IPMpack: an R package for demographic modeling with Integral Projection Models (v. 2.1)

Jessica Metcalf, Sean M. McMahon, Rob Salguero-Gomez,
Eelke Jongejans, Cory Merow

March 10, 2014

The goal of IPMpack is to provide a suite of demographic tools based on Integral Projection Models (IPMs) to support biologists interested in making projections for populations where demography is strongly linked to a changing continuous variable, such as size. The package includes functions that can take data, such as size or age, as well as environmental covariates, and build models of growth, survival and fecundity. Functions are defined that then take these statistical models and construct IPMs. IPMpack has tools that compare different functional forms for the underlying statistical models, plotting them and returning model scores, as well as tools for diagnostic tests of the IPM models themselves. There are also methods to build population models for varying environments, estimate longevity and passage time, sensitivity and elasticity (of either parameters or matrix elements), and much more.

The basic ideas of IPMpack are published in Methods in Ecology and Evolution (Metcalf et al. 2013). Please cite that paper if you want to refer to IPMpack. This vignette is intended to introduce the concepts of IPMs as well as the implementation of IPMpack to biologists with a wide range of quantitative skills. This vignette is for IPMpack version 2.1, and so we encourage users to contact the IPMpack team at IPMpack@gmail.com with any feedback or mistakes they find. We also host a blog at R-forge (http://ipmpack.r-forge.r-project.org/) that contains news of updates, new features, and announcements of papers and meetings relevant to IPMs.

1 Introduction to Integral Projection Models

An Integral Projection Model (IPM) is a demographic tool that can estimate the dynamics of populations where individuals' fates depend on state variables that are continuous (e.g., weight, diameter at breast height, height, limb length, rosette diameter) or quasi-continuous (e.g., number of leaves, age, number of reproductive structures) and may be a mixture of discrete and continuous variables. IPMs track the distribution of individuals n across these state variables between census times (e.g., year t and year $t+1$) by projecting from models that define the underlying vital rates (e.g., survival, growth, and reproduction) as a function of the (quasi-)continuous state variables. For detailed introductions to IPMs see Easterling et al. (2000), and Ellner & Rees (2006, 2007).

Briefly, an IPM is defined by a kernel K that represents probabilities of growth between discrete or continuous stages, survival across these stages, and the production of offspring and offspring recruitment. For example, in the simplest case, where the population is structured by a continuous covariate, size, then

$$n(y,t+1) = \int_L^U K(y,x)n(x,t)\,dx \tag{1}$$

where $n(y,t+1)$ is the distribution across size y of both established and new individuals in census time $t+1$, $n(x,t)$ the distribution across size of individuals in census time t, and L and U the lower and upper size limits modeled in the IPM, respectively.

Multiple functional forms for both demographic processes as well as their error structures can be accommodated with IPMpack. The F kernel (equation 4) describes per-capita contributions of reproductive individuals to number of new individuals at the next census. Multiple size-dependent or size-independent vital rates can be fitted within the F kernel, reflecting for example reproductive probability, number of reproductive structures (e.g. flowers in plants, basidia in fungi), number of propagules within reproductive structure (e.g. seeds in inflorescences), and so on. Additionally, a range of constants (c_1, c_2, ...) can be included if there are no state-dependent data. All of these will be multiplied to obtain the eventual fertility for individuals of each size. Finally, the F kernel definition includes a probability density function describing the size of offspring recruiting into the population, f_d.

From equation 1:

$$n(y,t+1) = \int_L^U K(y,x)n(x,t)dx = \int_L^U [P(y,) + F(y,x)]n(x,t)dx, \tag{2}$$

where

$$\int_L^U P(y,x)n(x,t)dx = \int_L^U surv(x)growth(y,x)dx, \tag{3}$$

and

$$\int_L^U F(y,x)n(x,t)dx = \int_L^U c_1c_2c_3...fec1(x)fec2(x)fec3(x)...f_d(y,x)dx \tag{4}$$

After numerically solving these kernels, key ecological and evolutionary quantities such as the population rate of increase λ, the stable population size structure, the net reproductive rate R_0, and many others can be estimated (see Caswell 2001 for more a comprehensive discussion).

Essentially, the same tools are available for IPMs as for discrete projection matrices (matrix population models), e.g., estimation of population growth rate, sensitivities, elasticities, life table response experiment [LTRE] analyses, passage time calculations, etc. (Caswell 2001, Cochran & Ellner 1992, and others); as well as some additional tools based on exploring the impact of the underlying statistical relationships. The main difference between an IPM and a matrix model is that while in discrete projection matrices the number of classes (i.e., number of stages in the

life cycle of the study species) must be defined `a priori`, IPMs impose the discretization of the three-dimensional surface defined by equation 1 in the last step for the purposes of numerical integration. This produces a typically large matrix (e.g., 100 x 100 cells) that is more robust to biases from matrix dimensionality (Zuidema et al. 2010, Salguero-Gomez & Plotkin 2010) and sample size (Ramula et al. 2009) than classical matrix models.

The goal of IPMpack is to provide a centralized set of quantitative techniques based on IPMs to help ecologists and evolutionary biologists model populations. IPMpack can accommodate multiple vital rates from complex life cycles all grouped into two main sub-kernels: P and F (equation 2) [1].

This vignette walks through the steps of a basic IPM analysis. We first describe the kind of data necessary to build an IPM. If users begin 'from scratch', they must input data in a specific format (described below). However it is possible to jump past this step and use IPMpack capabilities on IPMs that were developed outside of IPMpack. That is, if a user wants quick figures, summary statistics, other analyses on an IPM matrix already built, IPMpack can readily accommodate that. However there are some features that, because of the object-oriented coding, require some specific structures (and other features that do not). Please refer to the manual files, and the rest of this vignette for this information. The vignette will begin with data input. We will then walk through how to build and analyse a basic IPM model. More complex models will be introduced later, as well as comparative model testing.

2 Getting started: setting up the data for IPMpack

For users who prefer to define IPM matrices using their own statistical tools, there is no requirement for the data to be in any particular format, and most of the functions in IPMpack will operate on the matrices directly (e.g., life expectancy, sensitivity of matrix elements, etc.). However, to use IPMpack's full capacities, the individual-level demographic data must be organized in a `data frame` (a class of object in R [see help file for 'data.frame' in base]), where each row represents one observation of an organism in the population at one census time t with the following potential column names:

- `size`: size of individuals in census time t *

- `sizeNext`: size of individuals in census time $t+1$ *

- `surv`: survival of individuals from census time t to $t+1$ (contains: 0 for death or 1 for survival) *

- `fec1, ...`: as many columns as desired relating size to sexual reproduction. For example, this might be:

    - `fec1`: probability of reproduction (output: 0 for no reproductive or 1 for reproductive)

    - `fec2`: number of reproductive structures (output: 1, 2, 3, ...) when individual is reproductive, that is, when fec1 = 1

    - `fec3`: number of propagules (output: 1, 2, 3, ...) per reproductive structure (e.g. seeds per flower in reproductive plant individual)

    - ...

- `stage`: stage of individuals in census time t, used to distinguish discrete and continuous stages, etc. For rows in the data frame where `size` is not an NA, then this must be the word "continuous". Where `size` is NA, any variety of named discrete stages may be defined (e.g. "seed bank"). If this column is missing, many procedures in IPMpack are designed to simply fill in this column assuming that only "continuous" state variables describe the life cycle of

---

[1]Note that we follow the terminology of the seminal paper by Easterling et al. (2000). The P kernel (equation 3) describes `growth` between demographic censuses conditional on individuals' survival (`surv`).

the species, i.e. there are no discrete stages. For running `makeFecObj`, the column must be a factor. If not supplied, the function will generate this column assuming all individuals are "continuous".

- `stageNext`: stage of individuals in census time $t + 1$, in the simples case, "continuous" or "dead" (which is redundant with "0" in the `surv` column. As above, this column is not essential for many procedures in IPMpack. For running `makeFecObj`, the column must be a factor. If not supplied, the function will generate this column assuming all individuals that are alive are "continuous".

- `number`: number of individuals corresponding to each row in the data frame. For all rows corresponding to movement between continuous stages, this value will be 1, but for movement between discrete stages (e.g., from "dormant seeds" to "seeds ready to germinate") then this number may be > 1, potentially directly reflecting observed individuals in the data. This information avoids having a data frame with a row for every individual in a discrete stage (e.g. seeds). As above, many procedures in IPMpack will simply assume that this value is always 1.

- `covariate`: value of a discrete covariate in census time t, such as light environment at time t, age at t, patch at t, etc.

- `covariateNext`: value of a discrete covariate in census time $t + 1$.

- ...any other covariates of interest, named as desired by the user are possible too (e.g., precipitation, habitat, temperature, etc).

- `offspringNext`: if the size contained in sizeNext corresponds to the size of an offspring, this column will contain either the value "sexual" or "clonal" (depending on whether sexual or clonal reproduction is being considered). If this column exists, rows that take these two values will be excluded from the growth analyses (functions `makeGrowthObj` and variants thereof, see below).

The * symbol above indicates the minimum columns in the data frame required to obtain passage time and life expectancy calculations. These values form the P kernel. If sufficient additional columns are available, a full life-cycle model, containing the F kernel, can be produced and further analyses are possible. Although `size` and `sizeNext` can be transformed, many of the utility functions assume no transformations in columns in the original data frame pertaining to fertility. Transformations can be formally called in various parts of the package and appropriate F matrices built that account for these transformations.

3 The basics: building an IPM

First, the user must install IPMpack from CRAN using `install.packages("IPMpack")` and then load IPMpack into an R session (`library(IPMpack)`) (see help files for problems with installation or loading).

```
> library(IPMpack)
>
```

Next, the user must input demographic data. As mentioned above, most functions of IPMpack require a data file with at minimum columns called `size`, `sizeNext`, `surv`, where 'size' is size at time t, 'sizeNext' is size one census later, and 'surv' is a series of 0s and 1s, indicating if the individual survived or not. In the case of 'size' and 'sizeNext', data can be transformed (e.g., onto a log scale), if appropriate via functions built into IPMpack. For the purpose of learning how to use IPMpack, the user can either use his/her own data (adjusted to have the appropriate headings, as aforementioned), or generate them with a function built into IPMpack:
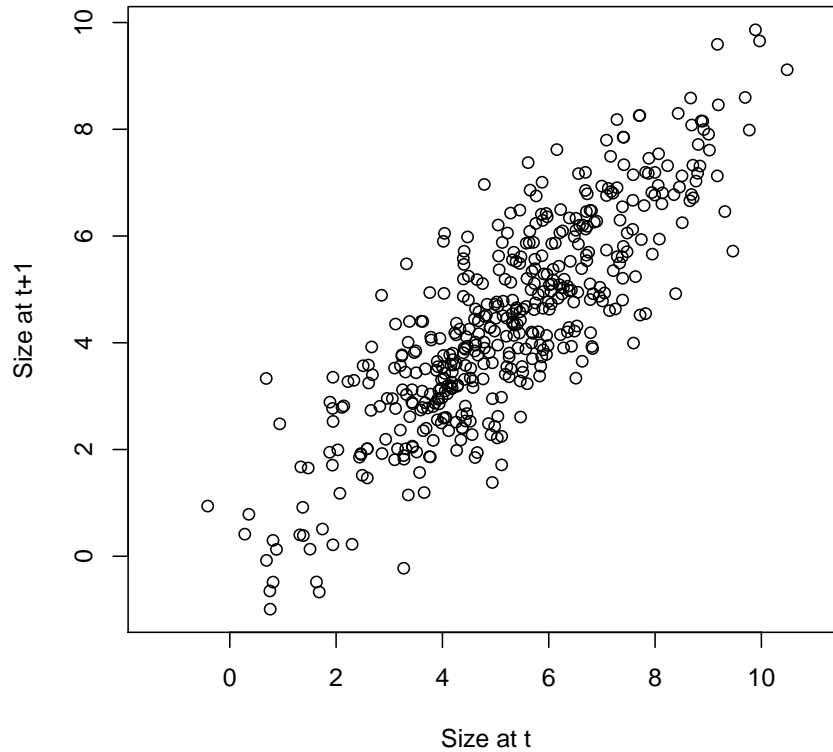
Figure 1: Size at t and size at t+1

```
> dff <- generateData()
```

A quick check indicates that this contains sensible (fictional) information:

```
> head(dff)
```

```
      size  sizeNext surv covariate covariateNext      fec      stage  stageNext
1 5.677142       NA    0         1             0 3.959199 continuous continuous
2 5.751298       NA    0         0             1 7.607546 continuous continuous
3 5.848601 3.561237    1         1             1 0.000000 continuous continuous
4 5.591256 5.204678    1         0             0 4.692577 continuous continuous
5 3.912582 2.945348    1         1             1 0.000000 continuous continuous
6 4.914662 2.277824    1         0             0 0.000000 continuous continuous
```

for simplicity, no discrete covariates are included in this first example. Figure 1 (p. 6) is produced by the following code:

```
> plot(dff$size, dff$sizeNext, xlab = "Size at t", ylab = "Size at t+1")
```

IPMpack is written in object-oriented code, using *S*4 objects. This means that extra object classes are used by IPMpack, with methods assigned to those classes that do particular things to specific objects. An example for those familiar with *R* is the `plot` function. When applied to two vectors, it produces an x-y plot, but when applied to a fitted linear regression, it provides a series

of diagnostic plots. In other words, the 'plot' method is object-specific and does different things to objects of class 'numeric' and objects of class 'lm'.

IPMpack contains defined classes for growth, survival and fertility objects, and associated methods that allow the user to build IPM objects. In addition, this object-oriented structure in IPMpack uses methods from IPM objects to calculate life expectancy, passage times, and other population estimates of interest. The advantage of object-oriented programming is its flexibility: for example, the same machinery can be applied to suites of underlying regression forms and the user can take advantage of pre-existing highly generalized *R* functions, such as `predict`. The needs of any particular dataset may require different object and method definitions. Towards the end of this vignette we also describe how to define a new class and a new method (e.g., a new growth object for a specific life-history structure, and a new growth method applicable to plotting information from that object).

As an example, let us first define objects built as simple polynomial regressions from the generated data. The source code of `generateData` will confirm that the survival data is built around a polynomial logistic regression relating size at t to survival from t to $t+1$, and the growth data is built around a polynomial regression relating size at t to size at $t+1$. To make growth and survival objects that reflect this, the user must implement:

```
> gr1 <- makeGrowthObj(dataf = dff, Formula = sizeNext~size+size2)
> sv1 <- makeSurvObj(dff, Formula = surv~size+size2)
```

In both these functions, the argument `Formula` contains formulas of the type used in linear or logistic regressions in *R*, built around the possible defined range of transforms of `size` currently available (`size2` which is size^2, `size3` which is size^3, and `logsize` which is log(size)). Currently further transforms of `size` are not possible. This function can also be used to fit models that include a single discrete covariate (e.g., light environment, age, etc) as long as this exists in the `dataf` in a column named `covariate`. For instance, the user could model the population dynamics according to `size + covariate` or `size + logsize*covariate`, etc. For more complex analyses, other covariates (time since fire, precipitation, etc) can be fitted as long as they exist within `dataf`. For the growth model, possibilities for the response variable in the `Formula` are: `sizeNext` meaning that the reponse variable is size at the next census time, or `incr` meaning that the response variable is the size increment that has accrued between the two census times (common among tree demographic studies), and `logincr` meaning that the response variable is the log of the size increment that has acrrued between the two census intervals.

Glancing at the source code will confirm that all these functions simply fit a linear regression relating size at t+1 or increment to size at t and covariates for growth, as for survival. The survival and growth objects created have a slot called 'fit' that holds the regression and a slot 'sd' that holds the variance around the regression.

```
> gr1

An object of class "growthObj"
Slot "fit":

Call:
lm(formula = Formula, data = dataf)

Coefficients:
(Intercept)         size         size2
   0.203268     0.770399      0.005991



Slot "sd":
[1] 1.043882
```
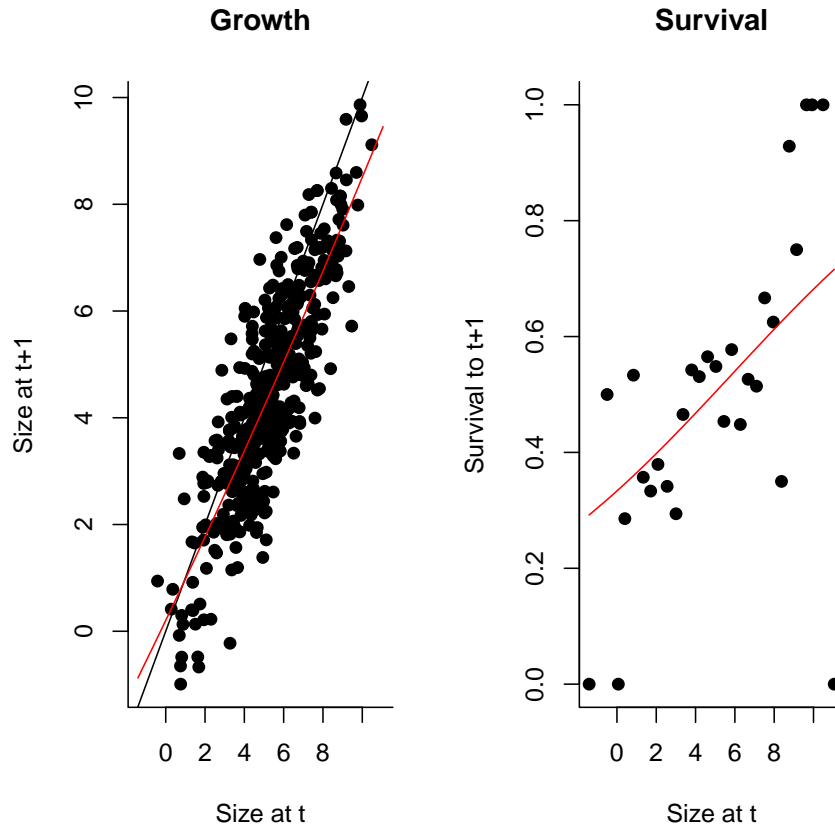
Figure 2: Growth and survival objects

Note that before building growth or survival objects in IPMpack, careful model assessement and comparison are recommended, using all the usual regression tools available in R (plotting the fitted `lm` or `glm` to check for patterns of residuals, outliers etc). IPMpack also contains two functions that allow the user to check these two relationships against the data used for them in order to explore goodness of fit and effect of mesh size, shown in Figure 2 (p. 8).

```
> par(mfrow = c(1, 2), bty = "l", pty = "m")
> p1 <- picGrow(dff, gr1)
> p2 <- picSurv(dff, sv1, ncuts = 30)
```

To build a demographic model describing survival and growth transitions from these objects, the user can use the function `makeIPMPmatrix`, i.e.:

```
> Pmatrix <- makeIPMPmatrix(nBigMatrix = 50,
                            minSize = -5, maxSize = 35,
                            growObj = gr1, survObj = sv1,
                            correction = "constant")
```

where `nBigMatrix` is the numbers of bins used, `minSize` and `maxSize` define the limits of the IPM, L and U in the equations above. Typically, these ranges should usually extend to beyond the smallest and largest size measurement, but the user might want to exclude outliers). The objects `growObj` and `survObj` define changes in size and survival as defined above.

7

```
[1] "Range of Pmatrix is "
[1] 1.480868e-298  3.572901e-01
[1] "Please hit any key for the next plot"
```
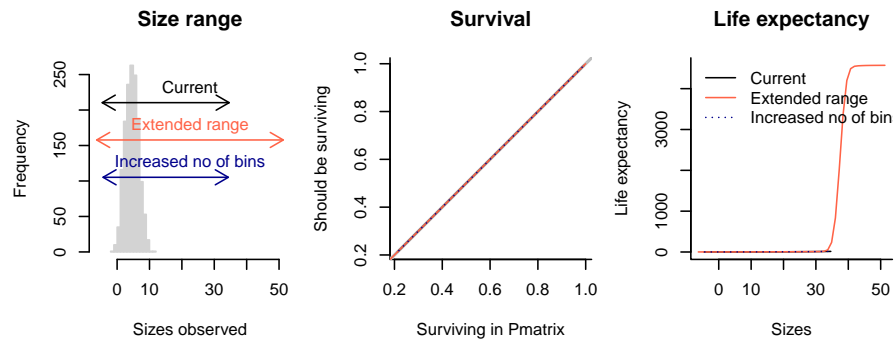

Figure 3: Diagnostic plots for the Pmatrix.

IPMpack includes a useful function `diagnosticsPmatrix` that provides a series of plots indicative of whether bin choice and size range is adequate. Applying this function as a preliminary step before obtaining demographic and evolutionary output from IPMs can help identify basic problems in the fitting of vital rate functions or in the creation of the IPM matrices before proceeding. Several common problems can be diagnosed with the panelled figure produced by `diagnosticsPmatrix` (Figure 3).

The left panel on the first plot shows the range of the data (if the data is provided) and the range of the state variable fitted in the matrix (top line, black). If these are mis-matched, the limits of the data used in building the Pmatrix can be adjusted with the `minSize` and `maxSize` arguments in `makeIPMPmatrix`. The range for two additional Pmatrices that will be used in subsequent comparisons is also provided; one with an increased size range (red) and one with an increased number of bins (blue).

The discretization of a continuous function can result in under- or over-estimation of the true density. Where this occurs, the sum of the columns of the discretized matrix will not match predictions from the fitted survival model. The middle panel plots these against each other for the three matrices in the first panel (current, extended range and increased bin number) using the same colours as in the first panel. Lines should fall along the (0,1) line shown in grey; if they do not, the argument `correction="constant"` in `makeIPMPmatrix` may be of use. This ensures that the columns sum to the fitted survival by multiplying every column in the Integral Projection Model by the value that allows this. The third panel checks whether extending the size range included in the matrix and increasing the number of bins (by increasing `nBigMatrix` and thereby having narrower bins) does not alter basic predictions from the IPM.

The six panels on the next plot (that may not be present in the pdf of the vignette, but should appear if you run the function) show the discretized IPM (histograms) for the current IPM (top) and one with an increased number of bins (bottom) and the theoretical density function (red line). These are plotted either for three chosen sizes (`sizesToPlot`) or the 0.25, 0.5 and 0.75 quantiles of either the observed data (if supplied) or the range of meshpoints (if not); this size is printed in the top right hand of every plot. If the theoretical density function curve is very distant from the histograms, increasing the `nBigMatrix` argument may correct this discrepancy.

Another useful function for verifying that sufficient bins have been used is `convergeIPM`; see the help file and try it out.

The `makeIPMPmatrix` function builds around methods defined so that it will provide appropriate output whatever the survival and growth objects are (e.g. error structure, covariates). The P matrix contains a matrix defining the transitions, but also other useful slots, e.g., the meshpoints, covariates, grid size. The user can access this information by writing:

```
> slotNames(Pmatrix)

[1] ".Data"        "nDiscrete"      "nEnvClass"      "nBigMatrix"
[5] "meshpoints"    "env.index"      "names.discrete"
```

and obtain the slots by using the @ symbol, e.g.,

```
> Pmatrix@meshpoints

 [1] -4.6 -3.8 -3.0 -2.2 -1.4 -0.6  0.2  1.0  1.8  2.6  3.4  4.2  5.0  5.8  6.6
[16]  7.4  8.2  9.0  9.8 10.6 11.4 12.2 13.0 13.8 14.6 15.4 16.2 17.0 17.8 18.6
[31] 19.4 20.2 21.0 21.8 22.6 23.4 24.2 25.0 25.8 26.6 27.4 28.2 29.0 29.8 30.6
[46] 31.4 32.2 33.0 33.8 34.6
```

Finally, the user can plot the Pmatrix using `persp` or `image` (Figure 4), or using `image.plot` of the 'fields' package. Next, with this, the user can obtain the life expectancy, and passage time to a chosen size (here set at the mean) for the range of meshpoints

```
> LE <- meanLifeExpect(Pmatrix)
> pTime <- passageTime(mean(dff$size, na.rm = TRUE), Pmatrix)
```
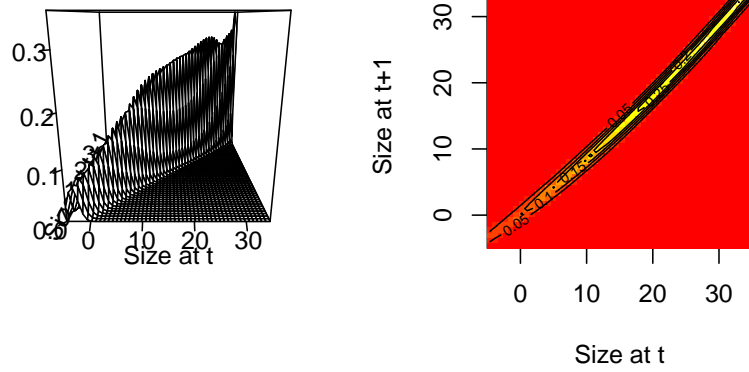
Figure 4: Transition matrix encompassing survival and growth transitions only
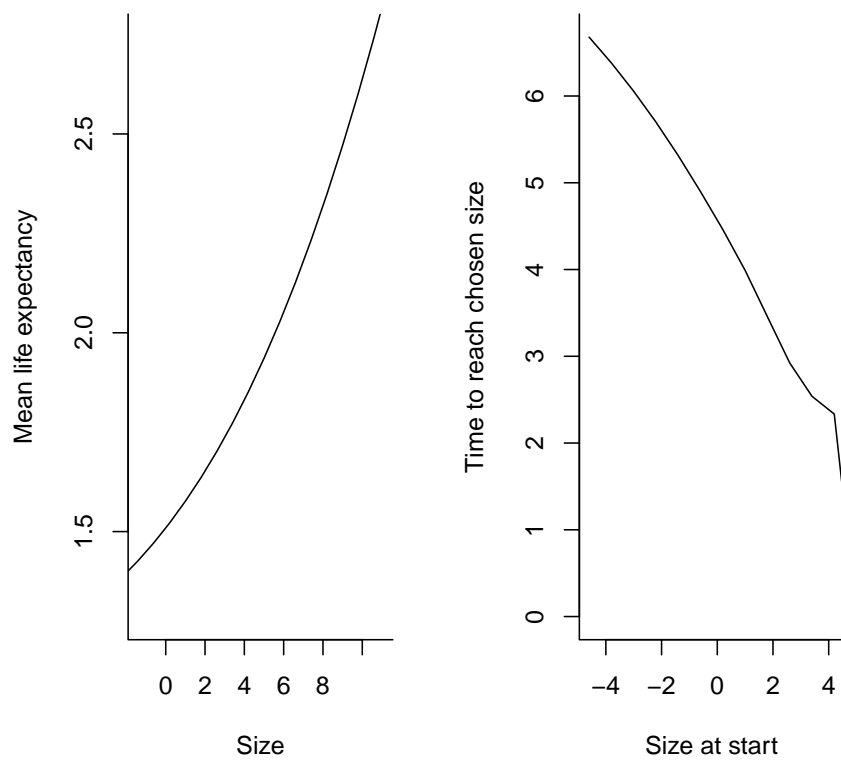
Figure 5: Associated Life Expectancy and Passage Time

11

and the user can also plot these against `Pmatrix@meshpoints` to examine how life expectancy and passage vary as a function of size (Figure 5 p. 12).

If the user defines a fertility object - which is not always easy with for example trees - IPMpack can also create a transition matrix describing movement between sizes attributable to fertility.

```
> fv1 <- makeFecObj(dff, Formula = fec~size,
                     Family = "gaussian",
                     Transform = "log")
> Fmatrix <- makeIPMFmatrix(nBigMatrix = 50, minSize = -5,
                            maxSize = 35,
                            fecObj = fv1,
                            correction = "constant")
```

Note that `makeFecObj` will use the relationships defined in `Formula` and the family defined in `Family` with transforms defined in `Transform` in the order supplied. Please note that the order of elements in fecundity `Formula` MUST match the `Transform` since IPMpack needs to combine the parameters from `Formula` with the right `Transform` to appropriately build the F matrixs.

The default arguments required to run `makeFecObj` to create a fecundity object from which an F matrix with no discrete stage can be built are `offspringSplitter=data.frame(continuous=1)`, `vitalRatesPerOffspringType=data.frame(NA)` and `fecByDiscrete=data.frame(NA)`. Additionally, note that if there are values other than "`continuous`" in the `stage` column of the data-frame named `dff` in the example above, then the function will assume that multiple offspring classes are required, and the result will be an IPM with nBigMatrix + the number of offspring classes deduced (which is the number of names in `stage` other than "`continuous`"). This may lead to a mismatch with the size of the P matrix unless a discrete transition matrix is explicitly being included in the P matrix (see below, incorporating discrete stages).

If the data-frame contains an extra column `offspringNext` that takes the values `sexual`, and that corresponds to rows where both `size` and `sizeNext` are different from NA, the user can define a relationship between maternal size and offspring size through the `makeFecObj` argument `offspringSizeExplanatoryVariables`. The default is to only fit an intercept, equivalent to simply having a mean and variance of offspring size. The function `makeFecObj` also allows users to simply over-write the mean and variance of offspring size with the values of their choice (arguments `meanOffspringSize` and `sdOffspringSize`).

The function `makeClonalObj` operates identically to `makeFecObj` except that offspring are only considered for fitting the distribution of mean and standard deviation of offspring size if the column `offspringNext` takes the values `clonal`. Rows where `offspringNext` takes the values "sexual" or "clonal" are excluded from the survival and growth analyses (`makeGrowthObj` and `makeSurvObj`).

The user can combine the F matrix with (an identically built, i.e., same bin number, size limits and discrete classes) survival-growth transition P matrix to obtain a full Integral Projection Model, and its population growth rate λ, sensitivity, elasticity, etc.

```
> IPM <- Pmatrix + Fmatrix
> Re(eigen(IPM)$value[1])

[1] 3.125397

> sensitivity <- sens(IPM)
> elasticity <- elas(IPM)
```

These outputs can be plotted against the meshpoints (Figure 6 p. 14). In addition to perturbation measures from mesh cells, the user can also obtain sensitivity and elasticity of particular parameters that underlie the kernels, e.g., doing:

```
> res <- sensParams(growObj = gr1, survObj = sv1, fecObj = fv1,
                    nBigMatrix = 50, minSize = -5, maxSize = 15)
> res
```

Sensitivity

Elasticity

Figure 6: Measures off a full IPM
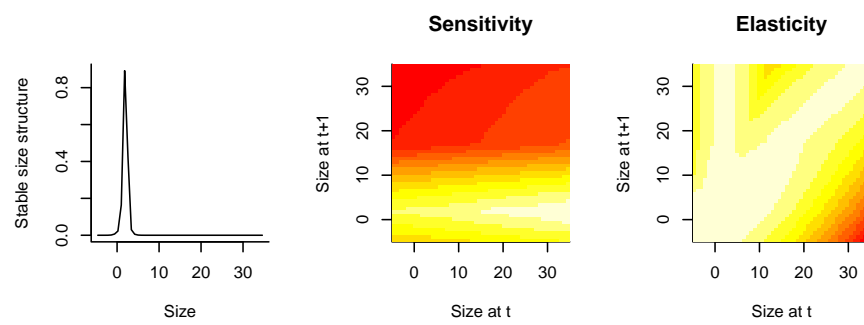
```
$sens
             survival: (Intercept)                           survival: size
                       0.23362673                             0.47395071
                   survival: size2                       growth: (Intercept)
                       1.04734059                             0.10336367
                     growth: size                           growth: size2
                       0.21185748                             0.47254291
                       growth: sd         fecundity: func 1 (Intercept)
                       0.02867356                             2.73815431
          fecundity: func 1 size  fecundity: offspring rel  (Intercept)
                       5.57718191                             0.72714319
       fecundity: sd offspring size
                       0.09345940


$elas
             survival: (Intercept)                           survival: size
                      -0.0517424079                            0.0208574192
                   survival: size2                       growth: (Intercept)
                       0.0002620001                            0.0067183343
                     growth: size                           growth: size2
                       0.0521895532                            0.0009052531
                       growth: sd         fecundity: func 1 (Intercept)
                       0.0095709980                            0.4248049276
          fecundity: func 1 size  fecundity: offspring rel  (Intercept)
                       0.4659667482                            0.4575108981
       fecundity: sd offspring size
                       0.0144525510
```

and this output can be plotted out (Figure 7 p. 16) using

```
> par(mfrow = c(2, 1), bty = "l", pty = "m")
> barplot(res$sens, main = expression("Parameter sensitivity of "*lambda),
                  las = 2, cex.names = 0.5)
> barplot(res$elas, main = expression("Parameter elasticity of "*lambda),
                  las = 2, cex.names = 0.5)
```

4   Incorporating discrete stages

Populations are often structured by both discrete and a continuous stages. For example, many plant populations may persist for many years in a seedbank as well as having size-determined fates after they germinate. IPMpack can incorporate this variability for complex life cycles (Ellner & Rees 2006). To illustrate using discrete stages in an IPM, we generate data that includes both discrete and continuous life-history stages:

```
> dff <- generateData(type="discrete")
```

A quick check indicates that these data contain several types of stage classification (and not just "continuous" as seen up till now):

```
> table(dff$stage)
```

```
continuous    dormant    seedAge1    seedOld
       950         50          35         32
```

Parameter sensitivity of λ
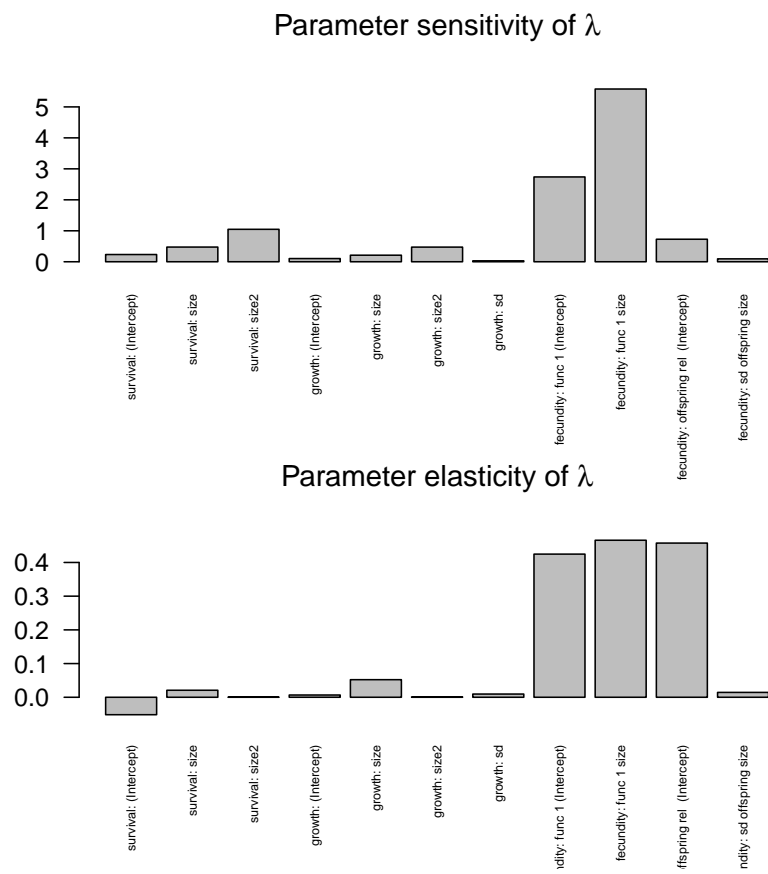

Parameter elasticity of λ

Figure 7: Sensitivity and elasticity of parameter values

Given this data structure, the user can make a fertility object that reflects the fact that propagules (seeds in this example) produced in one year may recruit directly into the continuous phase (e.g., seedling), or may end up in a discrete stage (e.g., seed bank). The `makeFecObj` (and similar functions) has an argument called `offspringSplitter` that allows the user to define these paths:

```
> fv1 <- makeFecObj(dataf = dff, Transform = "log",
                    offspringSplitter = data.frame(continuous = 0.2,
                    dormant = 0, seedAge1 = 0.8, seedOld = 0),
                    fecByDiscrete = data.frame(dormant = 0,
                    seedAge1 = 0, seedOld = 0))
```

In this example, 20% of seeds produced at t end up in the continuous part of the population structure at $t+1$ (for example, they might directly recruit as rosettes from one year to the next) and 80% of seeds recruit into the "one year old seeds" stage. Although in this case no individuals are recruited at $t+1$ into the "dormant" or "old seeds" stages (since these will come from adult plants or the seed bank), they are included because `offspringSplitter` is where IPMpack identifies all the existing discrete stages. The argument `fecByDiscrete` reflects the fact that none of the discrete classes addressed in this example are likely to directly produce offspring (which may not always be the case). The resulting fecundity object can be used with `makeIPMFmatrix` in the usual way:

```
> Fmatrix <- makeIPMFmatrix(fecObj = fv1, nBigMatrix = 5,
                            minSize = min(dff$size, na.rm = TRUE),
                            maxSize = max(dff$size, na.rm = TRUE),
                            correction = "constant")
```

The user also needs a `Pmatrix` that reflects the same structure. The continuous part of the P matrix will be the standard structure:

```
> gr1 <- makeGrowthObj(dataf = dff,
                       Formula = sizeNext~size)
> sv1 <- makeSurvObj(dff,  Formula = surv~size)
```

Movement in and out of discrete stages is defined via an add-on of a transition matrix, that is defined using:

```
> discTrans <- makeDiscreteTrans(dff)
```

which captures survival and transitions between discrete stages and the continuous stage (note that this function will not work unless the data frame `dff` contains appropriate columns `stage` and `stageNext`). The user can then construct the P matrix:

```
> Pmatrix <- makeIPMPmatrix(nBigMatrix = 5,
                            minSize = min(dff$size, na.rm = TRUE),
                            maxSize = max(dff$size, na.rm = TRUE),
                            growObj = makeGrowthObj(dff),
                            survObj = makeSurvObj(dff),
                            discreteTrans = discTrans,
                            correction = "constant")
```

Note that both the P matrix and the F matrix in this example have a rather small number of bins just for ease of comparison, and that a higher number is almost certainly advisable. The user can examine both matrices:

```
> print(Pmatrix)

An object of class "IPMmatrix"
            [,1]            [,2]            [,3]            [,4]            [,5]
```

```
[1,] 0.360000000 0.0000000000 0.0000000000 6.594480e-04 2.772003e-03
[2,] 0.000000000 0.0000000000 0.0000000000 0.000000e+00 0.000000e+00
[3,] 0.000000000 0.4439560440 0.4323308271 0.000000e+00 0.000000e+00
[4,] 0.000127675 0.0047979721 0.0036987569 1.487670e-01 7.599387e-04
[5,] 0.013308240 0.0297160090 0.0413307826 1.705068e-01 2.940840e-01
[6,] 0.186465715 0.0318915653 0.0575506428 1.157318e-04 6.739691e-02
[7,] 0.351189135 0.0059308069 0.0099858593 4.651996e-11 9.147115e-06
[8,] 0.088909234 0.0001911192 0.0002159133 1.107393e-20 7.351974e-13
                [,6]          [,7]          [,8]
[1,] 1.241362e-02 5.479516e-02 2.031374e-01
[2,] 0.000000e+00 0.000000e+00 0.000000e+00
[3,] 0.000000e+00 0.000000e+00 0.000000e+00
[4,] 4.089008e-08 2.722281e-14 1.855485e-22
[5,] 5.342794e-03 1.200995e-06 2.763904e-12
[6,] 4.134229e-01 3.137796e-02 2.438171e-05
[7,] 1.894507e-02 4.854942e-01 1.273741e-01
[8,] 5.141309e-07 4.448556e-03 3.940703e-01
Slot "nDiscrete":
[1] 3

Slot "nEnvClass":
[1] 1

Slot "nBigMatrix":
[1] 5

Slot "meshpoints":
[1] -1.218588  1.435891  4.090370  6.744849  9.399327

Slot "env.index":
[1] 1 1 1 1 1

Slot "names.discrete":
[1] "dormant"  "seedAge1" "seedOld"

> print(Fmatrix)

An object of class "IPMmatrix"
     [,1] [,2] [,3]         [,4]          [,5]          [,6]         [,7]
[1,]    0    0    0 0.000000000 0.0000000000 0.0000000000 0.000000000
[2,]    0    0    0 0.109208863 0.1991010039 0.3629852788 0.661766189
[3,]    0    0    0 0.000000000 0.0000000000 0.0000000000 0.000000000
[4,]    0    0    0 0.001364397 0.0024874619 0.0045349448 0.008267754
[5,]    0    0    0 0.008832214 0.0161021974 0.0293562589 0.053520021
[6,]    0    0    0 0.012729269 0.0232070017 0.0423091789 0.077134765
[7,]    0    0    0 0.004084534 0.0074466013 0.0135760574 0.024750799
[8,]    0    0    0 0.000291801 0.0005319886 0.0009698798 0.001768209
           [,8]
[1,] 0.000000000
[2,] 1.206480028
[3,] 0.000000000
[4,] 0.015073119
[5,] 0.097573489
[6,] 0.140626032
```

```
[7,] 0.045123709
[8,] 0.003223659
Slot "nDiscrete":
[1] 3

Slot "nEnvClass":
[1] 1

Slot "nBigMatrix":
[1] 5

Slot "meshpoints":
[1] -1.218588  1.435891  4.090370  6.744849  9.399327

Slot "env.index":
[1] 1 1 1 1 1

Slot "names.discrete":
[1] "dormant"  "seedAge1" "seedOld"
```

and check for example that the slot `namesDiscrete` is aligned for both of them, and add them together:

```
> print(Pmatrix+Fmatrix)

           [,1]           [,2]           [,3]          [,4]          [,5]
[1,] 0.360000000 0.0000000000 0.0000000000 0.000659448 0.0027720032
[2,] 0.000000000 0.0000000000 0.0000000000 0.109208863 0.1991010039
[3,] 0.000000000 0.4439560440 0.4323308271 0.000000000 0.0000000000
[4,] 0.000127675 0.0047979721 0.0036987569 0.150131363 0.0032474006
[5,] 0.013308240 0.0297160090 0.0413307826 0.179339059 0.3101862201
[6,] 0.186465715 0.0318915653 0.0575506428 0.012845001 0.0906039081
[7,] 0.351189135 0.0059308069 0.0099858593 0.004084534 0.0074557484
[8,] 0.088909234 0.0001911192 0.0002159133 0.000291801 0.0005319886
            [,6]         [,7]        [,8]
[1,] 0.0124136226 0.054795158 0.20313740
[2,] 0.3629852788 0.661766189 1.20648003
[3,] 0.0000000000 0.000000000 0.00000000
[4,] 0.0045349856 0.008267754 0.01507312
[5,] 0.0346990528 0.053521222 0.09757349
[6,] 0.4557320828 0.108512727 0.14065041
[7,] 0.0325211304 0.510245016 0.17249781
[8,] 0.0009703939 0.006216764 0.39729392
```

The first three rows and columns concern transitions in and out of the discrete stages; the remainder are the usual P and F matrices describing moving across the continuous variables. The usual types of calculations (sensitivity via `sens`, life expectancy via `meanLifeExpect`, etc) can be applied here too.

5   Discretely varying environments

One extension of IPMs is to create a compound IPM matrix where, in addition to moving between continuous sizes, individuals move through discrete environments where the discrete environmental states have an expected sequence, and therefore can be described by a transition matrix of their own (e.g. light environments for tropical trees, as in Metcalf et al. 2009).

To explore this type of dynamic, the user needs either to provide or simulate an environmental variable at t and the corresponding value at $t+1$. Here, it has been generated as part of the `generateData` function (See above). From this generated data, the user can then create an environmental transition matrix that describes how the environment tends to move between these states from one census time to the next. If the data has been set up as described, there is a function that will do this for the user:

```
> dff <- generateData()
> env1 <- makeEnvObj(dff)
> env1

An object of class "envMatrix"
          [,1]      [,2]
[1,] 0.2111399 0.2061404
[2,] 0.7888601 0.7938596
Slot "nEnvClass":
[1] 2
```

The user can now use IPMpack to create a survival-growth transition P matrix that encompasses movement across environments, first redefining the survival and growth objects to fit a discrete covariate, by changing the `Formula` argument:

```
> gr1 <- makeGrowthObj(dff, Formula = sizeNext~size+covariate)
> sv1 <- makeSurvObj(dff, Formula = surv~size+covariate)
```

Note that these functions will only work appropriately for a discrete covariate if the value of the covariate at time t is available as a column in the data frame names `covariate` and the value of the covariate at the next census is available as a column in the data frame called `covariateNext`. IPMpack functions use the presence of a column in the data frame called `covariate` as a cue to renumber values in these two columns to numeric levels between 1 and the observed number of covariate levels to facilitate looping, and changes them into factors. Once this step is implemented, these functions can be used to create a compound P matrix, using `makeCompoundPmatrix`:

```
> Pmatrix <- makeCompoundPmatrix(nBigMatrix = 50, minSize = -5,
                                  maxSize = 35,
                                  envMatrix = env1, growObj = gr1,
                                  survObj = sv1,
                                  correction = "constant")
```

Essentially, the compound P matrix is a large matrix with stacked IPMs corresponding to each environment, modified to reflect movement between environmental states defined by env1. Passage time can be calculated using a similar function, but now including the environmental matrix as an argument (equivalent life expectancy functions are in development):

```
> pTimes <- stochPassageTime(Pmatrix@meshpoints[15], Pmatrix, env1)
```

The resulting vectors contain the life expectancy and time to reach each size for individuals starting in each different environmental class, concatenated together (i.e. there are `nBigMatrix` values in the LE matrix ranging over the first environment, then `nBigMatrix` values ranging over the second environment, etc). The user can plot these against meshpoints (Figure 11 p. 26), each colour indicates a different starting environment. Similar syntax can be used for passage time (although note that here the function name has changed):

Adding a fertility object to this analysis, the user can also define a full life cycle IPM model for across environments. With such information, obtaining the stochastic population rate of increase λ_s in this environment is relatively straight-forward. IPMpack does this by sampling a very large number of environments and corresponding IPMs, and multiplying them together (Childs et al. 2004). At the moment, this is only defined for the case where environments (defined by the
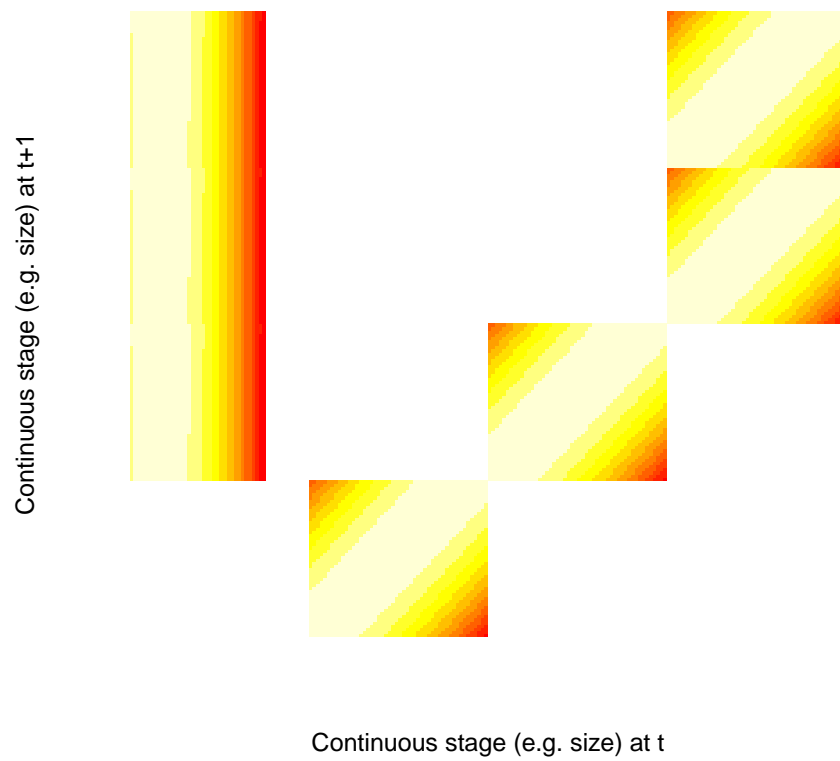
Figure 8: Compound P matrix IPM shown on a log scale: whiter colours indicate more likely transitions; and the four different panels indicate each of the four environments.
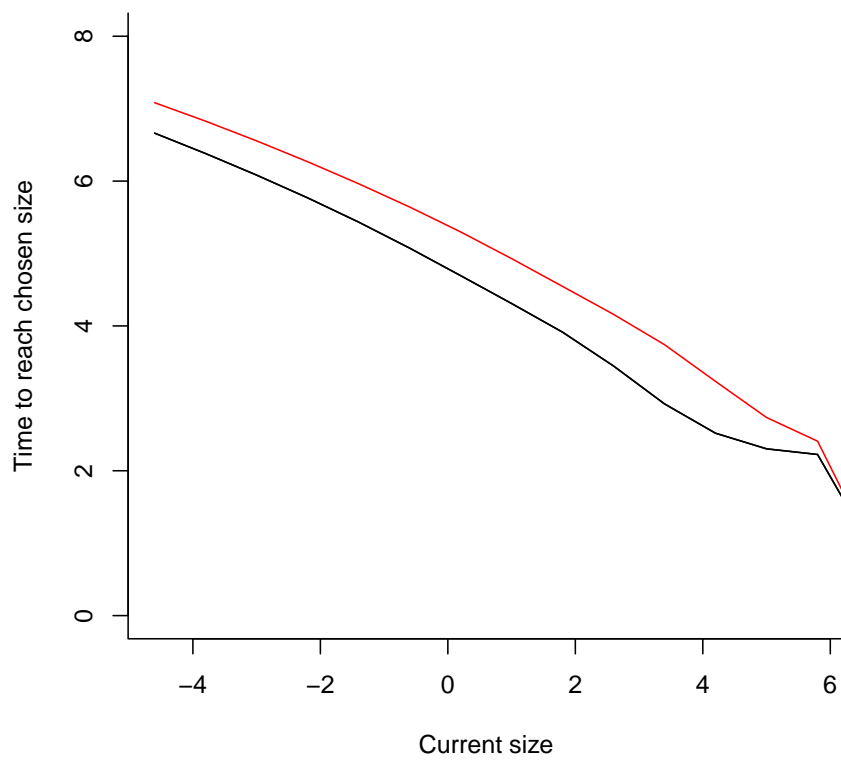
Figure 9: Passage time for a compound IPM; different colours reflect predictions for individuals starting in different environments

discrete covariates) are distributed independently (i.e. the next state does not depend on the previous state). To do this, the user must first define a list of IPMs (each the sum of a matrix of survival-growth transitions, and a matrix of fecundity transitions corresponding to a particular environment):

```
> IPMlist <- sampleSequentialIPMs(dataf = dff, nBigMatrix = 25, minSize = -5,
                        maxSize = 35, explSurv = surv~size+covariate,
                        explGrow = sizeNext~size+size2+covariate,
                        explFec = fec~size, Transform="log",correction = "constant")
```

```
surv ~ size + covariate
```

Note that in this example IPMpack uses an arbitrary selection of formulas for the various linear and logistic regressions (`explGrow`, `explSurv`, etc). In reality, model selection will need to be used to establish this. Importantly, each environment type requires sufficient data (years of data) to quantify population behaviours in that environment as if it were an IPM in a single environment. Next, the user can estimate λ_s using:

```
> stochGrowthRateSampleList(listIPMmatrix = IPMlist,
                        nRunIn = 30, tMax = 50)
```

```
[1] 0.9650726
```

where `nRunIn` defines the number of time steps to discard from the start of the time series in order to remove transient dynamics, and `tMax` is the total number of time steps to run, and should be large enough that increasing it does not substantially change the result (numbers presented here for efficiency are almost certaintly not large enough). Note that `stoch.growth.rate` in the package `popbio` will operate in essentially the same way, and provides more information, more efficiently.

6   Kernels describing transitions dependent on size and age

We can use the same structures as used for environmental transitions to build models that encompass transitions both over size and between age classes. For illustration, we need to generate a data-set with an age covariate - since IPMpack doesn't have a function to generate one at the moment. We are going to call the age variable "covariate".

```
> dff <- generateData()
> dff$covariate <- sample(c(1:4),nrow(dff),replace=TRUE)
```

Note that there is going to be no particular relationship between size and age since we've made it up in a rather random way.... - but this will serve for the purpose of illustration. Next, we create growth and survival objects that incorporate age using:

```
> gr1 <- makeGrowthObj(dataf = dff,
                    Formula=sizeNext~size+covariate,
                    regType="constantVar",
                    Family="gaussian")
> sv1 <- makeSurvObj(dff, Formula = surv~size+covariate)
```

We now also need to create a matrix that will move individuals between the different age classes:

```
> n.age.classes <- max(dff$covariate,na.rm=TRUE)
> ageMat <- new("envMatrix", nEnvClass = n.age.classes)
> ageMat@.Data <- matrix(0,n.age.classes,n.age.classes)
> ageMat@.Data[cbind(2:n.age.classes,1:(n.age.classes-1))] <- 1
> ageMat@.Data[n.age.classes,n.age.classes] <- 1
```

This matrix is set to be of the same class as above (`envMatrix`), so as it will move individuals between age classes appropriately. It can be shown as follows:

```
> print(ageMat)

An object of class "envMatrix"
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    1    0    0    0
[3,]    0    1    0    0
[4,]    0    0    1    1
Slot "nEnvClass":
[1] 4
```

To build a demographic model describing growth and survival transitions, and encompassing movement between different age classes from these objects, we can use the function `makeCompoundPmatrix`, i.e.:

```
> Pmatrix <- makeCompoundPmatrix(nBigMatrix = 50, minSize = -5,
                                 maxSize = 35,
                                 envMatrix = ageMat,
                                                     nEnvClass = 4,
                                 growObj = gr1,
                                 survObj = sv1,
                                 correction = "constant")
```

where `nBigMatrix.matrix` is the number of bins used, `minSize` and `maxSize` define the limits of the IPM (which should usually extend to beyond the smallest and largest size measurements) and the growth and survival objects are as above. We can plot this Pmatrix using the below:

To run the following code, the package "sparseMatrix" is required. Load that library, and then this code can be run.

And we can also make a fecundity object

```
> fv1 <- makeFecObj(dff, Formula = fec~size,
                    Family = "gaussian",
                    Transform = "log")
```

then create a new age matrix to place the young into the first age class,

```
> n.age.classes <- max(dff$covariate,na.rm=TRUE)
> ageMat1 <- new("envMatrix", nEnvClass = n.age.classes)
> ageMat1@.Data <- matrix(0,n.age.classes,n.age.classes)
> ageMat1@.Data[1,2:n.age.classes] <- 1
```

and then build the age/size fertility matrix -

```
> Fmatrix <- makeCompoundFmatrix(nBigMatrix = 50, minSize = -5,
                                 maxSize = 35,
                                 envMatrix = ageMat1,
                                                     nEnvClass = 4,
                                 fecObj = fv1,
                                 correction = "constant")
```

and then plot the full matrix

```
> image(Pmatrix[,],
          xlab = "Continuous stage (e.g. size) at t",
                  ylab = "Continuous stage (e.g. size) at t+1", axes = FALSE)
```
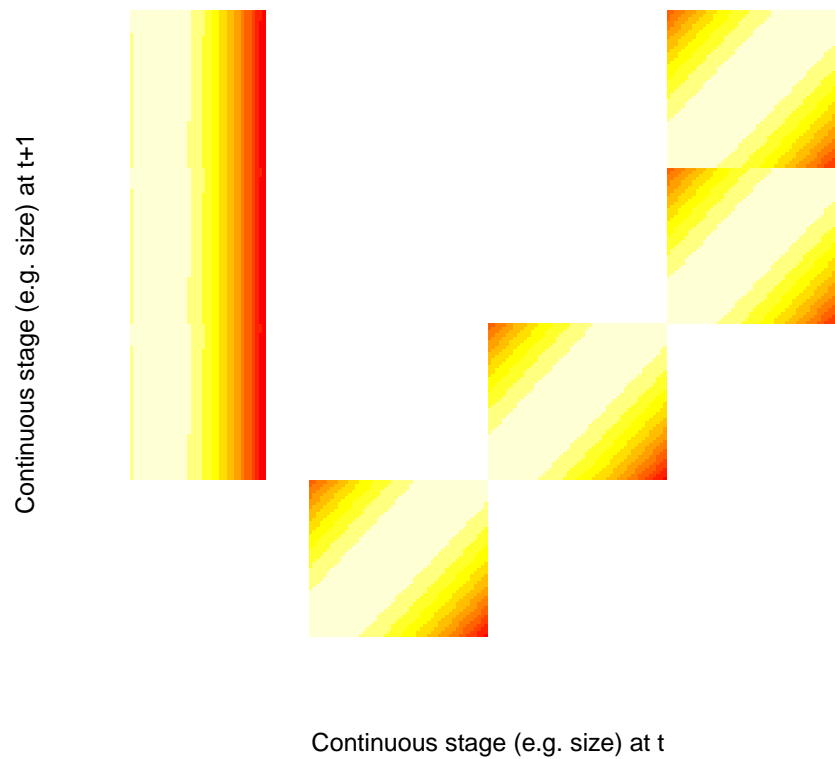

Figure 10: Compound P matrix IPM shown on a log scale: darker colours indicate more likely transitions; and the four different panels indicate each of the four ages.

```
> IPM <- Pmatrix+Fmatrix
> image(log(IPM),
        xlab = "Continuous stage (e.g. size) at t",
            ylab = "Continuous stage (e.g. size) at t+1", axes = FALSE)
```
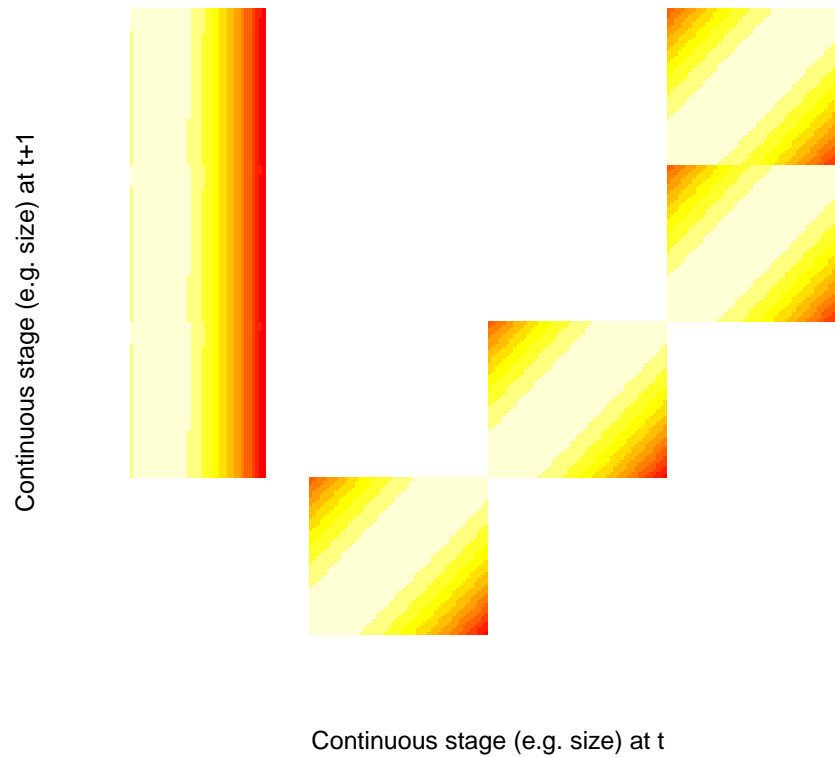

Figure 11: Compound IPM shown on a log scale: light colours indicate more likely transitions; and the four different panels indicate each of the four ages.

7   More generally varying environments

An alternative way of inhabiting stochastic environments is to experience continuously changing covariates (rather than moving between discrete states, as the above describes). In this case, rather than building a single megamatrix, desired variables are obtained by multiplying up a suite of matrices and relying on the weak ergodic theorem for convergence (e.g., Tuljapurkar 1990; as described for obtaining λ_s, above). IPMpack contains code to do this. The user must first define a new data frame containing several time-varying covariates, and then build the associated survival, growth and fertility objects (note that the default in IPMpack is to assume that if you have a covariate called "coviariate", your aim is to build a compound matrix, so ideally other covariates should have other names):

```
> dff <- generateData(type="stochastic")
> sv1 <- makeSurvObj(dataf = dff,
                           Formula = surv~size+covariate1+covariate3)
> gr1 <- makeGrowthObj(dataf = dff,
                            Formula = sizeNext~size+covariate1+covariate2)
> fv1 <- makeFecObj(dataf = dff, fecConstants = data.frame(1.8),
                    Formula = fec~size, Transform = "log")
```

As before, the user can explore the data:

```
> head(dff)

      size  sizeNext surv covariate1 covariate2 covariate3       fec      stage
1 5.865137 12.648636    1  2.0251669  0.4306256  1.5575108  0.000000 continuous
2 4.398762  2.411977    1 -0.8729842  0.4739232 -0.8457222  0.000000 continuous
3 6.744685        NA    0  1.1816678  1.8026082  0.5533295  0.000000 continuous
4 4.421893  7.748363    1  0.8621840  0.5152927  1.3946205  5.039011 continuous
5 4.197129        NA    0  1.7154103  1.6193036  0.8610345  9.815698 continuous
6 4.660496        NA    0  1.4131384 -1.7130040 -0.8585491 22.582961 continuous
   stageNext number
1 continuous      1
2 continuous      1
3 continuous      1
4 continuous      1
5 continuous      1
6 continuous      1
```

and glance at the objects, e.g.,

```
> gr1

An object of class "growthObj"
Slot "fit":

Call:
lm(formula = Formula, data = dataf)

Coefficients:
(Intercept)         size    covariate1    covariate2
    1.00671      0.89856       3.02825       0.00298


Slot "sd":
[1] 0.2253727
```

To explore demographic projections for this model, the user must decide on a time scale and time length, and define it by a vector called 'tVals'. This time series can be used to reflect patterns in environmental data that repeat. In this example we set `tVals` to reflect monthly intervals over 4 years, with years as the time scale and build covariates that vary seasonally, i.e., they fluctuates randomly around a sine wave which peaks once a year. From this simulation, the user can generate a matrix containing time as rows and different covariates in columns.

```
> tVals <- seq(1, 4, by = 1/12)
> covTest <- c(1 + 0.5*sin(2*pi*tVals))
> covMatTest <- data.frame(covariate1 = rnorm(length(covTest), covTest, 0.5) - 1,
                           covariate2 = rnorm(length(covTest), covTest, 0.5) - 1,
                           covariate3 = rnorm(length(covTest), covTest, 0.5) - 1)
```

Note that if there is no apparent temporal pattern to the data, one could simply generate random normal distributions of the covariates using their observed mean and variance. Other types of temporal patterns (multiannual, etc) are also possible. With this setup, the user can then estimate the stochastic growth rate over these years, using the geometric mean of the population growth rate (Tuljapurkar 1990; Childs et al. 2004), for these particular covariates using:

```
> r <- stochGrowthRateManyCov(covariate = covMatTest, nRunIn = 12*1,
                              tMax = length(tVals), growthObj = gr1,
                              survObj = sv1, fecObj = fv1, nBigMatrix = 20,
                              minSize = 2*min(dff$size, na.rm = TRUE),
                              maxSize = 1.6*max(dff$size, na.rm = TRUE),
                              nMicrosites = 50, correction = "constant")
> print(r)

$Rt
[1] 0.02048429
```

Setting nRunIn $= 12 * 1$ in this example is equivalent to discarding the first year of the simulation (likely to contain transients) since the chosen time step is months. Note that in this formula, it was assumed that density-dependence acts on seedling establishment, and that 50 microsites are available for seedling establishment in every time step. Setting nMicrosites $= 0$ allows for calculations without density-dependence, and nMicrosites can also be a vector, if the number of microsites fluctuates through time. It may also be interesting to have a glance at what has been happening to the population structure over this time-course, and the arguments `trackStruct` and `plot` allows this.

8 Building your own objects and methods

What if growth is best reflected by a saturating function, rather than by the linear regression models provided? In this case the user may want to define a new class of growth object. An example of this follows:

```
> setClass("growthObjSaturate", representation(paras = "numeric", sd = "numeric"))
```

Then define the functional form of the mean prediction, with relevant parameters:

```
> fSaturate <- function(size, pars) {
      u <- exp(pmin(pars[1] + pars[2] * size, 50))
      u <- pars[3] * 1/(1+u)
      return(u)
  }
```

where the third parameter indicates the asymptotic size. The user can then estimate the parameters by fitting this function to the data using a wrapper function and the R function `optim`.

```
> wrapSaturate <- function(par, dataf) {
      pred <- fSaturate(dataf$size, par[1:3])
      ss <- sum((pred - dataf$sizeNext)^2, na.rm = TRUE)
      return(ss)
      }
> tmp <- optim(c(1, 1, 1), wrapSaturate, dataf = dff, method = "Nelder-Mead")
> tmp

$par
[1]  2.1741123 -0.7829247  6.7141642

$value
[1] 4456.995

$counts
function gradient
     310       NA

$convergence
[1] 0

$message
NULL
```

For simplicity, one can assume normally distributed errors:

```
> resids <- fSaturate(dff$size, tmp$par) - dff$sizeNext
> sdSaturate <- sd(resids, na.rm = TRUE)
```

With these parameters, the user can then define the new growth object:

```
> gr1 <- new("growthObjSaturate")
> gr1@paras <- tmp$par
> gr1@sd <- sdSaturate
```

Finally, the user must define a method appropriate for this type of object.

```
> setMethod("growth", c("numeric", "numeric", "numeric", "growthObjSaturate"),
          function(size, sizeNext, cov, growthObj){
              mux <- fSaturate(size, growthObj@paras)
              sigmax <- growthObj@sd
              u <- dnorm(sizeNext, mux, sigmax, log = F)
              return(u);
          })
```

```
[1] "growth"
```

By putting `growthObjSaturate` in the signature, R will use this particular method for all objects with this signature. Now, the user can go ahead and use all the other code as previously, without a need for further definitions.

If the user wishes to fit a growth model with, for example, gamma errors, a similar approach can be used, but with 'dgamma' instead of dnorm in the last line of growth method, and appropriate slots defined in the object, etc.

Selected References

- Caswell. 2001. Matrix population models: analysis, construction and interpretation. 2nd ed. Sinauer. Massachussetts, USA.

- Childs, Rees, Rose, Grubb & Ellner. 2004. Evolution of size-dependent flowering in a variable environment: Construction and analysis of a stochastic integral projection model. Proc. Roy. Soc. Lond. Ser. B. 271: 471-475.

- Cochran & Ellner. 1995. Simple methods for calculating age-based life history parameters for stage-structured populations. Ecological Monographs 62: 345-364.

- Ellner & Rees. 2006. Integral projection models for species with complex life-histories. American Naturalist 167: 410-428.

- Metcalf, Horvitz, Tuljapurkar & Clark. 2009. A time to grow and a time to die: a new way to analyze the dynamics of size, light, age and death of tropical trees. Ecology 90: 2766-2778.

- Metcalf, McMahon, Salguero-Gómez & Jongejans. 2013. IPMpack: an R package for Integral Projection Models. Methods in Ecology and Evolution 4: 195-200

- Rees & Rose. 2002. Evolution of flowering strategies in *Oenothera glazioviana*: an integral projection model approach. Proc. Roy. Soc. Lond. Ser. B. 269: 1509-1515.

- Ramula, Rees & Buckley. 2009. Integral projection models perform better for small demographic data sets than matrix population models: a case study of two perennial herbs. Journal of Applied Ecology 46: 1048-1053.

- Salguero-Gomez & Plotkin. 2010. Matrix dimensionality bias demographic inferences: implications for comparative plant demography. American Naturalist 176: 710-722

- Tuljapurkar. 1990. Population Dynamics in Variable Environments. Springer. New York, USA.

- Zuidema, Jongejans, Chien, During & Schieving. 2010. Integral Projection Models for trees: a new parameterization and a validation of model output. Journal of Ecology 98: 345-355.