Case Study 2: Ellner, Childs & Rees 2016

A more complicated, age and size structured model

Many life cycles cannot be described by a single, continuous state variable. For example, some plants are best modeled using height or diameter at breast height (DBH) as a state variable, and may also form a seed bank. Seeds in the seed bank can't have a value for height or DBH, but may lose their viability as they age. Thus, we require a discrete state to capture the dynamics of the seed bank. Models that include discrete states and/or multiple continuous state variables are *general IPMs*.

Many species exhibit age-dependent demography. Age may interact with a measure of size, for example body mass, resulting in neither single variable reliably predicting demography on its own. Data sets for which individuals are cross-classified by age and size represent an interesting opportunity for demographic research.

This case study will use an age and size structured population to illustrate how to implement general IPMs in `ipmr`. We will use an age and size structured model from Ellner, Childs, & Rees (2016) to explore how to work with general IPMs in `ipmr`. The data are from a long term study of Soay sheep (*Ovis aries*) on St. Kilda. The population has been studied in detail since 1985 (Clutton-Brock & Pemberton 2004). Individuals are caught, weighed, and tagged shortly after birth, and re-weighed each subsequent year. Maternity can be inferred from field observations, so we can also model the link between parental state and offspring state. More detailed methods are provided in Clutton-Brock & Pemberton (2004) and Childs et al. 2011.

In addition to computing the per-capita growth rate (λ) and right and left eigenvectors ($w_a(z)$ and $v_a(z)$, respectively), we will also show how to compute age specific survival and fertility for these models.

With size denoted z, z', age denoted a, and the maximum age an individual can have denoted M, the model can be written as:

1. $n_0(z', t+1) = \sum_{a=0}^{M} \int_L^U F_a(z', z) n_a(z, t) dz$
2. $n_a(z', t+1) = \int_L^U P_{a-1}(z', z) n_{a-1}(z, t) dz$ for $a = 1, 2, ..., M$

In this case, there is also an "greybeard" age class $M+1$ defined as $a \geq M+1$. We need to define one more equation to indicate the number of individuals in that age group.

3. $n_{M+1}(z', t+1) = \int_L^U [P_M(z', z) n_M(z, t) + P_{M+1}(z', z) n_{M+1}(z, t)] dz$

Below, f_G and f_B denote normal probability density functions. The sub-kernel $P_a(z', z)$ is comprised of the following functions:

4. $P_a(z', z) = s(z, a) * G(z', z, a)$
5. Survival: $Logit(s(z, a)) = \alpha_s + \beta_{s,z} * z + \beta_{s,a} * a$
6. Growth: $G(z', z, a) = f_G(z', \mu_G(z, a), \sigma_G)$
7. Mean Growth: $\mu_G(z, a) = \alpha_G + \beta_{G,z} * z + \beta_{G,a} * a$

and the sub-kernel $F_a(z', z)$ is comprised of the following functions:

8. $F_0 = 0$
9. $F_a = s(z, a) * p_b(z, a) * p_r(a) * B(z', z) * 0.5$

This model only follows females, and we assume that the population is half female. Thus, we multiply the F_a kernel by 0.5. If needed, we could adjust the sex ratio based on observed data, and update this multiplication accordingly.

10. Probability of reproducing: $Logit(p_b(z,a)) = \alpha_{p_b} + \beta_{p_b,z} * z + \beta_{p_b,a} * a$
11. Probability of recruiting: $Logit(p_r(a)) = \alpha_{p_r} + \beta_{p_r,a} * a$
12. Recruit size distribution: $B(z',z) = f_B(z', \mu_B(z), \sigma_B)$
13. Mean recruit size: $\mu_B(z) = \alpha_B + \beta_{B,z} * z$

Equations 5-7 and 10-13 are parameterized from regression models. The parameter values are taken from Ellner, Childs & Rees, Chapter 6 (2016). These can be found here. In the code from the book, these parameters were used to simulate an individual based model (IBM) to generate a data set. The data were then used to fit regression models and an age×size IPM. We are going to skip the simulation and regression model fitting steps steps and just use the "true" parameter estimates to generate the IPM.

Model Code

First, we will define all the model parameters and a function for the F_a kernels.

```
library(ipmr)

# Set parameter values and names

param_list <- list(
  ## Survival
  surv_int  = -1.70e+1,
  surv_z    =  6.68e+0,
  surv_a    = -3.34e-1,
  ## growth
  grow_int  =  1.27e+0,
  grow_z    =  6.12e-1,
  grow_a    = -7.24e-3,
  grow_sd   =  7.87e-2,
  ## reproduce or not
  repr_int  = -7.88e+0,
  repr_z    =  3.11e+0,
  repr_a    = -7.80e-2,
  ## recruit or not
  recr_int  =  1.11e+0,
  recr_a    =  1.84e-1,
  ## recruit size
  rcsz_int  =  3.62e-1,
  rcsz_z    =  7.09e-1,
  rcsz_sd   =  1.59e-1
)

# define a custom function to handle the F kernels. We could write a rather
# verbose if(age == 0) {0} else {other_math} in the define_kernel(), but that
# might look ugly. Note that we CANNOT use ifelse(), as its output is the same
# same length as its input (in this case, it would return 1 number, not 10000
# numbers).

r_fun <- function(age, s_age, pb_age, pr_age, recr) {
```

```
  if(age == 0) return(0)

  s_age * pb_age * pr_age * recr * 0.5

}
```

Next, we set up the P_a kernels (Equations 4-7 above). Because this is a general, deterministic IPM, we use `init_ipm(sim_gen = "general", di_dd = "di", det_stoch = "det", uses_age = TRUE)`. We set `uses_age = TRUE` to indicate that our model has age structure as well as size structure. There are 3 key things to note:

1. the use of the suffix `_age` appended to the names of the `"P_age"` kernel and the `mu_g_age` variable.
2. the value `age` used in the vital rate expressions.
3. the list in the `age_indices` argument.

The values in the `age_indices` list will automatically get substituted in for `"age"` each time it appears in the vital rate expressions and kernels. We add a second variable to this list, `"max_age"`, to indicate that we have a "greybeard" class. If we wanted our model to kill all individuals above age 20, we would simply omit the `"max_age"` slot in the `age_indices` list.

This single call to `define_kernel()` will result in 22 actual kernels, one for each value of `age` from 0-21. For general IPMs that are not age-structured, we would use `uses_par_sets` and `par_set_indices` in the same way we're using `age` below.

The `plogis` function is part of the `stats` package in *R*, and performs the inverse logit transformation.

```
age_size_ipm <- init_ipm(sim_gen = "general",
                         di_dd = "di",
                         det_stoch = "det",
                         uses_age = TRUE) %>%
  define_kernel(
    name          = "P_age",
    family        = "CC",
    formula       = s_age * g_age * d_z,
    s_age         = plogis(surv_int + surv_z * z_1 + surv_a * age),
    g_age         = dnorm(z_2, mu_g_age, grow_sd),
    mu_g_age      = grow_int + grow_z * z_1 + grow_a * age,
    data_list     = param_list,
    states        = list(c("z")),
    uses_par_sets = FALSE,
    age_indices   = list(age = c(0:20), max_age = 21),
    evict_cor     = FALSE
  )
```

The F_a kernel (equations 8-13) will follow a similar pattern - we append a suffix to the `name` paramter, and then make sure that our functions also include `_age` suffixes and `age` values where they need to appear.

```
age_size_ipm <-   define_kernel(
  proto_ipm     = age_size_ipm,
  name          = "F_age",
  family        = "CC",
  formula       = r_fun(age, s_age, pb_age, pr_age, recr) * d_z,
  s_age         = plogis(surv_int + surv_z * z_1 + surv_a * age),
  pb_age        = plogis(repr_int + repr_z * z_1 + repr_a * age),
  pr_age        = plogis(recr_int + recr_a * age),
  recr          = dnorm(z_2, rcsz_mu, rcsz_sd),
```

```
  rcsz_mu       = rcsz_int + rcsz_z * z_1,
  data_list     = param_list,
  states        = list(c("z")),
  uses_par_sets = FALSE,
  age_indices   = list(age = c(0:20), max_age = 21),
  evict_cor     = FALSE
)
```

Once we've defined the P_a and F_a kernels, we need to define starting and ending states for each kernel. Age-size structured populations will look a little different from other models, as we need to ensure that all fecundity kernels produce `age-0` individuals, and all survival-growth kernels produce `age` individuals. We define the implementation arguments using `define_impl()`, and set each kernel's `state_start` to `"z_age"`. Because the fecundity kernel produces age-0 individuals, regardless of the starting age, its `state_end` is `"z_0"`.

```
age_size_ipm <-  define_impl(
  proto_ipm = age_size_ipm,
  make_impl_args_list(
    kernel_names = c("P_age", "F_age"),
    int_rule     = rep("midpoint", 2),
    state_start    = c("z_age", "z_age"),
    state_end      = c("z_age", "z_0")
  )
)
```

We define the domains using `define_domains()` in the same way we did for Case Study 1.

```
age_size_ipm <- age_size_ipm %>%
  define_domains(
    z = c(1.6, 3.7, 100)
  )
```

Our definition of the initial population state will look a little different though. We want to create 22 copies of the initial population state, one for each age group in the model. We do this by appending `_age` to the `n_z` in the `...` part of `define_pop_state`. We'll also set `make_ipm(...,return_all_envs = TRUE)` so we can access the computed values for each vital rate function in the model.

```
age_size_ipm <-  define_pop_state(
  proto_ipm = age_size_ipm,
  n_z_age = rep(1/100, 100)
  ) %>%
  make_ipm(
    usr_funs = list(r_fun = r_fun),
    iterate  = TRUE,
    iterations = 100,
    return_all_envs = TRUE
  )
```

Basic analysis

We see that the population is projected to grow by about 1.5% each year. As in Case Study 1, we can check for convergence using the `is_conv_to_asymptotic()` function.

```
lamb <- lambda(age_size_ipm)
lamb
```

```
##   lambda
## 1.014833
```

```
is_conv_to_asymptotic(age_size_ipm)
```

```
## lambda
##   TRUE
```

For some analyses, we may want to get the actual vital rate function values, rather than the sub-kernels and/or iteration kernels. We can access those with `vital_rate_funs()`. Note that right now, this function always returns a full bivariate form of the vital rate function (i.e. for survival, it returns a n X n kernel, rather than a 1 x n vector, where n is the number of meshpoints). It is also important to note that these functions are **not yet discretized**, and so need to be treated as such (i.e. any vital rate with a probability density function will contain probability densities, not probabilities).

```
vr_funs <- vital_rate_funs(age_size_ipm)

# Age 0 survival and growth vital rate functions

vr_funs$P_0
```

```
## s_0 (not yet discretized): A 100 x 100 kernel with minimum value: 0.0019 and maximum value: 0.9995
## g_0 (not yet discretized): A 100 x 100 kernel with minimum value: 0 and maximum value: 5.0691
## mu_g_0 (not yet discretized): A 100 x 100 kernel with minimum value: 2.2556 and maximum value: 3.528
```

```
# Age 12 fecundity functions

vr_funs$F_12
```

```
## s_12 (not yet discretized): A 100 x 100 kernel with minimum value: 0 and maximum value: 0.9744
## pb_12 (not yet discretized): A 100 x 100 kernel with minimum value: 0.0217 and maximum value: 0.9345
## pr_12 (not yet discretized): A 100 x 100 kernel with minimum value: 0.965 and maximum value: 0.965
## recr (not yet discretized): A 100 x 100 kernel with minimum value: 0 and maximum value: 2.5091
## rcsz_mu (not yet discretized): A 100 x 100 kernel with minimum value: 1.5038 and maximum value: 2.97
```

We can also update the model to use a new functional form for a vital rate expression. For example, we could add parent size dependence for the probability of recruiting function. This requires 3 steps: extract the `proto_ipm` object, set the new functional form, and update the parameter list. We have to wrap the assignment in `new_fun_form()` to prevent parsing errors.

```
new_proto <- age_size_ipm$proto_ipm

vital_rate_exprs(new_proto,
                 kernel = "F_age",
                 vital_rate = "pr_age") <-
  new_fun_form(plogis(recr_int + recr_z * z_1 + recr_a * age))

parameters(new_proto) <- list(recr_z = 0.05)

new_ipm <- make_ipm(new_proto,
                    return_all_envs = TRUE)

lambda(new_ipm)
```

```
##   lambda
## 1.017868
```

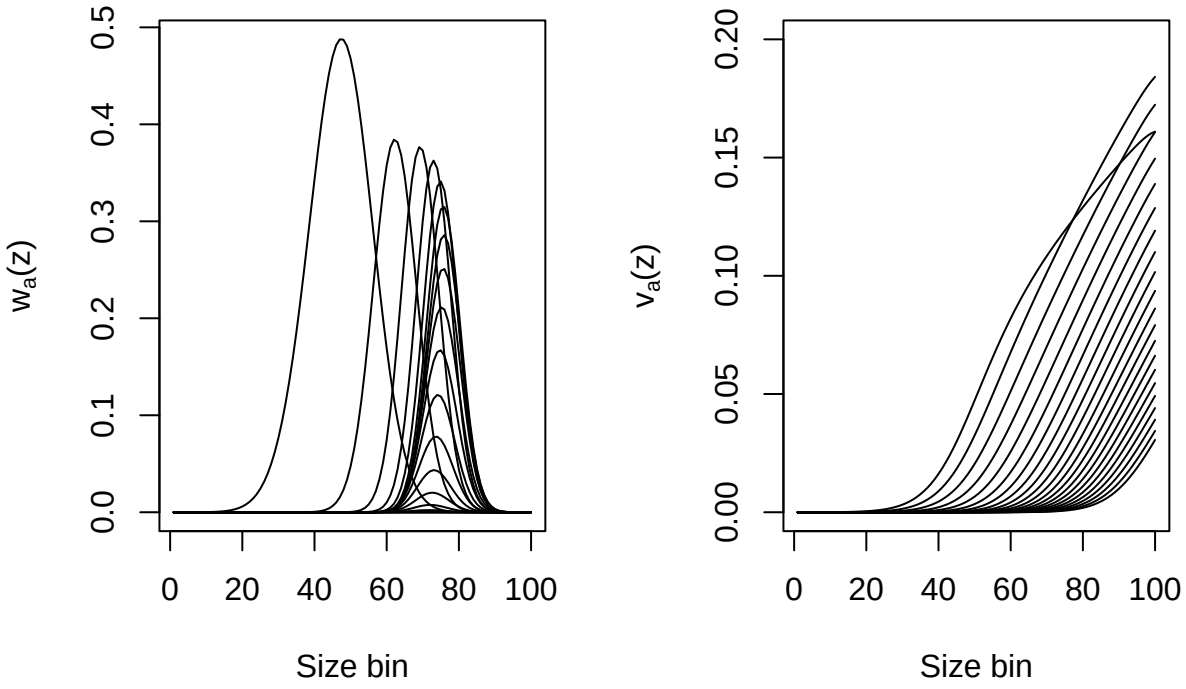
Next, we'll extract and visualize eigenvectors and compute age specific fertility and survival.

Further analyses

The `right_ev` and `left_ev` functions also work for age × size models. We can use extract these, and plot them using a call to `lapply`. We will use the notation from Ellner, Childs, & Rees (2016) to denote the left and right eigenvectors ($v_a(z)$ and $w_a(z)$, respectively).

NB: we assign the `lapply` call to a value here because `lines` returns `NULL` invisibly, and this clogs up the console. You probably don't need to do this for interactive use. We'll also divide the dz value back into each eigenvector so that they are continuous distributions, rather than discretized vectors.

```
d_z <- int_mesh(age_size_ipm)$d_z

stable_dists <- right_ev(age_size_ipm)

w_plot <- lapply(stable_dists, function(x, d_z) x / d_z,
         d_z = d_z)

repro_values <- left_ev(age_size_ipm)

v_plot <- lapply(repro_values, function(x, d_z) x / d_z,
                 d_z = d_z)


par(mfrow = c(1, 2))
plot(w_plot[[1]], type = 'l',
     ylab = expression(paste("w"[a],"(z)")),
     xlab = "Size bin")

x <- lapply(w_plot[2:22], function(x) lines(x))

plot(v_plot[[1]], type = 'l',
     ylab = expression(paste("v"[a],"(z)")),
     xlab = "Size bin",
     ylim = c(0, 0.2))

x <- lapply(v_plot[2:22], function(x) lines(x))
```



Next, we'll compute age-specific survival ($\tilde{l}_a$/`l_a`) and fecundity ($\tilde{f}_a$/`f_a`) values. These are defined as follows:

$$\tilde{l}_a = eP^a c$$

$$\tilde{f}_a = (eFP^a c)/l_a$$

where c is some distribution of newborns.

We'll initialize a cohort using the stable size distribution for age-0 individuals that we obtained above. Next, we'll iterate them through our model for 100 years, and see who's left, and how much they reproduced.

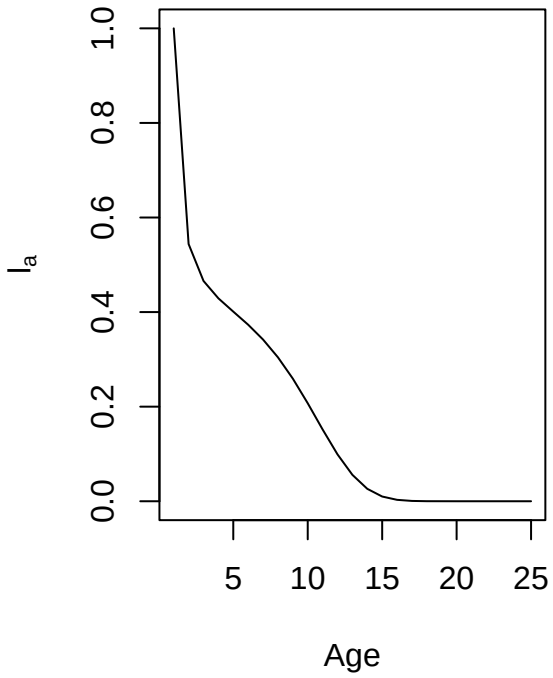
NB: do not try to use this method for computing R_0 - it will lead to incorrect results because in this particular model, parental state affects initial offspring state. For more details, see Ellner, Childs & Rees (2016), Chapters 3 and 6.

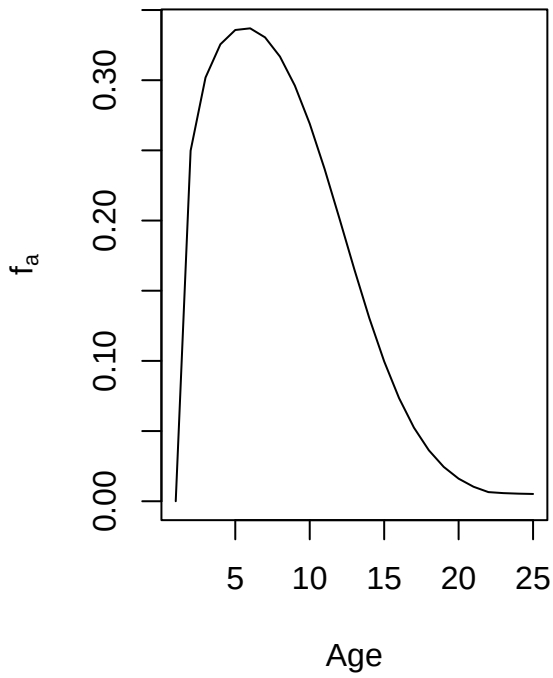
A couple technical notes:

1. We are going to split out our P and F sub-kernels into separate lists so that indexing them is easier during the iteration process.
2. We need to set our `max_age` variable to 22 now, so that we don't accidentally introduce an "off-by-1" error when we index the sub-kernels in our IPM object.
3. We use an identity matrix to compute the initial value of `l_a`, because by definition all age-0 individuals must survive to age-0.

```
# Initialize a cohort and some vectors to hold our quantities of interest.

init_pop <- stable_dists[[1]] / sum(stable_dists[[1]])
```

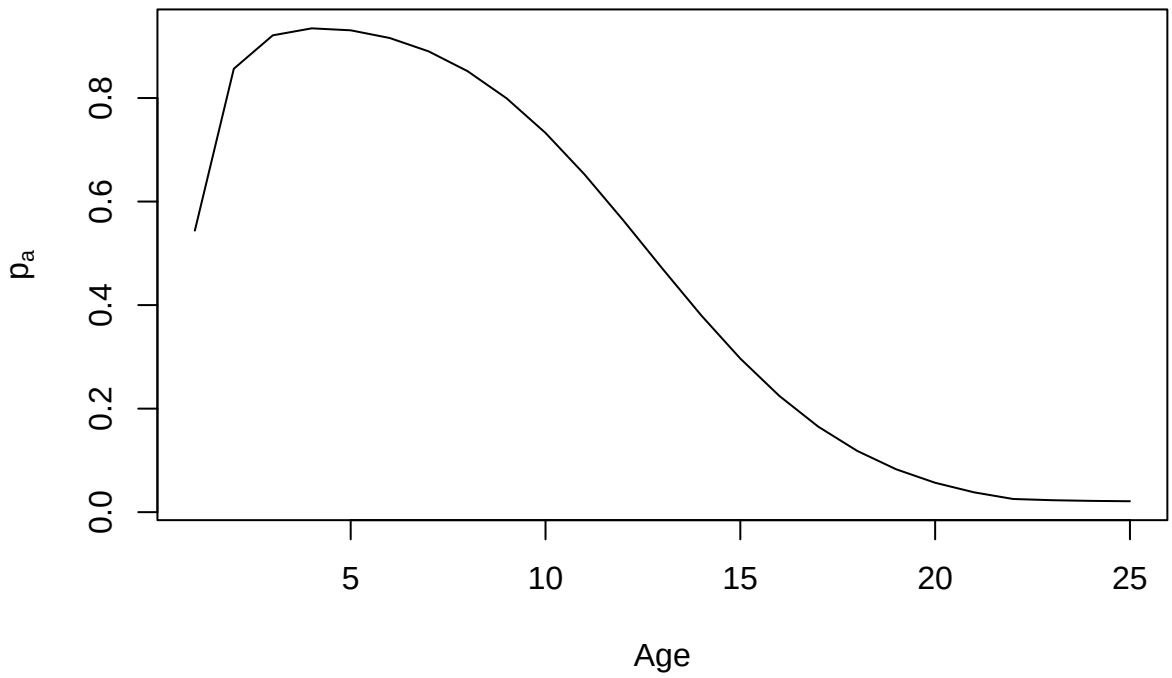
```r
n_yrs    <- 100L

l_a <- f_a <- numeric(n_yrs)

P_kerns <- age_size_ipm$sub_kernels[grepl("P", names(age_size_ipm$sub_kernels))]
F_kerns <- age_size_ipm$sub_kernels[grepl("F", names(age_size_ipm$sub_kernels))]

# We have to bump max_age because R indexes from 1, and our minimum age is 0.

max_age <- 22

P_a <- diag(length(init_pop))

for(yr in seq_len(n_yrs)) {

  # When we start, we want to use age-specific kernels until we reach max_age.
  # after that, all survivors have entered the "greybeard" class.

  if(yr < max_age) {

    P_now <- P_kerns[[yr]]
    F_now <- F_kerns[[yr]]

  } else {

    P_now <- P_kerns[[max_age]]
    F_now <- F_kerns[[max_age]]

  }

  l_a[yr] <- sum(colSums(P_a) * init_pop)
  f_a[yr] <- sum(colSums(F_now %*% P_a) * init_pop)

  P_a <- P_now %*% P_a
}

f_a <- f_a / l_a

# Looks like most are dead at after 25 years, so we'll restrict our
# plot range to that time span

par(mfrow = c(1, 2))

plot(l_a[1:25], type = 'l',
     ylab = expression(paste("l"[a])),
     xlab = "Age")
plot(f_a[1:25], type = 'l',
     ylab = expression(paste("f"[a])),
     xlab = "Age")
```





We can also calculate the age-specific survival probability p_a as $\frac{l_{a+1}}{l_a}$. We'll restrict our calculations to the first 25 years of life, as we'll see that almost no sheep live longer than that.

```
p_a <- l_a[2:26] / l_a[1:25]

plot(p_a, type = 'l',
     ylab = expression(paste("p"[a])),
     xlab = "Age")
```



Citations

Childs DZ, Coulson T, Pemberton JM, Clutton-Brock TH, & Rees M (2011). Predicting trait values and measuring selection in complex life histories: reproductive allocation decisions in Soay sheep. Ecology Letters 14: 985-992.

Clutton-Brock TH & Pemberton JM (2004). Soay Sheep: Dynamics and Selection in an Island Population. Cambridge University Press, Cambridge.