

BIO311: Population Ecology
Practical 10:
Stochasticity in Matrix Models
Population Viability Analysis
& Metapopulation Dynamics.

Timothée Bonnet &* Koen van Benthem

timothee.bonnet@ieu.uzh.ch
koen.vanbenthem@ieu.uzh.ch

Spring 2016

Contents

1	Environmental Stochasticity	2
2	Population Viability Analysis	6
3	Demographic Stochasticity (Demonstration only)	8
4	Metapopulation dynamics	12

*This document was co-authored by Tina Cornioley

1 Environmental Stochasticity

In practical 9 we investigated the population dynamics for two populations of yellow-necked mice. Here we focus on one of the two populations presented in that practical: the population living on the plain. For this population the Leslie matrix was given by:

$$\mathbf{A} = \begin{pmatrix} 0.00 & 2.00 \\ 0.25 & 0.50 \end{pmatrix}$$

It turns out however that the conditions for the population are not the same every year. Measurements have been done (okay, this is what we say, in reality we made up some numbers that give reasonable results in the end...) for three years in a row and the matrices that were found for these years are shown below:

$$\begin{array}{lll} \mathbf{A}_1 = \begin{pmatrix} 0.00 & 2.00 \\ 0.25 & 0.50 \end{pmatrix} & \mathbf{A}_2 = \begin{pmatrix} 0.00 & 2.00 \\ 0.27 & 0.54 \end{pmatrix} & \mathbf{A}_3 = \begin{pmatrix} 0.00 & 1.80 \\ 0.20 & 0.44 \end{pmatrix} \\ \lambda = 1 & \lambda = 1.052879 & \lambda = 0.8590618 \end{array}$$

We see that the year that we considered was a relatively average year. The year after the survival rates of the individuals was higher at both stages. In the third year however the population experienced some severe conditions with the survival rates and the fertility rate being lower than in both preceding years. We thus have to realise that the transition rates in nature will vary from year to year. This would mean that the Leslie matrix would also look differently every year. Two possibilities of taking this into account when predicting the behaviour of the population are given below:

1. From the matrices we see that the mean fertility rate is $\mu_F = 1.9333333$ with a standard deviation of $\sigma_F = 0.1154701$. For the survival rates we find:

$$\begin{array}{ll} \mu_{S_j} = 0.24 & \sigma_{S_j} = 0.0360555 \\ \mu_{S_a} = 0.4933333 & \sigma_{S_a} = 0.0503322 \end{array}$$

For every time step that we project the population in the future, we generate a matrix \mathbf{A} with a random fertility rate (drawn from a normal distribution with mean μ_F and standard deviation σ_F), a random juvenile survival rate (drawn from a normal distribution with mean μ_{S_j} and standard deviation σ_{S_j}) and a random adult survival rate (drawn from a normal distribution with mean μ_{S_a} and standard deviation σ_{S_a}). This way we will incorporate the fact that the rates can vary over time.

Quick refresher: standard deviation

The standard deviation describes how much variation there is in the data from the average. A large standard deviation means that the data points deviate a lot from the average whereas a small standard deviation indicates that the data points are close to the average.

Mathematically it is defined as:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu)^2}.$$

In this equation μ is the average value of x (thus $\mu = \frac{1}{N} \sum_{i=1}^N x_i$) and N is the total number of data points. In R you can use the function `sd()` to calculate the standard deviation.

2. Every time step we randomly choose one of the measured matrices (A_1, A_2, A_3) with equal probabilities. Thereby taking into account the variation in the environment.

Which of these approaches do you prefer in this case? And why?

1.1 Independent sequences

In the rest of this practical, we will explore what happens if we decide to model the dynamics by randomly choosing one of the three matrices each time step. Assume an initial population of 5 juveniles and 5 adults. Create a matrix `N` to store the numbers that you will calculate. Now use the `sample()` function to generate a random number every time step and use this number (using `if/else`) to decide which matrix is used in a specific time step. To illustrate the use of `sample`

```
sample(c(1,2,3,4),1)

## [1] 1
```

randomly draws 1 entry out of the vector `c(1,2,3,4)`. Now use this function to generate one possible prediction of the population dynamics until time step 50.

```
A1<-...
A2<-...
A3<-...

timesteps<-... # Total number of timesteps

N<-... # Create matrix to store the calculated values
N[,1]<-...

for(t in ...){

  test<-..... # Use sample to generate
```

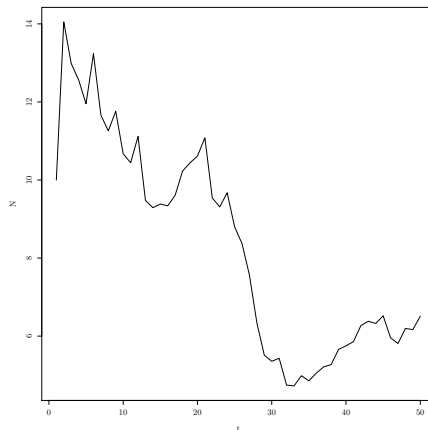
```

# a random number (1,2,3)

if(test==..){
  ...
}else if(test==...){
  ...
}else{
  ...
}
}
}
Ntot<-colSums(N) # Calculate total population size
                 # per timestep

plot(...)

```



You will probably notice that your result looks different from the result in the graph above. The reason for this is that the outcome depends on random numbers. It is actually not really interesting to see what one prediction tells us. Instead, write a loop that repeats the whole prediction multiple times (for example 200 times) and plot the results of all these predictions in one graph. For this it is best to start with an empty plot and add the different realisations using the `lines()` function (see practical 7, page 20 where we used a similar approach for stochasticity in the logistic growth model). In English, what we want to do is:

1. Define the number of different simulations (replicates) that you want to perform
2. Create an empty plot
3. Run one of the simulations

4. Use `lines()` to add a line with that outcome to the plot. Use the argument `col` to specify a color for this line. If you want to automatically generate a large number of different colors, take a look at the function `rainbow()`.
5. If you have not yet done all the simulations, return to step 3

The more lines you draw on one plot, the more complicated the interpretation of the plot becomes. In order to make the interpretation a bit easier, we recommend drawing a line that corresponds to the average behaviour of all the lines. To do so, we create a vector `TotalN` at the beginning of the code (before the loops) that first contains only zeros:

```
TotalN<-rep(0,timesteps)
```

at the end of the calculation of each simulation we have the vector `Ntot` that contains the simulated number of individuals at each time step. (In `Ntot`, `tot` refers to the fact that it is both the juveniles and the adults. In `TotalN`, `Total` refers to the fact that this vector contains the sum of all the different predictions), we add that line to `TotalN`:

```
TotalN<-TotalN+Ntot
```

Finally, after having calculated all the different predictions, divide `TotalN` by the number of predictions to obtain the main behaviour. You can add this behaviour to the plot with `lines()` after the calculations have been finished. Change the `lwd` parameter to draw a thicker (more visible) line and the colour with `col=` to be able to distinguish the line. What is the trend of this average line? Is this what you expected based on the initial matrices?

1.2 Temporal environmental correlations

So far we have assumed that the chances of applying either matrix (\mathbf{A}_1 , \mathbf{A}_2 , \mathbf{A}_3) is the same. In some cases however, the conditions experienced by a population vary slowly: It is more likely that a population is exposed to similar conditions at the next time step compared to the current time step than to different conditions. In terms of matrices, this means that if at time t matrix \mathbf{A}_1 is applied, it is more likely that at the next time step the same matrix is applied than that one of the other ones is used. As an example you can think of copper pollution: this pollution is expected to last multiple years, so if there is pollution this year, there will probably still be some pollution next year. Assume for example that matrix \mathbf{A}_3 describes the dynamics of the population in case of copper pollution and the population experiences this pollution at a time t . In this case we expect the copper pollution to still be there at time $t + 1$ and therefore we assume that the chance of applying \mathbf{A}_3 at time step $t + 1$ should be larger than the chance of applying one of the other two matrices that describe good conditions.

You will now incorporate this property in the model that you have already developed. The `sample()` function can be given a parameter `prob` that specifies

the probability of drawing a specific number. Try this by repeatedly performing the following line:

```
sample(c(1,2,3),1,prob=c(4/7,2/7,1/7))
```

Now you can use this property to change the previous simulation to account for temporal auto-correlation. First, before entering the time loop (but after entering the loop that is used for calculating the different predictions), specify a new vector that will hold the probabilities. At this time (the first time point), all the matrices are still equally likely to occur.

```
probs=c(1/3,1/3,1/3)
```

Now adapt your previous code to use these probabilities to draw a new matrix every time step. Once a matrix has been drawn in a time step, change the vector **probs** such that the next time step there will be a chance of $\frac{5}{7}$ to have the same matrix again and a chance of $\frac{1}{7}$ to have either of the other matrices in the next time step. In English, what you have to do is:

1. Define a **probs** vector within the first loop but outside of the second. This is the vector that determines which one of the matrix is the first matrix of the time step.
2. Specify the **prob** argument of the **sample** function to be the vector **probs**.
3. After each **if/else** statements within the second loop, define a new **probs** vector which will change the probability of the matrices to be used.

How do these predictions differ from the predictions in the non-correlated case? Does this make sense to you?

2 Population Viability Analysis

What managers want to know generally is not so much the average prediction, but rather the chance of extinction. What is the chance that this population will go extinct in the next 50 year? In order to look into this, we first need to define 'extinct': with the matrix models the vectors will never really reach 0, but they can become really small (say a total population size of 0.001). We thus need to decide when we call a population extinct. In this case we decide that if the population contains less than 2 individuals, it basically went extinct. If the population size falls below this threshold, we do not actually multiply the population by the matrix, but we just set the new value of the population vector to be $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$. To make the code easier to adapt, we first define a threshold variable:

```
threshold<-2
```

Now within the second loop and before the other statements, add an `if` statement to your code to account for this possibility. You want that if the total population size at the former time step is below the threshold, the current population size is set to 0.

Finally we would like to keep track of the number of extinctions over time. In order to so, create a vector `extinct` at the beginning of your code, before entering any loop:

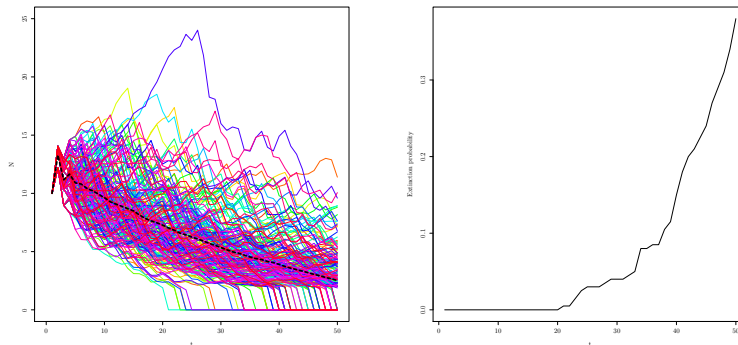
```
extinct<-rep(0,timesteps)
```

Now adapt your code such that if the population size of a specific prediction at some point in time falls below the threshold, not only the population vector is set to $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$, but also the extinction is counted, that is that in that case:

```
extinct[t]<-extinct[t]+1
```

At the end, to find the proportion of predictions that has gone extinct, you just have to divide the vector `extinct` by the number of predictions that it is based on. If this number is large enough, the proportion of extinct predictions is the probability of extinction for that population. Implement this in the code after the loops and plot how the probability of extinction changes over time. What happens if your initial population is smaller? (for example 2 adults and 2 juveniles) What is the difference in extinction probabilities between the case of time correlated environments and time uncorrelated environments? Does this make sense to you?

If you start with a population of 5 adults and 5 juveniles and make 200 simulations, your (average) result should look similar to the following graphs (in time uncorrelated environments):



3 Demographic Stochasticity (Demonstration only)

A problem that we have not talked about so far is the fact that matrices allow for half individuals. You may for example find that the expected number of juveniles after 10 time steps equals 2.4, but what are 0.4 individuals? We interpret this as an 'average'. It is however also possible to take into account the fact that half an individual can not exist. For this we do not use the matrix framework, but we can use the rates that we also used in the matrix framework. In this section we will only consider \mathbf{A}_1 . That is: we have juvenile survival of 0.25, and adult survival of 0.5 and an average fertility rate of 2. If we would do the matrix multiplication, we would write for the number of juveniles (N_j) and the number of adults (N_a):

$$\begin{aligned}N_j(t+1) &= 2 \cdot N_a(t) \\ N_a(t+1) &= 0.25 \cdot N_j(t) + 0.5 \cdot N_a(t)\end{aligned}$$

This will however give rise to half individuals as well.

In order to get rid of this problem we interpret the juvenile survival rate as follows: the juvenile survival rate is the chance that an individual juvenile survives. We can simulate this in R with the function `rbinom(n,size,prob)`. This function simulates how many times an event occurs if we make `size` attempts and the chance of an attempt succeeding is `prob`. The element `n` specifies how many simulations of this type we want to have.

For example we have a dice with 6 sides and we want to roll it 100 times and we want to know how many of these 100 times the number on the dice is either 1 or 2. The chance of rolling either a 1 or a 2 is $\frac{1}{6} + \frac{1}{6} = \frac{1}{3}$, so we expect this to happen 33.3333... times if we roll the dice 100 times. Let us simulate this in R:

```
rbinom(n=1,size=100,prob=1/3)
## [1] 31
```

We see that this function indeed returns us an integer number. R has in a sense simulated 100 dice rolls and counted how many times the outcome was 1 or 2. If we let R simulate this experiment multiple times, we will find different outcomes:

```
rbinom(n=10,size=100,prob=1/3)
## [1] 33 32 25 36 22 25 30 42 34 32
```

However, if we average the outcomes of very many of these simulations, we should find back the expected 33.333...:


```
mean(rbinom(n=10000,size=100,prob=1/3))  
## [1] 33.346
```

We can now use this function to simulate the number of juveniles and adults that survive as follows: (recall that $N[1,t]$ describes the number of juveniles at time t and $N[2,t]$ the number of adults at time t)

```
N[2,t]<-rbinom(n=1,size=N[1,t-1],prob=0.25)  
+ rbinom(n=1,size=N[2,t-1],prob=0.5)
```

Now all that is left is calculating the amount of offspring that is produced at time t . One way of doing this is by drawing a number from a Poisson distribution with $\lambda = 2$. The Poisson distribution is a distribution that is frequently used for count data, but it is beyond the scope of this course to treat it in detail. To draw a number from a Poisson distribution in R, we use the function `rpois(n,lambda)`. Here `n` specifies the number of numbers that we want to draw from the distribution and `lambda` specifies the mean of the distribution. So to draw one number we write:

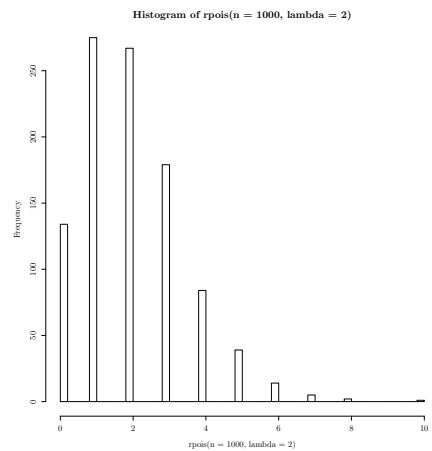
```
rpois(n=1,lambda=2)  
## [1] 3
```

Let us now check whether the mean value of many numbers drawn from a Poisson distribution is indeed close to 2:

```
mean(rpois(n=10000,lambda=2))  
## [1] 2.0304
```

Finally, let us have a look at which numbers are drawn most frequently when we use a Poisson distribution:

```
hist(rpois(n=1000,lambda=2),breaks=40)
```



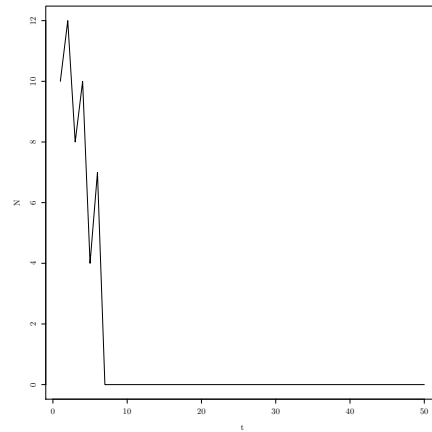
This function would thus describe how many offspring each adult produces. If we want to know the total number of offspring produced, we can write:

```
N[1,t]<-sum(rpois(N[2,t-1],lambda=2))
```

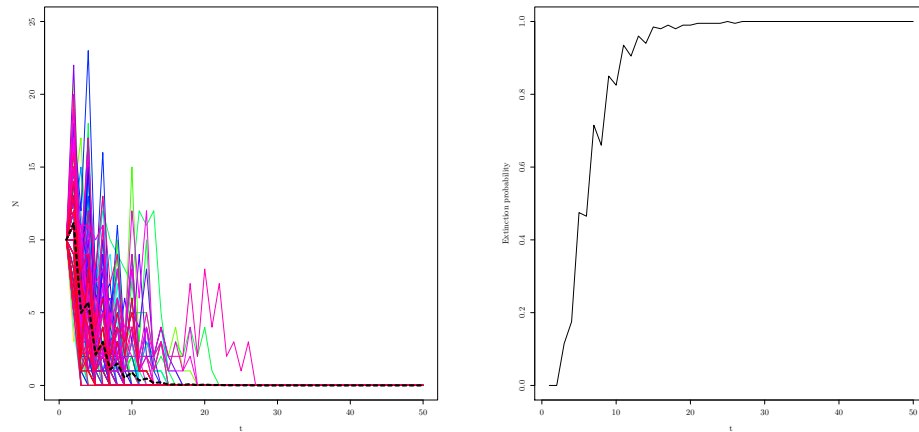
Let us now use this to run one simulation:

```
timesteps<-50
N<-matrix(NA,ncol=timesteps,nrow=2)

N[,1]<-c(5,5)
for(t in 2:timesteps){
  N[1,t]<-sum(rpois(N[2,t-1],lambda=2))
  N[2,t]<-rbinom(n=1,size=N[1,t-1],prob=0.25)
  + rbinom(n=1,size=N[2,t-1],prob=0.5)
}
Ntot<-colSums(N)
plot(1:timesteps,Ntot,type="l",xlab="t",ylab="N")
```



Now we repeat the simulation 200 times and look at the results (the code is not shown, else the previous exercise would become a bit too easy...)



Only because we took into account demographic stochasticity, we see that the population goes extinct in 100% of the simulations. If we compare this to the case where we have just A_1 and deterministically apply this matrix every time step:

```
A1<-matrix(c(0,0.25,2,0.5), nr=2)
timesteps<-50
N<-matrix(NA,ncol=timesteps,nrow=2)
N[,1]<-c(5,5)

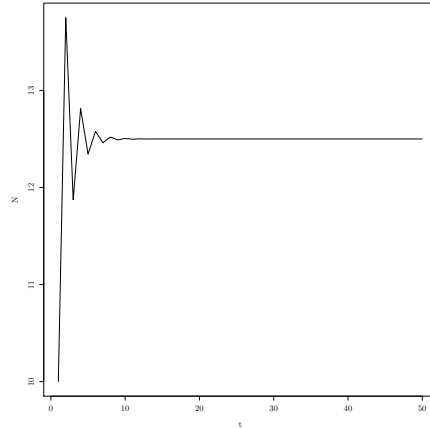
for(t in 2:timesteps){
  N[,t]<-A1*%N[,t-1]
```

```

}

Ntot<-colSums(N)
plot(1:timesteps,Ntot,type="l",xlab="t",ylab="N")

```



We see that here as expected (because $\lambda = 1$) the population size stabilizes at one single value (12.5).

4 Metapopulation dynamics

So far we have considered only populations in isolation of each other. For instance, in practical 9 we have compared two populations of yellow-necked mice, but have assumed that they were not exchanging migrants. More in general, in all matrix models so far, we have completely ignored the fact that new individuals appear in a population not only because of birth, but also because of immigration. Likewise, death is not the only factor removing individuals from a population, permanent emigration does the job as effectively.

Let us consider two populations of peregrine falcons. The demography of the two populations is summarized by the Leslie matrices \mathbf{B}_1 and \mathbf{B}_2 and by the associated asymptotic growth rates:

$$\mathbf{B}_1 = \begin{pmatrix} 0.00 & 0.25 \\ 0.72 & 0.77 \end{pmatrix} \quad \mathbf{B}_2 = \begin{pmatrix} 0.00 & 0.32 \\ 0.74 & 0.79 \end{pmatrix}$$

$$\lambda = 0.9579092 \quad \lambda = 1.021758$$

One of the populations seem to be in good shape, while the other one is likely to be declining (but remember that stochasticity can make predictions based on asymptotic behaviour unreliable). Let us assume that the differences between the two populations is due to the massive use of DDT as an insecticide around population 1, while population 2 is located in a remote mountain area with less human influence.



Figure 1: One of these birds is a peregrine falcon in a dive attack. The other birds are starlings and are in trouble. Populations of peregrine falcons, and other bird species, have declined during the 20th century in North America and Europe as a result of the massive use of the insecticide dichlorodiphenyl-trichloroethane. DDT, which causes eggshell thinning, has now been banned from these regions but is still used in tropical regions to fight malaria. Picture by Rob Palmer.

For convenience, we can write these two matrices into a single matrix, \mathbf{B} , of dimension 4×4 :

$$\mathbf{B} = \begin{pmatrix} 0.00 & 0.25 & 0.00 & 0.00 \\ 0.72 & 0.77 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.32 \\ 0.00 & 0.00 & 0.74 & 0.79 \end{pmatrix}$$

These two populations are not ecologically isolated, however, and they exchange a number of migrants, both juveniles and adults every year. Migration rates vary depending on age, and are not symmetrical. Thus, 73% of juveniles born in the population 1 stay there (and therefore 50% emigrate to population 2), while 50% of juveniles born in population 2 stay there (50% emigrate). These rates are summarized in the matrix \mathbf{M}_j :

$$\mathbf{M}_j = \begin{pmatrix} 0.73 & 0.50 \\ 0.27 & 0.50 \end{pmatrix}$$

Similarly, migration rates for adults are summarized in matrix \mathbf{M}_a :

$$\mathbf{M}_a = \begin{pmatrix} 0.50 & 0.20 \\ 0.50 & 0.80 \end{pmatrix}$$

We also define a matrix \mathbf{M} summarizing the migration rates:

$$\mathbf{M} = \begin{pmatrix} 0.73 & 0.50 & 0.00 & 0.00 \\ 0.27 & 0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.50 & 0.20 \\ 0.00 & 0.00 & 0.50 & 0.80 \end{pmatrix}$$

What is the proportion of adults migrating from population 2 to population 1? (dumb question, just to check)

We would like to modify the matrix \mathbf{B} so that it contains the migration rates. What are the two assumptions you have to make before you can start? (hints: one is about population size, and one about timing)

This task is not absolutely trivial to do mathematically because \mathbf{M} is organized by age, while \mathbf{B} is organized by site. Nevertheless, by thinking carefully, you can find what should be the elements of the matrix one by one. We just need to understand how the matrix is organized. You can visualize four compartments in \mathbf{A} :

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

In red, the sub-matrix \mathbf{A}_1 contains the contribution of the population 1 to itself, that is, the fertility (f_1) and survival (ϕ_{j1} ϕ_{a1}) rates, corrected for emigration

out of 1 (d_1). Within \mathbf{A}_1 , a_{11} is the contribution of the juvenile age class to itself, within population 1. Clearly this is zero, for juveniles do not reproduce. a_{21} is the contribution of juveniles to adults, within population 1, and is the product of juvenile survival by the proportion of non-migrating adults (not the proportion of non-migrating juveniles, because of the order of sequence of life-history events we assume). What are a_{12} and a_{22} ?

Let's now look at the rest of our matrix \mathbf{A} . The sub-matrix \mathbf{A}_2 represents the contributions of the population 2 to the population 1, that is, the fertility (f_2) and survival (ϕ_{j2} ϕ_{a2}) rates within population 2, multiplied by the rate of emigration out of 2 (d_2).

You should now be able to guess the rest and write \mathbf{A} completely, using symbols f (fertility), ϕ (survival) and d (emigration), and indexes 1 (population 1), 2 (population 2), j (juvenile) and a (adult). Do it, and then proceed to the numerical application.

During the lectures, you have seen how one can use matrix algebra, instead of biological thinking (as you just did), to efficiently obtain this meta-population matrix \mathbf{A} , using a vec-permutation matrix, \mathbf{P} .

\mathbf{P} can be defined as:

$$\mathbf{P} = \sum_{i=1}^s \sum_{j=1}^p \mathbf{E}_{ij} \otimes \mathbf{E}_{ij}^T,$$

where s and p are the dimensions of the two square matrices to be combined, \otimes is the Kronecker product (google it if you don't know it, it's nothing difficult) and where \mathbf{E}_{ij} is a matrix of dimension $s \times p$ containing a 1 on the i^{th} line of the j^{th} column, and 0 elsewhere. For instance, for $s = p = 2$, $\mathbf{E}_{1,2} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$.

Can you write a code that calculates \mathbf{P} ?

From the lecture, you know that

$$\mathbf{A} = \mathbf{P}^T \mathbf{M} \mathbf{P} \mathbf{B},$$

where \mathbf{P}^T is the transposition of \mathbf{P} . Write a code that computes \mathbf{A} . You will need to know how to multiply matrices in R, and you might want to know how to transpose matrices (although here it is useless because $\mathbf{P}^T = \mathbf{P}$).

```
# Create M, the block diagonal matrix of movement matrices
```

```
A1 #check what it looks like
```

```
##      [,1] [,2]
## [1,] 0.00 2.0
## [2,] 0.25 0.5
```

```
eigen.analysis(A1) # Check local and metapopulation growth rates and have a look at the sensi
```

```

## $lambda1
## [1] 1
##
## $stable.stage
## [1] 0.6666667 0.3333333
##
## $sensitivities
##           [,1]      [,2]
## [1,] 0.000000 0.1666667
## [2,] 1.333333 0.6666667
##
## $elasticities
##           [,1]      [,2]
## [1,] 0.0000000 0.3333333
## [2,] 0.3333333 0.3333333
##
## $repro.value
## [1] 1 4
##
## $damping.ratio
## [1] 2

lambda <- eigen.analysis(A)$lambda1 # how does the metapoulation growth rate compare to loc

## Sensitivity & elasticity of metapoulation growth rate to demography and dispersal

Asens1<-eigen.analysis(A1)$sensitivities # the A sensitivity matrix
Asens2<-eigen.analysis(A2)$sensitivities # the A sensitivity matrix

## Error in as.matrix(x):  object 'A2' not found

Asens3<-eigen.analysis(A3)$sensitivities # the A sensitivity matrix

## Error in as.matrix(x):  object 'A3' not found

Asens4<-eigen.analysis(A4)$sensitivities # the A sensitivity matrix

## Error in as.matrix(x):  object 'A4' not found

Bsens1<-t(P) %*% t(M) %*% P %*% Asens1 # demography sensitivities

## Error in t(P) %*% t(M) %*% P %*% Asens1:  non-conformable arguments

Bsens2<-Asens2 %*% t(P) %*% t(M) %*% P # demography sensitivities

## Error in eval(expr, envir, enclos):  object 'Asens2' not found

Bsens3<-t(P) %*% t(M) %*% Asens3 %*% P # demography sensitivities

```



```

## Error in eval(expr, envir, enclos): object 'Asens3' not found
Bsens4<-t(P) %*% Asens4 %*% t(M) %*% P # demography sensitivities
## Error in eval(expr, envir, enclos): object 'Asens4' not found
Belas<-B * Bsens / lambda # demography elasticities
## Error in eval(expr, envir, enclos): object 'Bsens' not found
sum(Belas[1:2,1:2]) # contribution from patch 1 to metapopulation growth rate
## Error in eval(expr, envir, enclos): object 'Belas' not found
sum(Belas[3:4,3:4]) # contribution from patch 2 to metapopulation growth rate
## Error in eval(expr, envir, enclos): object 'Belas' not found
Msens=P %*% t(B) %*% Asens %*% t(P) # movement sensitivities
## Error in eval(expr, envir, enclos): object 'Asens' not found
# You'll realize that interpreting Movement sensitivities is rather difficult. A better approach
-1 * Msens[1,1] + 1 * Msens[1,2] + 1 * Msens[2,1] - 1 * Msens[2,2]
## Error in eval(expr, envir, enclos): object 'Msens' not found
#####

# Gull example (Lebreton, 1966)
rm(list=ls(all=TRUE))

B1<-matrix(c(0,0.096,0.160,0.224,0.320,0.80,0,0,0,0,0,0.82,0,0,0,0,0,0.82,0.82),5,5)
B2<-matrix(c(0,0.1,0.160,0.20,0.20,0.80,0,0,0,0,0,0.82,0,0,0,0,0,0.82,0.82),5,5)
M1<-matrix(c(0.75,0.375,0.25,0.625),2,2,byrow=T) # for the first age class
M5<-M4<-M3<-M2<-matrix(c(1,0,0,1),2,2,byrow=T) # for the rest

# This time, dispersal takes place before demography, and assume that population vector is c
# Using a similar approach to the Gull example above, try to estimate the relative contribu

```

```
#####

s=nrow(B1) # number of stages

p=nrow(M1) # number of patches

# Create the block diagonal matrix of demographic projection matrices

B<-matrix(0,p*s,p*s)

B[1:5,1:5]<-B1

B[6:10,6:10]<-B2

B #check

##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  0.0 0.096 0.16 0.224 0.32  0.0 0.00 0.00 0.00  0.00
## [2,]  0.8 0.000 0.00 0.000 0.00  0.0 0.00 0.00 0.00  0.00
## [3,]  0.0 0.820 0.00 0.000 0.00  0.0 0.00 0.00 0.00  0.00
## [4,]  0.0 0.000 0.82 0.000 0.00  0.0 0.00 0.00 0.00  0.00
## [5,]  0.0 0.000 0.00 0.820 0.82  0.0 0.00 0.00 0.00  0.00
## [6,]  0.0 0.000 0.00 0.000 0.00  0.0 0.10 0.16 0.20  0.20
## [7,]  0.0 0.000 0.00 0.000 0.00  0.8 0.00 0.00 0.00  0.00
## [8,]  0.0 0.000 0.00 0.000 0.00  0.0 0.82 0.00 0.00  0.00
## [9,]  0.0 0.000 0.00 0.000 0.00  0.0 0.00 0.82 0.00  0.00
## [10,] 0.0 0.000 0.00 0.000 0.00  0.0 0.00 0.00 0.82  0.82

# Create M, the block diagonal matrix of movement matrices

M=matrix(0,p*s,p*s)

M[1:2,1:2]<-M1

M[3:4,3:4]<-M2
M[5:6,5:6]<-M3
M[7:8,7:8]<-M4
M[9:10,9:10]<-M5

M #check

##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] 0.75 0.375  0  0  0  0  0  0  0  0
## [2,] 0.25 0.625  0  0  0  0  0  0  0  0
## [3,] 0.00 0.000  1  0  0  0  0  0  0  0
```

```

## [4,] 0.00 0.000 0 1 0 0 0 0 0 0
## [5,] 0.00 0.000 0 0 1 0 0 0 0 0
## [6,] 0.00 0.000 0 0 0 1 0 0 0 0
## [7,] 0.00 0.000 0 0 0 0 1 0 0 0
## [8,] 0.00 0.000 0 0 0 0 0 1 0 0
## [9,] 0.00 0.000 0 0 0 0 0 0 1 0
## [10,] 0.00 0.000 0 0 0 0 0 0 0 1

# create P, the vec-permutation matrix

P<-matrix(0,s*p,s*p)
for (i in 1:s){
  for (j in 1:p){
    E=matrix(0,s,p)
    E[i,j]<-1
    P<-P+E%x%t(E) # %x% is the Kronecker product
  }}

# create the Metapopulation Projection Matrix, A

A<- P %*% B %*% t(P) %*% M
A #check what it looks like

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] 0.0 0.0 0.096 0.00 0.16 0.00 0.224 0.00 0.32 0.00
## [2,] 0.0 0.0 0.000 0.10 0.00 0.16 0.000 0.20 0.00 0.20
## [3,] 0.6 0.3 0.000 0.00 0.00 0.00 0.000 0.00 0.00 0.00
## [4,] 0.2 0.5 0.000 0.00 0.00 0.00 0.000 0.00 0.00 0.00
## [5,] 0.0 0.0 0.820 0.00 0.00 0.00 0.000 0.00 0.00 0.00
## [6,] 0.0 0.0 0.000 0.82 0.00 0.00 0.000 0.00 0.00 0.00
## [7,] 0.0 0.0 0.000 0.00 0.82 0.00 0.000 0.00 0.00 0.00
## [8,] 0.0 0.0 0.000 0.00 0.00 0.82 0.000 0.00 0.00 0.00
## [9,] 0.0 0.0 0.000 0.00 0.00 0.00 0.820 0.00 0.82 0.00
## [10,] 0.0 0.0 0.000 0.00 0.00 0.00 0.000 0.82 0.00 0.82

eigen.analysis(A) # Check local and metapoulation growth rates an have a look at the sensitivities

## $lambda1
## [1] 0.9970859
##
## $stable.stage
## [1] 0.12952855 0.05171024 0.09350268 0.05191211 0.07689628
## [6] 0.04269234 0.06323924 0.03511004 0.29283071 0.16257780
##
## $sensitivities

```

```

##          [,1]      [,2]      [,3]      [,4]
## [1,] 0.00000000 0.00000000 0.07042373 0.00000000
## [2,] 0.00000000 0.00000000 0.00000000 0.03256113
## [3,] 0.13514559 0.05395267 0.00000000 0.00000000
## [4,] 0.08092901 0.03230839 0.00000000 0.00000000
## [5,] 0.00000000 0.00000000 0.11038107 0.00000000
## [6,] 0.00000000 0.00000000 0.00000000 0.03546815
## [7,] 0.00000000 0.00000000 0.00000000 0.00000000
## [8,] 0.00000000 0.00000000 0.00000000 0.00000000
## [9,] 0.00000000 0.00000000 0.00000000 0.00000000
## [10,] 0.00000000 0.00000000 0.00000000 0.00000000
##          [,5]      [,6]      [,7]      [,8]      [,9]
## [1,] 0.05791623 0.00000000 0.04763011 0.00000000 0.2205523
## [2,] 0.00000000 0.02677817 0.00000000 0.02202227 0.00000000
## [3,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [4,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [5,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [6,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [7,] 0.09908034 0.00000000 0.00000000 0.00000000 0.00000000
## [8,] 0.00000000 0.03024314 0.00000000 0.00000000 0.00000000
## [9,] 0.00000000 0.00000000 0.08606919 0.00000000 0.3985453
## [10,] 0.00000000 0.00000000 0.00000000 0.02487186 0.00000000
##          [,10]
## [1,] 0.00000000
## [2,] 0.1019746
## [3,] 0.00000000
## [4,] 0.00000000
## [5,] 0.00000000
## [6,] 0.00000000
## [7,] 0.00000000
## [8,] 0.00000000
## [9,] 0.00000000
## [10,] 0.1151697
##
## $elasticities
##          [,1]      [,2]      [,3]      [,4]
## [1,] 0.00000000 0.00000000 0.006780437 0.00000000
## [2,] 0.00000000 0.00000000 0.000000000 0.00326563
## [3,] 0.08132434 0.01623311 0.000000000 0.00000000
## [4,] 0.01623311 0.01620141 0.000000000 0.00000000
## [5,] 0.00000000 0.00000000 0.090777014 0.00000000
## [6,] 0.00000000 0.00000000 0.000000000 0.02916889
## [7,] 0.00000000 0.00000000 0.000000000 0.00000000
## [8,] 0.00000000 0.00000000 0.000000000 0.00000000
## [9,] 0.00000000 0.00000000 0.000000000 0.00000000

```

```

## [10,] 0.00000000 0.00000000 0.00000000 0.00000000
##          [,5]          [,6]          [,7]          [,8]
## [1,] 0.00929368 0.00000000 0.01070033 0.00000000
## [2,] 0.00000000 0.004297029 0.00000000 0.004417327
## [3,] 0.00000000 0.00000000 0.00000000 0.00000000
## [4,] 0.00000000 0.00000000 0.00000000 0.00000000
## [5,] 0.00000000 0.00000000 0.00000000 0.00000000
## [6,] 0.00000000 0.00000000 0.00000000 0.00000000
## [7,] 0.08148333 0.00000000 0.00000000 0.00000000
## [8,] 0.00000000 0.024871858 0.00000000 0.00000000
## [9,] 0.00000000 0.00000000 0.07078301 0.00000000
## [10,] 0.00000000 0.00000000 0.00000000 0.020454531
##          [,9]          [,10]
## [1,] 0.07078301 0.00000000
## [2,] 0.00000000 0.02045453
## [3,] 0.00000000 0.00000000
## [4,] 0.00000000 0.00000000
## [5,] 0.00000000 0.00000000
## [6,] 0.00000000 0.00000000
## [7,] 0.00000000 0.00000000
## [8,] 0.00000000 0.00000000
## [9,] 0.32776228 0.00000000
## [10,] 0.00000000 0.09471516
##
## $repro.value
## [1] 1.0000000 0.8327907 1.3852923 0.8295523 1.5673846
## [6] 0.9071412 1.7107525 0.9405502 1.8070331 0.9405502
##
## $damping.ratio
## [1] 1.117914

lambda <- eigen.analysis(A)$lambda1

## A sensitivity & elasticity

analysis<-eigen.analysis(A)

Asens<-analysis$sensitivities # the A sensitivity matrix

Bsens<-t(P) %*% Asens %*% t(M) %*% P # demography sensitivities

Belas<-B * Bsens / lambda # demography elasticities

sum(Belas[1:5,1:5]) # contribution from patch 1 to metapopulation growth rate
## [1] 0.7659205

```

```

sum(Belas[6:10,6:10]) # contribution from patch 2 to metapopulation growth rate
## [1] 0.2340795

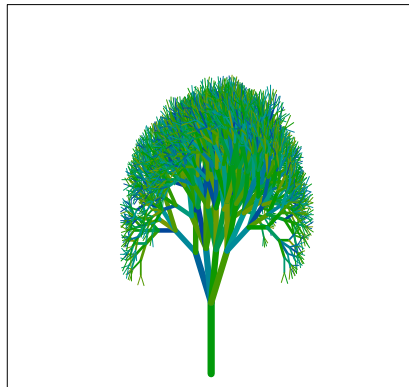
Msens=P %*% t(B) %*% t(P) %*% Asens # movement sensitivities

Melas=M * Msens / lambda # movement elasticities

```

This is where the practical ends. If it ends a bit too soon for you and you would like to continue fighting R, think about a model that combines both types of stochasticity: environmental and demographic. Alternatively, do something less biologically relevant, but quite beautiful programming wise: understand (or write) a recursive function. This is a function that calls itself, an example that draws a tree is given below. Finally, you could of course also consider doing a project in our research group. (These options are not mutually exclusive).

Ciao!



```

tree<-function(x,y,l,dir,n,nmax){ # We define a function 'tree()'
  # x,y: start of tree, l: length of first branch, dir: direction
  # n and nmax: should be the same number: number of levels in the
  # tree.

  # An important escape argument, leave this out and the function
  # will run forever:
  if(n==0){
    return()
  }

  # Picking a colour at random using the rainbow function

```

```

# (without this the code would also work, there would just be
# no colors)
pos<-round(runif(1,1,199))
colour=rainbow(200,start=0.2,end=0.6,v=0.6)[pos]

# Draw a line starting a (x,y) with length 'l' and in direction
# (in radials) dir. On top of that, we make the width of the
# line depend on how far the branch is from the stem.
lines(c(x,x+l*sin(dir)),c(y,y+l*cos(dir)),
      lwd=20*(n/nmax),col=colour)

# Generate a random number that defines how many branches the
# tree has at this point in the structure
branches<-round(runif(1,2,4))

# Now we go over the separate branches
for(i in 1:branches){
  # to make sure not all branches point in the same direction,
  # we calculate a direction for the branch
  angle<-dir+(-pi/6)+(pi/3)*(i-1)/(branches-1)

  # Also, we would like the later branches to be smaller than
  # the first one:
  l2<-runif(1,0.7,0.85)*l

  # And finally the magic of recursion, we draw the new branch
  # simply by using the exact same function: the function 'tree'.
  tree(x+l*sin(dir),y+l*cos(dir),l2,angle,n-1,nmax)
}

# Now, to actually draw the tree, we first make an empty plot
plot(0,0,type="n",xlim=c(-10,10),ylim=c(0,10),
     xaxt="n",yaxt="n",ylab="",xlab="")

# And then call the function tree() with the parameters we like
tree(0,0,2,0,8,8)

```