`mpmsim`: An R package for simulating matrix population models

Owen R. Jones[1*]

[1] Department of Biology, University of Southern Denmark, Campusvej 55, Odense M, Denmark

* Corresponding author. Email: jones@biology.sdu.dk

Abstract

1. Matrix population models (MPMs) are widely used in ecology and evolution to explore population dynamics, including assessing management impacts and extinction risk. In comparative studies, MPMs can be used to test life history theory or investigate macro-evolutionary patterns in demographic traits.
2. Simulated MPMs can help researchers explore the effects of life cycle structure, vital rate trajectories, and uncertainty in transition rates due to sampling error. They are also valuable teaching tools.
3. The **mpmsim** R package enables users to simulate random or semi-random Lefkovitch and Leslie MPMs based on life history archetypes or mortality and reproductive output patterns. It also allows the exploration of sampling error effects and uses parametric bootstrapping to calculate confidence intervals for matrix-derived estimates.
4. **mpmsim** provides a convenient toolset for addressing questions about MPMs and life history, with full documentation and user-friendly vignettes.

Keywords: simulation, sampling error, bias, teaching tools, mortality trajectory, fertility trajectory, Leslie matrix, Lefkovitch matrix, life history archetypes

Introduction

Matrix population models (MPMs) are a versatile tool in population biology and evolution (Caswell, 2001), first introduced by Leslie (1945) to study age-structured population dynamics. Lefkovitch (1965) expanded the approach to stage-classified life histories, followed by developments like stochastic (Cohen, Christensen & Goodyear, 1969) and density-dependent MPMs (Pennycuick, 1969), broadening their applications. Analytical methods such as elasticity and perturbation analysis, life table response experiments (LTRE), and Markov chain methods (Caswell, 2001) have further enhanced their utility.

MPMs describe a population's demography at a specific time and place by modelling individuals categorised by (st)age over a discrete projection interval (Caswell, 2001). At the core is the projection matrix (**A**), representing transitions between stages through survival, growth, and reproduction, which can be split into submatrices **U** (growth/survival), **F** (sexual reproduction), and **C** (clonal reproduction), such that $\mathbf{A} = \mathbf{U} + \mathbf{F} + \mathbf{C}$. MPMs provide biologically meaningful outputs to estimate population growth rates, extinction risk, responses to vital rate perturbations, transient dynamics, effective population size, and life history traits. Consequently, MPMs have been pivotal in advancing population biology and life history theory (Caswell, 2001; Crone et al., 2011). Supporting this, the COMPADRE and COMADRE databases (Salguero-Gomez et al., 2015, 2016) provide >12,000 MPMs for >1,100 species, ranging from annual plants to whales, which address topics from population management to evolutionary theory.

Several R packages support MPM analysis (e.g., `popbio` (Stubben & Milligan, 2007), `popdemo` (Stott et al., 2012), `Rage` (Jones et al., 2022), `exactLTRE` (Hernández et al., 2023)), but none provide broad scope for simulating MPMs with specific characteristics. This limits researchers' ability to explore population dynamics beyond empirical data constraints (Römer et al., 2024). To address this, I introduce `mpmsim`, an R package designed to simulate MPMs with defined characteristics, enabling users to explore life history and population dynamics. The core functions are `make_leslie_mpm` and its wrapper `rand_leslie_set` for Leslie matrices, and `rand_lefko_mpm` and `rand_lefko_set` for Lefkovitch matrices. The functions `compute_ci` and `compute_ci_U` calculate confidence intervals via parametric bootstrapping, and `add_mpm_error` allows users to simulate MPMs with sampling error. Together, these functions enable simulations of diverse life histories and the assessment of the impact of sampling error on inferences.

Illustrating use of `mpmsim`

To demonstrate `mpmsim`'s versatility, I provide three examples in code boxes below. First, I show how to generate Leslie matrices based on mortality and reproductive trajectories. Second, I generate Lefkovitch models using defined life cycle characteristics. In both cases, the simulations assume a post-breeding census, thus avoiding the often overlooked issue of unaccounted survival to reproduction highlighted by Kendall et al. (2019). Finally, I calculate confidence intervals for MPM-derived estimates, which can include a diverse set of demographic and life history parameters, such as population growth rate (λ), generation time, mean age at reproduction, and many others. These code boxes are concise demonstrations, illustrating key functions and workflows. However, they are not exhaustive. The vignettes (Supporting Information Vignettes S1, S2, and S3) provide more detailed explanations, covering additional use cases, variations in parameter choices, and practical considerations for different types of analyses. Readers seeking a deeper understanding or additional applications are encouraged to consult these resources. The package can be installed directly from CRAN using the command `install.packages("mpmsim")`. By default, all required dependencies will be installed automatically.

Example 1: Generating Leslie matrices

Leslie MPMs model age-classified populations, with survival probabilities (*p_x*) in the subdiagonal representing survival probabilities from age *x* to *x+1*, and fecundity (*f_x*) in the first row. Matrix **A** thus combines survival/growth (**U**) and sexual reproduction (fecundity) (**F**), such that **A** = **U** + **F**. For clonal organisms, a third submatrix **C** can be added (**A** = **U** + **F** + **C**).

The main function for generating Leslie MPMs in the package is `rand_leslie_set`, which creates MPMs based on randomly drawn parameters from specified mortality and fecundity models. Users use arguments to define the model types (`mortality_model`, `fecundity_model`), parameter distributions (`mortality_params`, `fecundity_params`, `fecundity_maturity_params`), and the number of MPMs (`n_models`). The function outputs MPMs either as a `list` or a `compadreDB` object (Jones et al., 2022), depending on the output argument. The underlying functions `model_mortality` and `model_fecundity` calculate age-specific survival (*p_x*) and reproductive output (*f_x*) using standard functional forms.

The available mortality models include Siler, Exponential, Gompertz, Gompertz-Makeham, Weibull, and Weibull-Makeham (Table 1). The `model_mortality` function calculates survival probabilities for each age based on age-specific hazard rates. It first calculates hazard rates (*h_x*) from the chosen mortality model using the model argument and a vector of parameters (`params`). The cumulative hazard (*H_x*) is then computed by integrating *h_x* up to each age, giving total mortality risk. Survivorship (*l_x*) is determined as *$exp(-H_x)$*, and the age-specific survival probability (*p_x*) is the ratio of survivorship values at *x+1* and *x*. The function outputs a life table as a data frame that extends by default until *l_x* drops below 0.01.

The `model_fecundity` function calculates age-specific reproductive output (*f_x*), the average number of offspring produced at age *x*, using canonical models such as logistic, step-function, von Bertalanffy, Hadwiger, and Normal (Table 1). Key arguments include the model type (`model`), parameters (`params`), and a vector of ages (`age`), and the output is a vector of age-specific reproductive output values corresponding to the input ages.

Code Box 1 shows how to create 500 Leslie MPMs with Gompertz-Makeham mortality and step-function fecundity, with ages at maturity varying from one to four, and parameter values drawn from uniform distributions. In this case, the output is in the form of a `compadreDB` object (Jones et al. 2022), but this can be set to output a standard R `list` object using the `output` argument. The code runs in 1.89 seconds (SD = 0.07; 100 runs) on a MacBook running macOS (Sequoia 15.3) with an Intel quad Core i5 2.40 GHz CPU, 16 GB of RAM, and R version 4.4.2. Naturally, the simulation time increases with the number of models requested (Fig. 1A).

Code box 1: Leslie matrix models

```r
# Load package
library(mpmsim)

# Define mortality model parameters
# min/max values in Gompertz-Makeham model
mortParams <- data.frame(
  minVal = c(0, 0.01, 0.1),
  maxVal = c(0.05, 0.15, 0.2)
)
# Define fecundity model parameters
# min/max values in step model
fecundityParams <- data.frame(
  minVal = 2,
  maxVal = 10
)
# Define age-at-maturity
# min/max values
maturityParam <- c(1, 4)

# Produce 500 MPMs
# Gompertz-Makeham mortality model and step function fecundity
# Parameters drawn from uniform distribution
# Output compadreDB
myMatrices <- rand_leslie_set(
  n_models = 500,
  mortality_model = "GompertzMakeham",
  fecundity_model = "step",
  mortality_params = mortParams,
  fecundity_params = fecundityParams,
  fecundity_maturity_params = maturityParam,
  dist_type = "uniform",
  output = "Type1"
)
```

Example 2: Generating Lefkovitch matrices

Lefkovitch MPMs are stage-based, making them ideal when age data is unavailable or less relevant, such as in life cycles governed by developmental stages (e.g., juvenile, adult). A key advantage is their adaptability to various life cycles, including transitions like retrogression or dormancy. This flexibility is essential for studying species with non-age-based life cycles. While these models accommodate diverse life cycles, some rules apply: survival probabilities cannot exceed 1, and reproduction cannot be negative. Thus, transition probabilities in **U** range from 0 to 1, with column sums constrained to ≤ 1, while fecundity in **F** has a lower limit of zero.

In `mpmsim`, the function `rand_lefko_set` generates sets of Lefkovitch MPMs and is a wrapper for `rand_lefko_mpm`, which creates individual MPMs. These functions model the **U** matrix by drawing values from a random Dirichlet distribution, ensuring survival probabilities for each stage are ≤ 1. Users can generate various life cycle structures using the `archetype` argument, based on Takada *et al.*'s (2018) four archetypes. In Archetype 1, individuals can move freely between stages, either progressing or retrogressing, with no constraints on the transition rate. Archetype 2 adds the assumption that survival improves with stage progression. Archetype 3 allows only forward progression, mimicking species with slow development. Archetype 4, similar to Archetype 3, includes improved survival with stage progression but without retrogression.

In Takada *et al.*'s models, fecundity was placed in the top-right of the matrix, restricting reproduction to the final life cycle stage. In `mpmsim`, this constraint is relaxed through the `fecundity` argument, which offers four options: (1) a single value representing fecundity in the most developed stage, (2) a vector matching the number of stages to assign stage-specific fecundity across the top row, (3) a matrix defining fecundity for each element, or (4) a list of two matrices setting upper and lower fecundity limits, with values drawn from a uniform distribution. This flexibility captures various reproductive strategies, accommodating diverse life histories.

In addition to generating matrices based on selected archetypes, outputs can be fine-tuned using the `constraint` argument, allowing users to set limits based on any metric derived from the **A** matrix, such as asymptotic population growth rates within a defined range, to ensure viable life cycles. This enables tailored simulations for specific ecological or evolutionary scenarios.

Code Box 2 generates 500 Lefkovitch models in a `compadreDB` object for Archetype 4, constrained to have a λ between 0.9 and 1.1. Fecundity is set to 0 for the first two stages, and 8 and 14 for the last two. The code runs in 3.20 seconds (SD = 0.17; 100 runs). The simulation time increases with the number of models requested, with `rand_lefko_set` being slower than `rand_leslie_set` for a given number of models (Fig. 1B).

Code box 2: Lefkovitch matrix models

```r
# Load packages
library(popdemo)
library(mpmsim)

# Define constraints
# Lambda between 0.9 and 1.1
constrain_df <- data.frame(
  fun = "eigs", arg = "lambda", lower = 0.9, upper = 1.1
)

# Produce 500 MPMs, 3 stages, Archetype 4
# Set fecundity of 8 for stage 3 and 14 for stage 4.
# Output as compadreDB object
myMatrices <- rand_lefko_set(
  n = 500, n_stages = 4, fecundity = c(0,0,8,14),
  constraint = constrain_df, archetype = 4, output = "Type1")
```

Example 3: Calculating confidence intervals

MPMs are parameterised in various ways, often by estimating transition rates from repeated surveys of stage classes. Typically, only a population sample is used, and sample sizes may vary. For instance, juveniles might be common, resulting in larger sample sizes for estimating juvenile survival, while adults may be rarer, leading to smaller samples. As a result, sampling error varies across the matrix and between years in multi-year studies, potentially influencing life history or population dynamics analyses. I will illustrate how to explore these effects on inferences. Accounting for, and understanding this uncertainty is crucial, because incomplete propagation of sampling error can bias estimates of key demographic parameters such as population growth rate (λ), potentially leading to misleading conclusions (Simmonds & Jones, 2023). This has practical implications for population monitoring, conservation decision-making, and forecasting demographic trends. Below, I illustrate how mpmsim enables users to explore these effects, helping to improve the robustness of MPM-based inferences.

I start with the matrix model, **A**, the sum of the **U** and **F** submatrices. For example, $\mathbf{A} = \begin{bmatrix} 0.1 & 5.0 \\ 0.2 & 0.4 \end{bmatrix}$, $\mathbf{U} = \begin{bmatrix} 0.1 & 0 \\ 0.2 & 0.4 \end{bmatrix}$ and $\mathbf{F} = \begin{bmatrix} 0 & 5.0 \\ 0 & 0 \end{bmatrix}$. Transition rates are assumed to result from specific statistical distributions: sexual reproduction follows a Poisson process, while growth/survival transitions follow a binomial process. Specifically, survival probabilities (*p*) are drawn from a binomial distribution: $X \sim \text{Binomial}(n,\ p)$ where *n* is the sample size. The estimated survival probability is $\hat{p} = \frac{1}{n}\sum X$. Fecundity values (*F*) are drawn from a Poisson distribution: $Y \sim \text{Poisson}(\lambda)$, where λ represents the expected reproductive output. The estimated fecundity is $\hat{F} = \frac{1}{n}\sum Y$. The well-known properties of these statistical processes enable the estimation of expected distributions for each matrix element, based on their average values and sample sizes, which , in `mpmsim`, are provided to `compute_ci` as arguments `mat_U`, `mat_F` and `sample_size`. The `compute_ci` function then applies parametric bootstrapping, a resampling technique that assumes matrix elements follow a specified statistical distribution, defined by mean values and sample sizes. The function repeatedly draws random samples from these distributions, generating multiple simulated models. From these simulations, confidence intervals (CIs) are derived by analysing the variation in the resulting trait estimates, providing a measure of uncertainty that accounts for sampling variability. For tractability, this method currently assumes no covariance between rates, though trade-offs among elements may introduce nuances. Covariance among rates will be addressed in a future version of `mpmsim`. **Code Box 3** shows how to calculate confidence intervals for generation time and λ, with sample sizes of 15 for adult fecundity and 40 for the survival/growth transitions. In practice, users should use estimated mean values and sample sizes from their own studies. When survival or reproduction estimates are derived from multiple studies with varying sample sizes, users should calculate an effective sample size rather than summing sample sizes across studies, which overestimates precision. The effective sample size can be approximated using the harmonic mean ($N_{\text{eff}} = \frac{k}{\sum_{i=1}^{k}\frac{1}{N_i}}$). The harmonic mean gives more weight to smaller sample sizes, reflecting the reality that estimates from smaller studies contribute more variability. For example, if an estimate is based on two studies with sample sizes of N = 10 and N = 25, the effective sample size is: $N_{\text{eff}} = \frac{2}{\left(\frac{1}{10}+\frac{1}{25}\right)} = 14.29.$

Code box 3 – Confidence Intervals

```r
# Load packages
library(popbio)
library(mpmsim)

# Define U matrix
matU <- matrix(c(
  0.1, 0.0,
  0.2, 0.4
), byrow = TRUE, nrow = 2)

```

```r
# Define F matrix
matF <- matrix(c(
  0.0, 5.0,
  0.0, 0.0
), byrow = TRUE, nrow = 2)

# Combine matrices to create A matrix
matA <- matU + matF

# Define sample sizes for F
mat_F_ss <- matrix(c(
  0.0, 15,
  0.0, 0.0
), byrow = TRUE, nrow = 2)

# Define the sample sizes for U
mat_U_ss <- matrix(c(
  40, 40,
  40, 40
), byrow = TRUE, nrow = 2)

# Combine sample sizes into list
sampleSizes <- list(mat_U_ss = mat_U_ss, mat_F_ss = mat_F_ss)

# Calculate lambda
lambda(matA)
#> [1] 1.261187

# Calculate CI for lambda
compute_ci(
  mat_U = matU, mat_F = matF, sample_size = sampleSizes,
  FUN = lambda
)
#>      2.5%     97.5%
#> 0.9033941 1.6022154

# Calculate generation time
generation.time(matA)
#> [1] 2.65536

# Calculate CI for generation time
compute_ci(
  mat_U = matU, mat_F = matF, sample_size = sampleSizes,
  FUN = popbio::generation.time
)
#>     2.5%    97.5%
#> 2.371819 3.106800
```

An example analysis: the effect of life history constraints on life history structuring

In this section, I present a simplified analysis using `mpmsim` mto illustrate its application in a research context. Researchers are increasingly interested in understanding how life history traits are structured along specific axes or continua, as explored by Salguero-Gómez et al. (2016), Paniw et al. (2018), and Jones et al. (2020) among others. In these studies, MPMs are used to calculate key life history metrics, including generation time, survivorship curve type, age at sexual maturity, growth rate, life expectancy, mean sexual reproduction, iteroparity degree, and net reproductive rate and these are then analysed using multivariate techniques such as Principal Components Analysis (PCA) to reduce complexity and identify dominant patterns.

For instance, Salguero-Gómez et al. (2016) applied PCA to COMPADRE MPMs (Salguero-Gómez et al., 2015) and identified two major axes explaining 55% of the variation in plant life history strategies. The first axis corresponded to the fast–slow continuum, with longevity-related traits positively correlated and traits associated with growth and reproduction negatively correlated. The second axis represented reproductive strategy, capturing variation in iteroparity and net reproductive rate.

To investigate whether broad life history archetypes influence life history structuring, I employed a simulation-based approach. Using `mpmsim`'s `rand_lefko_set` function, I generated 250 random virtual species for each of two distinct life history archetypes, producing a total of 500 simulated species. These species were constrained to remain viable, with population growth rates (λ) between 0.9 and 1.1. The first archetype describes a life history in which individuals can progress and retrogress rapidly, such as growing and shrinking in response to environmental fluctuations. In contrast, the second archetype represents a life history where survival increases with age or size, retrogression is absent, and individuals can only progress to the next stage.

The hypothesis underlying my analysis is that PCA would reveal life history structuring consistent with the empirical findings of Salguero-Gómez et al. (2016) and that, since life history structuring is thought to be universal, the patterns should be similar for the two archetypes. Specifically, the first PCA axis was expected to represent the fast–slow continuum, with longevity-related traits positively correlated and growth and reproductive traits negatively correlated. The second PCA axis was anticipated to capture reproductive strategy, with iteroparity and net reproductive rate positively correlated and retrogressive growth negatively correlated.

The results (Figure 2) show that the two life history archetypes exhibit markedly different patterns of structuring. These findings suggest that life history constraints strongly influence the major axes of life history variation, reinforcing the idea that demographic constraints shape how species are organised within life history space. This simple analysis effectively demonstrates how `mpmsim` can be used to generate and analyse simulated life history data, providing a powerful tool for exploring theoretical questions in population biology. By leveraging such simulations, researchers can test hypotheses about life history structuring in a controlled and reproducible manner. The code to implement this analysis is given in Supporting Information S4.

Discussion

This package addresses a significant gap in the population modeling toolbox by providing a specialised framework for simulating matrix population models (MPMs) with controlled life history characteristics. While several R packages, such as `popbio` (Stubben & Milligan, 2007), `popdemo` (Stott et al., 2012), and `Rage` (Jones et al., 2022), facilitate MPM analysis, their primary focus is on extracting demographic parameters from empirical data. In contrast, `mpmsim` is explicitly designed for simulation, allowing researchers to explore theoretical and comparative scenarios that extend beyond the constraints of existing datasets.

A key strength of `mpmsim` is its flexibility. Unlike other MPM-related packages, which often require fully parameterized matrices as inputs, `mpmsim` generates matrices based on predefined life history traits, mortality and reproductive trajectories, or theoretical life cycle structures. This feature makes it particularly useful for hypothesis testing in macro-evolutionary and ecological research, where the goal is to examine how different life history strategies emerge under varying demographic constraints. Furthermore, the package's ability to incorporate sampling error and generate confidence intervals via parametric bootstrapping enhances the robustness of demographic inferences, an aspect often overlooked in prior studies.

Beyond research applications, `mpmsim` can also be a valuable tool for education in population ecology and evolutionary biology. By enabling students to generate and manipulate theoretical MPMs, the package provides a platform for illustrating fundamental demographic principles. Concepts such as life cycle complexity, survival trade-offs, and fertility trajectories can be directly explored, reinforcing the link between theoretical models and real-world population dynamics. This capability makes `mpmsim` a powerful complement to empirical analyses, bridging the gap between abstract theoretical models and observed demographic patterns.

Conclusion

In summary, `mpmsim` provides a novel and flexible framework for MPM simulation, supporting both theoretical and applied research in population biology. By allowing researchers to model demographic dynamics beyond empirical constraints, the package will play a crucial role in testing life history theory, quantifying uncertainty, and generating synthetic data for comparative analyses.

Acknowledgements

I thank Rob Salguero-Gómez, Samuel Gascoigne, Emily Simmonds, Patrick Barks, and Iain Stott for their discussions on the functionality of the package at various stages of its development. I also thank the two anonymous reviewers whose insightful and thoughtful comments have improved both the paper and the R package.

Conflict of Interest Statement

I have no known conflicts of interest.

Data availability statement

The package R code is available at GitHub: https://github.com/jonesor/mpmsim

The package can be installed from CRAN: https://CRAN.R-project.org/package=mpmsim

Supporting Information

Vignette S1: Generating Leslie models

Vignette S2: Generating Lefkovitch models

Vignette S3: Sampling error and its propagation

Supplemental Code S4: life history archetypes and life history structuring

References

Cochran, M. E., & Ellner, S. (1992). Simple methods for calculating age-based life history parameters for stage-structured populations. Ecological Monographs, 62(3), 345–364.

Cohen, J. E., Christensen, S. W., & Goodyear, C. P. (1983). An age-structured fish population model with random survival of eggs: calculation of asymptotic growth rates and application to Potomac River striped river bass. Journal of Fisheries and Aquatic Science, 40, 2170–2183.

Colchero, F., Jones, O. R., & Rebke, M. (2012). BaSTA: an R package for Bayesian estimation of age-specific survival from incomplete mark--recapture/recovery data with covariates. Methods in Ecology and Evolution, 3(3), 466–470.

Colchero, F., Jones, O. R., Conde, D. A., et al. (2019). The diversity of population responses to environmental change. Ecology Letters, 22(2), 342–353.

Cox, D.R. & Oakes, D. (1984) Analysis of Survival Data. Chapman and Hall, London, UK.

Crone, E. E., Menges, E. S., Ellis, M. M., et al. (2011). How do plant ecologists use matrix population models? *Ecology Letters*, *14*(1), 1–8.

Franco, M., & Silvertown, J. (1996). Life History Variation in Plants: An Exploration of the Fast-Slow Continuum Hypothesis. *Philosophical Transactions of the Royal Society of London*, 351(1345), 1341–1348.

Gompertz, B. (1825). XXIV. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. In a letter to Francis Baily, Esq. F. R. S. &c. Philosophical Transactions of the Royal Society of London, 115(0), 513–583.

Hadwiger, H. (1940). Eine analytische Reproduktionsfunktion für biologische Gesamtheiten. Scandinavian actuarial journal, 1940(3–4), 101–113.

Hernández, C. M., Ellner, S. P., Adler, P. B., Hooker, G., & Snyder, R. E. (2023). An exact version of Life Table Response Experiment analysis, and the R package exactLTRE. Methods in Ecology & Evolution, 14(3), 939–951.

Jones, O. R., Barks, P., Stott, I., et al. (2022). Rcompadre and Rage—Two R packages to facilitate the use of the COMPADRE and COMADRE databases and calculation of life-history traits from matrix population models. Methods in Ecology & Evolution, 13(4), 770–781.

Jones, O. R., Ezard, T. H. G., Dooley, C., Healy, K., Hodgson, D. J., Mueller, M., Townley, S., & Salguero-Gómez, R. (2020). My family and other animals: Human demography under a comparative cross-species lens. In O. Burger, R. Lee, & R. Sear (Eds.), Human Evolutionary Demography (pp. 1–31). OSF.

Kendall, B. E., Fujiwara, M., Diaz-Lopez, J., Schneider, S., Voigt, J., & Wiesner, S. (2019). Persistent problems in the construction of matrix population models. Ecological Modelling, 406, 33–43.

Lefkovitch, L. P. (1965). The study of population growth in organisms grouped by stages. Biometrics, 21(1), 1-18

Leslie, P. H. (1945). On the use of matrices in certain population mathematics. Biometrika, 33, 183–212.

Makeham, W. M. (1867). On the law of mortality. *Journal of the Institute of Actuaries (1866)*, *13*(06), 325–358.

Morris, W. F., & Doak, D. F. (2004). Buffering of life histories against environmental stochasticity: accounting for a spurious correlation between the variabilities of vital rates and their contributions to fitness. American Naturalist, 163(4), 579–590.

Paniw, M., Ozgul, A., & Salguero-Gómez, R. (2018). Interactive life-history traits predict sensitivity of plants and animals to temporal autocorrelation. Ecology Letters, 21(2), 275–286.

Pennycuick, L. (1969) A computer model of the Oxford great tit population. Journal of Theoretical Biology, 22, 381-400.

Peristera, P. & Kostaki, A. (2007) Modeling fertility in modern populations. Demographic Research. 16. Article 6, 141-194.

Pfister, C. A. (1998). Patterns of variance in stage-structured populations: evolutionary predictions and ecological implications. Proceedings of the National Academy of Sciences of the United States of America, 95(1), 213–218.

Pinder III, J.E., Wiener, J.G. & Smith, M.H. (1978) The Weibull distribution: a method of summarizing survivorship data. Ecology, 59, 175–179.

Pletcher, S. (1999) Model fitting and hypothesis testing for age-specific mortality data. Journal of Evolutionary Biology, 12, 430–439.

Ricklefs, R. E., & Scheuerlein, A. (2002). Biological implications of the Weibull and Gompertz models of aging. Journal of Gerontology, 57(2), B69-76.

Römer, G., Dahlgren, J. P., Salguero-Gómez, R., Stott, I. M., & Jones, O. R. (2024). Plant demographic knowledge is biased towards short-term studies of temperate-region herbaceous perennials. Oikos, Article e10250

Sæther, B.-E., Coulson, T., Grøtan, V., et al. (2013). How Life History Influences Population Dynamics in Fluctuating Environments. American Naturalist, 182(6), 743–759.

Salguero-Gómez, R., Jones, O. R., Archer, C. R., et al. (2015) The COMPADRE Plant Matrix Database: an open online repository for plant demography. Journal of Ecology, 103, 202-218.

Salguero-Gómez, R., Jones, O. R., Archer, C. R., et al. (2016). COMADRE: a global database of animal demography. Journal of Animal Ecology, 85(2), 371–384.

Salguero-Gómez, R., Jones, O. R., Jongejans, E., Blomberg, S. P., Hodgson, D. J., Mbeau-Ache, C., Zuidema, P. A., de Kroon, H., & Buckley, Y. M. (2016). Fast-slow continuum and reproductive strategies structure plant life-history variation worldwide. Proceedings of the National Academy of Sciences of the United States of America, 113(1), 230–235.

Siler, W. (1979) A competing-risk model for animal mortality. Ecology, 60, 750–757.

Stubben, C., Milligan, B., & Others. (2007). Estimating and analyzing demographic models using the popbio package in R. Journal of Statistical Software, 22(11), 1–23.

Takada, T., Kawai, Y., & Salguero-Gómez, R. (2018). A cautionary note on elasticity analyses in a ternary plot using randomly generated population matrices. Population Ecology, 60(1), 37–47.

Tuljapurkar, S. (1989). An uncertain life: demography in random environments. Theoretical Population Biology, 35(3), 227–294.

Vaupel, J., Manton, K. & Stallard, E. (1979) The impact of heterogeneity in individual frailty on the dynamics of mortality. Demography, 16, 439–454.

von Bertalanffy, L. (1957). Quantitative laws in metabolism and growth. The Quarterly Review of Biology, 32(3), 217–231.Caswell, H. (2001). Matrix Population Models. Sinauer Associates.

Tables

Table 1: Mathematical expressions for mortality and reproductive output models in `mpmsim`. In mortality models, h_x is the hazard function at age x, with parameters a_0, a_1, b_0, b_1, and c reflecting factors like aging rate and baseline risk. In the fecundity models, f_x is the reproductive output at age x, with parameters A, k, x_m, m, x_0, μ, σ, a, b, and c determining the timing of peak output, distribution, and overall levels. See `?mortality_model` and `?fecundity_model`.

Mortality models		
Gompertz	$h_x = b_0 \mathrm{e}^{\mathrm{b}_1 \mathrm{x}}$	Gompertz (1825), Pletcher (1999), Ricklefs & Scheuerlein (2002), Colchero et al. (2012)
Gompertz-Makeham	$h_x = b_0 \mathrm{e}^{\mathrm{b}_1 \mathrm{x}} + c$	Pletcher (1999), Colchero et al. (2012)
Exponential	$h_x = c$	Cox & Oakes (1984), Colchero et al. (2012)
Siler	$h_x = a_0 \mathrm{e}^{-\mathrm{a}_1 \mathrm{x}} + c + b_0 \mathrm{e}^{\mathrm{b}_1 \mathrm{x}}$	Siler (1979), Colchero et al. (2012)
Weibull	$h_x = b_0 \, b_1^{b_0} x^{(b_0-1)}$	Pinder *et al.* (1978), Ricklefs & Scheuerlein

		(2002), Colchero et al. (2012)
Weibull-Makeham	$h_x = b_0\, b_1^{b_0} x^{(b_0-1)} + c$	Colchero et al. (2012)
Fecundity models		
Logistic	$f_x = A/\left(1 + \exp\left(-k(x - x_m)\right)\right)$	
Step	$f_x = \begin{cases} A, x \geq m \\ A, x < m \end{cases}$	
Normal	$f_x = A \times \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$	
Hadwiger	$f_x = \frac{ab}{c}\left(\frac{c}{x}\right)^{\frac{3}{2}} \exp\left\{-b^2\left(\frac{c}{x} + \frac{x}{c} - 2\right)\right\}$	Hadwiger (1940) Peristera & Kostaki (2007)
von Bertalanffy	$f_x = A\left(1 - exp\left(-k(x - x_0)\right)\right)$	von Bertalanffy (1938)

Figure 1. Benchmarking results for the simulation time of `rand_leslie_set` (A) and `rand_lefko_set` (B) as a function of the number of matrices generated, using the code from Boxes 1 and 2. The elapsed time (in seconds) increases with the number of models requested, with `rand_lefko_set` running slower than `rand_leslie_set` for equivalent sample sizes. Each point represents the time taken to generate a given number of matrices. Benchmarking was performed on a MacBook running macOS (Sequoia 15.3) an Intel quad Core i5 2.40 GHz CPU, 16 GB of RAM, and R version 4.4.2.

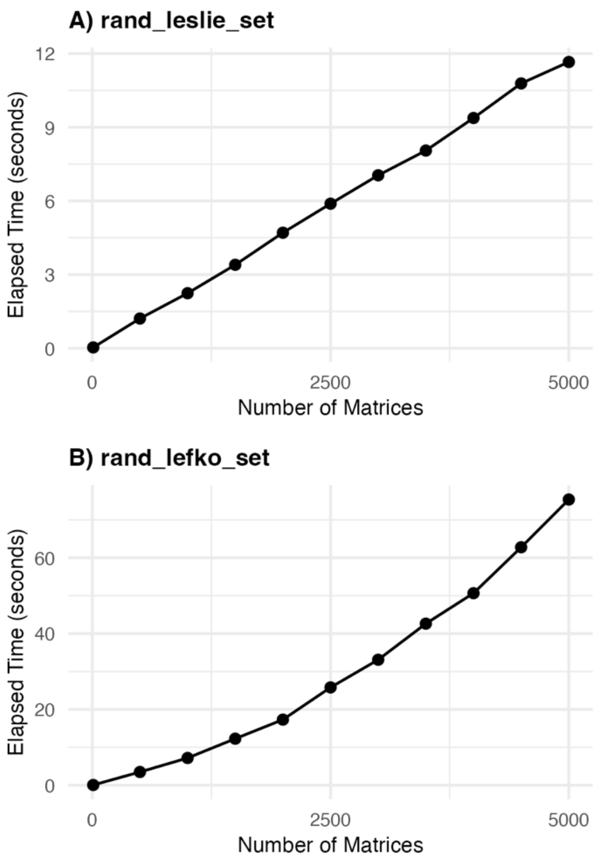
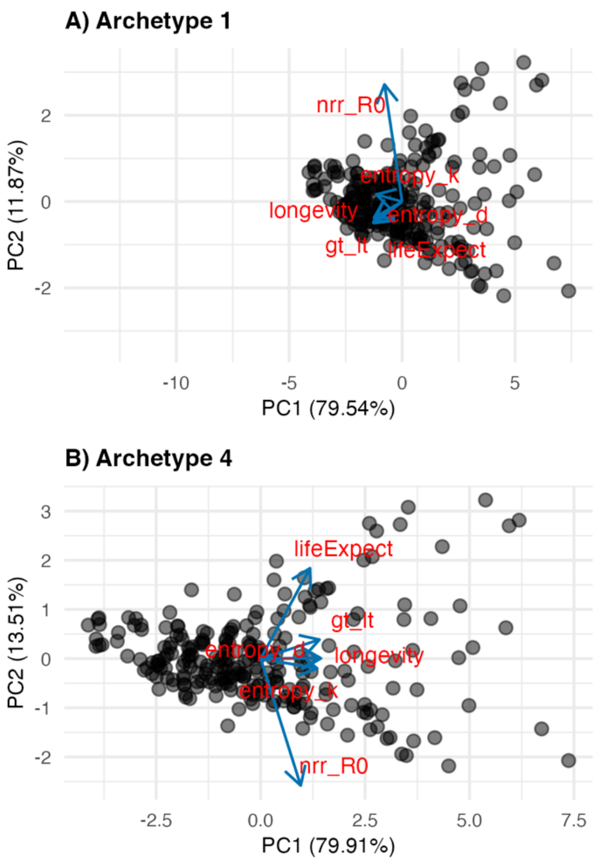


Figure 2. Principal Components Analysis (PCA) of simulated life history archetypes. The plots show the first two principal components (PC1 and PC2) for 250 simulated species in each archetype. Archetype 1 (A) allows both progression and retrogression, while Archetype 4 (B) has increasing survival and no retrogression. Points represent individual species, arrows show key life history trait loadings, and red labels indicate metrics such as longevity, net reproductive rate (nrr_R0), generation time (gt_lt), life expectancy (lifeExpect), and entropy measures (entropy_k, entropy_d). PC1 explains most variation (79.54% in Archetype 1, 79.91% in Archetype 4), with PC2 capturing additional structure (11.87% and 13.51%, respectively). Differences in trait loadings suggest broad-scale life history structuring may be shaped by demographic constraints.



Generating Leslie models

2024-10-12

Introduction

Leslie matrix models, named after Patrick Leslie who introduced them in the 1940s, are a type of matrix population model (MPM) used to describe the demography of age-structured populations. They are commonly used in studies of wildlife, conservation and evolutionary biology.

In a Leslie MPM, the square matrix is used to model discrete, age-structured population growth with a projection interval, most often representing years, as a time step. Each element in the matrix represents a transition probability between different age classes or indicates the average reproductive output (often referred to as fecundity in population biology and fertility in human demography) of the age class. The information in the MPM ($\mathbf{A}$) can be split into two submatrices ($\mathbf{U}$ and $\mathbf{F}$), representing survival/growth and reproduction, respectively. $\mathbf{A} = \mathbf{F} + \mathbf{C}$.

- **Survival Probabilities**: The subdiagonal (immediately below the main diagonal) of the MPM consists of survival probabilities. Each entry here shows the probability that an individual of one age class will survive to the next age class. These probabilities can be understood as an age trajectory of survival that can be modelled using a mathematical model describing how age-specific mortality changes with age.
- **Reproductive Output**: The first row of the MPM contains the reproductive output of each age class, representing the number of new individuals produced in each projection interval. This is often referred to as fecundity in ecological contexts.

All other entries in the MPM are typically zero, indicating that those transitions are impossible.

To project the population size and structure through time, the MPM is multiplied by a vector that represents the current population structure (number of individuals in each age class). This process results in a new vector that shows the predicted structure of the population in the next time step. This calculation can be iterated repeatedly to project population and structure through time.

Leslie matrices are useful for studying population dynamics under different scenarios, such as changes in survival rates, fecundity rates, or management strategies. They have been widely applied in both theoretical and applied ecology.

Aims

The aim of this vignette is to demonstrate how to simulate Leslie matrix population models (MPMs) using functional forms for mortality and reproduction. This simulation is useful for various purposes, including:

- Investigating the influence of senescence parameters on population dynamics.
- Generating MPMs based on empirical parameter estimates of mortality and reproduction from the literature.
- Creating MPMs with specific properties for educational and research purposes.

In the following sections, this document will:

1) Explain the basics of mortality and reproduction trajectories.
2) Show how to produce life tables reflecting trajectories of mortality and reproduction.
3) Show how to produce MPMs from these life tables.
4) Show how to generate sets of many MPMs based on defined mortality and reproduction characteristics.

Preparation

Before beginning, users will need to load the required packages.

```
library(mpmsim)
library(dplyr)
library(Rage)
library(ggplot2)
library(Rcompadre)
```

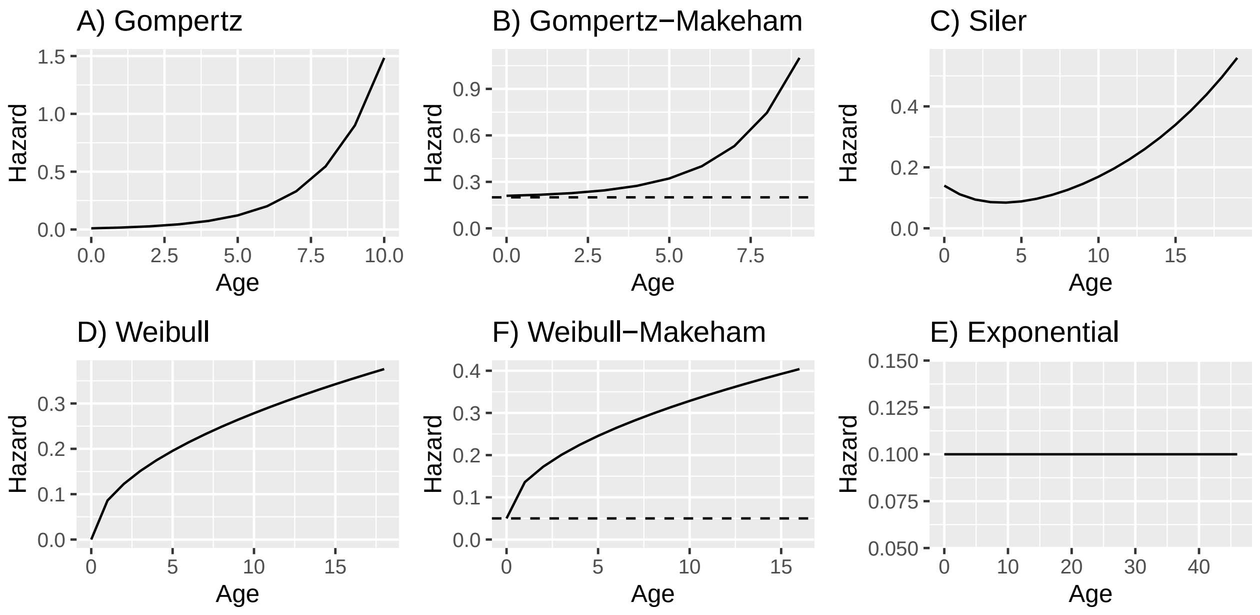
1. Functional forms of mortality and reproduction

There are numerous published and well-used functional forms used to describe how mortality risk (hazard) changes with age. The `model_mortality` function (and its synonym `model_survival`) handles 6 of these models: Gompertz, Gompertz-Makeham, Weibull, Weibull-Makeham, Siler and Exponential.

In a nutshell:

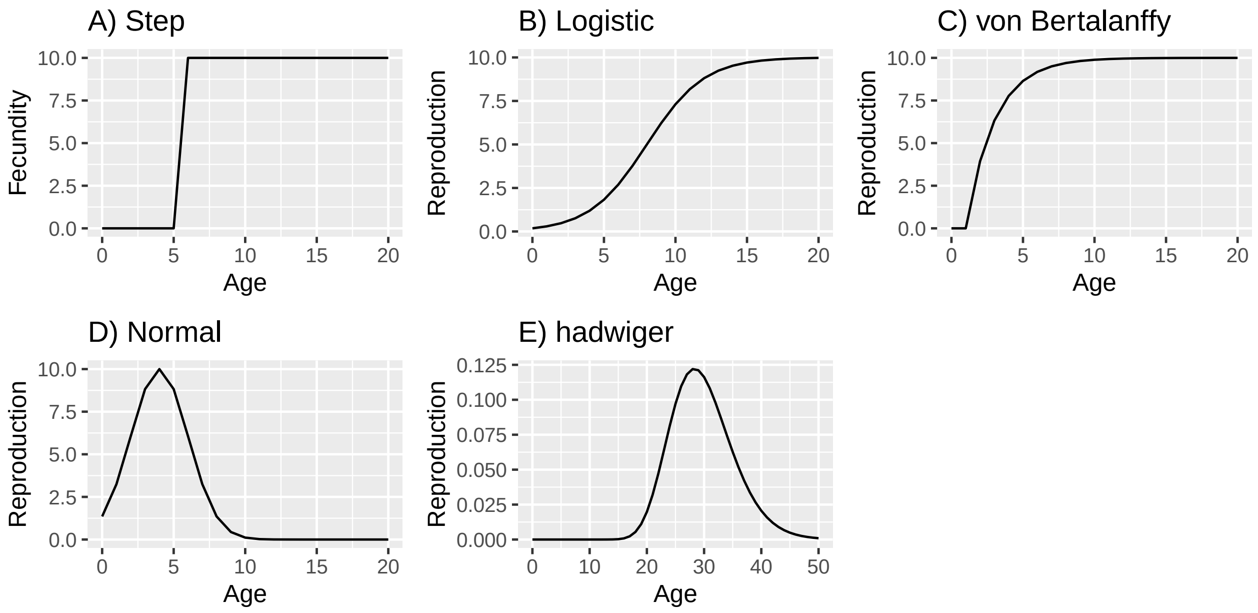
- Gompertz: A mortality rate that increases exponentially with age. $h_x = b_0 \mathrm{e}^{b_1 x}$
- Gompertz-Makeham: A mortality rate that increases exponentially with age, with an additional age-independent constant mortality. $h_x = b_0 \mathrm{e}^{b_1 x} + c$
- Weibull: A mortality rate that scales with age, increasing at a rate that can either accelerate or decelerate, depending on the parameters of the model. $h_x = b_0 b_1 (b_1 x)^{(b_0 - 1)}$
- Weibull-Makeham: as the basic Weibull, but with an additional age-independent constant mortality. $h_x = b_0 b_1 (b_1 x)^{(b_0 - 1)} + c$
- Siler: A mortality model that separates mortality rates into two age-related components — juvenile mortality, which declines exponentially with age and adult mortality, which increases exponentially. $h_x = a_0 \mathrm{e}^{-a_1 x} + c + b_0 \mathrm{e}^{b_1 x}$
- Exponential: Constant mortality that is unchanging with age. $h_x = c$

These are illustrated below.



In addition to these functional forms of mortality, there are, of course, functional forms that have been used to model reproductive output. The `model_fecundity` function (and its synonyms `model_fedundity` and `model_reproduction`) handles five types: logistic, step, von Bertalanffy, normal and Hadwiger. Some of these models originate from human demography, where *fertility* is used for realised reproductive output and *fecundity* refers to reproductive potential. In ecology and population biology, however, *fecundity* typically describes actual reproductive output. Since `mpmsim` is designed for population biologists, we will use the terms *fecundity*, or simply *reproduction/reproductive output*.

- Logistic: Reproductive output initially increases rapidly with age then slows to plateau as it approaches a maximum value. $f_x = A/(1 + exp(-k(x - x_m)))$
- Step: Reproductive output is initially zero, then jumps to a particular level at a specified age, after which it remains constant. $f_x = \begin{cases} A, x \geq m \\ A, x < m \end{cases}$
- von Bertalanffy: This model is often used in growth dynamics but has been adapted for reproduction to describe changes over age or time following a logistic growth form not limited by a strict upper asymptote. It shows how reproductive output might increase and then decrease, following a sigmoid curve. $f_x = A(1 - exp(-k(x - x_0)))$
- Normal : Reproductive output is modelled as normal distribution to describe how reproductive output increases, peaks, and then decreases in a bell curve around a mean age of reproductive capacity. $f_x = A \times \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$
- Hadwiger: The outcomes of this model are qualitatively similar to the normal distribution. $f_x = \frac{ab}{C}\left(\frac{C}{x}\right)^{\frac{3}{2}} \exp\left\{-b^2\left(\frac{C}{x} + \frac{x}{C} - 2\right)\right\}$



Collectively, these mortality and fecundity functions offer a large scope for modelling the variety of demographic trajectories apparent across the tree of life.

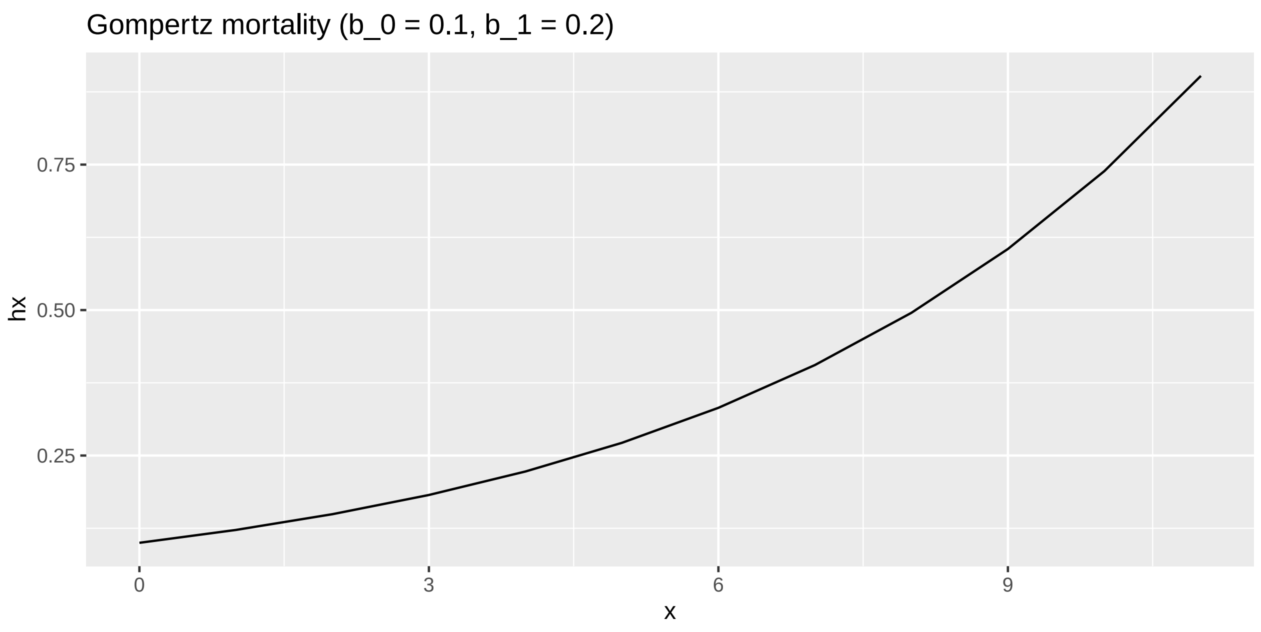
2. Trajectories of mortality and reproductive output, and production of life tables

To obtain a trajectory of mortality, users can use the `model_mortality` function, which takes as input the parameters of a specified mortality model. The output of this function is a standard life table `data.frame` including columns for age (`x`), age-specific hazard (`hx`), survivorship (`lx`), age-specific probability of death and survival (`qx` and `px`). By default, the life table is truncated at the age when the survivorship function declines below 0.01 (i.e. when only 1% of individuals in a cohort would remain alive).

```
(lt1 <- model_mortality(params = c(b_0 = 0.1, b_1 = 0.2), model = "Gompertz"))
#>     x        hx         lx        qx        px
#> 1   0 0.1000000 1.00000000 0.1051240 0.8948760
#> 2   1 0.1221403 0.89487598 0.1268617 0.8731383
#> 3   2 0.1491825 0.78135045 0.1526972 0.8473028
#> 4   3 0.1822119 0.66204041 0.1832179 0.8167821
#> 5   4 0.2225541 0.54074272 0.2190086 0.7809914
#> 6   5 0.2718282 0.42231542 0.2606027 0.7393973
#> 7   6 0.3320117 0.31225886 0.3084127 0.6915873
#> 8   7 0.4055200 0.21595427 0.3626343 0.6373657
#> 9   8 0.4953032 0.13764186 0.4231275 0.5768725
#> 10  9 0.6049647 0.07940180 0.4892807 0.5107193
#> 11 10 0.7389056 0.04055203 0.5598781 0.4401219
#> 12 11 0.9025013 0.01784784 0.6330059 0.3669941
```

It can be useful to explore the impact of parameters on the mortality hazard (`hx`) graphically, especially for users who are unfamiliar with the chosen models.

```
ggplot(lt1, aes(x = x, y = hx)) +
  geom_line() +
  ggtitle("Gompertz mortality (b_0 = 0.1, b_1 = 0.2)")
```

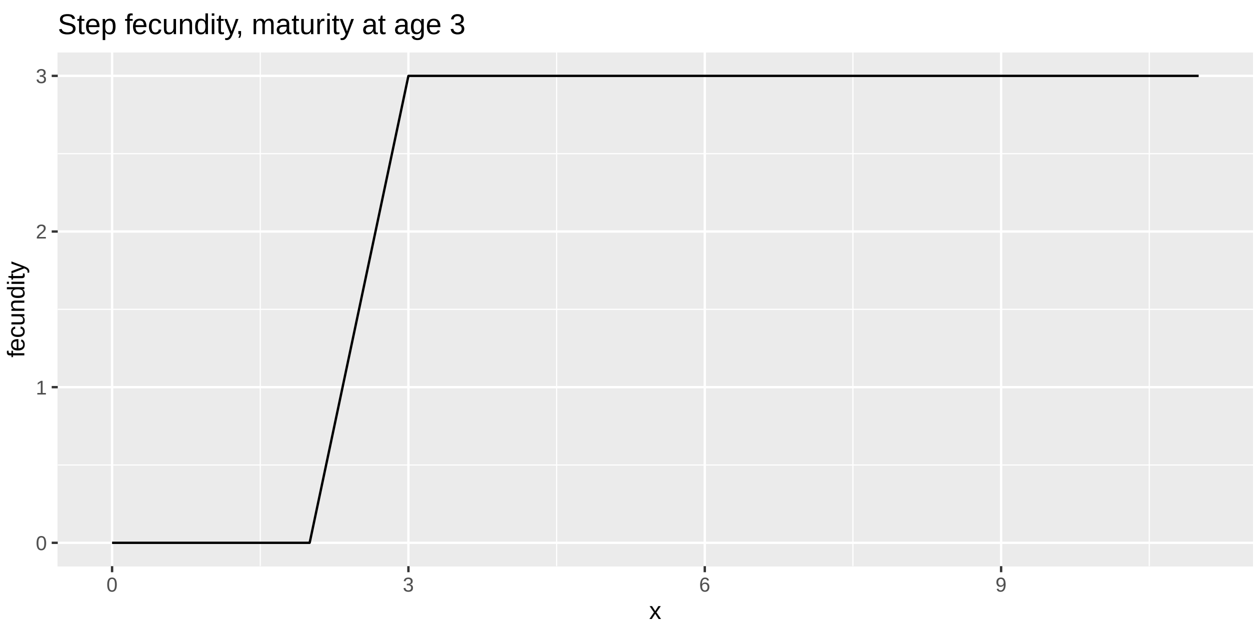


The `model_fecundity` function is similar to the `model_mortality` function, as it has arguments for the type of fecundity model, and its parameters. However, the output of the `model_fecundity` function is a vector of reproductive output values rather than a `data.frame`. This allows us to add a fecundity column (`fecundity`) directly to the life table produced earlier, as follows:

```
(lt1 <- lt1 |>
  mutate(fecundity = model_fecundity(
    age = x, params = c(A = 3),
    maturity = 3,
    model = "step"
  )))
#>     x        hx         lx        qx        px fecundity
#> 1   0 0.1000000 1.00000000 0.1051240 0.8948760         0
#> 2   1 0.1221403 0.89487598 0.1268617 0.8731383         0
#> 3   2 0.1491825 0.78135045 0.1526972 0.8473028         0
#> 4   3 0.1822119 0.66204041 0.1832179 0.8167821         3
#> 5   4 0.2225541 0.54074272 0.2190086 0.7809914         3
#> 6   5 0.2718282 0.42231542 0.2606027 0.7393973         3
#> 7   6 0.3320117 0.31225886 0.3084127 0.6915873         3
#> 8   7 0.4055200 0.21595427 0.3626343 0.6373657         3
#> 9   8 0.4953032 0.13764186 0.4231275 0.5768725         3
#> 10  9 0.6049647 0.07940180 0.4892807 0.5107193         3
#> 11 10 0.7389056 0.04055203 0.5598781 0.4401219         3
#> 12 11 0.9025013 0.01784784 0.6330059 0.3669941         3
```

Again, it can be useful to plot the relevant data to visualise it.

```
ggplot(lt1, aes(x = x, y = fecundity)) +
  geom_line() +
  ggtitle("Step fecundity, maturity at age 3")
```



3. From life table to MPM

Users can now turn these life tables, containing age-specific survival and reproductive trajectories, into Leslie matrices using the `make_leslie_mpm` function. These MPMs can be large or small depending on the maximum life span of the population: as mentioned above, the population is modelled until less than 1% of a cohort remains alive.

```
make_leslie_mpm(lifetable = lt1)
#>          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
#>  [1,] 0.000000 0.0000000 0.0000000 3.0000000 3.0000000 3.0000000 3.0000000
#>  [2,] 0.894876 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [3,] 0.000000 0.8731383 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [4,] 0.000000 0.0000000 0.8473028 0.0000000 0.0000000 0.0000000 0.0000000
#>  [5,] 0.000000 0.0000000 0.0000000 0.8167821 0.0000000 0.0000000 0.0000000
#>  [6,] 0.000000 0.0000000 0.0000000 0.0000000 0.7809914 0.0000000 0.0000000
#>  [7,] 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 0.7393973 0.0000000
#>  [8,] 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.6915873
#>  [9,] 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#> [10,] 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#> [11,] 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#> [12,] 0.000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>           [,8]      [,9]     [,10]     [,11]     [,12]
#>  [1,] 3.0000000 3.0000000 3.0000000 3.0000000 3.0000000
#>  [2,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [3,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [4,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [5,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [6,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [7,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [8,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
#>  [9,] 0.6373657 0.0000000 0.0000000 0.0000000 0.0000000
#> [10,] 0.0000000 0.5768725 0.0000000 0.0000000 0.0000000
#> [11,] 0.0000000 0.0000000 0.5107193 0.0000000 0.0000000
#> [12,] 0.0000000 0.0000000 0.0000000 0.4401219 0.3669941
```

4. Producing sets of MPMs based on defined model characteristics

It is sometimes desirable to create large numbers of MPMs with particular properties in order to test hypotheses. For Leslie MPMs, this can be implemented in a straightforward way using the function `rand_leslie_set`. This function generates a set of Leslie MPMs based on defined mortality and fecundity models, and using model parameters that are randomly drawn from specified distributions. For example, users may wish to generate MPMs for Gompertz models to explore how rate of senescence influences population dynamics.

Users must first set up a data frame describing the distribution from which parameters will be drawn at random. The data frame has a number of rows equal to the number of parameters in the model, and two values to describe the distribution. In the case of a uniform distribution, these are the minimum and maximum parameter values, respectively and with a normal distribution they represent the mean and standard deviation. The parameters should be entered in the order they appear in the model equations (see `?model_mortality`), with the exact order depending on the chosen mortality model.

For the Gompertz-Makeham model: $h_x = b_0 \mathrm{e}^{b_1 x} + c$

The `output` argument defines the output as one of six types (`Type1` through `Type6`). These outputs include `CompadreDB` objects or `list` objects, and the MPMs can be split into the component submatrices (**U** and **F**, where the MPM, $\mathbf{A} = \mathbf{U} + \mathbf{F}$). In the special case `Type6` the outputs are provided as a `list` of life tables rather than MPMs. If the output is set as a `CompadreDB` object, the mortality and fecundity model parameters used to generate the MPM are included as metadata.

The following example illustrates the production of 50 Leslie MPMs output to a `CompadreDB` object based on the Gompertz-Makeham mortality model and a step fecundity model with maturity beginning at age 0. An optional argument, `scale_output = TRUE` will scale the fecundity in the output MPMs to ensure that population growth rate is lambda. The scaling algorithm multiplies the fecundity part of the MPM (the **F** submatrix) by a simple scaling factor to ensure the population growth rate is 1 while maintaining the shape (but not the magnitude) of the fecundity trajectory. This should be used with care: The desirability of such a manipulation strongly depends on the use the MPMs are put to.

```r
mortParams <- data.frame(
  minVal = c(0, 0.01, 0.1),
  maxVal = c(0.05, 0.15, 0.2)
)

fecundityParams <- data.frame(
  minVal = 2,
  maxVal = 10
)

maturityParam <- c(0, 0)

(myMatrices <- rand_leslie_set(
  n_models = 50,
  mortality_model = "GompertzMakeham",
  fecundity_model = "step",
  mortality_params = mortParams,
  fecundity_params = fecundityParams,
  fecundity_maturity_params = maturityParam,
  dist_type = "uniform",
  output = "Type1"
))
#> A COM(P)ADRE database ('CompadreDB') object with ?? SPECIES and 50 MATRICES.
#>
#> # A tibble: 50 x 8
#>    mat        mortality_model      b_0     b_1      C fecundity_model      A
#>    <list>     <chr>              <dbl>   <dbl>  <dbl> <chr>            <dbl>
#>  1 <CompdrMt> gompertzmakeham 0.0457   0.141  0.129 step              8.64
#>  2 <CompdrMt> gompertzmakeham 0.0321   0.0827 0.174 step              3.08
#>  3 <CompdrMt> gompertzmakeham 0.0328   0.109  0.146 step              7.75
#>  4 <CompdrMt> gompertzmakeham 0.0467   0.0458 0.146 step              9.52
#>  5 <CompdrMt> gompertzmakeham 0.0489   0.0264 0.147 step              6.48
#>  6 <CompdrMt> gompertzmakeham 0.0452   0.0294 0.199 step              9.57
#>  7 <CompdrMt> gompertzmakeham 0.00412  0.0820 0.139 step              9.25
#>  8 <CompdrMt> gompertzmakeham 0.0223   0.127  0.174 step              8.49
#>  9 <CompdrMt> gompertzmakeham 0.0194   0.106  0.100 step              8.66
#> 10 <CompdrMt> gompertzmakeham 0.000367 0.0391 0.191 step              6.89
#> # i 40 more rows
#> # i 1 more variable: fecundity_scaling <dbl>
```

The function operates quite fast. For example, on an older MacBook (3.10GHz Intel with 4 cores), it takes 17 seconds to generate 5000 MPMs with the parameters mentioned above.

As an aid to assessing the simulation, users can produce a simple summary of the MPMs using the `summarise_mpms` function. Note, though, that this only works with `CompadreDB` outputs. In this case, because we are working with Leslie MPMs, the dimension of the MPMs is indicative of the maximum age reached by individuals in the population.

```
summarise_mpms(myMatrices)
#> Summary of matrix dimension:
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   15.00   18.25   22.00   22.98   26.00   39.00
#> Summary of lambda values:
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   2.845   6.010   7.970   7.557   9.150  10.672
#>
#> Summary of maximum F values:
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   2.002   5.157   7.142   6.720   8.327   9.788
#>
#> Summary of maximum U values:
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.7829  0.8159  0.8347  0.8365  0.8545  0.8925
#>
#> Summary of minimum non-zero U values:
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.5715  0.6881  0.7650  0.7514  0.8160  0.8814
```

After producing the output as a `CompadreDB` object, the matrices can be accessed using functions from the `RCompadre` R package. For example, to get the A matrix, or the U/F submatrices users can use the `matA`, `matU` or `matF` functions. The following code illustrates how to rapidly calculate population growth rate for all of the matrices.

```
# Obtain the matrices
x <- matA(myMatrices)

# Calculate lambda for each matrix
sapply(x, popdemo::eigs, what = "lambda")
#>  [1]  9.480062  3.889616  8.587381 10.343567  7.303660 10.356146 10.112353
#>  [8]  9.308862  9.549256  7.720332 10.672096  7.928430  5.604869  8.872518
#> [15]  6.966830  7.339943  9.468459  3.993036  8.631979  2.871674  8.012464
#> [22]  3.567593  2.844529  8.688251  7.778629  5.968972  8.664013  9.523154
#> [29]  7.123514  9.482587  7.556071 10.495633  7.644149  7.441366  7.792883
#> [36]  9.092281 10.412818  5.422186  9.018497  8.325535  4.386485  5.377892
#> [43]  6.131147  3.940014  8.952296  4.907484  9.138780  9.154187  3.317446
#> [50]  8.668165
```

Users can examine the vignettes for the `Rcompadre` and `Rage` packages for additional insight into other potential operations with the `compadreDB` object.

Conclusion

This vignette demonstrated how to generate Leslie matrices using functional forms of mortality and fecundity, allowing users to simulate virtual species with varied life histories. These matrices can be used to explore how life history or parameter differences affect population dynamics, supporting various research and educational applications.

Generating Lefkovitch models

2024-10-12

Introduction

Lefkovitch matrix population models (MPMs) were introduced by Leonard Lefkovitch in his 1965 paper, "*The Study of Population Growth in Organisms Grouped by Stages*", published in *Biometrics.*. This paper extended the concept of Leslie MPMs, which are structured by age, to stage-structured populations, providing a framework that has been widely used in ecology, evolution and conservation studies.

In a Lefkovitch MPM, the square matrix is used to model population growth across discrete projection intervals, typically representing years. Each matrix element represents either a transition probability between different stages or the reproductive output of a stage across the projection interval. The MPM can be divided into submatrices: one for survival/growth (the **U** matrix), one for sexual reproduction (the **F** matrix) and one for asexual reproduction (the **C** matrix), where $\mathbf{A} = \mathbf{U} + \mathbf{F} + \mathbf{C}$. Occasionally, these reproduction matrices are lumped together as a reproduction matrix, **R** (i.e. $\mathbf{R} = \mathbf{F} + \mathbf{C}$). Reproduction is often termed fecundity in the population biology literature.

The elements of the **U** matrix represent survival or growth from stage-to-stage between time steps. Therefore the column sums of the **U** submatrix cannot exceed 1. The reproductive output elements in the **F** and **C** (or **R**) submatrices do not have an upper bound and indicate the number of new individuals each stage can produce in each time interval.

Zero entries in the matrices indicate that those transitions do not occur.

To project population size and structure over time, the MPM is multiplied by a vector representing the current population structure (number of individuals in each stage). This results in a new vector that shows the predicted population structure for the next time step. This process can be repeated to project population dynamics over multiple time steps.

Lefkovitch models are useful for studying population dynamics under different scenarios, such as changes in survival or reproductive rates, or different management strategies. They have broad applications in both theoretical and applied ecology.

Aims

The purpose of this vignette is to illustrate how to simulate stage-based (Lefkovitch) MPMs based on defined life history archetypes. There are several reasons why one would want to do this, including, but not limited to:

- Exploring how life history or life cycle structure influences population dynamics.
- Generating MPMs with defined life cycle properties for teaching purposes.

In the following sections, this vignette will:

1) Explain how life cycles can be categorised into Archetypes.
2) Show how to generate a random Lefkovitch MPM based on an archetype.
3) Show how to rapidly produce sets of many random MPMs.
4) Show how to constrain the MPMs by matrix properties.

Preparation

Before beginning, users will need to load the required packages.

```r
library(mpmsim)
library(dplyr)
library(Rage)
library(ggplot2)
library(Rcompadre)
```

1. Life cycle archetypes and generating an MPM

In stage-based (Lefkovitch) matrix population models (MPMs), different life cycle types can be represented by various structural forms of the matrices. These life cycle types can be captured using different life history archetypes, which define the transitions between stages and the survival probabilities in the population.

The life history archetypes, based on Takada et al. (2018), are as follows:

- Archetype 1: All elements are positive, meaning transitions from/to any stage are possible. This model represents a life history where individuals can progress and retrogress rapidly.
- Archetype 2: Similar to Archetype 1, but with the survival probability increasing monotonically as individuals advance to later stages. This model also allows for rapid progression and retrogression but with more realistic stage-specific survival probabilities.
- Archetype 3: Positive non-zero elements for survival are only allowed on the diagonal and lower sub-diagonal of the matrix. This model represents a life cycle where retrogression is not allowed, and progression can only happen to the immediately larger/more developed stage (slow progression, e.g., trees).
- Archetype 4: Similar to Archetype 3 but with the additional assumption that stage-specific survival increases as individuals increase in size/developmental stage.

In all these archetypes, fecundity is placed in the top row of the matrix. In Takada et al.'s paper, fecundity was always placed in the top right of the MPM, meaning that only the "last" stage of the life cycle reproduced. This approach can be relaxed to allow reproduction from any stage.

2. Generate a random Lefkovitch MPM based on an archetype

In `mpmsim` the function `rand_lefko_mpm` can be used to generate a random MPM that conforms to one of the above four life cycle archetypes. The function allows for the generation of random MPMs based on these archetypes, with survival and growth (the **U** matrix) based on draws from a Dirichlet distribution to ensure biological plausibility. The Dirichlet distribution is used to draw survival and growth values because it ensures that the sum of the probabilities for each stage is equal to 1, which is necessary for biologically realistic models. The function allows users to specify a wide range of reproductive output scenarios , offering flexibility in how fecundity is modelled across stages.

The function is straightforward. In the following example, I create a three-stage MPM based on Archetype 2. I set fecundity, arbitrarily, to be 5. By default, if only a single value is given, this is placed in the top-right hand corner of the **F** matrix. Also, by default, all fecundity is assumed to be sexual.

```r
rand_lefko_mpm(n_stages = 3, fecundity = 5, archetype = 2)
#> $mat_A
#>             [,1]       [,2]       [,3]
#> [1,] 0.005877166 0.10766181 5.72412770
#> [2,] 0.509913175 0.03602856 0.06736799
#> [3,] 0.287840153 0.74138349 0.19955576
#>
#> $mat_U
#>             [,1]       [,2]       [,3]
#> [1,] 0.005877166 0.10766181 0.72412770
#> [2,] 0.509913175 0.03602856 0.06736799
#> [3,] 0.287840153 0.74138349 0.19955576
#>
#> $mat_F
#>      [,1] [,2] [,3]
#> [1,]    0    0    5
#> [2,]    0    0    0
#> [3,]    0    0    0
```

To introduce variability in fecundity, users can provide reproductive output as a list of two matrices, with numeric values of the same dimensions as `n_stages`, representing the lower and upper limits of mean fecundity for the entire matrix model. Reproductive output values are then drawn from a uniform distribution between the two values. Users should use 0 for both lower and upper limits in cases with no fecundity.

The following code provides an example:

```
lower_reprod <- matrix(c(
  0, 0, 0,
  0, 0, 0,
  0, 0, 0
), nrow = 3, ncol = 3, byrow = TRUE)
upper_reprod <- matrix(c(
  0, 4, 20,
  0, 0, 0,
  0, 0, 0
), nrow = 3, ncol = 3, byrow = TRUE)

rand_lefko_mpm(
  n_stages = 3, fecundity = list(lower_reprod, upper_reprod),
  archetype = 2
)
#> $mat_A
#>           [,1]      [,2]       [,3]
#> [1,] 0.0873477 2.6402006 15.2043937
#> [2,] 0.4082913 0.1851252  0.5516964
#> [3,] 0.3924441 0.1736190  0.1107596
#>
#> $mat_U
#>           [,1]      [,2]      [,3]
#> [1,] 0.0873477 0.5799473 0.3249008
#> [2,] 0.4082913 0.1851252 0.5516964
#> [3,] 0.3924441 0.1736190 0.1107596
#>
#> $mat_F
#>      [,1]     [,2]     [,3]
#> [1,]    0 2.060253 14.87949
#> [2,]    0 0.000000  0.00000
#> [3,]    0 0.000000  0.00000
```

3. Generate sets of Lefkovitch matrices

It is sometimes desirable to create large numbers of MPMs with particular properties in order to test hypotheses. For stage-based (Lefkovitch) MPMs, this can be implemented using the `rand_lefko_set` function. This function acts as a wrapper for the previously described function and generates a set of Lefkovitch MPMs based on a defined life cycle archetype and specified reproductive output. For example, users may wish to generate MPMs for different life history archetypes to explore how life cycle structure may influence population dynamics. By specifying the number of stages, fecundity values, and archetypes, users can produce MPMs that are tailored to their specific research needs. This capability is useful for exploring the effects of life history traits on population dynamics, testing ecological and evolutionary hypotheses, and for teaching purposes.

The function returns either a `list` or a `CompadreDB` object depending on the `output` argument. If the output is set as a `CompadreDB` object, the archetype of the MPM is included as a column of metadata.

The following code shows how users can generate 100 matrices in a `CompadreDB` object.

```
myMatrices <- rand_lefko_set(
  n = 100, n_stages = 3, fecundity = 12,
  archetype = 4, output = "Type1"
)
```

After producing the output as a `CompadreDB` object, the matrices can be accessed using functions from the `RCompadre` R package. For example, to get the **A** matrix, or the **U**/**F** submatrices users can use the `matA`, `matU` or `matF` functions. The following code illustrates how to rapidly calculate population growth rate for all of the matrices.

```
# Obtain the matrices
x <- matA(myMatrices)

# Calculate lambda for each matrix
lambdaVals <- sapply(x, popdemo::eigs, what = "lambda")
summary(lambdaVals)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.7378  1.1395  1.4965  1.4653  1.7106  2.1926
```

Users can examine the vignettes for the `Rcompadre` and `Rage` packages for additional insight into other potential operations with the `compadreDB` object.

4. Constraining the output matrices

Critically, users can impose constraints on the “acceptable” properties of these randomly generated MPMs. For example, in some analyses, it may be desirable to generate MPMs where the population growth rate is constrained to values near 1.

This is handled by the `constraint` argument, which takes a data frame specifying the criteria for acceptable MPMs. The data frame must have four columns: `fun`, `arg`, `lower`, and `upper`. The `fun` column contains the name of the function that calculates the metric to be constrained (e.g., `eigs`, from the `popdemo` package). The `arg` column specifies any additional argument that the function requires (e.g., `what = "lambda"` for the `eigs` function), using `NA` if no additional argument is needed. The `lower` and `upper` columns set the bounds of the acceptable range for the metric.

Here’s an example of how to use the constraint argument to ensure that the generated MPMs have a population growth rate (lambda) between 0.9 and 1.1.

```r
library(popdemo)

constrain_df <- data.frame(
  fun = "eigs", arg = "lambda", lower = 0.9, upper = 1.1
)

myMatrices <- rand_lefko_set(
  n = 100, n_stages = 3, fecundity = 12, constraint = constrain_df,
  archetype = 4, output = "Type1"
)
```

We can check that it has worked by examining the matrices.

```r
# Obtain the matrices
x <- matA(myMatrices)

# Calculate lambda for each matrix
lambdaVals <- sapply(x, popdemo::eigs, what = "lambda")
summary(lambdaVals)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.9019  0.9612  1.0148  1.0115  1.0635  1.0984
```

Conclusion

This vignette has provided a comprehensive guide to generating Lefkovitch matrix population models (MPMs) based on life history archetypes. By using the `rand_lefko_mpm` and `rand_lefko_set` functions, users can create individual MPMs or large sets of MPMs tailored to specific research needs. The ability to impose constraints on these models allows for precise control over their properties, ensuring that generated MPMs meet defined criteria, such as specific population growth rates.

The flexibility and power of these functions facilitate the exploration of population dynamics under various scenarios, aiding in hypothesis testing in studies of population biology and life history theory. Additionally, tight integration with the `RCompadre` package facilitates the use of generated models, enhancing their utility in both theoretical and applied ecological research.

Sampling error and its propagation

2024-10-12

Introduction

Uncertainty in the individual elements of a matrix population model (MPM) can propagate, affecting the accuracy of metrics derived from the model, such as population growth rate, generation time, etc.

One approach to estimate this uncertainty is parametric bootstrapping, which generates a sampling distribution for the matrix model based on assumptions about the underlying demographic processes and uncertainties in individual matrix elements. For example, reproductive output can be modelled as a Poisson-distributed process, suitable for count-based data, while survival can be represented by a binomial distribution, reflecting the probability of individual survival.

The `compute_ci` function estimates a 95% confidence interval (95% CI) for any MPM-derived metric by generating a sampling distribution through resampling based on the given assumptions. The 95% CI is derived from the 2.5th and 97.5th percentiles of this distribution, where a narrower interval indicates greater precision.

The width and shape of the sampling distribution are influenced by several factors, including the sample size used for estimating matrix elements, the matrix model's structure, the assumptions underlying the `compute_ci` function, and the distribution of uncertainties across matrix elements. To accurately assess the precision of MPM estimates, it is necessary to consider these factors when interpreting the results.

The following examples show how to use these functions.

Aims

The purpose of this vignette is to illustrate how to use `mpmsim` to assess and estimate sampling error in MPMs and how this uncertainty propagates into derived metrics. This approach is useful for several reasons, including:

- Estimating confidence intervals (CIs) for key demographic metrics, such as population growth rate using parametric bootstrapping methods.
- Exploring the impact of sample size on the precision of estimates for population growth rate and other metrics derived from MPMs.
- Evaluating the propagation of uncertainty across different matrix elements and submatrices (survival/growth vs. reproduction) in MPMs.

Estimate 95% Confidence Intervals

We can estimate the 95% CI for any metric derived from a matrix population model. In this example, we focus on the population growth rate, λ.

Consider a matrix model $\mathbf{A}$, which is composed of submatrices $\mathbf{U}$ (survival/growth) and $\mathbf{F}$ (sexual reproduction), such that $\mathbf{A} = \mathbf{U} + \mathbf{F}$.

The methods require that the matrix model be split into its component submatrices because the underlying processes are governed by distributions with different statistical properties: individual survival is treated as a binary process (0 = dies, 1 = survives), whereas individual reproduction follows a Poisson distribution.

In this example, the matrix is simple, with only the top-right element representing reproduction, while all other elements represent survival or growth.

$$\mathbf{A} = \begin{bmatrix} 0.1 & 5.0 \\ 0.2 & 0.4 \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} 0.1 & 0.0 \\ 0.2 & 0.4 \end{bmatrix}$$

$$\mathbf{F} = \begin{bmatrix} 0.0 & 5.0 \\ 0.0 & 0.0 \end{bmatrix}$$

We can enter these matrices into R as follows, first entering the **U** and **F** matrices, and then summing them to get the **A** matrix.

```r
matU <- matrix(c(
  0.1, 0.0,
  0.2, 0.4
), byrow = TRUE, nrow = 2)

matF <- matrix(c(
  0.0, 5.0,
  0.0, 0.0
), byrow = TRUE, nrow = 2)

matA <- matU + matF
```

The estimated population growth rate (λ) can be calculated using the `eigs` function from the `popdemo` package like this:

```r
popdemo::eigs(matA, what = "lambda")
#> [1] 1.261187
```

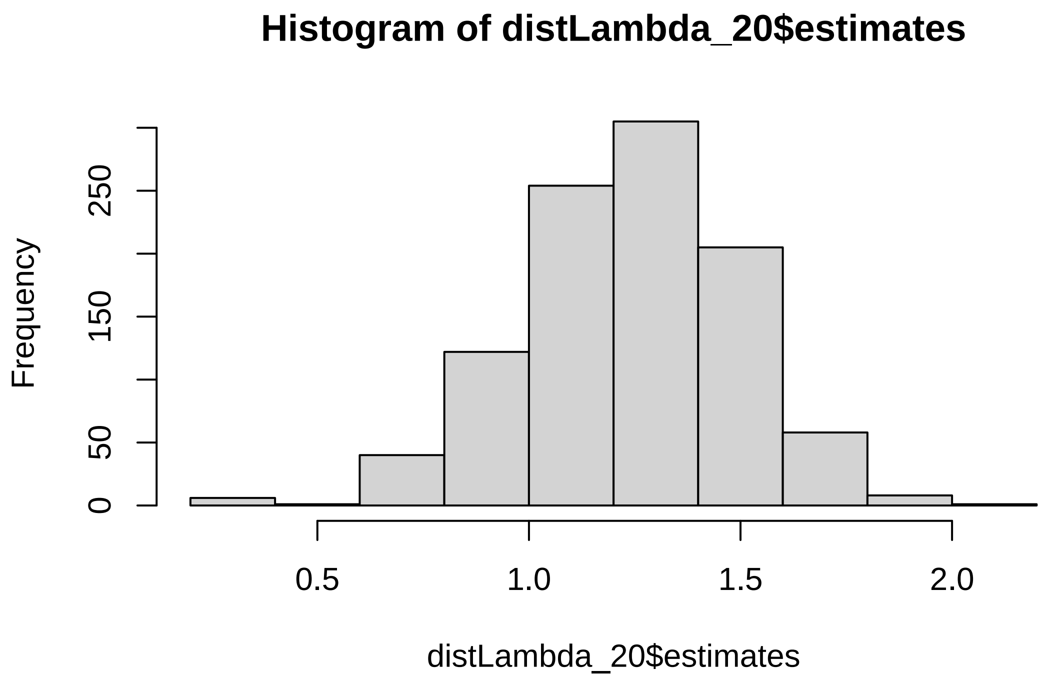
We can now estimate the 95% CI for this estimate, based on a knowledge of the sample size(s) used to parameterise the MPM.

If the sample size used to estimate each element of the matrix is `20` individuals, we can estimate the 95% CI for λ using the `compute_ci` function. This function requires several arguments: `mat_U` and `mat_F` represent the survival/growth matrix and reproductive output matrix respectively, and `sample_size` specifies the number of individuals used to estimate each element (in this case, `20`). The argument `FUN` defines the function to be applied to the resulting **A** matrix to calculate the desired metric.

```r
compute_ci(
  mat_U = matU, mat_F = matF, sample_size = 20,
  FUN = popdemo::eigs, what = "lambda"
)
#>      2.5%     97.5%
#> 0.7097788 1.7020301
```

We can examine the sampling distribution of these λ estimates estimates by using the argument `dist.out = TRUE`.

```r
distLambda_20 <- compute_ci(
  mat_U = matU, mat_F = matF,
  sample_size = 20, FUN = popdemo::eigs, what = "lambda",
  dist.out = TRUE
)
hist(distLambda_20$estimates)
```



Sample sizes that vary across the MPM

In the above example, it is assumed that the sample size used to make the parameter estimates (i.e. each element of the matrix model) was the same throughout the model. However, sample size might vary across different parts of the matrix or submatrices due to variations in data availability or biological processes. For example, data on survival and growth (represented in the **U** matrix) might be more abundant because these processes can often be tracked more easily in field studies. In contrast, reproductive output data (represented in the **F** matrix) may be harder to collect, especially for species with complex reproductive cycles, leading to smaller sample sizes. Additionally, environmental factors or study limitations can result in unequal sampling efforts across different life stages, contributing to varying sample sizes in the matrix elements.

To account for this, the `compute_ci` function allows flexibility in specifying sample size, which can be added in several ways to control how variability is modeled across different parts of the matrix. As an alternative to providing a single value to apply uniformly to all elements (as done above) you can provide a matrix specifying sample sizes for each element, or a list of matrices for distinct components (e.g., **U** and **F** matrices). This flexibility helps tailor the modeling of uncertainty to reflect different data availability across biological processes.

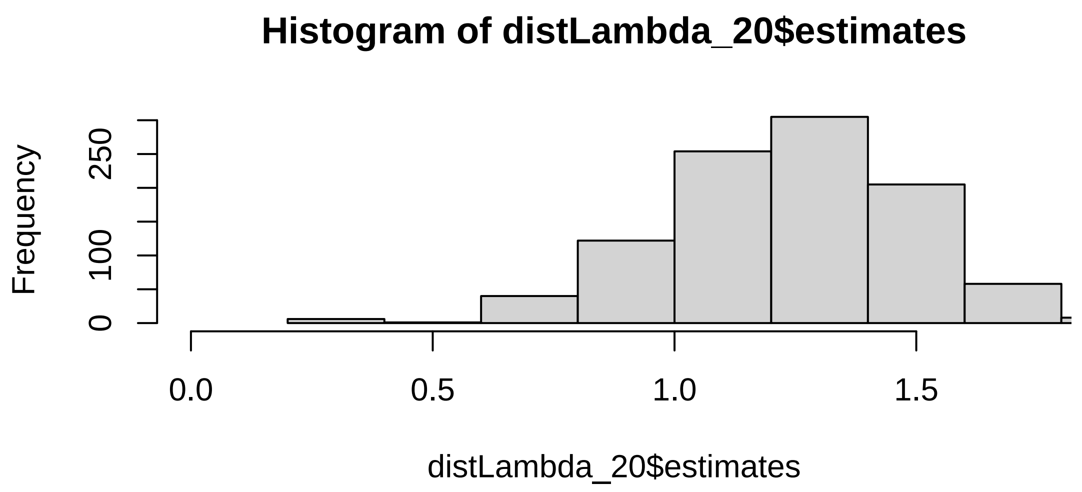
For instance, in the following code, we use the same MPM as above, split into **U** and **F** submatrices (`matU` and `matF`, respectively), but now assume that sample size varies between these components, with `40` individuals for **U** and `15` for **F**. Here, the sample size is consistent across all elements within the **U** matrix, but you could also assign different sample sizes to individual elements of the matrix, allowing for different sample sizes for different transitions.

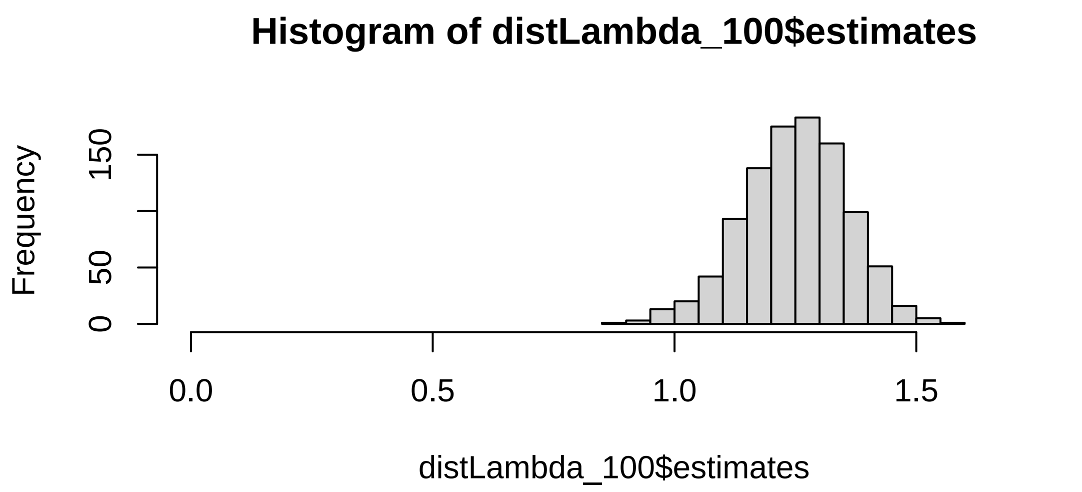
```
# Define the sample sizes for U
mat_U_ss <- matrix(c(
  40, 40,
  40, 40
), byrow = TRUE, nrow = 2)

# Define sample sizes for F
mat_F_ss <- matrix(c(
  0.0, 15,
  0.0, 0.0
), byrow = TRUE, nrow = 2)

# Combine sample sizes into list
sampleSizes <- list(mat_U_ss = mat_U_ss, mat_F_ss = mat_F_ss)

# Calculate CI for lambda
compute_ci(
  mat_U = matU, mat_F = matF, sample_size = sampleSizes,
  FUN = popdemo::eigs, what = "lambda"
)
#>     2.5%    97.5%
#> 0.895995 1.579860
```

Exploring the impact of sample size

Sample size is critical in determining the precision of statistical estimates. In demographic studies, small sample sizes can lead to high uncertainty in estimates of derived measures like λ, making it difficult to make strong inferences. Larger sample sizes reduce this uncertainty, as seen in the narrower confidence intervals around λ when we increase the sample size from 20 (calculated above) to 100 (below).

```
distLambda_100 <- compute_ci(
  mat_U = matU, mat_F = matF,
  sample_size = 100, FUN = popdemo::eigs, what = "lambda",
  dist.out = TRUE
)

par(mfrow = c(2, 1))
hist(distLambda_20$estimates, xlim = c(0, 1.75))
hist(distLambda_100$estimates, xlim = c(0, 1.75))
```





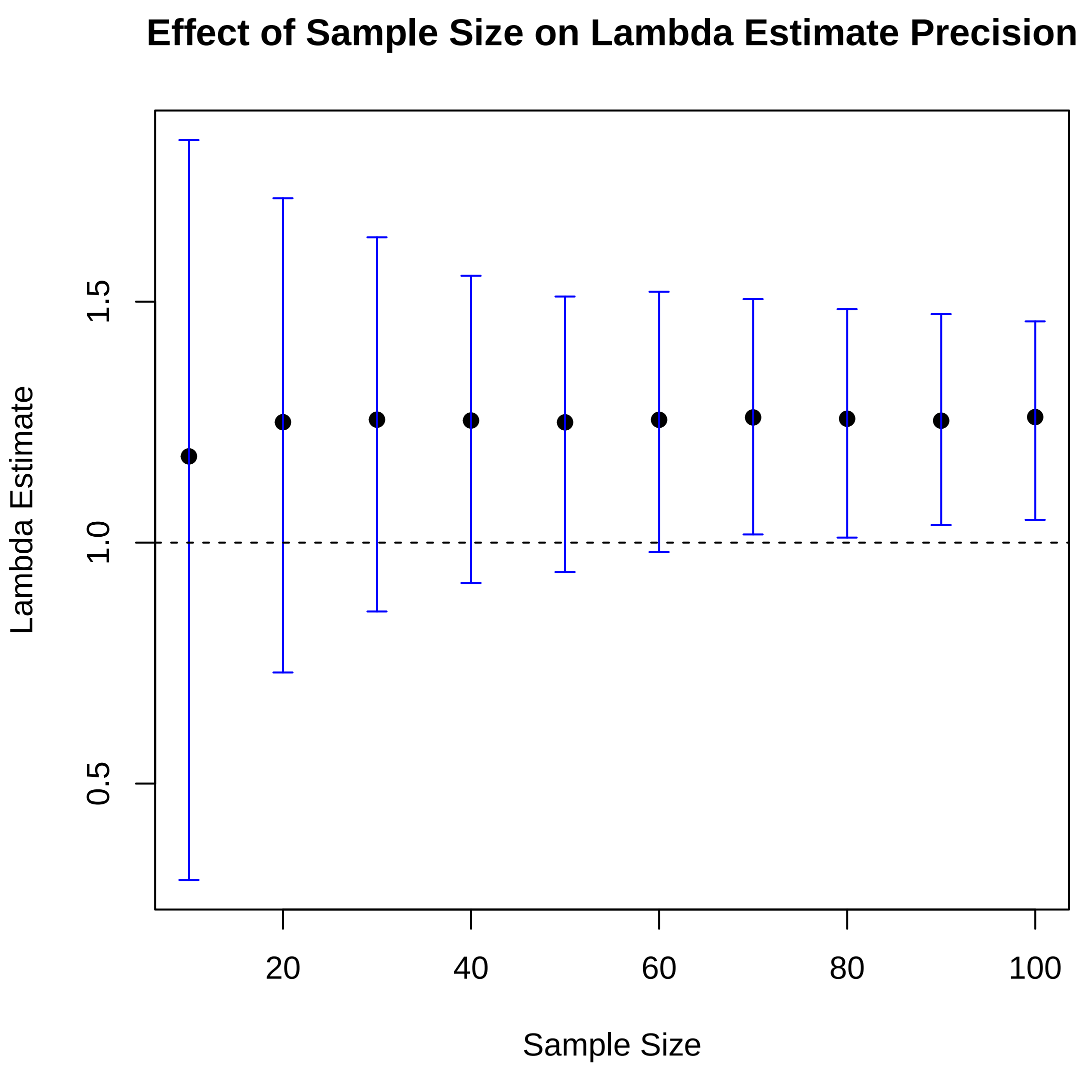
This approach can be used to perform a form of power analysis by simulation, a technique for determining the sample size required to detect an effect with a specified statistical power and significance level. For instance, one might ask, 'What sample size is needed to confidently conclude that the population growth rate is above 1.0?

The following code creates a plot to visualize how the precision of λ estimates improves as sample size increases. It first defines a set of sample sizes to iterate over, then it uses `compute_ci` to calculate the confidence intervals (CIs) for λ estimated from MPMs based on these sample sizes. It then plots λ estimates and their CIs, along with a reference line at $\lambda = 1$. The goal is to show that as the sample size increases, the width of the CIs shrinks, increasing our confidence in the value of λ.

In this case, a sample size of approximately 70 appears sufficient. However, sample size likely has greater importance for the more elastic elements of the MPM. Therefore, focusing on these elements could help users better understand the specific sample size requirements for their system. This targeted approach would lead to a more nuanced study design, allowing for optimized sampling efforts in the areas where precision matters most.

```r
# Define sample sizes to iterate over
sample_sizes <- seq(10,100,10)

# Lambda value for reference
```

```r
matA <- matF + matU
true_lambda <- popdemo::eigs(matA, what = "lambda")

# Initialize an empty data frame with predefined columns
ci_results <- data.frame(
  sample_size = sample_sizes,
  ci_lower = numeric(length(sample_sizes)),
  ci_upper = numeric(length(sample_sizes)),
  estimate_mean = numeric(length(sample_sizes))
)

# Loop through each sample size and calculate the CI for lambda
for (i in seq_along(sample_sizes)) {
  n <- sample_sizes[i]

  # Compute CI for the current sample size
  dist_lambda <- compute_ci(
    mat_U = matU, mat_F = matF,
    sample_size = n, FUN = popdemo::eigs, what = "lambda",
    dist.out = TRUE
  )

  # Calculate 95% CI from the distribution
  ci_results$ci_lower[i] <- quantile(dist_lambda$estimates, 0.025)
  ci_results$ci_upper[i] <- quantile(dist_lambda$estimates, 0.975)
  ci_results$estimate_mean[i] <- mean(dist_lambda$estimates)
}

# Calculate error bars
ci_lower_error <- ci_results$estimate_mean - ci_results$ci_lower
ci_upper_error <- ci_results$ci_upper - ci_results$estimate_mean

# Create the plot
plot(ci_results$sample_size, ci_results$estimate_mean,
     ylim = range(ci_results$ci_lower, ci_results$ci_upper),
     pch = 19, xlab = "Sample Size", ylab = "Lambda Estimate",
     main = "Effect of Sample Size on Lambda Estimate Precision")

# Add error bars and reference line
arrows(ci_results$sample_size, ci_results$ci_lower,
       ci_results$sample_size, ci_results$ci_upper,
       angle = 90, code = 3, length = 0.05, col = "blue")
abline(h = 1, lty = 2)
```



Conclusion

This vignette demonstrates how sampling error propagates through MPMs, influencing metrics like population growth rate, and provides a practical method for estimating confidence intervals for these metrics using `compute_ci`. By applying tools like `compute_ci`, users can also evaluate the effect of sample size on estimate precision, using this information to optimize data collection efforts and improve the reliability of demographic estimates.

Addendum

The `compute_ci` function is intended for metrics that rely on the full **A** matrix. However, some metrics, like `life_expect_mean` from the `Rage` package, only require the **U** matrix. For metrics such as these, users should use the `compute_ci_U` function, which works similarly to `compute_ci`, but is specifically designed for metrics that focus solely on survival and growth processes within the **U** matrix.

Supplemental Code S4: life history archetypes and life history structuring

```
library(mpmsim)
library(Rage)
library(Rcompadre)
library(dplyr)
library(popbio)
library(ggfortify)

set.seed(42)
constrain_df <- data.frame(fun = "lambda", arg = NA, lower = 0.9, upper =
1.1)
sim_life_hist_1 <- generate_mpm_set(
  n = 250, n_stages = 3, fecundity = c(0, 6, 6), archetype = 1, split =
TRUE,
  max_surv = 0.95, constraint = constrain_df,attempts = 3000
)

sim_life_hist_1 <- cdb_flag(sim_life_hist_1, checks = "check_irreducible")
%>%
  filter(check_irreducible == TRUE)

# Put the matrices into the metadata
sim_life_hist_1$matA <- matA(sim_life_hist_1)
sim_life_hist_1$matU <- matU(sim_life_hist_1)
sim_life_hist_1$matF <- matF(sim_life_hist_1)

# Use cdb_metadata to turn this into a data frame
sim_life_hist_1 <- cdb_metadata(sim_life_hist_1)


# New functions to calculate generation time from life table.
# Function to calculate generation time from the life table
gt_lt <- function(matU, matF, start = 1, ...) {
  tempLT <- mpm_to_table(matU, matF, start = start, ...)
  return(sum(tempLT$x * tempLT$lxmx) / sum(tempLT$lxmx))
}
```

```r
sim_life_hist_1$gt_lt <- mapply(
  gt_lt, sim_life_hist_1$matU,
  sim_life_hist_1$matF
)
sim_life_hist_1$longevity <- sapply(sim_life_hist_1$matU,
                                    Rage::longevity,
                                    x_max = 1000, lx_crit = 0.01
)
sim_life_hist_1$lifeExpect <- sapply(
  sim_life_hist_1$matU,
  Rage::life_expect_mean
)
sim_life_hist_1$entropy_d <- mapply(
  entropy_d,
  sim_life_hist_1$matU,
  sim_life_hist_1$matF
)

sim_life_hist_1$entropy_k <- mapply(entropy_k, sim_life_hist_1$matU)
sim_life_hist_1$nrr_R0 <- mapply(
  net_repro_rate, sim_life_hist_1$matU,
  sim_life_hist_1$matF
)

pcData <- sim_life_hist_1 %>%
  select(gt_lt, longevity, lifeExpect, entropy_d, entropy_k, nrr_R0) %>%
  na.omit()

PCA <- prcomp(pcData, scale = TRUE, center = TRUE)

# Add the PC data to the raw data.
pcData <- pcData %>%
  cbind(PCA$x[, 1:2])

PCA_plot <- autoplot(
  object = PCA, alpha = 0, size = 4, fill = "#55616D60",
  loadings.colour = "#0072B2", shape = 16,
  loadings = TRUE, loadings.label = TRUE, loadings.label.colour = "red",
  loadings.label.size = 3, loadings.label.repel = TRUE,
  frame = FALSE, frame.type = "norm", scale = 0
)

PCA_plot$layers <- c(
  geom_point(
    aes(
      x = pcData$PC1,
      y = pcData$PC2
    ),
    size = 2, alpha = 0.5
  ),
  PCA_plot$layers
)

A <- PCA_plot + ggtitle("A) Archetype 1") +
  theme_minimal()+
  theme(
    text = element_text(size = 8),
    axis.title = element_text(size = 8),
    axis.text = element_text(size = 7),
    plot.title = element_text(size = 9, face = "bold")
  )

ggsave("PCA_Archetype1.png", width = 5, height = 5)
####

set.seed(34)
constrain_df <- data.frame(fun = "lambda", arg = NA, lower = 0.9, upper =
1.1)
sim_life_hist_1 <- generate_mpm_set(
  n = 250, n_stages = 3, fecundity = c(0, 6, 6),archetype = 4, split =
TRUE,
  max_surv = 0.95, constraint = constrain_df,attempts = 3000
```

```
)


sim_life_hist_1 <- cdb_flag(sim_life_hist_1, checks = "check_irreducible")
%>%
  filter(check_irreducible == TRUE)


# Put the matrices into the metadata
sim_life_hist_1$matA <- matA(sim_life_hist_1)
sim_life_hist_1$matU <- matU(sim_life_hist_1)
sim_life_hist_1$matF <- matF(sim_life_hist_1)


# Use cdb_metadata to turn this into a data frame
sim_life_hist_1 <- cdb_metadata(sim_life_hist_1)



# New functions to calculate generation time from life table.
# Function to calculate generation time from the life table
gt_lt <- function(matU, matF, start = 1, ...) {
  tempLT <- mpm_to_table(matU, matF, start = start, ...)
  return(sum(tempLT$x * tempLT$lxmx) / sum(tempLT$lxmx))
}


sim_life_hist_1$gt_lt <- mapply(
  gt_lt, sim_life_hist_1$matU,
  sim_life_hist_1$matF
)
sim_life_hist_1$longevity <- sapply(sim_life_hist_1$matU,
                                    Rage::longevity,
                                    x_max = 1000, lx_crit = 0.01
)
sim_life_hist_1$lifeExpect <- sapply(
  sim_life_hist_1$matU,
  Rage::life_expect_mean
)
sim_life_hist_1$entropy_d <- mapply(
  entropy_d,
  sim_life_hist_1$matU,
  sim_life_hist_1$matF
)

sim_life_hist_1$entropy_k <- mapply(entropy_k, sim_life_hist_1$matU)
sim_life_hist_1$nrr_R0 <- mapply(
  net_repro_rate, sim_life_hist_1$matU,
  sim_life_hist_1$matF
)

pcData <- sim_life_hist_1 %>%
  select(gt_lt, longevity, lifeExpect, entropy_d, entropy_k, nrr_R0) %>%
  na.omit()

PCA <- prcomp(pcData, scale = TRUE, center = TRUE)

# Add the PC data to the raw data.
pcData <- pcData %>%
  cbind(PCA$x[, 1:2])

PCA_plot <- autoplot(
  object = PCA, alpha = 0, size = 4, fill = "#55616D60",
  loadings.colour = "#0072B2", shape = 16,
  loadings = TRUE, loadings.label = TRUE, loadings.label.colour = "red",
  loadings.label.size = 3, loadings.label.repel = TRUE,
  frame = FALSE, frame.type = "norm", scale = 0
)

PCA_plot$layers <- c(
  geom_point(
    aes(
      x = pcData$PC1,
      y = pcData$PC2
    ),
    size = 2, alpha = 0.5
  ),
  PCA_plot$layers
)


B <- PCA_plot + ggtitle("B) Archetype 4") +
  theme_minimal()+
  theme(
    text = element_text(size = 8),
    axis.title = element_text(size = 8),
    axis.text = element_text(size = 7),
    plot.title = element_text(size = 9, face = "bold")
  )



ggsave("PCA_Archetype4.png", width = 5, height = 5)


A/B


ggsave("/Users/jones/Dropbox/mpmsim manuscript/MEE Revision
1/PCAPlots.png",
       width = 3.35, height = 4.72, units = "in", dpi = 300)
```