HMM McLaughlin

Marina LaForgia

8/27/2021

# Markov Chain Parameter Functions

## 1. Seedbank transitions

This sets up a 2x2 matrix of seed bank transitions - I’m a little confused on this on what these values represent/how they are being used

* (1,1): 1-c -> probability that there are seeds present in the seed bank
* (2,1): (1-c)x(1-s)x(1-gr) -> of the seeds already present in the seedbank, these did not survive, and did not germinate/produce more seeds; given that no new seeds colonize into the seedbank and some portion of the seeds germinate, this represents those that died
* (1,2): c -> colonization probability
* (2,2): 1 - ((1-c)x(1-s)x(1-gr)) -> growth rate of the seed bank? new seeds colonize in, seeds survive, …

#matrix of Y transitions (Y = seedbank because Zt is a function of the seedbank at time t and flora (x) at time t+1)  
calculateQY = function(param) # calculate QY is a function of param; QY transition matrix   
{  
 result = matrix(0,2,2) # result is a matrix of zeros with 2 columns and 2 rows  
 result[1,1] = 1 - param$c # where the first row of the first column is 1 - colonization rate aka the probability that seeds are there already  
 result[2,1] = (1 - param$c)\*(1 - param$s)\*(1 - param$g\*param$r) # and the second row of the first column is (1-c) \* (1-s) \* (1-gF) = those seeds that don't colonize, and havent survived or germinated - > is this essentially who isn't contributing to the population; probability that seeds are there,   
 result[,2] = 1 - result[,1] # column 2 is the opposite of column one so that each row sums to 1  
 return(result) # I'm still unclear on what this matrix is exactly  
}

## 2. State transition probability matrix

Matrix of transitions Zt = (Yt-1, Xt). the three states in order are (0,0), (1,0), (1,1). where

* Zt = (0,0) = both seeds and standing flora are absent (AA)
* Zt = (1,0) = seeds are present but standing flora is absent the following year (PA)
* Zt = (1,1) = seeds are present and then standing flora is present (PP)

calculateQZ = function(param) # Qz transition matrix  
{  
 result = matrix(0,3,3) # 3 by 3 matrix of 0s  
 result[1,1] = 1 - param$c  
 result[2,1] = (1 - param$c)\*(1 - param$s)  
 result[3,1] = (1 - param$c)\*(1 - param$s)\*(1 - param$r)  
 result[,2] = (1 - result[,1])\*(1 - param$g)  
 result[,3] = (1 - result[,1])\*param$g  
 return(result)  
}

## 3. Probability invariant of a transition matrix.

no idea whats going on here

calculatePeq = function(Q)  
{  
 Vmat = eigen(t(Q))$vectors  
 #We look for an eigenvector whose components have the same sign and we renormalize it so that the sum makes 1  
 n = dim(Q)[1]  
 for (i in 1:n)  
 {  
 v = Vmat[,i]  
 if ( all(sign(v)>=0) | all(sign(v)<=0) ) return(v/sum(v))  
 }  
}

## 4. Putting it all together

Add the object param these additional fields;

Still not sure what PZeq and PYeq are

makeParametersCalculations = function(param)  
{  
 param$QY = calculateQY(param) # calculate seed bank transitions  
 param$QZ = calculateQZ(param) # calculate state transition probability matrix  
 param$PZeq = calculatePeq(param$QZ) # calculate the invariant of Qz transition matrix  
 param$PYeq = param$PZeq[2] + param$PZeq[3] # what is this?  
 param$pZ0 = c(1-param$p0, param$p0\*(1-param$g), param$p0\*param$g) # initial distribution vector  
 return(param)   
}

# Expectation Maximization

Computes maximum likelihood estimators of g, c, and s transforms HMM into HMC by grouping flora(t) and seedbank(t-1) to form hidden state

## 1. Forward algorithm for one time series

Recursive scheme for updating the likelihood and state probabilities of an HMM through time

pZXprespast = function(X,param)   
{  
 tf = length(X) # time frame is length of flora  
 result = matrix(0,3,tf) # result is a matrix of 0s with three rows (each state: AA, PA, PP) and tf columns  
 for (t in 1:tf) # for each column (time step)  
 {  
 if (t == 1) pZXpast = param$pZ0 else pZXpast = t(param$QZ) %\*% result[,t-1] # if it's the first time step, use the initial distribution vector, otherwise transpose the state transition probability matrix and do matrix multiplication on the result matrix from the previous time step  
 result[,t] = pZXpast # paste it in for the current time step  
 if (X[t] == 1) result[1:2,t] = 0 else result[3,t] = 0 # if flora observed in the current time step, the result for both rows one and two are 0 (because it's observed so it cant be absent belowground) otherwise the third row is 0)  
 }  
 return(result)  
}

## 2. Backward algorithm for one time series

not exactly sure what this is, is it the same as the previous section but going backward in time?

pXfuturegivenZ = function(X,param)  
{  
 tf = length(X) # number of time steps  
 result = matrix(0,3,tf) # same matrix as before  
 result[,tf] = 1 # starting at 1  
 for (t in tf:2)   
 {  
 PZXgivenOldZ = param$QZ # take the state transition probability matrix  
 if (X[t] == 1) PZXgivenOldZ[,1:2] = 0 else PZXgivenOldZ[,3] = 0 # if flora is observed in that time step, the first two columns are 0, otherwise the 3rd column is 0  
 result[,t-1] = PZXgivenOldZ %\*% result[,t]  
 }  
 return(result)  
}

## 3. Vector for forward backward algorithm

vectForwardBackward = function(X,param) #log-likelihood and lambda coefficients for one time series;   
{  
 tf = length(X)  
 pZXpp = pZXprespast(X,param)  
 pXfgZ = pXfuturegivenZ(X,param)  
 pZX = pZXprespast(X,param)\*pXfuturegivenZ(X,param) #P(Zt,whole vector X)  
 pZZX = array(rep(param$QZ,tf-1),c(3,3,tf-1)) #P(Zt,Zt+1,whole vector X)  
 for (j in 1:3) pZZX[,j,] = pZZX[,j,]\*pZXpp[,1:(tf-1)]  
 for (i in 1:3) pZZX[i,,] = pZZX[i,,]\*pXfgZ[,2:tf]  
 pZZX[,1:2,X[2:tf] == 1] = 0  
 pZZX[,3,X[2:tf] == 0] = 0  
 likelihood = mean(apply(pZX,2,sum))  
 result = numeric(0)  
 result$ll = log(likelihood)  
 result$coeffZ = pZX[,1]/likelihood  
 result$coeffQ = apply(pZZX,1:2,sum)/likelihood  
 return(result)  
}

## 4. Forward Backward Algorithm

Recursive scheme for calculating state probabilities for any point in time??

ForwardBackward = function(X,param) #same function extended for a N\*T matrix of time series  
{  
 d = dim(X)  
 if (length(d) == 0) return(vectForwardBackward(X,param)) else  
 {  
 N = d[1]  
 result = numeric(0)  
 llvect = apply(X,1,function(Xvect) vectForwardBackward(Xvect,param)$ll) #N-vector,i\_th component is the log-likelihood of patch i  
 result$ll = sum(llvect) #sum on the N patches  
 coeffZmat = apply(X,1,function(Xvect) vectForwardBackward(Xvect,param)$coeffZ) #3\*N-matrix,i-th column is the lambda(z) vector of patch i  
 result$coeffZ = apply(coeffZmat,1,sum) #sum on the N patches  
 coeffQarray = array(apply(X,1,function(Xvect) vectForwardBackward(Xvect,param)$coeffQ),c(3,3,N)) #3\*3\*N-matrix,[,,i] is the lambda(z,z') matrix of patch i  
 result$coeffQ = apply(coeffQarray,1:2,sum) #sum on the N patches  
 return(result)  
 }  
}

## 5. Log Likelihood

returns the loglikelihood of the model from the forward backward algorithm

logLikelihood = function(X,param) ForwardBackward(X,param)$ll

## 6. Maximization function

Maximization = function(coeffQ, coeffZ, p0 = 'free', g = 'free', c = 'free', s = 'free', r = 'free')  
{  
 # takes into account cases of non-identifiability  
 if (p0 == 'free') p0 = sum(coeffZ[2:3])/sum(coeffZ)  
 if (g == 'free') g = (coeffZ[3] + sum(coeffQ[,3]))/(sum(coeffZ[2:3]) + sum(coeffQ[,2:3]))  
 if (r == 'free') c2 = sum(coeffQ[3,2:3])/sum(coeffQ[3,]) #calculation of c double prime (first step)  
 if (s == 'free') #calculation of c prime (first step)  
 {  
 c1 = sum(coeffQ[2,2:3])/sum(coeffQ[2,])  
 if (r == 'free') if(c2 < c1) r = 0  
 if (r == 0) c1 = sum(coeffQ[2:3,2:3])/sum(coeffQ[2:3,])   
 }  
 if (c == 'free') #estimation of c  
 {  
 c0 = sum(coeffQ[1,2:3])/sum(coeffQ[1,]) #calculation of c (first step)  
 if (s == 'free') if(c1 < c0) s = 0  
 if (s == 0)  
 {  
 if(r == 0) c0 = sum(coeffQ[,2:3])/sum(coeffQ) else c0 = sum(coeffQ[1:2,2:3])/sum(coeffQ[1:2,])  
 c1 = c0  
 }  
 c = c0  
 }  
 if (s == 'free') if(c1 < c) s = 0 else s = 1 - (1-c1)/(1-c) #estimation of s  
 c1 = 1- (1-c)\*(1-s)  
 if (r == 'free') if(c2 < c1) r = 0 else r = 1 - (1-c2)/(1-c1) #estimation of r  
 result = numeric(0)  
 result$p0 = p0  
 result$g = g  
 result$c = c  
 result$s = s  
 result$r = r  
 return(result)  
}

## 7. Expected Maximization iteration

EMiteration = function(X, param, p0 = 'free', g = 'free', c = 'free', s = 'free', r = 'free')  
{  
 FBresult = ForwardBackward(X,param)  
 coeffZ = FBresult$coeffZ  
 coeffQ = FBresult$coeffQ  
 newparam = Maximization(coeffQ,coeffZ,p0,g,c,s,r)  
 newparam = makeParametersCalculations(newparam)  
 EMresult = numeric(0)  
 EMresult$ll = FBresult$ll  
 EMresult$newparam = newparam  
 return(EMresult)  
}

## 8. Expected maximization estimation

EMestimation = function(X, nIterations = 100, precision = 10^(-5), p0 = 'free', g = 'free', c = 'free', s = 'free', r = 'free')  
{  
 alwaysSeedSurvival = FALSE  
 if (s == 1 | c == 1 | (g == 1 & r == 1))  
 {  
 alwaysSeedSurvival = TRUE  
 if (r == 'free') print('r non identifiable')  
 r = 1  
 if (s == 'free') print('s non identifiable')  
 s = 1  
 }  
 if (p0 == 1 & alwaysSeedSurvival)  
 {  
 if (c == 'free') print('c non identifiable')  
 c = 1  
 }  
 if (g == 1)  
 {  
 if (s == 'free' & r == 'free') print('r non identifiable from s')  
 r = 0  
 }  
 lllist = rep(0,nIterations)  
 p0list = rep(0,nIterations)  
 glist = rep(0,nIterations)  
 clist = rep(0,nIterations)  
 slist = rep(0,nIterations)  
 rlist = rep(0,nIterations)  
 param = numeric(0)  
 if (p0 == 'free') param$p0 = runif(1) else param$p0 = p0  
 if (g == 'free') param$g = runif(1) else param$g = g  
 if (c == 'free') param$c = runif(1) else param$c = c  
 if (s == 'free') param$s = runif(1) else param$s = s  
 if (r == 'free') param$r = runif(1) else param$r = r  
 param = makeParametersCalculations(param)  
 oldLogLikelihood = -Inf  
 for (k in 1:nIterations)  
 {  
 print(t(param[c('p0','g','c','s','r')]))  
 EMresult = EMiteration(X,param,p0,g,c,s,r)  
 logLikelihood = EMresult$ll  
 lllist[k] = logLikelihood  
 p0list[k] = param$p0  
 glist[k] = param$g  
 clist[k] = param$c  
 slist[k] = param$s  
 rlist[k] = param$r  
 print(logLikelihood)  
 if(logLikelihood < oldLogLikelihood + precision) break  
 print(k)  
 oldLogLikelihood = logLikelihood  
 param = EMresult$newparam  
 }  
 EMresult = numeric(0)  
 EMresult$lllist = lllist  
 EMresult$p0list = p0list  
 EMresult$glist = glist  
 EMresult$clist = clist  
 EMresult$slist = slist  
 EMresult$rlist = rlist  
 EMresult$ll = logLikelihood  
 EMresult$param = param  
 return(EMresult)  
}

# Apply to McLaughlin 80 Site dataset

* Do I need to change the trueParams for each species? or are these just starting values?
* Why is n 5 and why print it out 5 times?
* Why print(logLikelihood(X, trueParam)) twice? this is the loglikelihood of the data given the true params
* It’s iterative in that it is going through the EM process of estimating these parameters given all the algorithms specified above, then updating the loglikelihood and comparing to the previous model, once it reaches a certain cutoff, it stops (hopefully before it gets to 100), then is starts over (it does this 5 times, for each n? k?) not from the previous estimates but from what?

# do I need to change these for my species?  
trueParam = numeric(0) #object grouping together the parameters and other quantities which depend on them.  
trueParam$c = 0.2 #colonization rate  
trueParam$g = 0.5 #germination rate (sigma)  
trueParam$r = 1 #reproductive rate (phi)  
trueParam$s = 0.5 #seed survival  
trueParam$p0 = 0.5 #initial state of the seed bank (probability that there were seeds in the soil the year before the first obs of existing flora)  
  
trueParam = makeParametersCalculations(trueParam)  
  
species.list <- readRDS("McL\_Species-List-for-HMM\_Site.RDS") # list; each object is a dataframe of PA data for a single species where each row is a site and each column is a year  
  
mcl.df <- expand.grid(Species\_Name = names(species.list), p0 = NA, g = NA, c = NA, s = NA, r = NA) # empty df to fill with rates  
  
# Notes  
# Site version (the version specified here) took about 20 minutes or so  
# Also tried it on individual quads instead of sites, took about 12 hours  
  
for(j in names(species.list)){ # for each species  
 X = as.matrix(species.list[[j]]) # use their time series PA data  
 n = 5 # why n = 5?  
 p0Results = rep(0,n)  
 gResults = rep(0,n)  
 cResults = rep(0,n)  
 sResults = rep(0,n)  
 rResults = rep(0,n)  
 llResults = rep(0,n)  
 lltrueParam = rep(0,n)  
   
 for (i in 1:n) {  
 for (k in 1:5) print(i) # I don't understand what's going on here, why print it 5 times?  
 print(logLikelihood(X, trueParam)) # print the log-likelihood given the "true params"  
 EMresult = EMestimation(X, r = 1) # update the model params, stop when new log likelihood is lower than the old log likelihood plus some precision   
 print(logLikelihood(X, trueParam)) # print the log-likelihood given the "true params"; why is this on here twice?  
 # fill in results with new params from the best log likelihood model  
 p0Results[i] = EMresult$param$p0 # initial seed bank prob  
 gResults[i] = EMresult$param$g # germ  
 cResults[i] = EMresult$param$c #col  
 sResults[i] = EMresult$param$s #surv  
 rResults[i] = EMresult$param$r #prod  
 llResults[i] = EMresult$ll # log-likelihood  
 lltrueParam[i] = logLikelihood(X,trueParam) # log likelihood of x given "true" parameters  
 }  
  
 # here I assumed that the last estimated parameters, were what the model converged on but I'm not sure that's correct?  
 mcl.df[mcl.df$Species\_Name == j,]$p0 <- EMresult$param$p0  
 mcl.df[mcl.df$Species\_Name == j,]$g <- EMresult$param$g  
 mcl.df[mcl.df$Species\_Name == j,]$c <- EMresult$param$c  
 mcl.df[mcl.df$Species\_Name == j,]$s <- EMresult$param$s  
 mcl.df[mcl.df$Species\_Name == j,]$r <- EMresult$param$r  
   
}

Look at correlation between s and c!

