Demography_metric_calulation

Kevin Healy

8 March 2017

This is a brief introduction to the calulating demogrpahic metrics. Many fo the functions are from the following packages

```
library(popbio)
library(popdemo)

#devtools to get the Mage package
library(devtools)
##there seems to be a bug in there code to downlaod at the moment
#install_github("jonesor/compadreDB/Mage")
```

Lets get some data

```
dir <- setwd("/Users/kevinhealy/Desktop/Comadre from the big mac/Final analysis")
load("COMADRE_v.2.0.1.RData")
##these are the Mage functions mostly
#I will update this doc when the Mage package is fixed
source("COMADRE_functions.R")
```

```
## Loading required package: mvtnorm
```

```
##
## Attaching package: 'phytools'
```

```
## The following object is masked from 'package:Matrix':
##
##     expm
```

Now lets subset based on some criteria

```
##subset somthing from it
mean_Metadata <- (subset(comadre$metadata,
                         MatrixComposite == "Mean"
                         & MatrixDimension > 2
                         & StudyDuration > 2
                         & MatrixSplit == "Divided"
                         & MatrixFec == "Yes"
                         & MatrixTreatment == "Unmanipulated"
                         & AnnualPeriodicity == "1"
                         & SurvivalIssue<1.01
                          ))

###lets just pick a random matrix
meta_1 <- mean_Metadata[80,]
mat_1 <- comadre$mat[80]

#nd an example that doesnt work
meta_bad <- mean_Metadata[100,]
mat_bad <- comadre$mat[100]
```

Ok one of the first things to do is check that the metrices are well behaved. This includes checking whther they are irreducible, ergodic and primative. This is all highlighted well in this paper http://onlinelibrary.wiley.com/doi/10.1111/j.2041-210X.2010.00032.x/full. This is also were the functions to check come from.

Irreduciblity checks if a matrix contains direct or indirect pathways from every stage class to every other stage class. Post reproductive stages can result in biological meaninful matrices that are reproducable. However, otherwise a reproducable matrix doestn really make sense.

```
is.matrix_irreducible(mat_1[[1]]$matA)
```

```
## [1] TRUE
```

```
is.matrix_irreducible(mat_bad[[1]]$matA)
```

```
## [1] FALSE
```

Ergodic matrices will always show the same outcome irrespective of intial condictions. If a matrix is irreducible it is also egodic, however not all egodic matrices are irreducible.

```
is.matrix_ergodic(mat_1[[1]]$matA)
```

```
## [1] TRUE
```

```
is.matrix_ergodic(mat_bad[[1]]$matA)
```

```
## [1] FALSE
```

Primative matrices are postive square matrices (all elements are poative for matrix A when put to a power k). A sufficient condition for a matrix to be a primitive matrix is for the matrix to be a nonnegative. You dont useally need to check this as all irraducable and ergodic matrices are primative

```
is.matrix_primitive(mat_1[[1]]$matA)
```

```
## [1] TRUE
```

```
is.matrix_primitive(mat_bad[[1]]$matA)
```

```
## [1] FALSE
```

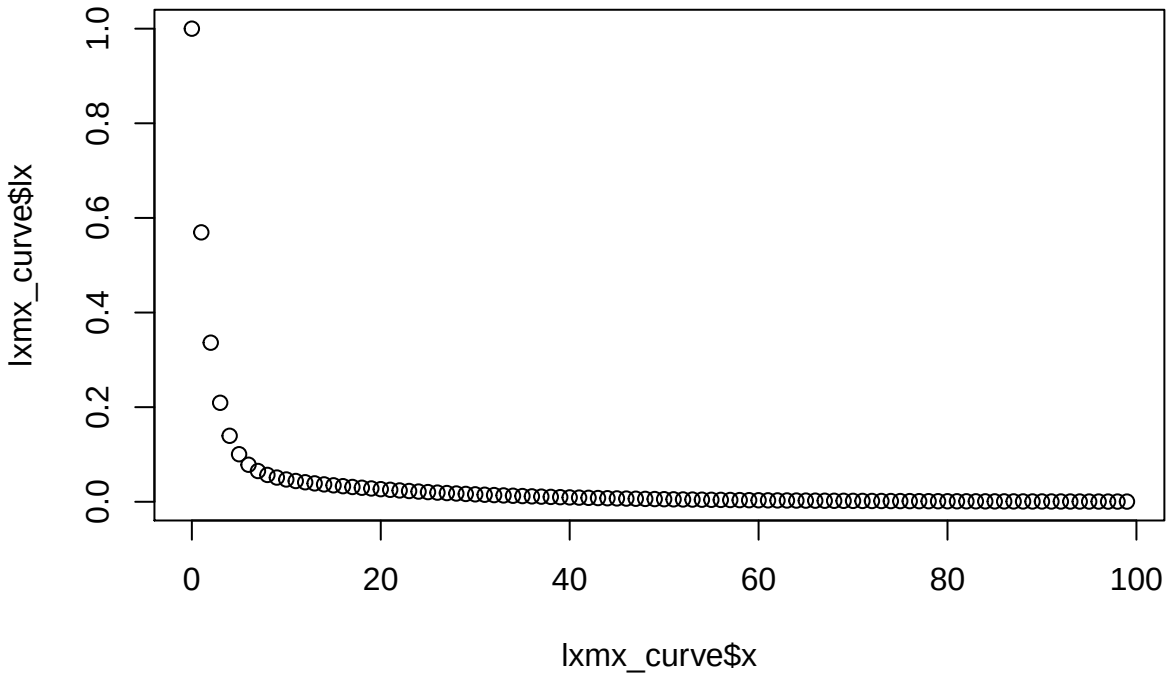
Since we know our "good" matrix is well behaved lets start calulating stuff

The First thing thats useful is the lifetable function that converts the matrix into lx (survival probabilty to time x or %still alive at time x) and mx (reporduction per capita at stage x). Since we are only using annual matrices each time step x is a year so its nice a easy to interpret

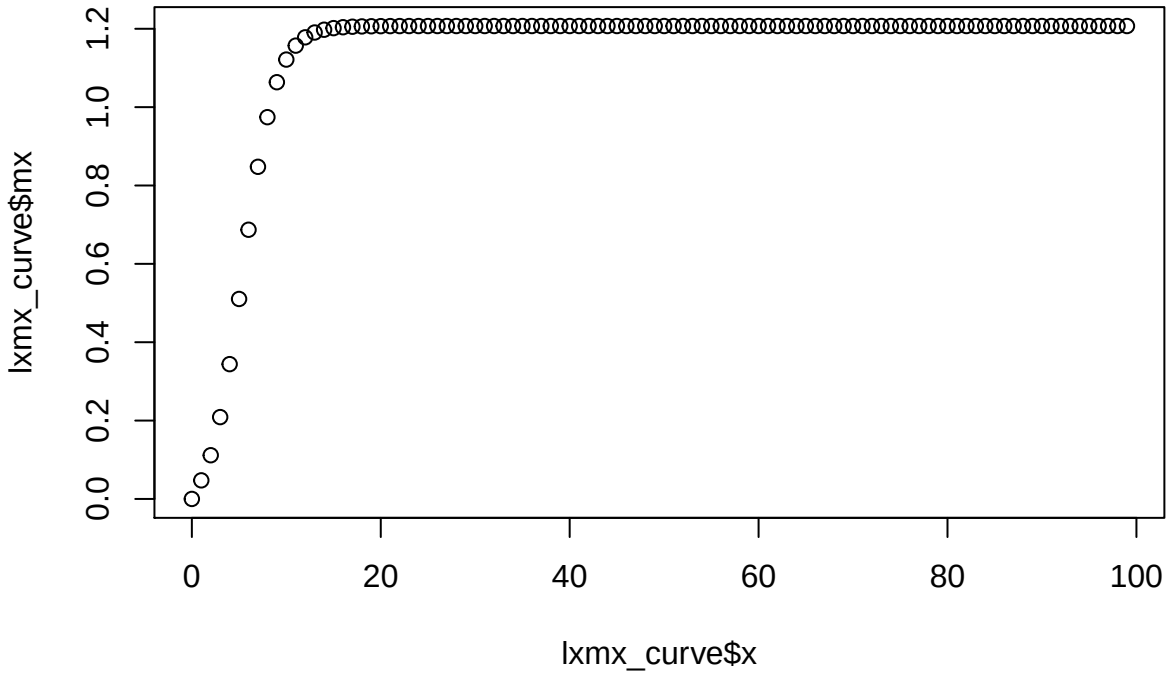
```
lxmx_curve <-  makeLifeTable(matU =  mat_1[[1]]$matU, matF =  mat_1[[1]]$matF,
                             matC =  mat_1[[1]]$matC, startLife = 1, nSteps = 100)
```

```
## Warning in makeLifeTable(matU = mat_1[[1]]$matU, matF = mat_1[[1]]$matF, :
## matC contains only 0 values
```

```
plot(lxmx_curve$lx~lxmx_curve$x)
```




```
plot(lxmx_curve$mx~lxmx_curve$x)
```



From these graphs we can calulate how long things live to easily enough by seeing how many are still alive at time point x

```
exceptionalLife<-function(matU,startLife=1){
  popVector=rep(0,dim(matU)[1])
  popVector[startLife]=100
  lifespanLeftover=matrix(0,1000,1)
  for (n in 1:1000){
    lifespanLeftover[n]=sum(popVector)
```

```
    popVector=matU%*%popVector
  }
  Lexcept.50=min(which(lifespanLeftover<50))
  if(Lexcept.50==Inf) {Lexcept.50=NA}

  Lexcept.95=min(which(lifespanLeftover<5))
  if(Lexcept.95==Inf) {Lexcept.95=NA}
  Lexcept.99=min(which(lifespanLeftover<1))
  if(Lexcept.99==Inf) {Lexcept.99=NA}

  return(list(Lexcept.50 = Lexcept.50, Lexcept.95 = Lexcept.95, Lexcept.99 = Lexcept.99))
}

perc_dead <- exceptionalLife(mat_1[[1]]$matU)

#when 50% are dead
perc_dead$Lexcept.50
```

```
## [1] 4
```

```
#when 95% are dead
perc_dead$Lexcept.95
```

```
## [1] 12
```

```
#when 99% are dead
perc_dead$Lexcept.99
```

```
## [1] 41
```

We can however also calulate the mean life expectancy using matrix magic were you calulate the the inverse of (I-U) to give matirx N. N gives somthing like the number of times visting each possible state and so by summing up the first column we get the mean life expecatncy starting at stage 1. (Caution this is my interpretation, In other words see Caswell)

```
meanLifeExpectancy
```

```
## function (matU = matU, startLife = 1)
## {
##     uDim = dim(matU)[1]
##     N = solve(diag(uDim[startLife]) - matU)
##     eta = colSums(N)[startLife]
##     return(eta)
## }
```

```
mean_life_expect_1 <-   meanLifeExpectancy(matU =  mat_1[[1]]$matU, startLife = 1)
```

We can calulate lots of other useful lifespan derived things using similar approach with the `lifeTimeRepEvents` function from the Mage package https://github.com/jonesor/compadreDB/tree/master/Mage/R

```
life_time_1 <-  lifeTimeRepEvents(matU =  mat_1[[1]]$matU,
                                  matF =  mat_1[[1]]$matF,
                                  startLife = 1)


lifeTimeRepEvents
```

```
## function (matU, matF, startLife = 1)
## {
```

```
##     uDim = dim(matU)[1]
##     surv = colSums(matU)
##     repLifeStages = colSums(matF)
##     repLifeStages[which(repLifeStages > 0)] = 1
##     if (missing(matF) | missing(matU)) {
##         stop("matU or matF missing")
##     }
##     if (sum(matF, na.rm = T) == 0) {
##         stop("matF contains only 0 values")
##     }
##     Uprime = matU
##     Uprime[, which(repLifeStages == 1)] = 0
##     Mprime = matrix(0, 2, uDim)
##     for (p in 1:uDim[1]) {
##         if (repLifeStages[p] == 1)
##             Mprime[2, p] = 1
##         else Mprime[1, p] = 1 - surv[p]
##     }
##     Bprime = Mprime %*% (ginv(diag(uDim) - Uprime))
##     pRep = Bprime[2, startLife]
##     out = data.frame(pRep = pRep)
##     D = diag(c(Bprime[2, ]))
##     Uprimecond = D %*% Uprime %*% ginv(D)
##     expTimeReprod = colSums(ginv(diag(uDim) - Uprimecond))
##     La = expTimeReprod[startLife]
##     out$La = La
##     firstRepLifeStage = min(which(repLifeStages == 1))
##     N = solve(diag(uDim[1]) - matU)
##     meanRepLifeExpectancy = colSums(N)[firstRepLifeStage]
##     out$meanRepLifeExpectancy = meanRepLifeExpectancy
##     remainingMatureLifeExpectancy = colSums(N)[startLife] - La
##     out$remainingMatureLifeExpectancy = remainingMatureLifeExpectancy
##     return(out)
## }
```

Mean life expectancy conditional on entering the life cycle is the sum of colunm for the first reproductive stage. Essentaily you find out when you reproduce and sum up the colunm from there.

```
##notice that as 100% of indaviduals make it to stage 2 its 1 year less then the mean life expectancy
life_time_1$meanRepLifeExpectancy
```

```
## [1] 3.488421
```

Probability of survival to first reprod event works by finding first col with reporduction and setting survival to zero at that stage than multpying a matrix with either 1 if reporducing and 1-survaial rate if not and multiplying that with the

```
life_time_1$La
```

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

The age at first reproduction La is not calulated as simply as above. I didnt get to check this one out as much as I would have liked so will just refer you to see Caswell 2001, p 124)

```
life_time_1$La
```

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

Age of first repoduction is just the mean life expectancy - La. Its not a great measure as it gives negative numbers which are hard to interpret and work with in a comparative sense.

```
life_time_1$remainingMatureLifeExpectancy
```

```
## [1] 2.488421
```

Before moving on a side note on QSD and Stable state distribution.

Stable state distibution is the distribution of a population across its age/classes when it reaches equalibrium. It is given by the eigenvector for the eigenvalue lambda, which in turn is the dominant eigenvalue. Its usefull for weighting things across the population basedon on where the differnt proportions of indaviduals lie.

```
SSD <- eigen.analysis(mat_1[[1]]$matA)$stable.stage
#in this case 26.3% in stage 1, 53.3% in stage 2 and 20.4 in stage 3
SSD
```

```
## [1] 0.2629547 0.5333809 0.2036644
```

You can test whether a population has converged to this using the quasi-convergence which tests how close your populations distribution is to its SSD.

```
QSD_1 <- qsdConverge(mat_1[[1]]$matU, conv = 0.05, startLife = 1, nSteps = 10000)
#this population converges after 12 years]
QSD_1
```

```
## [1] 12
```

Back to calulating stuff

Net reporductive Rate is the average number of offspring an indavidual is expected to produce over a lifespan. Using the function from popbio package it calulates the N matrix of how long you spend in each stage (I-U)^-1 and multiply this by the repordive reporductive matrix F to get R. The net reprodive rate is then given by the right eigenvalue of R.

```
net_rep_rate  <- net.reproductive.rate(A =  mat_1[[1]]$matA)

##this appraently also gives a rough estimate of net reporductive rate
sum(lxmx_curve$lx*lxmx_curve$mx)
```

```
## [1] 1.47303
```

Reproductive value and mean reporductive rate

Reproductive value is the relative reproductive potential of an indavidual at any age. Its a weighted average (using 1/lambda^x) of present and future repoduction contributions to the populations by an indavidual aged x. See this paper for a good description http://onlinelibrary.wiley.com/doi/10.2307/20168257/pdf

```
repo_value <- reproductive.value(mat_1[[1]]$matA)
repo_value
```

```
##        A1        A2        A3
##  1.000000  1.030996 14.631013
```

```
##the mean reporductive value can be also weighted using the stable state distribution
mean_repo_rate <- repo_value %*% SSD
mean_repo_rate
```

```
##          [,1]
## [1,] 3.792685
```

The mean reproductive rate is much simplier and is calulated using the F matrix more directly

```
##first sum up the reporductive rates across the F matrix
N <- length(mat_1[[1]]$matA[,1])
repo_sum <- vector()
for(j in 1:(N)){
  repo_sum[j] <- sum(mat_1[[1]]$matF[,j])
}
##then weight it against the stable state distribtion
mean_repo_rate <- repo_sum %*% SSD
```

Generation time

Basically some type of measure of how long it takes for a coort to relace itself. Turns out there is a bunch of ways to do this. I used the popbio package which calculates the time it takes for a pop to grow by a factor of R0 the net reprodictive rate (how long it would for everyone to replace themselves) and is calulated log(net_repo_rate)/log(population growth rate)

```
generation.time
```

```
## function (A, ...)
## {
##     if (!is.matrix(A)) {
##         stop("A projection matrix is required")
##     }
##     A1 <- splitA(A, ...)
##     Tmat <- A1[[1]]
##     Fmat <- A1[[2]]
##     s <- length(diag(Tmat))
##     N <- try(solve(diag(s) - Tmat), silent = TRUE)
##     if (class(N) == "try-error") {
##         generation.time <- NA
##     }
##     else {
##         R <- Fmat %*% N
##         Ro <- lambda(R)
##         lambda <- lambda(A)
##         generation.time = log(Ro)/log(lambda)
##     }
##     generation.time
## }
## <environment: namespace:popbio>
```

```
generation.time(mat_1[[1]]$matA)
```

```
## [1] 17.39415
```

```
##apparentaly this gives you generation time using tables
sum(lxmx_curve$x*lxmx_curve$lx*lxmx_curve$mx)/sum(lxmx_curve$lx*lxmx_curve$mx)
```

```
## [1] 21.26367
```

Another way to caluated it and with a new formula is the average disatance between mother and offspring with a new paper out to do it https://zenodo.org/record/49440

Progression and Retrogression

A measure of how quickly you move through either the bottem or top triangle (including the diagnol) of a matrix.

```
#length of the matrix
N <- length(mat_1[[1]]$matA[,1])
##this is to get rid of repo but not clonal retregression
A2 <-  mat_1[[1]]$matA - mat_1[[1]]$matF

#Progression
##blank out the upper trinagle
lowerA2 <- A2
lowerA2[upper.tri(lowerA2,diag = TRUE )] <- c(0)
###sum up the proggression from each stage
prog_sum <- vector()
for(j in 1:(N)){
     prog_sum[j] <- sum(lowerA2[,j])
}
##and weight it against the stable state distribution
 prog_1 <- prog_sum %*% SSD

 #Retrogression
 ##blank out the upper trinagle
 upperA2 <- A2
 upperA2[lower.tri(upperA2,diag = TRUE )] <- c(0)
 ###sum up the proggression from each stage
 retr_sum <- vector()
 for(j in 1:(N)){
   retr_sum[j] <- sum(upperA2[,j])
 }
 ##and weight it against the stable state distribution
 retr_1 <- retr_sum %*% SSD
```

H entropy

This is a measure of the shape of the life history. If a species is type 3 it has a high mortality rate with the few survivors living to late age it gives a value of >1 (1.5). If it has constant mortality it has a value of 1 (type 2) and if it has a "human" like shape were most indaviduals live until old age it has a value approaching zero. This is base on a dirvation from information theory (See Demetrius 1978 and refs therin) and so can do funny things. For example, its not monotonic across the types with increaseing type 3 curves begining to give lower values at the extremes.

```
##using the Mage function
kentropy <- function(lx, trapeze = TRUE){

  if(max(lx) > 1) stop("`lx` should be bounded between 0 and 1")
  if(sum(is.na(lx))>1) stop("There are missing values in `lx`")
  #if(sum(!diff(lx) <= 0)) stop("`lx` does not monotonically decline")
  if(sum(!diff(lx) <= 0)!=0)stop("`lx` does not monotonically decline")


  if(trapeze == TRUE){
```

```
    ma <- function(x,n=2){filter(x,rep(1/n,n), sides=2)}
    lx2 <- na.omit(as.vector(ma(lx)))
    return(-sum(lx2*log(lx2))/sum(lx2))
  }else{
    return(-sum(lx*log(lx))/sum(lx))
  }
}
H_entropy <- kentropy(lxmx_curve$lx)

##if you set trapeze == FALSE you get my formulation
H_entropy_kev <- sum(-log(lxmx_curve$lx)*lxmx_curve$lx)/sum(lxmx_curve$lx)
```