APS 273 - Practical 1

Dr. Andrew Beckerman

24 February 2016

The Life Table

During the module, you have learned about life tables. Here you will construct one in R. The starting point is this.

Age	Population Size	Fecundity
0	500	0
1	400	2
2	200	3
3	100	1
4	50	0

Create this data frame in R. Use the data.frame() function. Call it lt (life table).

```
lt
```

```
##   age pop_size ma
## 1   0      500  0
## 2   1      400  2
## 3   2      200  3
## 4   3       50  1
## 5   4        0  0
```

Before we go any further, you need to recall/remember/be taught how [ ] work in R. It is a bit like the machine that makes select() and slice() work in dplyr. These are a tool to get pieces of a data frame or matrix. The convention is `[row, column]`

For example, in your data frame

```
lt[1,]
```

```
##   age pop_size ma
## 1   0      500  0
```

```
lt[,2]
```

```
## [1] 500 400 200  50   0
```

```
lt[3,2]
```

```
## [1] 200
```

```
lt$ma[2] # look carefully at this.... using $ and [ ]
```

```
## [1] 2
```

Importantly, note that R can do operations with these, and you can work with many rows or columns.

```
lt[2:5,2] # rows 2:5 in column 2
```

```
## [1] 400 200  50   0
```

```
lt[2:5,2]/lt[1:4,2] # divide rows 2:5 by 1:4 in column 2
```

```
## [1] 0.80 0.50 0.25 0.00
```

Now, use [ ]'s, dplyr and mutate() to add colums for la and pa. Recall that la is the proportion of individuals surviving TO age a from age 0. Recall that pa is the proportion of individuals surving from age a-1 to age a.

One other think you need to recall is `c()`. It is probably clear to you that you need to have an empty cell in the pa column.... and we never leave things empty, but fill them with `NA`. So, think about how to use `c(NA, ....)` to generate your column.

After you've been successful, you should have this:

```
lt
```

```
##   age pop_size ma  la   pa
## 1   0      500  0 1.0   NA
## 2   1      400  2 0.8 0.80
## 3   2      200  3 0.4 0.50
## 4   3       50  1 0.1 0.25
## 5   4        0  0 0.0 0.00
```

Now, lets think about how to create a pre-breeding census Matrix. The first step is calculating fertlilites. In the pre-breeding matrix, Fertilities are a function of the ma and of the survival from age 0-1 only. So only a single pa is used....

Locate this pa and again use mutate() to generate a column in lt called Fa. When you are successful, this should be what you have.

```
lt
```

```
##   age pop_size ma  la   pa  Fa
## 1   0      500  0 1.0   NA 0.0
## 2   1      400  2 0.8 0.80 1.6
## 3   2      200  3 0.4 0.50 2.4
## 4   3       50  1 0.1 0.25 0.8
## 5   4        0  0 0.0 0.00 0.0
```

OK. Now we have the basics of a pre-breeding Model. We have pa values for the transitions and we have Fa for the combination of babies produced and their survival to age 1. Super.

One a piece of paper now, draw the life cycle.

OK. Now, matrices. They are quite easy in R, and map onto your life cycle.

.	.	.	.
F1	F2	F3	F4
p1	0	0	0
0	p2	0	0
0	0	p3	0
.	.	.	.

To construct this, you need to just think about [ ]'s again. You've got a column of Fa that needs to go into a row in a matrix. Matrices have row and column dimensions. Lets make one with NA's in it, and then fill it in.

```
BP <- matrix(NA, nrow = 4, ncol = 4)
BP
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   NA   NA   NA   NA
## [2,]   NA   NA   NA   NA
## [3,]   NA   NA   NA   NA
## [4,]   NA   NA   NA   NA
```

Great... perhaps you can see what we can do?

```
# dump Fa column into BP row 1.
# (note we left out the first element)
BP[1,] <- lt$Fa[2:5]
BP
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.6  2.4  0.8    0
## [2,]   NA   NA   NA   NA
## [3,]   NA   NA   NA   NA
## [4,]   NA   NA   NA   NA
```

Now, add the pa's yourself

```
BP
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.6  2.4 0.80    0
## [2,]  0.8   NA   NA   NA
## [3,]   NA  0.5   NA   NA
## [4,]   NA   NA 0.25   NA
```

Before we move on, however, we need to replace all those NA's in the matrix with 0... This will make things work later on.

```
is.na(BP) # shows you where the NA's are
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] FALSE FALSE FALSE FALSE
## [2,] FALSE  TRUE  TRUE  TRUE
## [3,]  TRUE FALSE  TRUE  TRUE
## [4,]  TRUE  TRUE FALSE  TRUE
```

```
BP[is.na(BP)]<-0 # replaces the NA's with 0
BP # see the result
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.6  2.4 0.80    0
## [2,]  0.8  0.0 0.00    0
## [3,]  0.0  0.5 0.00    0
## [4,]  0.0  0.0 0.25    0
```

We are ready to calculate things. Lets first calcuate Ro from the life table. Remember that Ro is the sum of the product of la and ma. See if you can use dplyr and summarise() to generate this:

```
Ro
```

```
##    Ro
## 1 2.9
```

Cool. Now, there are function in R to help get the eigenvalues. Remember that the dominant eigenvalue is = lambda and r = ln(lambda) because eˆr is lambda. lambda >1 and r>0 are growing populations.

The function that does this is, wait... eigen. And it takes a matrix. The matrix.

```
eigen(BP)
```

```
## $values
## [1]  2.4404641 -0.6334746 -0.2069895  0.0000000
##
## $vectors
##              [,1]       [,2]        [,3] [,4]
## [1,] -0.948297661  0.5168677 -0.06590256    0
## [2,] -0.310858134 -0.6527399  0.25470881    0
## [3,] -0.063688322  0.5152060 -0.61526987    0
## [4,] -0.006524202 -0.2033254  0.74311726    1
```

So, `eigen(BP)$values[1]` is the dominant eigenvalue and is lambda, the population growth rate

```
eigen(BP)$values[1]
```

```
## [1] 2.440464
```

```
log(eigen(BP)$values[1])
```

```
## [1] 0.8921882
```

Projection

OK. So now we need to recall a tiny bit about how the matrix multiplication works to project a population into the future. Projection requires that we specify a starting value (population size) for each age. We can do that by creating a `vector` called N0.

```
N0 <- c(2,2,2,2) # 4 values for 4 ages we are counting.
```

Matrix multiplication in R works with the %*% symbols. We provide the matrix and vector.

```
BP %*% N0
```

```
##      [,1]
## [1,]  9.6
## [2,]  1.6
## [3,]  1.0
## [4,]  0.5
```

So, if we start Year 0 with 2 individuals in EACH age, in Year 1, this is what we have. How can we do this a tonne of times and collect the data? Here's how.

First, we set up a collection zone for all the numbers. We have 4 age classes, and we'll make these the columns. Lets plan to project the matrix 25 years. So we need 100 rows.

In R, one of the ways to make things fast is to generate all these slots to fill. 4 100 rows x 4 columns worth. We can quickly make this with a matrix. And then we can turn this into a data frame.

```
nits <- 25 # how many years will we project
tmp <- matrix(NA, nrow = nits, ncol = 4)
collect <- data.frame(tmp)
names(collect)<-c("Age1", "Age2","Age3", "Age4")
head(collect) # just look at the first six rows
```

```
##   Age1 Age2 Age3 Age4
## 1   NA   NA   NA   NA
## 2   NA   NA   NA   NA
## 3   NA   NA   NA   NA
## 4   NA   NA   NA   NA
## 5   NA   NA   NA   NA
## 6   NA   NA   NA   NA
```

Right. NOW, lets put our starting values into the first row.

```
collect[1,] <- N0
head(collect)
```

```
##   Age1 Age2 Age3 Age4
## 1    2    2    2    2
## 2   NA   NA   NA   NA
## 3   NA   NA   NA   NA
## 4   NA   NA   NA   NA
## 5   NA   NA   NA   NA
## 6   NA   NA   NA   NA
```

Sweet. Now for some REAL programming. Lets iteratively populate each row. We start with Row 1 of collect, multiple this vector by the matrix BP, and push the results into Row 2. If we then repeat this with Row 2, we get Row 3 and so on.

To do this, we will make a `for loop`:

```
for (a in 2:nits){
  collect[a,] <- BP %*% t(collect[(a-1),])
  }
```

There is quite a lot there. Lets look at the pieces.

- a is the counter. It starts at 2, because that is the next row we will add to.

- a-1 looks back one row to get the values of the population size.
- collect is our collection zone; collect[a,] is the a'th row
- BP is our projection matrix. It makes the population grow. We multiple BP times the numbers at a-1 to get the numbers now at a.
- `t()` is a function that transposes the vector. We grabbed a row. It needs to be a column!

If this worked, we should now have a full collection box

```
head(collect)
```

```
##       Age1     Age2    Age3 Age4
## 1   2.0000  2.00000  2.0000 2.00
## 2   9.6000  1.60000  1.0000 0.50
## 3  20.0000  7.68000  0.8000 0.25
## 4  51.0720 16.00000  3.8400 0.20
## 5 123.1872 40.85760  8.0000 0.96
## 6 301.5578 98.54976 20.4288 2.00
```

```
tail(collect)
```

```
##          Age1       Age2      Age3     Age4
## 20   80114279   26261981   5380530   551179
## 21  195516024   64091423  13130990  1345132
## 22  477149846  156412819  32045711  3282748
## 23 1164467088  381719877  78206410  8011428
## 24 2841840173  931573671 190859938 19551602
## 25 6935409036 2273472138 465786835 47714985
```

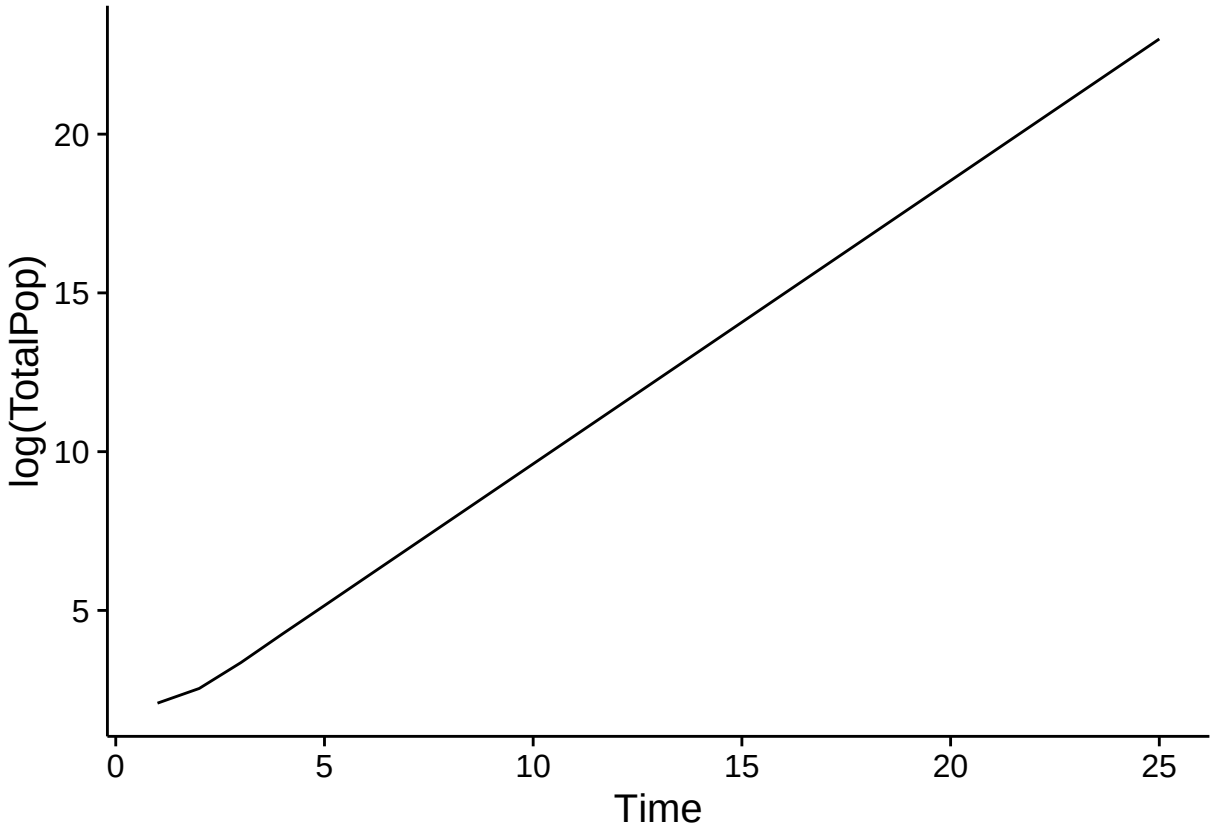
Fantastic. It should not be too surprising that we have exponetial growth here, with MASSIVE populations sizes. I hope you all remember that if we plot these data, it WILL look like an exponential growth curve. And I hope you all remember that if we `log` exponential data, we turn it into a straight line. This DOES have value, as we'll see.

Visualising these data

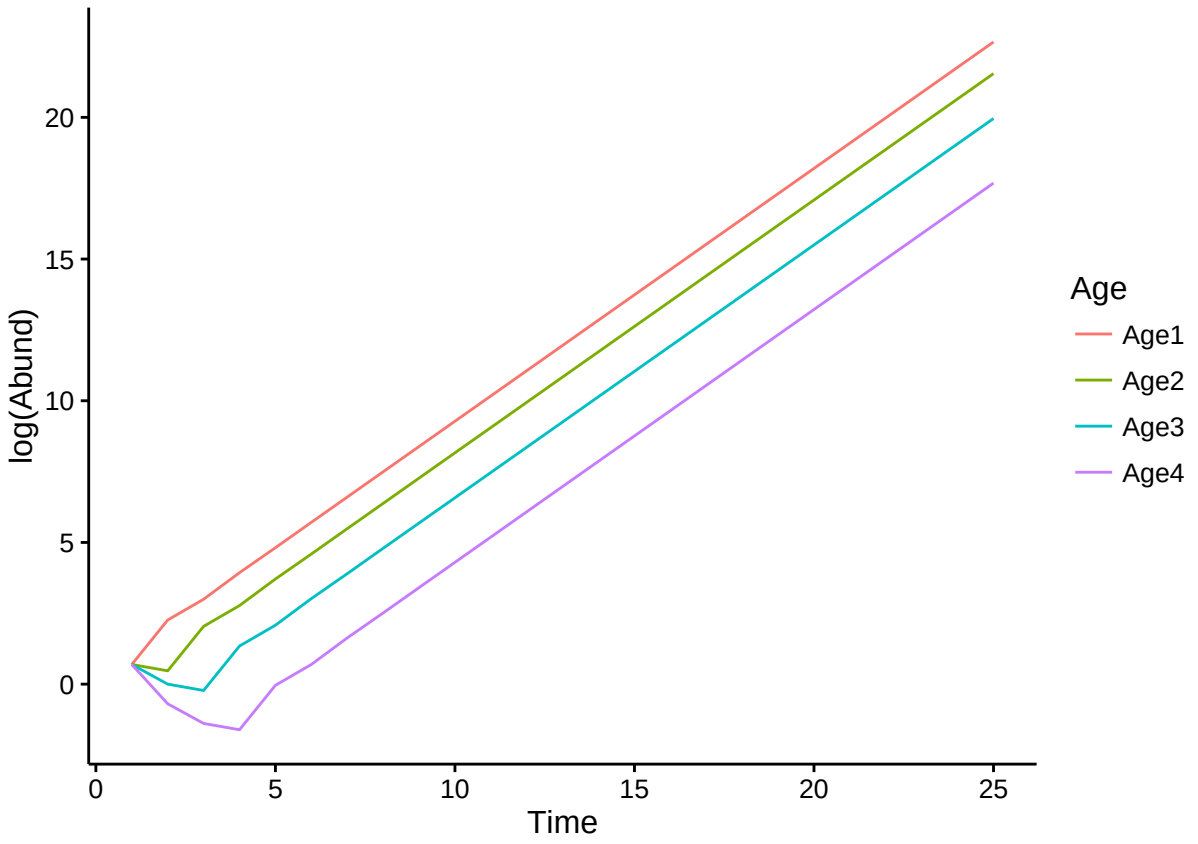
The first thing we do is create a total populaton size column in the data frame. After that, we can plot these data.

```
Total<-mutate(collect,
  TotalPop = Age1+Age2+Age3+Age4,
  Time = 1:nits)
```

```
ggplot(Total, aes(x = Time,  y = log(TotalPop)))+
  geom_line()+
  theme_classic(base_size = 15)
```



That's nice, but what we want is the following, showing the trajectory of each age class; note how log(Abundance) is on the y-axis and the parallel lines indicate something very useful!:

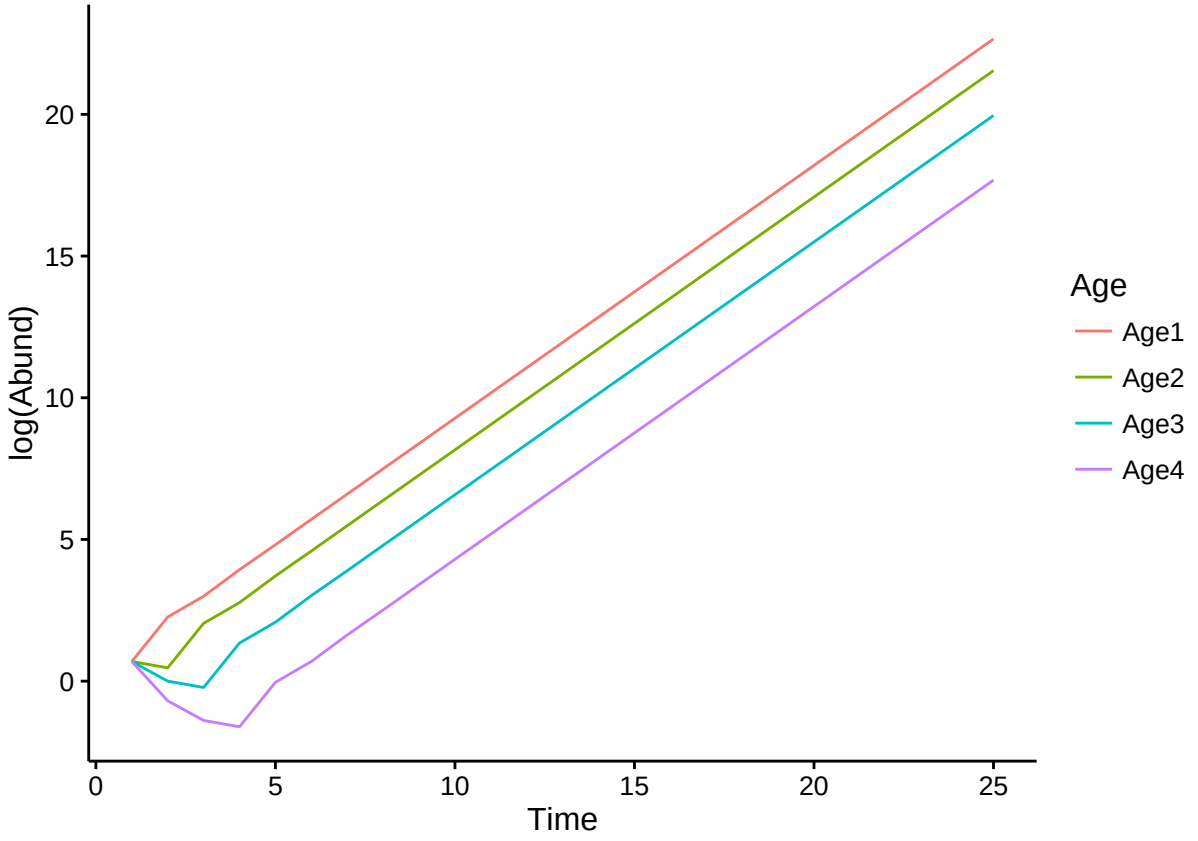


How to do this

To really take advantage of ggplot here, and the automatic legend etc, we use the reshape2 package to turn these "wide" data into a long format data frame. We take Age1 - Age 4 labels and make them into a variable called Age and the values from those 4 columns go into a column called Abund. Time gets replicated for us!

```
library(reshape2)
long<-melt(Total,
           # this is the fixed variable
           id.vars = c("Time"),
           # these are what we want to stack up
           measure.vars=c("Age1","Age2","Age3","Age4"),
           # this is what we will label the stacked categories
           variable.name = "Age",
           # this is what we label the stacked values
           value.name = "Abund")

# now use ggplot!
ggplot(long, aes(x =Time, y = log(Abund), group = Age, col = Age))+
  geom_line()+
  theme_classic()
```



One thing we can see, having plotted this on the log scale, is that the lines are parallel, indicating that by year 5, we have reached a stable age distribution. Can you see that? There is a fixed proportion of the population in each age now. And here is how you can use dplyr to PROVE this.

```
summarise(group_by(Total, Time),
          Age1_Prop = Age1/TotalPop,
          Age2_Prop = Age2/TotalPop,
          Age3_Prop = Age3/TotalPop,
          Age4_Prop = Age4/TotalPop)
```

```
## Source: local data frame [25 x 5]
##
##     Time Age1_Prop Age2_Prop  Age3_Prop   Age4_Prop
##    (int)     (dbl)     (dbl)      (dbl)       (dbl)
## 1      1 0.2500000 0.2500000 0.25000000 0.250000000
## 2      2 0.7559055 0.1259843 0.07874016 0.039370079
## 3      3 0.6961364 0.2673164 0.02784546 0.008701706
## 4      4 0.7181910 0.2249972 0.05399933 0.002812465
## 5      5 0.7120450 0.2361645 0.04624149 0.005548979
## 6      6 0.7136848 0.2332338 0.04834803 0.004733321
## 7      7 0.7132560 0.2339963 0.04779407 0.004953718
## 8      8 0.7133676 0.2337981 0.04793852 0.004895755
## 9      9 0.7133386 0.2338496 0.04790097 0.004910863
## 10    10 0.7133461 0.2338362 0.04791072 0.004906936
## ..   ...       ...       ...        ...         ...
```

Finally, it is worth noting that when we did the `eigen()` analysis to get the growth rate, we also received

some eigenvectors. The first column of this set of vectors, when normalised to sum to 1, returns exactly what we just saw above... the stable age distribution.

```
v1 <- eigen(BP)$vectors[,1]
v1_Sum <- sum(eigen(BP)$vectors[,1])
# Stable Age
v1 / v1_Sum
```

```
## [1] 0.713344562 0.233838982 0.047908711 0.004907746
```

Brute Force Sensitivity

The final thing to learn is how to change values in the matrix and re-evaluate the growth rate (e.g. using eigen() to get population growth). We might want to do this to assess whether increasing fertility or survival will benefit the population. Or simulate whether habitat loss for juveniles or adults might mess things up.

Remember that our original matrix is called BP. Lets look at it again.

```
BP
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.6  2.4 0.80    0
## [2,]  0.8  0.0 0.00    0
## [3,]  0.0  0.5 0.00    0
## [4,]  0.0  0.0 0.25    0
```

Lets imagine now that a hotel development along the beach is threatening age 3 reproduction, with the potential for a 50% decrease. It is currently 0.8. 50% less is 0.8 * 0.5.... right? Or 0.8 - 0.5 * 0.8.

To do this, recall [row, column] syntax. And some good practice, which involves using a copy of the original matrix.

```
# create a copy of BP
BP_change<-BP

# decrease row 1, col 3 by 50% and make sure it is in the BP_change
BP_change[1,3] <- BP[1,3]*0.5

# compare row 1, col 3
BP
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.6  2.4 0.80    0
## [2,]  0.8  0.0 0.00    0
## [3,]  0.0  0.5 0.00    0
## [4,]  0.0  0.0 0.25    0
```

```
BP_change
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.6  2.4 0.40    0
## [2,]  0.8  0.0 0.00    0
## [3,]  0.0  0.5 0.00    0
## [4,]  0.0  0.0 0.25    0
```

```r
# Now check growth rate of both to compare
round(eigen(BP)$value[1],2)
```

```
## [1] 2.44
```

```r
round(eigen(BP_change)$value[1],2)
```

```
## [1] 2.42
```

Hooray. A 50% decrease in age 3 fertility has almost no effect on population growth.