`stagePop`: Modelling stage-structured populations in R

Helen Kettle and David Nutter
Email: Helen.Kettle@bioss.ac.uk

Biomathematics and Statistics Scotland (BioSS), Kings Buildings, Edinburgh, EH9 3FD, UK.

May 5, 2015

Keywords— population dynamics, food webs, predator-prey, host-parasitoid, life cycles, life stages, generations, age structure, food webs

Summary

1. We present an R-package, `stagePop`, which uses delay-differential equations to model the deterministic dynamics and interactions of stage-structured populations (i.e. where the life cycle consists of distinct stages - e.g. eggs, juveniles and reproductive adults).

2. The continuous time formulation enables `stagePop` to easily simulate time-varying stage durations and overlapping generations.

3. The package can be used to model predator-prey interactions, host-parasitoid interactions, resource competition, intra-specific competition and the effects of environmental change on stage-structured (and non-stage structured) species.

4. Our code is based on the formulation by Nisbet and Gurney (1983) using delay differential equations, which are solved using the R-packages `deSolve` or `PBSddesolve`.

1 Introduction

`stagePop` is an R package that can be used to model the deterministic dynamics and interactions of stage-structured populations. These are populations where the life cycle consists of distinct stages - e.g. eggs, juveniles and reproductive adults. Explicitly including stage structure when modelling the population dynamics of stage-structured organisms can have an enormous effect on the resulting dynamics. This may be because the organism is only predated upon when it is in certain life stages. Or that environmental variables, such as temperature, only influence the development rate of certain stages.

`stagePop` has been specifically designed to investigate these sorts of ecological problems and can therefore simulate the dynamics of stage-structured populations that are involved in predator-prey interactions, host-parasitoid interactions, resource competition, environmental change and so on. It also has the ability to simulate the dynamics of any number of strains within a species and therefore can be used to test questions about diversity and intra-specific competition. This means it is ideally suited to investigate the timely issues of biological engineering and control, biodiversity and climate change.

The package is based on the formulation by Bill Gurney and Roger Nisbet (Nisbet and Gurney (1983); Gurney et al. (1983); Gurney and Nisbet (1998)), described in detail in Appendix 1. Broadly speaking the model assumes that once an individual is born, unless it dies, it moves through its different life stages as if on a conveyor belt which may speed up and slow down as its development rate changes. Thus, an organism begins life by being born into the first stage and then, if it survives long enough, will mature into each successive stage. Within each stage it is assumed that each individual has the same vital rates e.g. the same death rates, the same rate of maturation etc. The formulation is based on delay differential equations (which are solved within `stagePop` using the R-packages `deSolve` (Soetaert et al., 2010) or `PBSddesolve` (Schnute et al., 2013) and this continuous time formulation copes easily with time-varying stage durations. Non-stage-structured species (which don't require delay equations) may also be modelled using `stagePop` which is useful when modelling the interactions of a number of species where not all have distinct life stages. In this paper we give a description of `stagePop` and provide a number of examples demonstrating how `stagePop` can be used for different modelling projects.

2 Running `stagePop`

To install the stagePop package, in R, type **install.packages('stagePop')** followed by **library(stagePop)**. To run the model the function **popModel()** is called. In this section we give a brief overview of this function (further details are given in Appendix 2). The output from **popModel()** is a matrix which contains the values of the state variables, the probabilities of surviving each stage, the durations of each stage (if time-varying) and the rates of change of each state variable at the times specified, via '`timeVec`', in the input to **popModel()**. Each column of the output matrix is named using the '`speciesNames`' and '`stageNames`' specified in the **popModel** inputs (see Section 1.2, Appendix 2). Different species may be specified in any order (as long as the definition is consistent) but the life stages must be referred to in the same order as they are in the life cycle. Furthermore, the birth of new organisms is assumed to be into the first stage only.

The input arguments to **popModel()** are used to completely define the model system and are described in detail in Appendix 2 (section 1.1). One of these inputs is a list containing all the rate functions (e.g. death rates, reproduction rates etc) for all entities in the model. Appendix 2 (section 1.1.1) gives a detailed description on how these functions must be defined.

Also included in the `stagePop` package are the following functions:

- **checkSolution()** produces warnings if the solution contains any negative values

- **genericPlot()** provides a basic plot of the results (most of the figures in this paper have been generated automatically by `stagePop`)
- **plotStrains()** if there are multiple strains in a species, will plot them individually (**genericPlot()** only plots the sum of the strains).
- **runStagePopExample** runs the examples shown in section 3, e.g. runStagePopExample('BlowFlies')
- **sumStrains** if there are multiple strains in a species, sums the model output over the strains in each species.

These functions are automatically called in the **popModel()** function, unless the user specifies otherwise, but can also be used on a stand alone basis. In Appendix 2 we give some tips on how to check the solution generated from `stagePop` is accurate (Appendix 2, section 3) and some ideas on trouble shooting typical problems that may occur with the delay differential equation solvers (Appendix 2, section 4).

3 Example Applications

In this section we demonstrate how `stagePop` can be used to simulate a wide range of problems involving stage-structured populations. Where possible, in order to verify our software is working correctly, we have reproduced published examples. The scripts for all of these examples are included in the `stagePop` package[1], are reproduced in Appendix 3 and are also attached as supplementary files. They are intended to serve as a template for users when defining their own problems. The name of the appropriate script is given in square brackets in each example heading and they can be run in R (after ('`library(stagePop)`') using **runStagePopExample('BlowFlies')** (for the BlowFlies.R example).

3.1 Single Species with fixed death rates and stage durations [`BlowFlies.R`]

A classic example of a stage-structured population is Nicholson's Blowflies (Gurney et al. (1983); Nicholson (1954, 1957)). Australian sheep blowflies, which have five distinct developmental stages, grown under controlled conditions in a laboratory experiment were found to exhibit sustained, large, quasi-cyclic fluctuations in their adult populations. To reproduce these experiments, the per capita death rates and duration of each stage are assumed to be constant (values are given in the caption for Fig. 1 and in Script 1, Appendix 3) and the reproductive rate (i.e. rate of egg production) is defined as $8.5 \exp\left(\frac{-A(t)}{600}\right)$ where the $A(t)$ is the reproductive adult stage (stage 5) and eggs are stage 1. The simulation is initiated with the immigration of 100 adults per day over the first day (this is fairly arbitrary however - the magnitude of the rate of immigration does not affect the equilibrium state results; similarly the immigration can be into any life stage). Script 1 (Appendix 3) shows how **popModel** can be used to simulate this situation. Fig 1 shows the plot automatically generated by `stagePop` (compare with Fig. 3 by Gurney et al. (1983)).

[1] These files are located in the `stagePop` installation directory and may be accessed by '`system.file("DemoFiles/ExampleFileName.R", package = "stagePop")`'

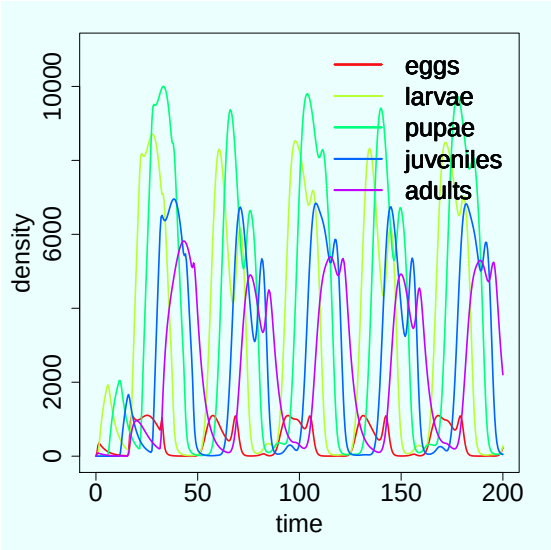

Figure 1: Simulation of the five life stages of Nicholson's blow flies (Section 3.1). The per capita death rates for stages 1-5 are 0.07, 0.004, 0.003, 0.0025, 0.27 d^{-1} and the durations of stages 1-4 are 0.6, 5.0, 5.9, 4.1 d. Compare with Fig. 3 by Gurney et al. (1983).

3.2 A single species with density-dependent death rates: Larval Competition [`LarvalComp.R`]

In this second example from Gurney et al. (1983), a two stage moth population (larvae and adults) is considered in which larval competition for resources results in a density dependent per capita larval death rate given by $\alpha L(t)$ where α=5x10^{-5} $moths^{-1}$ d^{-1} and $L(t)$ is the density of larvae at time t. The larval stage duration is 28 d and reproduction (by adults) is given by $qA(t)$ where $q = 9.4$ eggs/adult/d and $A(t)$ is the density of adults at time t. Two different cases are considered; in the first, the adult death rate is fixed at 0.2 d^{-1}, in the second, the adults are assumed to die after 5 days. In this second case the adult death rate is set to zero and a third stage (corpses) is added to the model. We begin both of the simulations with the immigration of adults at a rate of 20 d^{-1} over the first day. The results are shown in Fig. 2 and Figs. 2c and 2d can be compared with Fig. 4 by Gurney et al. (1983). This example demonstrates the huge difference in population dynamics caused by different ways of modelling the death of adults. The code required to run either of these cases in `stagePop` is shown in Script 2, Appendix 3.

3.3 A single species whose stage durations depend on temperature [`VarDurEnv.R`]

Unfortunately we could not find a sufficiently simple published example of a continuous-time model of a stage-structured population affected by temperature change (although there are more complex ones, e.g. Beck-Johnson et al. (2013)), so we have formulated our own example.

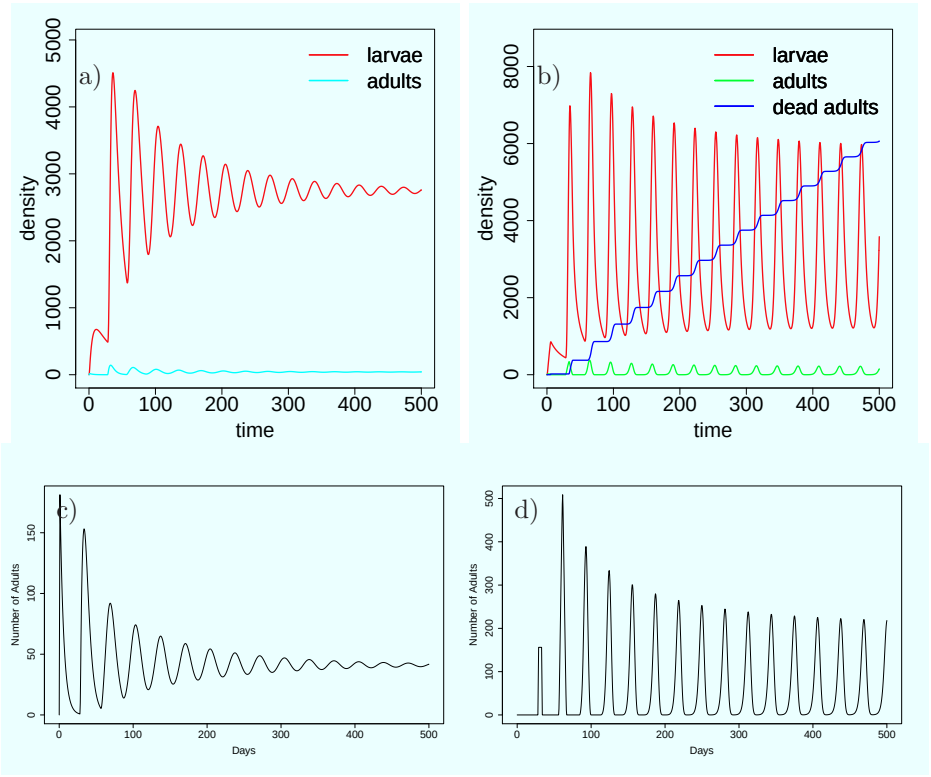

Figure 2: Modelling larval competition. a) the adult death rate is constant at 0.2 d^{-1}; b) the adults have a fixed lifetime of 5 d; c) and d) are the same as a) and b) but show only adults for comparison with Fig. 4 a and b, by Gurney et al. (1983).

We consider a theoretical species with two stages (juvenile and adult) where growth experiments conducted over a range of different but constant temperatures, T_c, have shown that the length of the juvenile stage, τ, is affected by temperature according to:

$$\tau(T_c) = \tau_{\min} + \left(\frac{T_c - T_{opt}}{w}\right)^2 \tag{1}$$

where $T_{opt} = 20^o$C, w = 2^oC d and $\tau_{\min} = 60$ d (see Fig. 3a and **tauFunc** in Script 3 in Appendix 3). We use this relationship to define the instantaneous juvenile development rate,

$$g(T(t)) = \frac{1}{\tau(T(t))} \tag{2}$$

which we assume will also apply to time-varying temperatures, $T(t)$. To inform `stagePop` that we are dealing with a problem involving a time-varying stage duration we set '`timeDependDuration`' equal to TRUE. Since in this example τ is now changing with time, we define **develFunc** using Eq. 2 and this is used to compute the rate of change of $\tau(T(t))$ (Eq. 9 in Appendix 1. The **durationFunc** (which is used to define non-time-varying durations) is now only used to define the length of the stage duration at the begining of the simulation (see Script 3, Appendix 3).

We then simulate the growth of the species over a number of years where the temperature, T, varies over an annual cycle according to,

$$T(t) = T_a(1 - cos(2\pi(t + 80)/365)) \tag{3}$$

where the yearly average temperature, T_a, is 15^o C, t is in days and the time offset of 80 d is required to prevent the species dying out due to low temperatues when the population is small at the start of the simulation. This is defined in the function **tempFunc** in Script 3 (Appendix 3) and displayed in Fig. 3c.

The simulation begins with the immigration of adults and reproduction is assumed to be density dependent. The definition of this model is described for `stagePop` using Script 3 (Appendix 3) and the results are shown in Fig. 3. The changes in the juvenile stage duration τ over time, computed from `stagePop` are shown in Fig. 3d and these are compared with the value of τ computed from Eq. 1 which is the stage duration if the current temperature, $T(t)$, had been constant over the stage duration.

3.4 Two interacting species: Predator-Prey System [`PredPrey1.R`] and [`PredPrey2.R`]

In this example we show how `stagePop` can be used to model two species – a predator and its prey. We begin with the classic Lotka-Volterra predator-prey model where neither species has stage structure (PredPrey1.R) and then increase the complexity by adding in stage structure for the predator and then density dependent death for the prey (PredPrey2.R) which is the system studied by Gourley and Kuang (2004).

The classic Lotka-Volterra equations (Lotka, 1925) are given by

$$\dot{x}(t) = rx(t) - py(t)x(t) \tag{4}$$

$$\dot{y}(t) = bpy(t)x(t) - Dy(t) \tag{5}$$

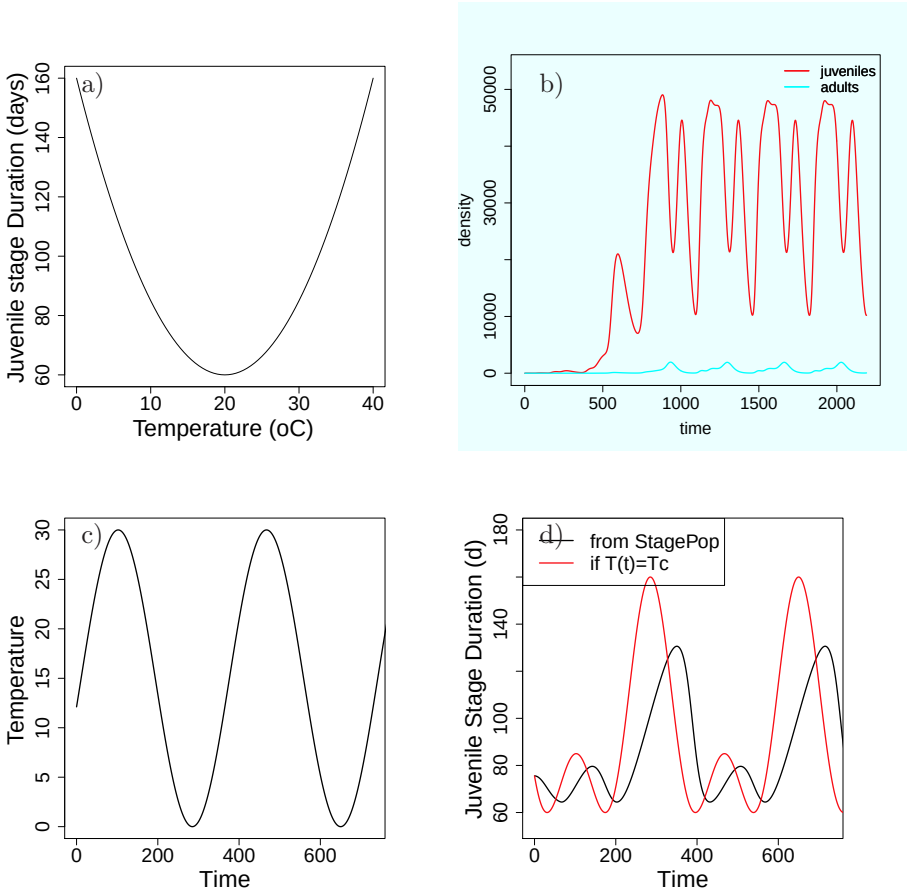

Figure 3: Simulation of a single species with a juvenile and adult stage where the juvenile development rate is temperature dependent (Section 3.3). a) Relationship between juvenile stage duration and temperature (Section 3.3 (Eq. 1)); b) Results from `stagePop`; c) temperature time series; d) comparison of calculations of τ. Note the lower two plots are shown for a shorter time period (2 years rather than 6 years) for clarity.

where $x(t)$ and $y(t)$ are the prey and predator densities at time t respectively, $rx(t)$ is the prey reproduction rate, $py(t)$ is the per capita death rate of prey due to predation, $bpy(t)x(t)$ is the predator reproduction rate and D is the per capita death rate of the predator.

This system is defined in `stagePop` as shown in Script 4 (Appendix 3). The results of the simulation are shown in Fig. 4. The analytical solution to Equations 4 and 5 is the closed loop shown in Fig. 4b. However, if the tolerances on the DDE solver are not strict enough, the solution will be subject to numerical errors and the predator-prey loop in Fig. 4b will not be closed. For example changing the value of '`tol`' in '`solverOptions`' from '`1e-7`' to '`1e-3`' gives the result shown in Figs. 4c and 4d (see Appendix 2, Section 3, for further tips on how to check your solution is accurate).

We now look at the case where the predator has juvenile and adult stages (y_j and y respectively) and only the adult stage consumes the prey. The equations now become

$$\dot{x}(t) = rx(t) - py(t)x(t) \quad (6)$$

$$\dot{y_j}(t) = bpy(t)x(t) - bpy(t-\tau_j)x(t-\tau_j)\exp(-D_j\tau_j) - D_j y_j(t) \quad (7)$$

$$\dot{y}(t) = bpy(t-\tau_j)x(t-\tau_j)\exp(-D_j\tau_j) - Dy(t). \quad (8)$$

where D_j is the juvenile predator per capita death rate (here set at 1 d^{-1}) and τ_j is the length of the juvenile predator stage duration. The `stagePop` code for this new situation is in Script 5 (Appendix 3) (where 'case=1') and the results of the simulation for τ_j=0.1 are shown in Fig. 5. It is clear that adding in a juvenile stage which does not predate causes large changes compared to the equilibrium situation shown in Fig. 4 (a and b).

We now add in a density dependent death rate for the prey such that the prey equation becomes

$$\dot{x}(t) = rx(t)(1 - \frac{x(t)}{K}) - py(t)x(t) \quad (9)$$

which is the system investigated by Gourley and Kuang (2004) (Gourley and Kuang, 2004). To run this in `stagePop` we only need state a value for K and modify **deathFunc** as shown for cases>1 in Script 5 (Appendix 3) With K=1 and the other parameters as before, Fig. 6 shows the results when the predator juvenile stage duration is 0.1 d and 1.8 d (cases 2 and 3 respectively in Script 5 (Appendix 3). Note the length of the simulation has been increased so that these plots can be more easily compared with results in Gourley and Kuang (2004). The results from `stagePop` compare well with these until $\tau_j \geq 15$ after which there are large, unaccounted for discrepancies between the simulations (cases 6 and 7 in PredPrey2.R (Script 5, Appendix 3)).

3.5 Multiple interacting species: Host-Parasitoid System [`Briggs.R`]

This example considers three interacting species, all with stage structure, in the host-parasitoid system investigated by Briggs (1993). The host species has 3 life stages (eggs, E; larvae, L and adult, A) and is attacked by two competing parasitoids: P, which attacks the host eggs and, Q, which attacks the host

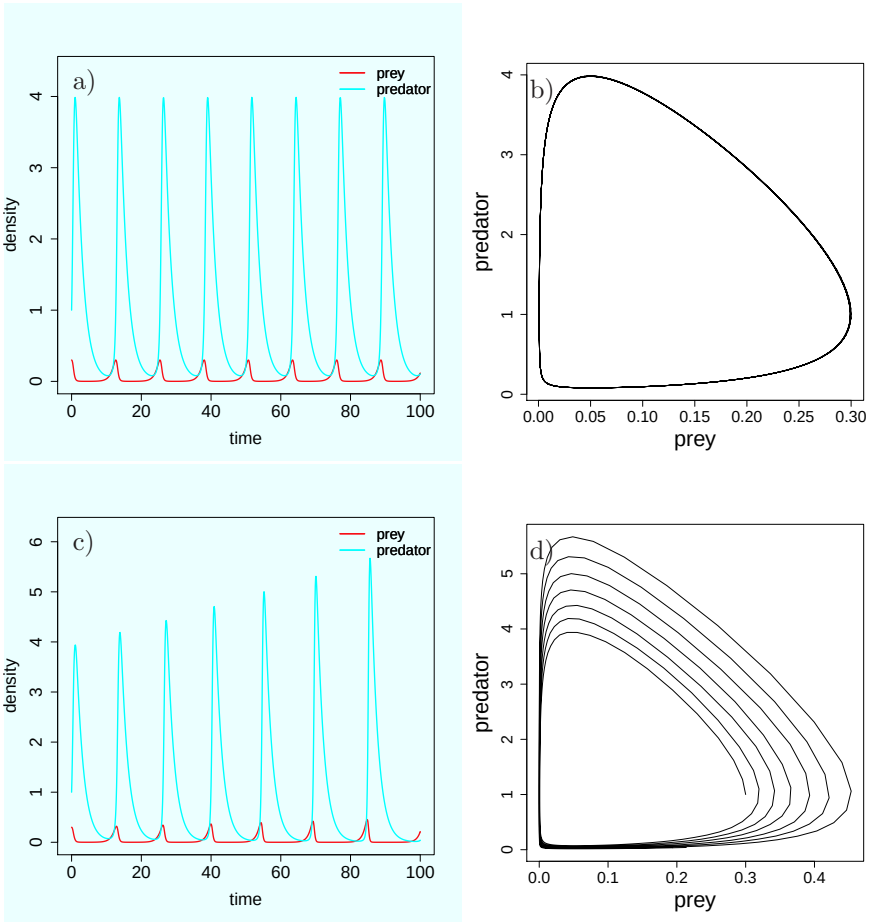

Figure 4: a) Predator-prey dynamics with no stage structure. The closed loop in b) indicates the solution is numerically accurate ('tol'=1e-7 in '`solverOptions`'); d) when 'tol'=1e-3 in '`solverOptions`', numerical accuracy has not been achieved (and plot c) is incorrect) as the predator-prey graph is no longer a closed loop.

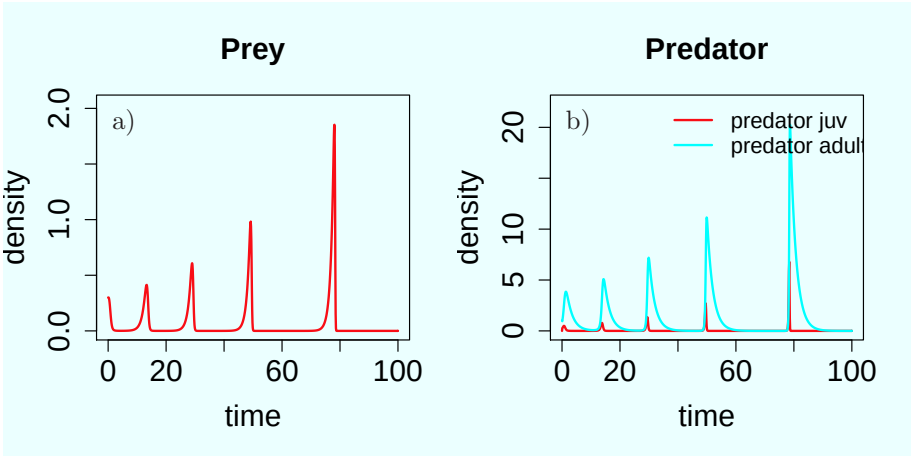

Figure 5: Predator-prey dynamics where the predator has a juvenile stage of duration 0.1 d.

larvae. Both parasitoids have 2 life stages – juvenile (P_J and Q_J) and adult (P_A and Q_A). The egg and larval attack rates are denoted by a_P and a_Q resepctively. Each parasitised host becomes a single juvenile parasitoid, thus the death rates due to parasitoids are $a_P P_A(t)E(t)$ and $a_Q Q_A(t)L(t)$ for the host eggs and larvae respectively, and reproduction into the parasitoid juvenile class is $a_P E(t)P_A(t)$ for P and $a_Q L(t)Q_A(t)$ for Q. Reproduction of the host is given by $\rho D_A A(t)$ where ρ is host lifetime fecundity and D_A is the adult death rate (see Briggs (1993) for the equations describing this system). In this example we use the steady solution (provided in Appendix B by Briggs (1993)) to set the values for the immigration rates (these parameters are identified by the 'star' in their name - e.g. Qstar). The `stagePop` code for this system is shown in Script 6 (Appendix 3).

Theory dictates (Briggs, 1993) that in this situation the two parasitoids can not co-exist. We use `stagePop` to simulate an invasion of P (after 20 time units) into a situation where only Q and the host are initially present. For the case in which the parasitoid Q has twice the attack rate of P (a_P=1, a_Q=2) and all their other parameters are identical, P still manages to displace Q since it attacks at an earlier stage (eggs rather than larvae) and the system settles at a new equilibrium with a higher adult host density (Fig. 7). These simulations are interesting from the point of view of biological control - if the adult host is a pest which causes damage e.g. to people or crops, the most desirable parasitoid to release is that which minimises the adult host density when in equilibrium.

3.6 Consumer-Resource problem with variable stage duration [`VarDurFood.R`]

Here we use `stagePop` to reproduce an example given by Nisbet and Gurney (1983) in which the length of the larval stage of a 2-stage species (loosely based on the damselfly) is determined by the availability of their food. Specifically, an individual larva becomes an adult once it has assimilated enough food to raise its body mass by m mass units. Thus, by definition the larval stage duration,

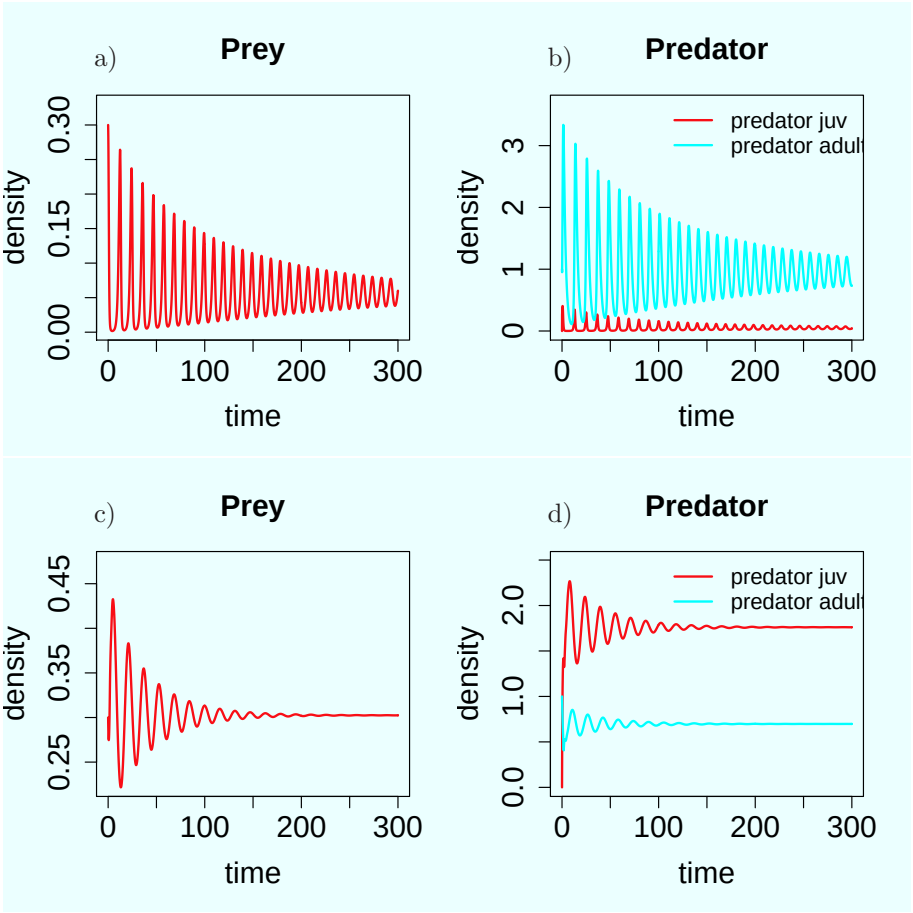

Figure 6: Predator-prey dynamic where the prey has density dependent death and the predator has a juvenile stage of duration 0.1 d (top row) and 1.8 d (bottom row). Compare with Fig. 3 by Gourley and Kuang (2004).

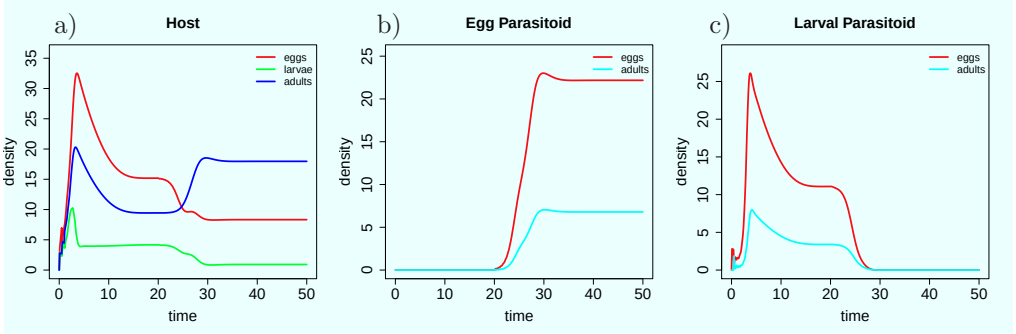

Figure 7: Competition between two parasitoids attacking different life stages of one host (compare with Fig.3 (Briggs, 1993)). See Script 6 (Appendix 3) for parameter values.

$\tau_L(t)$ is determined by the equation

$$\int_{t-\tau_L(t)}^{t} g_L(x)dx = m, \tag{10}$$

where $g_L(t)$ is the larval development rate (see Appendix 1, Eq. 7). It is assumed that $g_L(t)$, is proportional to the rate of food consumption per larva, $f_L(t)$, such that

$$g_L(t) = \epsilon f_L(t), \tag{11}$$

and

$$f_L(t) = f_{\max}\frac{F(t)}{K + F(t)}, \tag{12}$$

where $F(t)$ is the food density, K is the half saturation constant and $f_{\max}$ is the maximum food consumption rate. Thus the rate of food uptake by larvae is $f_L L(t)$. Food is supplied to the larvae at a constant rate f_s and the adults have a fixed rate of reproduction, $qA(t)$ where $A(t)$ is the adult density at time t. Both larvae and adults are assumed to have fixed per capita death rates D_L and D_A respectively.

To solve this problem in `stagePop` we define it as a two species problem where food is one species and the damselfly is the other. The 'reproduction' of food is the constant rate of food supply, f_s, and its 'death' is modelled by the per capita rate uptake rate, $f_L L(t)/F(t)$, i.e.

$$\frac{f_{\max}}{K + F(t)}L(t). \tag{13}$$

A time dependent death rate is specified for the food species and a time dependent duration for the 'damselfly' using the **popModel()** arguments '`timeDependLoss`' and '`timeDependDuration`' respectively. Since the length of the stage duration is changing in time, the **durationFunc** is only required at t=0, and the development rate is set in **develFunc** using Eqs. 11 and 12. To set the initial value of τ_L it is assumed that for $t < 0$ the development rate is constant, thus Eq. 10 implies $g_L(0)\tau_L(0) = m$. At the beginning of the simulation the larvae have an initial amount of food, $F(0)$ and thus $g_L(0)$ can be computed from Eqs. 11 and 12 to give

$$\tau_L(0) = m\frac{(K + F(0))}{\epsilon f_{\max} F(0)}. \tag{14}$$

The instructions for `stagePop` are given in Script 7 (Appendix 3) and the results are shown in Fig. 8.

This example shows the flexibility of using the Nisbet-Gurney formulation – the stage duration can be controlled by any model variable, allowing size, age, weight etc to determine the time of transition into the next life stage.

3.7 Consumer-Resource model with multiple strains in one species [`MultipleStrains.R`]

In this example we demonstrate how a species with multiple strains can be modelled in `stagePop`. We begin with looking at a simplified model of bacteria in the human colon. The bacteria feed on a resource R (e.g. food that has not

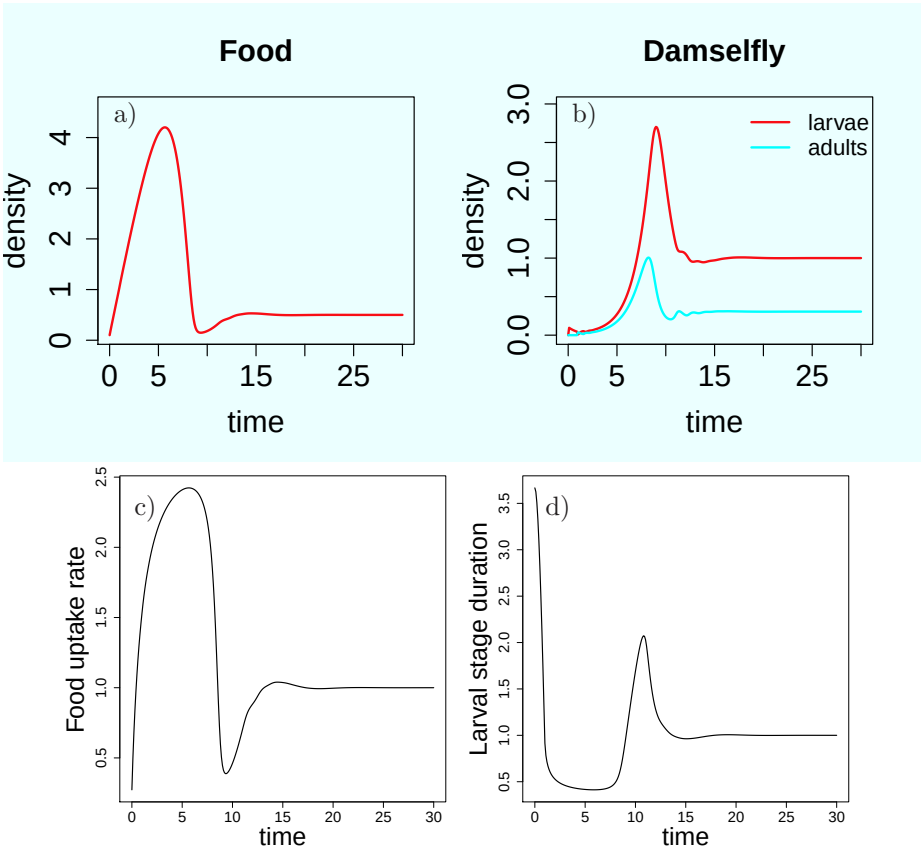

Figure 8: Example where length of larval maturation time is determined by food availability (Section 3.6). The simulation begins with $L(0) = A(0) = 0$, $F(0) = 0.1$ with immigration into the larval stage at rate 1 per unit time for the first 0.1 time units. The parameters used to achieve these plots (which compare well with Fig. 3 (Nisbet and Gurney, 1983)) are $f_s = m = \epsilon = K = 1$, D_A=2, q=5, $D_L = \ln(\frac{q}{D_A})$ and $f_{\max} = 3$.

been digested further up the gut) and are subject to transport through the gut at a rate of V. Assuming Monod-equation type growth, the rate of change in concentration (or density) of bacterial strain i, is given by

$$\frac{dB^i(t)}{dt} = G^i \frac{R(t)}{R(t)+K} B^i(t) - VB^i(t) \tag{15}$$

(e.g. Kettle et al. (2014)) where G^i is the maximum specific growth rate of strain i and K is the half saturation constant (assumed constant over all strains). The rate of change of resource is given by

$$\frac{dR(t)}{dt} = VR_{in} - \frac{R(t)}{R(t)+K} \sum_{i=1}^{N} \frac{G^i B^i(t)}{Y} - VR(t) \tag{16}$$

where N is the number of strains, Y is the yield (i.e. the number of grams of B produced from one gram of R) and R_{in} is the concentration of the incoming resource. This is modelled in `stagePop` as two species – species one is the resource and species 2 is the bacteria (Script 8, Appendix 3). With no stage structure the system rapidly reaches steady state with one strain dominating the system (competitive exclusion; Fig. 9a and b).

However, bacteria may have a lag phase during which time there is little or no cell growth but the cells are busy replicating various proteins and DNA in preparation for the reproductive phase. For demonstration purposes, we assume the length of this phase, τ_i, varies slightly between strains, and whilst in this stage the bacteria are not subject to the usual transport through the system. Thus for the lagged stage, B_1,

$$\frac{dB_1^i(t)}{dt} = G^i \frac{R(t)}{R(t)+K} B_2^i(t) - m^i(t) \tag{17}$$

where $m^i(t)$ is the maturation rate of strain i from stage one at time t; and for the reproductive stage, B_2,

$$\frac{dB_2^i(t)}{dt} = m^i(t) - VB_2^i(t). \tag{18}$$

The rate of change of resource is now given by

$$\frac{dR(t)}{dt} = VR_{in} - \frac{R(t)}{R(t)+K} \sum_{i=1}^{N} \frac{G^i B_2^i(t)}{Y} - VR(t). \tag{19}$$

When assigning the strain traits (G^m and τ_i), we assume a trade-off such that a longer lag time leads to faster growth. We incorporate this second case (Script 8, Appendix 3) and see that this has a significant effect on the results (Fig. 9c-e). The system now does not reach steady state even over the extended time period shown and multiple strains are still co-existing after 100 time units.

4 Conclusion

The preceding sections demonstrate only a small range of the possible problems `stagePop` can be used to investigate. However, we hope that these demonstrate its flexibility and potential, and that other researchers will find `stagePop` useful in their own fields.

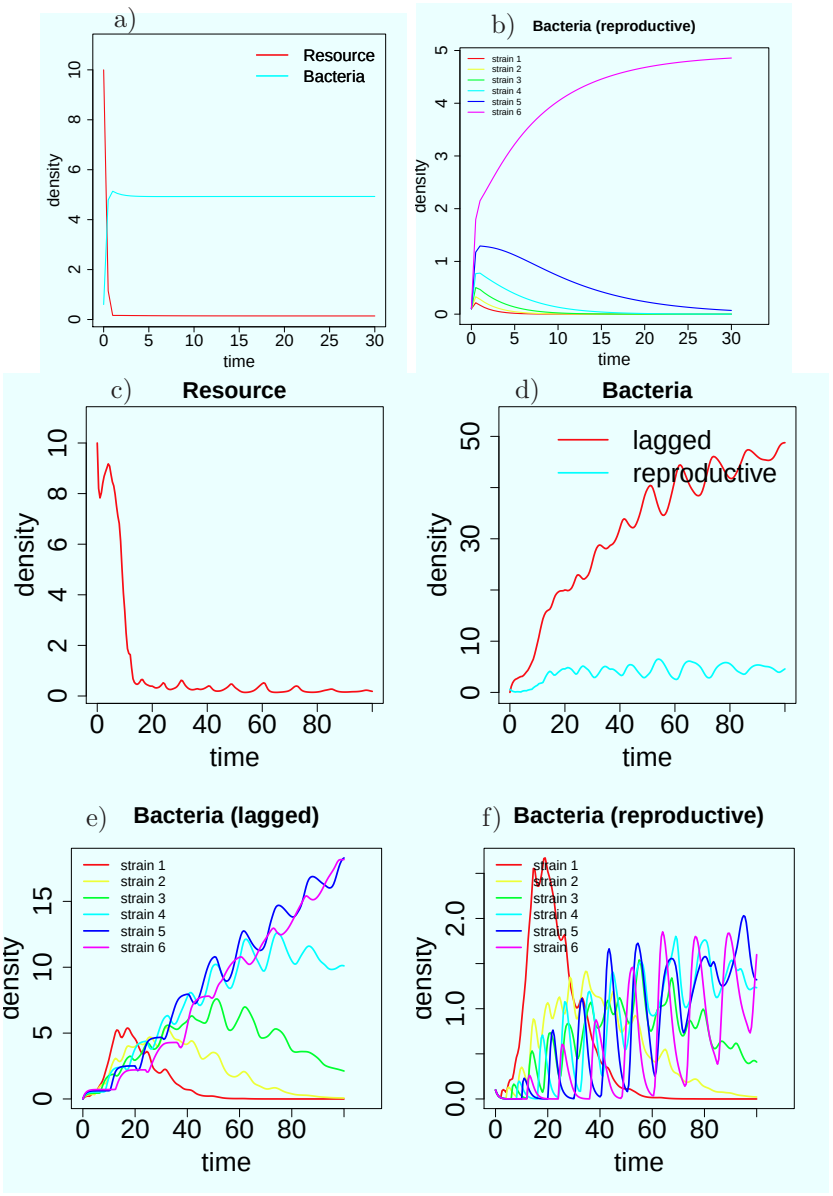

Figure 9: Example using multiple strains in one species. Model with no stage structure: a) where 'Bacteria' is the sum over all strains, and b) time evolution of the 6 invidual strains. Model with stage-structure: Time evolution of resource (c), the two stages of bacteria (d), and the individual strains for the lagged stage (e) and the reproductive stage (f). Note extended time period for the stage-structured model.

Acknowledgements

This work was financially supported by the German Federal Ministry for Education and Research (research grant no. FKZ01LL0917A-01LL0917O, for the LEGATO project) and the Scottish Goverment's Rural and Environment Science and Analytical Services Division (RESAS).

A The Model Formulation

Here we briefly describe the mathematical formulation used in the `stagePop` however for full details the reader should consult (Gurney et al. (1983); Nisbet and Gurney (1983); Gurney and Nisbet (1998)).

For any stage-structured species, if the number of individuals in stage i is given by N_i, then based on simple book-keeping:

$$\dot{N}_i(t) = R_i(t) - M_i(t) - L_i(t)N_i(t) \tag{20}$$

where R_i is the rate of recruitment into stage i (includes immigration, and reproduction or maturation from the previous stage), M_i is the rate of maturation out of stage i and $L_i(t)$ is the per capita loss rate of individuals in stage i, given by

$$L_i(t) = D_i(t) + E_i(t). \tag{21}$$

where D_i is the per capita death rate (due to natural causes and predation/-parasitoid attack etc) and E_i is the per capita emigration rate for individuals in stage i.

The rate of maturation, M_i, from stage i at time t is given by

$$M_i(t) = R_i(t - \tau_i(t))P_i(t)[1 - \dot{\tau}_i(t)] \tag{22}$$

where $P_i(t)$ is the fraction of individuals entering stage i at $t - \tau_i(t)$ that has survived to time t. Assuming individuals are born into stage 1, the reproductive stage is r, β is the reproductive function and I is the immigration rate, then recruitment into the first stage is given by

$$R_1(t) = \beta(N_r(t)) + I_1(t). \tag{23}$$

and, into subsequent stages by

$$R_{i+1}(t) = M_i(t) + I_{i+1}(t), \text{ for } i = 1, ..., r. \tag{24}$$

Note that immigrants are assumed to be 'new born' into whichever stage they enter (i.e. with respect to age or size etc they are at the beginning of their entry stage).

The fraction of individuals which enter stage i at $t - \tau_i$ and survive to time t is given by

$$P_i(t) = \exp\left(\int_{t-\tau_i(t)}^{t} -L(x)dx\right). \tag{25}$$

However in order to avoid solving integro-differential equations, P_i is replaced by its time derivative,

$$\dot{P}_i(t) = P_i(t)[L_i(t - \tau_i(t))[1 - \dot{\tau}_i(t)] - L_i(t)], \tag{26}$$

plus an initial condition, $P_i(0) = \exp(-L_i(0)\tau_i(0))$. The rate of change of the stage duration, $\dot{\tau}_i$, is defined in terms of $g_i(t)$ which is the rate at which an individual develops within stage i at any given time, t. If $g(t)$ is considered to be the rate of change of a development index (e.g. size, age, nutrient levels

etc) that goes from 0 at the start of the stage to γ at the end, then $\tau_i(t)$ is determined by the requirement

$$\int_{t-\tau_i(t)}^{t} g_i(x)dx = \gamma. \tag{27}$$

Differentiating this gives,

$$g(t) - g(t-\tau(t))\frac{d}{dt}(t-\tau(t)) = 0, \tag{28}$$

and thus the rate of change of τ is given by

$$\dot{\tau_i}(t) = 1 - \frac{g_i(t)}{g_i(t-\tau_i(t))}. \tag{29}$$

For a more detailed explanation, consult Section 8.5 in Gurney and Nisbet (1998).

If the functions (or parameters) D_i, g_i (or τ_i if τ_i is time independent), E_i, I_i and β are defined, and the initial histories (i.e for $-\tau_i \leq t \leq 0$) for N_i, τ_i and P_i are provided, Equations 20-29 completely define the population dynamics of any stage-structured species.

A.1 Assumptions

In `stagePop` we make the following assumptions for the initial histories:

$$\begin{aligned}
R_i(t) &= 0 \text{ for all } t \leq 0, && (30)\\
I_i(t) &= 0 \text{ for all } t \leq 0, && (31)\\
E_i(t) &= 0 \text{ for all } t \leq 0, && (32)\\
N_i(t) &= N_i(0) \text{ for all } t \leq 0, && (33)\\
D_i(t, N_i(t)) &= D_i(0, N_i(0)) \text{ for all } t \leq 0 \text{ if } D = D(t), && (34)\\
g_i(t, N_i(t)) &= g_i(0, N_i(0)) \text{ for all } t \leq 0 \text{ if } \tau = \tau(t). && (35)\\
& && (36)
\end{aligned}$$

These are somewhat restrictive and not a result of the mathematical formulation but rather an attempt to produce robust and simple code for

A.2 Multiple Species

For predator-prey, host-parasitoid and consumer-resource systems there can be multiple interacting species (bearing in mind here we can consider food as a 'species'). In this case each species is essentially modelled separately but the species may interact in the functions for death rates, reproductive rates and developmental rates.

A.3 Simplifications

If the stage duration, τ_i, does not change with time (i.e. when '`timeDependDuration`' is '`FALSE`' in the call to **popModel()**, as in the blowflies, larval competition,

predator-prey and host-parasitoid examples), then Equations 22, 26 and 29 reduce to

$$M_i(t) = R_i(t-\tau_i)P_i(t) \quad (37)$$
$$\dot{P}_i(t) = P_i(t)((D_i(t-\tau_i)+E_i(t-\tau_i))-(D_i(t)+E_i(t))) \quad (38)$$
$$\dot{\tau}_i(t) = 0. \quad (39)$$

Furthermore, if D_i and E_i also do not change with time (i.e. when '`timeDependLoss`' is '`FALSE`' in the call to **popModel()** as in the blowflies example), then Equations 25 and 26 reduce to

$$P_i(t) = \exp(-((D_i+E_i)\tau_i) \quad (40)$$
$$\dot{P}_i(t) = 0. \quad (41)$$

B Description of `stagePop` functions

B.1 Functions contained in `stagePop`

Also included in the `stagePop` package are the following functions:

- **checkSolution()** produces warnings if the solution contains any negative values
- **genericPlot()** provides a basic plot of the results (most of the figures in this paper have been generated automatically by `stagePop`)
- **plotStrains()** if there are multiple strains in a species, will plot them individually (**genericPlot()** only plots the sum of the strains).
- **runStagePopExample** runs the examples shown in section 3, e.g. runStagePopExample('BlowFlies')
- **sumStrains** if there are multiple strains in a species, sums the model output over the strains in each species.

These functions are automatically called in the **popModel()** function, unless the user specifies otherwise, but can also be used on a stand alone basis.

B.2 Inputs and Outputs for popModel()

B.2.1 The input arguments to popModel()

'`numSpecies`' (integer) is the number of different species to be modelled.

'`numStages`' is a vector containing the number of life stages in each species, e.g. '`numStages=c(2,1)`' if there are two species - one with two stages and another with one.

'`numStrains`' is a vector containing the number of strains in each species, e.g. '`numStrains=c(5,10)`' if there are two species - one with five strains and another with ten.

'`timeVec`' is a vector describing the output times for the solution
e.g. '`seq(0,10,0.5)`'.

'`speciesNames`' This is a vector of the strings containing the species' names
e.g. '`speciesNames=c('host','egg parasitoid','larval parasitoid')`'
These can also be used in the user-defined rate functions to index the variables in the input argument 'x' (see Section B.2.2) and will be used to name the columns of the output matrix from **popModel()**.

'`stageNames`' This is a list containing vectors of the stage names (strings) for each species. E.g. for 2 species with one stage and three stages respectively this could have the form
'`stageNames=list('adults',c('eggs','juvs','adults'))`'.

'`rateFunctions`' is a list containing the user-defined rate functions – see Section B.2.2 for details.

'`ICs`' is a list of matrices containing the initial conditions for every stage and strain of each species. These must be zero for all stages apart from the reproductive stage (usually the last stage). Thus it is recommended that more complicated initial conditions are defined through immigration rates in the **immigrationFunc** (see section B.2.2). However, the ICs can be set as follows: Each species has a matrix with the number of columns equal to the number of strains in that species and the number of rows equal to the number of stages in that species. E.g. for 2 species, the first with 2 strains and 3 stages, the second with 1 strain and 1 stage, then for zero starting conditions: '`ICs=list(matrix(0,ncol=2,nrow=3),matrix(0,ncol=1,nrow=1))`'.

'`timeDependLoss`' (optional; default is rep(TRUE,numSpecies)) is a vector containing TRUE/FALSE for each species. It is TRUE if the per capita loss rate varies with time for any stage of the species (e.g. TRUE for density dependent death and/or density dependent emigration) and FALSE otherwise. E.g. '`timeDependLoss=c(TRUE, FALSE)`' if there is no emigration and the first species has time dependent per capita death rates but the second does not.

'`timeDependDuration`' (optional; default is rep(FALSE,numSpecies)) is a vector containing TRUE/FALSE for each species. It is TRUE if the stage duration varies with time for any stage of the species and FALSE otherwise. E.g. '`timeDependDuration=c(TRUE, FALSE)`' if the first species has any time dependent stage durations but the second does not.

'`solverOptions`' (optional; default is '`list(DDEsolver='PBS', tol=1e-7, hbsize=1e3, method='lsoda', atol=1e-7, dt=0.1))`' is a list of instructions for the DDE solver, containing: 'DDEsolver', 'tol', 'hbsize', 'method' and 'atol'. 'DDEsolver' must be either 'deSolve' or 'PBS' (these are the R packages used to solve the DDEs). The 'tol' option sets the relative tolerances and 'hbsize' sets the size of the history buffer. The remaining two items, 'method' and 'atol' set the numerical integration scheme and the absolute tolerance if DDEsolver='deSolve' (PBS does not have these options).

'`checkForNegs`' (optional; default is TRUE) is TRUE if you would like to check your solution for negative values using the function **checkSolution()**. Note **checkSolution()** can also be called separately using '`checkSolution(output,numSpecies,numStages,numStrains,ntol)`'.

'`ntol`' (optional; default is 0.01) is the tolerance on the magnitude of the negative values detected by **checkSolution()** . For example if ntol=0.01 then a warning is triggered if a stage or strain of any species falls below less than minus 1% of its maximum value at any time. If ntol is zero then any values below zero will trigger a warning even if their magnitude is insignificant (eg 10^{-30}).

'`plotFigs`' (optional; defalt is TRUE) is TRUE if you would like a basic plot of the solution. The function that is called to do this is called **genericPlot()** and can also be called separately using '`genericPlot(modelOutput, numSpecies, numStages, varNames, speciesNames, stageNames, saveFig, figType, figName)`'

To create more sophisticated plots the user is recommended to use the results in the matrix generated from **popModel()** with their own plotting scripts.

'`saveFig`'(optional; default is FALSE.) To save the figure generated by **popModel()** make this TRUE.

'`figType`'(optional; default is 'eps'.) Format for the saved figure from **popModel()**. This can be 'eps', 'pdf', 'png' or 'tiff'.

'`figName`'(optional; default name is 'stagePopFig' and it will be saved in your working directory). A string containing the filepath for where the figure file should be saved.

'`sumOverStrains`' (optional; default is TRUE). If there is more than one strain in any species then the results for each strain are given in the model output. If you are only interested in the results for the species as a whole then change this to FALSE to simplify the model output.

'`plotStrainsFig`' (optional; default is TRUE (if max(numStrains)>1). This will produce plots for each individual strain.

'`saveStrainsFig`' (optional; default is FALSE). Change to TRUE to save the plot.

'`strainsFigType`' (optional; default is 'eps'). Format for the saved figure. This can be 'eps', 'pdf', 'png' or 'tiff'.

'`strainsFigName`' (optional; default is 'strainFig' plus the species name). A string containing the filepath describing where the strain plots should be saved.

B.2.2 The Rate Functions (user-defined)

The user must define a list containing all the functions named below. These must have the input arguments specified below in order for `stagePop` to run (however, these arguments do not necessarily need to be used within the function). The output from each of them must be a single value which equals the rate for the stage, species and time given in the input arguments. The input arguments to these functions are all single values apart from '`x`' which is a vector of all the state variables at the input time. This is included to allow the user to specify density-dependent rates and can be indexed using the names specified in the **popModel()** arguments '`speciesNames`' and '`stageNames`' using x$speciesName['stage',strain]. For more details see Appendix 3 and/or use the R help function for each of the functions below.

reproFunc(x,time,species,strain) The output from this is a value for the rate at which new organisms enter the first stage through reproduction of the species in the input argument at the given time. The units are: organisms time unit^{-1}. Note, unlike the other rate functions, this does not have the 'stage' input argument (this is because new organisms produced by reproduction are only allowed to enter the first stage).

deathFunc(stage,x,time,species,strain) The output from this is the per capita death rate of organisms in the stage and species given in the input arguments at the given time. The units are: time unit^{-1}.

durationFunc(stage,x,time,species,strain) The output from this is the length of the stage duration (in time units) for the stage and species given in the input arguments. Note that if the stage durations vary in time then this function will only be used to compute the initial values of the stage durations; for future values the user must define **develFunc**.

develFunc(stage,x,time,species,strain) The output from this is the rate of development rate of the stage and species given in the input arguments at the given time. The units are: time unit^{-1}. Note that if the stage duration (and hence development rate) is constant in time then there is no need to define this function e.g. if timeDependDuration is FALSE for all species. The output from **develFunc** must always be strictly greater than zero.

immigrationFunc(stage,x,time,species,strain) The output from this is the rate of immigration (individuals per unit time) into the stage specified in the input arguments for the input species at the given time. Since the initial conditions for `stagePop` are quite restrictive, the user will generally start the simulation by specifying immigration into a given stage over a short interval at the start of the simulation. Note that `stagePop` assumes that individuals immigrating into a stage are at the beginning of that stage (and therefore will not move to the next stage until the time for the stage duration has elapsed). Also the user should be aware that if these rates are set very high the DDE solvers may fail so the user should endeavour to enter realistic (or at least low) values.

emigrationFunc(stage,x,time,species,strain) The output from this is the per capita emigration rate of individuals in the stage and species given in the input arguments. The units are: time unit^{-1}.

B.2.3 Ouput from popModel()

The output from **popModel()** is a matrix which contains the values of the state variables, the probabilities of survival, the durations of each stage (if time-varying) and the rates of change of each state variable at the times specified in the input time vector ('`timeVec`'). Each column of the output matrix is named according to the **popModel()** input option '`variableNames`':

'`time`' is the time point t at which the solution is output (specified by input argument '`timeVec`').

'`speciesName[i].StageNames[[i]][j]`' is the density (or number) of stage j of species i at time t. For multiple strains, for strain k in species i, stage j, this is '`speciesName[i].StageNames[[i]][j].strain[k]`'

'`prob.speciesName[i].StageNames[[i]][j]`' is the probability that stage j of species i will survive the length of the stage duration at time t. Note this is only present if species i has time dependent per capita death rates or has time dependent stage durations.

'`dur.speciesName[i].StageNames[[i]][j]`' is the duration of stage j for species i at time t. Note this is only present if the stage durations for species i are time dependent.

'`dot.speciesName[i].StageNames[[i]][j]`' is the rate of change of the density (or number) of stage j of species i at time t.

B.3 Assumptions made in `stagePop`

In `stagePop` the following assumptions are hard-coded

- The birth (via reproduction) of new individuals is always into the first life stage.
- The simulation time always begins at zero.
- For all simulation times less than zero:
  - there is no reproduction or immigration,
  - the number or density of organisms in each stage is equal to those at time zero,
  - the death rate is equal to that at time zero,
  - the development function is equal to that at time zero.

B.4 Checking the solution from `stagePop` is accurate

As with any numerical integration the results from `stagePop` are subject to error. Ideally, the user should check the results against an analytical solution (e.g. at equilibrium conditions). However, since this is frequently not possible, the simplest way to check for inaccuracies is to use the **checkSolution()** function (this is called if the **popModel()** input argument '`checkForNegs`' is TRUE (default)). This will find negative values in your solution within a limit specified by '`ntol`'. If negative values do occur then the user can try reducing the size of the tolerances in 'solverOptions' to improve the numerical accuracy. For further confirmation of the solution this can be repeated with both DDE solvers (i.e. deSolve and PBS). The user should be aware that the size of the tolerances needed may differ vastly between modelling projects and DDE solvers. As the tolerance size decreases the run time for `stagePop` will increase, thus if CPU time is important the user is recommended to find the largest tolerance size required for an accurate solution. To do this we recommend repeating the simulation with increasingly smaller tolerances until the solution no longer changes.

B.5 Trouble Shooting

If error messages or warnings appear in the console window that mention the integration step size this generally means that the DDE solver can not solve the problem without making the step size too small to compute (generally because the problem is stiff). The default solver method for deSolve is '`lsoda`' which is designed to deal with stiff and non-stiff problems but the user can also try different methods via the '`method`' option. If time lags are long then errors may

occur saying the lag history is not long enough (or that the lag for a variable is too long). To deal with this the history buffer size of the solver can be changed in using '`hbsize`'. If both the DDE solvers (deSolve and PBS) fail to complete the integration it is likely that the user has incorrectly specified the problem or has generated extremely high rates in the user-defined rate functions. In this case it is recommended that the rate functions be carefully checked for internal consistency and/or the problem be non-dimensionalised or simply more appropriately scaled. If an error says 'The number of derivatives returned by func() must equal the length of the initial conditions vector' then the vector of initial conditions is incorrect (e.g. perhaps an entry is missing).

C. List of Example Scripts

Script 1. BlowFlies.R (Section 3.1): Single Species with fixed death rates and stage durations.

```
library(stagePop)
#All the vectors are specified in the order of the life cycle
#e.g. start with eggs and finish with reproducing adults

solver.options=list(DDEsolver='deSolve', atol=1e-3, rtol=1e-3,
    hbsize=1e4)
#solver.options=list(DDEsolver='PBS', tol=1e-7, hbsize=1e4, dt=0.01)

blowFliesFunctions <- list(
                        reproFunc=function(x,time,species,strain){
                          A0=600
                          q=8.5
                          reprod=q*x$blowflies['adults',1] *
                               exp(-x$blowflies['adults',1]/A0)
                          return(reprod)
                        },
                        deathFunc=function(stage,x,time,species,strain){
                        #per capita death rate (/d)
                          a=c(0.07,0.004,0.003,0.0025,0.27)
                          return(a[stage])
                        },
                        durationFunc=function(stage,x,time,species,strain){
                        #duration of each stage in days
                          a=c(0.6,5.0,5.9,4.1)
                          return(a[stage])
                        },
                        immigrationFunc=function(stage,x,time,species,strain){
                          v=0
                          if (stage==5){
                            if (time>=0 & time<=1){v=100}
                          }
                          return(v)
                        },
                        emigrationFunc=function(stage,x,time,species,strain){return(0)}
                        )

modelOutput = popModel(
  numSpecies=1,
  numStages=5,
  ICs=list(matrix(0,nrow=5,ncol=1)),
  timeVec=seq(0,200,0.5),
  timeDependLoss=TRUE,
  timeDependDuration=FALSE,
  rateFunctions=blowFliesFunctions,
  solverOptions=solver.options,
  stageNames=list(c('eggs','larvae','pupae','juveniles','adults')),
  speciesNames=c('blowflies'),
  saveFig=TRUE,
  figType='eps',
  figName='blowflies'
  )
```

Script 2. LarvalComp.R (Section 3.2): A single species with density-dependent death rates.

```
library(stagePop)

solver.options=list(DDEsolver='deSolve',atol=1e-4,rtol=1e-4,method='lsoda',hbsize=1e6)
#solver.options=list(DDEsolver='PBS',tol=1e-6,hbsize=1e3,dt=0.1)

case=1 #choose case (1 or 2)
if (case==1) {
  num.stages=2
  stage.names=c('larvae','adults')
} else{
  num.stages=3
  stage.names=c('larvae','adults','dead adults')
}

larvalCompFunctions <- list(
                        reproFunc=function(x,time,species,strain){
                          reprod=9.4*x$flies['adults',1]
                          return(reprod)
                        },
                        deathFunc=function(stage,x,time,species,strain){
                          if (stage==1){v=5e-5*x$flies['larvae',1]}
                          if (stage>=2){if (case==1){v=0.2}else{v=0}}
                          return(v)
                        },
                        durationFunc=function(stage,x,time,species,strain){
                          a=c(28,5)
                          return(a[stage])
                        },
                        immigrationFunc=function(stage,x,time,species,strain){
                          v=0
                          if (time>=0 & time<=1){
                            if (stage==2){v=20}}
                          return(v)
                        },
                        emigrationFunc=function(stage,x,time,species,strain){return(0)}
                        )

modelOutput=popModel(
  numSpecies=1,
  numStages=num.stages,
  ICs=list(matrix(0,nrow=num.stages,ncol=1)),
  timeVec=seq(0,500,0.5),
  timeDependLoss=TRUE,
  solverOptions=solver.options,
  rateFunctions=larvalCompFunctions,
  stageNames=list(stage.names),
  speciesNames='flies',
  saveFig=TRUE,
  figType='eps',
  figName=paste('LarvalComp',case,sep=''))
```

Script 3. VarDurEnv.R (Section 3.3): A single species whose stage durations depend on temperature

```
solver.options=list(DDEsolver='deSolve',atol=1e-6,rtol=1e-6,hbsize=1e5)
#solver.options=list(DDEsolver='PBS',tol=1e-8,hbsize=1e4,dt=0.01)

tempFunc=function(time){
  T=15*(1-cos(2*pi*(time+80)/365))
  return(T)}

tauFunc=function(T){
  maxDur=200; minDur=60
  v=min(minDur+((T-20)/2)^2,maxDur)
  return(v)}

varDurEnvFunctions<-list(
                       reproFunc=function(x,time,species,strain){
                         A0=600; q=11.5
                         reprod=q*x$Nematodes['adults',1] *
                              exp(-x$Nematodes['adults',1]/A0)
                         return(max(0,reprod))
                       },
                       deathFunc=function(stage,x,time,species,strain){
                         a=c(0.05,0.05)
                         v=a[stage]
                         return(max(0,v))
                       },
                       develFunc=function(stage,x,time,species,strain){
                         T=tempFunc(time)
                         v=1/tauFunc(T)
                         return(v)
                       },
                       durationFunc=function(stage,x,time,species,strain){
                         if (time==0){
                           T=tempFunc(time)
                           v=tauFunc(T)}
                         return(v)
                       },
                       immigrationFunc=function(stage,x,time,species,strain){
                         v=0
                         if (stage==2){if (time>=0 & time <=0.1){v=1}}
                         return(v)
                       },
                       emigrationFunc=function(stage,x,time,species,strain){return(0)}
)

modelOutput=popModel(
  numSpecies=1,numStages=2,
  timeDependLoss=FALSE,timeDependDuration=TRUE,
  ICs=list(matrix(0,nrow=2,ncol=1)),
  timeVec=seq(0,365*6,1),
  solverOptions=solver.options,
  rateFunctions=varDurEnvFunctions,
  stageNames=list(c('juveniles','adults')),speciesNames=c('Nematodes'))
```

Script 4. PredPrey1.R (Section 3.4): Classic Predator-Prey model.

```
growthRatePred=10
growthRatePrey=1
deathRatePrey=1
deathRatePred=0.5

#solver.options=list(DDEsolver='deSolve',atol=1e-6,rtol=1e-6,method='lsoda',hbsize=1e4)
solver.options=list(DDEsolver='PBS',tol=1e-7,hbsize=1e4,dt=0.01)

ppFunctions <- list(
                  reproFunc=function(x,time,species,strain){
                    if(species==1){
                      reprod=growthRatePrey*x$prey['adults',1]}
                    if(species==2){
                      reprod=growthRatePred*x$prey['adults',1]*x$predator['adults',1]}
                    return(max(0,reprod))
                  },
                  deathFunc=function(stage,x,time,species,strain){
                    if(species==1){
                      v=deathRatePrey*x$predator['adults',1]}
                    if(species==2){
                      v=deathRatePred}
                    return(max(0,v))
                  },
                  durationFunc=function(stage,x,time,species,strain){return(0)},
                  immigrationFunc=function(stage,x,time,species,strain){return(0)},
                  emigrationFunc=function(stage,x,time,species,strain){return(0)}
                  )

modelOutput=popModel(
  numSpecies=2,
  numStages=c(1,1),
  timeDependLoss=c(TRUE,FALSE),
  timeDependDuration=c(FALSE,FALSE),
  ICs=list(matrix(0.3,1,1),matrix(1,1,1)),
  timeVec=seq(0,100,0.1),
  solverOptions=solver.options,
  plotFigs=TRUE,
  rateFunctions=ppFunctions,
  speciesNames=c('prey','predator'),
  stageNames=list('adults','adults')
)
```

Script 5. PredPrey2.R (Section 3.4): Predator-Prey model with stage-structured predator

```
growthRatePred=10; growthRatePrey=1
deathRatePrey=1; deathRatePred=c(1.0,0.5)
preyCarryCapacity=1

#solver.options=list(DDEsolver='deSolve',atol=1e-6,rtol=1e-6,method='lsoda',hbsize=1e4)
solver.options=list(DDEsolver='PBS',tol=1e-8,hbsize=1e4,dt=0.01)

case=1 #choose case
if (case==1){juvPredDuration=0.1;lenTime=100}
if (case==2){juvPredDuration=0.1;lenTime=300}
if (case==3){juvPredDuration=1.8;lenTime=300}
if (case==4){juvPredDuration=0.1;lenTime=400;deathRatePred[1]=0}
if (case==5){juvPredDuration=5;lenTime=400;deathRatePred[1]=0}
if (case==6){juvPredDuration=15;lenTime=400;deathRatePred[1]=0}
if (case==7){juvPredDuration=20;lenTime=400;deathRatePred[1]=0}

ppFunctions <- list(
                  reproFunc=function(x,time,species,strain){
                    if(species==1){reprod=growthRatePrey*x$Prey['adult',1]}
                    if(species==2){reprod=growthRatePred*x$Prey['adult',1]*x$Predator['adult',1]}
                    return(max(0,reprod))
                  },
                  deathFunc=function(stage,x,time,species,strain){
                    if(species==1){
                      if(case==1){v=deathRatePrey*x$Predator['adult',1]}
                      if(case>1){v=deathRatePrey*x$Predator['adult',1]+
                        growthRatePrey*x$Prey['adult',1]/preyCarryCapacity}}
                    if(species==2){v=deathRatePred[stage]}
                    return(max(0,v))
                  },
                  durationFunc=function(stage,x,time,species,strain){
                    return(juvPredDuration)
                  },
                  immigrationFunc=function(stage,x,time,species,strain){return(0)},
                  emigrationFunc=function(stage,x,time,species,strain){return(0)}
                  )
modelOutput=popModel(
  numSpecies=2,
  numStages=c(1,2),
  timeDependLoss=c(TRUE,FALSE),
  timeDependDuration=c(FALSE,FALSE),
  ICs=list(matrix(0.3),matrix(c(0,1),nrow=2,ncol=1)),
  timeVec=seq(0,lenTime,0.1),
  solverOptions=solver.options,
  plotFigs=TRUE,
  rateFunctions=ppFunctions,
  stageNames=list('adult',c('juvenile','adult')),
  speciesNames=c('Prey','Predator')
  )
```

Script 6. Briggs.R (Section 3.5): Host-Parasitoid model (1 host, 2 parasitoids)

```
attackRateP=1; attackRateQ=2;
TE=0.5; TL=0.5; TJP=0.4; TJQ=0.4
deathE=0.1;deathL=0.1;deathA=0.1;deathJP=0.1;
deathP=8.0;deathJQ=0.1;deathQ=8.0
rho=33 #total lifetime fecundity
LstarQ=4.16;AstarQ=9.44;Qstar=3.40

BriggsFunctions <- list(
             reproFunc=function(x,time,species,strain){
               if (species==1){reprod=rho*deathA*x$Host['adults',1]}
               if (species==2){reprod=attackRateP*x$'Egg
                    Parasitoid'['adults',1]*x$Host['eggs',1]}
               if (species==3){reprod=attackRateQ*x$'Larval
                    Parasitoid'['adults',1]*x$Host['larvae',1]}
               return(max(0,reprod))
             },
             deathFunc=function(stage,x,time,species,strain){
               if (species==1){a=c(deathE,deathL,deathA);v=a[stage]
                   if (stage==1){v=a[stage]+attackRateP*max(x$'Egg
                        Parasitoid'['adults',1],0)}
                   if (stage==2){v=a[stage]+attackRateQ*max(x$'Larval
                        Parasitoid'['adults',1],0)}}
               if (species==2){a=c(deathJP,deathP);v=a[stage]}
               if (species==3){a=c(deathJQ,deathQ);v=a[stage]}
               return(max(0,v))
               },
              durationFunc=function(stage,x,time,species,strain){
                if (species==1){a=c(TE,TL)}
                if (species==2){a=TJP}
                if (species==3){a=TJQ}
                return(a[stage])
              },
              immigrationFunc=function(stage,x,time,species,strain){
                v=0
                if (species==1){if (time>=0 &
                     time<=0.1){f1=rho*deathA*AstarQ
                  if (stage==1){v=f1}
                  if (stage==2){v=f1*exp(-deathE*TE)}
                  if (stage==3){v=f1*exp(-deathE*TE-deathL*TL)}}}
                if(species==2){if(time>=20 & time<=20.1){
                    if(stage==2){v=1}}}
                if(species==3){if(time>=0 & time<=0.1){
                    if(stage==1){v=attackRateQ*Qstar*LstarQ}}}
                return(v)
              },
              emigrationFunc=function(stage,x,time,species,strain){return(0)}
              )

modelOutput=popModel(numSpecies=3,
  numStages=c(3,2,2),
  timeVec=seq(0,50,0.1),
```

```
  rateFunctions=BriggsFunctions,
  timeDependLoss=c(TRUE,FALSE,FALSE),
  timeDependDuration=c(FALSE,FALSE,FALSE),
  ICs=list(matrix(0,nrow=3,ncol=1),matrix(0,nrow=2,ncol=1),matrix(0,nrow=2,ncol=1)),
  solverOptions=list(DDEsolver='PBS',tol=1e-7,hbsize=1e4,dt=0.01),
  speciesNames=c('Host','Egg Parasitoid','Larval Parasitoid'),
  stageNames=list(c('eggs','larvae','adults'),c('eggs','adults'),c('eggs','adults'))
)
```

Script 7. VarDurFood.R (Section 3.6): Consumer-resource problem with variable stage duration.

```
fs=1; fmax=3; F0=0.1 #rate of food supply; max rate; initial food density
m=1 #number of mass units a larva must increase to become an adult
epsilon=1#const of proportionality between development and food
    consumption
K=1 #half sat constant for food consumption
q=5; dA=2 #reproduction rate; adult death rate
dL=log(q/dA) #larval death rate

solver.options=list(DDEsolver='deSolve',atol=1e-9,rtol=1e-9,method='lsoda',hbsize=1e4)
#solver.options=list(DDEsolver='PBS',tol=1e-9,hbsize=1e4,dt=0.01)

varDurFoodFunctions <- list(
                        reproFunc=function(x,time,species,strain){
                          if(species==1){reprod=fs}
                          if(species==2){reprod=q*x$Damselfly['adults',1]}
                          return(max(0,reprod))
                        },
                        deathFunc=function(stage,x,time,species,strain){
                          if(species==1){v=fmax*x$Damselfly['larvae',1]/(K+x$Food[1,1])}
                          if(species==2){a=c(dL,dA);v=a[stage]}
                          return(max(0,v))
                        },
                        durationFunc=function(stage,x,time,species,strain){
                          if(time==0 & species==2 & stage==1){
                            v=m/(epsilon*fmax*F0/(K+F0))}
                          return(v)
                        },
                        develFunc=function(stage,x,time,species,strain){
                          if (species==2 & stage==1){
                            v=epsilon*fmax*x$Food[1,1]/(K+x$Food[1,1])}
                          return(v)
                        },
                        immigrationFunc=function(stage,x,time,species,strain){
                          v=0
                          if (species==2 & stage==1){
                            if (time>=0 & time<=0.1){v=1}}
                          return(v)
                        },
                        emigrationFunc=function(stage,x,time,species,strain){return(0)}
                        )

modelOutput=popModel(
  numSpecies=2,speciesNames=c('Food','Damselfly'),
  numStages=c(1,2),stageNames=list('one',c('larvae','adults')),
  numStrains=c(1,1),
  timeDependLoss=c(TRUE,FALSE),timeDependDuration=c(FALSE,TRUE),
  ICs=list(matrix(F0,1,1),matrix(0,nrow=2,ncol=1)),
  timeVec=seq(0,30,0.1),
  solverOptions=solver.options, rateFunctions=varDurFoodFunctions
)
```

Script 8. MultipleStrains.R (Section 3.7): Consumer-resource with multiple strains which have a trade-off between maximum growth rate and stage duration.

```
Rin=10; V=1; K=1;Yield=0.5; num.strains=6
if (num.strains>1){Gmax=2+seq(1,num.strains)}else{Gmax=2}

case=2#choose case
if(case==1){num.stages=1;stage.names='reproductive';start=0.1}
if(case==2){num.stages=2;stage.names=c('lagged','reproductive');start=c(0,0.1)}

strainsFunctions <- list(
            reproFunc=function(x,time,species,strain){
              if (species==1){reprod=Rin*V}
              if (species==2){reprod=x$Bacteria['reproductive',strain]*
                  Gmax[strain]*x$Resource['food',1]/(x$Resource['food',1]+K)}
              return(reprod)
            },
            deathFunc=function(stage,x,time,species,strain){
              if (species==1){uptake=0*seq(1,num.strains)
                for (s in seq(1,num.strains)){
                  uptake[s]=(Gmax[s]/(x$Resource['food',1]+K))*(x$Bacteria['reproductive',s]/Yield)}
                death=sum(uptake)+V}
              if (species==2){
                if (stage==1){if(num.stages==2){death=0}else{death=V}}
                if (stage==2){death=V}}
              return(death)
            },
            durationFunc=function(stage,x,time,species,strain){
              durations=2*seq(1,num.strains)
              return(durations[strain])
            },
            immigrationFunc=function(stage,x,time,species,strain){return(0)},
            emigrationFunc=function(stage,x,time,species,strain){return(0)}
            )

modelOutput = popModel(
  numSpecies=2,
  numStrains=c(1,num.strains),
  numStages=c(1,num.stages),
  ICs=list(matrix(Rin,nrow=1,ncol=1),matrix(start,nrow=num.stages,ncol=num.strains)),
  timeVec=seq(0,100,0.5),
  timeDependLoss=c(TRUE,FALSE),
  timeDependDuration=c(FALSE,FALSE),
  rateFunctions=strainsFunctions,
  solverOptions=list(DDEsolver='PBS',tol=1e-7,hbsize=1e4,dt=0.01),
  stageNames=list(c('food'),stage.names),
  speciesNames=c('Resource','Bacteria'),
  saveFig=TRUE,figType='eps',figName=paste('multiStrain',case,sep=''),
  sumOverStrains=FALSE,
  plotStrainsFig=TRUE,saveStrainsFig=TRUE,strainsFigType='eps',strainsFigName='strainFig'
  )
```

References

Beck-Johnson, L., Nelson, W., Paaijmans, K., Read, A., Thomas, M., and Bjørnstad, O. (2013). The effect of temperature on anopheles mosquito population dynamics and the potential for malaria transmission. *PLOS one*, 8:79276.

Briggs, C. (1993). Competition among parasitoid species on a stage-structured host and its effect on host suppression. *The American Naturalist*, 141:372–397.

Gourley, S. and Kuang, Y. (2004). A stage structured predator-prey model and its depdendence on maturation delay and death rate. *Mathematical Biology*, 49:188–200.

Gurney, W. and Nisbet, R. (1998). *Ecological Dynamics*. Oxford University Press.

Gurney, W., Nisbet, R., and Lawton, J. (1983). The systematic formulation of tractable single-speices population models incorporating age structure. *Journal of Animal Ecology*, 52:479–495.

Kettle, H., Donnelly, R., Flint, H., and G., M. (2014). ph feedback and phenotypic diversity within bacterial functional groups of the human gut. *Journal of Theoretical Biology*, 342:62–69.

Lotka, A. (1925). *Elements of Physical Biology*. Williams and Wilkins.

Nicholson, A. (1954). An outline of the dynamics of animal populations. *Australian Journal of Zoology*, 2:9–65.

Nicholson, A. (1957). The self-adjustment of populations to change. *Cold spring Harbour symposium on Quantitative Biology*, 22:153–173.

Nisbet, R. and Gurney, W. (1983). The systematic formulation of population models for insects with dynamically varying instar duration. *Theoretical Population Biology*, 23:114–135.

Schnute, J., Couture-Beil, A., Haigh, R., and Kronlund, A. (2013). PBSmodelling R package. http://code.google.com/p/pbs-modelling/.

Soetaert, K., Petzoldt, T., and Woodrow Setzer, R. (2010). Solving differential equations in r: Package desolve. *Journal of Statistical Software*, 33(9):1–25. http://www.jstatsoft.org/v33/i09/.