p)$.

a person's response and his/her response time). Specifically, it compares the distance between (co)variances of response, the distance between covariances of log response time conditioning on response, the distance between variances of log response time conditioning on response, and the distance between item-wise mean log response time conditioning on response. The enhanced limited information criterion includes four elements: $\rho(vech(cov(\mathbb{Y})), vech(cov\ (\mathbb{Y}^{\text{obs}})))$,

$\rho(ltri(cov\ (\log\mathbb{T}|\mathbb{Y})), ltri(cov(\log\mathbb{T}^{\text{obs}}|\mathbb{Y}^{\text{obs}})))$,

$\rho(var\ (\log\mathbb{T}|\mathbb{Y}), var(\log\mathbb{T}^{\text{obs}}|\mathbb{Y}^{\text{obs}}))$, and $\rho(\mathrm{C}\cdot\log\mathbb{T}|\mathbb{Y}, \mathrm{C}\cdot\log\mathbb{T}^{\text{obs}}|\mathbb{Y}^{\text{obs}})$. $ltri(\cdot)$ takes the lower triangle elements of a matrix, *excluding* the diagonal elements. The reason why we separate the covariances and variances of conditional log response time is that the number of events being conditioned upon differs. When considering covariances between item *i* and *j*, we must condition upon four different events (i.e., $[Y_i, Y_j] = [0,0], [0,1], [1,0], [1,1]$). When considering variance of item *i*, we condition upon two events (i.e., $Y_i = 0,1$).

It is hypothesized that the enhanced limited-information criterion produces better estimates of log population scale parameters. A simple example illustrates the idea. Consider data 1:

$$Y = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \qquad T = \begin{bmatrix} 10 & 10 \\ 1 & 1 \end{bmatrix}.$$

The first person answered more questions correctly than the second person, but the first person spent more time working on questions. We are not sure what if the second person also spent longer time. It is less likely that the between-person variance on power is large. Consider data 2:

$$Y = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 1 \\ 10 & 10 \end{bmatrix}.$$

The first person outperformed the second person, with even shorter time working on questions. It is more likely that the between-person variance on power is large. The original limited-information criterion cannot distinguish data 1 and data 2, but the enhanced limited-information criterion can.

3.1.2 Initial Iteration

Prior Distributions

The first set of particles come from the prior distributions (line 1). We use weakly informative normal distributions as priors for $\log(\alpha)$, $\log(\boldsymbol{\beta})$, $\log(\sigma_a)$, and $\log(\sigma_b)$, and use uniform distributions as the prior for $\boldsymbol{\tau}$. The prior distributions are:

$$\log(\alpha) \sim N(\log(\alpha^{guessed}), 1)$$
$$\log(\beta_j) \sim N(\log(\beta_j^{guessed}), 1)$$
$$\tau_j \sim uniform(0, \tau_j^{guessed})$$
$$\log(\sigma_a) \sim N(\log(0.3), 1)$$
$$\log(\sigma_b) \sim N(\log(0.3), 1)$$

The normal priors have a standard deviation of 1, which is large enough to cover realistic data generating parameters. The prior location parameters of $\log(\sigma_a)$ and $\log(\sigma_b)$ were chosen as $\log(0.3)$ for convenience, given that a standard deviation of 1 is very large such that realistic values around $\log(0.3)$ have similar prior densities[16]. The prior location parameters of α, $log\beta_j$, and τ_j depends on the initial

[16] With $\log(\sigma_a) \sim N(\log(0.3), 1)$, 68% σ_a values fall between 0.110 and 0.815. 95% σ_a values fall between 0.042 to 2.130. For $\sigma_a = 2.130$, 95% person caution parameter (i.e., *a*) ranges from 0.015 to 65.027, which is very unrealistic considering the range of item boundary separation parameters. For $\sigma_a = 0.815$, 95% person caution parameter (i.e., *a*) ranges from 0.202 to 4.940, which is less unrealistic.

guess of parameter values. From observed response and response time data of item j, we can guess α_j, β_j, and τ_j by using Equations 24, 25, and 26 (see derivations in Appendix A):

$$\tau_j^{guesssed} \approx min\left(\mathbb{T}_j^{\text{obs}}\right) \tag{24}$$

$$\alpha_j^{guessed}\beta_j^{guessed} \approx \frac{e^{\mu+\frac{\sigma^2}{2}}}{logitP\left(\mathbb{Y}_j^{\text{obs}}=1\right)} \tag{25}$$

$$\frac{\alpha_j^{guessed}}{\beta_j^{guessed}} \approx \frac{e^{\mu\prime+\frac{\sigma^2}{2}}(2P\left(\mathbb{Y}_j^{\text{obs}}=1\right)-1)}{2\times\left(E\left(\mathbb{T}_j^{\text{obs}}\right)-\tau_j^{guessed}\right)} \tag{26}$$

where μ and σ are the location and scale for $\theta = ab$, and μ' and σ are location and scale for $\gamma = a/b$. Given that a and b both comes from $lognormal(0,0.3^2)$, we have $\mu = \mu' = 0$ and $\sigma^2 = 0.3^2 + 0.3^2$. Since we constrain all items to have the same time pressure (i.e., equal α_j), we take the mean of $\alpha_j^{guessed}$ and use it as the common guessed item time pressure (i.e., $\alpha^{guessed}$).

Data Simulation

ABC-SMC sampler requires using particle values to simulate data (line 2). Similar to how we generated the observed data, we use marginal probability to generate responses, and use rejection sampling or sample paths to generate response times. For generating response times, we choose to simulate sample paths when the rejection sampling has an acceptance rate below 0.01. For sample paths that have not reached a boundary until time $t = 1.05 \times \max\left(\mathbb{T}^{\text{obs}}\right)$, we stop simulating sample paths and assign the decision time as $1.05 \times \max\left(\mathbb{T}^{\text{obs}}\right)$.

3.1.3 Later iterations

Deciding the Final Tolerance ϵ_T

ϵ_T defines sufficient closeness between data generated from the same parameters, and it decides when to stop the ABC-SMC sampler (line 6). ϵ_T is supposed to be close to zero, but smaller ϵ_T requires more iterations in the ABC-SMC algorithm. We decide ϵ_T by empirically examining how small it should be to distinguish data generated from incorrect item parameters from those generated from true item parameters.

Table 3 summaries the steps for deciding ϵ_T. In step 1, we generate "close-enough" data, and get their distances from the observed data. In ideal cases, we use true parameters (i.e., $\mathbf{\Lambda}^{true}$). In empirical cases, we use the initially guessed parameters (i.e., $\mathbf{\Lambda}^{guessed\ true}$). In both ideal and empirical cases in step 1, we use person parameters randomly generated from $a \sim lognormal(0, 0.3^2)$, and $b \sim lognormal(0, 0.3^2)$. Each of the 5,000 data uses a different set of person parameters. In step 2, we generate data with different model parameters and get their distances from the observed data. We simulate 5,000 data using 5,000 sets of randomly generated item parameters (i.e., $\mathbf{\Lambda}^{imputed}$ or $\mathbf{\Lambda}^{imputed\prime}$). $\mathbf{\Lambda}^{imputed}$ and $\mathbf{\Lambda}^{imputed\prime}$ should be on the same scale of $\mathbf{\Lambda}^{true}$, and be different from $\mathbf{\Lambda}^{true}$. In the ideal case, we obtain $\mathbf{\Lambda}^{imputed}$ in the same way as we obtain $\mathbf{\Lambda}^{true}$. Specifically, we use $\beta^{imputed} \sim exp\big(unif(0,1)\big)$, and $\tau^{imputed} \sim unif(0.5,1.5)$. In empirical case, a plausible way is to generate $\mathbf{\Lambda}^{imputed\prime}$ from uniform distributions with guessed bounds. Specially, in the empirical case, we use $\beta^{imputed\prime} \sim uniform(\min(\boldsymbol{\beta}^{guessed}), \max(\boldsymbol{\beta}^{guessed}))$, and

$\tau^{imputed\prime} \sim uniform(\min(\boldsymbol{\tau}^{guessed}), \max(\boldsymbol{\tau}^{guessed}))$. $\alpha^{imputed\prime} = \alpha^{guessed}$ since we only have one value for $\alpha^{guessed}$. In both ideal and empirical cases in step 2, we use person parameters randomly generated from $a \sim lognormal(0, 0.3^2)$, and $b \sim lognormal(0, 0.3^2)$, which is the same as step 1. Besides, the imputed item boundary separation in step 2 is the same as the true/guessed item boundary separation in step 1. Having some parameters in step 2 the same as that in step 1 will result in smaller and stricter $\boldsymbol{\epsilon_T}$, which is not a problem. In step 3, we compare the two groups of distances, and find cutoff values. The cutoff value could be the intersect points of the two distributions of distances, or the 50% or 95% quantile of distances obtained from step 1. We take whichever the smaller value. The stricter criterion, 50% quantile, is used for response, because response data is dichotomous and does not have outlying values.

Table 3

Steps for Deciding $\boldsymbol{\epsilon_T}$ in Ideal and Empirical Cases

	"True" $\boldsymbol{\epsilon_T}$ (i.e., ideal case)	Approximated $\boldsymbol{\epsilon_T}$ (i.e., empirical case)
Step 1	Do $X_{1:5000} \sim f(\cdot \mid \boldsymbol{\Lambda}^{true})$. Calculate 5,000 distances between these data and the observed data	Do $X_{1:5000} \sim f(\cdot \mid \boldsymbol{\Lambda}^{guessed\ true})$. Calculate 5,000 distances between these data and the observed data
Step 2	Do $X \sim f(\cdot \mid \boldsymbol{\Lambda}^{imputed})$ for 5,000 different sets of $\boldsymbol{\Lambda}^{imputed}$. Calculate 5,000 distances between these data and the observed data	Do $X \sim f(\cdot \mid \boldsymbol{\Lambda}^{imputed\prime})$ for 5,000 different sets of $\boldsymbol{\Lambda}^{imputed'}$. Calculate 5,000 distances between these data and the observed data
Step 3	Compare the 5,000 distances obtained in step 1 and the 5,000 in step 2. Find a value that separates the two groups well.	

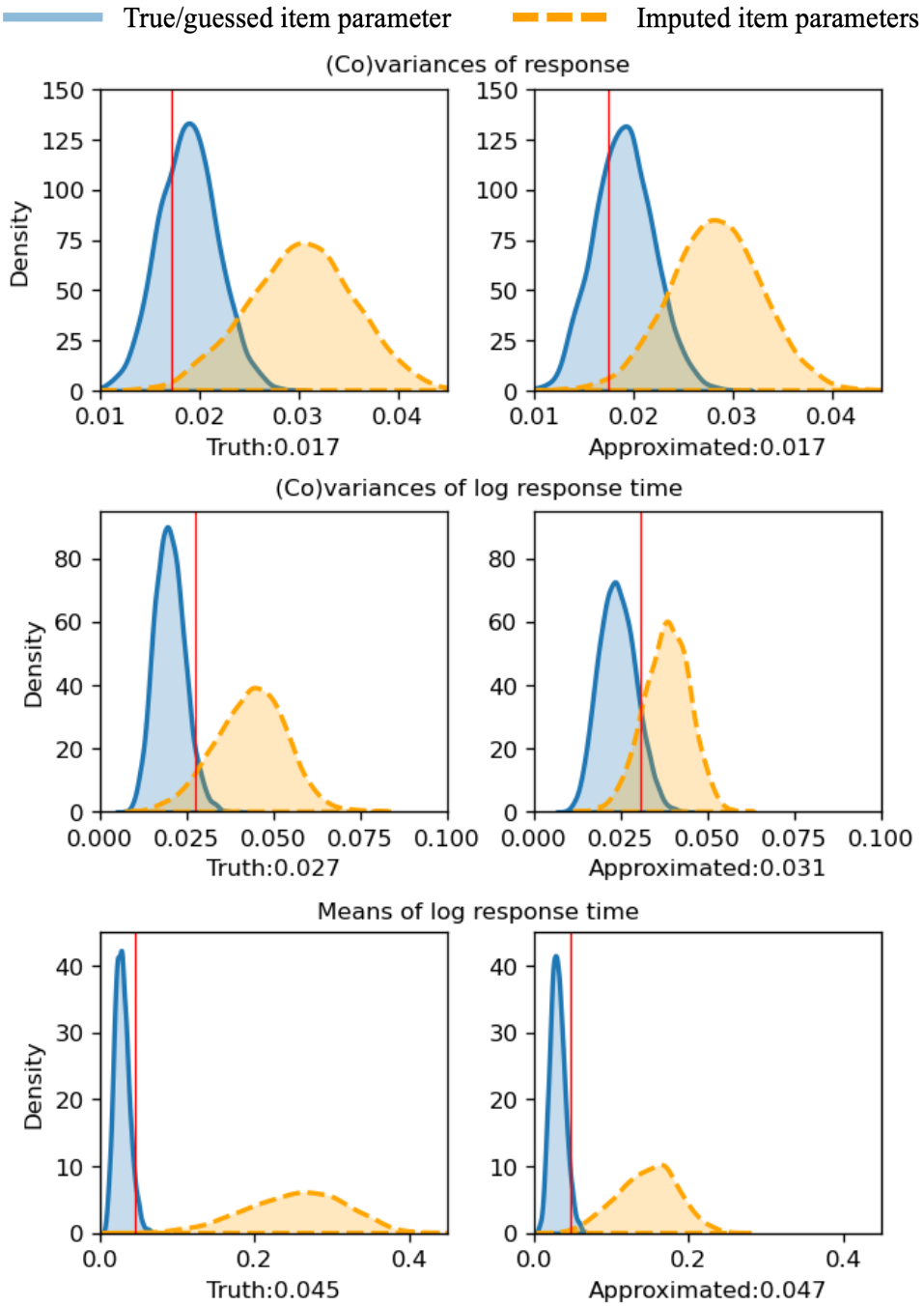
Figure 3 shows $\boldsymbol{\epsilon_T}$ decided in ideal and empirical cases. The ideal and approximated $\boldsymbol{\epsilon_T}$ values are close, which supports our method of empirically deciding

$\boldsymbol{\epsilon_T}$. The blue solid lines in the right column (i.e., empirical case) is similar to that in the left column (i.e., ideal case). This implies that the ideal and empirical distances obtained in step 1 have similar distributions. In the second row of Figure 3b, the imputed item parameters produced even smaller distance on conditional covariances of log response time, which indicates that the distance in conditional covariances may not well distinguish data generated from true and imputed item parameters. This may be due to the small sample size. Among 200 respondents, the response pattern $[Y_i, Y_j] = [0,0]$ appears only 5 to 20 times for almost all pairs of items.

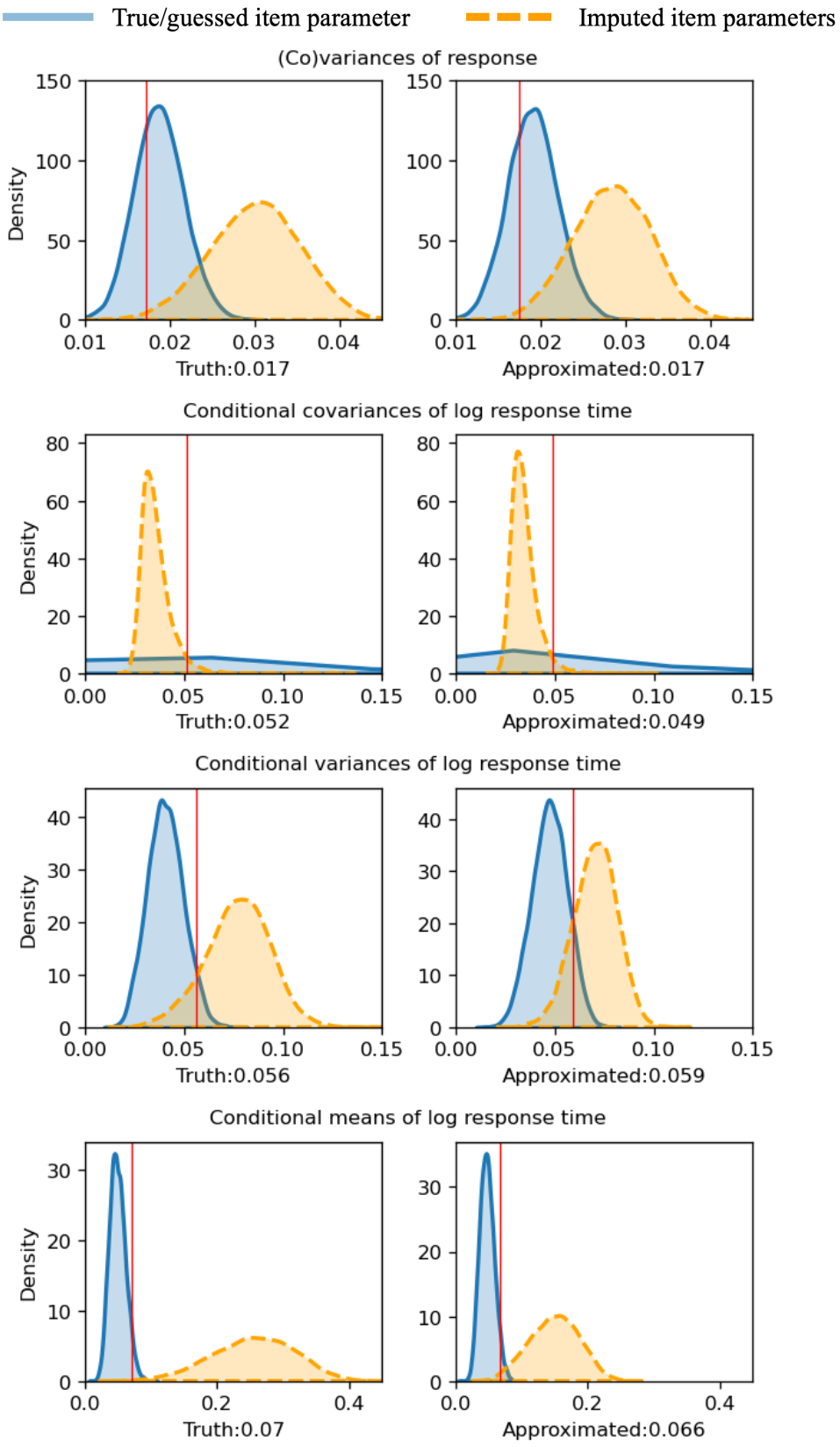
Figure 3

Deciding ϵ_T that Distinguishes Data Generated from True/Guessed or Imputed Item Parameters

(a) Limited Information



(b) Enhance Limited Information



Deciding an Intermediate Tolerance $\boldsymbol{\epsilon_t}$

$\boldsymbol{\epsilon_t}$ for $t > 0$ should be decided first in each of the later iterations (line 7). Since $\boldsymbol{\epsilon_t}$ is a vector containing three or four elements, we first decide which element of $\boldsymbol{\epsilon_t}$ will be reduced in the current iteration. For limited information criterion, there are three elements in $\boldsymbol{\epsilon_t}$ (i.e., $\boldsymbol{\epsilon_t} = (\epsilon_{t,1}, \epsilon_{t,2}, \epsilon_{t,3})$). For $\boldsymbol{\epsilon_t} = \boldsymbol{\epsilon_1}, \boldsymbol{\epsilon_2}, \boldsymbol{\epsilon_3}, \boldsymbol{\epsilon_4}, \boldsymbol{\epsilon_5}, \boldsymbol{\epsilon_6}, \ldots$, the distance we reduce is $\epsilon_{1,1}, \epsilon_{2,2}, \epsilon_{3,3}, \epsilon_{4,1}, \epsilon_{5,2}, \epsilon_{6,3}, \ldots$. Elements that already equal to the final tolerated distance will be skipped. The enhanced limited information criterion follows a similar sequentially iterative schedule. Table 4 illustrates how we reduce $\boldsymbol{\epsilon_t}$ such that $ESS\left(\left\{W_t^{(n)}\right\}, \boldsymbol{\epsilon_t}\right) = s \times ESS\left(\left\{W_{t-1}^{(n)}\right\}, \boldsymbol{\epsilon_{t-1}}\right)$.

Table 4

An Illustrative Example Showing How We Change $\boldsymbol{\epsilon_t}$ in Each Iteration

	Limited Information	Enhanced Limited Information[17]
$t = 0$	$\boldsymbol{\epsilon_0} = [inf, inf, inf]$	$\boldsymbol{\epsilon_0} = [inf, inf, inf, inf]$
$t = 1$	$\boldsymbol{\epsilon_1} = [0.064, inf, inf]$	$\boldsymbol{\epsilon_1} = [0.064, inf, inf, inf]$
$t = 2$	$\boldsymbol{\epsilon_2} = [0.064, 0.574, inf]$	$\boldsymbol{\epsilon_2} = [0.064, 21.427, inf, inf]$
$t = 3$	$\boldsymbol{\epsilon_3} = [0.064, 0.574, 1.411]$	$\boldsymbol{\epsilon_3} = [0.064, 21.427, 1.093, inf]$
$t = 4$	$\boldsymbol{\epsilon_4} = [0.056, 0.574, 1.411]$	$\boldsymbol{\epsilon_4} = [0.064, 21.427, 1.093, 1.364]$
.....	.....	.....
$t = T - 3$	$\boldsymbol{\epsilon_{T-1}} = [0.016, 0.023, 0.053]$	$\boldsymbol{\epsilon_{T-1}} = [0.016, 0.028, 0.040, 0.076]$
$t = T - 2$	$\boldsymbol{\epsilon_{T-1}} = [0.016, 0.023, 0.050]$	$\boldsymbol{\epsilon_{T-1}} = [0.016, 0.028, 0.040, 0.074]$
$t = T - 1$	$\boldsymbol{\epsilon_{T-1}} = [0.016, 0.023, 0.047]$	$\boldsymbol{\epsilon_{T-1}} = [0.016, 0.028, 0.040, 0.072]$
$t = T$	$\boldsymbol{\epsilon_T} = [0.016, 0.023, 0.046]$	$\boldsymbol{\epsilon_T} = [0.016, 0.028, 0.040, 0.0715]$

Note. The numbers are from one replication.

When solving the equation $ESS\left(\left\{W_t^{(n)}\right\}, \boldsymbol{\epsilon_t}\right) = s \times ESS\left(\left\{W_{t-1}^{(n)}\right\}, \boldsymbol{\epsilon_{t-1}}\right)$, a strict equality is not enforced. First, any $\boldsymbol{\epsilon_t}$ such that $ESS\left(\left\{W_t^{(n)}\right\}, \boldsymbol{\epsilon_t}\right) \in$

[17] At $t = 2$, we obtained a large number, 21.427, because if a conditional event (e.g., $[Y_1, Y_2] = [0,0]$) has no respondents, the distance between observed and simulated data for this conditional covariance is imputed as 99. This is more likely to happen at initial iterations, as the priors are weakly informative.

$s \times ESS\left(\left\{W_{t-1}^{(n)}\right\}, \boldsymbol{\epsilon_{t-1}}\right) \pm 20$ is acceptable. Having 20 more of less particles is negatable given that the total number of particles is much larger (i.e., 600). Second, when the change in an element of $\boldsymbol{\epsilon_t}$ is smaller than 0.0001 (e.g., from 0.8 to 0.79999), we stop dealing with unnecessary precision even though the equality on ESS is not satisfied.

Covariance Matrix for the Proposal Distribution

We use symmetric multivariate normal[18] distribution as the proposal distribution (line 22-23). The means of *MVN* are the current particle values, and the (co)variances of *MVN* is 0.1 times the weighted (co)variances of 30% nearest alive particles. We use l^2 distance to find which alive particles are near the current particle. In algorithm 3, $neighbor\_ind_{(n)}$ (line 22) contains the indices of neighboring alive particles for particle n. As for the covariance of *MVN* proposal, the multiplier 0.1 is to yield an acceptance rate approximately equal to 40% in the pseudo-marginal MCMC move of early iterations. Since we track the autocorrelation to ensure sufficient movement in each iteration (line 28), the choice of the multiplier value is not crucial in determining the performance of the sampler.

3.2 Simulation Design and Evaluation Criteria

Simulation Design

This simulation aims to compare the performance of limited information and enhanced limited information criteria when applying the SMC-ABC algorithm. Both

[18] In case that $\tau_j < 0$ or $\tau_j > \tau_j^{guessed}$ is proposed, this proposal will not be accepted, due to zero value on prior density and $R = 0$ in line 25 of algorithm 3.

criteria consider distances based on summary statistics of data (i.e., $\rho(S(X), S(X^{obs}))$), but the enhanced limited information criterion further considers the relationship between a person's response and response time.

We conducted 50 replications[19]. All computations were implemented in Python 3.8 (Van Rossum & Drake, 2009) and carried out on the High-Performance Computer cluster (University of Maryland, 2022).

Evaluation Criteria

We are interested in the point estimates and posterior distributions of Q-diffusion model parameter. For the point estimates, we used the median of particles. The bias and rooted mean squared error (RMSE) of each parameter were compared for two criteria. For posterior distributions, we examined the empirical coverage of 90% equi-tailed credible intervals.

[19] A relatively small number of replications was conducted due to the long computation time. With parallel computation on 8 CPU cores (AMD EPYC 7763, 2.45 GHz base, 3.5 GHz turbo), a replication with limited information criterion usually takes 3 to 6 hours, and a replication with enhanced limited information criterion usually takes 3 to 12 hours. Increasing the number of cores for parallel computation does not substantially reduce computation time, probably because (1) the communication and synchronization between different cores is time consuming; and (2) the parallelizable computation was already greatly accelerated by python package *numba*.

Chapter 4: Results

4.1 Point Estimates

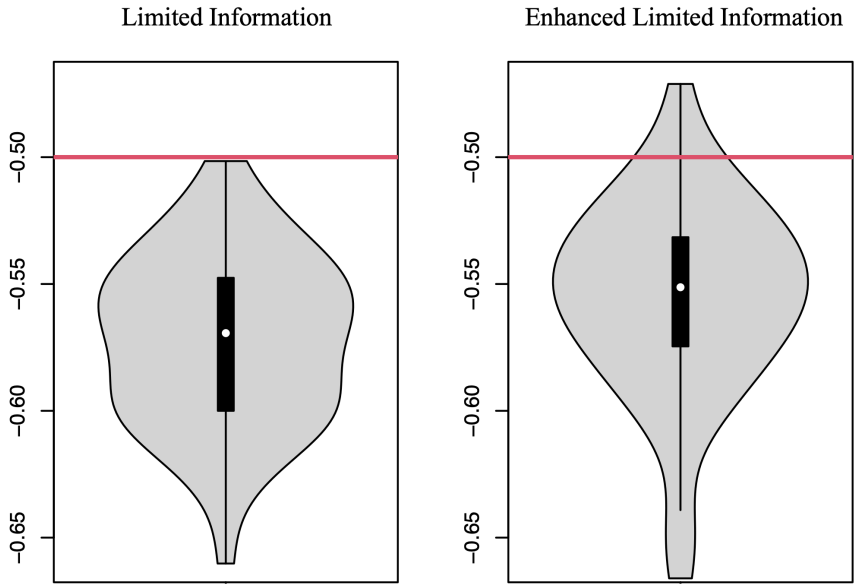
The enhanced limited information criterion yielded slightly better estimates on log of item time pressure. Figure 4 shows the distribution of point estimates across 50 replications. The horizontal solid line represents the true parameter value, -0.5. With the limited information criterion, the median and mean of 50 point estimates are -0.569 and -0.572, respectively. The bias is -0.072, the absolute relative bias is 14.4%, and the RMSE is 0.079. With the enhanced limited information criterion, the median and mean of 50 point estimates are -0.551 and -0.555, respectively. The bias is -0.055, the absolute relative bias is 11%, and the RMSE is 0.067. Both absolute bias and RMSE of enhanced limited information criterion are smaller than that of the limited information criterion, which means better estimation. Figure D1 and Table D1 in Appendix D shows similar results when we use posterior means rather than medians as the point estimate.

Though biased, the recovered log of item pressure indicates comparable probability of answering a moderately difficult item correctly. The true parameter value implies item time pressure of 0.607. For an item of moderate difficulty ($\beta = e^{0.5} = 1.649$) and for a typical person (i.e., $a = b = e^{0}$), an item time pressure of 0.607 indicates that the typical person is expected to answer correctly with a probability of 0.731 and a decision time of 0.627 unit. With the limited information criterion, a bias of -0.072 on $log\alpha$ results in item time pressure of 0.564. This indicates that a typical person is expected to answer a moderately difficult item correctly with a probability of 0.746 and a decision time of 0.718 unit. With the

enhanced limited information criterion, a bias of -0.055 on $log\alpha$ results in item time pressure of 0.574. This indicates that a typical person is expected to answer a moderately difficult item correctly with a probability of 0.742 and a decision time of 0.695 unit. The bias does not substantially influence the probability of answering a moderately difficult item correctly. As for decision time, whether a change from 0.627 unit to 0.718 or 0.695 unit is substantial depends on the time unit (e.g., millisecond, second, minute, etc.).

Figure 4

The Distribution of 50 Point Estimates of Log Item Time Pressure (i.e., $log\alpha$) across Replications



The enhanced limited information criterion yielded slightly better but comparable estimates of log item pure difficulty. The first row of Figure 5 shows the boxplot of 50 point estimates for log β_j. Both criteria slightly overestimate log of item pure difficulty. Table 5 summarizes the bias, absolute relative bias, and RMSE of 50 point estimates for log β_j. All five absolute biases are smaller for the enhanced

limited information, which means better estimation. Except item 3, the RMSEs are smaller for the enhanced limited information, which means better recovery. Figure D2 and Table D1 in Appendix D shows similar results when we use posterior means as the point estimate.

Similarly, the enhanced limited information criterion yielded slightly better but comparable estimates of item residual time. The second row of Figure 5 shows the boxplot of 50 point estimates for τ_j. Both criteria slightly underestimate item residual time. Table 5 summarizes the bias and RMSE of 50 point estimates for τ_j. All five absolute biases are smaller or similar for the enhanced limited information. Except item 3 and 5, the RMSEs are smaller for the enhanced limited information. Figure D2 and Table D2 in Appendix D shows similar results when we use posterior means as the point estimate, despite that the posterior means are generally smaller than medians due to skewed posteriors (Figure D3).

Figure 5

The Boxplot of 50 Point Estimates of Log Item Pure Difficulty (i.e., log β_j*) and Item Residual Time (i.e.,* τ_j*) across Replications*

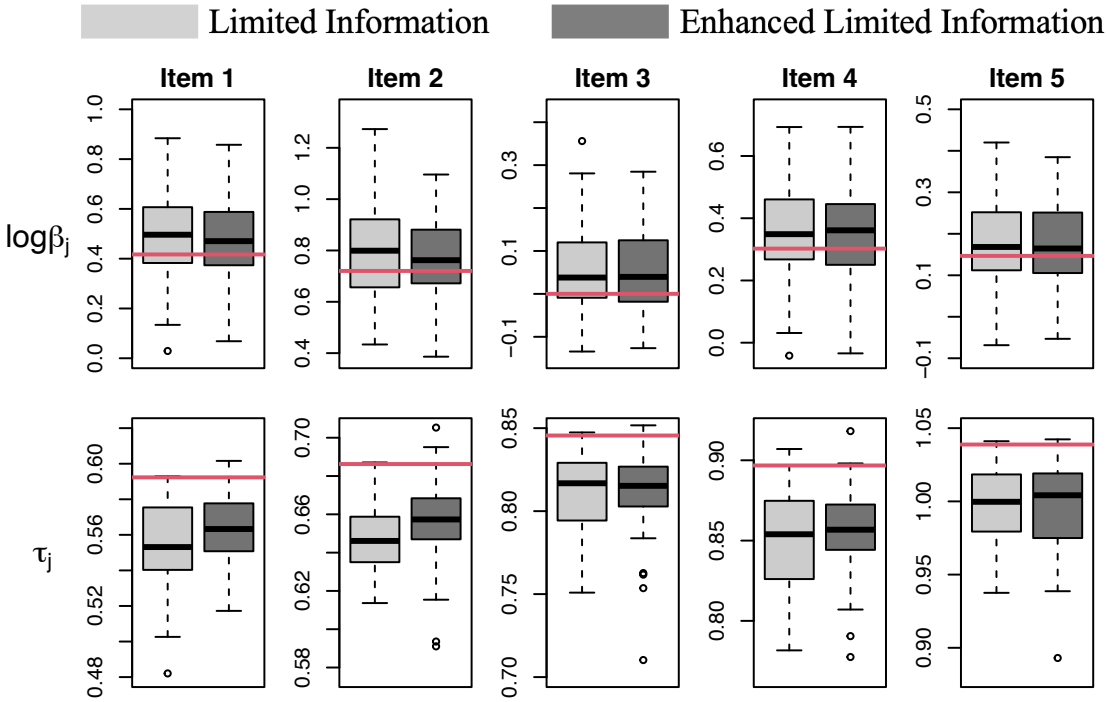


Table 5

The Bias, Absolute Relative Bias, and RMSE of 50 Point Estimates of Log Item Pure Difficulty (i.e., logβ_j*) and Item Residual Time (i.e.,* τ_j*)*

	$log\beta_j$		τ_j	
	Limited Information	Enhanced Limited Information	Limited Information	Enhanced Limited Information
Bias (Absolute Relative Bias)				
Item 1	0.066 (15.8%)	0.058 (13.9%)	-0.039 (6.6%)	-0.029 (4.9%)
Item 2	0.075 (10.4%)	0.047 (6.5%)	-0.040 (5.8%)	-0.032 (4.7%)
Item 3	0.054	0.051	-0.034 (4.0%)	-0.034 (4.0%)
Item 4	0.065 (21.5%)	0.058 (19.2%)	-0.047 (5.2%)	-0.041 (4.6%)
Item 5	0.027 (18.4%)	0.021 (14.3%)	-0.042 (4.0%)	-0.042 (4.0%)
RMSE				
Item 1	0.180	0.175	0.046	0.035
Item 2	0.195	0.174	0.044	0.039
Item 3	0.115	0.117	0.041	0.043
Item 4	0.170	0.161	0.055	0.049
Item 5	0.115	0.108	0.049	0.051

Note. The relative bias of $log\beta_3$ is not reported since the true parameter value is close to 0.

The enhanced limited information criterion yielded more variable point estimates on log of population scale on person caution (i.e., $log\sigma_a$). Figure 6 depicts the distribution of 50 point estimates. The horizontal solid line represents the true parameter value, -1.204. With the limited information criterion, in 50 replications, all point estimates are smaller than the true value. The closest estimate is -1.248. The median and mean of 50 point estimates are -1.438 and -1.439, respectively. The bias is -0.235, the absolute relative bias is 19.5%, and the RMSE is 0.245. With the enhanced limited information criterion, out of 50 replications, 4 replications yielded estimates closer to the true value (i.e., greater than -1.248). The median and mean of 50 point estimates are -1.451 and -1.465, respectively. The bias is -0.261, the absolute relative bias is 21.7%, and the RMSE is 0.303. The absolute bias and RMSE of enhanced limited information criterion are greater than that of the limited information criterion, which means overall worse estimation across 50 replications. Figure D4 and Table D3 in Appendix D shows similar results when we use posterior means as the point estimate.

The enhanced limited information criterion yielded better estimates on log of population scale on person power (i.e., $log\sigma_b$, see Figure 6). The horizontal solid line represents the true parameter value, -1.204. With the limited information criterion, the median and mean of 50 point estimates are -0.968 and -0.952, respectively. The bias is 0.252, the absolute relative bias is 20.9%, and the RMSE is 0.371. With the enhanced limited information criterion, the median and mean of 50 point estimates are -1.224 and -1.216, respectively. The bias is -0.012, the absolute relative bias is 1.0%, and the RMSE is 0.208. Both absolute bias and RMSE of enhanced limited

information criterion are smaller than that of the limited information criterion, which means better estimation. However, Figure D4 and Table D3 in Appendix D shows that when using posterior means as the point estimate of $log\sigma_b$, the enhanced limited information criterion is not favored. This is because for a negatively skewed distribution (e.g., Figure D5), mean is smaller than median.

Table 6 interprets how the true population with $\sigma_b = 0.3$ (i.e., $log\sigma_b = -1.204$) differs from a recovered population with $\sigma_b = 0.386$ (i.e., $log\sigma_b = -0.952$). For example, in the true population, the person power parameter follows $logN(0,0.3^2)$, and 95% persons have a power parameter within $[0.555,1.800]$. Given a typical item (i.e., $\alpha = 0.607$, $\beta = 1.649$) and persons of typical caution (i.e., $a = e^0 = 1$), the probability of answering correctly for the person with 2.5% quantile power is 0.635, with expected decision time of 0.662.

Figure 6

The Distribution of 50 Point Estimates of Log Population Scale (i.e., $log\sigma_a$ and $log\sigma_b$) across Replications

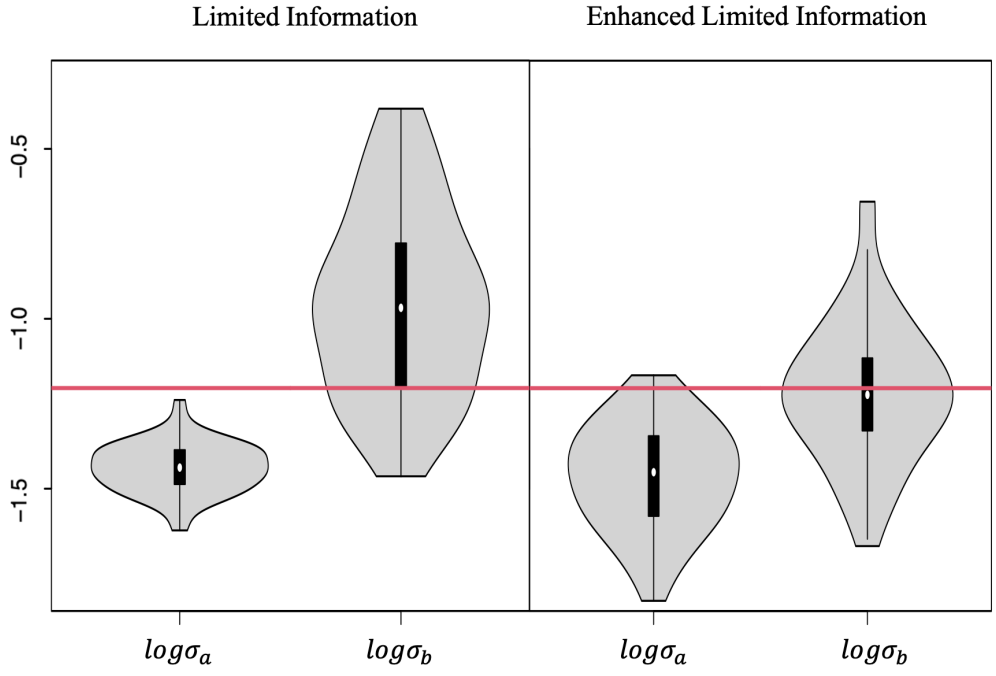


Table 6

The Person Power, Probability of Answering a Typical Item Correctly, and the Expected Decision Time for Populations of Different Scales on Person Power

	Person Power	$P(Y_{typical} = 1)$	Expected Decision Time
True population ($\sigma_b = 0.3$)			
2.5% Quantile	0.555	0.635	0.662
97.5% Quantile	1.800	0.858	0.540
Population recovered by the limit information criterion ($\sigma_b = 0.386$)			
2.5% Quantile	0.469	0.615	0.666
97.5% Quantile	2.131	0.894	0.502

Note. This table assumes persons have typical caution (i.e., $a = e^0 = 1$).

For readers information, the results of MLE are presented and compared with SMC-ABC in Appendix C. MLE came across numerally intractable likelihood for 18 out of 50 times and yielded instable solutions.

4.2 Posterior Distributions

The posteriors produced by the enhanced limited information criterion cover true values more frequently (Table 7). This indicates that overall, the enhanced limited information criterion produced better estimates on posteriors. Appendix E depicts the 90% credible intervals for each of 50 replications. Appendix E shows trends consistent with summative findings reported in Section 4.1. One finding worth noting is for $log\sigma_a$: When the enhanced limited information is used, the 90% credible interval of posterior is wider if the center of posterior is farer away from the true value. In contrast, when the limited information is used, the 90% credible interval of posterior are of comparable length.

Table 7

The Number of Replications that yielded a Posterior with 90% Credible Interval Covering the True Parameter Value

	Limited Information	Enhanced Limited Information
α	46	48
$log\beta_1$	48	50
$log\beta_2$	50	50
$log\beta_3$	50	50
$log\beta_4$	50	50
$log\beta_5$	50	50
τ_1	50	50
τ_2	50	50
τ_3	50	50
τ_4	50	50
τ_5	50	50
σ_a	50	50
σ_b	50	50

Chapter 5: Discussion

This study applies a likelihood-free SMC-ABC algorithm for estimating model parameters of the Q-diffusion model. When accepting/rejecting proposed parameter values, instead of evaluating the likelihood function, the SMC-ABC algorithm compares if the data simulated from proposed model parameters is close to the observed data. As for gauging the closeness between observed and simulated data, we explored the performance of standard and enhanced limited information criteria. The standard criterion considers means and (co)variances of response and response time matrices. The enhanced criterion further considers the dependencies between a respondent's responses and his/her response times. We compared the resulted point estimates and posteriors.

5.1 Summary

The findings suggest that the enhanced limited information criterion can yield more accurate estimates of log population scale on person power when using posterior median as the point estimate. This can be explained by the explicit consideration of data features that are relevant to interested parameters and underlying model assumptions. In the Q-diffusion model, all individuals are assumed to possess non-negative ability, and individuals who answered an item correctly are expected to have spent a longer time solving it. This assumption in conditional mean response time needs to be accounted for by the criterion when using ABC.

The findings show that the enhanced limited information criterion yielded a little improvement on estimating item parameters. This can be explained by the more

information carried by the enhanced criterion. However, this additional information is less related to item parameters. The estimation of item parameters relies more on the whole population's performance, rather than the dependencies between each person's responses and response times. Though the enhanced criterion yielded better estimation, the slight improvement may not worth paying additional computation time.

True model parameters, sample size, and priors are determining factors of the simulation results. The true parameters determine how important is the additional data feature captured by the enhanced criterion. For example, holding other parameters constant, Q-diffusion model assumes that as item pressure decreases (i.e., boundary separation increases), the difference between conditional decision times is larger[20]. Q-diffusion model also assumes that a more heterogeneous population (e.g., $\sigma_a = \sigma_b = 0.5$) has larger difference between conditional decision times[21]. In those cases, ignoring the difference between conditional decision times and adopting the standard limited information criterion may introduce bias. Besides, sample size may affect the performance of the enhanced criterion. Remember that when calculating conditional covariances, among 200 respondents, the response pattern $[Y_i, Y_j] = [0,0]$ appears only 5 to 20 times for almost all pairs of items. A larger sample size may help

[20] For an item of typical difficulty (i.e., $\beta = e^{0.5} = 1.649$) and for a 100,000-person population with $\sigma_a = \sigma_b = 0.3$, if item time pressure is high (i.e., $\alpha = e^{-0.25}$), we have the averaged decision time for persons who answered incorrectly is 0.412, and the averaged decision time for persons who answered correctly is 0.469. The difference is 0.469 - 0.412 = 0.057. If item time pressure is medium (i.e., $\alpha = e^{-0.5}$), the two quantities are 0.636 and 0.732, respectively. The difference is 0.732 - 0.636 = 0.096. If item time pressure is low (i.e., $\alpha = e^{-0.75}$), the two quantities are 0.970 and 1.124, respectively. The difference is 1.124 - 0.970 = 0.154. We have $0.057 < 0.096 < 0.154$.

[21] For a typical item, the more heterogenous population with $\sigma_a = \sigma_b = 0.5$ yields averaged decision times 0.669 and 0.924, conditioning on incorrect and correct answers, respectively. The difference is 0.924 – 0.669 = 0.255, which is larger than 0.096.

stabilize the conditional covariances. As for the priors, both criteria use the same set of data-dependent priors, which may bring bias regardless of the criterion used.

5.2 Future Research

This SMC-ABC algorithm can be easily applied to other process models that have intractable likelihood functions, as long as we are able to generate data from the assumed model. Q-diffusion model is a relatively simple cognitive model that only considers a *single* cognitive process with *time-invariant* drift rate. Other complicated models are more appropriate under some circumstances. For example, the race model is better for tasks that can be solved in multiple ways. It assumes a race between several information accumulation processes, and the first process reaching its bound determines the response and response time (e.g., Tuerlinckx & De Boeck, 2005; Ranger & Kuhn, 2014, Rouder et al., 2015; Tillman et al., 2020). For another example, the Q-diffusion model with time-variant person drift rate is more appropriate when respondents' fatigue plays a role in responses and response times. These models provide interesting descriptions of the cognitive process in decision making and can potentially be applied to various types of tasks, but they have difficult likelihood functions to evaluate. With SMC-ABC algorithm, applied researcher can simply modify the data generating mechanism and priors. The ease of estimation may improve the understanding of cognitive processes and the underlying decision-making mechanisms. However, remember that complicated models with many parameters may not be identified if we only have the response and response time matrices. For under-identified model, the results may be sensitive to priors, and

multiple sets of solutions may provide different interpretations. Additional data may be needed to identify the assumed process model.

5.3 Limitations

This study does not explore if the criteria on gauging similarity is the robust to contaminated data. In empirical data, it is likely that some outlying response time reflects distractions or noises irrelevant to the cognitive process. Both criteria in this study have limitations on dealing with outlying response time data, due to the use of mean. Despite the response time was log transformed, the mean is still vulnerable to outliers. Median may be a better choice in empirical data analysis.

This study does not consider how missing data affects the performance of the algorithm criteria. The missing data issue also involves assumptions regarding the missing mechanism. If missing is at random (e.g., due to technical glitch), then similar to the full information maximum likelihood method, the missingness poses no problem to the SMC-ABC method. We may use pairwise deletion when calculating covariances. If missing is not at random (e.g., related to lower person power), we may treat missing data as another category when calculating summary statistics.

The ABC-SMC algorithm has decreasing acceptance rate as $\boldsymbol{\epsilon_t}$ decreases (i.e., in later iterations), resulting in decreasing efficiency. For example, in one replication, with the limited information criterion, the acceptance rates at the 1st, 50th, and 100th iteration are about 45%, 25%, and 7%, respectively. With the enhanced criterion, the acceptance rates at the 1st, 50th, and 100th iteration are about 45%, 18%, and 3%, respectively. A low acceptance rate means more times of perturbance for sufficient movement. A plausible reason is that we perturb all elements of a particle at the same

time, and this may be more difficult as we get stricter with closeness. However, it may not worth using more efficient ways of proposing new values, if the cost is more computation time.

The computation time of the SMC-ABC algorithm is long, especially with the enhanced limited information criterion that considers more data features. A replication with limited information criterion usually takes 3 to 6 hours, and a replication with the enhanced limited information criterion may take 3 to 12 hours. The computation time is related to observed data size, the criterion gauging similarity, and tuning aspects of the algorithm.

First, larger observed data means longer computation time. Since the SMC-ABC algorithm simulates data of the same size as the observed data (line 24, algorithm 3), having larger observed data means simulating more data entries. This increase in computation time is linear. For example, if the observed data has 10 times more persons, then the time for simulating responses and response times will also increase by approximately 10 times.

Second, the criterion gauging similarity impacts computation time by affecting the acceptance rate (line 25, algorithm 3). The more data features to match, the more likely that a proposed particle value fails to satisfy all. This results in lower acceptance rate and more times of perturbing.

Third, as for tuning aspects, a longer computation time is related to larger number of simulated data for one particle, larger number of total particles, a shrinkage rate closer to 1, larger minimum effective sample size (i.e., less dead particles that do not need computation), smaller threshold for the product of

autocorrelation, and larger percentage of interested parameters that have a product of autocorrelation below the threshold. These abovementioned tuning aspects cost longer computation time for finer approximation and more sufficient exploration of parameter space, which is beneficial to estimation. Besides, the computation time may also be increased by an inappropriate multiplier for deciding the covariance matrix of proposal. A large multiplier is related to large movements of a small portion of particles. A small multiplier is related to small movements of a large proportion of particles. A too large/small multiplier may increase the number of perturbing (lines 20-28 in algorithm 3) before sufficient movement. Nevertheless, since the algorithm enforces sufficient movement, the choice of the multiplier does not harm estimation, despite possibly increased computation time. Similarly, the percentage of neighboring alive particles affect computation time by affecting the covariance matrix of proposal. However, an inappropriate percentage may bring bias to estimation. A large percentage may result in a covariance matrix reflecting features of the whole set of particles, rather than reflecting features near a specific particle. This may bring bias when proposing new particle values. Nevertheless, the percentage cannot be too small, such that there are enough neighboring particles for calculating a positive finite covariance matrix.

Another limitation of this study is that the decision of $\boldsymbol{\epsilon_T}$ is specific to the Q-diffusion model and relies on initially guessed parameters. For other models, the ABC-SMC algorithm may stop when the estimates barely change.

5.4 Conclusions

In conclusion, this study has demonstrated the feasibility of using a likelihood-free SMC-ABC algorithm in estimating the parameters of the Q-diffusion model. The results highlight the importance of considering relevant data features and model assumptions in the criteria used to gauge the similarity between observed and simulated data. Future research can extend this method to more complicated cognitive process models with intractable likelihood functions, allowing for a better understanding of underlying decision-making mechanisms. However, limitations such as the robustness to contaminated or missing data, long running time, and decreasing acceptance rates must be considered. Overall, the SMC-ABC algorithm is a useful tool for estimating parameters in models where likelihood functions are intractable, opening up new avenues for research in cognitive modeling and related fields.

Appendices

Appendix A

Obtaining the Initial Guessed Values from the Observed Response and Response Time Data

Equation 25 Derivation

From the observed response data, we can calculate the logit of passing rate for item *j* across all persons (i.e., $logitP(Y_j = 1)$ is observed). This observed value can be rewritten in terms of model parameters as follows:

$$\begin{aligned} logitP\big(Y_j = 1\big) &= logit\big[\int P\big(Y_{ij} = 1, \theta = \theta_i\big)d\theta\big] \\ &= logit\left[\int f_\theta(\theta_i)P\big(Y_{ij} = 1 \mid \theta = \theta_i\big)d\theta\right] \end{aligned} \tag{A1}$$

where $\theta = ab$ is the unified person parameter, and $f(\cdot)$ is the density function for θ.

Expressing $P\big(Y_{ij} = 1|\theta = \theta_i\big)$ in Equation A1 using Q-diffusion model parameters enables us to connect model parameters with the observed response data. According to the Q-diffusion model (Equation 7), we have:

$$logitP\big(Y_{ij} = 1|\theta = \theta_i\big) = \frac{\theta_i}{\alpha_j\beta_j} - log\big(O_j - 1\big)$$

$$\xrightarrow{\times f_\theta(\theta_i)} f_\theta(\theta_i)logitP\big(Y_{ij} = 1|\theta = \theta_i\big) = \frac{\theta_i f(\theta_i)}{\alpha_j\beta_j} - f_\theta(\theta_i)\,log\big(O_j - 1\big)$$

$$\xrightarrow{integrate\ \theta\ out} \int f_\theta(\theta_i)\ logitP\big(Y_{ij} = 1|\theta = \theta_i\big)\,d\theta = \int \frac{\theta_i f_\theta(\theta_i)}{\alpha_j\beta_j}d\theta - log\ \big(O_j - 1\big)$$

$$= \frac{1}{\alpha_j\beta_j}E(\theta) - log\ \big(O_j - 1\big)$$

Assuming $\theta \sim lognormal(\mu, \sigma^2)$ and plugging in its formula for expectation, we have:

$$\int f_\theta(\theta_i)\ logitP\big(Y_{ij} = 1|\theta = \theta_i\big)\,d\theta = \frac{1}{\alpha_j\beta_j}e^{\mu+\frac{\sigma^2}{2}} - log\ \big(O_j - 1\big) \tag{A2}$$

The right-hand-side (RHS) of Equation A1 is similar to the left-hand-side (LHS) of Equation A2. In some conditions, those are approximately equal. Next, we describe when the approximate relationship (Equation A3) is valid.

$$\begin{aligned} logit\big[\int f_\theta(\theta_i)P\big(Y_{ij} = 1 \mid \theta = \theta_i\big)d\theta\big] &\overset{?}{\approx} \int f_\theta(\theta_i)\ logitP\big(Y_{ij} \\ &= 1|\theta = \theta_i\big)\,d\theta \end{aligned} \tag{A3}$$

The logit function is approximately linear in some range (Figure A1). For $0.2 \le p \le 0.8$, $logit(p) \approx 4p - 2$. The RHS of Equation A3 can be rewritten as:

$$
\begin{aligned}
RHS &= \int f_\theta(\theta_i) logit P\big(Y_{ij}=1|\theta=\theta_i\big)\, d\theta \\
&\overset{O_j=2}{\geq} \int f_\theta(\theta_i)\big[4P\big(Y_{ij}=1|\theta=\theta_i\big)-2\big]\, d\theta \\
&= 4\int f_\theta(\theta_i) P\big(Y_{ij}=1|\theta=\theta_i\big)\, d\theta - 2\int f_\theta(\theta_i)\, d\theta \\
&= 4\int f_\theta(\theta_i) P\big(Y_{ij}=1|\theta=\theta_i\big)\, d\theta - 2 \\
&\overset{0.2\leq P(Y_j=1)\leq 0.8}{\approx} logit \int f(\theta_i) P\big(Y_{ij}=1|\theta=\theta_i\big)\, d\theta = LHS
\end{aligned}
$$

The first inequality holds because when $O_j = 2$, we have $P\big(Y_{ij}=1\big|\theta_i\big) \geq 0.5$. According to Figure A1, the logit function is higher than the linear function when $p \geq 0.5$. The approximation happens when $0.2 \leq P\big(Y_j=1\big) \leq 0.8$, which is a reasonable range for the passing rate of an item.

Thus, for $O_j = 2$, we have:

$$\frac{1}{\alpha_j\beta_j} e^{\mu+\frac{\sigma^2}{2}} \geq 4P\big(Y_j=1\big)-2 \approx logit P\big(Y_j=1\big),$$

which yields Equation 25.

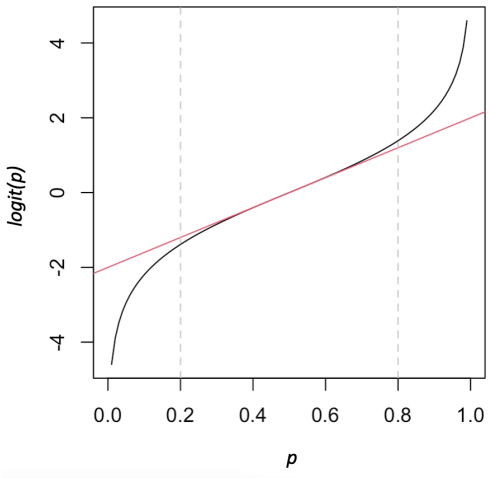
Similarly, when getting the initial guess of the product of person boundary separation and drift rate, we have:

$$a_i^{guessed} b_i^{guessed} \approx \frac{logit\big(P(Y_i=1)\big)}{E\big(\frac{1}{\alpha\beta}\big)}.$$

Assuming that $\alpha \sim \exp\,(unif(-1,0))$ and $\beta \sim \exp\,(unif(0,1))$, the denominator $E\left(\frac{1}{\alpha\beta}\right) = e + \frac{1}{e} - 2$.

Figure A1

The Relationship between the Logit Function and a Linear Function $y = 4x - 2$.



Equation 26 Derivation

Equation 26 is derived from Equations 4 and 8, and the fact that total response time is the sum of residual time and the time spent on solving the problem (i.e., $E(\mathbb{T}_{ij}^{\text{obs}}) = \tau_j + E(T_{ij})$). In the Q-diffusion model, treating α_j and β_j as parameters of interested, from Equation 4, we have:

$$P(Y_{ij} = 1|a_i, b_i) = \frac{1}{1 + \exp\left(-\frac{a_i b_i}{\alpha_j \beta_j}\right)}$$

$$\Leftrightarrow \quad \exp\left(-\frac{a_i b_i}{\alpha_j \beta_j}\right) = \frac{1 - P(Y_{ij} = 1|a_i, b_i)}{P(Y_{ij} = 1|a_i, b_i)} \tag{A4}$$

Applying Equation A4 to Equation 8, we have:

$$E(\mathbb{T}_{ij}^{\text{obs}}|a_i, b_i) - \tau_j = \left(\frac{a_i}{2b_i}\frac{\beta_j}{\alpha_j}\right)\frac{1 - \exp\left(-\frac{a_i b_i}{\alpha_j \beta_j}\right)}{1 + \exp\left(-\frac{a_i b_i}{\alpha_j \beta_j}\right)}$$

$$\Leftrightarrow \quad E(\mathbb{T}_{ij}^{\text{obs}}|a_i, b_i) - \tau_j = \left(\frac{a_i}{2b_i}\frac{\beta_j}{\alpha_j}\right)\left(2P(Y_{ij} = 1|a_i b_i) - 1\right) \tag{A5}$$

where $E(\mathbb{T}_{ij}^{\text{obs}}|a_i, b_i)$ is the expected response time of person *i* when he/she repeatedly respond to item *j*, assuming we can erase his/her memory after each response. $E(\mathbb{T}_{ij}^{\text{obs}}|a_i, b_i)$ is non-observable, but the expected response time on item *j* throughout all persons, $E(\mathbb{T}_j^{\text{obs}})$, is observable. Therefore, we consider integrating throughout all persons (i.e., all values of a_i and b_i) in Equation A5. By observing the RHS of Equation A5, we replace the notations as $\gamma_i = a_i/b_i$ and $\theta_i = a_i b_i$, and integrate out γ and θ as follows:

$$\int f_\gamma(\gamma_i) E(\mathbb{T}_{ij}^{\text{obs}}|\gamma_i, \theta_i)\, d\theta - \tau_j = \left(\int f_\gamma(\gamma_i)\gamma_i\, d\gamma\right)\left(\frac{\beta_j}{2\alpha_j}\right)\left(2P(Y_{ij} = 1|\theta_i) - 1\right)$$

$$E(\mathbb{T}_{ij}^{\text{obs}}|\theta_i) - \tau_j = E(\gamma)\left(\frac{\beta_j}{2\alpha_j}\right)\left(2P(Y_{ij} = 1|\theta_i) - 1\right)$$

where $f_\Gamma(\cdot)$ is the density of γ. We further integrate out θ:

$$\int f_\theta(\theta_i) E(\mathbb{T}_{ij}^{\text{obs}}|\theta_i)\, d\theta - \tau_j = E(\gamma)\left(\frac{\beta_j}{2\alpha_j}\right)\left(2\int f_\theta(\theta_i) P(Y_{ij} = 1|\theta_i) - 1\right)$$

$$E(\mathbb{T}_j^{\text{obs}}) - \tau_j = E(\gamma)\left(\frac{\beta_j}{2\alpha_j}\right)\left(2P(Y_j = 1) - 1\right)$$

Thus, we have

$$\frac{\alpha_j}{\beta_j} = E(\gamma)\frac{2P(Y_j = 1) - 1}{2(E(\mathbb{T}_j^{\text{obs}}) - \tau_j)}.$$

where $\gamma = a/b$ also has a lognormal distribution, with $log\gamma$ having the mean $\mu' = \mu_a - \mu_b$ and variance $\sigma^2 = \sigma_a^2 + \sigma_b^2$. $E(\gamma) = e^{\mu' + \sigma^2/2}$. We use observed passing rate on item j and the average response time to approximate $P(Y_j = 1)$ and $E(\mathbb{T}_j^{\text{obs}})$.

Similarly, when getting the initial guess of the ratio of person boundary separation and drift rate, we have:

$$\frac{a_i^{guessed}}{b_i^{guessed}} = \frac{2\left(E\left(\mathbb{T}_i^{\text{obs}}\right) - E(\tau)\right)}{E\left(\frac{\beta}{\alpha}\right)(2P(Y_i = 1) - 1)}.$$

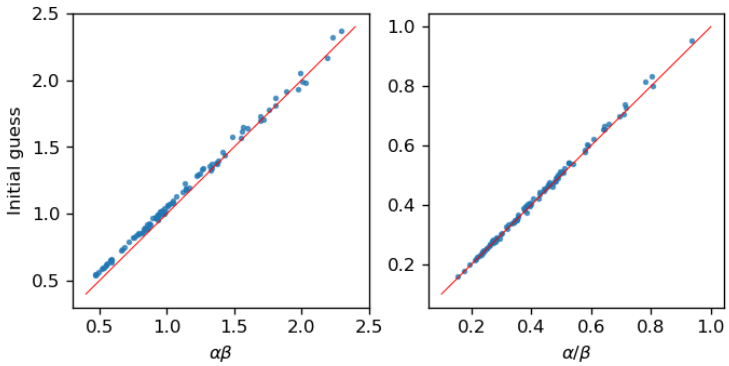
Assuming that $\alpha \sim \exp(unif(-1,0))$ and $\beta \sim \exp(unif(0,1))$, we have $E\left(\frac{\beta}{\alpha}\right) = e1 + e(e-2)$. $E(\tau)$ can be approximated by the mean of minimum item-wise observed response times.

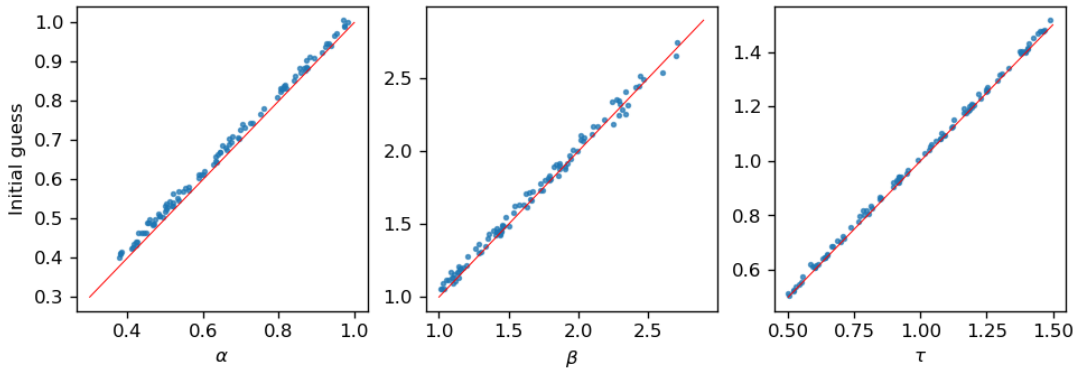
Evidence for Approximations

As for initially guessing item parameters, we simulated observed data of 20000 persons on 100 items. The two population scale parameters were both fixed at 0.3. From the observed data, we calculated our guesses of $\alpha_j\beta_j$ and α_j/β_j for 100 items. The first row of Figure A2 plots the guessed values against true values. The guessed $\alpha_j\beta_j$ is larger than true $\alpha_j\beta_j$ when the true value is relatively small, and the guessed $\alpha_j\beta_j$ is smaller than true $\alpha_j\beta_j$ when the true value is relatively large. This is consistent with Figure A1, where the linear approximation is larger/smaller than the logit function at small/large p values. The guessed α_j/β_j is very close to the true α_j/β_j. From guessed $\alpha_j\beta_j$ and α_j/β_j, we can guess values of α_j and β_j. The second row of Figure A2 depicts the relationship between guessed and true values of α_j, β_j, and τ_j. Guessed τ_j is always larger than the true τ_j, because the minimal observed time is τ_j plus a small but non-negative problem-solving time. Guessed α_j is slightly larger than the true α_j. Guessed β_j is close to the true β_j.

Figure A2

The Relationship between Guessed and True Item Parameter Values Based on a Data of 20,000 Persons Responding to 100 Items

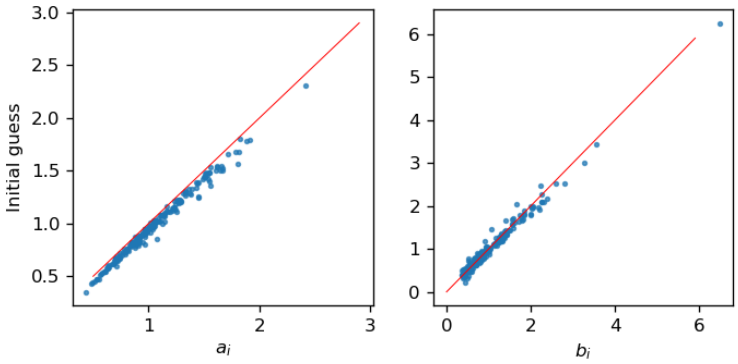




As for initially guessing person population parameters, we simulated observed data of 200 persons on 1000 items. The item parameters come from $\alpha \sim \exp(unif(-1,0))$, $\beta \sim \exp(unif(0,1))$, and $\tau \sim unif(0.5,1.5)$. The person population parameters are 0.326 (for population boundary separation) and 0.535 (for population drift rate). For individual person drift rate and boundary separation, Figure A3 plots the guessed values against true values. For the two interested population parameters, the initial guesses are 0.325 (for population boundary separation) and 0.646 (for population drift rate), which are close to the true values.

Figure A3

The Relationship between Guessed and True Person Parameter Values Based on a Data of 200 Persons Responding to 1,000 Items



The Interdependency of Guessing Item and Person Parameters

As shown in Equations 25, and 26 and this Appendix, guessing item parameters requires the distributions of person parameters, and guessing person parameters requires the distributions of item parameters. To obtain good initial guesses, we could iterate between guessing item and person parameters. However, good initial guesses may not be necessary for Algorithm 3, given that we will explore the parameter space sufficiently and have non-informative priors with relatively large standard deviation. Therefore, this project was not too strict with the starting values.

Appendix B

List of Frequently Used Notations

Notation	Meaning	Notes
General Diffusion Process		
Y	Y represents which boundary a diffusion process hits.	Used in Equations 1-7. $Y = 0$ means hitting the lower boundary. $Y = 1$ means hitting the upper boundary.
T	T is the time spent on the diffusion process.	Used in Equations 1-9. In the scenario of Q-diffusion model, T is the decision time (i.e., time spent on information accumulation).
$\mathbb{X}(t)$	The position of a diffusion process in the state space at time t ($t > 0$).	
A	The boundary separation for a diffusion process.	In the Q-diffusion model, A is decomposed as item boundary separation α and person boundary separation a (i.e., $A = a/\alpha$).
B	The drift rate for a diffusion process.	In the Q-diffusion model, V is decomposed as item drift rate β and person drift rate b (i.e., $V = b/\beta$).
V	The volatility for a diffusion process	In this study, volatility was fixed at 1 for identification purpose.
Z	The starting point for a diffusion process	In this study, $Z = A/2$.
Q-diffusion Model		
O	Number of options	$O = 2$ was assumed for all items in this study.
α	Item boundary separation	α_j is the item boundary separation (i.e., item time pressure) for item j. $\boldsymbol{\alpha} = (\alpha_1, \ldots\ldots, \alpha_J)$.
β	Item drift rate	β_j is the item drift rate (i.e., item

		difficulty) for item *j*. $\boldsymbol{\beta} = (\beta_1, \ldots\ldots, \beta_J)$.
τ	Item residual time	τ_j is the item residual time (i.e., non-decision time) for item *j*. $\boldsymbol{\tau} = (\tau_1, \ldots\ldots, \tau_J)$.
$\boldsymbol{\Lambda}$	$\boldsymbol{\Lambda} = (\log(\boldsymbol{\alpha}), \log(\boldsymbol{\beta}), \boldsymbol{\tau})$. This is interested item parameters.	$\boldsymbol{\Lambda}$ represents all (transformed) item parameters for all *J* items. Un-bolded $\Lambda_j = (\alpha_j, \beta_j, \tau_j)$ is the item parameters for item *j*.
a	Person boundary separation	a_i is the person boundary separation (i.e., person caution) for person *i*.
b	Person drift rate	b_i is the person drift rate (i.e., person power) for person *i*.
θ	$\theta = ab$.	θ_i is the person composite ability (i.e., person power with caution) for person *i*.
γ	$\gamma = a/b$.	Used in appendix A for deriving initial guess of item parameters.
Approximate Bayesian Computation		
Θ	The parameter whose posterior distribution is of interested	This is a general notation. It is not something specific to Q-diffusion models.
X	Simulated data from a particle	X_m is the *m*th simulated data for a particle. $X_{1:M}$ is all *M* simulated data for a particle.
x^{obs}	Observed data	This is the response data that X is trying to approximate.
$\rho(X, x^{obs})$	The distance between observed and simulated data	$\rho(\cdot)$ is a distance measure. For example, ρ may measure l^2 norm.
$\pi(\cdot)$	Prior distribution	
$\pi(\Theta \mid x^{obs})$	Posterior distribution given observed data X^{obs}	See Equation 10

$\pi(\Theta \mid \rho(x^{obs}, X) < \epsilon)$	Approximate posterior distribution	See Equation 11
Sequential Monte Carlo		
$\mathbb{P}_t(d\Theta))$	One of a sequence of probability distributions $\mathbb{P}_0(d\Theta)), \ldots, \mathbb{P}_T(d\Theta))$	$0 \leq t \leq T$.
$\mathbb{K}_t(\Theta_{t-1}, \mathrm{d}\Theta_t)$	A forward Markov kernel that leaves *invariant* $\mathbb{P}_{t-1}(d\Theta_{\mathrm{t}})$	Equation 19.
$q_t(\Theta,\cdot)$	A convenient proposal distribution for proposing a new value from Θ at time t	In algorithms 2 and 3, this is the Gaussian random walk.
$\sum_t$	$\sum_t$ is the covariance matrix for $q_t(\Theta,\cdot)$.	This is valid if $q_t(\Theta,\cdot)$ is a Gaussian random walk.
ϵ_t	The tolerance level at time t. $\rho(X, X^{obs}) < \epsilon_t$ is considered as close.	A decreasing sequence $\infty = \epsilon_0 > \epsilon_1 > \cdots > \epsilon_t$ is used in algorithm 2. $\boldsymbol{\epsilon_t}$ can be a vector.
s	Shrinkage rate	$(1 - s) \times 100\%$ is the percentage of removed particles in each iteration.
R	Acceptance rate of proposed value	Calculated based on prior, proposal, and simulated data.
N	The number of particles	*N* = 1000 in this study
M	Number of simulated data from one particle	*M* = 30 in this study
ABC-SMC for Q-diffusion Model		
$x^{obs} = (\mathbb{Y}^{\mathrm{obs}}, \mathbb{T}^{\mathrm{obs}})$.	Observed data for Q-diffusion model.	It includes response data $\mathbb{Y}^{\mathrm{obs}}$ and response time data $\mathbb{T}^{\mathrm{obs}}$.
$X = (\mathbb{Y}, \mathbb{T})$.	Simulated data for Q-diffusion model.	
r	Threshold for the product of autocorrelation.	It quantifies "sufficient" movement.

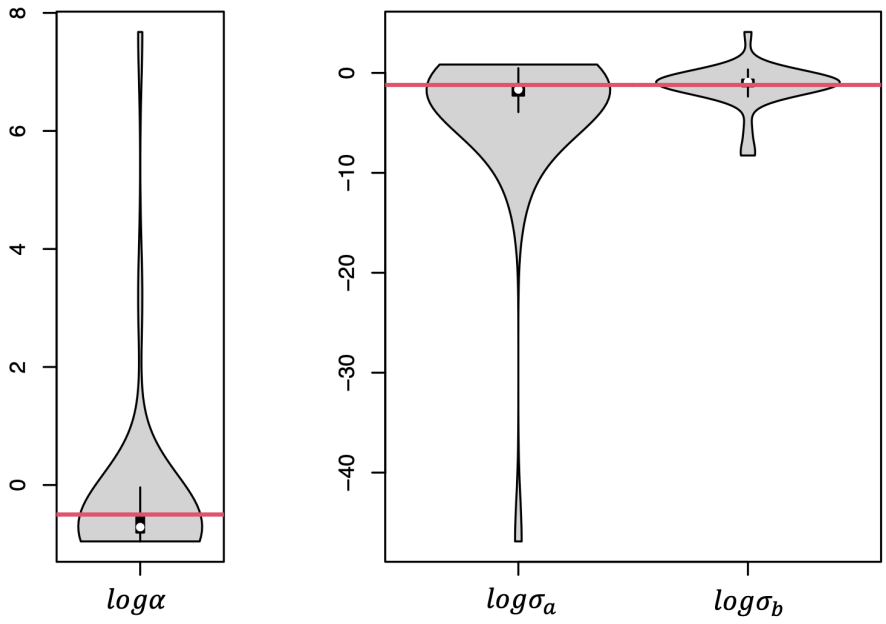
Appendix C

Results of Maximum Likelihood Estimation

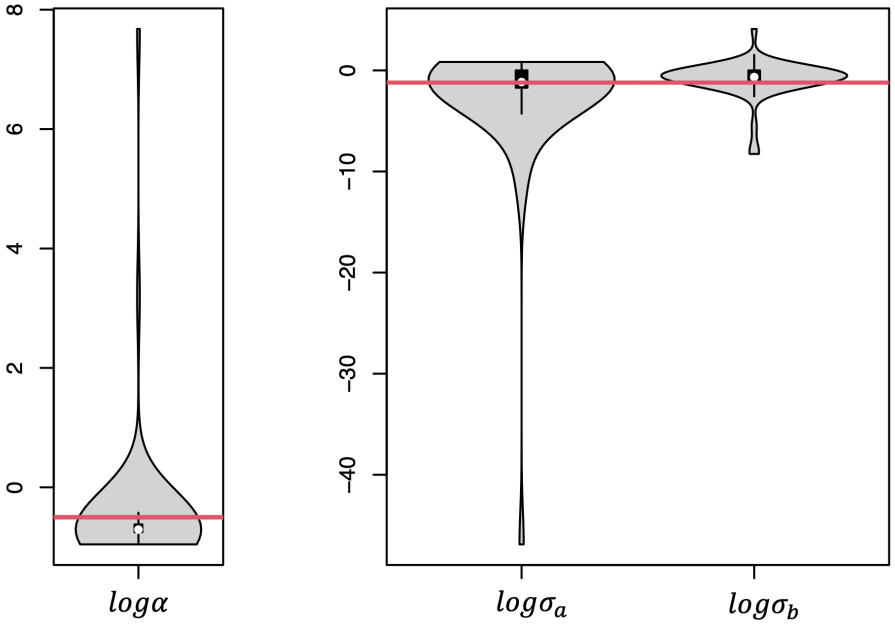
Figure C1

The Distribution of MLE Estimates on Item Time Pressure and Population Scales

(a) Across 32 Replications



(b) Across 50 Replications

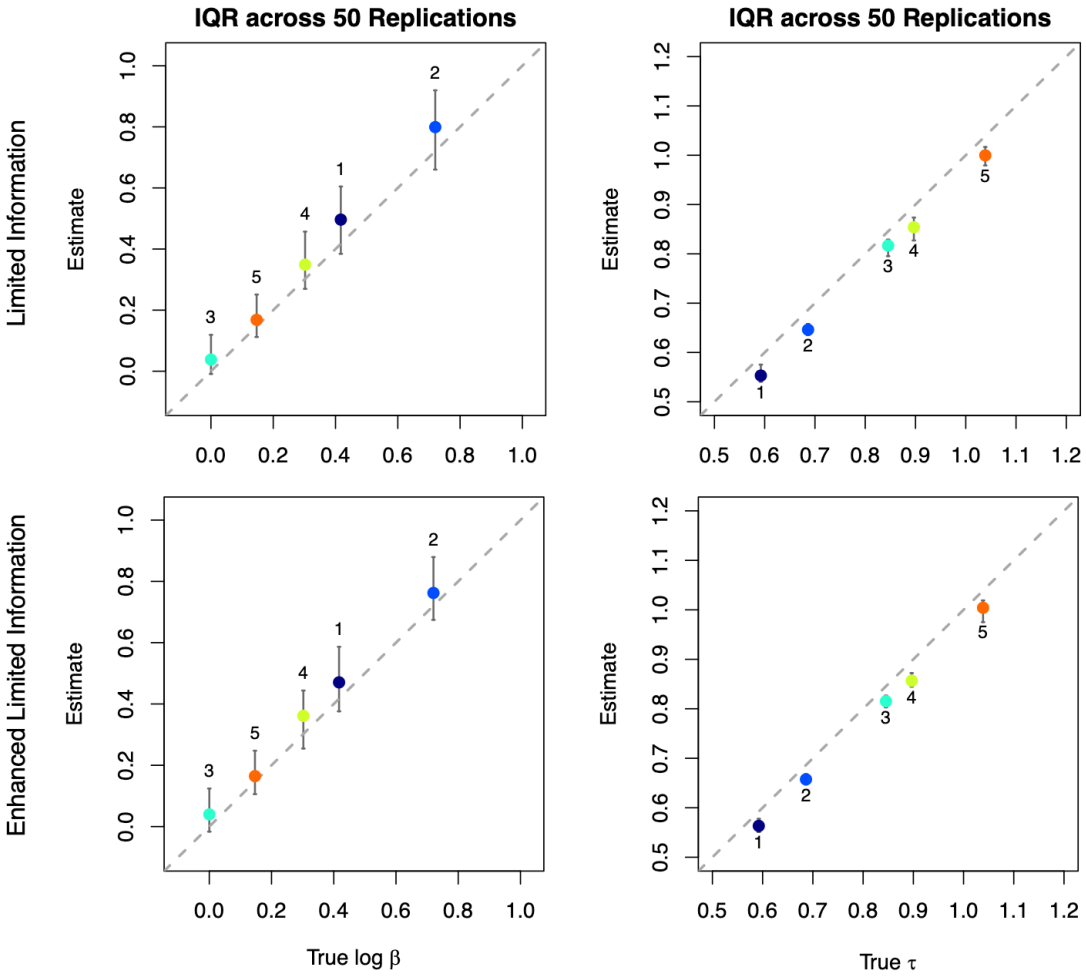


Note. With MLE, 18 out of 50 replications have computationally interactable likelihood. This may be due to the approximation of infinite terms in Equations 1 and 2.

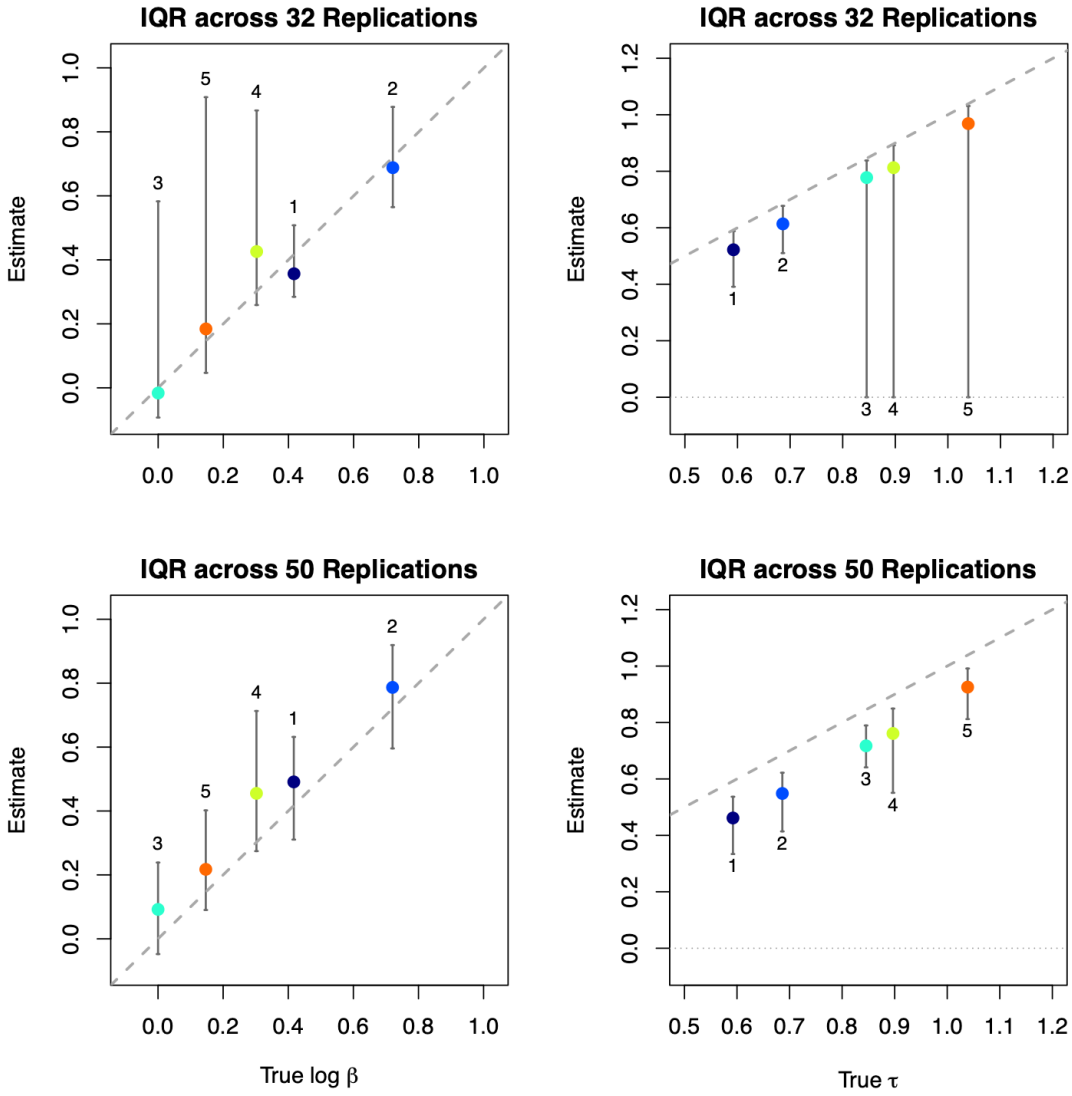
Figure C2

Comparing MLE with SMC-ABC Estimates on Item Pure Difficulty and Residual Times

(a) SMC-ABC Estimates across 50 Replications



(b) MLE Estimates across Replications



Note. The IQR of 32 replications that do not have computational issue is wider, probably due to the small number of replications.

Appendix D

Results using Posterior Means as the Point Estimate

Figure D1

The Distribution of 50 Point Estimates (Posterior Means) of Log Item Time Pressure (i.e., $\log\alpha$) across Replications

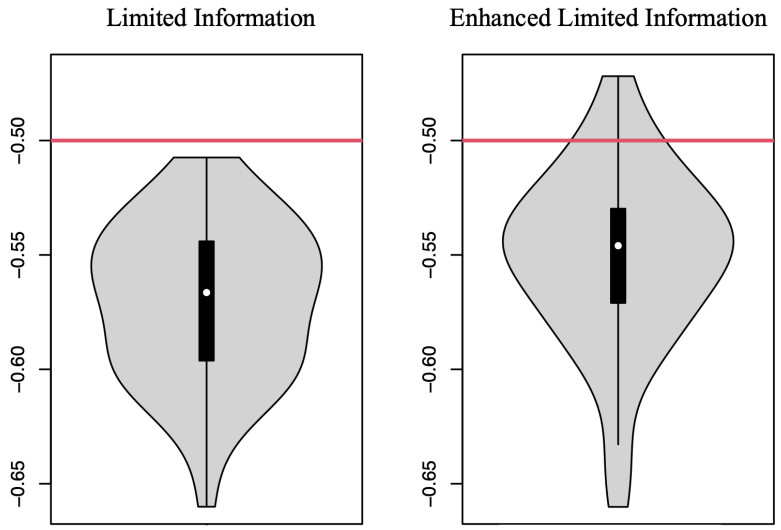


Table D1

The Bias, Absolute Relative Bias, and RMSE of 50 Point Estimates (Posterior Means) of Log Item Time Pressure (i.e., $\log\alpha$)

	Limited Information	Enhanced Limited Information
Bias (Absolute Relative Bias)	-0.070 (14.0%)	-0.051 (10.2%)
RMSE	0.078	0.064

Figure D2

The Boxplot of 50 Point Estimates (Posterior Means) of Log Item Pure Difficulty (i.e., log β_j*) and Item Residual Time (i.e.,* τ_j*) across Replications*

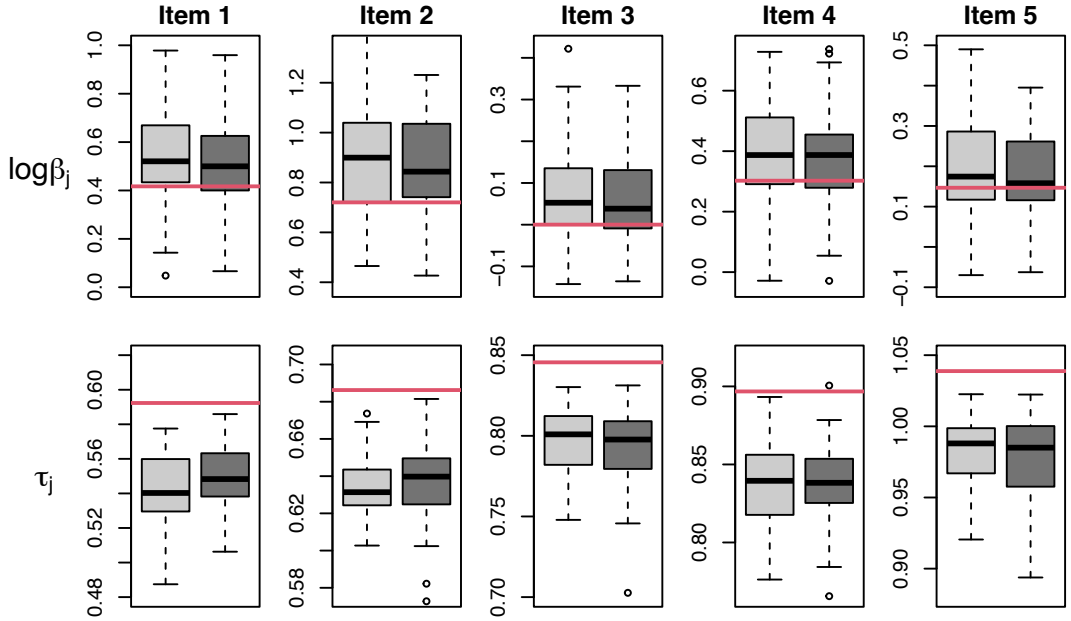


Table D2

The Bias, Absolute Relative Bias, and RMSE of 50 Point Estimates (Posterior Means) of Log Item Pure Difficulty (i.e., logβ_j*) and Item Residual Time (i.e.,* τ_j*)*

	$log\beta_j$		τ_j	
	Limited Information	Enhanced Limited Information	Limited Information	Enhanced Limited Information
Bias (Absolute Relative Bias)				
Item 1	0.114 (27.3%)	0.102 (24.5%)	-0.049 (8.3%)	-0.044 (7.4%)
Item 2	0.174 (24.2%)	0.138 (19.2%)	-0.053 (7.7%)	-0.049 (7.1%)
Item 3	0.062	0.057	-0.048 (5.7%)	-0.053 (6.3%)
Item 4	0.100 (33.1%)	0.082 (27.2%)	-0.060 (6.7%)	-0.060 (6.7%)
Item 5	0.044 (29.9%)	0.035 (23.8%)	-0.058 (5.6%)	-0.062 (6.0%)
RMSE				
Item 1	0.220	0.214	0.053	0.048
Item 2	0.275	0.242	0.055	0.053
Item 3	0.127	0.126	0.052	0.058
Item 4	0.195	0.181	0.065	0.065
Item 5	0.128	0.117	0.063	0.069

Note. The relative bias of $log\beta_3$ is not reported since the true parameter value is close to 0.

Figure D3

Overlaying 50 Posteriors of Item Residual Time

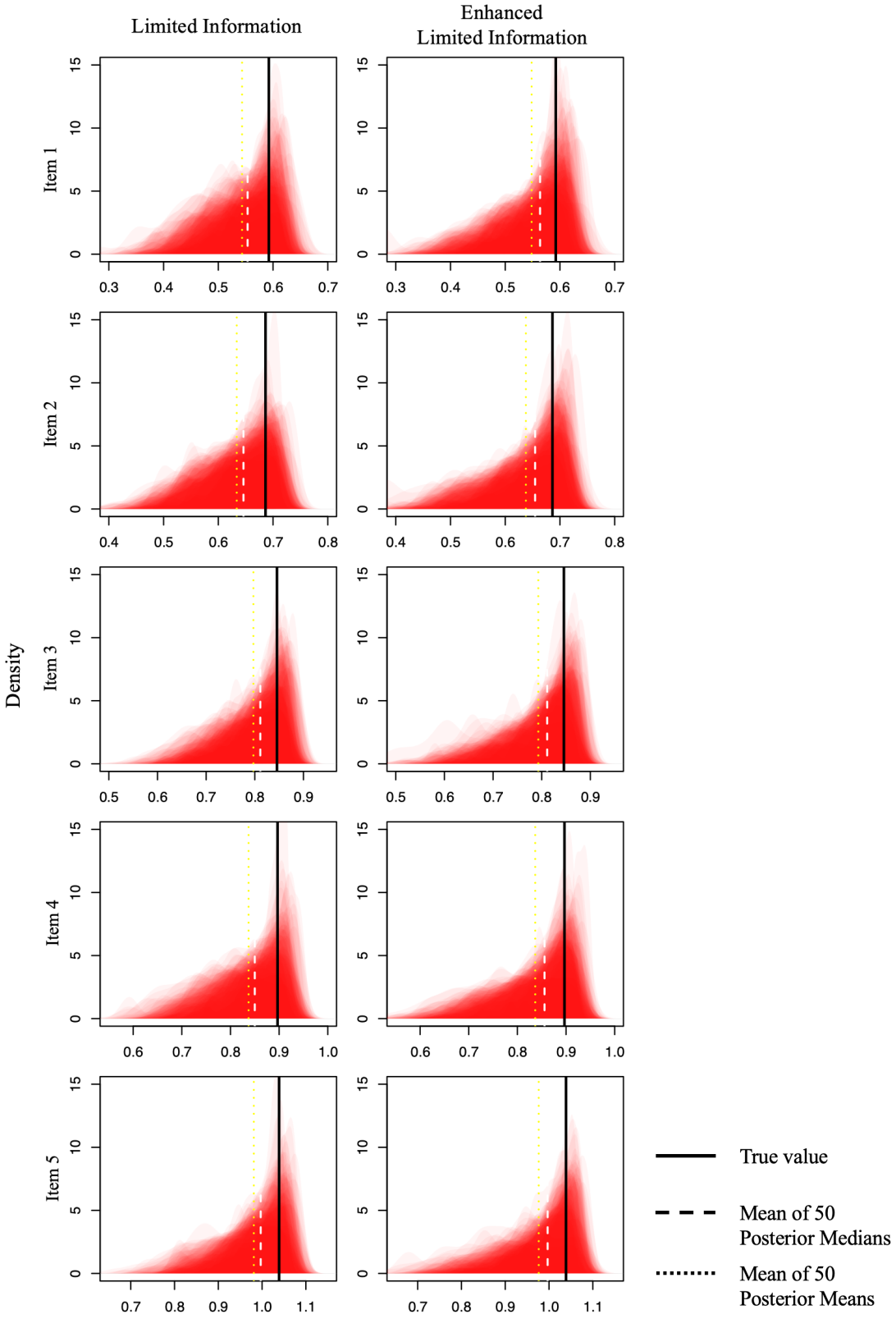


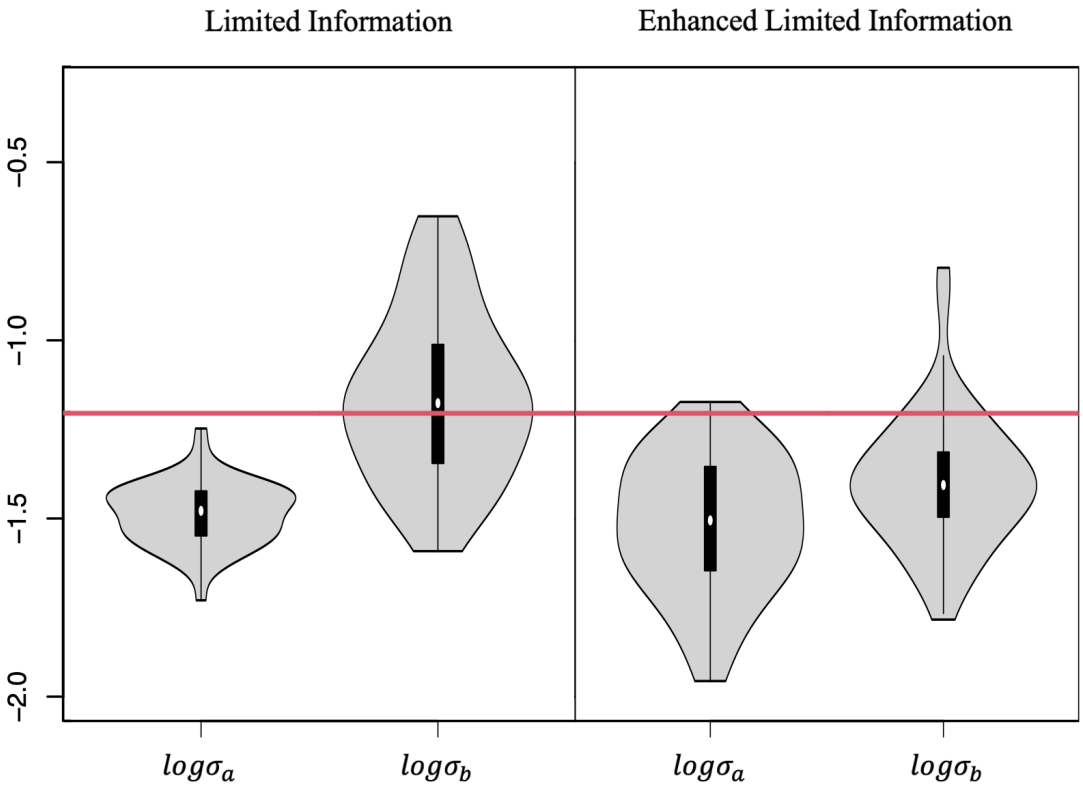
Table D3

The Bias, Absolute Relative Bias, and RMSE of 50 Point Estimates (Posterior Means) of Log Population Scale (i.e., $log\sigma_a$ and $log\sigma_b$)

	Limited Information	Enhanced Limited Information
$log\sigma_a$		
Bias (Absolute Relative Bias)	-0.282 (23.4%)	-0.314 (26.1%)
RMSE	0.295	0.367
$log\sigma_b$		
Bias (Absolute Relative Bias)	0.044 (23.7%)	-0.195 (16.2%)
RMSE	0.235	0.267

Figure D4

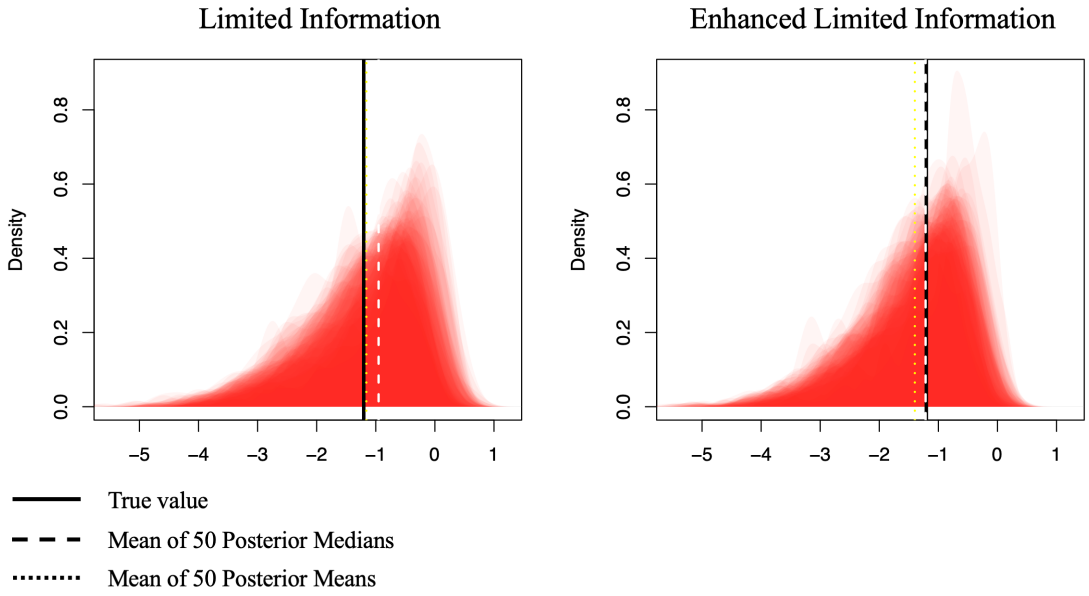
The Distribution of 50 Point Estimates (Posterior Means) of Log Population Scale (i.e., $log\sigma_a$ and $log\sigma_b$) across Replications[22]



[22] For $log\sigma_a$, the closest estimate yielded by the limited information criterion is -1.247. With the enhanced limited information criterion, out of 50 replications, 3 replications yielded estimates closer to the true value (i.e., greater than -1.247).

Figure D5

Overlaying 50 Posteriors of Log Population Scale on Person Power

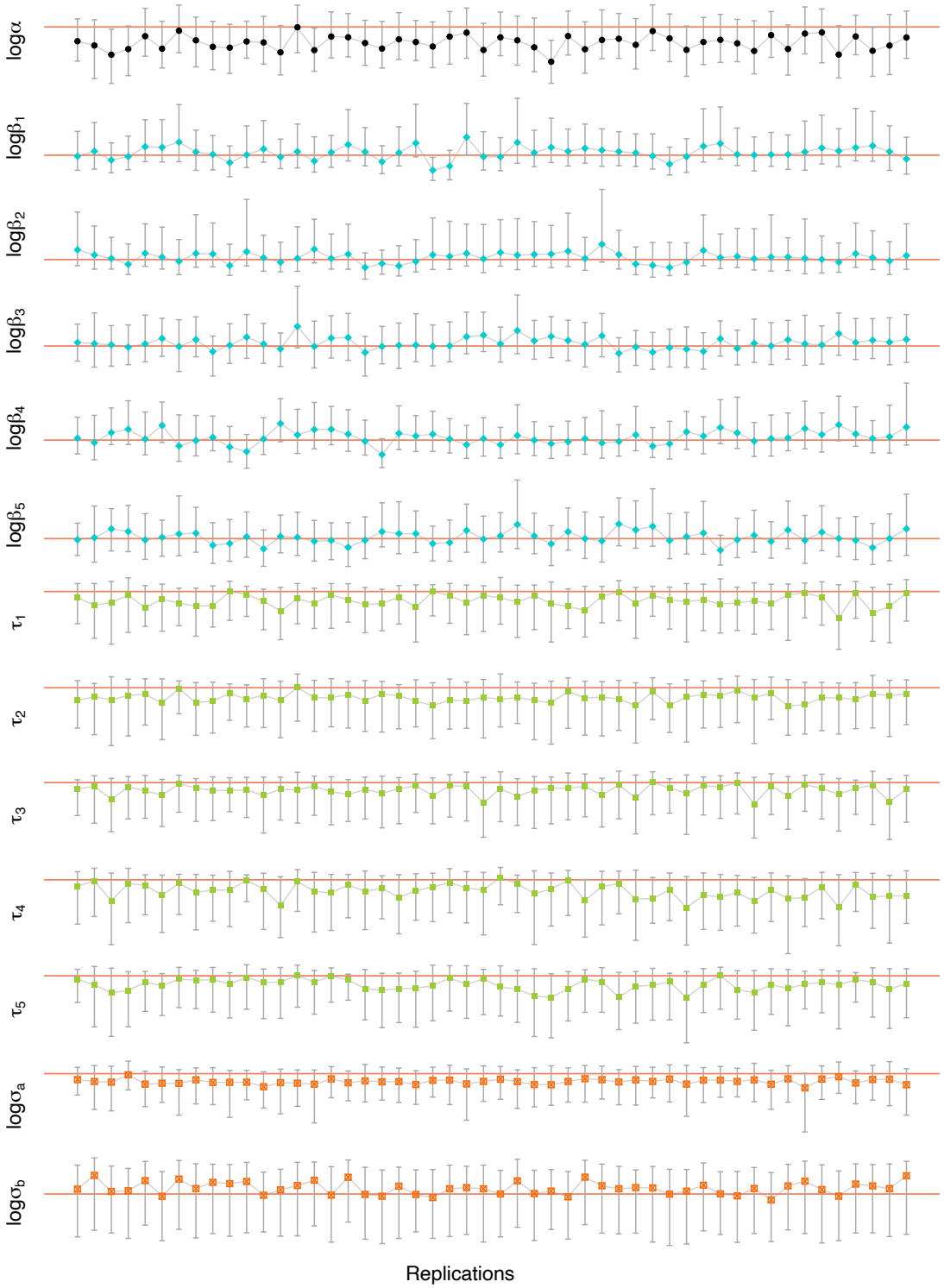


As for why the posteriors have long left tails, a speculation is that particles indicating larger population scale on power may be rejected easier. A large population scale is more likely to yield extreme response time, which may yield large distance between imputed and observed data.

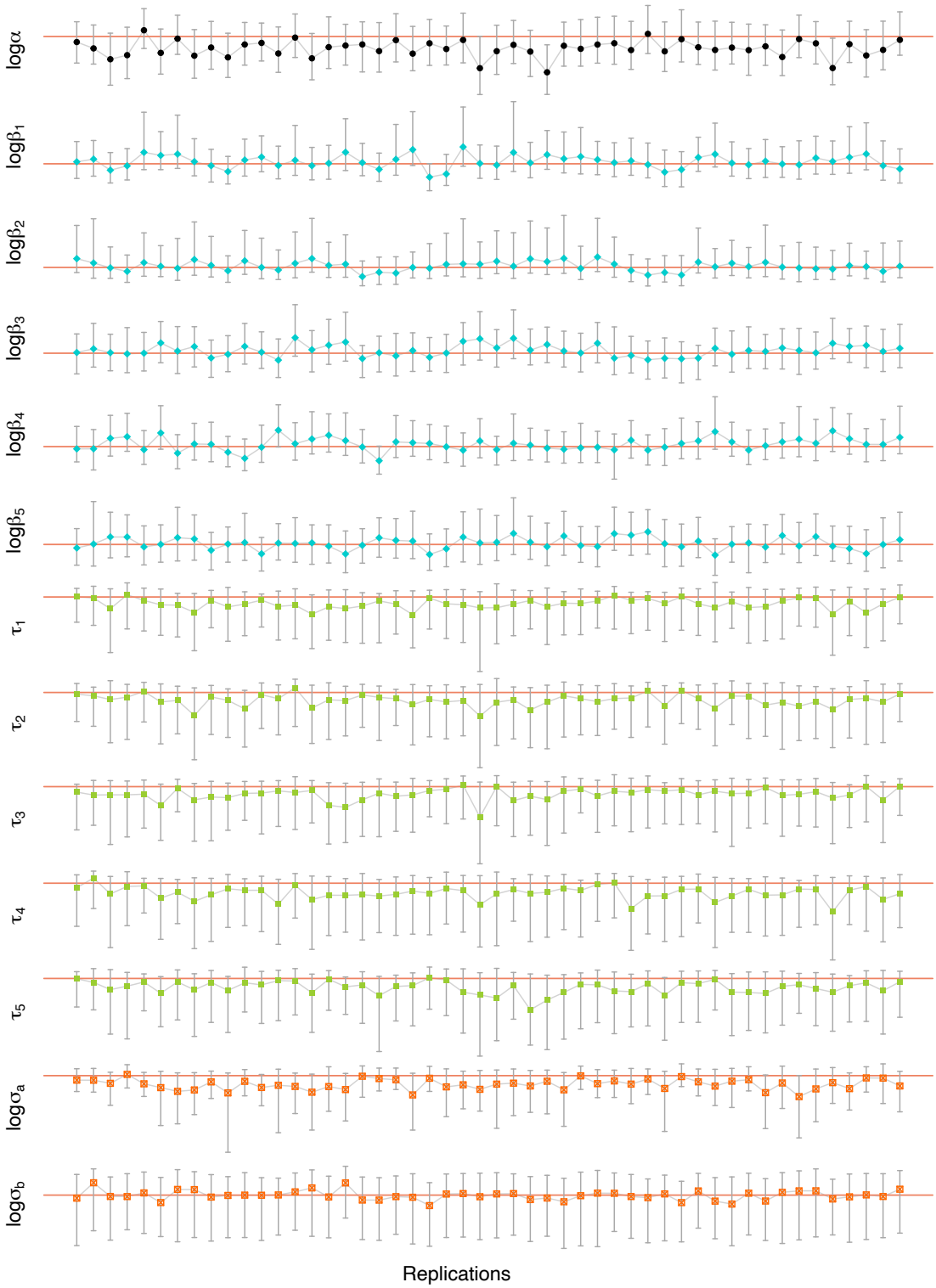
Appendix E

The 90% Credible Interval of Posteriors for 50 Replications

(a) Limited Information Criterion



(b) Enhanced Limited Information Criterion



Appendix F
Python Codes

Part I. Data Simulation

```
import numpy as np
from numpy import array
from numba import jit
from numba.typed import List
import random
```

Function for Simulating Observed Data

```
#%% simulate responses based on Q-diffusion model parameter
@jit(nopython = True)
def simuM(ai, ap, vi, vp, ter, ysize, nit, n_options, seedid = None,
          MM = 1, t_s = 1/100, max_t = 100, eps = 1e-15, rej_M = 20):
    """
    Simulate MM datasets given ONE particle. Used for simulating the observed data.

    Parameters
    ----------
    nit: number of items
    ysize: number of persons.
    ap: person boundary separaion for ysize persons
    vp: person drift rate for ysize persons
    ai: item boundary separation for nit items
    vi: item drift rate for nit items
    ter: residual time for nit items

    n_options: the number of options in the Qdiffusion models. n_options = 2, 3, 4, ...
```

```
seedid: the seed passed for "random" package, for generating data
eps: when using the rejection sampling for simulating RT, a formula
     (Eq.16 in Tuerlinckx, 2002) involves an infinite sum.
     This is the tolerance for the approximation of this infinite sum.
rej_M: when the acceptance rate for rejection sampling is smaller than 1/rej_M,
       the algorithm will not use rejection sampling,
       but use traceline to simulate decision time.
t_s: t-s. When simulating the traceline, we first simulate the increment
     between time t and time s (i.e., X(t) - X(s) ~ N( mu * (t-s), sigma^2 * (t-s))).
     The smaller t-s, the finer the simulated traceline, thus the finer the simulated decision
     time. For example, if we use t_s = 1, then we are simulating X(t) value at every
     time unit (t = 0, 1, 2, 3, ...). The simulated decision time will be integer.
     If we use t_s = 0.1, then we are simulating X(t) value at t = 0, 0.1, 0.2, ....
     The simulated decision time will have one decimal.
     t_s depends on how precise you want the final decision time be.
     Setting t_s to a very small value may unnecessarily increase the compuation demand.
max_t: When simulating the traceline, we will simulate between t = 0 and t = max_t.
       If the traceline hits the boundary between t = 0 and max_t,
       the decision time will be the time that hits boundary.
       If the traceline did not hit the boundary until t = max_t, the decision time will be max_t.
       This number can be set according to the maximum time observed in your data.
       Setting t_s to a very large value may unnecessarily increase the compuation demand.
MM : int. The number of datasets to be generated given this set of particle.

Returns
-------
X, RT : a List of M arraies. Each array is one of the MM simulated Data

"""
```

```
if seedid != None:
    random.seed(seedid)
X = List()
T = List()
P = List()
for m in range(MM):

    # simulate: nit items * ysize examinees
    for j in range(nit):

        xj = np.zeros(ysize)
        rtj = np.zeros(ysize)
        probj = np.zeros(ysize)

        for p in range(ysize):

            # get overall drift rate and boundary separation
            a = ap[p]/ai[j]; a2 = a ** 2
            mu = vp[p]/vi[j] - (ai[j]/ap[p]) * np.log(n_options - 1)
            si = 1; si2 = si ** 2 # volatility fixed

            M = np.pi * si2/a2 * (np.exp(a * mu/(2 * si2)) + np.exp(-a * mu/(2 * si2))) *\
                1/(mu ** 2/(2 * si2) + np.pi ** 2 * si2/(2 * a2)) # Eq. 12

            # If the acceptance rate 1/M >= 1/rej_M, use rejection sampling for decision time.
            # The default value 20 corresponds to acceptance rate >= 0.05
            if (M <= rej_M):
                lmb = mu ** 2/(2 * si2) + np.pi ** 2 * si2/(2 * a2) # Eq. 13 in Tuerlinckx, 2002
                FF = np.pi**2 * si2**2 * 1/(np.pi**2 * si2**2 + mu**2 * a2) # Eq. 16
```

```
    ou = -999
    while ou == -999:
        u = random.uniform(a = 0, b = 1) # Eq. 15 & 16
        v = random.uniform(a = 0, b = 1) # Eq. 15 & 16

        sh1 = 1
        sh2 = 0
        sh3 = 0
        i = 0
        # check approximation/convergence of the infinite sum in Eq. 16
        while abs(sh1 - sh2) > eps or abs(sh2 - sh3) >  eps:
            sh1 = sh2
            sh2 = sh3
            i += 1
            sh3 = sh2 + (2 * i + 1) * (-1)**i * (1 - u)**(FF * (2 * i + 1)**2)
            # calculating the summation by adding one more term
        myeval = 1 + (1 - u)**(-FF) * sh3 # Eq. 16
        if v <= myeval:
            ou = 1/lmb * abs(np.log(1 - u))

# If the acceptance rate 1/M < 0.05, use traceline for generating decision time.
else:
    nstep = int(np.ceil(max_t/t_s))
    position = a/2
    stepi = 0
    while stepi <= nstep:
        increment = random.normalvariate(mu * t_s, si * np.sqrt(t_s))
        position = position + increment
```

```
                stepi += 1
                if position > a or position < 0:
                    break
            ou = stepi * t_s

        # generate & store response and response time [WIP]
        uuu = random.uniform(a = 0, b = 1)
        score = ((1/(1 + np.exp(- mu * a))) > uuu) * 1
        xj[p] = score
        rtj[p] = ou + ter[j]
        probj[p] = (1/(1 + np.exp(- mu * a)))

    # horizontally stack each person's data
    xj = np.expand_dims(xj, 1)  # make it a 2-D array s.t. it can be vstacked (numba requires)
    rtj = np.expand_dims(rtj, 1)
    probj = np.expand_dims(probj, 1)

    if j == 0:
        xx = xj
        tt = rtj
        pp = probj
    else:
        # In numba, xx and xp should have the same dimension (i.e., 2d) to be stacked,
        # though python does not require that
        xx = np.hstack((xx,xj))
        tt = np.hstack((tt,rtj))
        pp = np.hstack((pp,probj))

#store
```

```
        X.append(xx)
        T.append(tt)
        P.append(pp)
    return X, T, P
```

Functions for Simulating M Imputed Data for a Particle

```
#%% simulate responses based on Q-diffusion model parameter, using different person for each M
@jit(nopython = True)
def simuM_Mp(ai, apM, vi, vpM, ter, ysize, nit, n_options, #seedid = None,
             MM = 30, t_s = 1/100, max_t = 100, eps = 1e-15, rej_M = 20):
    """
    Simulate MM datasets given ONE set of item particles and randomly generated person parameters.
    Different MM uses different persons.

    Parameters
    ----------
    ap_M: An MM * ysize array. MM different sets of person boundary separaions for ysize persons
    vp_M: An MM * ysize array. MM different sets of person drift rates for ysize persons.

    Returns
    -------
    X, RT : a List of M arrays. Each array is one of the MM simulated Data

    """
    X = List()
    T = List()

    for m in range(MM):
        # persons for mth data
```

```
ap = apM[m,]
vp = vpM[m,]
# simulate: nit items * ysize examinees
for j in range(nit):
    xj = np.zeros(ysize)
    rtj = np.zeros(ysize)

    for p in range(ysize):
        # get overall drift rate and boundary separation
        a = ap[p]/ai[j]; a2 = a ** 2
        mu = vp[p]/vi[j] - (ai[j]/ap[p]) * np.log(n_options - 1)
        si = 1; si2 = si ** 2 # volatility fixed

        M = np.pi * si2/a2 * (np.exp(a * mu/(2 * si2)) + np.exp(-a * mu/(2 * si2))) *\
            1/(mu ** 2/(2 * si2) + np.pi ** 2 * si2/(2 * a2)) # Eq. 12

        # If the acceptance rate 1/M >= 1/rej_M, use rejection sampling for decision time.
        if (M <= rej_M):
            lmb = mu ** 2/(2 * si2) + np.pi ** 2 * si2/(2 * a2) # Eq. 13 in Tuerlinckx, 2002
            FF = np.pi**2 * si2**2 * 1/(np.pi**2 * si2**2 + mu**2 * a2) # Eq. 16

            ou = -999
            while ou == -999:
                u = random.uniform(a = 0, b = 1) # Eq. 15 & 16
                v = random.uniform(a = 0, b = 1) # Eq. 15 & 16

                sh1 = 1
                sh2 = 0
                sh3 = 0
```

```
        i = 0
        # check approximation/convergence of the infinite sum in Eq. 16
        while abs(sh1 - sh2) > eps or abs(sh2 - sh3) >  eps:
            sh1 = sh2
            sh2 = sh3
            i += 1
            sh3 = sh2 + (2 * i + 1) * (-1)**i * (1 - u)**(FF * (2 * i + 1)**2)
            # calculating the summation by adding one more term
        myeval = 1 + (1 - u)**(-FF) * sh3 # Eq. 16
        if v <= myeval:
            ou = 1/lmb * abs(np.log(1 - u))

else: # If the acceptance rate 1/M < 0.05, use traceline for generating decision time
    nstep = int(np.ceil(max_t/t_s))
    position = a/2
    stepi = 0
    while stepi <= nstep:
        increment = random.normalvariate(mu * t_s, si * np.sqrt(t_s))
        position = position + increment
        stepi += 1
        if position > a or position < 0:
            break
    ou = stepi * t_s

# generate & store response and response time [WIP]
uuu = random.uniform(a = 0, b = 1)
score = ((1/(1 + np.exp(- mu * a))) > uuu) * 1
xj[p] = score
rtj[p] = ou + ter[j]
```

```
            # horizontally stack each person's data
            xj = np.expand_dims(xj, 1)  # make it a 2-D array s.t. it can be vstacked (numba requires it)
            rtj = np.expand_dims(rtj, 1)

            if j == 0:
                xx = xj
                tt = rtj
            else:
                xx = np.hstack((xx,xj))
                tt = np.hstack((tt,rtj))

        #store
        X.append(xx)
        T.append(tt)
    return X, T

#%% wrap-up for simuM_Mp: simulate M different sets of person parameters,
#   and generate M data based on same item parameters and different person parameters
def simuM_varp(ai, ap_sd, vi, vp_sd, ter, ysize, nit, n_options, #seedid = None,
               MM = 30, t_s = 1/100, max_t = 100, eps = 1e-15, rej_M = 20):
    # randomly simulate MM sets of person parameters
    apM = array([random.lognormvariate(0, ap_sd) for i in range(ysize*MM)]).reshape(MM,ysize)
    vpM = array([random.lognormvariate(0, vp_sd) for i in range(ysize*MM)]).reshape(MM,ysize)

    x, t = simuM_Mp(ai = ai, apM = apM, vi = vi, vpM = vpM, ter = ter,
                    ysize = ysize, nit = nit, n_options = n_options,
                    MM = MM, t_s = t_s, max_t = max_t, eps = eps, rej_M = rej_M)
    return x, t
#note: use this function because numba does not support generating random lognormal
```

Part II. Frequently Used Functions

```
import numpy as np
from numpy import array
from numba import jit
from numba.typed import List
import numba as nb
import random
import scipy
```

Function for Calculating ESS

```
#%%[General] Functions that calculates ESS based on normalized weights
@jit(nopython = True)
def ESS(ws):
    """
    ws : numpy.ndarray. Normalized weights of each particle.
    """

    if round(np.sum(ws),5) != 1:
        if np.sum(ws) == 0:
            print("WARNING: ESS is 0 due to all 0 weights.")
            ess = 0
        else:
            raise ValueError("Weights are not normalized")
    else:
        ess = 1/np.sum(ws ** 2)
    return ess
```

Function for Calculating Distance Defined by the Limited Information Criterion[23]

```
#%%[User-defined] Functions that gets M distances btw M simulated X and observed Y
def dist_limit(X, RT, Y, RTY, nit, MM = 30, which_dist = None):
    # sample variance: unbiased with ysize-1 as denominator
    covY =  np.cov(np.transpose(Y), bias = False)
    covRTY = np.cov(np.transpose(RTY), bias = False)

    # initialize
    Dx = np.zeros(MM)
    Drt_cov = np.zeros(MM)
    Drt_mean = np.zeros(MM)

    for m in range(MM):
        if which_dist == None or which_dist == 1:
            # X distance: l2 for uppder trangle elements of cov(X) and cov(Y). 52.1 µs ± 550 ns
            covXm = np.cov(np.transpose(X[m]), bias = False)
            Dx[m] = np.sqrt(np.sum(np.triu((covXm - covY)**2))/(nit*(nit+1)/2))
                # nit*(nit+1)/2 upper diagonal elements (include diagonal)

        if which_dist == None or which_dist == 2:
            # RT distance: l2 for cov. 52 µs ± 264 ns
            covRTm = np.cov(np.transpose(RT[m]), bias = False)
            Drt_cov[m] = np.sqrt(np.sum(np.triu((covRTm - covRTY)**2))/(nit*(nit+1)/2))
                # nit*(nit+1)/2 upper diagonal elements (include diagonal)
```

[23] The function for calculating distance defined by the limited information criterion has the same name as the function for calculating distance defined by the enhanced limited information criterion. These two functions can be saved in different python scripts when sourced/imported by the main script of SMC-ABC algorithm. The function *want0* has similar situations.

```
        if which_dist == None or which_dist == 3:
            # RT distance: l2 for RT column means. 116 µs ± 3.45 µs
            colM_RTm = array([np.mean(RT[m][:, i]) for i in range(nit)])
            colM_RTY = array([np.mean(RTY[:, i]) for i in range(nit)])
            Drt_mean[m] = np.sqrt(np.mean((colM_RTY - colM_RTm)**2))

    Dx = np.expand_dims(Dx, 1)              # rho(upper(cov(X)), upper(cov(Y)))
    Drt_cov = np.expand_dims(Drt_cov, 1)    # rho(colMeans(RTX), colMeans(RTX))
    Drt_mean = np.expand_dims(Drt_mean, 1)  # rho(upper(cov(RTX)), upper(cov(RTY)))
    out = np.hstack((Dx, Drt_cov, Drt_mean))
    return out
```

Function for Calculating Conditional Covariances for the Enhanced Limited Information Criterion

```
#%% return the conditional cov of rt ([y1,y2] = [0, 0], [0, 1], [1, 0], [1, 1])
@jit(nopython = True)
def cond_cov(Y, RT, nit, all_patterns):
    cond_cov = list()
    for ii in range(nit):
        for jj in range(ii+1, nit):
            cov_ij =  np.zeros(len(all_patterns))
            for kk in range(len(all_patterns)):
                pattern = all_patterns[kk]
                c1 = (Y[:,ii] == pattern[0]) # condition 1: Y1 = xx
                c2 = (Y[:,jj] == pattern[1]) # condition 2: Y2 = xx
                count = np.sum(c1 & c2)      # N in that cell
                if count > 1:
                    mycol = array([ii,jj], dtype=np.int64)
```

```
                    tmp = np.cov(RT[(c1 & c2),][:, mycol], rowvar = False)[0,1]
                else:
                    if count == 1: # if this cell just has 1 observation. cov is 0.
                        tmp = 0
                    if count == 0: # if this cell just has no observation. cov is missing (denoted as 99).
                        tmp = 99
                cov_ij[kk] = tmp
            cond_cov.append(cov_ij)
    return cond_cov
```

Function for Calculating Conditional Variances for the Enhanced Limited Information Criterion

```
#%% return the conditional Var of RT (Y = 0 and Y = 1, respectively)
@jit(nopython = True)
def cond_var(Y, RT, nit, ysize):
    var0, var1 =  np.zeros(nit), np.zeros(nit)
    pct = np.sum(Y, axis = 0)/ysize
    for i in range(nit):
        if pct[i] != 0 and pct[i] != 1:
            var0[i] = np.var(RT[Y[:,i] == 0, i])
            var1[i] = np.var(RT[Y[:,i] == 1, i])
        else:
            if pct[i] == 1: # if no Y = 0, var(RT|Y = 0) is missing, denoted by 99
                var0[i] = 99
                var1[i] = np.var(RT[Y[:,i] == 1, i])
            if pct[i] == 0: # if no Y = 1, var(RT|Y = 1) is missing, denoted by 99
                var0[i] = np.var(RT[Y[:,i] == 0, i])
                var1[i] = 99
    return [var0,var1]
```

Function for Calculating Conditional Means for the Enhanced Limited Information Criterion

```
#%% return the conditional Mean of RT (Y = 0 and Y = 1, respectively)
@jit(nopython = True)
def cond_mean(Y, RT, nit, ysize):
    mean0, mean1 =  np.zeros(nit), np.zeros(nit)
    pct = np.sum(Y, axis = 0)/ysize
    for i in range(nit):
        if pct[i] != 0 and pct[i] != 1:
            mean0[i] = np.mean(RT[Y[:,i] == 0, i])
            mean1[i] = np.mean(RT[Y[:,i] == 1, i])
        else:
            if pct[i] == 1: # if no Y = 0, mean(RT|Y = 0) is missing, denoted by 99
                mean0[i] = 99
                mean1[i] = np.mean(RT[Y[:,i] == 1, i])
            if pct[i] == 0: # if no Y = 1, mean(RT|Y = 1) is missing, denoted by 99
                mean0[i] = np.mean(RT[Y[:,i] == 0, i])
                mean1[i] = 99
    return [mean0,mean1]
```

Function for Calculating Distance Defined by the Enhanced Limited Information Criterion

```
#%%[User-defined] Functions that gets M distances btw M simulated X and observed Y
def dist_limit(X, RT, Y, RTY, nit, MM, all_patterns, ysize):
    # observed data
    covY = np.cov(np.transpose(Y), bias = False)  #1, cov(Y) # unbiased with ysize-1 as denominator
    cond_covRTY = array(cond_cov(Y, RTY, nit = nit, all_patterns = all_patterns)) #2, cov(RT|Y)
    cond_varRTY = array(cond_var(Y, RTY, nit = nit, ysize = ysize))      # 3, var(RT|Y)
    cond_meanRTY = array(cond_mean(Y, RTY, nit = nit, ysize = ysize))    # 4, mean(RT|Y)
```

```
# initialize
Dx = np.zeros(MM)
Drt_cov = np.zeros(MM)
Drt_var = np.zeros(MM)
Drt_mean = np.zeros(MM)

for m in range(MM):
    # 1
    covXm = np.cov(np.transpose(X[m]), bias = False)
    Dx[m] = np.sqrt(np.sum(np.triu((covXm - covY)**2))/(nit*(nit+1)/2))

    # 2
    cond_covRTm = array(cond_cov(X[m], RT[m], nit = nit, all_patterns = all_patterns))
    Drt_cov[m] = np.sqrt(np.mean((cond_covRTm - cond_covRTY)**2))

    # 3
    cond_varRTm = array(cond_var(X[m], RT[m], nit = nit, ysize = ysize))
    Drt_var[m] = np.sqrt(np.mean((cond_varRTY - cond_varRTm)**2))

    # 4
    cond_meanRTm = array(cond_mean(X[m], RT[m], nit = nit, ysize = ysize))
    Drt_mean[m] = np.sqrt(np.mean((cond_meanRTm - cond_meanRTY)**2))

Dx = np.expand_dims(Dx, 1)
Drt_cov = np.expand_dims(Drt_cov, 1)
Drt_var = np.expand_dims(Drt_var, 1)
Drt_mean = np.expand_dims(Drt_mean, 1)
out = np.hstack((Dx, Drt_cov, Drt_var, Drt_mean))
return out
```

Functions for Calculating $ESS\left(\left\{W_t^{(n)}\right\}, \boldsymbol{\epsilon_t}\right) - s \times ESS\left(\left\{W_{t-1}^{(n)}\right\}, \boldsymbol{\epsilon_{t-1}}\right)$

Limited Information Criterion

```
#%%[General*] Functions that calculates f(e) = ESS({W_n^i}) - alpha * ESS({W_n-1^i})
# for the limited info criterion

# Input: weights from the previous iteration
#        particles from previous iteration
#        epsilon from previous iteration
#        alpha: shrinkage rate
#        M
# Return: ESS_n - alpha * ESS_{n-1}

def want0(e, e_p, ws_p, x_p, rt_p, y, rty, distance_func, which_dist, alpha, ESS = ESS):
    """
    Usage
    -----
    Functions that calculates f(e) = ESS({W_n^i}) - alpha * ESS({W_n-1^i})

    Parameters
    ----------
    e : float
        \epsilon_{n}. A plausible epsilon value for the current iteration.
    e_p : an array
        \epsilon_{n-1}. The epsilon value used for the previous iteration.
    ws_p : array
        W_{n-1} for all particles. The weights used for the previous iteration.
```

```
x_p : List
    Level 1: len(x_p) = N # number of particles
    Level 2: len(x_p[0]) = M # number of simulated data, given one particle.
rt_p: List
    Level 1: len(rt_p) = N # number of particles
    Level 2: len(rt_p[0]) = M # number of simulated data, given one particle.
y : 2-d array
    The observed response data.
rty: 2-d array
    The observed response time data.
distance_func : function
    The functions that calculates the distance between simulated data x and observed data y
which_dist: int. 1, 2, 3, ...
    the new e is the epsilon for which distance?

Returns
-------
ws : W_{n} for all particles.
fvalue : function value given the plausible e
"""

M = len(x_p[0])
N = len(x_p)
nit = x_p[0][0].shape[1]

ee = array(e_p)
ee[which_dist - 1] = e

ws = np.zeros(N) # initialize: un-normalized weights resulted by the plausible e
```

```
# Calculate the new weights based on the plausible epsilon e
for i in range(N):
    if ws_p[i] == 0:
        ws[i] = 0
    else:
        # calculate M distances
        di = distance_func(X = x_p[i], RT = rt_p[i], Y = y, RTY = rty,
                           nit = nit, MM = M, which_dist = None)
        # W_n^(i) is proportional to the result of the following
        numerator_i = np.sum(np.all(di < ee, axis = 1)) # all distance should be met
        denominator_i = np.sum(np.all(di < e_p, axis = 1))
        ws[i] = ws_p[i] * numerator_i / denominator_i

# make sure distance function and the input distance have the same length
if di.shape[1] != len(e_p):
    print('WARNNING: The number of epsilon does not match the number of columns returned by dist_func')

# Normalized weights
if np.sum(ws) == 0:
    Ws = ws
else:
    Ws = ws/np.sum(ws)

# Function value
fvalue = ESS(Ws) - alpha * ESS(ws_p)
# Return
out = {'fvalue': fvalue, # numpy.float64
       'Ws': Ws} # numpy.ndarray
return(out)
```

Enhanced Limited Information Criterion

```
def want0(e, e_p, ws_p, x_p, rt_p, y, rty, distance_func, which_dist, alpha,
          all_patterns, ysize, ESS = ESS): # dist change
    """
    Parameters
    ----------
    all_patterns: a tuple containing all possible response patterns for a pair of item

    """

    M = len(x_p[0])
    N = len(x_p)
    nit = x_p[0][0].shape[1]

    ee = array(e_p)
    ee[which_dist - 1] = e

    ws = np.zeros(N) # initialize: un-normalized weights resulted by the plausible e

    # Calculate the new weights based on the plausible epsilon e
    for i in range(N):
        if ws_p[i] == 0:
            ws[i] = 0
        else:
            # calculate M distances
            di = distance_func(X = x_p[i], RT = rt_p[i], Y = y, RTY = rty,
                               nit = nit, MM = M, all_patterns = all_patterns, ysize = ysize) # dist change
```

```
        # W_n^(i) is proportional to the result of the following
        numerator_i = np.sum(np.all(di < ee, axis = 1)) # all distance should be met
        denominator_i = np.sum(np.all(di < e_p, axis = 1))

        ws[i] = ws_p[i] * numerator_i / denominator_i

# make sure distance function and the input distance have the same length
if di.shape[1] != len(e_p):
    print('WARNNING: The number of epsilon does not match the number of columns returned by dist_func')

# Normalized weights
if np.sum(ws) == 0 :
    Ws = ws
else:
    Ws = ws/np.sum(ws)

# Function value
fvalue = ESS(Ws) - alpha * ESS(ws_p)

# Return
out = {'fvalue': fvalue, # numpy.float64
       'Ws': Ws} # numpy.ndarray
return(out)
```

Functions for Finding a Large Number in Place of Infinities in ϵ_0

Limited Information Criterion

```
#%% This function is used right after t = 0, where we accept all particles no matter how large the dist is.
# This func finds the largest distance from N particles * M data per particle,
# which can be used in place of inf for epsilon_0
def max_dist0(xs_0, rt_0, Y, RTY, distance_func, which_dist):
    N = len(xs_0) # number of particles
    M = len(xs_0[0])
    nit = xs_0[0][0].shape[1]

    ds = np.zeros(N)
    for i in range(N):
        disti = distance_func(X = xs_0[i], RT = rt_0[i], Y = Y, RTY = RTY,
                              nit = nit, MM = M, which_dist = which_dist)
        ds[i] = np.max(disti[:, (which_dist - 1)]) # the max distance resulted by particle i
    D = np.max(ds) # the max distance resulted by N particles
    return D
```

Enhanced Limited Information Criterion

```
def max_dist0(xs_0, rt_0, Y, RTY, distance_func, which_dist, all_patterns, ysize): # dist change
    N = len(xs_0) # number of particles
    M = len(xs_0[0])
    nit = xs_0[0][0].shape[1]

    ds = np.zeros(N)
    for i in range(N):
```

```
        disti = distance_func(X = xs_0[i], RT = rt_0[i], Y = Y, RTY = RTY,
                              nit = nit, MM = M, all_patterns = all_patterns, ysize = ysize) # dist change
        ds[i] = np.max(disti[:, (which_dist - 1)]) # the max distance resulted by particle i
    D = np.max(ds) # the max distance resulted by N particles
    return D
```

Functions for Finding an Intermediate Tolerance ϵ_t

Limited Information Criterion

```
#%%[General] Bisection function for finding an epsilon value such that want0 is 0
def bisection(e_previous,
              want0, ws_p, x_p, rt_p, y, rty, distance_func, which_dist, alpha, # params passed to want0
              tol = 1,
              etol = 0.0001): # dividing |x1-x2| < etol to find x3 = (x1+x2)/2 is meaningless (more precise
than necessary)
    e_previous = array(e_previous)
    ### Find x1 and x2 such that f(x1) < 0 and f(x2) > 0. where f is want0, x are possible eplison values
    # note: f(e = e_previous) = N(1-alpha) > 0 is always true, because the weights (which result in ESS)
    # does not change if we do not change epsilon.
    if e_previous[which_dist - 1] == float('inf'):  # if e_previous is infinity
        x2 = max_dist0(xs_0 = x_p, rt_0 = rt_p, Y = y, RTY = rty,
                       distance_func = distance_func, which_dist = which_dist)
    else:  # if e_previous is not infinity
        x2 = e_previous[which_dist - 1]

    x1 = 0

    ### Find a value x3 between x1 and x2 such that |f(e = x3)| < tol, use bisection
```

```
    while True:
        x3 = (x1 + x2)/2 # bisection
        tmp = want0(e = x3, e_p = e_previous, ws_p = ws_p, x_p = x_p, rt_p = rt_p, y = y,
                    rty = rty, distance_func = distance_func, which_dist = which_dist, alpha = alpha)
        print("The final ",tmp['fvalue'],'x3 = ', x3)

        if tmp['fvalue'] > 0:
            x2 = x3
        else:
            x1 = x3

        if abs(tmp['fvalue']) < tol or (abs(x1-x2) < etol and tmp['fvalue'] > 0):
            break # stop if ESS error is less than 1, or x1 and x2 are already close enough for yielding a
precise enough epsilon. Python cannot handle too small numbers

    out = {'e': x3,  # numpy.float64
           'f(e)': tmp['fvalue'], # numpy.float64
           'Ws': tmp['Ws']} # numpy.ndarray
    return(out)
```

Enhanced Limited Information Criterion

```
def bisection(e_previous,
              want0, ws_p, x_p, rt_p, y, rty, distance_func, which_dist, alpha, all_patterns, ysize,
              tol = 1,
              etol = 0.0001):
    e_previous = array(e_previous)
    ### Find x1 and x2 such that f(x1) < 0 and f(x2) > 0. where f is want0, x are possible eplison values
    if e_previous[which_dist - 1] == float('inf'):  # if e_previous is infinity
```

```
        x2 = max_dist0(xs_0 = x_p, rt_0 = rt_p, Y = y, RTY = rty,
                       distance_func = distance_func, which_dist = which_dist,
                       all_patterns = all_patterns, ysize = ysize) # dist change
    else:  # if e_previous is not infinity
        x2 = e_previous[which_dist - 1]

    x1 = 0

    ### Find a value x3 between x1 and x2 such that |f(e = x3)| < tol, use bisection
    while True:
        x3 = (x1 + x2)/2 # bisection
        tmp = want0(e = x3, e_p = e_previous, ws_p = ws_p, x_p = x_p, rt_p = rt_p, y = y,
                    rty = rty, distance_func = distance_func, which_dist = which_dist, alpha = alpha,
                    all_patterns = all_patterns, ysize = ysize) # dist change
        print("The final ",tmp['fvalue'],'x3 = ', x3)

        if tmp['fvalue'] > 0:
            x2 = x3
        else:
            x1 = x3

        if abs(tmp['fvalue']) < tol or (abs(x1-x2) < etol and tmp['fvalue'] > 0):
            break # stop if ESS error is less than 1, or x1 and x2 are already close enough for yielding a
precise enough epsilon. Python cannot handle too small numbers

    out = {'e': x3,  # numpy.float64
           'f(e)': tmp['fvalue'], # numpy.float64
           'Ws': tmp['Ws']} # numpy.ndarray
    return(out)
```

Function for Calculating Starting Values

```
#%% Get starting value of ai, vi, from the boserved Y and RTY
def start_value(Y, RTY, mu = 0, sigma2 = 0.18, n_options = 2):
    """
    Parameters
    ----------
    Y : TYPE
        observed responses.
    RTY : TYPE
        Observed response times.
    mu : TYPE, optional
        the mu for the lognormal distribution of ap*vp. Assuming ap ~ logN(0,0.3^2) and vp ~ logN(0,0.3^2).
The default mu is 0, and the default sigma2 is 0.09+0.09=0.18
    sigma2 : TYPE
        see mu.
    n_options : TYPE, optional
        DESCRIPTION. The default is 2.

    Returns
    -------
    app_ai : TYPE
        DESCRIPTION.
    app_vi : TYPE
        DESCRIPTION.
    """
    # use (logitP(Y=1)+log(n_options - 1))/exp(mu + sigma2/2)to approximate 1/(ai*vi)
    p = np.mean(Y, axis = 0)
    p[p==1] = 0.999
```

```
    p[p==0] = 0.001
    p[p==0.5] = 0.499
    logity0 = np.log(p/(1-p)) + np.log(n_options - 1)
    logity = logity0/np.exp(mu + sigma2/2)

    # use min RT to approximate ter
    app_ter = np.min(RTY, axis = 0)

    # use (1-exp(-logity))/(2 * (ET - ter) * (1 + exp(-logity))) to approximate ai/vi
    ET = np.mean(RTY, axis = 0)
    #bb = (1-np.exp(-logity))/((1+np.exp(-logity)) * 2 * (ET - app_ter)) # ai/vi
    bb = (2*p-1)*(np.exp(mu + sigma2/2))/(2*(ET-app_ter))
    # approximate vi and ai:
    app_vi = 1/np.sqrt(logity*bb)
    app_ai = 1/np.sqrt(logity/bb)

    return app_ai, app_vi, app_ter
```

Function for Standardizing Array

```
#%% column-wise standardize 2-d array
def col_scale(x):
    mx = np.mean(x, axis = 0)
    sdx = np.sqrt(np.var(x, axis = 0))
    std_x = (x - mx)/sdx
    return std_x
```

Function for Finding the Intersection of Two Density Curves

```
#%% Given two datasets, find the intersection of their density curves (only return x axix value)
# a and b are two data. a is generally smaller than b.
def x_intersect(a, b, hist_bin = 15, left_a = 0.3, right_a = 0.975):
    # get the density of a and b: need f(x) = y as density
    y1, x1 = np.histogram(a, bins=hist_bin, density=True) # returns density, corresponding x value (left
and right borders)
    y2, x2 = np.histogram(b, bins=hist_bin, density=True) # returns density, corresponding x value (left
and right borders)

    # bin width
    bin1 = x1[1] - x1[0]
    bin2 = x2[1] - x2[0]

    # get the exact
    x1 = x1[1:len(x1)] - bin1/2 # remove the first element
    x2 = x2[1:len(x2)] - bin2/2 # remove the first element

    # define the range where we find the intersection: 2.5% and 97.5% of a
    xaxis = np.linspace(np.quantile(a,left_a),
                        np.quantile(a,right_a), 100)
    # get 2 sets of densty on the same x range
    ordered_y1 = np.interp(xaxis, x1, y1) # the right one should be the density!
    ordered_y2 = np.interp(xaxis, x2, y2)

    # find where they meet
    xxx = xaxis[np.argmin(np.abs(ordered_y2 - ordered_y1))]
    xxx = np.round(xxx, 4)
    return xxx
```

Function for Loading and Saving Objects

```
#%% save and read python objects
import pickle

def save_object(obj,filename):
    try:
        with open(filename, "wb") as f: # define file path here
            pickle.dump(obj, f, protocol=pickle.HIGHEST_PROTOCOL)
    except Exception as ex:
        print("Error during pickling object (Possibly unsupported):", ex)

def load_object(filename):
    try:
        with open(filename, "rb") as f:
            return pickle.load(f)
    except Exception as ex:
        print("Error during unpickling object (Possibly unsupported):", ex)
```

Part III. SMC-ABC Algorithm

The following is for the enhanced limited information criterion. The codes for the criterion are very similar, except for the lines with comments `# add dist` and `# dist change`.

```
#%% Prepare
print("====================== Python codes start ======================")
print("This is revised python code.")
import numpy as np
from numpy import array
from numba import jit
from numba.typed import List
import numba as nb
from matplotlib import pyplot as plt
import seaborn as sns
import scipy.stats as stats
from datetime import datetime
from multiprocessing import Pool # iPython does not support multiprocessing, but python should be.
from sys import getsizeof
import os
import psutil
import random
import multiprocessing
import sys

# Run chunks from the following 3, depending on where the codes are being implemented
#%% [local][iPython]
runfile('/Users/chen/Documents/Dissertation/Codes_cond/4info/a_functions.py')
runfile('/Users/chen/Documents/Dissertation/Codes_cond/4info/a_simulate.py')
ncores = 8
```

```
jobseed = 1
from multiprocess import Pool

#%% [HPC]
ncores = int(sys.argv[1])
print("Number of requested CPU cores:", ncores) # 120
jobseed = int(sys.argv[2])
print("Job seed for simulating observed data:", jobseed)

#%% [HPC] import functions
from a_functions import ESS
from a_functions import dist_limit
from a_functions import dist_full
from a_functions import want0
from a_functions import max_dist0
from a_functions import bisection
from a_functions import start_value
from a_functions import save_object
from a_functions import load_object
from a_functions import col_scale
from a_functions import which_cor
from a_functions import x_intersect
from a_simulate import simuM
from a_simulate import simuM_Mp
from a_simulate import simuM_varp

#%% Simulate Observed Data 🏃
nit = 5
ysize = 200
```

```
n_options = 2
all_patterns = ((0,0), # add dist
                (0,1),
                (1,0),
                (1,1))

# true item parameters
np.random.seed(1)
ai = array([np.exp(-0.5), np.exp(-0.5), np.exp(-0.5), np.exp(-0.5), np.exp(-0.5)]) # item time pressure
vi = np.exp(np.random.uniform(low = 0, high = 1, size = nit))  # item pure difficulty
ter = np.random.uniform(low = 0.5, high = 1.5, size = nit)     # residual time for each item
true_ps = [0.3, 0.3]

#%% simulate observed data using some seed
random.seed(jobseed+100) # seed for the main process
ap = array([random.lognormvariate(0, true_ps[0]) for i in range(ysize)])
vp = array([random.lognormvariate(0, true_ps[1]) for i in range(ysize)])

Y, RTY, PY = simuM(ai = ai, ap = ap, vi = vi, vp = vp, ter = ter,
              ysize = ysize, nit = nit, n_options = 2, seedid = jobseed, #set local seed for simulated data
              MM = 1, t_s = 1/100, max_t = 100, eps = 1e-15, rej_M = 1000)

Y = Y[0]
RTY = RTY[0]
logRTY = np.log(RTY) #!
emp_maxt = np.max(RTY)*1.05 # when simulating RT, crop RT by 2*observed max rt

# check
print("Observed RT quantile across items (2.5%, 97.5%, 100%):",
```

```
       np.round(np.quantile(RTY, q = [0.025,0.975,1]), 3))
print("Observed mean RT by item", np.round(np.mean(RTY,axis=0), 3))
print("Observed pass rate by item", np.round(np.mean(Y,axis=0), 3))

# observed sample size for each pair of items x 4 response pattern # add dist
out_N = list()
for ii in range(nit):
    for jj in range(ii+1, nit):
        out_ij = list()
        for pattern in all_patterns:
            c1 = (Y[:,ii] == pattern[0])
            c2 = (Y[:,jj] == pattern[1])
            out_ij.append(np.sum(c1 & c2))
        out_N.append(out_ij)
out_N = array(out_N) # columns: four patterns, rows: each pair. 0 1, 0 2,...
np.min(out_N)
print("The sample sizes are:\n", out_N)
print("Each column represents a pattern: [0,0], [0,1], [1,0], [1,1]")
print("Each row represents a pair of items: [1,2], [1,3], [1,4],\
       [1,5], [2,3], [2,4], [2,5], [3,4],[3,5],[4,5]")
print("The min N = ", np.min(out_N))

#%% Decide initial values and epsilon_T 🏃
loc_ai, loc_vi, loc_ter = start_value(Y = Y, RTY = RTY, sigma2 = 0.3**2 + 0.3**2, n_options = 2) # get the
prior locations
loc_ai = array([np.mean(loc_ai)] * nit) #!!! ai constrain

#%% find epsilon_T
rep = 5000
```

```
ap_sd_guess = 0.3
vp_sd_guess = 0.3

### 1. Simulate true data: guessed same item, different person
loc_X2, loc_RTX2 = simuM_varp(ai = loc_ai, ap_sd = ap_sd_guess, vi = loc_vi,
                              vp_sd = vp_sd_guess, ter = loc_ter,
                              ysize = ysize, nit = nit, n_options = 2,
                              MM = rep, t_s = 1/100, max_t = emp_maxt, eps = 1e-15, rej_M = 100)

# calculate true distances
loc_dl2 = dist_limit(X = loc_X2, RT = np.log(loc_RTX2), Y = Y, RTY = logRTY, nit = nit, MM = rep,
                     all_patterns = all_patterns, ysize = ysize) # dist change

### 2. Simulate false data: guessed different items, different persons
loc_dl9, loc_df9 = [],[]

for pp in range(rep):
    diff_ai = array([random.uniform(np.min(loc_ai), np.max(loc_ai)) for i in range(nit)])
    diff_vi = array([random.uniform(np.min(loc_vi), np.max(loc_vi)) for i in range(nit)])
    diff_ter = array([random.uniform(np.min(loc_ter), np.max(loc_ter)) for i in range(nit)])


    loc_X9, loc_RTX9 = simuM_varp(ai = diff_ai, ap_sd = ap_sd_guess, vi = diff_vi,
                                  vp_sd = vp_sd_guess, ter = diff_ter,
                                  ysize = ysize, nit = nit, n_options = 2,
                                  MM = 1, t_s = 1/100, max_t = emp_maxt, eps = 1e-15, rej_M = 100)

    # Calculate distances
    loc_dl9_pp = dist_limit(X = loc_X9, RT = np.log(loc_RTX9), Y = Y, RTY = logRTY, nit = nit, MM = 1,
```

```
                            all_patterns = all_patterns, ysize = ysize)[0]  # dist change
    loc_dl9.append(loc_dl9_pp)
loc_dl9 = array(loc_dl9)

### 3. Decide epsilon_T for limited information distance
h_guess1 = x_intersect(loc_dl2[:,0], loc_dl9[:,0], right_a = 0.5)
h_guess2 = x_intersect(loc_dl2[:,1], loc_dl9[:,1], right_a = 0.95)
h_guess3 = x_intersect(loc_dl2[:,2], loc_dl9[:,2], right_a = 0.95)
h_guess4 = x_intersect(loc_dl2[:,3], loc_dl9[:,3], right_a = 0.95) # add dist

#%% Epsilon plot
fig, axs = plt.subplots(1, 4, figsize = (8,2.5), constrained_layout = True) # add dist
fig.suptitle(r"Limited information: empirical $\epsilon_T$", fontsize=10)
# cov(resp)
sns.kdeplot(loc_dl2[:,0], linewidth = 2, linestyle='-',fill= True, ax=axs[0])
sns.kdeplot(loc_dl9[:,0], color = "orange", linewidth = 2, linestyle='--',fill= True, ax=axs[0])
axs[0].set_xlabel('Approximated:' + str(round(h_guess1,3)))
axs[0].axvline(h_guess1, 0, 1000,color = 'red', linewidth = 0.8)

# cov(RT|Y)
sns.kdeplot(loc_dl2[:,1], linewidth = 2, linestyle='-',fill= True, ax=axs[1], clip = (0,0.2))
sns.kdeplot(loc_dl9[:,1], color = "orange", linewidth = 2, linestyle='--',
            fill= True, ax=axs[1], clip = (0,0.2))
axs[1].set_xlabel('Approximated:' + str(round(h_guess2,3)))
axs[1].axvline(h_guess2, 0, 1000,color = 'red', linewidth = 0.8)

# var(RT|Y)
sns.kdeplot(loc_dl2[:,2], linewidth = 2, linestyle='-',fill= True, ax=axs[2])
sns.kdeplot(loc_dl9[:,2], color = "orange", linewidth = 2, linestyle='--',fill= True, ax=axs[2])
```

```
axs[2].set_xlabel('Approximated:' + str(round(h_guess3,3)))
axs[2].axvline(h_guess3, 0, 1000,color = 'red', linewidth = 0.8)

# colMeans(RT|Y)   # add dist
sns.kdeplot(loc_dl2[:,3], linewidth = 2, linestyle='-',fill= True, ax=axs[3])
sns.kdeplot(loc_dl9[:,3], color = "orange", linewidth = 2, linestyle='--',fill= True, ax=axs[3])
axs[3].set_xlabel('Approximated:' + str(round(h_guess4,3)))
axs[3].axvline(h_guess4, 0, 1000,color = 'red', linewidth = 0.8)

# [HPC] save figure
plt.savefig("/home/chen/4info/output/limited_seed"+ str(jobseed) +"/epsilon_T.pdf")
plt.close()

#%% compare initial guess and true values: save figures
fig, axs = plt.subplots(1, 3, figsize = (3.25*3,3), constrained_layout = True)
fig.suptitle("Guessed initial values against true values", fontsize=10)
axs[0].scatter(ai, loc_ai)
axs[0].plot(array(range(30,80))/100, array(range(30, 80))/100, '-', color = 'skyblue', linewidth = 1)
axs[1].scatter(vi, loc_vi)
axs[1].plot(array(range(7,25))/10, array(range(7,25))/10, '-', color = 'skyblue', linewidth = 1)
axs[2].scatter(ter, loc_ter)
axs[2].plot(array(range(5, 11))/10, array(range(5, 11))/10, '-', color = 'skyblue', linewidth = 1)

# [HPC] save figure
plt.savefig("/home/chen/4info/output/limited_seed"+ str(jobseed) +"/initial_guess.pdf")
plt.close()

#%% SMC setting 🏃
e_final = [np.round(h_guess1,4), np.round(h_guess2,4), np.round(h_guess3,4), np.round(h_guess4,4)]
```

```
print("epsilon_T is", np.round(e_final,4))
M = 10          # number of simulated datasets given ONE particle
N = 600         # number of particles
N_final = 280   # minimum number of particles
prior_sd = 1

#%% Initialize
# Initialize: store for all N particles at all times
all_lais = list()  # particles at all times. A list of arrays. Each array contains thetas for all particles
all_lvis = list()  # particles at all times. A list of arrays. Each array contains thetas for all particles
all_lps = list()
all_ters = list()
all_ws = list()   # weights at all times t. A list of arrays. Each array contains weights for all particles
all_e = list() # A list of float
all_movement = list()

# Initialize RNG for each particle position: Nstates_0
random.seed(jobseed)
Nseeds = random.sample(range(10000000000), N) # sample without replacement

Nstates_0 = list()
for i in range(N):
    random.seed(Nseeds[i])
    Nstates_0.append(random.getstate())

#%% First iteration

### 1. Get N particles, their X, and weights
# i. Sample from prior
```

```
lai_0 = [[random.normalvariate(np.log(loc_ai[i]), prior_sd) for j in range(N)] for i in range(1)] #!!! ai
constrain
lai_0 = array(np.transpose(array(lai_0)), order = 'C')

lvi_0 = [[random.normalvariate(np.log(loc_vi[i]), prior_sd) for j in range(N)] for i in range(nit)]
lvi_0 = array(np.transpose(array(lvi_0)), order = 'C')

ter_0 = [[random.uniform(0, loc_ter[i]) for j in range(N)] for i in range(nit)]
ter_0 = array(np.transpose(array(ter_0)), order = 'C')

lps_0 = [[random.normalvariate(np.log(0.3), prior_sd) for j in range(N)] for i in range(2)]
lps_0 = array(np.transpose(array(lps_0)), order = 'C')

ai_0 = np.exp(lai_0)
vi_0 = np.exp(lvi_0)
ps_0 = np.exp(lps_0)

# ii. Simulate x from each of N sets of particles for M times
def task(args): # single i. the computation for each ith particle
    i, statei = args  # RNG
    random.setstate(statei) # RNG

    xs, rt = simuM_varp(ai = array(list(ai_0[i,]) * nit) , ap_sd = ps_0[i,0], vi = vi_0[i,],
                        vp_sd = ps_0[i,1], ter = ter_0[i,], # quick and dirty: uses global objects
                        ysize = ysize, nit = nit, n_options = 2,
                        MM = M, t_s = 1/100, max_t = emp_maxt, eps = 1e-15, rej_M = 100)

    statei = random.getstate() # RNG
    return array(xs), array(rt), i, statei # RNG
```

```
# initialize
xs_0 = list(); rt_0 = list(); qc = list(); t1 = datetime.now()#.strftime("%D, %H:%M:%S")
Nstates_t = list() # RNG

# i in 1:1000. parallel simulation
if __name__ == '__main__':
    with Pool(ncores) as pool:
        for xs, rt, i, statei in pool.map(task, [(i, Nstates_0[i]) for i in range(N)]): # RNG
            xs_0.append(xs)
            rt_0.append(rt)
            qc.append(i)
            Nstates_t.append(statei) # RNG
# check/QC
t2 = datetime.now(); print("Time duration for the first iteration:", t2-t1)
if np.unique(np.diff(qc)) != 1: print("ERROR!!! Check the append order of parallel computation!") # should
only include 1!

#%% record first iteration
# iii. Assign weight to this particle
ws_0 = array([1/N] * N)

### 3. Save stage t output (t > 0)
all_lais.append(lai_0)
all_lvis.append(lvi_0)
all_ters.append(ter_0)
all_lps.append(lps_0)
all_ws.append(ws_0)
all_e0 = list([float('inf'), float('inf'), float('inf'), float('inf')]) # add dist
all_e.append(all_e0)
```

```
all_moves = [[1,1,1,1,1,1]]
last_t_xs = xs_0
last_t_rt = rt_0

#%% Prepare for parallel computation [Limited-information version]
# In later iterations, for particle i, propose & update.
# This operation is repeated for all N particles, and can be parallelized.
def task2(args):
    i, statei = args  # RNG
    random.setstate(statei) # RNG

    if ws_t[i] > 0: # for alive particles
        # 1. propose a new set of particle values from MVN
        # identify xx% nearest alive particles
        dis = np.mean((std_particles30 - std_particles30[i,]) ** 2, axis = 1) # l2 distance between ith
particle and all other particles
        dis_alive = dis[ws_t > 0]
        cut = np.quantile(dis_alive,[near_percentage]) # xx% nearest particles
        ind = np.logical_and(dis <= cut, ws_t > 0) # flag the xx% nearest as True

        # get proposal variances and means
        tmp_p = particles30[ind,]
        tmp_w = ws_t[ind,]/sum(ws_t[ind,])
        weighted_neighbor_ind = [random.choices(range(len(tmp_p)), weights=tmp_w)[0] for _ in range(3 *
len(tmp_p))]
        sigma2_30 = cov_coef * np.cov(np.transpose(tmp_p[weighted_neighbor_ind,]), bias = False)
        means_30 = array(list(p1[i,]) + list(p2[i,]) + list(p3[i,]) + list(p4[i,])) # add dist

        # propose from a MVN distribution
```

```
        L = np.linalg.cholesky(sigma2_30) # the Cholesky decomposition of the covariance matrix
        z = [random.normalvariate(0, 1) for i in range(len(means_30))] # std normal
        plai_lvi_ter_lps = array([means_30[j] + sum([L[j][k] * z[k] for k in range(len(means_30))]) for j
in range(len(means_30))])
        plai = plai_lvi_ter_lps[range(1)] #!!! ai constrain
        plvi = plai_lvi_ter_lps[range(1, 1 + 1 * nit)]
        pter = plai_lvi_ter_lps[range(1 + 1 * nit, 1 + 2 * nit)]
        plps = plai_lvi_ter_lps[range(1 + 2 * nit, 2 * nit + 3)]

        #plai = plai_lvi_ter_lps[range(nit)] #!!! ai constrain
        #plvi = plai_lvi_ter_lps[range(nit, 2 * nit)]
        #pter = plai_lvi_ter_lps[range(2 * nit, 3 * nit)]
        #plps = plai_lvi_ter_lps[range(3 * nit, 3 * nit + 2)]

        if any(pter < 0) or any(pter > loc_ter):
            ratio = 0
        else:
            # 2. calculate the acceptance rate
            # i. simulate x from this PROPOSED particle for M times
            pai = np.exp(plai)
            pvi = np.exp(plvi)
            pps = np.exp(plps)
            pxi_M, prti_M = simuM_varp(ai = array(list(pai) * nit),  #!!! ai constrain # pai,
                                       ap_sd = pps[0], vi = pvi, vp_sd = pps[1], ter = pter,
                                       ysize = ysize, nit = nit, n_options = 2, #seedid = None,
                                       MM = M, t_s = 1/100, max_t = emp_maxt, eps = 1e-15, rej_M = 100)

            # ii. get M distances for calculation
            pdistance_i_M = dist_limit(X = pxi_M, RT = np.log(prti_M), Y = Y, RTY = logRTY, nit = nit,
```

```
                                MM = M, all_patterns = all_patterns, ysize = ysize)  # dist change
            distance_i_M = dist_limit(X = x_previous[i], RT = np.log(rt_previous[i]), Y = Y, RTY = logRTY,
                                nit = nit, MM = M, all_patterns = all_patterns,
                                ysize = ysize) # dist change

            # iii. get acceptance rate
            pa = pdistance_i_M[:,0] < et[0]
            pb = pdistance_i_M[:,1] < et[1]
            pc = pdistance_i_M[:,2] < et[2]
            pd = pdistance_i_M[:,3] < et[3] # add dist

            a = distance_i_M[:,0] < et[0]
            b = distance_i_M[:,1] < et[1]
            c = distance_i_M[:,2] < et[2]
            d = distance_i_M[:,3] < et[3] # add dist
            r_indicator = np.sum(pa & pb & pc & pd) / np.sum(a & b & c & d) # add dist

            if r_indicator == 0 : # if ter falls outside prior limit, kill the proposal
                ratio = 0
            else:
                log_r_indicator = np.log(r_indicator)

                # log pi(theta*). log prior density at the proposed value
                log_nu_prior = np.log(array([stats.norm(np.log(loc_ai[jj]),prior_sd).pdf(plai[jj]) for jj
in range(1)]))  #!!! ai constrained
                log_nu_prior2 = np.log(array([stats.norm(np.log(loc_vi[jj]),prior_sd).pdf(plvi[jj]) for jj
in range(nit)]))
                log_nu_prior4 = np.log(array([stats.norm(np.log(0.3),prior_sd).pdf(plps[jj]) for jj in
range(2)])) # add dist
```

```
                # log pi(theta). log prior density at the old value
                log_deno_prior = np.log(array([stats.norm(np.log(loc_ai[jj]),prior_sd).pdf(p1[i,jj]) for jj
in range(1)]))  #!!! ai constrained
                log_deno_prior2 = np.log(array([stats.norm(np.log(loc_vi[jj]),prior_sd).pdf(p2[i,jj]) for
jj in range(nit)]))
                log_deno_prior4 = np.log(array([stats.norm(np.log(0.3),prior_sd).pdf(p4[i,jj]) for jj in
range(2)])) # add dist

                # log of acceptance rate
                log_r = log_r_indicator + np.sum(log_nu_prior) - np.sum(log_deno_prior) + \
                        np.sum(log_nu_prior2) - np.sum(log_deno_prior2) + \
                        np.sum(log_nu_prior4) - np.sum(log_deno_prior4)
                log_ratio = min(0, log_r) # quick and dirty: assume prior is uniform and the proposal is
symmetric
                ratio = np.exp(log_ratio)

        u = random.uniform(0,1)

        # iv. accept or not
        if u < ratio:
            accepted = 1
            laii = plai
            lvii = plvi
            teri = pter
            lpsi = plps
            xi_M = pxi_M
            rti_M = prti_M
        else:
```

```
            accepted = 0
            laii = p1[i]
            lvii = p2[i]
            teri = p3[i]
            lpsi = p4[i]
            xi_M = x_previous[i]
            rti_M = rt_previous[i]

    else: # for dead particles: copy from previous results
        accepted = 0
        laii = all_lais[t-1][i]
        lvii = all_lvis[t-1][i]
        teri = all_ters[t-1][i]
        lpsi = all_lps[t-1][i]
        xi_M = last_t_xs[i]
        rti_M = last_t_rt[i]

    statei = random.getstate() # RNG
    return laii, lvii, teri, lpsi, array(xi_M), array(rti_M), accepted, i, statei # RNG

# inputs: p1, p2, p3, x_previous, rt_previous

#%% Later iterations
t = 1
alpha = 0.8                     # ESS shrinkage after each iteration. s
sufficient_move_cor = 0.843     # r = 0.843
near_percentage = 0.3           # 30% percent neighbor. Choose 30% s.t. when the number of alive particles
are small (e.g., 200), the number of ovservations (200 * 20% = 40) is greater than the number of parameters
(e.g., 23).
```

```
cov_coef = 0.1                  # sigma2 = x * cov(neighbor_particles)

t_start = datetime.now()
while True:
    print("---------", " t = ", t, " ---------")
    print(datetime.now())
    if all(array(all_e[t-1]) - e_final == 0): break

    which_dist = t%4  # add dist
    if which_dist == 0: which_dist = 4 # add dist

    while (array(all_e[t-1]) - e_final)[which_dist - 1] == 0: # if already met, find next
        which_dist = (which_dist + 1)%4 # add dist

    ################ Setp 1. Find a good epsilon s.t. ESS_n = alpha * ESS_{n-1} ################
    find_e = bisection(e_previous = all_e[t-1],
                       want0 = want0, ws_p = all_ws[t-1], x_p = last_t_xs, rt_p = np.log(last_t_rt),
                       y = Y, rty = logRTY,
                       distance_func = dist_limit, which_dist = which_dist, alpha = alpha,# params to want0
                       all_patterns = all_patterns, ysize = ysize, # dist change
                       tol = 20,
                       etol = 0.0001)

    e = find_e['e']
    if e < e_final[which_dist - 1]:
        e = e_final[which_dist - 1]
    ws_t = find_e['Ws']

    # break if failed
```

```
    if all(ws_t == 0):
        print("Failed to force epsilon to be smaller with the current\
               setting (e.g., small alpha, large prior_sd)")
        break

    # update the set of e we use for the current iteration
    et = array(all_e[t-1])
    et[which_dist - 1] = e
    print(et)
    print("ESS = ", ESS(ws_t))

    ############################## Step 2. Resample if ESS < N_final ##########################
    # Resample using current weights, but sample particles & X from the previous step
    if ESS(ws_t) < N_final:
        # resample
        resample_ind = [random.choices(range(N), weights = ws_t)[0] for _ in range(N)] # Sample from
previous samples with CURRENT weights
        lai_t = [all_lais[t-1][i,] for i in resample_ind]
        lvi_t = [all_lvis[t-1][i,] for i in resample_ind]
        ter_t = [all_ters[t-1][i,] for i in resample_ind]
        lps_t = [all_lps[t-1][i,] for i in resample_ind]
        xs_t = [last_t_xs[i] for i in resample_ind]
        rt_t = [last_t_rt[i] for i in resample_ind]

        # denote abusively as the (t-1)th round
        all_lais[t-1] = array(lai_t)
        all_lvis[t-1] = array(lvi_t)
        all_ters[t-1] = array(ter_t)
        all_lps[t-1] = array(lps_t)
```

```
    last_t_xs = array(xs_t)
    last_t_rt = array(rt_t)

    # reassign weight
    ws_t = array([1/N] * N)

#################### Step 3. Perturb using K(, .) for alive particles ######################
# All particles (alive and deal) and data from last iteration
p1 = all_lais[t-1]
p2 = all_lvis[t-1]
p3 = all_ters[t-1]
p4 = all_lps[t-1]  # add dist
x_previous = last_t_xs
rt_previous = last_t_rt

######### Keep pertubring until sufficient movement
movement_t = []; move_count = 0; acceptance_rates_t = []

while True:
    ### All (live + dead) particles from latest movement
    particles30 = np.hstack((p1, p2, p3, p4))
    std_particles30 = col_scale(particles30)

    ### Prepare
    lai_t = np.zeros(N*1).reshape(N,1)  #!!! ai constrain, 1 or nit
    lvi_t = np.zeros(N*nit).reshape(N,nit)
    ter_t = np.zeros(N*nit).reshape(N,nit)
    lps_t = np.zeros(N*2).reshape(N,2)
    xs_t = list()
```

```
rt_t = list()
count_acc = 0; qc = []
tmp_Nstates = list() # RNG

### For i in 1:N
if __name__ == '__main__':
    with Pool(ncores) as pool:
        for laii, lvii, teri, lpsi, xi_M, rti_M, accepted, i, statei in
        pool.map(task2, [(i, Nstates_t[i]) for i in range(N)]):
            lai_t[i,] = laii
            lvi_t[i,] = lvii
            ter_t[i,]= teri
            lps_t[i,]= lpsi
            xs_t.append(xi_M)
            rt_t.append(rti_M)
            qc.append(i)
            count_acc = count_acc + accepted
            tmp_Nstates.append(statei) # RNG

Nstates_t = tmp_Nstates # RNG

# Check/QC
if np.unique(np.diff(qc)) != 1: print("ERROR!!! Check the append order of parallel computation!")
acceptance_rate = np.round(count_acc/sum(ws_t!=0),3)
acceptance_rates_t.append(acceptance_rate)
print("accepted/total alive: ",acceptance_rate * 100,"%")

### Track the movement
move_count += 1
```

```
        params_t = np.hstack((lai_t, lvi_t, ter_t, lps_t)) # right after movement
        params_p = np.hstack((p1, p2, p3, p4)) # right before movement
        movement = array([np.corrcoef(params_t[ws_t!=0, k], params_p[ws_t!=0, k])[0,1] for k in range(nit *
2 + 3)]) #!!! ai constrain nit*2+3 # movement for live particles
        movement_t.append(movement) # record

        ### Check if movement is sufficient
        move_t = array([list(movement_t[iii]) for iii in range(len(movement_t))]) # reformat
        grand_movement = array([np.prod(move_t[:,cc]) for cc in range(nit * 2 + 3)]) #!!! ai constrain
nit*2+3
        print(np.round(grand_movement,3))
        if np.mean(grand_movement < sufficient_move_cor) >= 0.90:
            break

        ### Update moved p1, p2, p3, p4
        p1 = lai_t
        p2 = lvi_t
        p3 = ter_t
        p4 = lps_t # add dist
        x_previous = xs_t
        rt_previous = rt_t

    ####### After sufficient movement, save stage t output (t > 0)
    all_lais.append(array(lai_t))
    all_lvis.append(array(lvi_t))
    all_ters.append(array(ter_t))
    all_lps.append(array(lps_t))
    last_t_xs = xs_t
    last_t_rt = rt_t
```

```
all_ws.append(ws_t)
all_e.append(list(et))
all_movement.append(grand_movement)
all_moves.append([move_count] +
                 list(np.round(np.quantile(acceptance_rates_t, [0, 0.25, 0.5, 0.75, 1]), 3)))
print("At iteration t = ",t, ", we moved", move_count," times.")
t += 1


#### Plot: point estimate
# For log ai, log vi, and ter
resample_ind = [random.choices(range(N), weights = all_ws[t-1])[0] for _ in range(N)]
resample_lai = array([all_lais[t-1][i,] for i in resample_ind])
resample_lvi = array([all_lvis[t-1][i,] for i in resample_ind])
resample_ter = array([all_ters[t-1][i,] for i in resample_ind])
resample_lps = array([all_lps[t-1][i,] for i in resample_ind])

wmedian = [np.quantile(resample_lai[:,i], 0.5) for i in range(1)] #!!! ai constrain
wmedian2 = [np.quantile(resample_lvi[:,i], 0.5) for i in range(nit)]
wmedian3 = [np.quantile(resample_ter[:,i], 0.5) for i in range(nit)]
wmedian4 = [np.quantile(resample_lps[:,i], 0.5) for i in range(2)]
print("Population parameter estimates (median): ",np.round(np.exp(wmedian4),3))

# point estimate
colors =  plt.get_cmap("jet")(array(range(nit))/nit)
# markers = ['o','H','p','h','*','X','d','v','D','^']
markers = ['o','*','X','d','v']
fig, axs = plt.subplots(1, 3, figsize = (3 * 3.25,3), constrained_layout = True)
fig.suptitle(r"Point estimate (weighted median) at $t=$" + str(t-1), fontsize=10)
```

```
    fig.text(0.5, -0.1, "Population parameter estimates: " + str(np.round(np.exp(wmedian4),3)),
             ha='center', fontsize=10)
    fig.text(0.5, -0.2, "$\epsilon_t = $" + str(np.round(et,3)))
    axs[0].plot(array(range(-11,0))/10, array(range(-11,0))/10, '-', color = 'skyblue', linewidth = 1)
    axs[1].plot(array(range(-1,11))/10, array(range(-1,11))/10, '-', color = 'skyblue', linewidth = 1)
    axs[2].plot(array(range(25,100))/100, array(range(25,100))/100, '-', color = 'skyblue', linewidth = 1)
    for spine in axs[0].spines.values(): spine.set_color('#7A8B8B')
    for spine in axs[1].spines.values(): spine.set_color('#7A8B8B')
    for spine in axs[2].spines.values(): spine.set_color('#7A8B8B')

    for item in range(nit):
        axs[0].scatter(wmedian, np.log(ai[item]), color = colors[item], marker = markers[item]) #!!!
        #axs[0].scatter(wmedian[item], np.log(ai[item]), color = colors[item], marker = markers[item]) #!!!
ai constrain
        #axs[0].text(wmedian[item]+0.03, np.log(ai[item]) - 0.02, str(item+1), color = "grey", fontsize =
8) #!!!
        axs[0].set_xlabel(r'Estimate of $log\alpha$')
        axs[0].set_ylabel('True values')

        axs[1].scatter(wmedian2[item], np.log(vi[item]), color = colors[item], marker = markers[item])
        axs[1].text(wmedian2[item]+0.03, np.log(vi[item]) - 0.02, str(item+1), color = "grey", fontsize =
8)
        axs[1].set_xlabel(r'Estimate of $log\beta$')

        axs[2].scatter(wmedian3[item], ter[item], color = colors[item], marker = markers[item])
        axs[2].text(wmedian3[item]+0.02, ter[item] - 0.02, str(item+1), color = "grey", fontsize = 8)
        axs[2].set_xlabel(r'Estimate of $\tau$')

    #[local comment] If run locally, should comment the following 3 lines, due to wrong path.
```

```
    plt.savefig("/home/chen/4info/output/limited_seed"+ str(jobseed) +"/point_t_"+ str(t-1) +".pdf",
                bbox_inches="tight")
    plt.close()

    # posterior distribution (resample with weights)
    fig2, axs2 = plt.subplots(3, nit, figsize = (nit*3.25,9), constrained_layout = True)
    fig2.suptitle(r"The estimated posteriors at $t=$" + str(t-1)
                  + r' (Resampled with weights. $ESS=$' + str(np.round(ESS(all_ws[t-1]),0))
                  +', perturbed '+str(move_count)+' times.)', fontsize=20)
    for i in range(nit):
        sns.kdeplot(resample_lai[:,0], linewidth = 2, linestyle='-', fill= True, ax=axs2[0,i],
                    color = colors[i], alpha = 0.4) #!!! ai constrain
        #sns.kdeplot(resample_lai[:,i], linewidth = 2, linestyle='-', fill= True, ax=axs2[0,i],
                    #color = colors[i], alpha = 0.4) #!!! ai constrain
        axs2[0,i].axvline(np.log(ai[i]),0, N ,color = 'red', linewidth = 2)
        #axs2[0,i].axvline(wmedian[i],0, N ,color = colors[i], linewidth = 2, linestyle = '--') #!!! ai
constrain
        axs2[0,i].axvline(wmedian,0, N ,color = colors[i], linewidth = 2, linestyle = '--') #!!! ai
constrain
        axs2[0,i].axvline(np.log(loc_ai[i]),0, N ,color = 'grey', linewidth = 1, linestyle = 'dashdot')
        axs2[0,i].set_title("Item " + str(i+1), fontsize=10)

        sns.kdeplot(resample_lvi[:,i], linewidth = 2, linestyle='-', fill= True, ax=axs2[1,i],
                    color = colors[i], alpha = 0.4)
        axs2[1,i].axvline(np.log(vi[i]),0, N ,color = 'red', linewidth = 2)
        axs2[1,i].axvline(wmedian2[i],0, N ,color = colors[i], linewidth = 2, linestyle = '--')
        axs2[1,i].axvline(np.log(loc_vi[i]),0, N ,color = 'grey', linewidth = 1, linestyle = 'dashdot')

        sns.kdeplot(resample_ter[:,i], linewidth = 2, linestyle='-', fill= True, ax=axs2[2,i],
```

```
                color = colors[i], alpha = 0.4)
    axs2[2,i].axvline(ter[i],0, N,color = 'red', linewidth = 2)
    axs2[2,i].axvline(loc_ter[i],0, N,color = 'grey', linewidth = 1, linestyle = 'dashdot')
    axs2[2,i].axvline(wmedian3[i],0, N ,color = colors[i], linewidth = 2, linestyle = '--')

    if i == 0:
        axs2[0,i].set_ylabel(r'Density for $log\alpha$', fontsize = 16)
        axs2[1,i].set_ylabel(r'Density for $log\beta$', fontsize = 16)
        axs2[2,i].set_ylabel(r'Density for $\tau$', fontsize = 16)
    else:
        axs2[0,i].set_ylabel(r'$log\alpha$', fontsize = 0)
        axs2[1,i].set_ylabel(r'$log\beta$', fontsize = 0)
        axs2[2,i].set_ylabel(r'$\tau$', fontsize = 0)

#[local comment]
plt.savefig("/home/chen/4info/output/limited_seed"+ str(jobseed) +"/posterior_t_"+ str(t-1) +".pdf")
plt.close()

# posterior distribution for 2 population parameters (resample with weights)
fig3, axs3 = plt.subplots(1, 2, figsize = (6.5,3), constrained_layout = True)
fig3.suptitle(r"The estimated posteriors at $t=$" + str(t-1) + "\n"
              + r' (Resampled with weights. $ESS=$' + str(np.round(ESS(all_ws[t-1]),0))
              +', perturbed '+str(move_count)+' times.)', fontsize=12)
for i in range(2):
    sns.kdeplot(resample_lps[:,i], linewidth = 2, linestyle='-', fill= True, ax=axs3[i],
                color = colors[i], alpha = 0.4)
    axs3[i].axvline(np.log(true_ps[i]),0, N ,color = 'red', linewidth = 2)
    axs3[i].axvline(wmedian4[i],0, N ,color = colors[i], linewidth = 2, linestyle = '--')
    axs3[i].axvline(np.log(0.3),0, N ,color = 'grey', linewidth = 1, linestyle = 'dashdot')
```

```
    #[local comment]
    plt.savefig("/home/chen/4info/output/limited_seed"+ str(jobseed) +"/posterior2_t_"+ str(t-1) +".pdf")
    plt.close()

    # [HPC] Report memory usage
    process = psutil.Process(pid=os.getpid())
    mem_info = process.memory_info()
    mem_usage_gb = mem_info.rss / 1024**3
    mem_usage_gb = round(mem_usage_gb, 2)
    print(f"Memory usage: {mem_usage_gb} GB")

t_end = datetime.now()
t_end.strftime("%D, %H:%M:%S")


#%% Save information
### Settings
run_time = (t_end-t_start)
print("-----------------------------------------------------")
print("The total run time for later iterations is", run_time)
setting = '-------------------- General Settings --------------------\nShrinkage rate = ' + str(alpha) \
    + '.\nESS_min = ' + str(N_final) \
        + '.\nr = ' + str(sufficient_move_cor) \
            + '.\nProposal covariance = ' + str(cov_coef) + ' * cov(alive neighbors)' \
                + '.\nRun time is ' + str(run_time.days) +' days and ' \
                + str(np.round(run_time.seconds/3600,1)) + ' hours on ' + str(ncores) + ' of HPC.' \
                    + '\nepsilon_T = ' + str(e_final) \
                        + '.\nT = ' + str(t-1) \
```

```
                            + '\n----------------------------------------------------'
print(setting)
settings = {'run_time': run_time,
            's': alpha,
            'ESS_min': N_final,
            'r': sufficient_move_cor,
            'near_percentage': near_percentage,
            'cov_coef': cov_coef,
            'setting': setting}

### Observed
# Set 1 parameters
obs = {'ter':ter,
       'ai': ai,
       'vi': vi,
       'ap': ap,
       'vp': vp,
       'Y': Y,
       'RTY': RTY,
       'emp_maxt': emp_maxt}

### Output
out = {'t': t,
       'lai_t': array(lai_t),
       'lvi_t': array(lvi_t),
       'ter_t': array(ter_t),
       'lps_t': array(lps_t),
       'ws_t': ws_t,
       'all_e': all_e,
```

```
        'all_lais': all_lais,
        'all_lvis': all_lvis,
        'all_ters': all_ters,
        'all_lps': all_lps,
        'all_ws': all_ws,
        'all_movement': all_movement,
        'all_moves': all_moves,
        'e_final': e_final,
        'xs_t': array(xs_t),
        'rt_t': array(rt_t)
        }


#%% Save data
fnm = "/home/chen/4info/output/limited_seed"+ str(jobseed) +"/" + 'out_seed_' + str(jobseed) + '.pkl'

final_out = [settings, obs, out] # objects
save_object(obj = final_out, filename = fnm)  # save

print('================ Python codes finished ====================')
```

References

Andrieu, C., & Thoms, J. (2008). A tutorial on adaptive MCMC. *Statistics and computing*, *18*(4), 343-373.

Beaumont, M. A. (2010). Approximate Bayesian computation in evolution and ecology. *Annual review of ecology, evolution, and systematics*, 379-406.

Beaumont, M. A., Cornuet, J. M., Marin, J. M., & Robert, C. P. (2009). Adaptive approximate Bayesian computation. *Biometrika*, *96*(4), 983-990.

Beaumont, M. A., Zhang, W., & Balding, D. J. (2002). Approximate Bayesian computation in population genetics. *Genetics*, *162*(4), 2025-2035.

Boehm, U., Marsman, M., van der Maas, H. L., & Maris, G. (2021). An Attention-Based Diffusion Model for Psychometric Analyses. *psychometrika*, *86*(4), 938-972.

Broderick, T., Wong-Lin, K. F., & Holmes, P. (2009). Closed-form approximations of first-passage distributions for a stochastic decision-making model. *Applied Mathematics Research eXpress*, *2009*(2), 123-141.

Buchholz, A., Chopin, N., & Jacob, P. E. (2021). Adaptive tuning of hamiltonian monte carlo within sequential monte carlo. *Bayesian Analysis*, *16*(3), 745-771.

Cox, D.R. & Miller, H.D. (1970). *The theory of stochastic processes*, London: Methuen.

Choe, E. M., Kern, J. L., & Chang, H. H. (2018). Optimizing the use of response times for item selection in computerized adaptive testing. *Journal of Educational and Behavioral Statistics, 43*(2), 135-158.

Chopin, N., & Papaspiliopoulos, O. (2020). *An introduction to sequential Monte Carlo*. New York: Springer International Publishing.

De Boeck, P., & Jeon, M. (2019). An overview of models for response times and processes in cognitive tests. *Frontiers in psychology*, *10*, 102.

Del Moral, P., Doucet, A., & Jasra, A. (2006). Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *68*(3), 411-436.

Del Moral, P., Doucet, A., & Jasra, A. (2012). An adaptive sequential Monte Carlo method for approximate Bayesian computation. *Statistics and computing*, *22*(5), 1009-1020.

Feller, W. (1968). *An introduction to probability theory and its applications*, volume 1. John Wiley: New York.

Jordan, P. (2020). The counterintuitive impact of responses and response times on parameter estimates in the drift diffusion model. *British Journal of Mathematical and Statistical Psychology*, *73*(2), 289-315.

Liu, J. S., & Liu, J. S. (2001). *Monte Carlo strategies in scientific computing* (Vol. 10, pp. 978-0). New York: springer.

Marin, J. M., Pudlo, P., Robert, C. P., & Ryder, R. J. (2012). Approximate Bayesian computational methods. *Statistics and Computing*, *22*(6), 1167-1180.

Molenaar, D., Tuerlinckx, F., & van der Maas, H. L. (2015a). A bivariate generalized linear item response theory modeling framework to the analysis of responses and response times. *Multivariate Behavioral Research*, *50*(1), 56-74.

Molenaar, D., Tuerlinckx, F., & van der Maas, H. L. J. (2015b). Fitting diffusion item response theory models for responses and response times using the r package diffirt. *Journal of Statistical Software, 66*(4), 1–34. doi: 10.18637/jss.v066.i04

Montzka, C., Pauwels, V. R., Franssen, H. J. H., Han, X., & Vereecken, H. (2012). Multivariate and multiscale data assimilation in terrestrial systems: A review. *Sensors*, *12*(12), 16291-16333.

Navarro, D. J., & Fuss, I. G. (2009). Fast and accurate calculations for first-passage times in Wiener diffusion models. *Journal of mathematical psychology*, *53*(4), 222-230.

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., & Feldman, M. W. (1999). Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular biology and evolution*, *16*(12), 1791-1798.

Ranger, J., & Kuhn, J.-T. (2018). Estimating diffusion-based item response theory models: Exploring the robustness of three old and two new estimators. *Journal of Educational and Behavioral Statistics, 43*(6), 635-662. doi: 10.3102/1076998618787791

Ranger, J., Kuhn, J.-T., & Szardenings, C. (2016). Limited information estimation of the diffusion-based item response theory model for responses and response times. *British Journal of Mathematical and Statistical Psychology, 69*(2), 122-138. doi: 10.1111/bmsp.12064

Ranger, J., Kuhn, J.-T., & Szardenings, C. (2020). Minimum distance estimation of multidimensional diffusion-based item response theory models. *Multivariate Behavioral Research, 55*(6), 941-956. doi: 10.1080/00273171.2019.1704676

Ratcliff (1978). A theory of memory retrieval. *Psychological Review*, *85* (2), 59–108

Ratcliff, R. (2013). Parameter variability and distributional assumptions in the diffusion model. *Psychological review*, *120*(1), 281.

Ratcliff, R., Smith, P. L., Brown, S. D., & McKoon, G. (2016). Diffusion decision model: Current issues and history. *Trends in cognitive sciences*, *20*(4), 260-281.

Ratcliff, R., & Tuerlinckx, F. (2002). Estimating parameters of the diffusion model: Approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic bulletin & review*, *9*(3), 438-481.

Rice, J. A. (2006). *Mathematical statistics and data analysis*. Cengage Learning.

Robert, C. P., Casella, G., & Casella, G. (1999). *Monte Carlo statistical methods* (Vol. 2). New York: Springer.

Rouder, J. N., Province, J. M., Morey, R. D., Gomez, P., & Heathcote, A. (2015). The lognormal race: A cognitive-process model of choice and latency with desirable psychometric properties. *Psychometrika*, *80*(2), 491-513.

Taylor, S. J. (1994). Modeling stochastic volatility: A review and comparative study. *Mathematical finance*, *4*(2), 183-204.

Tillman, G., Van Zandt, T., & Logan, G. D. (2020). Sequential sampling models without random between-trial variability: The racing diffusion model of speeded decision making. *Psychonomic Bulletin & Review*, *27*, 911-936.

Tuerlinckx, F., & De Boeck, P. (2005). Two interpretations of the discrimination parameter. *Psychometrika, 70*(4), 629-650.

Tuerlinckx, F., Molenaar, D., & van der Maas, H. L. J. (2016). Diffusion-based response time models. In W. J. van der Linden (Ed.), *Handbook of item response theory* (p. 283-300). Chapman and Hall/CRC.

Turner, B. M., & Van Zandt, T. (2012). A tutorial on approximate Bayesian computation. *Journal of Mathematical Psychology, 56*(2), 69-85.

University of Maryland (2022). The Zaratan HPC cluster. https://hpcc.umd.edu/hpcc/zaratan.html

van der Linden, W. J. (2007). A hierarchical framework for modeling speed and accuracy on test items. *Psychometrika*, *72*(3), 287-308.

van der Linden, W. J., Klein Entink, R. H., & Fox, J. P. (2010). IRT parameter estimation with response times as collateral information. *Applied Psychological Measurement*, *34*(5), 327-346.

van der Maas, H. L., Molenaar, D., Maris, G., Kievit, R. A., & Borsboom, D. (2011). Cognitive psychology meets psychometric theory: on the relation between process models for decision making and latent variable models for individual differences. *Psychological review*, *118*(2), 339.

van Ravenzwaaij, D., Donkin, C., & Vandekerckhove, J. (2017). The EZ diffusion model provides a powerful test of simple empirical effects. *Psychonomic bulletin & review*, *24*(2), 547-556.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

Van Zandt, T., Colonius, H., & Proctor, R. W. (2000). A comparison of two response time models applied to perceptual matching. *Psychonomic bulletin & review, 7*(2), 208-256.

Wagenmakers, E. J., Grasman, R. P., & Molenaar, P. C. (2005). On the relation between the mean and the variance of a diffusion model response time distribution. *Journal of Mathematical Psychology*, *49*(3), 195-204.

Wagenmakers, E. J., Van Der Maas, H. L., & Grasman, R. P. (2007). An EZ-diffusion model for response time and accuracy. *Psychonomic bulletin & review, 14*(1), 3-22.

Wang, C., & Xu, G. (2015). A mixture hierarchical model for response times and response accuracy. *British Journal of Mathematical and Statistical Psychology, 68*(3), 456-477.

Yuan, K. H., & Bentler, P. M. (1998). Structural equation modeling with robust covariances. *Sociological methodology, 28*(1), 363-396.