Deterministic population dynamics

These exercises require the latest version of `popdemo`. Head to `popdemo`'s GitHub (github.com/iainmstott/popdemo) for installation instructions (don't forget also to load the package using `library(popdemo)`!)

Complete the core exercises (in normal print) first, as code in each section continues from the last. Afterward, return to the "extras" sections (in *italics*), and try writing your own code. Code to be run is in chunks:

```
# a comment
an_input()
##   an output
```

Key terms in the text are in ***bold italic***, and functions or arguments are `fixed width`.

---

1. Data

We will use a matrix projection model (MPM) to explore population dynamics for the desert tortoise *Gopherus agassizzii*, with medium fecundity[1]. The population is found in the Mojave desert, USA. There are 8 stages are based on age and size (carapace length in mm):
- Yearling (age 0-1)
- Juvenile 1 (<60 mm)
- Juvenile 2 (90-99mm)
- Immature 1 (100-139mm)
- Immature 2 (140-179mm)
- Subadult (180-207mm)
- Adult 1 (208-239mm)
- Adult 2 (>240mm)

Load in the data:

```
data(Tort); Tort
##         Yr    J1    J2    I1    I2    SA    A1   A2
##   Yr 0.000 0.000 0.000 0.000 0.000 1.300 1.980 2.57
##   J1 0.716 0.567 0.000 0.000 0.000 0.000 0.000 0.00
##   J2 0.000 0.149 0.567 0.000 0.000 0.000 0.000 0.00
##   I1 0.000 0.000 0.149 0.604 0.000 0.000 0.000 0.00
##   I2 0.000 0.000 0.000 0.235 0.560 0.000 0.000 0.00
##   SA 0.000 0.000 0.000 0.000 0.225 0.678 0.000 0.00
##   A1 0.000 0.000 0.000 0.000 0.000 0.249 0.851 0.00
##   A2 0.000 0.000 0.000 0.000 0.000 0.000 0.016 0.86
```

The numbers in the matrix (called ***matrix elements*** or ***transitions***) describe the probability of moving FROM stages in each column TO stages in each row, within the time interval chosen. For example, for this desert tortoise matrix, in any year a subadult (stage 6) has approimately 24.9% probability of becoming an adult (stage 7): this may be called a growth or progression transition. Likewise, in any year a subadult has about 67.8% chance of staying a subadult: this is called a stasis transition. This means that 100 - (67.8 + 24.9) = 7.3% of subadults die every year. There are different types of transitions: in this matrix there are also fecundity transitions which describe offspring production, and subadults produce on average 1.3 offspring per year. Other species may have different transitions, including skipping stages through fast growth, shrinkage or fission (especially in modular organisms, e.g. most plants, corals), or asexual reproduction.

Matrix elements combine underlying ***vital rates*** such as survival, growth and reproduction. For example, the subadult to adult transition (24.9%) combines probability of growing to adult size in one year, and probability of surviving the year. The subadult fecundity (1.3) combines the average number of offspring produced by subadults per year, and probability that those offspring survive the year.

[1] Doak et al. (1994) Ecol. Appl., 4, 446-460.

Figure 1: A baby desert tortoise

---

2. Deterministic projections

The core function for understanding population dynamics is the `project` function. It can be used to understand a number of different types of population dynamics, including deterministic models, stochastic models, long-term dynamics and short-term dynamics.

In a deterministic model, there is no density dependence, and no stochasticity: the vital rates of the model, and therefore the matrix elements, don't change from timestep to timestep (in a density dependent model, they would change according to the size of all or part of the population, and in a stochastic model they would change randomly at each iteration of the model to incorporate random environmental or demographic variation). The projected population dynamics (time series of population size and structure over time) come from multiplying the matrix and the starting poulation vector:

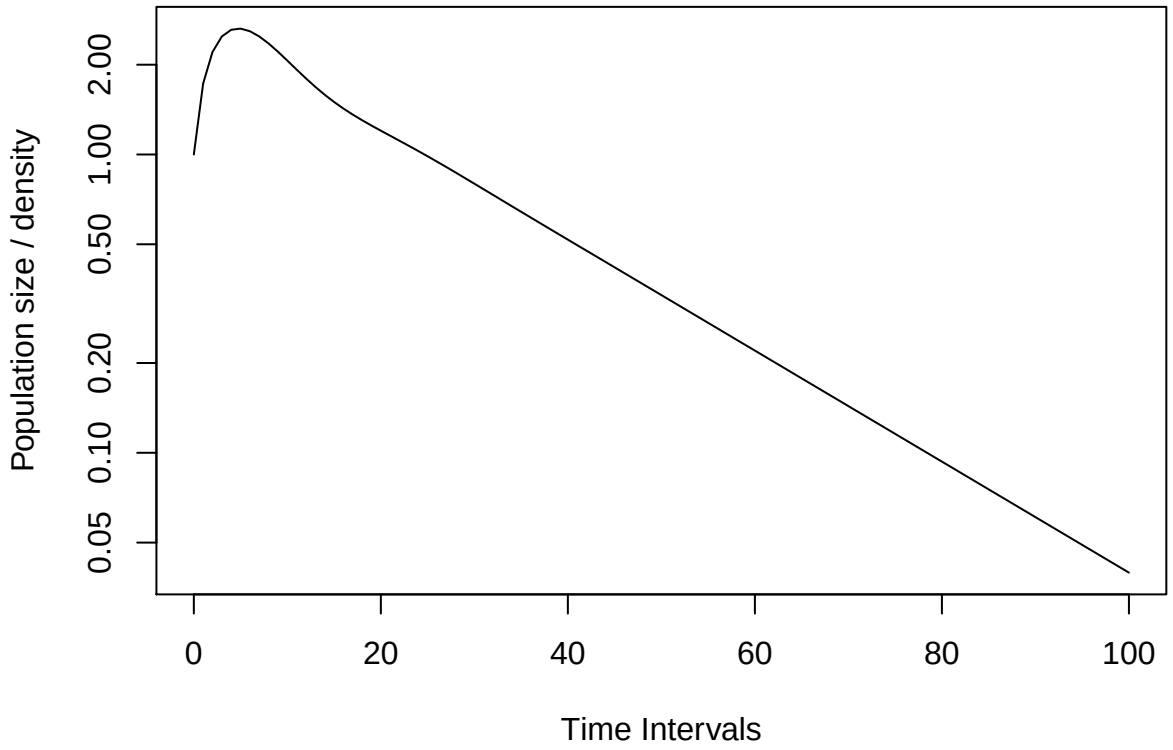
$$\mathbf{n}_t = \mathbf{A}^t \mathbf{n}_0$$

That is, the population size at time t $\mathbf{n}_t$ is equal to projection matrix $\mathbf{A}$ multiplied by the population size at time 0, $\mathbf{n}_0$. We have the matrix, but not a vector. We will:
- choose a vector using a random uniform distribution
- project this vector using the `project` function from `popdemo`
- plot the projection

```
Tortvec1 <- runif(8)
Tortvec1 <- Tortvec1/sum(Tortvec1) #scales the vector to sum to 1
( Tortp1.1 <- project(Tort, Tortvec1, time = 100) )
##   1 deterministic population projection over 100 time intervals.
##
```

```
##     [1] 1.00000 1.73014 2.20671 2.48719 2.61876 2.64205 2.58950 2.48700
##     [9] 2.35524 2.21036 2.06430 1.92520 1.79791 1.68463 1.58562 1.49984
##    [17] 1.42547 1.36043 1.30263 1.25020 1.20162 1.15571 1.11164 1.06890
##    [25] 1.02719 0.98641 0.94657 0.90773 0.87000 0.83349 0.79827 0.76440
##    [33] 0.73191 0.70080 0.67105 0.64261 0.61545 0.58950 0.56469 0.54097
##    [41] 0.51828 0.49657 0.47577 0.45585 0.43676 0.41847 0.40094 0.38414
##    [49] 0.36804 0.35261 0.33782 0.32365 0.31008 0.29707 0.28460 0.27266
##    [57] 0.26123 0.25027 0.23977 0.22971 0.22008 0.21085 0.20200 0.19353
##    [65] 0.18542 0.17764 0.17019 0.16305 0.15621 0.14966 0.14338 0.13737
##    [73] 0.13161 0.12609 0.12080 0.11573 0.11088 0.10623 0.10177 0.09751
##    [81] 0.09342 0.08950 0.08575 0.08215 0.07870 0.07540 0.07224 0.06921
##    [89] 0.06631 0.06353 0.06086 0.05831 0.05586 0.05352 0.05128 0.04913
##    [97] 0.04707 0.04509 0.04320 0.04139 0.03965
```

```
plot(Tortp1.1, log = "y")
```



The `project` function should usually have a matrix passed to the `A` argument, and a `vector`, which can be a single vector, multiple column vectors in a matrix, `"n"` (a default option which projects a set of stage-biased vectors) or `"diri"` (which projects large set of random vectors). We will encounter all these options in this vignette. The `time` argument gives the number of projection intervals, but the output will have length `time` + 1, because the population size at time 0 is included.

The `project` function returns an object of class 'Projection', containing the overall population size over time. In this case, this is a single vector but for objects containing multiple projections (as we'll encounter later), the projections are held in a matrix with one column per projection (therefore the number of rows is equal to `time + 1`).

The 'Projection' object also includes information on the time series of population vectors (i.e. (st)age-specific number / density). For an object containing a single projection, each row represents one timestep and each column represents one (st)age. We will access vectors for time intervals from 0 to 10:

```
vec(Tortp1.1)[1:11, ]
##              Yr     J1      J2      I1      I2      SA      A1      A2
##    [1,] 0.01731 0.2210 0.06048 0.05400 0.23287 0.07589 0.07183 0.26665
##    [2,] 0.92617 0.1377 0.06722 0.04163 0.14310 0.10385 0.08002 0.23047
##    [3,] 0.88575 0.7412 0.05863 0.03516 0.08992 0.10260 0.09396 0.19948
##    [4,] 0.83210 1.0545 0.14368 0.02997 0.05861 0.08980 0.10551 0.17306
##    [5,] 0.77040 1.1937 0.23858 0.03951 0.03987 0.07407 0.11215 0.15052
##    [6,] 0.70517 1.2284 0.31313 0.05941 0.03161 0.05919 0.11388 0.13124
##    [7,] 0.63972 1.2014 0.36058 0.08254 0.03166 0.04724 0.11165 0.11469
##    [8,] 0.57723 1.1392 0.38346 0.10358 0.03713 0.03916 0.10678 0.10042
##    [9,] 0.52040 1.0592 0.38717 0.11970 0.04513 0.03490 0.10062 0.08807
##   [10,] 0.47093 0.9732 0.37735 0.12999 0.05340 0.03382 0.09432 0.07735
##   [11,] 0.42950 0.8890 0.35897 0.13474 0.06045 0.03494 0.08868 0.06803
```

The time series of stage 2 sizes is:

```
vec(Tortp1.1)[ , 2]
##     [1] 0.22098 0.13769 0.74121 1.05446 1.19366 1.22841 1.20142 1.13924
##     [9] 1.05925 0.97320 0.88899 0.81158 0.74360 0.68593 0.63820 0.59926
##    [17] 0.56756 0.54145 0.51940 0.50008 0.48245 0.46576 0.44951 0.43342
##    [25] 0.41737 0.40135 0.38544 0.36972 0.35431 0.33931 0.32480 0.31084
##    [33] 0.29746 0.28467 0.27247 0.26084 0.24976 0.23919 0.22911 0.21949
##    [41] 0.21029 0.20149 0.19306 0.18499 0.17726 0.16984 0.16273 0.15592
##    [49] 0.14938 0.14312 0.13712 0.13136 0.12585 0.12057 0.11551 0.11066
##    [57] 0.10602 0.10157 0.09731 0.09323 0.08932 0.08557 0.08198 0.07854
##    [65] 0.07525 0.07209 0.06907 0.06617 0.06340 0.06074 0.05819 0.05575
##    [73] 0.05341 0.05117 0.04903 0.04697 0.04500 0.04311 0.04130 0.03957
##    [81] 0.03791 0.03632 0.03480 0.03334 0.03194 0.03060 0.02932 0.02809
##    [89] 0.02691 0.02578 0.02470 0.02366 0.02267 0.02172 0.02081 0.01994
##    [97] 0.01910 0.01830 0.01753 0.01680 0.01609
```

When a Projection object contains multiple projections, the vectors are contained in a 3-dimensional array. As for a single projection, the first dimension is time and the second is (st)age, whilst the third represents each separate population projection. Therefore, for example, the first 10 timesteps of the third projection would be accessed using `object[1:11, ,3]`; the sizes of the second stage for all projections would be accessed using `object[ , 2, ]`; the size of the second stage at the 10th timestep of the second stage in the third projection would be `object[1:11, 2, 3]`.

Population growth in an MPM is geometric: when you plot population size on a log scale (as we have here), it's easy to see that the population settles to a stable geometric rate of decline. At this point, the population also has a fixed structure: the relative numbers of individuals in each stage don't change (although the absolute numbers do; this is what causes the population decline). These stable long-term dynamics are often called ***asymptotic dynamics***. The short-term dynamics that happen before this stable state is reached and are different to asymptotic dynamics are called ***transient dynamics***. In this exercise we'll explore both asymptotic and transient dynamics of deterministic MPM models.

ii. EXTRAS...
'Projection objects contain further information that can be accessed in similar ways to the `vec()` function. Type `?Projection-class` to see further options.

Try altering the parameters in the `project` function: change the amount of time the model is projected using the `time` argument, or change the population vector using the `vector` argument. Have a look at how the asymptotic dynamics don't change as the vector changes, but observe how the transient dynamics change. See whether changing the vector changes the amount of time taken to reach stable state. Type `?project` to see further options (some of them will be covered in this vignette).

The `plot` function takes any of the usual graphical parameters: try changing the lines to points (`type` argument), changing the line colour, type or thickness(`col`, `lty`, `lwd`), changing the box around the plot (`bty`), or any other graphical parameters (see `?par`). There are further options available specifically for 'Projection' objects: see `?Projection-plots`.

3. Asymptotic dynamics

In the long term, the population dynamics are described by the dominant eigendata of the matrix:

```
eigs(Tort, "all")
##   $lambda
##   [1] 0.9581
##
##   $ss
##   [1] 0.22166 0.40585 0.15463 0.06508 0.03842 0.03087 0.07179 0.01171
##
##   $rv
##   [1] 0.1955 0.2616 0.6866 1.8019 2.7148 4.8029 4.3813 5.1237
```

3.1 Asymptotic growth

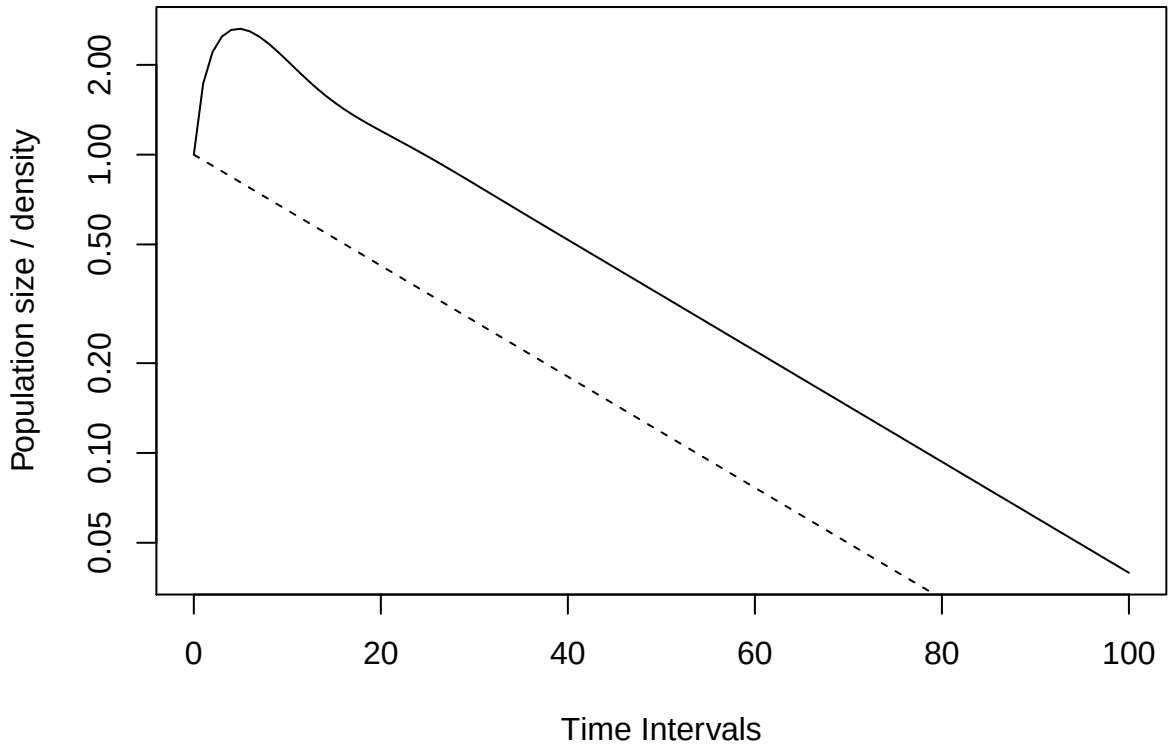
The `eigs` function returns the dominant eigendata (or "eigenstuff") of the matrix[2]. The growth rate is commonly referred to as ***lambda*** (λ). In this case, $\lambda < 1$ which means the population declines. If $\lambda = 1$ the population neither grows nor declines, and If $\lambda > 1$, the population grows.

3.2 Asymptotic population structure

The rest of the eigenstuff describes other aspects of stable dynamics. "ss" refers to the stable structure: this is the ratio of numbers of individuals in each stage once the population reaches stable state, and the vector is usually denoted with **w**. A population with this structure, then it will grow/decline at the stable rate from the outset (populations starting with a stable structure will always be shown with dashed lines):

```
Tortw <- eigs(Tort, "ss")
Tortpw <- project(Tort, Tortw, time = 100)
lines(0:100, Tortpw, lty = 2)
```

[2] for a detailed introduction to matrix models and eigendata, see Caswell (2001) Matrix Population Models, Sinauer.



$\mathbf{w}$ is scaled so that $||\mathbf{w}||_1 = 1$.[3]

3.3 Reproductive value

"rv" refers to the reproductive value vector, and is usually denoted with $\mathbf{v}$. This is the contribution that each stage makes to stable growth (through survival, growth and reproduction). Stages with high current and reproduction and survival have high reproductive value. $\mathbf{v}$ is scaled so that $\mathbf{v}^{\mathrm{T}}\mathbf{w} = 1$[4] when $||\mathbf{w}||_1 = 1$.

iii. EXTRAS...
The `eigs` *function allows you to choose what eigenstuff you want to calculate. Try replacing "all" with one or more of "lambda", "ss", or "rv". If you want to look at subdominant eigenstuff then the base function "eigen" does this, with* `eigen(A)` *giving the right eigenvectors and* `eigen(t(A))` *giving the left eigenvectors.*

---

4. Transient dynamics

Before settling to stable growth rate, a population will grow, decline or fluctuate in growth at rates faster and/or slower than asymptotic growth. We can see this in our plot of population dynamics of the desert tortoise. These population dynamics are called ***transient dynamics*** and are a little harder to characterise than asymptotic dynamics as they're so variable and depend on the population structure[5].

[3] $||\mathbf{w}||_1$ is the one-norm of $\mathbf{w}$, which is equal to its sum.
[4] $\mathbf{v}^{\mathrm{T}}$ is the transpose of $\mathbf{v}$: not to be confused with an exponent!
[5] Stott et al. (2011) Ecol. Lett., 14, 959-970 contains a review on measuring transient dynamics in MPMs.

4.1 Standardisations

`popdemo` contains functions that calculate deviations from stable growth at various points along the population projection. These ***transient indices*** make two standardisations: starting population vector $\mathbf{n}_0$ is scaled by $||\mathbf{n}_0||_1$ so that it sums to 1:

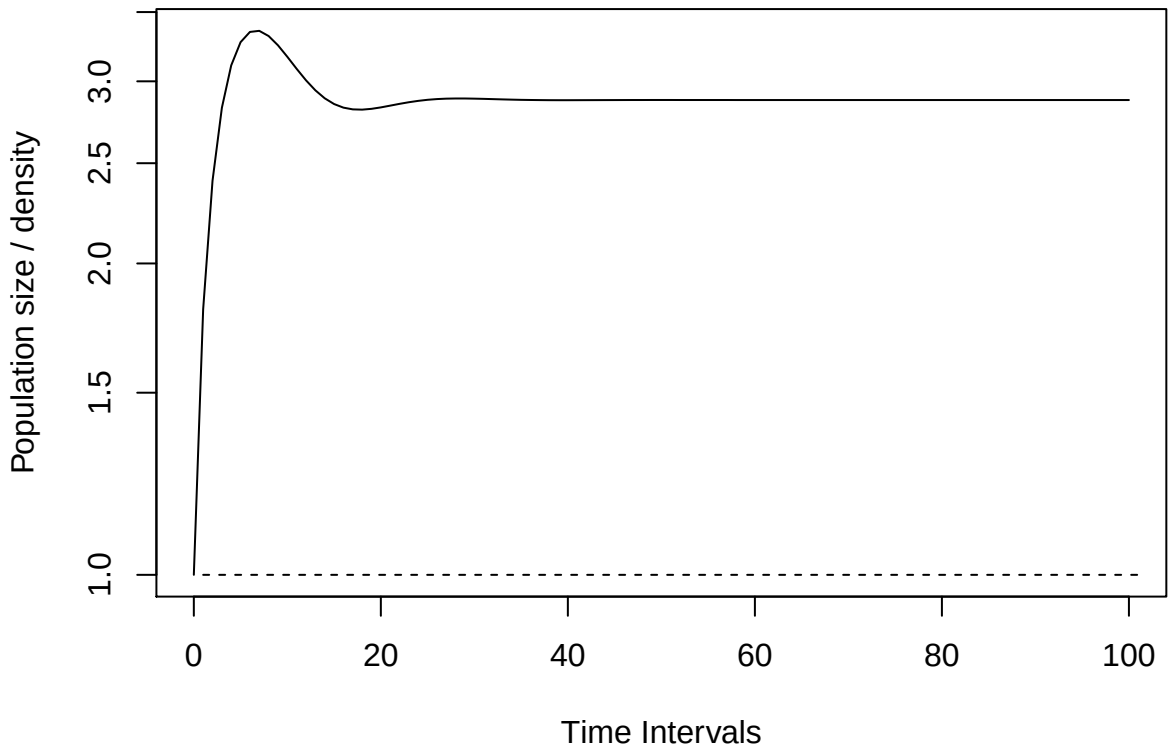
$$\hat{\mathbf{n}}_0 = \frac{\mathbf{n}_0}{||\mathbf{n}_0||_1}$$

The projection matrix is scaled by λ, so that lambda of the scaled matrix becomes 1:

$$\hat{\mathbf{A}} = \frac{\mathbf{A}}{\lambda}$$

These standardisations allow comparison of transient dynamics between populations with different sizes, and with different long-term dynamics. We can visualise this this in a ***standardised*** population projection:

```
Tortp1.1s <- project(Tort, Tortvec1, time = 100,
                     standard.A = TRUE, standard.vec = TRUE)
Tortpws <- project(Tort, Tortw, time = 100,
                   standard.A = TRUE, standard.vec = TRUE)
plot(Tortp1.1s, log = "y")
lines(Tortpws, lty = 2)
```



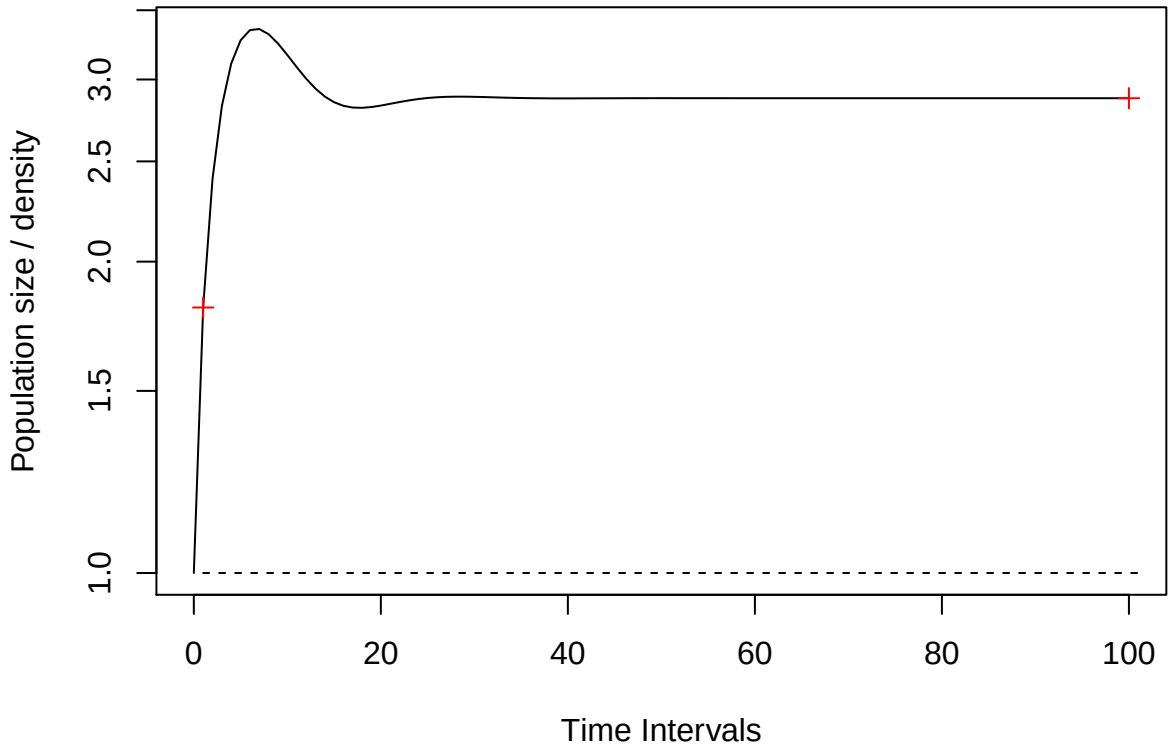
Transient dynamics vary according to the starting population vector. If there is an overrepresentation of individuals in stages with high survival and/or fertility, then the population will grow faster (or decline slower) than the stable rate (this is called 'amplification'). If there is an overrepresentation of individuals in stages with low survival and zero/low fertility, then the population will grow slower (or decline faster) than the stable rate (this is called 'attenuation'). Indices of amplification are always >1, which means that log-transformed indices of amplification are always >0. Indices of attenuation are always <1 but >0. This means that log-transformed indices of attenuation are always <0.

4.2 Transient indices

We can measure transient density at any time along the projection (we say "density" because a standardised dynamic is no longer directly equivalent to size... but this is not the same thing as spatial density!). But in comparative analysis, to make things comparable between populations, we can use comparable timepoints.

Two possibilities are at t = 1 and at $t \to \infty$. These indices are called ***reactivity*** and ***inertia*** respectively:

```
( r1 <- reac(Tort, Tortvec1) )
##   [1] 1.806

( i1 <- inertia(Tort, Tortvec1) )
##   [1] 2.878

points(c(1, 100), c(r1, i1), pch = 3, col = "red")
```



4.2.1 Reactivity

Reactivity (`r1`) is the population size in the first timestep of the projection (one year), relative to a population with stable growth. A reactivity of 2 would mean that in the first timestep, the population grows twice as fast as its stable growth rate. Reactivity is calculated using:

$$reactivity = P_1 = ||\hat{\mathbf{A}}\hat{\mathbf{n}}_0||_1$$

As we can see, measurements of transient dynamics use the standardisations we encountered earlier. These "standardisations" mean that transient indices are measured relative to long-term population growth, and initial population size. The interpretation of this is that transient indices measure the ratio of population size compared to a population growing at a stable rate.

4.2.2 Inertia

Inertia (`i1`) is the ratio of the size of the population in the long-term, relative to a population with stable growth. An inertia of 4 would mean that after the transient period, the population settles to a size 4 times as large as a population that grows with stable rate. Inertia is calculated using:

$$inertia = P_\infty = \frac{\mathbf{v}^{\mathrm{T}}\hat{\mathbf{n}}_0||\mathbf{w}||_1}{\mathbf{v}^{\mathrm{T}}\mathbf{w}}$$

$\mathbf{w}$ and $\mathbf{v}$ are the right and left eigenvectors as described in the "Asymptotic dynamics" section above. If $\mathbf{w}$ and $\mathbf{v}$ are scaled as described, then $inertia = \mathbf{v}^{\mathrm{T}}\hat{\mathbf{n}}_0$.

4.2.3 Maximum amplification and attenuation

Comparable points at which to measure transient dynamics don't have to be at the same time of the projection. ***maximum amplification*** and ***maximum attenuation*** describe the largest and smallest population sizes (relative to long-term growth λ). Maximum amplification and maximum attenuation can occur at any point along the projection. Use `return.t = TRUE` to return the time at which they occur, as well as their value.

Some populations only have a maximum amplification (because they never attenuate), some only have a maximum attenuation (because they never amplify), and some have both. With this proviso in mind, these indices are calculated using:

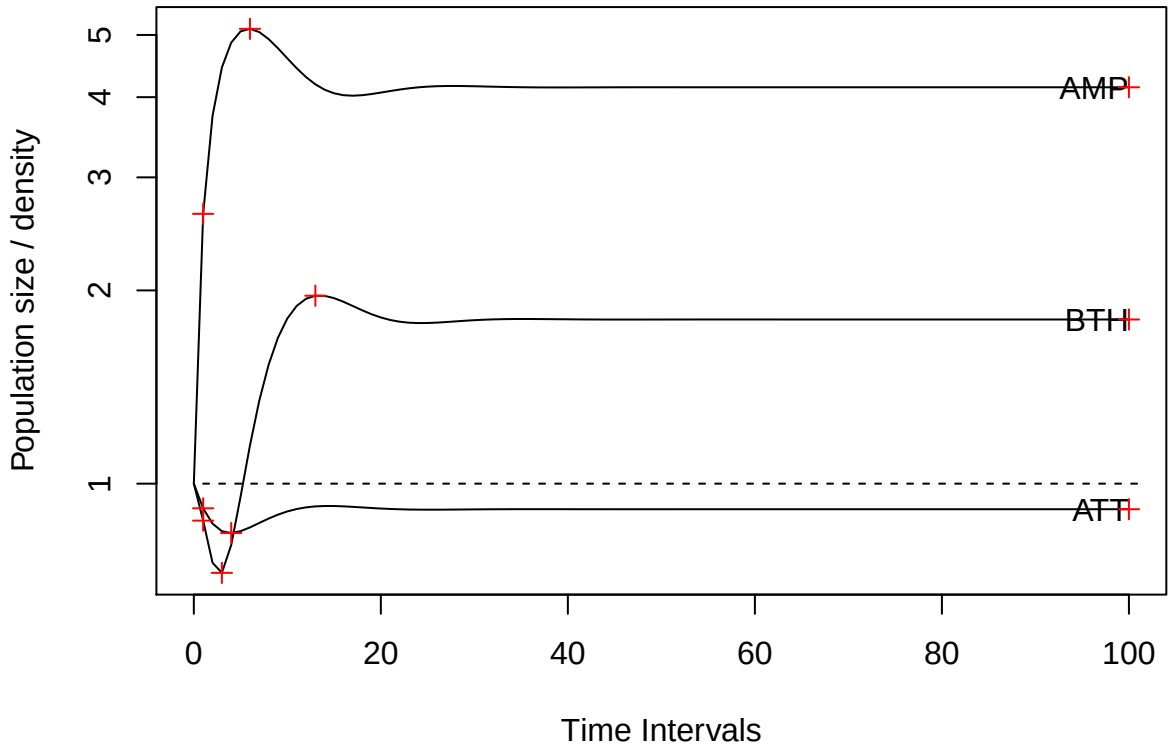
$$maximum\ amplification = \bar{P}_{max} = \max_{t>0} ||\hat{\mathbf{A}}^t\hat{\mathbf{n}}_0||_1$$

$$maximum\ attenuation = \underline{P}_{min} = \min_{t>0} ||\hat{\mathbf{A}}^t\hat{\mathbf{n}}_0||_1$$

If we want to compare several different population structures, then it's possible to project several simultaneously. We will:
- create two extra vectors; one which amplifies and one which attenuates
- bind these with the random vector into a matrix with 3 columns
- project the model with this matrix passed to the `vec` argument
- calculate reactivity, inertia and maximum amplification / attenuation for the extra vectors
- plot these on the graph

```
TortAMP <- c(1, 1, 2, 3, 5, 8, 13, 21) #a population that amplifies
TortATT <- c(21, 13, 8, 5, 3, 2, 1, 1) #a population that attenuates
TortBTH <- c(0, 0, 0, 1, 0, 0, 0, 0) #a population that does both
Tortvec3 <- cbind(AMP = TortAMP,
                  ATT = TortATT,
                  BTH = TortBTH)
Tortp3.1 <- project(Tort, Tortvec3, time = 100,
                    standard.A = TRUE, standard.vec = TRUE)
plot(Tortp3.1, log = "y"); lines(Tortpws, lty = 2)
( r3 <- apply(Tortvec3, 2, reac, A = Tort) )
##      AMP    ATT    BTH
##   2.6319 0.9153 0.8757
( r3t <- rep(1, 3) )
##   [1] 1 1 1

( i3 <- apply(Tortvec3, 2, inertia, A = Tort) )
##      AMP    ATT    BTH
##   4.1442 0.9123 1.8019
( i3t <- rep(100, 3) )
##   [1] 100 100 100

( max3 <- apply(Tortvec3[,c(1,3)], 2, maxamp, A = Tort) )
##     AMP   BTH
```

```
##   5.111 1.962
( max3t <- apply(Tortvec3[,c(1,3)], 2, function(x){
                maxamp(vector = x, A = Tort, return.t = TRUE)$t}) )
##   AMP BTH
##     6  13

( min3 <- apply(Tortvec3[,c(2,3)], 2, maxatt, A = Tort) )
##      ATT    BTH
##   0.8377 0.7261
( min3t <- apply(Tortvec3[,c(2,3)], 2, function(x){
                maxatt(vector = x, A = Tort, return.t = TRUE )$t}) )
##   ATT BTH
##     4   3

points(c(r3t, i3t, max3t, min3t),
       c(r3, i3, max3, min3),
       pch = 3, col = "red")
```



iv. EXTRAS...
Some further functionality offered by the ***reac****,* ***inertia****,* ***maxamp*** *and* ***maxatt*** *functions. Try using, for example,* ***return.N = TRUE*** *to give the transient population size (including influences of initial population size and asymptotic population growth), and use this to plot transient indices on non-standardised population projections.*

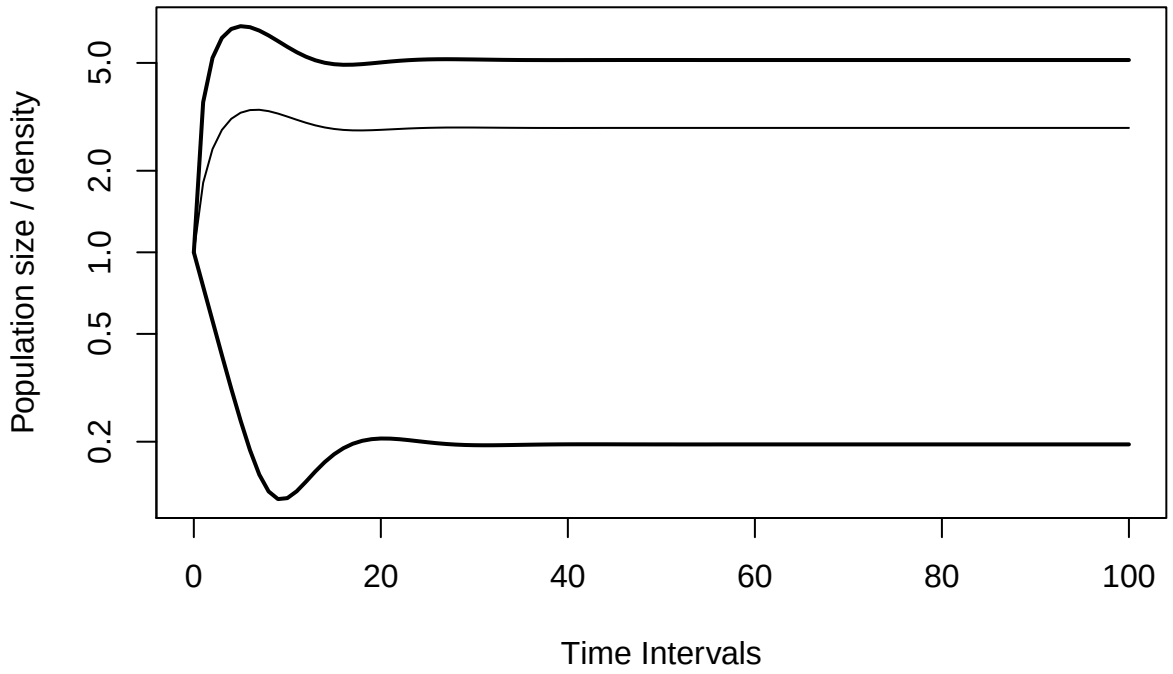
5. Transient bounds

Transient dynamics are very variable, and there are an infinite number of different starting population vectors. Sometimes we don't know what the population structure is (making a fair census of plants and animals is difficult!), but it is possible to get an idea of what the transient dynamics will be like. ***Transient bounds*** capture the outer extremes of transient dynamics; all population trajectories lie within the bounds.

5.1 Plotting transient bounds

It's easy to plot the bounds by adding `bounds = TRUE` when plotting a population projection:

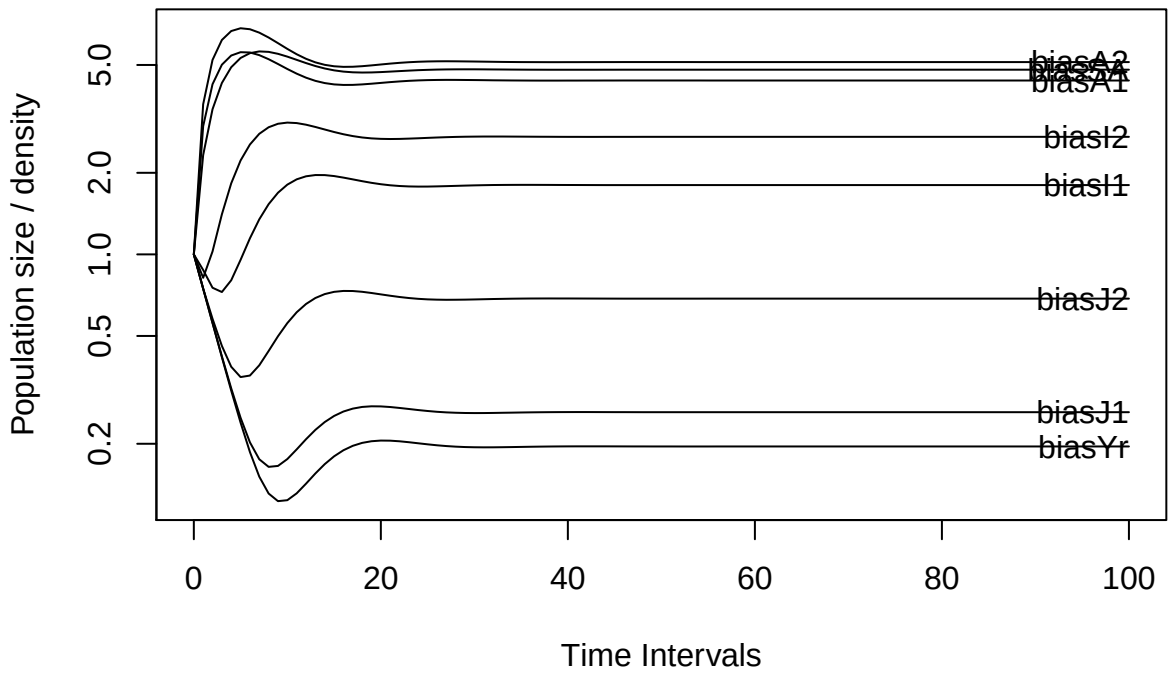
```
plot(Tortp1.1s, log = "y", bounds = TRUE)
```



5.1.1 Stage-biased projections

The bounds are calculated from the set of projections of "stage-biased" vectors, each of which has 100% of individuals in one stage. For example, the stage-biased vectors for a 3-by-3 matrix are [1,0,0], [0,1,0] and [0,0,1]. To project these, use `vector = "n"`; this is the default value, so in fact if `vector` isn't set then the stage-biased vectors will be projected automatically:

```
plot(project(Tort, standard.A = TRUE), log = "y")
```



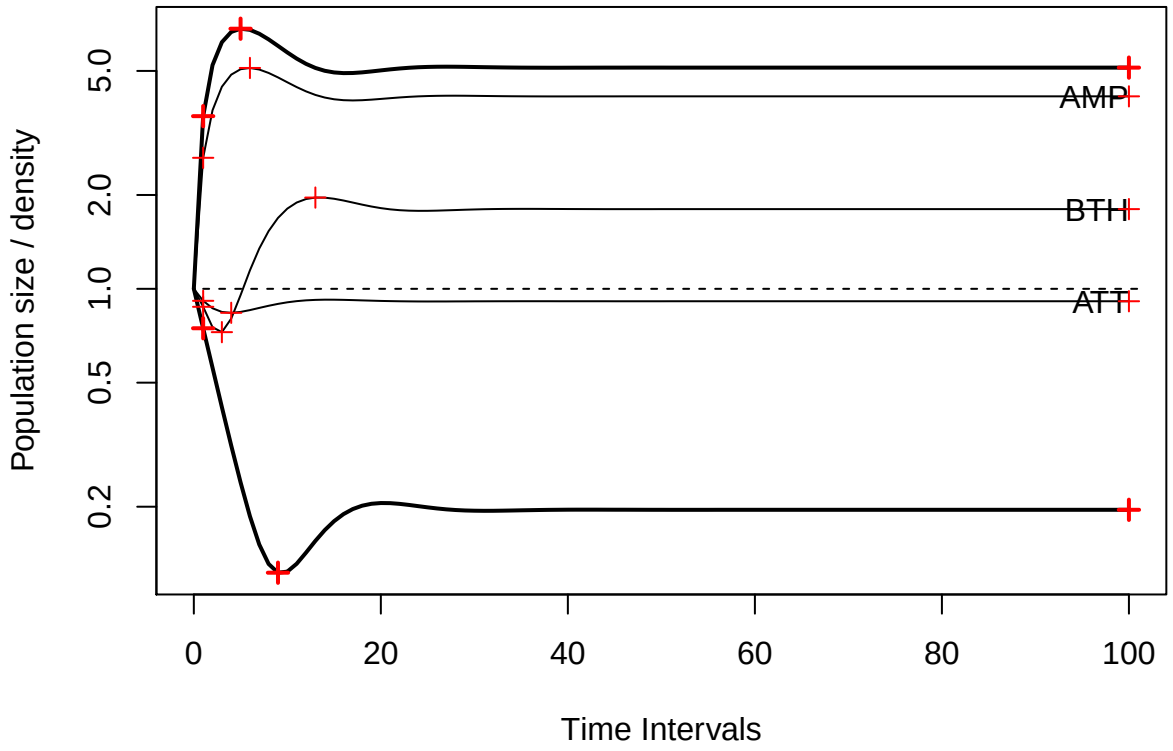
The top and bottom lines are the bounds on population dynamics. No deterministic projection can be outside these lines.

5.2 Bounds on transient indices

When calculating transient indices, use `bound = "upper"` or `bound = "lower"` to calculate bounds on reactivity and inertia. To calculate bounds on maximum amplification or attenuation, just don't pass anything to `vector`:

```
plot(Tortp3.1, log = "y", bounds = TRUE)
lines(Tortpws, lty = 2)
( ruprB <- reac(Tort, bound = "upper") )
##   [1] 3.58

( rlwrB <- reac(Tort, bound = "lower") )
##   [1] 0.7473

( iuprB <- inertia(Tort, bound = "upper") )
##   [1] 5.124

( ilwrB <- inertia(Tort, bound = "lower") )
##   [1] 0.1955

( maxB <- maxamp(Tort, return.t = TRUE) )
##   $maxamp
##   [1] 6.83
##
##   $t
```

```
##   [1] 5

( minB <- maxatt(Tort, return.t = TRUE) )
##   $maxatt
##   [1] 0.1228
##
##   $t
##   [1] 9

points(c(rep(1, 5), rep(100, 5), max3t, maxB$t, min3t, minB$t),
       c(r3, ruprB, rlwrB, i3, iuprB, ilwrB, max3, maxB$maxamp, min3, minB$maxatt),
       pch = 3, col = "red",
       lwd = c(rep(c(1, 1, 1, 2, 2), 2), rep(c(1, 1, 2) ,2 )) )
```



Using ρ_1 to refer to reactivity bounds, ρ_∞ to refer to inertia bounds, ρ_{max} to refer to the maxmimum amplification bound, ρ_{min} to refer to the maximum attenuation bound, and with overbars to refer to upper bounds and underbars to refer to lower bounds[6]:

$$\bar{\rho}_1 = ||\hat{\mathbf{A}}||_1 \qquad and \qquad \underline{\rho}_1 = minCS(\hat{\mathbf{A}})$$
$$\bar{\rho}_\infty = \frac{v_{max}||\mathbf{w}||_1}{\mathbf{v}^{\mathrm{T}}\mathbf{w}} \qquad and \qquad \underline{\rho}_\infty = \frac{v_{min}||\mathbf{w}||_1}{\mathbf{v}^{\mathrm{T}}\mathbf{w}}.$$
$$\bar{\rho}_{max} = \max_{t>0} ||\hat{\mathbf{A}}^t||_1 \qquad and \qquad \underline{\rho}_{min} = \min_{t>0}(minCS(\hat{\mathbf{A}}^t))$$

When $\mathbf{w}$ and $\mathbf{v}$ are scaled as decribed in the "Asymptotic dynamics" section, then $\bar{\rho}_\infty = v_max$ and $\bar{\rho}_\infty = v_min$.

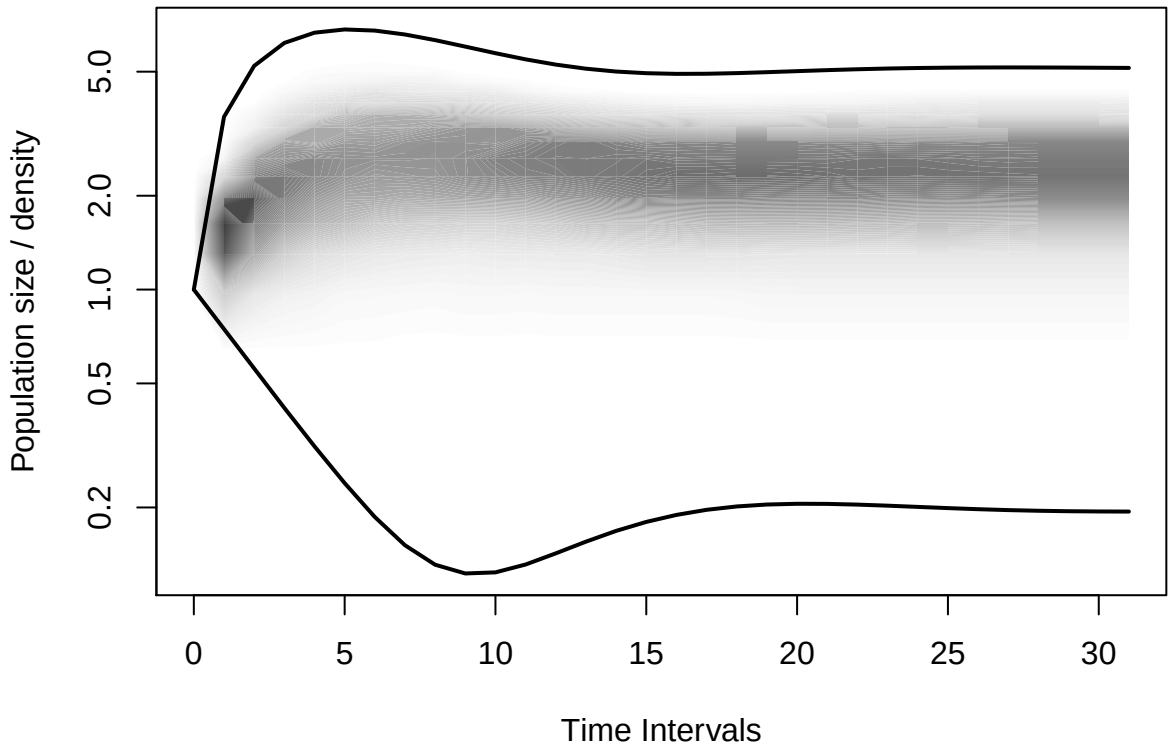
v. EXTRAS...

We've seen a little bit of the diversity of different transient dynamics that can be shown by a model. But

[6] $||\hat{\mathbf{A}}||_1$ is the one-norm of $\hat{\mathbf{A}}$, is equal to its maximum column sum. $minCS$ describes the minimum column sum. v_{max} and v_{min} are the maximum and minimum entries of $\mathbf{v}$, respectively.

there are an infinite number of different possible starting population vectors. If population vectors are drawn at random, it is possible to shade in the space within the transient bounds to work out the likelihood of different transient densities. This can be done by using `vector = "diri"` *in the* `project` *function, which draws populations at random from a dirichlet distribution and projects them. Plot this using* `plottype = "shady"`*:*

```
Tortpd <- project(Tort, "diri", time = 31,
                  standard.A = TRUE)
plot(Tortpd, plottype = "shady", bounds = T, log = "y")
```



Dirichlet projections have their own options within the `project` *and* `plot` *functions: see the "Projection-class" and "project" help pages.*