




Birth-and-Death Processes in Python: The BirDePy Package

Sophie Hautphenne 
The University of Melbourne

Brendan Patch 
The University of Melbourne

Abstract

Birth-and-death processes (BDPs) form a class of continuous-time Markov chains that are particularly suited to describing the changes in the size of a population over time. Population-size-dependent BDPs (PSDBDPs) allow the rate at which a population grows to depend on the current population size. The main purpose of our new Python package **BirDePy** is to provide easy-to-use functions that allow the parameters of discretely-observed PSDBDPs to be estimated. The package can also be used to estimate parameters of continuously-observed PSDBDPs, simulate sample paths, approximate transition probabilities, and generate forecasts. We describe in detail several methods which have been incorporated into **BirDePy** to achieve each of these tasks. The usage and effectiveness of the package is demonstrated through a variety of examples of PSDBDPs, as well as case studies involving annual population count data of two endangered bird species.

Keywords: ABC algorithm, birth-and-death processes, continuous-time Markov chains, diffusion approximation, EM algorithm, Erlangization, Laplace transform, maximum likelihood estimation, parameter estimation, Python, uniformization.

1. Introduction

Birth-and-death processes (BDPs) are continuous-time Markov chains that track the stochastically evolving number of individuals in a population over time (Feller 1957; Grimmett and Stirzaker 2020). These processes are effective modeling tools for areas such as biology, ecology, genetics, and epidemiology (Allen 2008; Novozhilov, Karev, and Koonin 2006; Karlin and Taylor 1975; Nåsell 2002), as well as operations research (Brockmeyer, Halstrom, and Jensen 1948; Boucherie and Van Dijk 2010; Gibbens, Hunt, and Kelly 1990). The state of a BDP transitions up by one unit when an individual is added to the population (e.g., due to a “birth” or “arrival”) and transitions down by one unit when an individual is removed from the

population (e.g., due to a “death” or “departure”). Population-size-dependent BDPs (PSDBDPs) allow realistic stochastic population dynamics to be modeled by letting the transition rates at any given time to depend upon the population size at that time (in a non-linear way). A recent review on PSDBDPs is provided by Crawford, Ho, and Suchard (2018).

This paper introduces a Python (see Van Rossum and Drake Jr. 1995) package, **BirDePy** (see Hautphenne and Patch 2023), which is primarily intended to make a variety of parameter estimation tools for discretely-observed PSDBDPs available to biologists and other practitioners. **BirDePy** is the first software package to contain a comprehensive set of methods for performing this estimation task for PSDBDPs. Approaching the estimation problem from multiple angles is important since, in general, different circumstances call for different methods; for example, diffusion-approximation methods may only be appropriate when population sizes are large, and matrix-exponential methods may only be possible when population sizes are small. In addition to targeting this estimation problem, **BirDePy** provides a complete set of tools for the computational analysis of PSDBDPs: it can also perform parameter estimation for continuously-observed PSDBDPs, simulate discretely- and continuously-observed sample paths, compute approximations to transition probabilities, and generate forecasts of future expected behavior.

Effective and accessible tools for estimating the parameters of PSDBDPs are crucial for the successful application of these processes. For continuously-observed processes in which the birth and death rates are linear functions of the unknown parameters, closed form estimators are available (Wolff 1965; Reynolds 1973). However, in practice observations are usually only available at discrete (often unequally-spaced) times. This makes the estimation task much more difficult since it is not at first sight clear how to account for the path that the process takes between observations. Furthermore, even if continuous observation is possible, despite a large and growing literature, determining exact estimators for realistic models that account for features such as logistic growth and the effect of a carrying capacity appears to be a highly non-trivial task. Tavaré (2018) and Asanjarani, Nazarathy, and Taylor (2021) provide recent surveys on parameter estimation for models related to PSDBDPs. We will soon return to parameter estimation for PSDBDPs after a short discussion on transition probabilities.

Research focused on determining the probability of transition between any pair of states after any given time in PSDBDPs provides a foundation for parameter estimation, as these transition probabilities are used to compute likelihood functions. In often disparate work, researchers have approached the problem of determining transition probabilities using numerical methods. For example, it is well known that for models with a finite or truncated state space, the matrix exponential method can be used for this purpose (Feller 1957). Other key advances in this area utilize Laplace transforms and continued fractions (Murphy and O’Donohoe 1975; Crawford and Suchard 2012), numerical approximations such as uniformization (Van Dijk, Van Brummelen, and Boucherie 2018; Grassmann 1977), Erlangization (Asmussen, Avram, and Usabel 2002; Mandjes and Taylor 2016), and diffusion approximations (Kurtz 1971; Whitt 2002). Furthermore, research on determining transition probabilities has been explicitly connected to maximum-likelihood-estimates (MLEs) for PSDBDPs (Ross, Taimre, and Pollett 2006, 2007; Ross, Pagendam, and Pollett 2009; Buckingham-Jeffery, Isham, and House 2018).

The expectation-maximization (EM) algorithm of Dempster, Laird, and Rubin (1977) is an approach to obtaining MLEs in the presence of unobserved data. Crawford, Minin, and Suchard (2014) apply the EM algorithm by treating the (unobserved) evolution of a PSDBDP between discrete observation points as missing data. There are several ways to compute the

expected value of the unobserved data, conditional on the observed data: Crawford *et al.* (2014) use a Laplace transform approach, but it can also be done using numerical integration, or by evaluating appropriate matrix exponentials. **DOBAD** is an R (see R Core Team 2024) package related to this work written by Doss, Minin, and Suchard (2017). Recently, Roney, Ferlic, Michor, and McDonald (2020) released the R-wrapped C++ (see Stroustrup 2000) package **EstiPop** for simulation and parameter estimation based on diffusion-approximation, of a class of models that includes PSDBDPs. **GillesPy2** is a Python package that can be used to perform simulation of PSDBDPs; it is used as a backend for **StochSS**, which is a software-as-a-service platform with a graphical user interface (Drawert *et al.* 2016). Combined with the Python package **sciope** (Singh, Wrede, and Hellander 2021), **StochSS** can be used to carry out approximate Bayesian computation (ABC) to estimate parameters for PSDBDPs. King, Nguyen, and Ionides (2016) developed **pomp**, which is an R package for working with partially observed Markov processes (POMPs); depending on the question and data at hand, POMPs and PSDBDPs can be used to address related research questions.

As will be elaborated on later in the paper, **BirDePy** includes the EM algorithm, methods based on diffusion-approximation, and ABC parameter estimation methods akin to those in these other packages. It also incorporates all of the methods for finding transition probabilities for PSDBDPs mentioned earlier, it can use these to obtain MLEs, and it incorporates least-squares based estimation procedures. In addition to the Laplace transform based EM algorithm of Crawford *et al.* (2014), **BirDePy** includes EM algorithm implementations that use numerical and matrix exponential based procedures for computing the required expected values. Additionally, **BirDePy** includes options to accelerate the EM algorithm (Jamshidian and Jennrich 1997). As well as being able to determine transition probabilities and perform parameter estimation, our package has several simulation algorithms built into it. The package can perform exact simulation (Kendall 1950; Gillespie 1977), tau-leaping (Gillespie 2001; Anderson, Ganguly, and Kurtz 2011), and a recently introduced piecewise approximation algorithm (Hautphenne and Patch 2021). Another key strength of our package is that it includes a variety of built-in PSDBDP models, making it extremely easy to use. Due to this, all of the methods mentioned thus far can be called using a single line of code (after **BirDePy** has been imported). Finally, our package is well documented online (Hautphenne and Patch 2023), further increasing its accessibility.

The remainder of this paper is organized as follows. Section 2 formally introduces PSDBDPs along with the simulation, estimation, and forecasting methods used by **BirDePy**. Section 3 demonstrates the capability of the package through a range of numerical examples, and Section 4 applies the package to two topical examples involving endangered bird species. In the final section we discuss future possible enhancements to the package. The description and usage of **BirDePy** are given in Appendix A. Software engineering aspects and a high-level diagram summarizing the codebase of the package are presented in Appendix B.

2. Birth-and-death processes

In this section, we briefly summarize the theory underpinning the functionality of **BirDePy**. In Section 2.1, we describe birth-and-death processes. In Section 2.2, we provide simulation algorithms. In Section 2.3, we summarize methods for computing transition probabilities; and in Section 2.4, we focus on parameter estimation frameworks.

Model label	$\lambda_z(\boldsymbol{\theta})$	$\mu_z(\boldsymbol{\theta})$	$\boldsymbol{\theta}$
"linear"	γz	νz	γ, ν
"linear-migration"	$\gamma z + \alpha$	νz	γ, ν, α
"pure-birth"	γz	0	γ
"pure-death"	0	νz	ν
"Poisson"	γ	0	γ
"Verhulst"	$\gamma(1 - \alpha z)z$	$\nu(1 + \beta z)z$	$\gamma, \nu, \alpha, \beta$
"Ricker"	$\gamma z \exp(-(\alpha z)^c)$	νz	γ, ν, α, c
"Hassell"	$\frac{\gamma z}{(1 + \alpha z)^c}$	νz	γ, ν, α, c
"MS-S"	$\frac{\gamma z}{1 + (\alpha z)^c}$	νz	γ, ν, α, c
"Moran"	$\frac{N-z}{N} \left(\frac{\alpha z(1-u) + \beta(N-z)v}{N} \right)$	$\frac{z}{N} \left(\frac{\beta(N-z)(1-v) + \alpha z u}{N} \right)$	α, β, u, v, N
"M/M/1"	γ	$\nu 1_{\{z > 0\}}$	γ, ν
"M/M/inf"	γ	νz	γ, ν
"loss-system"	$\gamma 1_{\{z < c\}}$	νz	γ, ν, c

Table 1: Transition rates and parameters for $z \geq 0$. Any rates which take on a negative value are modified to be equal to 0, and $1_{\{A\}}$ is the indicator of the event A . The order of the parameters in the last column is important for Appendix A.

2.1. Models

BirDePy is built for analyzing general birth-and-death processes. These continuous-time Markov chains (CTMCs) are widely used to study how the number of individuals in a population changes over time. Let $Z(t)$ denote the size of a population at time $t \geq 0$; the process $Z \equiv (Z(t), t \geq 0)$ evolves on the state space $\mathcal{S} = (0, 1, \dots, N)$, where N may be infinite. We assume that the evolution of such a process is fully described by non-negative real-valued functions $\lambda_z(\boldsymbol{\theta}) : \mathcal{S} \rightarrow \mathbb{R}_+$ and $\mu_z(\boldsymbol{\theta}) : \mathcal{S} \rightarrow \mathbb{R}_+$ of the current state z ; these functions are known as the population “birth-rate” and “death-rate”, respectively. The model has a finite number of real parameters recorded in the vector $\boldsymbol{\theta}$. When $Z(t) = z$, the time until the process transitions to another state is exponentially distributed with mean $(\lambda_z(\boldsymbol{\theta}) + \mu_z(\boldsymbol{\theta}))^{-1}$ (where we define $1/0 = \infty$). At a transition time from state z , the process jumps to state $z + 1$ with probability $\lambda_z(\boldsymbol{\theta})/(\lambda_z(\boldsymbol{\theta}) + \mu_z(\boldsymbol{\theta}))$, or to state $z - 1$ with probability $\mu_z(\boldsymbol{\theta})/(\lambda_z(\boldsymbol{\theta}) + \mu_z(\boldsymbol{\theta}))$.

As summarized in Table 1, we consider selected birth-and-death models of three types: (i) some variants of the linear BDP, (ii) some PSDBDPs, and (iii) some queueing models. We give a brief description of these models in the next paragraphs.

A linear BDP models the evolution of a population in which individuals give birth and die independently of each other and of the current population size. The birth and death rates per individual are given by some strictly positive constants γ and ν , respectively, leading to linear population birth and death rates $\lambda_z(\boldsymbol{\theta}) = \gamma z$ and $\mu_z(\boldsymbol{\theta}) = \nu z$. If, in addition, individuals immigrate at rate α into the population, the birth rate becomes $\lambda_z(\boldsymbol{\theta}) = \gamma z + \alpha$. A pure birth process, also called Yule process, is obtained by setting the death rate ν to zero, and a pure death process is obtained by setting the birth rate γ to zero. The Poisson process is a particular pure birth process in which the birth rate $\lambda_z(\boldsymbol{\theta})$, also called arrival rate, does not depend on the current population size z . Classical references discussing these models include Cinlar (2013) and Karlin and Taylor (1975).

In contrast to linear birth-and death processes, in a PSDBDP the birth and death rates per individual depend on the current population size z . This feature allows us to model competition for resources such as food or territory. The birth (respectively death) rate per individual is typically a decreasing (respectively increasing) function of z ; this leads to logistic growth of the population (S-shaped trajectories), which tend to fluctuate around a threshold value called the carrying capacity. One of the most popular logistic models is the Verhulst model, which has been reinvented several times since it first appeared in 1838 (see for instance [Nåsell 2001](#) and references therein). In this model, the birth and death rates per individual are linear functions of z . The susceptible-infectious-susceptible (SIS) epidemic model is a particular case of the Verhulst model with $\beta = 0$, implying the death (recovery) rate per individual does not depend on z . Other classical models of logistic growth include the Ricker model, in which the birth rate decreases exponentially in z , and the Hassell and MaynardSmith-Slatkin (MSS) models, in which the birth rate decreases according to a power law (here we assume that the death rate in these models is independent of the population size). Despite the apparent similarity between the Hassell and the MSS models, the MSS model has a more flexible and better descriptive form than the Hassell model according to [Bellows \(1981\)](#). An overview of several of these logistic models in the deterministic case is given by [Jemmer and McNamee \(2005\)](#). The particular case where $c = 1$, called the Beverton-Holt (BH) model, is also widely used, in particular as a model for exploited fish populations ([Subbey, Devine, Schaarschmidt, and Nash 2014](#)). Finally, the Moran process models the change in the numbers of particles of two types in a finite population of size N , where transitions between types occur at a rate proportional to the number of potential contacts between members of each type in a population; this is an important model for genetic evolution, see for example ([Nowak 2006](#), Chapter 6).

In classical queueing models (see [Kleinrock 1975](#); [Karlin and Taylor 1981](#)), the population birth rate $\lambda_z(\boldsymbol{\theta})$ does not depend on z : births (arrivals) occur according to a point process, and once in the population (system), individuals queue to be served by one or multiple servers. The M/M/1 and M/M/ ∞ queues have arrivals occurring according to a Poisson process and, respectively, a single server and an infinite number of servers (acting in parallel and independently). In a loss system, arrivals can only occur when the population is below a certain threshold value c .

In the remainder of this paper, when the dependence on $\boldsymbol{\theta}$ is clear from the context, we simply write $\lambda_z(\boldsymbol{\theta})$ and $\mu_z(\boldsymbol{\theta})$ as λ_z and μ_z .

2.2. Simulation

A few exceptions aside, explicit expressions for quantities of interest such as the probability distribution, or the expected value, of the population size of a PSDBDP at a given time, do not exist. By providing many examples of possible outcomes, Monte Carlo simulation can allow interesting questions related to these processes to be examined (e.g., [Karev, Wolf, and Koonin 2003](#); [Karev, Wolf, Berezovskaya, and Koonin 2004](#)). Simulation can also be used to compare the performance of other tools, such as those used for parameter estimation, by providing synthetic data for the tools to be tested on (e.g., [Davison, Hautphenne, and Kraus 2021](#); [Mandjes and Sollie 2021](#)). Last, but not least, some parameter estimation techniques directly rely on simulated sample paths to generate estimates (e.g., [Singh et al. 2021](#)).

Simulation of a birth-and-death process can be continuous or discrete. The goal of continuous

simulation is to obtain a sequence of times at which the process transitions, together with the corresponding sequence of states the process transitions into. In a continuous simulation, the output provides sufficient information to recover the state of the process at any time within the simulation horizon. This is desirable since the output contains enough information for a user to know whether certain events have occurred, for example whether the process has crossed a certain boundary by a certain time. In contrast, the goal of discrete simulation is to obtain samples of a birth-and-death process at pre-specified observation times. In this case, the state of the process at those observation times is the output. An advantage of discrete simulation is that the same observation times can be specified for each sample path so that they can be combined together to find, for example, the expected value or variance of a process as a function of time.

Given an initial population size z_0 , to perform continuous simulation of a birth-and-death process over the interval $[0, t]$, initialize $j := 0$, $t_0 := 0$, and then repeat these steps:

1. Generate an outcome Δ of an exponentially distributed random variable with mean $(\lambda_{z_j} + \mu_{z_j})^{-1}$.
2. If $t_j + \Delta \leq t$, set $t_{j+1} := t_j + \Delta$; otherwise stop.
3. With probability $\lambda_{z_j}/(\lambda_{z_j} + \mu_{z_j})$ set $z_{j+1} := z_j + 1$; otherwise set $z_{j+1} := z_j - 1$.
4. Set $j := j + 1$.

Discrete simulation proceeds in a similar fashion. Given an initial population size z_0 , to simulate a birth-and-death process at the time points $t_0 < t_1 < \dots < t_n$, initialize $j := 1$, $s := t_0 + \Delta$ where Δ is an outcome of an exponentially distributed random variable with mean $(\lambda_z + \mu_z)^{-1}$, and $z := z_0$, and then repeat these steps:

1. While $s \leq t_j$:
 - (i) With probability $\lambda_z/(\lambda_z + \mu_z)$ set $z := z + 1$; otherwise set $z := z - 1$.
 - (ii) Set $s := s + \Delta$ where Δ is an outcome of an exponentially distributed random variable with mean $(\lambda_z + \mu_z)^{-1}$.
2. Set $z_j := z$.
3. If $j < n$ set $j := j + 1$; otherwise stop.

Observe that t_j has a different interpretation depending on whether continuous or discrete simulation is being performed. It is thought within the probability community that this exact approach to simulation of CTMCs was first considered by Joseph L. Doob and his collaborators around 1945. To our knowledge, the first implementation on a computer is mentioned in [Kendall \(1950\)](#), and the algorithm was then popularized more widely in [Gillespie \(1977\)](#).

A well-known drawback of exact simulation algorithms is that they may take considerable computational time to produce sample paths. An alternative approximate simulation method for population processes known as tau-leaping, introduced by [Gillespie \(2001\)](#), allows the user to trade accuracy for speed. This algorithm partitions time into intervals of predetermined constant length τ . Conditional on $Z(0) = z$ at the start of an interval, the state of $Z(\tau)$ is approximated by the difference of two Poisson distributed random variables \mathfrak{L} and \mathfrak{M} which

are respectively intended to approximate the total number of births and deaths that occur within the interval. Given an initial population size z_0 , the basic Euler form of the algorithm can be used to perform discrete simulation by initializing $j := 1$, $s := t_0 + \tau$, and $z := z_0$, and then repeating these steps:

1. While $s \leq t_j$:
 - (i) Set $z := z + \mathfrak{L} - \mathfrak{M}$ where \mathfrak{L} and \mathfrak{M} are outcomes of Poisson distributed random variables with respective means $\lambda_z \tau$ and $\mu_z \tau$.
 - (ii) Set $s := s + \tau$.
2. Set $z_j := z$.
3. If $j < n$ set $j := j + 1$; otherwise stop.

Several variations of this algorithm have been developed which focus primarily on step-size selection (e.g., Gillespie and Petzold 2003; Cao, Gillespie, and Petzold 2006) and on ensuring that the population size does not go negative during the simulation (e.g., Cao, Gillespie, and Petzold 2005; Chatterjee, Vlachos, and Katsoulakis 2005; Anderson 2008).

In Anderson *et al.* (2011) a midpoint variant of the basic algorithm is developed and elegantly analyzed. The algorithm is the same as the Euler version described above with the exception that the two Poisson distributed random variables \mathfrak{L} and \mathfrak{M} have respective means $\lambda_{z+\rho(z)}\tau$ and $\mu_{z+\rho(z)}\tau$, where $\rho(z) = \frac{1}{2}\tau(\lambda_z - \mu_z)$ approximates $\mathbb{E}[Z(\frac{1}{2}\tau) - Z(0) \mid Z(0) = z]$. The basic idea here is that determining the number of births and deaths using an approximation to the population size at the midpoint of an interval may increase accuracy relative to using the population size at the beginning of the interval. This algorithm can provide substantial improvements in accuracy relative to the Euler version at only a minor increase in computational cost. Both of the above algorithms can be thought of as generating piecewise approximations to a PSDBDP, where zero-order approximations of the birth and death rates are utilized within each subinterval.

Recently a new approach to perform discrete simulation of a general birth-and-death processes utilizing linear (first-order) approximations of the birth and death rates within each subinterval was proposed in Hautphenne and Patch (2021). This algorithm approximates any birth-and-death process by ‘piecewise-linear’ birth-and-death processes. It also uses the fact that a discretely-observed linear birth-and-death process corresponds to a linear fractional Galton–Watson process (see Harris 1963). This means that for a linear birth-and-death process \dot{Z} with birth rates $\lambda_z = \lambda z$ and death rates $\mu_z = \mu z$, the probability that a family generated by a single individual at time 0 ‘becomes extinct’ before time τ is given by

$$\beta_1(\tau) = \begin{cases} \mu\{\exp((\lambda - \mu)\tau) - 1\} / \{\lambda \exp((\lambda - \mu)\tau) - \mu\} & \text{if } \lambda \neq \mu, \\ \lambda\tau / (1 + \lambda\tau) & \text{if } \lambda = \mu. \end{cases} \quad (1)$$

Each family which survives results in H individuals being present at time τ , where $\mathbb{P}(H = h) = (1 - \beta_2(\tau))\beta_2(\tau)^{h-1}$ with

$$\beta_2(\tau) = \begin{cases} \lambda\beta_1(\tau)/\mu & \text{if } \lambda \neq \mu, \\ \beta_1(\tau) & \text{if } \lambda = \mu. \end{cases} \quad (2)$$

Therefore, given $\mathring{Z}(0) = z$, a realization of $\mathring{Z}(\tau)$ can be obtained by generating an outcome of the random variable \mathfrak{B} which is binomially distributed with z trials and success probability $1 - \beta_1(\tau)$ given by (1), and then using the fact that $\mathring{Z}(\tau)$ follows a negative binomial distribution with parameters \mathfrak{B} and $1 - \beta_2(\tau)$, that is,

$$\mathbb{P}(\mathring{Z}(t_j) = k \mid \mathfrak{B}) = \binom{k + \mathfrak{B} - 1}{k} \beta_2(\tau)^{\mathfrak{B}} (1 - \beta_2(\tau))^k.$$

So, if Z is a PSDBDP, conditional on $Z(0)$, $Z(\tau)$ can be approximated by $\mathring{Z}(\tau)$ by setting the per-individual rates of the approximating linear process \mathring{Z} equal to those of Z at time 0, that is, $\lambda = \lambda_{Z(0)}/Z(0)$ and $\mu = \mu_{Z(0)}/Z(0)$. Therefore, to obtain an approximate simulation of $(Z(t_j), j = 0, 1, \dots, n)$, initialize $j := 1$, $s := t_0 + \tau$, and $z := z_0$, and repeat these steps:

1. While $s \leq t_j$:
 - (i) Set $\lambda := \lambda_z/z$ and $\mu := \mu_z/z$.
 - (ii) Generate a binomially distributed random variable \mathfrak{B} with success probability $1 - \beta_1(\tau)$ with $\beta_1(\tau)$ as given by (1) and number of trials z .
 - (iii) Generate a negatively binomially distributed random variable \mathfrak{C} with success probability $1 - \beta_2(\tau)$ with $\beta_2(\tau)$ as given by (2) and number of trials \mathfrak{B} .
 - (iv) Set $z := z + \mathfrak{C}$ and $s := s + \tau$.
2. Set $z_j := z$.
3. If $j < n$ set $j := j + 1$; otherwise stop.

This algorithm can be implemented efficiently and is highly accurate. It also explicitly avoids the possibility of the population size becoming negative during the simulation.

Table 2 summarizes the discrete simulation methods described in this section, and gives the label used to call them in the **BirDePy** function `birdepy.simulate.discrete()` as described in Appendix A.

2.3. Transition probabilities

Many of the general models described in Section 2.1 have no explicit expression for their transition probabilities $p_{i,j}(t) := \mathbb{P}(Z(t) = j \mid Z(0) = i)$, $i, j \in \mathcal{S}$, $t \geq 0$. However, it is often desirable to compute these transition probabilities since they allow for a deeper understanding of the future (random) evolution of the process. Practically speaking, this may underpin some performance analysis of a system which is being modelled by the process. Additionally, and of most interest in this paper, transition probabilities allow likelihood functions of discretely-observed sample paths to be evaluated, which opens up the possibility of parameter estimation (as discussed in the next section). In this section we outline nine methods of approximating $p_{i,j}(t)$. Three of these methods are matrix-based and rely on state space truncation, another method uses Laplace transforms, and the remaining four methods use approximation models.

Method (in order of appearance)	Label	Brief description
Exact	"exact"	Utilizes all jumps of the process.
Euler approximation	"ea"	Population changes between τ -sized intervals governed by Poisson random variables with parameters depending on population sizes at beginning of intervals.
Midpoint approximation	"ma"	Population changes between τ -sized intervals governed by Poisson random variables with parameters depending on estimate of population sizes at midpoints of intervals.
Galton–Watson approximation	"gwa"	Population changes between τ -sized intervals governed by linear birth-and-death processes with parameters depending on population sizes at beginning of intervals.

Table 2: Methods for simulating sample paths of general birth-and-death processes.

Matrix exponential

The Kolmogorov forward equation (e.g., [Grimmett and Stirzaker 2020](#), Section 6.9) for CTMCs provides the foundational property that the collection of transition probability functions $p_{i,j}(t)$ satisfy, for $t \geq 0$, the system of first order ordinary differential equations

$$\begin{aligned} \frac{dp_{i,0}(t)}{dt} &= \mu_1 p_{i,1}(t) - \lambda_0 p_{i,0}(t), \\ \frac{dp_{i,j}(t)}{dt} &= \lambda_{j-1} p_{i,j-1}(t) + \mu_{j+1} p_{i,j+1}(t) - (\lambda_j + \mu_j) p_{i,j}(t), \quad j \geq 1, \end{aligned} \quad (3)$$

with $p_{i,i}(0) = 1$ and $p_{i,j}(0) = 0$ for $j \neq i$. Let Q be the generator of Z , that is, the square matrix with diagonal entries $q_{i,i} = -(\lambda_i + \mu_i)$, upper diagonal entries $q_{i,i+1} = \lambda_i$, and lower diagonal entries $q_{i-1,i} = \mu_i$ (and zeros elsewhere). Also collect the transition probabilities into a matrix $P(t) = (p_{i,j}(t); i, j \in \mathcal{S})$. Then (3) can be written in matrix form as $\frac{d}{dt}P(t) = P(t)Q$. When Q is finite it immediately follows that

$$P(t) = \lim_{k \rightarrow \infty} \sum_{n=0}^k \frac{1}{n!} (Qt)^n =: \exp(Qt). \quad (4)$$

Matrix exponentials are ubiquitous and their efficient computation has received a great deal of attention over the years; see [Moler and Van Loan \(2003\)](#) for a review, and [Al-Mohy and Higham \(2010\)](#) for a method in popular usage. In particular, efficient methods for determining an appropriate truncation of the infinite series in (4) and approximating the remaining tail sum are available. Despite this attention, there have been doubts about the effectiveness of this approach when Q becomes large or nearly singular ([Ross et al. 2006](#); [Crawford and Suchard 2012](#)). In addition, to use this method to compute transition probabilities, the maximum possible population size must be bounded ($|\mathcal{S}| < \infty$). For models where this does not naturally occur, Q can be truncated to obtain approximate results. A modern perspective on the utility of the matrix exponential for computing transition probabilities of finite-state CTMCs can be found in [Sherlock \(2021\)](#).

Uniformization

Due to the special nature of the generator Q (specifically that it has non-negative off-diagonal entries and row-sums equal to 0), an alternative procedure known as uniformization, introduced by Jensen (1953), is available for computing $P(t)$. Consider a discrete-time Markov chain (DTMC) with probability transition matrix $A := Q/a + I$, where $a := \max_z |\lambda_z + \mu_z|$ when this exists (for example when $|\mathcal{S}| < \infty$), and where I denotes the identity matrix of appropriate size. Suppose that this DTMC transitions at the event times of a Poisson process $(N(t), t \geq 0)$ with rate a . Using this construction, we can show that

$$\mathbb{P}(Z(t) = j \mid Z(0) = i, N(t) = n) = (A^n)_{i,j}.$$

Therefore, conditioning on $N(t)$ provides

$$P(t) = \lim_{k \rightarrow \infty} \sum_{n=0}^k \frac{(ta)^n e^{-ta}}{n!} A^n. \quad (5)$$

This procedure is discussed in detail in Grassmann (1977); Gross and Miller (1984); Melamed and Yadin (1984), and a recent review of the method is provided in Van Dijk *et al.* (2018).

In addition to potentially requiring truncation of the state space for many models of interest, another source of approximation error for this method is that a finite k must be chosen in (5).

Erlangization

Using probabilistic arguments, Erlangization is yet another matrix-based method for computing $P(t)$. Let T be an exponentially distributed random variable with mean $\eta^{-1} > 0$. Define $r_{i,j}^{(\eta)} := \mathbb{P}(Z(T) = j \mid Z(0) = i)$ and collect these quantities into $R(\eta) = (r_{i,j}^{(\eta)}, i, j \in \mathcal{S})$. The matrix $R(\eta)$ can be viewed as the one-step transition probability matrix of a DTMC embedded in Z at the epochs of a Poisson process with rate η . By conditioning on the first transition of Z and the expiry of the time T , we obtain the recursive expressions

$$r_{i,j}^{(\eta)} = \frac{\mu_i r_{i-1,j}^{(\eta)} + \lambda_i r_{i+1,j}^{(\eta)} + \eta 1_{\{i=j\}}}{\lambda_i + \mu_i + \eta}, \quad i, j \in \mathcal{S}.$$

This system of equations can be written in matrix form in terms of $R(\eta)$, whose solution is given in terms of the generator Q by

$$R(\eta) = \eta(\eta I - Q)^{-1}.$$

Therefore, if we let $\eta := k/t$ and $S_{k,t}$ be an Erlang distributed random variable with rate parameter k/t and shape parameter k , then the (i, j) th entry of the matrix $R(k/t)^k$ contains $\mathbb{P}(Z(S_{k,t}) = j \mid Z(0) = i)$. The expected value of $S_{k,t}$ is t and the variance of $S_{k,t}$ is t/k . This means that

$$P(t) = \lim_{k \rightarrow \infty} R(k/t)^k. \quad (6)$$

The Erlangization method for approximating transition probabilities is discussed in Asmussen *et al.* (2002); Mandjes and Taylor (2016); Stanford, Yu, and Ren (2011); Mandjes and Sollie (2021) for models related to birth-and-death processes. Similar to the uniformization method,

error arises in the Erlangization method since the state space may need to be truncated, and the infinite limit in (6) needs to be approximated by a finite k .

Inverse Laplace transform

The Laplace transform of the transition function $p_{i,j}(t)$ is

$$f_{i,j}(s) = \mathcal{L}[p_{i,j}](s) = \int_0^\infty p_{i,j}(t)e^{-st}dt. \quad (7)$$

Let $\frac{u_1}{v_1 +} \frac{u_2}{v_2 +} \frac{u_3}{v_3 +} \dots$ be a short-hand notation for the continued fraction

$$\frac{u_1}{v_1 + \frac{u_2}{v_2 + \frac{u_3}{v_3 + \dots}}},$$

where $(u_i, i = 1, 2, \dots)$ and $(v_i, i = 1, 2, \dots)$ are sequences of real numbers. As first reported in [Murphy and O'Donohoe \(1975\)](#) and detailed in [Crawford and Suchard \(2012\)](#), the Laplace transform (7) takes the continued fraction form

$$f_{i,j}(s) = \begin{cases} \left(\prod_{k=j+1}^i \mu_k \right) \frac{B_j(s)}{B_{j+1}(s)+} \frac{B_i(s)}{b_{i+2}(s)+} \frac{a_{i+2}}{b_{i+3}(s)+} \dots, & \text{for } j \leq i, \\ \left(\prod_{k=i}^{j-1} \lambda_k \right) \frac{B_i(s)}{B_{j+1}(s)+} \frac{B_j(s)}{b_{j+2}(s)+} \frac{a_{j+1}}{b_{j+3}(s)+} \dots, & \text{for } j \geq i, \end{cases} \quad (8)$$

where

$$\begin{aligned} B_0(s) &= 1, \\ B_1(s) &= b_1(s), \quad \text{and} \\ B_k(s) &= b_k(s)B_{k-1}(s) + a_k B_{k-2}(s), \quad \text{for } k \geq 2, \end{aligned}$$

with $a_1 = 1$ and $a_j = -\lambda_{j-2}\mu_{j-1}$ for $j \geq 2$, and $b_1(s) = s + \lambda_0$ and $b_j(s) = s + \lambda_{j-1} + \mu_{j-1}$ for $j \geq 2$. An advantage of the continued fraction form of (8) is that it can be evaluated using the Lentz algorithm to a user-specified error tolerance ([Press, William, Teukolsky, Saul, Vetterling, and Flannery 2007](#), Chapter 5). The Laplace transform can then be numerically inverted to obtain $p_{i,j}(t)$.

Numerical Laplace transform inversion has been the subject of intense research efforts over the past few decades. Some recent reviews include [Kuhlman \(2013\)](#); [Wang and Zhan \(2015\)](#), while [Davies and Martin \(1979\)](#); [Abate, Choudhury, and Whitt \(2000\)](#) provide earlier (yet thorough) discussion on the topic. Methods which have stood the test of time are presented in [De Hoog, Knight, and Stokes \(1982\)](#); [Talbot \(1979\)](#); [Stehfest \(1970\)](#); [Valkó and Abate \(2004\)](#), a unified framework is developed in [Abate and Whitt \(1995, 2006\)](#), and a promising new approach is developed in [Horváth, Horváth, Almousa, and Telek \(2020\)](#).

Diffusion approximation

Define $(Z_i^{(r)}, r \in \mathbb{N})$ to be a parametric family of CTMCs which evolve according to transition rates $\lambda_z^{(r)} = r\lambda_{z/r}$ and $\mu_z^{(r)} = r\mu_{z/r}$ with $Z_i^{(r)}(0) = i$. For $s \in [0, t]$ define the diffusion-scaled process

$$\tilde{Z}_i(s) = \lim_{r \rightarrow \infty} \sqrt{r} \left(\hat{Z}_i^{(r)}(s) - \hat{z}_i(s) \right),$$

where $\hat{Z}_i^{(r)} := \frac{1}{r} Z_i^{(r)}(s)$ and $\hat{z}_i(s)$ satisfies

$$\frac{d}{ds} \hat{z}_i(s) = \lambda_{\hat{z}_i(s)} - \mu_{\hat{z}_i(s)}, \quad \hat{z}_i(0) = i.$$

Loosely speaking, Theorem 3.5 in Kurtz (1971) can be used to show that, under some regularity conditions, $(\hat{Z}_i^{(r)}(s), s \in [0, t])$ converges weakly, as $r \rightarrow \infty$, in the space of càdlàg functions on $[0, t]$ to a zero-mean Gaussian diffusion $(\tilde{Z}_i(s), s \in [0, t])$ with variance

$$\sigma_i^2(s) := \text{Var}(\tilde{Z}(s)) = M(s)^2 \int_0^s (\lambda_{\tilde{z}(\tau)} + \mu_{\tilde{z}(\tau)}) M(\tau)^{-2} d\tau$$

for each $s \in [0, t]$, where $M(s) := \exp(\int_0^s B(\tau) d\tau)$ with $B(\tau) = H(\hat{z}(\tau))$ defined in terms of

$$H(z) = \frac{d}{dz} (\lambda_z - \mu_z).$$

In particular this implies that

$$p_{i,j}(t) \approx \mathbb{P}(\tilde{Z}_i(t) = j).$$

Hence $p_{i,j}(t)$ can be approximated by the probability density of a normally distributed random variable with mean $\hat{z}_i(t)$ and variance $\sigma_i^2(t)$ as given above.

This approach is very closely related to the functional central limit theorem (e.g., Theorem 4.3.2 in Whitt 2002). The diffusion approach to approximate transition probabilities is used in Ross *et al.* (2009), and discussed at length in Allen (2008).

Ornstein-Uhlenbeck approximation

The diffusion approximation discussed above can be substantially simplified if it is assumed that Z is fluctuating about a steady state point z_{eq} of \hat{z} . Such a point occurs when the birth rate is equal to the death rate, i.e., z_{eq} satisfies $\lambda_{z_{\text{eq}}} = \mu_{z_{\text{eq}}}$. In this case, as argued in Ross *et al.* (2006), the limiting Gaussian diffusion is an Ornstein-Uhlenbeck process. Therefore, $p_{i,j}(t)$ can be approximated by the density of a normally distributed random variable with mean $\tilde{z} = z_{\text{eq}} + e^{H(z_{\text{eq}})t}(i - z_{\text{eq}})$ and variance $\tilde{\sigma}^2 = \frac{\lambda_{z_{\text{eq}}} + \mu_{z_{\text{eq}}}}{2H(z_{\text{eq}})} (e^{2H(z_{\text{eq}})t} - 1)$. This method assumes that z_{eq} is asymptotically stable (i.e., $H(z_{\text{eq}}) < 0$), and as such favors values of z_{eq} that minimize $H(z_{\text{eq}})$.

Galton-Watson approximation

Recall from Section 2.2 that a PSDBDP can be approximated by a piecewise-linear birth-and-death process by decomposing time into sub-intervals of finite length, and letting the per-individual birth and death rates be constant over each time interval; more precisely, if z is the population size at the beginning of a time interval, then we let $\lambda := \frac{1}{z} \lambda_z$ and $\mu := \frac{1}{z} \mu_z$ over that interval (in Section 2.2 we assumed the intervals to be of constant length τ , but this assumption is not necessary). We can extend this idea to obtain approximations for the transition probabilities in a PSDBP.

Suppose we want to approximate $p_{i,j}(t)$ for some $i, j \in \mathcal{S}$, and $t > 0$. Consider a linear birth-and-death process $\hat{Z}^{(b)}$ with per-individual birth rate $\lambda = \frac{1}{b} \lambda_b$ and per-individual death rate $\mu = \frac{1}{b} \mu_b$, where possible choices of b include $b = i$ (such as in Section 2.2), $b = j$,

$b = \max(i, j)$, $b = \min(i, j)$, and $b = (i + j)/2$. The transition probability $p_{i,j}(t)$ can then be approximated by

$$p_{i,j}(t) \approx \hat{p}_{i,j}(t) := \mathbb{P}(\hat{Z}^{(b)}(t) = j \mid \hat{Z}^{(b)}(0) = i).$$

What constitutes a good choice of b will depend highly on i , j and the model under study (this is a topic for future research). This approximation is particularly convenient since it is well known (see [Guttorp 1991](#)) that $\hat{p}_{i,0}(t) = \beta_1(t)^i$, and for $j \geq 1$,

$$\hat{p}_{i,j}(t) = \sum_{k=\max(0, i-j)}^{i-1} \binom{i}{k} \binom{j-1}{i-k-1} \beta_1(t)^k [\{1 - \beta_1(t)\} \{1 - \beta_2(t)\}]^{j-k} \beta_2(t)^{j-i+k}, \quad (9)$$

where $\beta_1(t)$ and $\beta_2(t)$ are given, respectively, by (1) and (2). When the binomial coefficients in (9) cause numerical problems or take a long time to compute, an alternative expression developed by [Davison et al. \(2021\)](#) using a saddlepoint approximation ([Butler 2007](#)) may be used.

Monte Carlo simulation

By using the simulation methods described in Section 2.2, it is possible to obtain k realizations of $Z(t)$ conditional on $Z(0) = i$. The proportion of these realizations which equal j can be used to approximate the transition probability $p_{i,j}(t)$. That is,

$$p_{i,j}(t) \approx \frac{1}{k} \sum_{n=1}^k 1_{\{\hat{Z}_n(t)=j\}}$$

where $\hat{Z}_n(t)$ is a simulated realization of $Z(t)$ with $Z(0) = i$.

We conclude this section with Table 3 which summarizes the methods for computing (or approximating) the transition probabilities and gives the label used to call them in **BirDePy**, as described in Appendix A.

2.4. Estimation

In this section, we outline the parameter estimation methods implemented in **BirDePy**.

Direct numerical maximization for continuously-observed data

For a continuously observed PSDBDP, m independent ‘observations’, or sample paths of the form $(Z(t), t \in [0, T_k])$, are assumed to be available. A convenient representation for observation k under this assumption is a list of jump times $t_{0,k} < t_{1,k} < \dots < t_{n_k,k}$ ($1 \leq k \leq m$ and $n_k \geq 1$), together with a corresponding list of states at those times $(z_{0,k}, z_{1,k}, \dots, z_{n_k,k})$. Under this representation, the data, denoted by \mathbf{y} , consists of two lists. The first list contains the lists of jump times, and the second list contains the corresponding lists of states. The log-likelihood function for continuously-observed PSDBDPs is given by

$$\tilde{\ell}(\mathbf{y}; \boldsymbol{\theta}) = \sum_{z=0}^{\infty} U_z \log(\lambda_z(\boldsymbol{\theta})) + D_z \log(\mu_z(\boldsymbol{\theta})) - (\lambda_z(\boldsymbol{\theta}) + \mu_z(\boldsymbol{\theta})) H_z, \quad (10)$$

where $U_z = \sum_k U_z^k$, $D_z = \sum_k D_z^k$, and $H_z = \sum_k H_z^k$, with U_z^k and D_z^k being, respectively, the number of births and deaths whilst in state z for sample path k , and H_z^k is the total

Method (in order of appearance)	Label	Brief description
Matrix exponential	"expm"	Uses $P(t) = \exp(Qt)$.
Uniformization	"uniform"	Evaluates probability using an approximating discrete-time process.
Erlangization	"Erlang"	Evaluates probability at an Erlang-distributed time.
Inverse Laplace transform	"ilt"	Numerically inverts Laplace transform.
Diffusion approximation	"da"	Approximates true model by a general diffusion-scaled model.
Ornstein-Uhlenbeck approx.	"oua"	Approximates true model by a simple diffusion process.
Galton-Watson approximation	"gwa"	Approximates true model with a linear model.
Saddlepoint approximation	"gwas"	As above combined with a saddlepoint approximation
Simulation	"sim"	Average of Monte Carlo samples.

Table 3: Methods for computing transition probabilities.

cumulative time spent in state z in the time interval $[0, T_k]$ of sample path k . Therefore U_z , D_z , and H_z are sufficient statistics for the data. For PSDBPs where the birth and death rates can be written in the form $\lambda_z = f(z)\lambda$ and $\mu_z = g(z)\mu$, where $f(z)$ and $g(z)$ are known,

$$\lambda^* = \frac{\sum_z U_z}{\sum_z f(z)H_z} \quad \text{and} \quad \mu^* = \frac{\sum_z D_z}{\sum_z g(z)H_z}$$

solve the maximization problem

$$(\lambda^*, \mu^*) =: \boldsymbol{\theta}^* = \arg \max_{\boldsymbol{\theta}} \tilde{\ell}(\mathbf{y}; \boldsymbol{\theta}).$$

This is discussed at length by [Wolff \(1965\)](#) and [Reynolds \(1973\)](#). These maximum likelihood estimates are the parameters that make the data most likely under the assumed model. For more general forms of the birth and death rates (or when the parameters defining the functions f or g are unknown), it is possible to find estimates $\boldsymbol{\theta}^*$ of $\boldsymbol{\theta}$ by directly numerically maximizing (10).

Direct numerical maximization for discretely-observed data

Discrete observation schemes have been receiving increasing attention in recent years. Under these schemes, m independent sample paths of Z are observed, where states $z_{0,k}, z_{1,k}, \dots, z_{n_k,k}$ of sample path k are observed at respective times $t_{0,k} < t_{1,k} < \dots < t_{n_k,k}$. In this case, the data, denoted \mathbf{z} , still consists of two lists. The first list contains the m lists of observation times, and the second list contains the m lists of corresponding states. Since the full trajectory of the process is now unobserved between observation times, \mathbf{z} is different from the data \mathbf{y} defined in the previous section.

The likelihood of the discrete-observation data \mathbf{z} is given by

$$\ell(\mathbf{z}; \boldsymbol{\theta}) = \prod_{k=1}^m \prod_{i=1}^{n_k} p_{z_{i-1,k}, z_{i,k}}(\Delta_{i,k}), \quad (11)$$

where $\Delta_{i,k} := t_{i,k} - t_{i-1,k}$ denote the inter-observation times. Since there are generally no explicit expressions for the transition probabilities $p_{i,j}(t)$, the likelihood in (11) can be approximated using the methods for approximating $p_{i,j}(t)$ discussed in Section 2.3. Direct numerical maximization of this function then leads to approximate maximum likelihood estimates

$$\boldsymbol{\theta}^* = \arg \max_{\boldsymbol{\theta}} \ell(\mathbf{z}; \boldsymbol{\theta}).$$

Expectation maximization

Another way to find maximum likelihood estimates is to treat each unobserved path of Z between times $t_{i-1,k}$ and $t_{i,k}$ as ‘missing data’, and use the EM algorithm of [Dempster et al. \(1977\)](#). The EM algorithm is an iterative procedure where each iteration consists of an ‘E-step’ and an ‘M-step’. The ‘E-step’ is concerned with the computation of the expected value of the log-likelihood function for continuously-observed PSDBDPs (10), conditional on the current iteration’s parameter estimate $\boldsymbol{\theta}$ and on the discretely-observed data \mathbf{z} . In the ‘M-step’, this expectation is then maximized to obtain a new parameter estimate.

Recall the log-likelihood function for continuously observed PSDBDPs (10) and the fact that it depends only on the sufficient statistics U_z , D_z and H_z . Given a pair of observations $Z(0) = i$ and $Z(t) = j$ of a birth-and-death process, it is well known (e.g., [Lange 1995a](#); [Holmes and Rubin 2002](#); [Bladt and Sørensen 2005](#)) that

$$\begin{aligned} u_{z;i,j}(t; \boldsymbol{\theta}) &:= \mathbb{E}[U_z \mid Z(0) = i, Z(t) = j] = \frac{\int_0^t p_{i,z}(s) \lambda_z p_{z+1,j}(t-s) ds}{p_{i,j}(t)}, \\ d_{z;i,j}(t; \boldsymbol{\theta}) &:= \mathbb{E}[D_z \mid Z(0) = i, Z(t) = j] = \frac{\int_0^t p_{i,z}(s) \mu_z p_{z-1,j}(t-s) ds}{p_{i,j}(t)}, \\ h_{z;i,j}(t; \boldsymbol{\theta}) &:= \mathbb{E}[H_z \mid Z(0) = i, Z(t) = j] = \frac{\int_0^t p_{i,z}(s) p_{z,j}(t-s) ds}{p_{i,j}(t)}, \end{aligned} \quad (12)$$

where λ_z , μ_z and $p_{i,j}(t)$ implicitly depend on $\boldsymbol{\theta}$. Using (10) and (12), the conditional expected value of the log-likelihood function for continuously-observed PSDBDPs is

$$\begin{aligned} f(\boldsymbol{\theta}', \boldsymbol{\theta}) &= \mathbb{E}[\tilde{\ell}(\mathbf{y}; \boldsymbol{\theta}') \mid \mathbf{z}, \boldsymbol{\theta}] \\ &= \sum_{k=1}^m \sum_{i=1}^{n_k} \sum_{z=0}^{\infty} \left\{ u_{z;z_{i-1,k};z_{i,k}}(\Delta_{i,k}; \boldsymbol{\theta}) \log(\lambda_z(\boldsymbol{\theta}')) \right. \\ &\quad \left. + d_{z;z_{i-1,k};z_{i,k}}(\Delta_{i,k}; \boldsymbol{\theta}) \log(\mu_z(\boldsymbol{\theta}')) \right. \\ &\quad \left. - (\lambda_z(\boldsymbol{\theta}') + \mu_z(\boldsymbol{\theta}')) h_{z;z_{i-1,k};z_{i,k}}(\Delta_{i,k}; \boldsymbol{\theta}) \right\}. \end{aligned} \quad (13)$$

Starting with an arbitrary vector $\boldsymbol{\theta}^{(0)}$, an EM algorithm estimate follows from repeating $\boldsymbol{\theta}^{(k+1)} = \arg \max_{\boldsymbol{\theta}'} f(\boldsymbol{\theta}', \boldsymbol{\theta}^{(k)})$, until $|\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}|$ is small or a maximum number of iterations has taken place.

Clearly, (12) needs to be computed in order to use the EM algorithm just described. A first possible way to do this is to simply use any of the methods from Section 2.3 to evaluate the transition probability functions $p_{i,j}(t)$, and then compute the integrals numerically using, for example, the trapezoidal rule. Upon choosing to compute $p_{i,j}(t)$ using the matrix exponential

method or the inverse Laplace transform method, more sophisticated approaches are available, as we briefly describe in the next two paragraphs.

Recalling $P(t) = \exp(Qt)$ (which requires numerical approximation) from Section 2.3, upon defining \mathbf{e}_i as the i -th basis vector in $(|\mathcal{S}|+1)$ -dimensional euclidean space, for any $a, b, i, j \in \mathcal{S}$, we can write

$$\int_0^t p_{i,a}(s) p_{b,j}(t-s) ds = \mathbf{e}_i \int_0^t \exp(Qs) \mathbf{e}_a^\top \mathbf{e}_b \exp(Q(t-s)) ds \mathbf{e}_j^\top.$$

Let $G(t) = \int_0^t \exp(Qs) \mathbf{e}_a^\top \mathbf{e}_b \exp(Q(t-s)) ds$. According to Van Loan (1978), if

$$C := \begin{bmatrix} Q & \mathbf{e}_a^\top \mathbf{e}_b \\ 0 & Q \end{bmatrix}, \quad \text{then} \quad \exp(Ct) = \begin{bmatrix} \cdot & G(t) \\ \cdot & \cdot \end{bmatrix},$$

where ‘ \cdot ’ denotes parts of the matrix that can be discarded. Hence the integrals in (12) can be computed by extracting the relevant parts of the matrix exponential $\exp(Ct)$.

On the other hand, if \mathcal{L}^{-1} denotes the inverse of a Laplace transform \mathcal{L} (e.g., (7)), then we also have

$$\int_0^t p_{i,a}(s) p_{b,j}(t-s) ds = \mathcal{L}^{-1}[f_{i,a}(s) f_{b,j}(s)](t)$$

by the properties of Laplace transforms. Numerical Laplace transform inversion therefore provides a third technique for evaluating the integrals in (12). This approach is discussed by Crawford *et al.* (2014).

The EM algorithm can exhibit slow convergence, calling for many iterations before a suitably accurate estimate is provided. Each iteration may itself use substantial computational time and effort due to requiring many matrix exponential computations, Laplace transform inversions, or numerical integration computations. To mitigate this, several schemes have been developed to accelerate the convergence rate of EM algorithms. In Jamshidian and Jennrich (1997), four such schemes are described in detail. Other seminal work includes Jamshidian and Jennrich (1993) and Lange (1995b). Using optimization-based ideas, these accelerators can yield substantial improvements in computational speed. Recall f as defined in (13) and let $\tilde{\mathbf{g}}(\boldsymbol{\theta}) = [\arg\max_{\boldsymbol{\theta}'} f(\boldsymbol{\theta}', \boldsymbol{\theta})] - \boldsymbol{\theta}$, which is the change in parameter values when a non-accelerated EM iteration is performed. Loosely speaking, EM acceleration techniques treat $\tilde{\mathbf{g}}$ as a generalized derivative of the likelihood function (11) and aim to find $\boldsymbol{\theta}^*$ such that $\tilde{\mathbf{g}}(\boldsymbol{\theta}^*) = \mathbf{0}$.

Least squares estimation

The two likelihood approaches discussed so far may be computationally demanding. A potential avenue to reducing the computational complexity is to instead use a least-squares-based estimator. Let $m_i(t) := \mathbb{E}[Z(t) \mid Z(0) = i]$ correspond to the expected population size after t units of time in a birth-and-death process starting with i individuals ($m_i(t)$ depends implicitly on the parameters $\boldsymbol{\theta}$). The computation of $m_i(t)$ may be numerically less demanding than the computation of $p_{i,j}(t)$. Hence it may be practical to consider the least-squares estimator

$$\boldsymbol{\theta}^\dagger = \arg \min_{\boldsymbol{\theta}} \sum_{k=1}^m \sum_{i=1}^{n_k} (z_{i,k} - m_{z_{i-1,k}}(\Delta_{i,k}))^2.$$

Indeed, for linear BDPs, $m_i(t) = i \exp((\lambda - \mu)t)$, which is straightforward to compute, and allows a least-squares estimate to be found in this case. Using the Galton-Watson approach discussed in Section 2.3, this can be extended to PSDBDPs. In addition, the deterministic approximation to the mean underlying the diffusion approximation discussed in Section 2.3 can also play the role of an approximate mean. This would certainly be less computationally demanding than calling upon the full diffusion approximation. Finally, any of the methods for computing approximate transition probabilities discussed in Section 2.3 can be used to compute an approximate expected value $m_i(t) = \sum_z p_{i,z}(t)z$.

Approximate Bayesian computation (ABC)

The methods discussed so far have relied on using approximations to model properties which have no explicit expression (specifically, the likelihood of the observed data, and the expectation of the population size). ABC bypasses the need to use these model properties in the first place. This method works by repeatedly comparing simulated data with the observed data to find an approximation to a distribution that characterizes uncertainty about the value of θ (called the posterior distribution). Given a distance measure d , the standard ABC method (as proposed in Pritchard, Seielstad, Perez-Lezaun, and Feldman 1999) consists of repeating the following steps:

1. Generate a parameter proposal θ from the prior distribution π .
2. Simulate observations $\hat{z}(\theta)$ consisting of points $\hat{z}_{i,k}$ generated using θ with initial conditions $z_{i-1,k}$ and elapsed times $\Delta_{i,k}$ (for $k = 1, \dots, m$, $i = 1, \dots, n_k$).
3. Accept θ as an approximate observation from the posterior distribution if $d(z, \hat{z}(\theta)) < \epsilon$, where z is the observed data and ϵ is a predetermined error tolerance threshold.

These steps are repeated until an arbitrarily large number of parameter proposals are accepted. A parameter point estimate can then be obtained by taking the mean or median of the accepted parameter proposals. While choices of d and π have been investigated for continuously-observed linear BDPs (e.g., Janzen, Höhna, and Etienne 2015), to the authors' knowledge, there is a lack of research on suitable choices of d and π for general discretely-observed birth-and-death processes. A potentially suitable choice, previously discussed in the context of linear birth-and-death processes (see Tavaré 2018) is

$$d(z, \hat{z}) = \sqrt{\sum_{k=1}^m \sum_{i=1}^{n_k} (z_{i,k} - \hat{z}_{i,k}(\theta))^2}. \quad (14)$$

The prior π can simply be taken as a uniform distribution on the set of allowed parameters. The choice of threshold ϵ in Step 3 above has a strong influence on the probability of a parameter proposal being accepted. Larger choices of ϵ may allow the desired number of accepted parameter proposals to be found more quickly, but this could be at the expense of accuracy. Similarly, the choice of the prior distribution π has a major impact on performance. To address these concerns, the standard ABC method can be applied iteratively with dynamically determined ϵ and π . Early work in this direction can be found in Beaumont, Cornuet, Marin, and Robert (2009) and Del Moral, Doucet, and Jasra (2012). More recently, Simola, Cisewski-Kehe, Gutmann, and Corander (2021) proposed a method for adaptively selecting

a threshold at each iteration by comparing the estimated posterior from the two previous iterations and the distances between the sampled data and the accepted parameter proposals in the previous iteration.

2.5. Forecasting

Confidence and prediction intervals are two simple ways to convey information about possible future states of a stochastic process. Confidence intervals are focused on parameter uncertainty, while prediction intervals incorporate parameter uncertainty and model stochasticity. Let Z_θ be a birth-and-death process evolving according to parameters θ . Given $Z(0) = i$, associated with each possible θ is an expected trajectory of the process $(m_\theta(t), t \in [0, T])$, where $m_\theta(t) = \mathbb{E}Z_\theta(t)$. The paths m_θ can, for example, be approximated using simulation or by the mean curve of the diffusion approximation described in Section 2.3. When uncertainty about parameter values is characterized by some distribution, a confidence interval can be formed by sampling from this distribution (e.g., the distribution of an estimator or the posterior distribution) and collecting together the associated approximations of m_θ . Alternatively, each sampled θ can be used to generate a sample $(Z_\theta(t), t \in [0, T])$ with $Z_\theta(0) = i$. Collecting these samples together allows for an approximate prediction interval to be formed.

3. Numerical examples

In this section, we present several numerical examples to show **BirDePy** in action. The goal of these examples is to demonstrate the execution of the vast majority of the **BirDePy** codebase. In particular, an effort is made to ensure that functions and methods which are not used in the case study in Section 4 are used in this section. It is out of the scope of this paper to provide comprehensive evidence to compare the alternative methods implemented within each function. We do, however, provide CPU times to give users a rough indication of how long each operation may be expected to take. Throughout the section, we assume the following commands have already been executed:

```
import birdepy as bd
import birdepy.gpu_functions as bdg
import numpy as np
```

In this section, numbers in $[0.0001, 1)$ are rounded to 4 decimal places, numbers in $[1, 10)$ are rounded to 2 decimal places, and numbers greater than 10 are rounded to the nearest whole number. Any output which was returned as a `nan` value is indicated by *. The models used in our examples are discussed in more detail in Section 2.1 and summarized in Table 1.

In Section 3.1, we demonstrate the use of the simulation functions `bd.simulate.discrete()`, `bd.simulate.continuous()` and `bdg.discrete()`. In Section 3.2, we explore the different methods implemented in `bd.probability()` and `bdg.probability()`. Finally, Section 3.3 utilizes the alternative parameter estimation frameworks and their underlying techniques. The code for these examples can be found at https://github.com/birdepy/birdepy_examples.

3.1. Simulation

To illustrate the simulation capabilities of **BirDePy**, we consider a Hassell model with $\gamma = 0.75$, $\nu = 0.25$, $\alpha = 0.01$ and $c = 1$ (recall that a Hassell model with $c = 1$ is also known as a

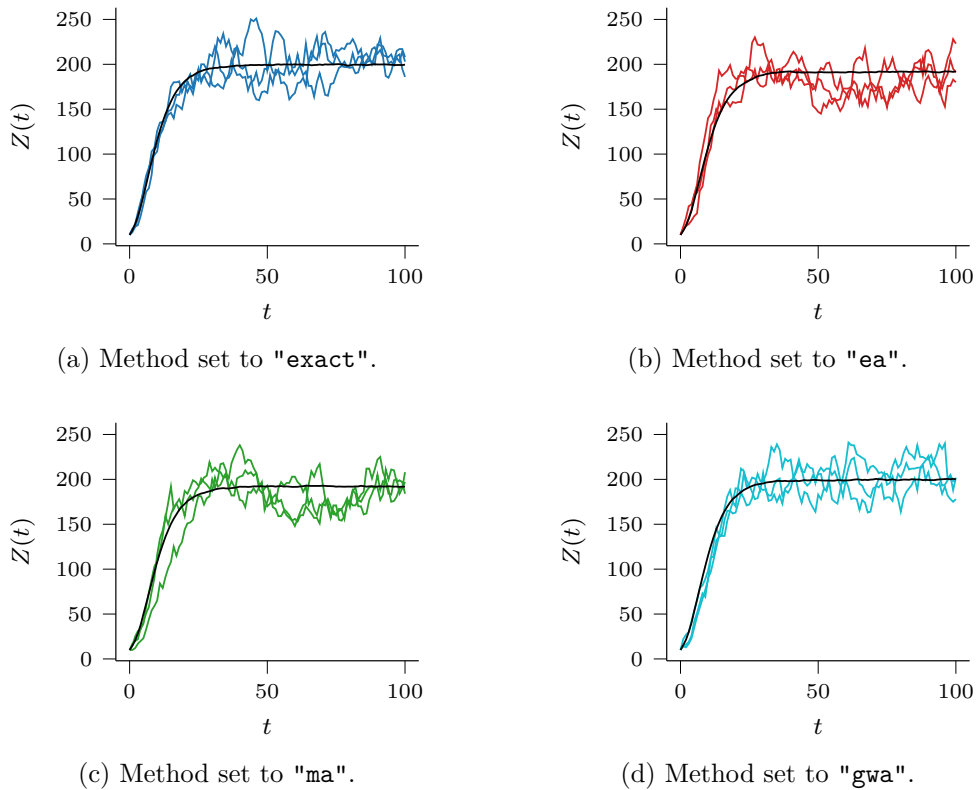


Figure 1: Sample paths and mean approximations of a Hassell model with $\gamma = 0.75$, $\nu = 0.25$, $\alpha = 0.01$ and $c = 1$ generated using `bd.simulate.discrete()` with method set to: (a) "exact", (b) "ea", (c) "ma" and (d) "gwa".

Beverton-Holt model). Initializing with $Z(0) = 10$, the four plots in Figure 1 each contain three simulated sample paths of the process, along with a mean curve estimated from 10^3 sample paths. The difference between each plot is the simulation algorithm used to generate the sample paths. In all cases,

```
bd.simulate.discrete(param = [0.75, 0.25, 0.01, 1], model = 'Hassell',
  z0 = 10, times = np.arange(0,101,1), k = 10**3, method = m, seed = 2021)
```

is executed, and for each plot (from left to right, and top to bottom), the value of `m` is set to "exact", "ea", "ma" and "gwa" (as described in Table 2). Observe that due to differences in the way that random numbers are utilized by each simulation method, the sample paths are not identical, even though the same seed is used. Despite this, the three approximation methods "ea", "ma" and "gwa" generate sample paths that appear highly similar to the one generated by the "exact" method.

Figure 2 provides a more detailed examination of the simulated output at time $t = 100$. The figure displays kernel density estimates (KDEs) for the distribution of $Z(100)$ generated from 10^3 samples. In addition to including output of the four algorithms implemented in `bd.simulate.discrete()`, the figure also includes a KDE generated using the output of executing

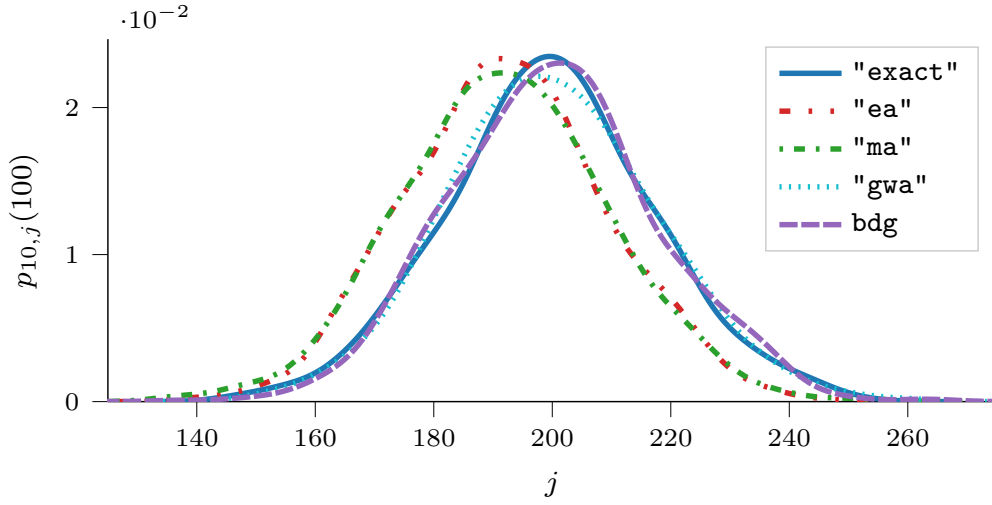


Figure 2: Kernel density estimates for the distribution of $Z(100)$ conditional on $Z(0) = 10$ for a Hassell model with $\gamma = 0.75$, $\nu = 0.25$, $\alpha = 0.01$ and $c = 1$, generated using different methods.

"exact"	"ea"	"ma"	"gwa"	bdg.discrete()
73	2.81	7.19	14	0.1090

Table 4: Computation time (CPU + GPU) in seconds used to generate the sample paths underlying the KDEs in Figure 2.

```
bdg.discrete(param = [0.75, 0.25, 0.01, 1], model = "Hassell", z0 = 10,
             times = [0, 100], k = 10**3, seed = 2021)
```

which provides samples generated using exact simulation performed on a GPU. This figure makes it clear that the approximated simulation methods can indeed differ notably from exact simulation. Despite the seed being the same and both using the same ‘exact’ simulation algorithm, `bd.simulate.discrete(method = "exact")` and `bdg.discrete()` generate different KDEs (for this sample size) due to the different way that each function handles random numbers. As displayed in Table 4, the methods and functions also differ substantially in their computation time. So that these times can be fairly compared, `np.arange(0, 101, 1)` is replaced by `[0, 100]` in the execution of `bd.simulate.discrete()` for the purpose of generating Figure 2.

We conclude this section by comparing the output of

```
bd.simulate.discrete(param = [0.5, 0.45], model = "linear", z0 = 10,
                    times = np.arange(0, 3, 0.1), seed = 2021)
```

which generates a discretely-observed trajectory using exact simulation, and

```
bd.simulate.continuous(param = [0.5, 0.45], model = "linear", z0 = 10,
                      t_max = 3, seed = 2021)
```

which generates a continuously-observed trajectory (also exactly). In both cases a linear birth-and-death process with $\gamma = 0.5$ and $\nu = 0.45$ initiated with $Z(0) = 10$ is simulated

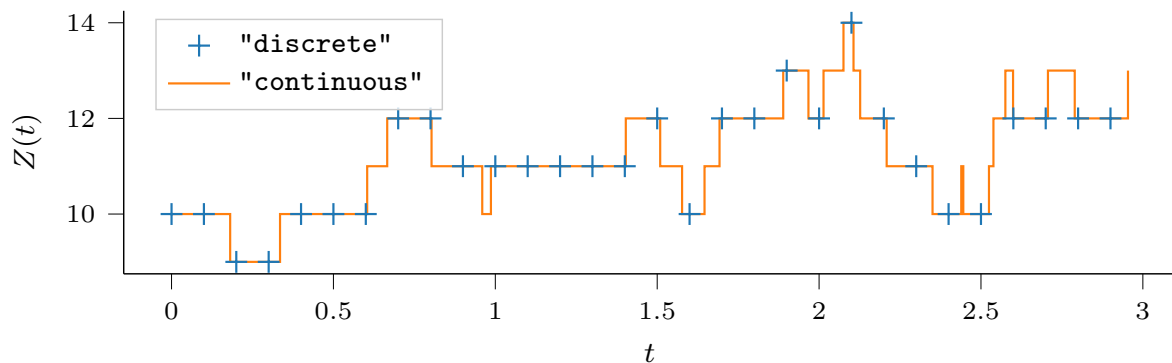


Figure 3: Discretely-observed and continuously-observed simulated trajectories of a linear birth-and-death process with $\gamma = 0.5$ and $\nu = 0.45$ initiated with $Z(0) = 10$.

bdg	"expm"	"uniform"	"Erlang"	"ilt"	"da"	"oua"	"gwa"	"gwasa"
0.5840	0.0070	0.7700	0.0030	2.0500	0.0050	0.0040	0.0250	0.0030

Table 5: Computation time (CPU + GPU) in seconds used to generate the densities in Figure 4.

over 3 units of time. The trajectories are plotted in Figure 3. Observe that in this case, setting the seeds equal ensures that the output from the two functions matches at the discrete observation times.

3.2. Transition probabilities

In this section, we illustrate how to compute approximate transition probabilities $p_{i,j}(t)$ using **BirDePy**. Consider a Verhulst model with $\gamma = 0.8$, $\nu = 0.4$, $\alpha = 0.025$, and $\beta = 0$. This model is also studied in Ross *et al.* (2006). Figure 4 displays approximations to $p_{15,j}(1) = \mathbb{P}(Z(1) = j \mid Z(0) = 15)$ for $0 \leq j < 40$, as outputted by executing

```
bd.probability(z0 = 15, zt = np.arange(0, 40, 1), t = 1,
param = [0.8, 0.4, 0.025, 0], model = "Verhulst", method = m, seed = 2021)
```

with `m` taking on the values "expm", "uniform", "Erlang", "ilt", "da", "oua", "gwa" and "gwasa" (as described in Table 3). In the background of each plot is a density generated by executing

```
bdg.probability(z0 = 15, zt = np.arange(0, 40, 1), t = 1,
param = [0.8, 0.4, 0.025, 0], model = "Verhulst", k = 10**6, seed = 2021)
```

which provides a simulation-based approximation for verification of the reliability of the output. All of the methods provide accurate transition probability approximations. The methods "gwa" and "gwasa" exhibit slight inaccuracy around the mode of the distribution. Table 5 reports the computation times needed to generate each of the plots in Figure 4. These times vary greatly, with "gwasa" being the fastest and "ilt" being the slowest.

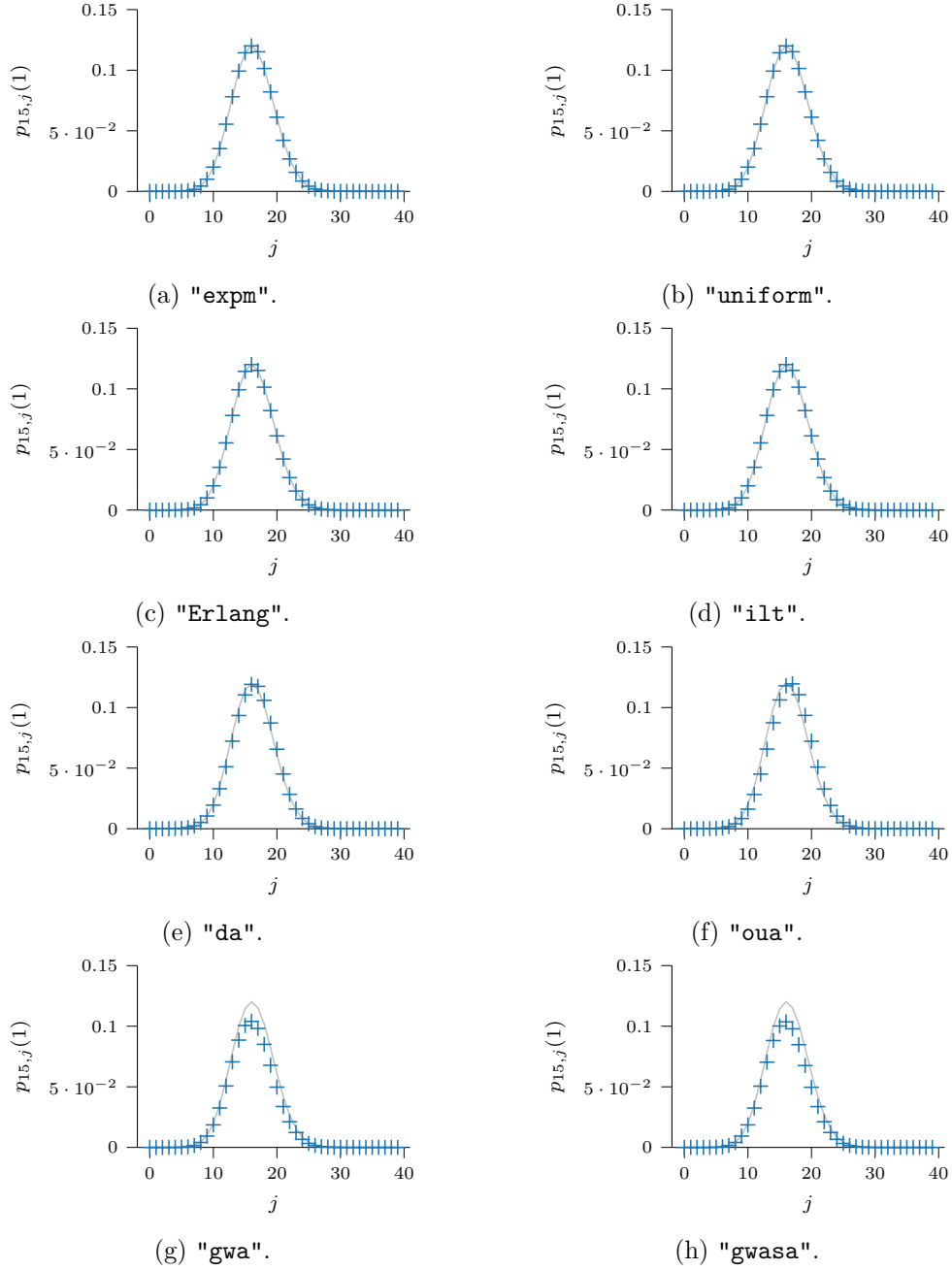


Figure 4: Transition probabilities $p_{15,j}(1) = \mathbb{P}(Z(1) = j \mid Z(0) = 15)$ for the Verhulst model with $\gamma = 0.8$, $\nu = 0.4$, $\alpha = 0.025$ and $\beta = 0$, approximated using simulation (gray) and as outputted by `bd.probability()` (blue crosses) with method set to: (a) "expm", (b) "uniform", (c) "Erlang", (d) "ilt", (e) "da", (f) "oua", (g) "gwa", and (h) "gwasa".

3.3. Estimation

We next demonstrate how to use the wide variety of estimation methods implemented in **BirDePy**. Consider a Verhulst model with $\gamma = 0.8$, $\nu = 0.4$, $\alpha = 0.025$, and $\beta = 0$. As per Section 3.1, executing

```

obs_times = list(range(100))
param = [0.8, 0.4, 0.025, 0]
p_data = bd.simulate.discrete(param, model = "Verhuls", z0 = 5,
    times = obs_times, k = num_sample_paths, seed = 2021)
t_data = [obs_times for _ in range(num_sample_paths)]

```

generates synthetic data for this model, which is used to generate all estimates in this section. This synthetic data consists of five sample paths, each with 100 equally-spaced observations. We first use the "abc" framework of `bd.estimate()`, which utilizes the ABC algorithm, to estimate γ from `t_data` and `p_data`. The usage of this framework is described in Section A.7. Executing

```

est = bd.estimate(t_data, p_data, p0 = [0.5], p_bounds = [[0,1]],
    framework = "abc", model = "Verhulst", known_p = [0.4, 0.025, 0],
    idx_known_p = [1, 2, 3], max_its = 1, seed = 2021)

```

results in an estimate of γ of 0.7468 being stored in the variable `est.p`, and a standard error of 0.1034 being stored in the variable `est.se`. To obtain this estimate we have specified 0.5 as an initial guess for the value of γ and placed bounds $[0, 1]$ on the estimate. The estimate took 74 seconds to be generated. Note that the above code utilises the option `max_its = 1`, meaning that this estimate is obtained after a single basic ABC iteration is performed. Removing this option (so that the default `max_its = 3` is utilised instead) results in a γ estimate equal to 0.7620 with standard error of 0.0533. By using more iterations the estimate becomes slightly more accurate and the standard error becomes substantially smaller. This improvement in accuracy brings with it a higher computational burden – in this case the estimate took 1853 seconds to be generated. This time is very large relative to other experiments we conducted using this framework with different models, however it indicates the framework is not working efficiently for all PSDBDPs. For this example we found that this framework failed to compute reasonable estimates of γ , unless values of ν , α and β were assumed to be known. These factors suggest that more research needs to be performed on developing ABC algorithms tailored to PSDBDPs.

Following this, we use the "dnm" framework of `bd.estimate()` which directly (numerically) maximizes approximations to the likelihood function. As described in Section A.8, when using this framework, several methods can be used to generate an approximation to the likelihood function, as determined by the argument of the `likelihood` parameter. For this framework, and in the following discussion of frameworks "em" and "lse", we prepare to use `bd.estimate()` by first executing

```

con = {"type": "ineq", "fun": lambda p: p[0]-p[1]}
alpha_max = 1/np.amax(np.array(p_data))
alpha_mid = 0.5 * alpha_max

```

which sets the constraint $\gamma > \nu$, provides an upper bound for α based on the maximum observed population, and uses this bound to provide an initial guess for α . We also set `opt_method` to "differential-evolution" as this produces more reliable estimates (at the cost of taking longer to compute them). Table 6 displays the estimates obtained for each parameter, and the corresponding asymptotic standard errors obtained by executing

Likelihood method (m)	γ	ν	α	Time (secs)
"da"	0.7817 (0.0612)	0.3906 (0.0292)	0.0252 (0.0013)	90
"Erlang"	0.7824 (0.0653)	0.3873 (0.0294)	0.0250 (0.0013)	0.8610
"expm"	0.7810 (0.0650)	0.3867 (0.0292)	0.0250 (0.0013)	1.1
"gwa"	0.4111 (0.0432)	0.3414 (0.0228)	0.0076 (0.0041)	88
"gwasa"	0.4198 (0.0438)	0.3494 (0.0239)	0.0075 (0.0040)	7.39
"ilt"	0.7782 (0.0047)	0.3852 (0.0052)	0.0251 (*)	4738
"oua"	0.6528 (0.0570)	0.3716 (0.0298)	0.0229 (0.0016)	16
"uniform"	0.7811 (0.0650)	0.3867 (0.0292)	0.0250 (0.0013)	2.85

Table 6: Estimates of (γ, ν, α) and standard errors (in brackets), along with the corresponding CPU times, for a Verhulst model with $\gamma = 0.8$, $\nu = 0.4$, and $\alpha = 0.025$ (with $\beta = 0$ assumed to be known), generated using `bd.estimate(framework = "dnm")` with different methods for approximating the likelihood function.

Technique	Accelerator	γ	ν	α	Time (secs)
"expm"	"cg"	0.7796 (0.0649)	0.3865 (0.0292)	0.0250 (0.0013)	4.66
"expm"	"none"	0.8175 (0.0712)	0.4079 (0.0325)	0.0248 (0.0013)	4.36
"expm"	"Lange"	0.8031 (0.0700)	0.4039 (0.0318)	0.0246 (0.0013)	4.44
"expm"	"qn1"	0.7810 (0.0650)	0.3867 (0.0292)	0.0250 (0.0013)	11.02
"expm"	"qn2"	0.7526 (0.0655)	0.4068 (0.0322)	0.0231 (0.0015)	3.6
"ilt"	"cg"	0.7815 (0.0649)	0.3862 (0.0289)	0.0250 (0.0012)	9410
"ilt"	"none"	0.8171 (0.0139)	0.4078 (0.0138)	0.0248 (0.0012)	8339
"ilt"	"Lange"	0.8060 (0.0378)	0.4025 (0.0305)	0.0248 (*)	9519
"ilt"	"qn1"	0.7802 (0.0649)	0.3864 (0.0290)	0.0250 (0.0013)	19995
"ilt"	"qn2"	0.7535 (0.0592)	0.4073 (0.0326)	0.0238 (0.0013)	6198
"num"	"cg"	0.8626 (0.0778)	0.4296 (0.0363)	0.0249 (0.0012)	12
"num"	"none"	0.6153 (0.0490)	0.4414 (0.0423)	0.0172 (0.0018)	88
"num"	"Lange"	0.7859 (0.0656)	0.3895 (0.0296)	0.0250 (0.0013)	27
"num"	"qn1"	0.5046 (*)	0.5039 (*)	0.0161 (*)	47
"num"	"qn2"	0.6609 (0.0625)	0.4275 (0.0352)	0.0184 (0.0020)	9.83

Table 7: Estimates (γ, ν, α) and standard errors (in brackets), along with the corresponding CPU time, for a Verhulst model with $\gamma = 0.8$, $\nu = 0.4$, and $\alpha = 0.025$ (with $\beta = 0$ assumed to be known), generated using `bd.estimate(framework = "em")` with different techniques for evaluating the integrals in (12) and different accelerators.

```
est = bd.estimate(t_data, p_data, p0 = [0.51, 0.5, alpha_mid],
  p_bounds = [[1e-6, 5], [1e-6, 5], [1e-6, alpha_max]], model = "Verhulst",
  framework = "dnm", known_p = [0], idx_known_p = [3], con = con,
  likelihood = likelihood, opt_method = "differential-evolution",
  seed = 2021)
```

for different choices of likelihood approximation method `m`. Many of the methods return accurate estimates. The final column of Table 6 reports the CPU times needed to obtain the estimates, which exhibit substantial variability. The Erlang likelihood method appears to be both the fastest and among the most accurate of the implemented methods for this example

Squares	γ	ν	α	Time (secs)
"expm"	0.7078 (0.0383)	0.2988 (0.0241)	0.0288 (0.0021)	21
"fm"	0.7109 (0.0367)	0.2955 (0.0257)	0.0279 (0.0025)	485
"gwa"	0.6792 (0.0480)	0.3278 (0.0354)	0.0244 (0.0024)	16

Table 8: Estimates of (γ, ν, α) and standard errors (in brackets), along with the corresponding CPU time, for a Verhulst model with $\gamma = 0.8$, $\nu = 0.4$, and $\alpha = 0.025$ (with $\beta = 0$ assumed to be known), generated using `bd.estimate(framework = "lse")` with different methods for evaluating the squared error.

when using the "dnm" framework. In our experience, Erlang performs well in many cases, and at times when it does not perform well, increasing the argument of parameter `k` reliably improves the performance (here `k = 150`, which is its default value).

Moving on, we use the "em" framework of `bd.estimate()` which implements EM algorithm approaches to obtaining maximum likelihood estimates. This framework has multiple choices for parameters `technique` and `accelerator` available in **BirDePy**, as detailed in Section A.9. The estimates, standard errors, and computational times are displayed in Table 7, and again exhibit a great deal of variability. The EM algorithm accelerated by the method of Lange (1995b) with the integrals in (12) evaluated using matrix exponentials appears to be highly accurate and efficiently computed. The corresponding estimate (0.8031, 0.4039, 0.0246) appears to outperform any estimate computed using the "dnm" framework at only a small cost in computation time of 4.44 seconds.

Finally, we use the "lse" framework of `bd.estimate()`. The estimates, standard errors, and computational times are displayed in Table 8, and are again highly variable, although not to the same scale as the "dnm" and "em" values. If a user is prepared to go without standard errors, the computation time is only a fraction of the displayed values, as the CPU times shown in the table are mostly consumed determining standard errors. For our example, this framework has not produced results which are as accurate as those produced by the other frameworks.

4. Case studies

In this section we consider two discretely-observed endangered bird populations. Section 4.1 examines the black robin population on Rangatira Island in New Zealand, and Section 4.2 studies whooping cranes which migrate annually between Canada's Wood Buffalo National Park (WBNP) and the Aransas National Wildlife Refuge (ANWR) in Texas. In general, **BirDePy** is intended as a decision support tool to be used in conjunction with expert opinion and combined with other analyses. These case studies are intended to illustrate the role that our package could play as part of a broader analysis of these bird populations. The code used for these case studies can be found at https://github.com/birdepy/birdepy_examples.

4.1. Chatham Island black robins

As reported by Butler and Merton (1992), the Chatham Island black robin (*Petroica traversi*) population started to come under threat approximately 450–500 years ago when humans arrived at the Chatham Islands. This was primarily due to the introduction of cats and rats.

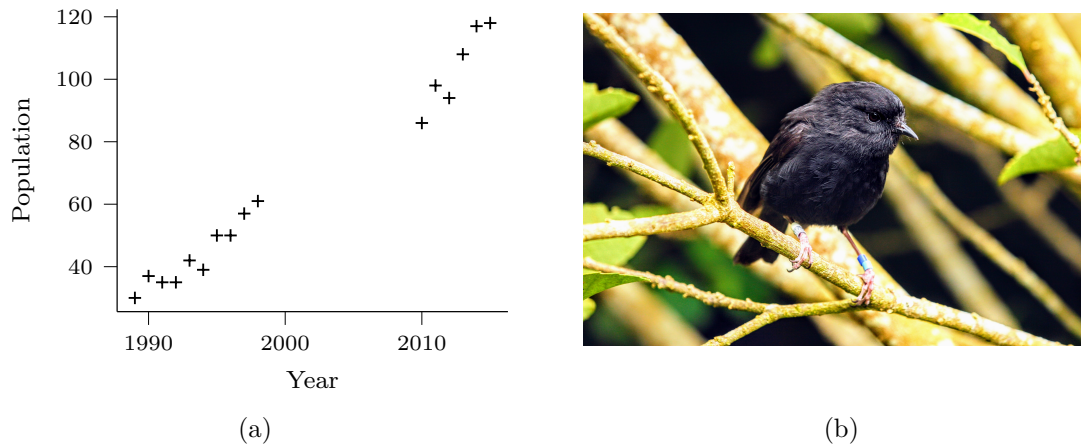


Figure 5: (a) Yearly population counts for the female black robins on Rangatira Island, and (b) a black robin on Rangatira Island (photo by Melanie Massaro).

By 1893, the species had been completely wiped out on the main island of the Chatham Islands, with the remaining 20–35 surviving birds found on Little Mangere Island. This population persisted for the following nine decades (1893–1976), but experienced a sharp fall in the period 1972–1976. The only surviving seven birds, which included just one breeding female, were relocated to a safer habitat on Mangere Island in 1976. Out of the five birds remaining in 1979, only one breeding pair produced offspring that bred successfully (Kennedy Euan 2009). A period of intensive management followed until the spring of 1990. As part of this management, a second population of black robins was established on Rangatira Island. This isolated population has not been subject to management, but has still been observed sporadically over the last few decades. Figure 5a displays annual counts of the females in this population for 1989–1998 and 2010–2015 (Massaro, Stanbury, and Briskie 2013; Davison *et al.* 2021). The gap in observation between 1999 and 2009 is due to a lack of government funding for this period (Massaro *et al.* 2013). Note that `bd.estimate()` does not require discretely-observed population counts to be evenly spaced, so this gap in observation is handled well by the function. Executing

```
t_data_b = [1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998,
            2010, 2011, 2012, 2013, 2014, 2015]
p_data_b = [30, 37, 35, 35, 42, 39, 50, 50, 57, 61, 86, 98, 94, 108, 117,
            118]
```

is a simple way to input this data into Python, although other methods where the data is first stored in a spreadsheet or other database are also possible.

Deep-forest interiors and dense forest-edges are known to provide suitable habitats for black robins to successfully breed in. Furthermore, since black robins are territorial, each breeding pair typically occupies a habitat patch that cannot be shared with other breeding pairs. When the availability of deep-forest interiors and dense forest-edges becomes limited, black robins are not particularly suited to expanding their territory into shrubland or scattered vegetation (Kennedy Euan 2009). This type of limitation on the capacity of a population to expand is a pervasive occurrence in ecology (e.g., Brook, O’Grady, Chapman, Burgman, Akcakaya, and Frankham 2000; Ford 2002; Hilderbrand 2003).

Model	γ	ν	α or β	Capacity	Likelihood
Verhulst 1	0.3516 (0.1307)	0.2391 (0.1007)	0.0023 (0.0018)	138	9.84e-22
Verhulst 2	0.3018 (0.1051)	0.1860 (0.1036)	0.0046 (0.0059)	135	9.57e-22
Ricker	0.3587 (0.1365)	0.2380 (0.1002)	0.0029 (0.0027)	142	9.99e-22
BH	0.3690 (0.1475)	0.2367 (0.0998)	0.0038 (0.0045)	146	1.02e-21
Hassell	0.3687 (0.1446)	0.2418 (0.1034)	0.0016 (0.0017)	143	1.01e-21
MSS	0.3283 (0.1190)	0.2413 (0.1019)	0.0045 (0.0026)	133	9.27e-22
linear	0.2845 (0.0957)	0.2350 (0.0956)	-	-	5.59e-22

Table 9: Parameter estimates and standard errors (in brackets), along with their corresponding carrying capacity values and likelihoods, as determined by `bd.estimate()` using the black robin data displayed in Figure 5a.

The lack of suitable habitat frequently slows the growth of populations of endangered species. Resources are restricted in remaining habitat patches, and therefore a population can grow only until it reaches the maximum population size a particular habitat can support, which is a biological definition of the carrying capacity. The knowledge of the carrying capacity can help policy makers to set realistic goals for population expansion within a particular habitat, and to devise conservation strategies.

Using **BirDePy** we are able to fit the data displayed in Figure 5a to the models in Table 1 and, consequently, to find estimates of the carrying capacity of Rangatira Island. For PSDBDPs, the carrying capacity corresponds to the closest integer to $z^* > 0$ such that $\lambda_{z^*} = \mu_{z^*}$. As an example, for a Ricker model $z^* = \frac{1}{\alpha} (\log(\gamma/\nu))^{1/c}$. The output of `bd.estimate()` stores an estimate of the carrying capacity in the attribute `capacity`.

In our analysis, we consider two Verhulst models, a Ricker model, a BH model, a Hassell model, an MSS model, and the linear model. In particular, our first Verhulst model attributes any decline in population growth to restrictions on the birth rate (by assuming $\beta = 0$), while our second Verhulst model assumes that an increase in mortality is the cause (by assuming $\alpha = 0$). Given the limited available data, we found that `bd.estimate()` performed better when only three parameters needed to be estimated. We therefore set $c = 1$ for the Ricker model, and $c = 2$ for the MSS and Hassell models. We do not consider a linear-migration model for this population since it is known that black robins do not migrate between the Chatham Islands.

To perform estimation for Verhulst 1, we execute

```
est_v1 = bd.estimate(t_data_b, p_data_b, p0 = [2, 2, 0.05],
  p_bounds = [[0,10], [0,10], [0, 1]], model = "Verhulst", known_p = [0],
  idx_known_p = [3], opt_method = "differential-evolution", seed = 2021)
```

and then parameter estimates, standard errors, the carrying capacity estimate, and the likelihood are stored, respectively, in `est_v1.p`, `est_v1.se`, `est_v1.capacity`, and `np.exp(est_v1.val)`. Observe that we initialize with a parameter guess of `[2, 2, 0.05]` and instate parameter bounds `[[0, 10], [0, 10], [0,1]]`. We also set `opt_method` to `"differential-evolution"`, as this produces more reliable estimates (at the cost of a long computation time). We also execute similar code to obtain results for the other models (i.e., `"Ricker"`, `"Beverton-Holt"`, `"Hassel"`, `"MS-S"` and `"linear"`).

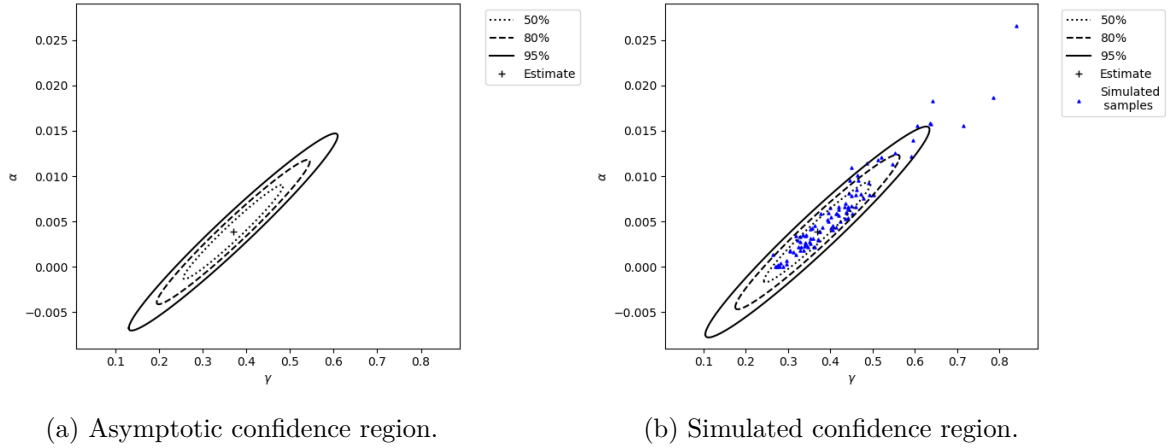


Figure 6: Asymptotic (left) and simulated (right) confidence regions for (γ, α) assuming $\nu = 0.2367$ for a BH model fitted to the black robin population data displayed in Figure 5a.

The results are presented in Table 9. The BH model has the highest likelihood ($1.02\text{e-}21$), and suggests that the population (of females) is subject to a carrying capacity of 146. The other PSDBDPs are not very different from the BH model in terms of likelihood ($9.27\text{e-}22$ to $1.01\text{e-}21$), and suggest a similar carrying capacity of 133 to 143. These figures are comparable to the upper limit on the carrying capacity of 170 breeding pairs that is reported by [Massaro, Chick, Kennedy, and Whitsed \(2018\)](#). It is worth highlighting that while our estimates are based solely on population size counts, the estimate in [Massaro *et al.* \(2018\)](#) requires precise knowledge of the habitat that is costly to acquire.

The black robin data we are using was previously fitted to a linear birth-and-death process in [Davison *et al.* \(2021\)](#). We see that the linear model has substantially lower likelihood ($5.59\text{e-}22$) than the PSDBDP models, which provides further evidence of the existence of a carrying capacity. Despite this, it should be noted that none of the PSDBDP models have α or β which is more than two standard errors from zero. This means that, despite the strong indication that a carrying capacity exists given by the higher likelihood of the PSDBDP models, there is not enough evidence in the data to reject a null hypothesis of $\alpha = \beta = 0$.

Obtaining confidence regions for the model parameters requires us to fix the value of one of the three parameters, since **BirDePy** cannot plot confidence regions consisting of more than two parameters. Here we assume that $\nu = 0.2367$, which is the BH model estimate for this parameter (comparable to the death rate estimate in [Hautphenne, Massaro, and Turner \(2019\)](#)). Then, executing

```
bd.estimate(t_data_b, p_data_b,
            p0 = [0.36, 0.0017], p_bounds = [[0,1], [0, 1]],
            model = 'Hassell', known_p = [0.2373, 1], idx_known_p = [1, 3],
            se_type = s, ci_plot = True, seed = 2021,
            xlabel = "$\\gamma$", ylabel = "$\\alpha$")
```

with `s = "asymptotic"`, and then again with `s = "simulated"`, results in the confidence regions for (γ, α) displayed in Figure 6, which show the likely range of values that (γ, α) take conditional on $\nu = 0.2367$. Note that here we have set an initial condition close to the

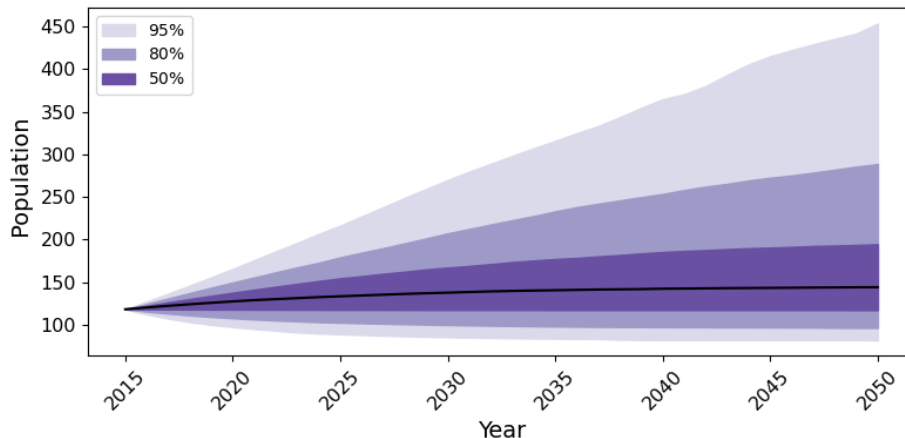


Figure 7: Confidence intervals for the expected future population of female black robins on Rangatira Island as generated by `bd.forecast()` assuming a BH model.

estimated parameter value and returned to the default optimization routine to speed up the computation of the simulated confidence region.

Figure 7 shows the effect of uncertainty in the value of (γ, ν, α) on the likely range of values taken by the expected future population size. We see that large increases in population size may occur, but the bulk of the projected population sizes are close to the carrying capacity estimates discussed earlier. Executing

```
bd.forecast(model = "Hassell", z0 = p_data_b[-1],
  times = np.arange(2015, 2051, 1), param = est_b.p[1:], cov = est_b.cov,
  p_bounds = [[0,10], [0,10], [0, 1]], known_p = [1], idx_known_p = [3])
```

produces this figure.

Finally we note that many PSDBDPs, including models with a carrying capacity, eventually become extinct with probability one (typically after a very long time), that is, $\mathbb{P}(\lim_{t \rightarrow \infty} Z(t) = 0) = 1$. Fitting a single population trajectory to these models (such as in the case of the black robins) then induces a small bias in the parameters estimates. We refer to [Braunsteins, Hautphenne, and Minuesa \(2021, 2022\)](#) for discussions and results on consistency and asymptotic normality of estimators for (discrete-time) Markov population processes with almost sure extinction.

4.2. Whooping cranes

According to [Allen \(1952\)](#), there were approximately 1300–1400 whooping cranes (*Grus americana*) in existence around 1860–1870. In the first part of the 20th century, hunting and a loss of natural habitat to agriculture pushed whooping cranes very close to extinction, with only 36 birds in 1912, declining down to 15 birds in 1941 ([Allen 1952](#)). The Migratory Bird Act of 1916, and the establishment of the WBNP in 1922 as the primary summer habitat and the ANWR in 1937 as the primary winter habitat, are thought to be major contributors to the survival of the species ([Luthy et al. 2005](#)). In 1967, whooping cranes were one of 75 species placed on the inaugural US endangered species list and put under federal protec-

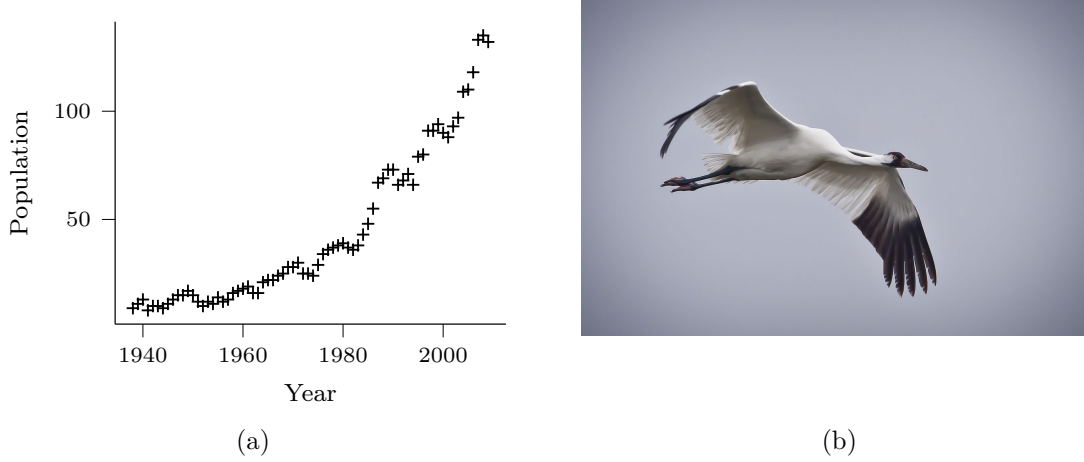


Figure 8: (a) Assumed yearly population counts for female whooping cranes, and (b) “Whooping Crane in flight in Texas” by U.S. Department of Agriculture is licensed under CC BY 2.0.

tion. The Whooping Crane Recovery Plan 2006 ([Canadian Wildlife Service and U.S. Fish and Wildlife Service 2007](#)) states that one way the species can be down-listed from ‘endangered’ to ‘threatened’ is for the ANWR flock to reach a self-sustaining population above 1000 individuals.

Estimates of the annual population in the ANWR flock for 1938–2009 are provided in [Butler, Harris, and Strobel \(2013\)](#). Following [Davison *et al.* \(2021\)](#), where a subset of this data is also analyzed, in Figure 8a we display estimates of the female population only, assuming a sex ratio of 1:1. Note that focusing on the female population eliminates dependencies between males and females due to reproduction, and makes birth-and-death models more suitable. Executing

```
t_data_w = [t for t in range(1938, 2010, 1)]
p_data_w = [9, 11, 13, 8, 10, 10, 9, 11, 13, 15, 15, 17, 15, 12, 10, 12,
11, 14, 12, 13, 16, 17, 18, 19, 16, 16, 21, 22, 22, 24, 25, 28, 28, 30,
25, 25, 24, 29, 34, 36, 37, 38, 39, 37, 36, 38, 43, 48, 55, 67, 69, 73,
73, 66, 68, 71, 66, 79, 80, 91, 91, 94, 90, 88, 93, 97, 109, 110, 118,
133, 135, 132]
```

inputs this data into `Python`. A large portion of the data was obtained using aerial surveys, as detailed by [Strobel and Butler \(2014\)](#).

Our analysis considers the same models as those described in Section 4.1 with the addition of a linear-migration model. Similar to Section 4.1, the **BirDePy** function `bd.estimate()` can be used to obtain parameter estimates, standard errors, carrying capacity estimates, and likelihoods. The results are shown in Table 10. The linear-migration model has the highest likelihood ($1.63\text{e-}81$), suggesting that the population is not subject to a carrying capacity and is growing exponentially. Interestingly, however, the PSDBDP models have likelihoods in the range $1.50\text{e-}81$ to $1.56\text{e-}81$, which are: (i) relatively close to the linear-migration likelihood, and (ii) greater than the linear model likelihood ($1.30\text{e-}81$), which is the lowest among the considered models. This means that if our assumption of possible external migration into the population does not hold (meaning $\alpha = 0$), then we would conversely conclude that the

Model	γ	ν	α or β	Capacity	Likelihood
Verhulst 1	0.1998 (0.0351)	0.1492 (0.0293)	0.0008 (0.0013)	330	1.51e-81
Verhulst 2	0.1931 (0.0303)	0.1423 (0.0321)	0.0011 (0.0021)	325	1.52e-81
Ricker	0.1999 (0.0354)	0.1493 (0.0293)	0.0008 (0.0015)	367	1.51e-81
Beverton-Holt	0.1999 (0.0357)	0.1493 (0.0293)	0.0008 (0.0016)	411	1.50e-81
Hassell	0.1999 (0.0356)	0.1493 (0.0293)	0.0004 (0.0008)	388	1.50e-81
MSS	0.1966 (0.0320)	0.1493 (0.0293)	0.0025 (0.0022)	223	1.56e-81
linear	0.1902 (0.0295)	0.1506 (0.0293)	-	-	1.30e-81
linear-migration	0.1812 (0.0317)	0.1489 (0.0294)	0.3157 (0.4769)	-	1.63e-81

Table 10: Parameter estimates and standard errors (in brackets), along with their corresponding carrying capacity values and likelihoods, as determined by `bd.estimate()` using the whooping crane population data displayed in Figure 8a.

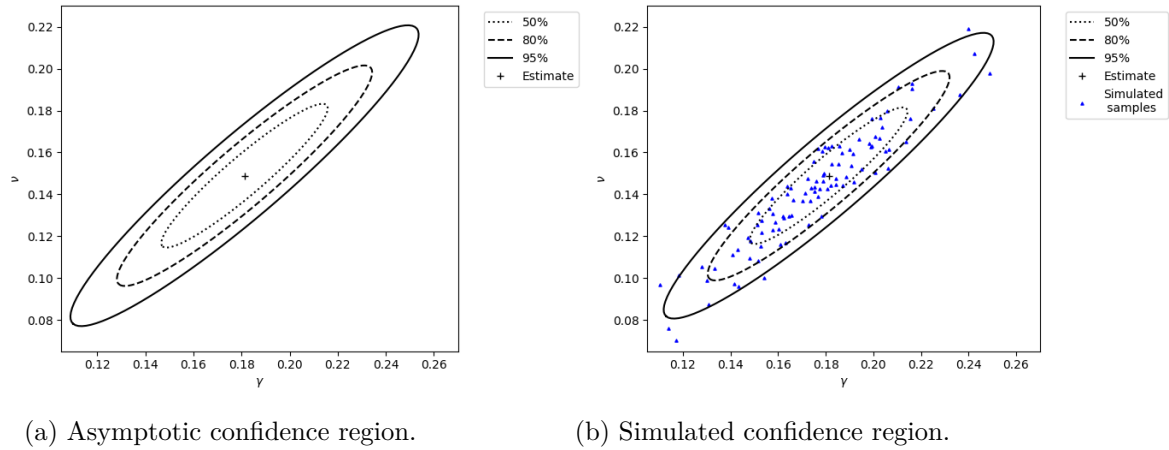
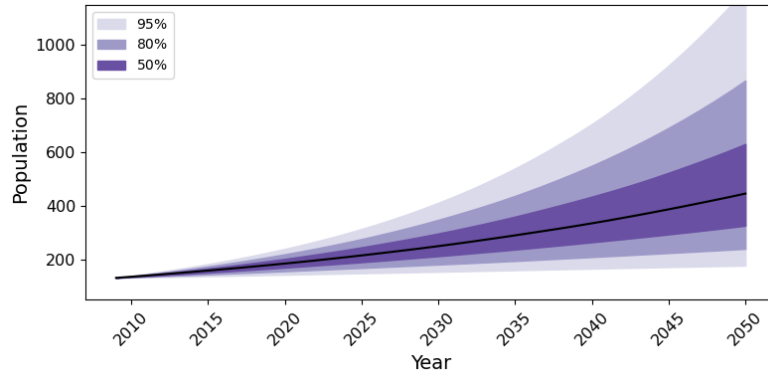


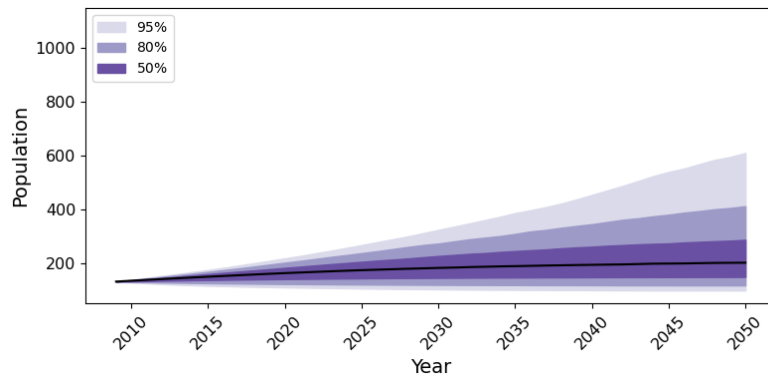
Figure 9: Asymptotic (left) and simulated (right) confidence regions for (γ, ν) assuming $\alpha = 0.3157$ for a linear-migration model fitted to the whooping crane data displayed in Figure 8a.

population is in fact subject to a carrying capacity. Therefore our assumption that external migration into the population is possible is crucial to determining whether a carrying capacity exists. In contrast to the black robin population, it is far less clear in this case whether the population is subject to a carrying capacity. The PSDBDP models suggest a carrying capacity of 223 females (likelihood 1.56e-81) to 411 females (likelihood 1.50e-81). This is in line with the analysis of [Stehn and Prieto \(2010\)](#), who estimate the current habitat can support up to 576 individual whooping cranes, with the possibility of up to 1156 if a nearby habitat is suitable for colonization and successfully colonized. Therefore our analysis suggests that there is cause for doubt about the ability of the population to achieve a threatened status by reaching a self-sustaining population of 1000 individuals in the ANWR flock.

The Verhulst 1 model (likelihood 1.51e-81) and the Verhulst 2 model (likelihood 1.52e-81) have highly similar likelihoods, so our analysis does not provide evidence in either direction as to whether a decline in the birth rate or an increase in the death rate is limiting population growth (assuming such a limit exists). Similar to Section 4.1, as displayed in Figure 9, we are able to generate asymptotic and simulated confidence regions for (γ, ν) in the linear-migration



(a) Linear-migration model.



(b) MSS model.

Figure 10: Confidence intervals for the mean future population of female whooping cranes as generated by `bd.forecast()` assuming (a) a linear-migration model, and (b) an MSS model.

model assuming $\alpha = 0.3157$. Figure 10 displays how this uncertainty and the choice of model affects the likely range of values that the expected future population size takes. Assuming a linear-migration model, more than 50% of values within the confidence interval exceed 500 by the year 2050, suggesting a long term outlook that may see whooping cranes removed from the endangered species list. On the other hand, if a MSS model is more appropriate, then the portion of values within the confidence interval exceeding 500 by the year 2050 is substantially less, suggesting whooping cranes will continue to be endangered for some time to come under present conditions. This highlights the fact that the choice of model is fundamental in order to draw biological conclusions, which cannot be made without the additional input of experts.

5. Concluding remarks and future work

BirDePy provides a collection of functions for working with PSDBDPs in Python. The ability of the package to approximate transition probabilities, to generate sample paths, to estimate parameters from discretely-observed population size data, and to forecast future population sizes has been illustrated with a variety of examples and through two pertinent case studies. We have demonstrated how this package can be of benefit to ecologists.

BirDePy is available from the Python Package Index <https://pypi.org/project/birdepy/> and is developed on GitHub <https://github.com/BirDePy>, where contributions in the form of pull requests are welcomed.

There are several models and features which would be natural for future versions of **BirDePy** to include:

- Currently, **BirDePy** is designed to work with PSDBDPs where individuals are homogeneous. However, many population processes involve individuals of multiple types. Key examples include epidemic models such as the Susceptible-Infectious-Recovered (SIR) model, queueing network models where individuals are classified by the server they are currently located at, and ecological models where individuals are classified according to their location or age. The package could be enhanced to accommodate these multi-type models.
- The transition rates of the PSDBDPs in **BirDePy** currently depend only on the population size. This could be expanded to a generalized framework that allows other covariates to influence birth and death rates. For example, if each transition could be associated with a vector of covariates, then the birth and death rates could be modeled as functions of regression coefficients in a generalized-linear-model framework. The birth and death functions would be augmented to depend on the covariate vector, in addition to the parameters they currently depend upon. Such a framework would provide insights into which covariates are important determinants of birth and death rates. A framework along these lines is discussed by Crawford *et al.* (2014).
- In our case studies we showed how the likelihood of the parametrized models and expert opinion could be combined to choose between the models in Table 1. It would be useful if a more sophisticated method of model selection could be developed and incorporated into **BirDePy**.
- Each of the parameter estimation and transition probability approximation methods implemented in **BirDePy** have strengths and weaknesses depending on the model, parameter values and population size at hand. More research needs to be conducted on when each method is optimal to use. This would be particularly useful since then an “ensemble” method could be developed that switches automatically between the methods we have already included.

Computational details

The numerical experiments in this paper were all performed in Python (v3.7.10) with **BirDePy** (v0.0.9) on a desktop computer running Windows 10 with an Intel i5-10400F CPU, Nvidia GTX 1070 GPU, and 16 gigabytes of random access memory. The runtimes might differ for higher Python versions and on other processors.

Acknowledgments

This research is funded by the Australian Government through the Australian Research Council (DP200101281). We thank Giovanni Ferron for setting up the CI/CD pipelines on GitHub, and two anonymous referees for their insightful and constructive feedback on our manuscript.

References

- Abate J, Choudhury GL, Whitt W (2000). “An Introduction to Numerical Transform Inversion and Its Application to Probability Models.” In *Computational Probability*, pp. 257–323. Springer-Verlag. doi:10.1007/978-1-4757-4828-4_8.
- Abate J, Whitt W (1995). “Numerical Inversion of Laplace Transforms of Probability Distributions.” *ORSA Journal on Computing*, **7**(1), 36–43. doi:10.1287/ijoc.7.1.36.
- Abate J, Whitt W (2006). “A Unified Framework for Numerically Inverting Laplace Transforms.” *INFORMS Journal on Computing*, **18**(4), 408–421. doi:10.1287/ijoc.1050.0137.
- Al-Mohy AH, Higham NJ (2010). “A New Scaling and Squaring Algorithm for the Matrix Exponential.” *SIAM Journal on Matrix Analysis and Applications*, **31**(3), 970–989. doi:10.1137/09074721x.
- Allen LJS (2008). “An Introduction to Stochastic Epidemic Models.” In *Mathematical Epidemiology*, pp. 81–130. Springer-Verlag. doi:10.1007/978-3-540-78911-6_3.
- Allen RP (1952). “The Whooping Crane.” *Technical report*, National Audubon Society.
- Anderson DF (2008). “Incorporating Postleap Checks in Tau-Leaping.” *The Journal of Chemical Physics*, **128**(5), 054103. doi:10.1063/1.2819665.
- Anderson DF, Ganguly A, Kurtz TG (2011). “Error Analysis of Tau-Leap Simulation Methods.” *The Annals of Applied Probability*, **21**(6), 2226–2262. doi:10.1214/10-aap756.
- Asanjarani A, Nazarathy Y, Taylor P (2021). “A Survey of Parameter and State Estimation in Queues.” *Queueing Systems*, **97**(1), 39–80. doi:10.1007/s11134-021-09688-w.
- Asmussen S, Avram F, Usabel M (2002). “Erlangian Approximations for Finite-Horizon Ruin Probabilities.” *ASTIN Bulletin: The Journal of the IAA*, **32**(2), 267–281. doi:10.2143/ast.32.2.1029.
- Beaumont MA, Cornuet JM, Marin JM, Robert CP (2009). “Adaptive Approximate Bayesian Computation.” *Biometrika*, **96**(4), 983–990. doi:10.1093/biomet/asp052.
- Bellows T (1981). “The Descriptive Properties of Some Models for Density Dependence.” *The Journal of Animal Ecology*, pp. 139–156. doi:10.2307/4037.
- Bladt M, Sørensen M (2005). “Statistical Inference for Discretely Observed Markov Jump Processes.” *Journal of the Royal Statistical Society B*, **67**(3), 395–410. doi:10.1111/j.1467-9868.2005.00508.x.
- Boucherie RJ, Van Dijk NM (2010). *Queueing Networks: A Fundamental Approach*, volume 154. Springer-Verlag. doi:10.1007/978-1-4419-6472-4.
- Braunsteins P, Hautphenne S, Minuesa C (2021). “Parameter Estimation in Branching Processes with Almost Sure Extinction.” *Bernoulli*, **28**(1), 33–63. doi:10.3150/21-bej1332.

- Braunsteins P, Hautphenne S, Minuesa C (2022). “Consistent Least Squares Estimation in Population-Size-Dependent Branching Processes.” *arXiv 2211.10898*, arXiv.org E-Print Archive. doi:10.48550/arXiv.2211.10898.
- Brockmeyer E, Halstrom HL, Jensen A (1948). *The Life and Works of A.K. Erlang*. Academy of Technical Sciences, Copenhagen.
- Brook BW, O’Grady JJ, Chapman AP, Burgman MA, Akcakaya HR, Frankham R (2000). “Predictive Accuracy of Population Viability Analysis in Conservation Biology.” *Nature*, **404**(6776), 385–387. doi:10.1038/35006050.
- Buckingham-Jeffery E, Isham V, House T (2018). “Gaussian Process Approximations for Fast Inference from Infectious Disease Data.” *Mathematical Biosciences*, **301**, 111–120. doi:10.1016/j.mbs.2018.02.003.
- Butler D, Merton D (1992). *The Black Robin: Saving the World’s Most Endangered Bird*. Oxford University Press.
- Butler MJ, Harris G, Strobel BN (2013). “Influence of Whooping Crane Population Dynamics on Its Recovery and Management.” *Biological Conservation*, **162**, 89–99. doi:10.1016/j.biocon.2013.04.003.
- Butler RW (2007). *Saddlepoint Approximations with Applications*. Cambridge University Press. doi:10.1017/cbo9780511619083.
- Canadian Wildlife Service, US Fish and Wildlife Service (2007). “International Recovery Plan for the Whooping Crane.” *Technical report*, Recovery of Nationally Endangered Wildlife (RENEW), and U.S. Fish and Wildlife Service.
- Cao Y, Gillespie DT, Petzold LR (2005). “Avoiding Negative Populations in Explicit Poisson Tau-Leaping.” *The Journal of Chemical Physics*, **123**(5), 054104. doi:10.1063/1.1992473.
- Cao Y, Gillespie DT, Petzold LR (2006). “Efficient Step Size Selection for the Tau-Leaping Simulation Method.” *The Journal of Chemical Physics*, **124**(4), 044109. doi:10.1063/1.2159468.
- Chatterjee A, Vlachos DG, Katsoulakis MA (2005). “Binomial Distribution Based τ -Leap Accelerated Stochastic Simulation.” *The Journal of Chemical Physics*, **122**(2), 024112. doi:10.1063/1.1833357.
- Cinlar E (2013). *Introduction to Stochastic Processes*. Courier Corporation.
- Crawford FW, Ho LST, Suchard MA (2018). “Computational Methods for Birth-Death Processes.” *Wiley Interdisciplinary Reviews: Computational Statistics*, **10**(2), e1423. doi:10.1002/wics.1423.
- Crawford FW, Minin VN, Suchard MA (2014). “Estimation for General Birth-Death Processes.” *Journal of the American Statistical Association*, **109**(506), 730–747. doi:10.1080/01621459.2013.866565.
- Crawford FW, Suchard MA (2012). “Transition Probabilities for General Birth-Death Processes with Applications in Ecology, Genetics, and Evolution.” *Journal of Mathematical Biology*, **65**(3), 553–580. doi:10.1007/s00285-011-0471-z.

- Davies B, Martin B (1979). “Numerical Inversion of the Laplace Transform: A Survey and Comparison of Methods.” *Journal of Computational Physics*, **33**(1), 1–32. doi:[10.1016/0021-9991\(79\)90025-1](https://doi.org/10.1016/0021-9991(79)90025-1).
- Davison AC, Hautphenne S, Kraus A (2021). “Parameter Estimation for Discretely Observed Linear Birth-and-Death Processes.” *Biometrics*, **77**(1), 186–196. doi:[10.1111/biom.13282](https://doi.org/10.1111/biom.13282).
- De Hoog FR, Knight JH, Stokes AN (1982). “An Improved Method for Numerical Inversion of Laplace Transforms.” *SIAM Journal on Scientific and Statistical Computing*, **3**(3), 357–366. doi:[10.1137/0903022](https://doi.org/10.1137/0903022).
- 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. doi:[10.1007/s11222-011-9271-y](https://doi.org/10.1007/s11222-011-9271-y).
- Dempster AP, Laird NM, Rubin DB (1977). “Maximum Likelihood from Incomplete Data via the EM Algorithm.” *Journal of the Royal Statistical Society B*, **39**(1), 1–22. doi:[10.1111/j.2517-6161.1977.tb01600.x](https://doi.org/10.1111/j.2517-6161.1977.tb01600.x).
- Doss C, Minin V, Suchard M (2017). **DOBAD**: *Analysis of Discretely Observed Linear Birth-and-Death(-and-Immigration) Markov Chains*. R package version 1.0.6, URL <https://CRAN.R-project.org/package=DOBAD>.
- Drawert B, Hellander A, Bales B, Banerjee D, Bellesia G, Daigle Jr BJ, Douglas G, Gu M, Gupta A, Hellander S, *et al.* (2016). “Stochastic Simulation Service: Bridging the Gap Between the Computational Expert and the Biologist.” *PLoS Computational Biology*, **12**(12), e1005220. doi:[10.1371/journal.pcbi.1005220](https://doi.org/10.1371/journal.pcbi.1005220).
- Feller W (1957). *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons.
- Ford MJ (2002). “Selection in Captivity During Supportive Breeding May Reduce Fitness in the Wild.” *Conservation Biology*, **16**(3), 815–825. doi:[10.1046/j.1523-1739.2002.00257.x](https://doi.org/10.1046/j.1523-1739.2002.00257.x).
- Fulford DS (2020). **gwr-Inversion**, Version 1.0.1. Houston. URL <https://github.com/petbox-dev/gwr>.
- Gibbens RJ, Hunt PJ, Kelly FP (1990). “Bistability in Communication Networks.” *Disorder in Physical Systems*, pp. 113–128. doi:[10.1093/imamci/7.1.77](https://doi.org/10.1093/imamci/7.1.77).
- Gillespie DT (1977). “Exact Stochastic Simulation of Coupled Chemical Reactions.” *The Journal of Physical Chemistry*, **81**(25), 2340–2361. doi:[10.1021/j100540a008](https://doi.org/10.1021/j100540a008).
- Gillespie DT (2001). “Approximate Accelerated Stochastic Simulation of Chemically Reacting Systems.” *The Journal of Chemical Physics*, **115**(4), 1716–1733. doi:[10.1063/1.1378322](https://doi.org/10.1063/1.1378322).
- Gillespie DT, Petzold LR (2003). “Improved Leap-Size Selection for Accelerated Stochastic Simulation.” *The Journal of Chemical Physics*, **119**(16), 8229–8234. doi:[10.1063/1.1613254](https://doi.org/10.1063/1.1613254).

- Grassmann WK (1977). “Transient Solutions in Markovian Queueing Systems.” *Computers & Operations Research*, **4**(1), 47–53. doi:10.1016/0305-0548(77)90007-7.
- Grimmett G, Stirzaker D (2020). *Probability and Random Processes*. Oxford university press. doi:10.2307/3621637.
- Gross D, Miller DR (1984). “The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Processes.” *Operations Research*, **32**(2), 343–361. doi:10.1287/opre.32.2.343.
- Guttorp P (1991). *Statistical Inference for Branching Processes*, volume 122. John Wiley & Sons.
- Harris CR, Millman KJ, Van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, van Kerkwijk MH, Brett M, Haldane A, del Río JF, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE (2020). “Array Programming with NumPy.” *Nature*, **585**(7825), 357–362. doi:10.1038/s41586-020-2649-2.
- Harris TE (1963). *The Theory of Branching Processes*, volume 6. Springer-Verlag. doi:10.1007/978-3-642-51866-9.
- Hautphenne S, Massaro M, Turner K (2019). “Fitting Markovian Binary Trees Using Global and Individual Demographic Data.” *Theoretical Population Biology*, **128**, 39–50. doi:10.1016/j.tpb.2019.04.007.
- Hautphenne S, Patch B (2021). “Simulating Population-Size-Dependent Birth-and-Death Processes Using CUDA and Piecewise Approximations.” In *Proceedings of the International Congress on Modelling and Simulation*. doi:10.36334/modsim.2021.a1.hautphenne.
- Hautphenne S, Patch B (2023). *BirDePy: Birth-and-Death Processes in Python*. Version 0.0.26, URL <https://birdepy.github.io/>.
- Hilderbrand RH (2003). “The Roles of Carrying Capacity, Immigration, and Population Synchrony on Persistence of Stream-Resident Cutthroat Trout.” *Biological Conservation*, **110**(2), 257–266. doi:10.1016/s0006-3207(02)00224-0.
- Holmes I, Rubin GM (2002). “An Expectation Maximization Algorithm for Training Hidden Substitution Models.” *Journal of Molecular Biology*, **317**(5), 753–764. doi:10.1006/jmbi.2002.5405.
- Horváth G, Horváth I, Almousa SAD, Telek M (2019). “Inverse Laplace Transform with Concentrated Matrix-Exponential Functions.” URL <https://inverselaplace.org/>.
- Horváth G, Horváth I, Almousa SAD, Telek M (2020). “Numerical Inverse Laplace Transformation Using Concentrated Matrix Exponential Distributions.” *Performance Evaluation*, **137**, 102067. doi:10.1016/j.peva.2019.102067.
- Hunter JD (2007). “Matplotlib: A 2D Graphics Environment.” *Computing in Science & Engineering*, **9**(3), 90–95. doi:10.1109/mcse.2007.55.

- Jamshidian M, Jennrich RI (1993). “Conjugate Gradient Acceleration of the EM Algorithm.” *Journal of the American Statistical Association*, **88**(421), 221–228. doi:10.2307/2290716.
- Jamshidian M, Jennrich RI (1997). “Acceleration of the EM Algorithm by Using Quasi-Newton Methods.” *Journal of the Royal Statistical Society B*, **59**(3), 569–587. doi:10.1111/1467-9868.00083.
- Janzen T, Höhna S, Etienne RS (2015). “Approximate Bayesian Computation of Diversification Rates from Molecular Phylogenies: Introducing a New Efficient Summary Statistic, the nLTT.” *Methods in Ecology and Evolution*, **6**(5), 566–575. doi:10.1111/2041-210x.12350.
- Jemmer P, McNamee J (2005). “From the Discrete to the Continuous: Relationships and Results for Single-Species Population Models.” *Mathematical and Computer Modelling*, **41**(1), 71–98. doi:10.1016/j.mcm.2003.10.052.
- Jensen A (1953). “Markoff Chains as an Aid in the Study of Markoff Processes.” *Scandinavian Actuarial Journal*, **1953**(sup1), 87–91. doi:10.1080/03461238.1953.10419459.
- Johansson F, et al. (2013). *mpmath: A Python Library for Arbitrary-Precision Floating-Point Arithmetic (Version 0.18)*. URL <http://mpmath.org/>.
- Karev GP, Wolf YI, Berezovskaya FS, Koonin EV (2004). “Gene Family Evolution: An In-Depth Theoretical and Simulation Analysis of Non-Linear Birth-Death-Innovation Models.” *BMC Evolutionary Biology*, **4**(1), 1–23. doi:10.1186/1471-2148-4-32.
- Karev GP, Wolf YI, Koonin EV (2003). “Simple Stochastic Birth and Death Models of Genome Evolution: Was There Enough Time for Us to Evolve?” *Bioinformatics*, **19**(15), 1889–1900. doi:10.1093/bioinformatics/btg351.
- Karlin S, Taylor HM (1975). *A First Course in Stochastic Processes*, volume 1. Academic press.
- Karlin S, Taylor HM (1981). *A Second Course in Stochastic Processes*. Elsevier.
- Kendall DG (1950). “An Artificial Realization of a Simple “Birth-and-Death” Process.” *Journal of the Royal Statistical Society B*, **12**(1), 116–119. doi:10.1111/j.2517-6161.1950.tb00048.x.
- Kennedy Euan S (2009). *Extinction Vulnerability in Two Small, Chronically Inbred Populations of Chatham Island Black Robin Petroica Traversi*. Ph.D. thesis, Lincoln University.
- King AA, Nguyen D, Ionides EL (2016). “Statistical Inference for Partially Observed Markov Processes via the R Package **Pomp**.” *Journal of Statistical Software*, **69**(1), 1–43. doi:10.18637/jss.v069.i12.
- Kleinrock L (1975). *Queueing Systems, Volume I: Theory*. John Wiley & Sons.
- Krekel H (2023). *pytest, Version 7.3.1*. URL <https://docs.pytest.org/en/7.3.x/>.
- Kuhlman KL (2013). “Review of Inverse Laplace Transform Algorithms for Laplace-Space Numerical Approaches.” *Numerical Algorithms*, **63**(2), 339–355. doi:10.1007/s11075-012-9625-3.

- Kurtz TG (1971). “Limit Theorems for Sequences of Jump Markov Processes.” *Journal of Applied Probability*, **8**(2), 344–356. doi:[10.1017/s002190020003535x](https://doi.org/10.1017/s002190020003535x).
- Lam SK, Pitrou A, Seibert S (2015). “Numba: A LLVM-Based Python JIT Compiler.” In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6. doi:[10.1145/2833157.2833162](https://doi.org/10.1145/2833157.2833162).
- Lange K (1995a). “A Gradient Algorithm Locally Equivalent to the EM Algorithm.” *Journal of the Royal Statistical Society B*, **57**(2), 425–437. doi:[10.1111/j.2517-6161.1995.tb02037.x](https://doi.org/10.1111/j.2517-6161.1995.tb02037.x).
- Lange K (1995b). “A Quasi-Newton Acceleration of the EM Algorithm.” *Statistica Sinica*, pp. 1–18. doi:[10.3923/itj.2006.749.752](https://doi.org/10.3923/itj.2006.749.752).
- Luthy RG, Rose JB, Allen-King RM, Baecher GB, Bradbury KR, Crook J, Foufoula-Georgiou E, Gleick P, Letey, Jr J, Moe CL, Perciasepe R, Schnoor JL, Shabman L, Trussell RR, Turekian KK, Watt HM, Wescoat JLJ (2005). *Endangered and Threatened Species of the Platte River*. National Academies Press. doi:[10.17226/10978](https://doi.org/10.17226/10978).
- Mandjes M, Sollie B (2021). “A Numerical Approach for Evaluating the Time-Dependent Distribution of a Quasi Birth-Death Process.” *Methodology and Computing in Applied Probability*, pp. 1–23. doi:[10.1007/s11009-021-09882-6](https://doi.org/10.1007/s11009-021-09882-6).
- Mandjes M, Taylor P (2016). “The Running Maximum of a Level-Dependent Quasi-Birth-Death Process.” *Probability in the Engineering and Informational Sciences*, **30**(2), 212–223. doi:[10.1017/s026996481500039x](https://doi.org/10.1017/s026996481500039x).
- Martynus G, Vanduyndslager P, Travi M (2023). **semantic-Release**, Version 21.0.1. URL <https://github.com/semantic-release/semantic-release>.
- Massaro M, Chick A, Kennedy ES, Whitsed R (2018). “Post-Reintroduction Distribution and Habitat Preferences of a Spatially Limited Island Bird Species.” *Animal Conservation*, **21**(1), 54–64. doi:[10.1111/acv.12364](https://doi.org/10.1111/acv.12364).
- Massaro M, Stanbury M, Briskie J (2013). “Nest Site Selection by the Endangered Black Robin Increases Vulnerability to Predation by an Invasive Bird.” *Animal Conservation*, **16**(4), 404–411. doi:[10.1111/acv.12007](https://doi.org/10.1111/acv.12007).
- Melamed B, Yadin M (1984). “Randomization Procedures in the Computation of Cumulative-Time Distributions over Discrete State Markov Processes.” *Operations Research*, **32**(4), 926–944. doi:[10.1287/opre.32.4.926](https://doi.org/10.1287/opre.32.4.926).
- Moler C, Van Loan C (2003). “Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later.” *SIAM Review*, **45**(1), 3–49. doi:[10.1137/s00361445024180](https://doi.org/10.1137/s00361445024180).
- Murphy JA, O’Donohoe MR (1975). “Some Properties of Continued Fractions with Applications to Markov Processes.” *IMA Journal of Applied Mathematics*, **16**(1), 57–71. doi:[10.1093/imamat/16.1.57](https://doi.org/10.1093/imamat/16.1.57).
- Nåsell I (2001). “Extinction and Quasi-Stationarity in the Verhulst Logistic Model.” *Journal of Theoretical Biology*, **211**(1), 11–27. doi:[10.1006/jtbi.2001.2328](https://doi.org/10.1006/jtbi.2001.2328).

- Nåsell I (2002). “Stochastic Models of Some Endemic Infections.” *Mathematical Biosciences*, **179**(1), 1–19. doi:[10.1016/s0025-5564\(02\)00098-6](https://doi.org/10.1016/s0025-5564(02)00098-6).
- Novozhilov AS, Karev GP, Koonin EV (2006). “Biological Applications of the Theory of Birth-and-Death Processes.” *Briefings in Bioinformatics*, **7**(1), 70–85. doi:[10.1093/bib/bbk006](https://doi.org/10.1093/bib/bbk006).
- Nowak MA (2006). *Evolutionary Dynamics: Exploring the Equations of Life*. Harvard University Press.
- Nvidia (2018). *cudatoolkit, Version 9.2*. Santa Clara. URL <https://docs.nvidia.com/cuda/archive/9.2/>.
- Press WH, William H, Teukolsky SA, Saul A, Vetterling WT, Flannery BP (2007). *Numerical Recipes: The Art of Scientific Computing*. 3rd edition. Cambridge University Press.
- Pritchard JK, Seielstad MT, Perez-Lezaun A, Feldman MW (1999). “Population Growth of Human Y Chromosomes: A Study of Y Chromosome Microsatellites.” *Molecular Biology and Evolution*, **16**(12), 1791–1798. doi:[10.1093/oxfordjournals.molbev.a026091](https://doi.org/10.1093/oxfordjournals.molbev.a026091).
- R Core Team (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. URL <https://www.R-project.org/>.
- Reynolds JF (1973). “On Estimating the Parameters of a Birth-Death Process.” *Australian Journal of Statistics*, **15**(1), 35–43. doi:[10.1111/j.1467-842x.1973.tb00120.x](https://doi.org/10.1111/j.1467-842x.1973.tb00120.x).
- Roney JP, Ferlic J, Michor F, McDonald TO (2020). “**ESTIpop**: A Computational Tool to Simulate and Estimate Parameters for Continuous-Time Markov Branching Processes.” *Bioinformatics*, **36**(15), 4372–4373. doi:[10.1093/bioinformatics/btaa526](https://doi.org/10.1093/bioinformatics/btaa526).
- Ross JV, Pagendam DE, Pollett PK (2009). “On Parameter Estimation in Population Models II: Multi-Dimensional Processes and Transient Dynamics.” *Theoretical Population Biology*, **75**(2-3), 123–132. doi:[10.1016/j.tpb.2008.12.002](https://doi.org/10.1016/j.tpb.2008.12.002).
- Ross JV, Taimre T, Pollett PK (2006). “On Parameter Estimation in Population Models.” *Theoretical Population Biology*, **70**(4), 498–510. doi:[10.1002/9780470757468.ch2](https://doi.org/10.1002/9780470757468.ch2).
- Ross JV, Taimre T, Pollett PK (2007). “Estimation for Queues from Queue Length Data.” *Queueing Systems*, **55**(2), 131–138. doi:[10.1007/s11134-006-9009-2](https://doi.org/10.1007/s11134-006-9009-2).
- Sherlock C (2021). “Direct Statistical Inference for Finite Markov Jump Processes via the Matrix Exponential.” *Computational Statistics*, pp. 1–25. doi:[10.1007/s00180-021-01102-6](https://doi.org/10.1007/s00180-021-01102-6).
- Simola U, Cisewski-Kehe J, Gutmann MU, Corander J (2021). “Adaptive Approximate Bayesian Computation Tolerance Selection.” *Bayesian Analysis*, **16**(2), 397–423. doi:[10.1214/20-ba1211](https://doi.org/10.1214/20-ba1211).
- Singh P, Wrede F, Hellander A (2021). “Scalable Machine Learning-Assisted Model Exploration and Inference Using **Sciope**.” *Bioinformatics*, **37**(2), 279–281. doi:[10.1093/bioinformatics/btaa673](https://doi.org/10.1093/bioinformatics/btaa673).
- Stanford DA, Yu K, Ren J (2011). “Erlangian Approximation to Finite Time Ruin Probabilities in Perturbed Risk Models.” *Scandinavian Actuarial Journal*, **2011**(1), 38–58. doi:[10.1080/03461230903421492](https://doi.org/10.1080/03461230903421492).

- Stehfest H (1970). “Algorithm 368: Numerical Inversion of Laplace Transforms [D5].” *Communications of the ACM*, **13**(1), 47–49. doi:[10.1145/361953.361969](https://doi.org/10.1145/361953.361969).
- Stehn TV, Prieto F (2010). “Changes in Winter Whooping Crane Territories and Range 1950–2006.” In *Proceedings of Logistics Operations Management (GOL)*. IEEE.
- Strobel BN, Butler MJ (2014). “Monitoring Whooping Crane Abundance Using Aerial Surveys: Influences on Detectability.” *Wildlife Society Bulletin*, **38**(1), 188–195. doi:[10.1002/wsb.374](https://doi.org/10.1002/wsb.374).
- Stroustrup B (2000). *The C++ Programming Language*. Pearson Education India.
- Subbey S, Devine JA, Schaarschmidt U, Nash RD (2014). “Modelling and Forecasting Stock-Recruitment: Current and Future Perspectives.” *ICES Journal of Marine Science*, **71**(8), 2307–2322. doi:[10.1093/icesjms/fsu148](https://doi.org/10.1093/icesjms/fsu148).
- Talbot A (1979). “The Accurate Numerical Inversion of Laplace Transforms.” *IMA Journal of Applied Mathematics*, **23**(1), 97–120. doi:[10.1093/imamat/23.1.97](https://doi.org/10.1093/imamat/23.1.97).
- Tavaré S (2018). “The Linear Birth–Death Process: An Inferential Retrospective.” *Advances in Applied Probability*, **50**(A), 253–269. doi:[10.1017/apr.2018.84](https://doi.org/10.1017/apr.2018.84).
- Valkó PP, Abate J (2004). “Comparison of Sequence Accelerators for the Gaver Method of Numerical Laplace Transform Inversion.” *Computers & Mathematics with Applications*, **48**(3–4), 629–636. doi:[10.1016/j.camwa.2002.10.017](https://doi.org/10.1016/j.camwa.2002.10.017).
- Van Dijk NM, Van Brummelen SPJ, Boucherie RJ (2018). “Uniformization: Basics, Extensions and Applications.” *Performance Evaluation*, **118**, 8–32. doi:[10.1016/j.peva.2017.09.008](https://doi.org/10.1016/j.peva.2017.09.008).
- Van Loan C (1978). “Computing Integrals Involving the Matrix Exponential.” *IEEE Transactions on Automatic Control*, **23**(3), 395–404. doi:[10.1109/tac.1978.1101743](https://doi.org/10.1109/tac.1978.1101743).
- Van Rossum G, Drake Jr FL (1995). *Python Tutorial*, volume 620. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.
- Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, *et al.* (2020). “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.” *Nature Methods*, **17**(3), 261–272. doi:[10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
- Wang Q, Zhan H (2015). “On Different Numerical Inverse Laplace Methods for Solute Transport Problems.” *Advances in Water Resources*, **75**, 80–92. doi:[10.1016/j.advwatres.2014.11.001](https://doi.org/10.1016/j.advwatres.2014.11.001).
- Whitt W (2002). *Stochastic-Process Limits: An Introduction to Stochastic-Process Limits and Their Application to Queues*. Springer-Verlag. doi:[10.1007/b97479](https://doi.org/10.1007/b97479).
- Wolff RW (1965). “Problems of Statistical Inference for Birth and Death Queuing Models.” *Operations Research*, **13**(3), 343–357. doi:[10.1287/opre.13.3.343](https://doi.org/10.1287/opre.13.3.343).

A. BirDePy description

In this appendix we describe how to use **BirDePy**. As is usual for Python we use a shorthand `bd` in place of the full package name `birdepy`. The package has five core functions: `bd.simulate.continuous()`, `bd.simulate.discrete()`, `bd.probability()`, `bd.estimate()` and `bd.forecast()`. In addition the two functions `bd.gpu_functions.probability()` and `bd.gpu_functions.discrete()` are available when the system has a CUDA¹-enabled graphics processing unit (GPU) with compute capability 2.0 or above with an up-to-date Nvidia driver. These CUDA based functions provide a limited version of the functionality in `bd.probability()` and `bd.simulate.discrete()` but are often capable of providing output substantially faster.

BirDePy can be installed by opening Python (v3.7) and executing `pip install birdepy`. It can then be imported to a session using `import birdepy as bd`.

In order to use the core functions of **BirDePy** the packages **NumPy** \geq v1.17.0 (see [Harris et al. 2020](#)), **mpmath** v1.1.0 (see [Johansson et al. 2013](#)), **SciPy** v1.7.0 (see [Virtanen et al. 2020](#)), **matplotlib** v3.4.2 (see [Hunter 2007](#)) and **gwr-inversion** (see [Fulford 2020](#)) need to be installed. The CUDA based functions contained in the module `bd.gpu_functions` additionally require **Numba** v0.53.1 (see [Lam, Pitrou, and Seibert 2015](#)) and **cuda-toolkit** v9.2 (see [Nvidia 2018](#)) to be installed.

In all instances parameters are passed to functions as a list in the canonical order they are listed in Table 1.

A.1. Continuous simulation: `bd.simulate.continuous()`

This function is used to simulate PSDBDPs at birth and death event times.

Usage and default argument values

```
bd.simulate.continuous(param, model, z0, t_max, k = 1, survival = False,
    seed = None, **options)
```

Arguments

param (array like) – The parameters governing the evolution of the birth-and-death process to be simulated. Array of real elements with size `m`, where `m` is the number of parameters.

model (string) – Choice of model as described in Section 2.1. Should be one of the choices in Table 1 or "custom" (see Section A.12 for details on specifying custom models).

z0 (integer or callable) – The initial population size for each sample path. If it is a callable it should be a function that has no arguments and returns an integer, for example if the initial population follows a geometric distribution with parameter 0.35, then `z0` could be specified by `z0 = lambda: np.random.default_rng().geometric(0.35)`.

t_max (scalar) – The simulation horizon. All events up to and including this time are included in the output.

k (integer, optional) – The number of independent sample paths to be simulated.

¹CUDA is the acronym for compute unified device architecture

survival (boolean, optional) – If set to `True`, then the simulated sample paths are conditioned to have a positive population size at the final observation time. Since this uses acceptance-rejection it can greatly increase computation time.

seed (integer, Generator, optional) – If **seed** is not specified the random numbers are generated according to `numpy.random.default_rng()`. If **seed** is an integer, random numbers are generated according to `numpy.random.default_rng(seed)`. If **seed** is a `Generator`, then that object is used.

Returns

jump_times (list) – If **k** is 1, then this records a list containing jump times, generated according to **model**. If **k** is greater than 1, then this records a list of lists where each list corresponds to the jump times from one sample path.

pop_sizes (list) – If **k** is 1, then this records a list containing population sizes at the corresponding elements of **jump_times**, generated according to **model**. If **k** is greater than 1, then this records a list of lists where each list corresponds to the population sizes corresponding to **jump_times** from one sample path.

A.2. Discrete simulation: `bd.simulate.discrete()`

This function is used to simulate PSDBDPs at discrete observation times.

Usage and default argument values

```
bd.simulate.discrete(param, model, z0, times, k = 1, method = "exact",
    tau = 0.1, survival = False, seed = None, display = False, **options)
```

Arguments

This function has all of the parameters listed for `bd.simulate.continuous()` described in Section A.1 except for parameter **t_max**. In addition it has parameters:

times (array like) – The discrete times at which the simulated birth-and-death is observed. Array of real elements of size **n**, where **n** is the number of observation times.

method (string, optional) – Simulation algorithm used to generate samples as described in Section 2.2. Should be one of the choices in Table 2.

tau (scalar, optional) – Time between samples for the approximation methods "ea", "ma" and "gwa" described in Table 2.

display (boolean, optional) – If set to `True`, then a progress indicator is printed as the simulation is performed.

Returns

out (array like) – If **k** is equal to 1, then this records a list containing sampled population size observations at **times**, generated according to **model**. Or if **k** is greater than 1, then a **NumPy** array containing **k** sample paths, each contained in a row of the array, is generated.

A.3. Discrete simulation on GPU: `bd.gpu_functions.discrete()`

Exact simulation of continuous-time birth-and-death processes at a specified time using CUDA.

Usage and default argument values

This function is not imported by default with **BirDePy**, the `gpu_functions` module needs to be imported:

```
import birdepy.gpu_functions as bdg
bdg.discrete(param, model, z0, t, k = 1, survival = False, seed = 1)
```

Arguments

This function has almost the same parameters as `bd.simulate.discrete()` described in Section A.2. It does not have the parameters `method`, `tau` or `display`, and parameter `t` replaces parameter `time`. It does not accept custom model functionality.

`t` (scalar) – The time at which the simulated birth-and-death is observed.

Returns

`out` (array like) – A list containing `k` sampled population size observations at time `t` which are generated according to `model`.

A.4. Transition probabilities: `bd.probability()`

This function computes approximations to transition probabilities of PSDBDPs.

Usage and default argument values

```
bd.probability(z0, zt, t, param, model = "Verhulst", method = "expm",
               **options)
```

Arguments

`z0` (array like) – States of birth-and-death process at time 0

`zt` (array like) – States of birth-and-death process at time(s) t

`t` (array like) – Elapsed time(s) (if this has size greater than 1, then it must be increasing)

`param` (array like) – The parameters governing the evolution of the birth-and-death process. Array of real elements of size `n`, where `n` is the number of parameters.

`model` (string, optional) – Model specifying birth and death rates of process as described in Section 2.1. Should be one of the choices in Table 1 or "custom" (see Section A.12 for details on specifying custom models).

`method` (string, optional) – Transition probability approximation method as described in Section 2.3. Should be one of the choices in Table 3.

`options` (dictionary, optional) – A dictionary of method specific options. Methods "uniform", "Erlang", "ilt", "da" enable an optional parameter `k` which tunes accuracy. Methods

Method label	Brief description/reference
"cme-talbot" (default)	Attempts method "cme-mp" and if an error occurs attempts method "talbot".
"cme"	'CME' method of Horváth <i>et al.</i> (2020), implementation downloaded from Horváth, Horváth, Almousa, and Telek (2019).
"euler"	'Euler' method of Abate and Whitt (2006), implementation downloaded from Horváth <i>et al.</i> (2019).
"gaver"	Function <code>mpmath.invertlaplace(method = "gaver")</code> of Johansson <i>et al.</i> (2013).
"talbot"	Function <code>mpmath.invertlaplace(method = "talbot")</code> of Johansson <i>et al.</i> (2013).
"stehfest"	Function <code>mpmath.invertlaplace(method = "stehfest")</code> of Johansson <i>et al.</i> (2013).
"dehoog"	Function <code>mpmath.invertlaplace(method = "dehoog")</code> of Johansson <i>et al.</i> (2013).
"cme-mp"	Version of "cme" coded using <code>mpmath</code> operations.
"gwr"	Method of Valkó and Abate (2004) as implemented in <code>gwr_inversion.gwr()</code> .

Table 11: Methods for performing numerical Laplace transform inversion available in **BirDePy**.

"expm", "uniform", "Erlang" enable an optional parameter `z_trunc` which dictates truncation thresholds, i.e., minimum and maximum states of process considered; this is an array of real elements of size 2, by default `z_trunc = [z_min, z_max]` where `z_min = max(0, min(z0, zt) - 100)` and `z_max = max(z0, zt) + 100`. Method "ilt" also enables optional parameters: (i) `lent_eps` which is a scalar termination threshold for the Lentz algorithm computation of (8), (ii) `laplace_method` which is a str that determines which Laplace inversion method from Table 11 to use, and (iii) `precision` which is an integer that determines the numerical precision to use.

Returns

`transition_probability` (**NumPy** array) – An array of transition probabilities. If `t` has size bigger than 1, then the first coordinate corresponds to `t`, the second coordinate corresponds to `z0` and the third coordinate corresponds to `zt`; for example if `z0 = [1,3,5,10]`, `zt = [5,8]` and `t = [1,2,3]`, then `transition_probability[2,0,1]` corresponds to $\mathbb{P}(Z(3) = 8 | Z(0) = 1)$. If `t` has size 1, then the first coordinate corresponds to `z0` and the second coordinate corresponds to `zt`.

A.5. Transition probabilities on GPU: `bd.gpu_functions.probability()`

This function computes transition probabilities for PSDBDPs using Monte Carlo simulation on a GPU.

Usage and default argument values

This function is not imported by default with **BirDePy**, the `gpu_functions` module needs to be imported:

```
import birdepy.gpu_functions as bdg
bdg.probability(z0, zt, t, param, model, k = 10**6, seed = 1)
```

Arguments

This function has almost the same parameters as `bd.probability()`, described in Section A.4. It does not have the parameters `method` or `options`. In addition it does have:

k (integer, optional) – Minimum number of samples used to generate each probability estimate (actual number of samples will usually be higher due to the way memory is allocated on GPU). The total number of samples used will be at least `z0.size * k`.

Returns

As for `bd.probability()` described in Section A.4.

A.6. Parameter estimation: `bd.estimate()`

This function performs parameter estimation for (continuously or discretely observed) PSDB-DPs. Depending on the argument of `framework`, various optional input parameters become available; these optional input parameters are described in Section A.7, Section A.8, Section A.9 and Section A.10.

Usage and default argument values

```
bd.estimate(t_data, p_data, p0, p_bounds, framework = "dnm",
            model = "Verhulst", scheme = "discrete", con = (), known_p = (),
            idx_known_p = (), se_type = "asymptotic", ci_plot = False, export = False,
            display = False, **options)
```

Arguments

t_data (list) – Observation times of birth-and-death process. If one trajectory is observed, then this is a list. If multiple trajectories are observed, then this is a list of lists where each list corresponds to a trajectory.

p_data (list) – Observed population sizes of birth-and-death process at times in parameter `t_data`. If one trajectory is observed, then this is a list. If multiple trajectories are observed, then this is a list of lists where each list corresponds to a trajectory.

p0 (array like) – Initial parameter guess. Array of real elements of size `m`, where `m` is the number of unknown parameters.

p_bounds (list) – Bounds on parameters. Should be specified as a sequence of (min, max) pairs for each unknown parameter. See Section A.13 for more information.

framework (string, optional) – Parameter estimation framework. Should be one of: (i) "abc" (see Section A.7), (ii) "dnm" (see Section A.8), (iii) "em" (see Section A.9), or (iv) "lse" (see Section A.10).

model (string, optional) – Model specifying birth and death rates of process as described in Section 2.1. Should be one of the choices in Table 1 or "custom" (see Section A.12 for details on specifying custom models).

scheme (string, optional) – Observation scheme. Should be one of: (i) "discrete" or (ii) "continuous". If set to "continuous", then it is assumed that the population is observed continuously with jumps occurring at times in **t_data** into corresponding states in **p_data**.

con ({Constraint, dictionary} or List of {Constraint, dictionary}, optional) – Constraints definition for parameters. See Section A.13 for more information.

known_p (array like, optional) – List of known parameter values. For built in models these must be in their canonical order as given in Table 1. If this argument is given, then **idx_known_p** must also be specified. See Section A.13 for more information.

idx_known_p (array like, optional) – List of indices of known parameters (as given in argument **known_p**). For built in models indices must correspond to canonical order as given in Table 1. If this argument is given, then argument **known_p** must also be specified. See Section A.13 for more information.

se_type (string, optional) – Should be one of: (i) "none", (ii) "simulated", or (iii) "asymptotic". See Section A.14 for more information.

ci_plot (boolean, optional) – Enables confidence region plotting for 2-dimensional parameter estimates. See Section A.14 for more information.

export (string, optional) – File name for export of confidence region figure to a L^AT_EX file.

display (boolean, optional) – If set to True, then a progress indicator is printed for some methods.

Returns

An object **res** with the following attributes is returned. Note: not all attributes are relevant to all frameworks and schemes.

p (list) – Parameter estimate.

capacity (list) – Estimated possible carrying capacity values.

val (scalar) – If **framework** is "dnm" or "em": value of log-likelihood at parameter estimate. If **framework** is "lse": value of squared error at parameter estimate.

cov (array like) – Covariance matrix describing uncertainty of parameter estimates.

se (list) – Standard errors of parameter estimates.

compute_time (scalar) – Total central processing unit (CPU) time used by function (seconds).

framework (string) – Value of **framework** argument input.

message (string) – Message from optimization routine (if used).

success (boolean) – Success indicator from optimization routine (if used).

iterations (list) – Parameter estimates at each iteration. Relevant to ABC and EM frameworks.

method (string) – Value of **method** argument input.

p0 (list) – Initial parameter value.

`scheme` (string) – Value of `scheme` argument input.

`samples` (list) – Accepted samples for the ABC framework.

A.7. ABC algorithm: `bd.estimate(framework = "abc")`

This function performs parameter estimation for discretely-observed PSDBDPs using an ABC algorithm. This framework is called using the function `bd.estimate()` described in Section A.6.

Usage and default argument values

```
bd.estimate(t_data, p_data, p0, p_bounds, framework = "abc",
            eps_abc = "dynamic", k = 100, max_its = 3, max_q = 0.99, eps_change = 5,
            gam = 5, method = "gwa", tau = None, seed = None, distance = None,
            stat = "mean", display = False)
```

Arguments

In addition to the parameters of `bd.estimate()` described in Section A.6:

`eps_abc` (list, string, optional) – Threshold error tolerance for distance between simulated data and observed data for accepting parameter proposals. If this is set to "dynamic" (default), then a slightly modified version of the procedure described in [Simola *et al.* \(2021\)](#) is used. Otherwise `eps_abc` must be a list which specifies the threshold for each iteration.

`k` (integer, optional) – Number of successful parameter samples used to obtain the estimate.

`max_its` (integer, optional) – Maximum number of iterations of the algorithm.

`max_q` (scalar, optional) – Tolerance threshold for stopping algorithm (see (2.5) in [Simola *et al.* \(2021\)](#)). This is only checked after at least two iterations have occurred. Should be selected from (0, 1].

`eps_change` (scalar, optional) – An iteration is only performed if the percentage decrease in the tolerance threshold compared to the previous iteration is greater than this value. If `eps_change = 0` and `eps_abc = "dynamic"`, then the procedure described in [Simola *et al.* \(2021\)](#) is used.

`gam` (integer, optional) – If `eps_abc` is set to "dynamic", then `k*gam` parameter values are initially sampled and the distance between the data and the k -th largest distance corresponding to these samples is used as the first tolerance threshold.

`distance` (callable, optional) – Computes the distance between simulated data and observed data. The default value is (14).

`stat` (string) – Determines which statistic is used to summarize the posterior distribution. Should be one of: "mean" or "median".

This function also accepts the optional parameters `method`, `tau`, `seed`, `display` as described in Section A.2.

A.8. DNM algorithm: `bd.estimate(framework = "dnm")`

This function performs parameter estimation for discretely observed PSDBDPs using direct numerical maximization of approximate likelihood functions. This framework is called using the function `bd.estimate()` described in Section A.6.

Usage and default argument values

```
bd.estimate(t_data, p_data, p0, p_bounds, framework = "dnm",
  likelihood = "expm", z_trunc = ())
```

Arguments

In addition to the parameters of `bd.estimate()` described in Section A.6: (i) the parameter `likelihood` as described in Section A.4 is available, and (ii) when parameter `likelihood` has value "expm", "uniform" or "Erlang" the parameter "z_trunc" as described in Section A.4 becomes available.

A.9. EM algorithm: `bd.estimate(framework = "em")`

This function performs parameter estimation for discretely-observed PSDBDPs using an EM algorithm. This framework is called using the function `bd.estimate()` described in Section A.6.

Usage and default argument values

```
bd.estimate(t_data, p_data, p0, p_bounds, framework = "em",
  technique = "expm", accelerator = "none", likelihood = "expm",
  laplace_method = "cme-talbot", lentz_eps = 1e-6, max_it = 100,
  i_tol = 1e-3, j_tol = 1e-2, h_tol = 1e-2, z_trunc = ())
```

Arguments

In addition to the parameters of `bd.estimate()` described in Section A.6:

technique (string, optional) – Determines how the integrals in (12) are evaluated. Must be one of "expm", "ilt" or "num" which respectively result in matrix exponential, inverse Laplace transform and numerical methods being called upon.

accelerator (string, optional) – EM accelerator to be used. Must be one of "cg", "none", "Lange", "qn1" or "qn2". The first option uses the method of Jamshidian and Jennrich (1993), the second uses the method of Dempster *et al.* (1977), the third uses the method of Lange (1995b), and the final two use methods from Jamshidian and Jennrich (1997).

i_tol (scalar, optional) – Algorithm terminates when $\sum(\text{abs}(p(i) - p(i-1))) < i_tol$ where $p(i)$ and $p(i-1)$ are parameter estimates corresponding to iteration i and $i - 1$.

j_tol (scalar, optional) – States z with expected number of upward transitions $u_{z;i;j}$ (as defined in (12)) or expected number of downward transitions $d_{z;i;j}$ (as defined in (12)) greater than j_tol are included in E steps.

h_tol (scalar, optional) – States z with expected holding time $h_{z;i;j}$ (as defined in (12)) greater than h_tol are included in E steps.

As described in Section A.4 this function also accepts the optional parameters `likelihood`, `laplace_method`, `lantz_eps` and `z_trunc`.

A.10. LSE algorithm: `bd.estimate(``framework = "lse")`

This function performs parameter estimation for PSDBDPs using least squares estimation. This framework is called using the function `bd.estimate()` described in Section A.6.

Usage and default argument values

```
bd.estimate(t_data, p_data, p0, p_bounds, framework = "lse", squares = "fm",
            z_trunc = ())
```

Arguments

In addition to the parameters of `bd.estimate()` described in Section A.6:

squares (string, optional) – Method used to compute approximate expected values so that the squared difference between observed data and the expected value can be approximated. Should be one of "expm", "fm" or "gwa" which respectively use a matrix exponential, diffusion mean or linear approximation approach to approximate the expected value.

When parameter **squares** is set to "expm" optional parameter "z_trunc" as described in Section A.4 becomes available.

A.11. Forecasting: `bd.forecast()`

This function performs simulation or numerical based forecasting for PSDBDPs. It produces a plot of the likely range of mean population sizes subject to parameter uncertainty (confidence intervals) or the likely range of population sizes subject to parameter uncertainty and model stochasticity (prediction intervals).

Usage and default argument values

```
bd.forecast(model, z0, times, param, cov = None, interval = "confidence",
            method = None, percentiles = (0, 2.5, 10, 25, 50, 75, 90, 97.5, 100),
            labels = ('$95\\%$', '$80\\%$', '$50\\%$'), p_bounds = None, con = (),
            known_p = (), idx_known_p = (), k = 10 ** 3, n = 10 ** 3, seed = None,
            colormap = cm.Purples, xlabel = "Time", ylabel = "default",
            xticks = "default", rotation = 45, display = False, export = False,
            **options)
```

Arguments

This function shares some parameters with `bd.simulate()` and `bd.estimate()`. Parameters `model`, `z0`, `times`, `param`, `seed` and `display` are described in Section A.2. Parameters `p_bounds`, `con`, `known_p`, and `idx_known_p` are described in Section A.6. In addition:

interval (string, optional) – Type of forecast. Should be one of "confidence" (default) or "prediction". Confidence intervals show the likely range of mean future population

values, reflecting parameter uncertainty. Prediction intervals show the likely range of future population values, incorporating parameter uncertainty and model stochasticity.

method (string, optional) – Method used to generate samples. For confidence intervals samples are trajectories of future expected values. For prediction intervals samples are trajectories of future population values. Should be one of: "fm" (default for confidence intervals), "exact", "ea", "ma" or "gwa" (default for prediction intervals) .

cov (array like, optional) – If this is specified, then the parameters are assumed to follow a truncated normal distribution with this covariance (and mean **param**). In this case parameter **p_bounds** should also be specified to avoid unwanted parameters.

percentiles (list, optional) – List of percentiles to split data into.

labels (list, optional) – List of strings containing labels for each percentile split.

k (integer, optional) – Number of samples used to generate the forecast. For confidence intervals each sample corresponds to an estimate of the mean for a parameter value sampled from a normal distribution with mean given by **param** and covariance given by **cov**. For prediction intervals each sample corresponds to a simulated trajectory of population size for a parameter value sampled in the way just described.

n (integer, optional) – Number of samples used to estimate each sample of a mean for confidence interval samples. Only applicable when method is "exact", "ea", "ma" or "gwa".

colormap (matplotlib.colors.LinearSegmentedColormap, optional) – Colors used for plot.

xlabel (string, optional) – Label for x axis of plot.

ylabel (string, optional) – Label for y axis of plot.

xticks (array like, optional) – Locations of x ticks.

rotation (integer, optional) – Rotation of x tick labels.

export (boolean, optional) – If True, then a L^AT_EX file containing the figure in PGF/TikZ is generated.

A.12. Custom models

It is possible to specify custom birth and death rates for the five core **BirDePy** functions. This is achieved by setting **model** to "custom" and additionally passing callables to arguments **b_rate** and **d_rate**. These callables must take as input a population size **z** and a list of parameters **p**, and respectively return scalars corresponding to the birth and death rates at population size **z**. For example: **b_rate = lambda z, p: p[0]*z**2** and **b_rate = lambda z, p: p[0]*z**.

A.13. Known parameters, optimization options and constraints

It may be the case that some of the parameters of a model are known and others are unknown. In this case, the parameters **known_p** and **idx_known_p** can be passed to **bd.estimate()** or **bd.forecast()**. The parameter **known_p** is an ordered list of known parameter values and **idx_known_p** is an ordered list containing the indices of the known parameters as they appear in the parameter **param** in the function **bd.probability()**. For built-in models the indices of parameters correspond to their canonical order in Table 1.

By default, when an optimization routine is needed, **birdepy.estimate()** uses the option

"L-BFGS-B" of `scipy.optimize.minimize()`, unless constraints are specified in which case "SLSQP" is used instead. The other choices of method in `scipy.optimize.minimize()` can be accessed in `birdepy.estimate()` using the optional parameter `opt_method`. It is also possible to use `scipy.optimize.differential_evolution()` by setting `opt_method` to "differential-evolution". In addition, many of the optional arguments of the functions `scipy.optimize.minimize()` and `scipy.optimize.differential_evolution()` can be used as optional parameters in `birdepy.estimate()`.

Constraints on parameters are passed to `bd.estimate()` or `bd.forecast()` through the parameter `con`. This is in addition to the bounds that are passed to parameter `p_bounds`. Constraints are used by the `scipy.optimize` functions `minimize()` or `differential_evolution()`. Each constraint must be a dictionary with fields:

type (string) – Constraint type: "eq" for equality, "ineq" for inequality.

fun (callable) – The function defining the constraint.

args (sequence, optional) – Extra arguments to be passed to the function.

Equality constraint means that the constraint function result is to be zero, whereas inequality means that it is to be non-negative. Note that setting `opt_method` to "COBYLA" means only inequality constraints are supported.

A.14. Standard errors and confidence regions

The attributes and parameters in this section relate to the function `bd.estimate()` and its output `res`. The confidence regions described here are related to—but not the same as—the confidence intervals described in Section 2.5 (which are relevant to `bd.forecast()` as described in Section A.11). Available methods for computing confidence regions and standard errors to quantify the uncertainty of parameter estimates depends on the framework being employed. This section is concerned with the attributes `cov` and `se` of the output of `bd.estimate()`, and the confidence region plot that is printed when `ci_plot` is set to `True`. The behavior of these objects depends on the argument of `se_type`, which may be set to "none", "asymptotic" or "simulated". If "none" is chosen, then `cov` and `se` will be empty and setting `ci_plot` to `True` will result in an error.

For frameworks "dnm" and "em", when `se_type` is set to "asymptotic", a covariance matrix is returned in attribute `cov` as the negative of the inverse of the Hessian matrix of the approximate log-likelihood function at the point of the parameter estimate contained in `res.p`. In this case, standard errors are returned in the attribute `se` by assuming a normal distribution with mean given by attribute `p` and covariance given by attribute `cov`. In addition, when two parameters are being estimated, setting the optional parameter `ci_plot` to `True` results in a plot of confidence regions generated according to this assumed normal distribution.

For frameworks "dnm", "em" and "lse", when `se_type` is set to "simulated", given a parameter estimate, the following steps are repeated:

1. Simulate data \hat{z} which has the same form as z , that is, consisting of points generated using `bd.simulate.discrete()` with the arguments of `param`, `z0` and `times` set respectively to `res.p`, `p_data[i][0]` and `t_data[i]` for `i` in `range(len(p_data))`.
2. Apply the same parameter estimation technique to find the estimate corresponding to \hat{z} .

These steps are repeated 100 times (or a different number of times, as specified by parameter `num_samples`), and then a multivariate normal is fitted to the collection of estimates that are generated at Step 2 of each iteration. From this a covariance matrix is returned as the attribute `cov`, which characterizes the uncertainty of the estimate given by attribute `p`. In this case, this multivariate normal is used to generate: (i) standard errors which are recorded in attribute `se`, and (ii) the confidence region plotted when `ci_plot` is set to `True`.

For framework "abc", the attribute `samples` is returned, which is a recording of the accepted parameter proposals. Performing a density estimate on these samples is a standard method of characterizing uncertainty for ABC estimates. In addition, attribute `cov` is returned as an estimate of the covariance of the final iteration accepted parameter proposals, and this is also used to generate the standard errors returned in attribute `se`.

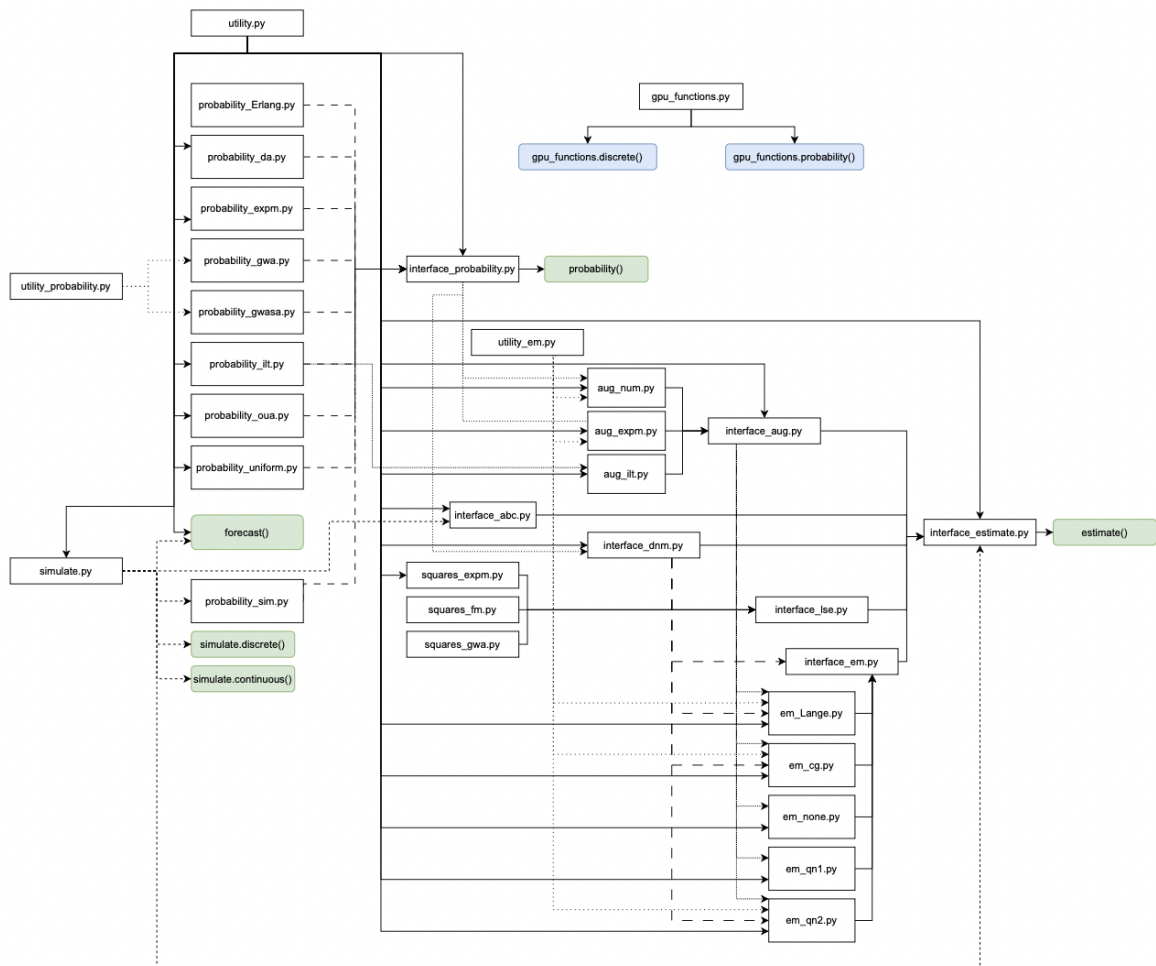


Figure 11: High-level diagram summarizing the codebase of the **BirDePy** package. White boxes are Python files containing code, green boxes are core functions, and blue boxes are functions which use CUDA. Arrows indicate code dependency; an arrow going from one box to another indicates code from the source box is used in the destination box.

B. BirDePy development

Since the package incorporates various simulation and estimation methods, we have organized each method into its own Python file. These methods are then consolidated and made available for use by the corresponding user-facing function. This approach ensures that the codebase remains structured and facilitates the accessibility of specific methods as needed. For example, the package includes eight Python files specifically dedicated to computing probabilities (`probability_Erlang.py`, `probability_da.py`, etc.); these files are collected together in `interface_probability.py` that contains the function `birdepy.probability()`.

By distributing methods across multiple files, we are able to minimize code duplication. This approach proves beneficial when a particular functionality is required in multiple instances, as it can be stored in a separate file and imported as needed. For instance, the file `utility.py` contains numerous functions used repetitively throughout the package.

Figure 11 shows a high level diagram which describes the codebase of the package.

To foster ongoing development and encourage contributions from the community, the code repository (https://github.com/birdepy/birdepy_project) has been configured with the use of **semantic-release** (Martynus, Vanduyndslager, and Travi 2023). This setup ensures that new versions of the package are made available on the Python package index from the main branch. We note that the pipeline for releasing new versions will only execute if commit messages adhere to the correct formatting, as outlined in the **semantic-release** documentation. For instance, using a commit message like ‘Fix stuff’ will not trigger the release pipeline, whereas a properly formatted message such as ‘fix: stuff’ is necessary for successful execution. To ensure that changes are accepted for merging into the main branch, they must successfully pass a set of unit tests. These tests have been set up using the **unittest** module from the standard library, as well as the **pytest** framework (Krekel 2023).

Affiliation:

Sophie Hautphenne, Brendan Patch
School of Mathematics and Statistics
The University of Melbourne
Victoria 3010, Australia

E-mail: sophiemh@unimelb.edu.au

URL: <http://www.sophiehautphenne.info/>, <https://github.com/bpatch>