Ecological Distance in Spatial Capture-Recapture Models: Supplemental files

J. Andrew Royle
USGS Patuxent Wildlife Research Center, Laurel MD

Richard B. Chandler
USGS Patuxent Wildlife Research Center, Laurel MD

Kimberly D. Gazenski
USGS Patuxent Wildlife Research Center, Laurel MD

Tabitha A. Graves
Northern Arizona University, Flagstaff AZ

October 24, 2012

Supplement 1: R code for computing least-cost path distance and likelihood analysis of the SCR model

As an example of the cost-weighted distance calculation consider the following landscape comprised of 16 pixels with unit spacing identified as follows, along with the pixel-specific cost:

```
  pixel ID                Cost
 4 8 12 16            100   1   1  1
 3 7 11 15            100 100   1  1
 2 6 10 14            100 100 100  1
 1 5  9 13            100 100   1  1
```

Then we assigned low cost of 1 to "good habitat" pixels (or pixels we think of as "highly connected" by virtue of being in good habitat) and, conversely, we assign high cost (100) to "bad habitat". So the shortest cost-weighted distance between pixels 5 and 9 in this example is just 1 unit, the shortest cost-distance between pixels 5 and 10 is $\sqrt{2}(1+1)/2 = 1.414214$ units, the shortest distance between pixels 4 and 8 is 100 units, while the shortest cost-distance between 4 and 12 is 150.5. A tough one is: what is the shortest distance between 7 and 16? An individual at pixel 7 can move diagonal and pay $sqrt(2) * (100+1)/2 + 1 = 72.41778$. This simple cost raster is shown in Fig. 1.

The **R** commands to create a raster containing the pixel-specific costs are given as follows for a simple 4×4 raster containing values either $z = 1$ (good habitat) or $z = 100$ (bad habitat):

```
library(raster)
r<-raster(nrows=4,ncols=4)
projection(r)<- "+proj=utm +zone=12 +datum=WGS84" #sets the projection
#We use UTM here because distances and directions are correct
#i.e., they are adjusted for the earth's curvature
extent(r)<-c(.5,4.5,.5,4.5) #sets the extent of the raster
costs1<- c(100,100,100,100,1,100,100,100,1,1,100,1,1,1,1,1)
values(r)<-matrix(costs1,4,4,byrow=FALSE) #assign the costs to the raster
```

Once the cost raster is created, the least-cost path distances are computed with just a couple **R** commands, and those can be inserted directly into the likelihood construction for an ordinary spatial capture-recapture model (Appendix 1). The **R** package `gdistance` uses the implementation of Dijkstra's algorithm (**?**) found in the `igraph` package (**?**). Using

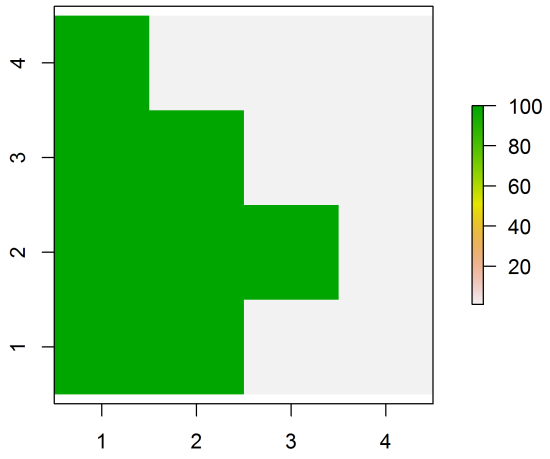

Fig. 1: A 4×4 raster with cost $= 1$ (white) or 100 (shaded) to represent ease of movement across a pixel.

`gdistance`, we define the incremental cost of moving from one pixel to another as the distance-weighted *average* of the 2 pixel costs.

To compute the least-cost path, or the minimum cost-weighted distances between every pixel and every other pixel, we make use of the helper functions `transition`, which calculates the cost of moving between neighboring pixels, and `geoCorrection` which modifies the costs of moving diagonally by the additional distance, produce output which feeds into the function `costDistance` to compute the pair-wise distance matrix. For that, we define the center points of each raster. The commands altogether are as follows:

```
library(gdistance)
tr1<-transition(r,transitionFunction=function(x) 1/mean(x),directions=8)

#The geoCorrection function corrects the conductances for the diagonal neighbors
#and, if the raster is in latitude longitude, also corrects for
#curvature of the earth's surface

tr1CorrC<-geoCorrection(tr1,type="c",multpl=FALSE,scl=FALSE)

#here we specify the locations that we'd like to calculate distances among
```

```
#
pts<-cbind( sort(rep(1:4,4)),rep(1:4,4))

#the costDistance function calculates the least cost distane among pts using
#the conductances specified in tri1CorrC
costs1<-costDistance(tr1CorrC,pts)
#here we convert costs1 into a matrix
outD<-as.matrix(costs1)
```

Now we can look at the result and see if it makes sense to us. Here we print the first 4 columns of this distance matrix and illustrate a couple of examples of calculating the minimum cost-weighted distance between points:

```
> outD[1:5,1:5]
          1        2        3        4        5
1   0.0000 100.0000 200.0000 205.2426 100.0000
2 100.0000   0.0000 100.0000 200.0000 141.4214
3 200.0000 100.0000   0.0000 100.0000 126.1604
4 205.2426 200.0000 100.0000   0.0000 105.2426
5 100.0000 141.4214 126.1604 105.2426   0.0000
```

So, to calculate the distance from the first cell in the lower left corner to the cell above it, the least cost distance is just the mean of the two cells because that is both the shortest line and all cells surrounding the first the cost in the first cell is 100 and the cost in the cell immediately above it is also 100. So the distance is (100+100)/2 = 100, which is the same as the mean of the 2 pixels.

```
 plot(r)
 points(pts[1,1],pts[1,2],col="red")
 points(pts[2,1],pts[2,2],col="blue")
 lines(c(1,1),c(1,2))
```

To move from the pixel in the lower left corner to the upper left corner points 1 & 4, the shortest distance is to go directly up.

```
 points(pts[4,1],pts[4,2],col="blue")
 lines(c(1,1),c(1,4))
#but the least cost distance is to go around the outside of the high cost area.
 lines(c(1,2),c(1,1)) #with cost = (100 + 100)/2 = 100
```

```
 lines(c(2,3),c(1,1)) #with cost = (100 + 1)/2 = 50.5
#To account for the increased distance along the diagonal, we multiply by sqrt(2)
 lines(c(3,4),c(1,2)) #with cost = sqrt(2) * (1+1)/2 = 1.414214
 lines(c(4,3),c(2,3)) #with cost = sqrt(2) * (1+1)/2 = 1.414214
 lines(c(3,2),c(3,4)) #with cost = sqrt(2) * (1+1)/2 = 1.414214
 lines(c(2,1),c(4,4)) #with cost = (100 + 1)/2 = 50.5
 100+50.5* 2+(1.414214)* 3 # = 205.2426
```

This matches the distance in our distance matrix between points 1 and 4.

Supplement 2: R code for computing the marginal likelihood and obtaining the MLEs

```
### {\bf R} Code
########################################################################
#### Define the likelihood function.
####

intlik3ed<-function(start=NULL,y=y,K=NULL,X=traplocs,distmet="ecol",
    covariate,alpha2=NA){

#start is the starting values for the parameters to be estimated
#y is the data matrix
#K is the number of occasions
#X is the traplocation matrix  with
#   (one column for x-coords one column for y-coords)
#distmet is either "ecol" or "euclid" for least cost or Euclidean distance

#First, some input checks
if(is.null(K)) return("need sample size")
if(class(covariate)!="RasterLayer") {
 cat("make a raster out of this",fill=TRUE)
 return(NULL)
 }


# Build integration grid. This derives from the covariate raster
# i.e., potential values of s are the mid-point of each raster pixel
nc<-covariate@ncols  #number of columns in the covariate raster
nr<-covariate@nrows  #number of rows in the covariate raster
Xl<-covariate@extent@xmin  #minimum X of the covariate raster
Xu<-covariate@extent@xmax  #maximum X of the covariate raster
Yl<-covariate@extent@ymin  #minimum Y of the covariate raster
Yu<-covariate@extent@ymax  #maximum Y of the covariate raster
SSarea<- (Xu-Xl)*(Yu-Yl)   #Total area of the area over estimating density


###Create potential activity centers evenly distributed across the area
delta<- (Xu-Xl)/nc  #identify offset between locations
xg<-seq(Xl+delta/2,Xu-delta/2,delta)
yg<-seq(Yl+delta/2,Yu-delta/2,delta)
npix.x<-length(xg)
npix.y<-length(yg)
area<- (Xu-Xl)*(Yu-Yl)/((npix.x)*(npix.y))  # area of pixels

G<-cbind(rep(xg,npix.y),sort(rep(yg,npix.x)))
nG<-nrow(G) #total number of potential activity centers
ntraps<- nrow(X)

if(distmet=="euclid")  #if distance metric is Euclidean,
D<- e2dist(X,G)        #calculate distance matrix among traplocs and integration grid
```

```
143
144 if(distmet=="ecol"){   #if distance metric is Ecological distance
145    if(is.na(alpha2)) alpha2<-exp(start[4]) # if estimating alpha2, use
146                                            #  this starting value
147   #the next series of commands calculates the ecological distance matrix
148   cost<- exp(alpha2*covariate)  #create resistance surface
149   #find neighbors and calculate conductances among neighbors
150   tr1<-transition(cost,transitionFunction=function(x) 1/mean(x),directions=8)
151   tr1CorrC<-geoCorrection(tr1,type="c",multpl=FALSE,scl=FALSE) #adjust diag.conductances
152   D<-costDistance(tr1CorrC,X,G)  #calculate the ecological distance matrix
153   }
154
155 if(is.null(start)) start<-c(0,0,0,0) #if no starting values were given, assign as 0s
156 alpha0<-start[1]
157 alpha1<-start[2]
158 n0<-exp(start[3])
159
160 #calculate the capture probability at each trap location for each potential
161 #activity center, equivalent to each location in the integration grid
162
163 probcap<- (exp(alpha0)/(1+exp(alpha0)))*exp(-alpha1*D*D)
164
165 #create integrand matrix (# traplocations X #potential activity centers)
166
167 Pm<-matrix(NA,nrow=ntraps,ncol=nG)
168 ymat<-y
169
170 #Add a zero capture record to the capture history.
171 #We weight this last record later to get the total
172 #estimated number of individuals
173
174 ymat<-rbind(y,rep(0,ncol(y)))
175 lik.marg<-rep(NA,nrow(ymat)) #create vector to store the marginal likelihood
176
177 #calculate the likelihood
178 #loop through each individual (plus the all zero-capture record) in capture history
179
180 for(i in 1:nrow(ymat)){
181 #calculate the probability of capturing individual i the recorded #of times
182 #in each trap (rows) when there are K occasions
183 #at each potential activity center (columns).
184 #These are logged for numerical stability (so very small probabilities recorded)
185
186 Pm[1:length(Pm)]<- (dbinom(rep(ymat[i,],nG),rep(K,nG),probcap[1:length(Pm)],log=TRUE))
187
188 #conditional on s likelihood (for individual i for each potential activity center)
189 #exponentiate to remove log above now that the numbers are larger
190
191 lik.cond<- exp(colSums(Pm,na.rm=TRUE) )
192
193 #the marginal likelihood is formed by explicitly evaluating the integrand
194 #on our integration grid, averaging over the conditional likelihood
195 #at all potential activity centers as described above
196 lik.marg[i]<- sum( lik.cond*(1/nG) )
197 }
198
199 #define weights = 1 for all individuals we captured,
200 #and estimate weight for the number of individuals that were not captured =n0
201
202 nv<-c(rep(1,length(lik.marg)-1),n0)
203 #calculate combinatorial term to account for (N choose n) ways to
204 #realize sample of size n
205 part1<- lgamma(nrow(y)+n0+1) - lgamma(n0+1) #combinatorial term
206
207 ###Multiply by weight vector(nv) and sum resulting marginal likelihoods for all individuals
208 part2<- sum(nv*log(lik.marg))
209 out<-  -1*(part1+ part2)  #final negative log likelihood
210 attr(out,"SSarea")<- SSarea   #save an attribute that has just the
211                               #  area of the integration grid
212 out   # the value of the neg log likelihood
213 }
```

Supplement 3: R code for simulation study

```
### We require 3 R libraries
library("shapefiles")
library("gdistance")
library("raster")

###Before we get started, we'll run some utility functions
## UTILITY FUNCTIONS
##
## PUT ALL OF THE OBJECTS BELOW INTO YOUR WORKSPACE
##

####This function creates a color scale used with the function image
image.scale <-
function (z, col, x, y = NULL, size = NULL, digits = 2, labels = c("breaks",
    "ranges"))
{
    # sort out the location
    n <- length(col)
    usr <- par("usr")
    mx <- mean(usr[1:2]); my <- mean(usr[3:4])
    dx <- diff(usr[1:2]); dy <- diff(usr[3:4])
    if (missing(x))
        x <- mx + 1.05*dx/2 # default x to right of image
    else if (is.list(x)) {
        if (length(x$x) == 2)
          size <- c(diff(x$x), -diff(x$y)/n)
        y <- x$y[1]
        x <- x$x[1]
    } else x <- x[1]
    if (is.null(size))
        if (is.null(y)) {
          size <- 0.618*dy/n # default size, golden ratio
          y <- my + 0.618*dy/2 # default y to give centred scale
        } else size <- (y-my)*2/n
    if (length(size)==1)
        size <- rep(size, 2) # default square boxes
    if (is.null(y))
        y <- my + n*size[2]/2
    # draw the image scale
    i <- seq(along = col)
    rect(x, y - i * size[2], x + size[1], y - (i - 1) * size[2],
        col = rev(col), xpd = TRUE)
    # sort out the labels
    rng <- range(z, na.rm = TRUE)
    bks <- seq(from = rng[2], to = rng[1], length = n + 1)
    bks <- formatC(bks, format="f", digits=digits)
    labels <- match.arg(labels)
    if (labels == "breaks")
        ypts <- y - c(0, i) * size[2]
    else {
        bks <- paste(bks[-1], bks[-(n+1)], sep = " - ")
        ypts <- y - (i - 0.5) * size[2]
    }
    text(x = x + 1.2 * size[1], y = ypts, labels = bks, adj =
        ifelse(size[1]>0, 0, 1), xpd = TRUE)
}

####This function makes a nice 3D plot
spatial.plot <-
function(x,y,add=TRUE,cx=1){
 nc<-as.numeric(cut(y,20))
if(!add) plot(x,pch=" ")
 points(x,pch=20,col=terrain.colors(20)[nc],cex=cx)
image.scale(y,col=terrain.colors(20))
}

####This function calculates geographic distances among two sets of locations
####e.g., will work for traplocs and integration grid instead of just among traplocs
e2dist <- function (x, y) {
    i <- sort(rep(1:nrow(y), nrow(x)))
    dvec <- sqrt((x[, 1] - y[i, 1])^2 + (x[, 2] - y[i, 2])^2)
    matrix(dvec, nrow = nrow(x), ncol = nrow(y), byrow = F)
```

```
    }
####This function calculates summary statistics for an input x
smy.fn <- function(x){
    c(mean(x),sqrt(var(x)),quantile(x,c(0.025,0.50,0.975)))
    }

#########################################################################
####CREATE COVARIATES
### Now we create the covariates that we will use in the simulation.
### The following block of code creates a "patchy" looking covariate to use
### in our cost function. It uses a standard method for generating a correlated
### multivariate normal vector of length, in this case, 400 (one value for each
### pixel). One can use any correlation function here but we chose a standard
### exponential model with range parameter 0.5.
par(mfrow=c(1,1))
#setting the seed ensures that your results will match ours
#delete this line for stochastic results
set.seed(12)

n.pix= 20 #number of pixels in raster
r<-raster(nrows=n.pix,ncols=n.pix) #create an empty raster
projection(r)<- "+proj=utm +zone=12 +datum=WGS84" #set the projection to be UTMs
xmin<- ymin<- .5
xmax<- ymax<- 4.5
extent(r)<-c(xmin,xmax,ymin,ymax) #set the extent of the raster
delta<- (xmax-xmin)/n.pix  #find the distance between points
gx<- seq(xmin + delta/2, xmax-delta/2,,n.pix) #create a sequence for the x locations
gy<- rev(gx) #create a sequence for the y locations in reverse order
gx<-sort(rep(gx,20)) #sort and repeat the sequence of x locations for all columns
gy<-rep(gy,20) #repeat the y locations for all rows
grid<-cbind(gx,gy) #bind the x and y components of the locations
Dmat<-as.matrix(dist(grid)) #calculate the Euclidean distances among the potential activity centers
V<-exp(-Dmat/.5)  #Create the correlation function: standard exponential with range paramater 0.5
z<-t(chol(V))%*%rnorm(400) #Create the correlated variable z, with some random noise
z<- (z-mean(z))/sqrt(var(as.vector(z))) #center the correlated variable
values(r)<-matrix(z,20,20,byrow=FALSE)  #assign z to the raster r
#plot the covariate
par(mfrow=c(2,1))
hist(z)
plot(r)
points(grid)
#dev.off() #removes figure
covariate.patchy<- z*sqrt(1.68) + .168   #approx. same scale as systematic covariate below
values(r)<-matrix(covariate.patchy,20,20,byrow=FALSE) #assign the scaled covariate values to r
covariate.patchy<- r  #create a new raster called covariate.patchy
#class(covariate.patchy) #check to confirm that covariate.patchy is a RasterLayer

## SYSTEMATIC COVARIATE
## It's much simpler to build a systematic (or trend) covariate.
## Here we defined as a trend from NW to SE
##
cost<-matrix(NA,nrow=n.pix,ncol=n.pix)
cost<-row(cost)+col(cost)
covariate.trend<- (cost-20)/10 #define cost values
values(r)<-matrix(covariate.trend,n.pix,n.pix,byrow=FALSE) #assign costs to raster r
covariate.trend<-r
class(covariate.trend)
plot(covariate.trend)


####SIMULATION FUNCTION
####Next we define a few conditions and then the simulation function,
####where we simulate activity centers and trap locations, calculate
####the probability of capture, simulate captures, and then fit the
####simulated data.  To do this a single time, first run the utility
####functions below, run the following conditions, and then run only
####the code inside of the function.

N<-200  #the number of individuals=activity centers
alpha0<- -2
sigma<- .5  #scale parameter for the distance distribution
K<- 5
nsim=1
covariate<-covariate.patchy
```

```
361
362 sim.fn<-function(N=200,nsim=100,alpha0= -2, sigma=.5, K=5,covariate){
363 #N= true number of individuals
364 #nsim = the number of simulations you'd like to run
365 #alpha0 is the resistance covariate
366 #sigma is the scale parameter
367 #K is the number of occasions
368 #covariate refers to the RasterLayer object that is the covariate
369
370 cl<-match.call() #stores all the arguments in the function call for later reference
371 #Create matrices to store the output
372 simout2<-simout1<-simout3<-matrix(NA,nrow=nsim,ncol=5)
373 alpha1<- 1/(2*sigma*sigma) #calculate alpha based on the sigma (scale parameter)
374
375 #Create a raster object
376 r<-raster(nrows=20,ncols=20)
377 projection(r)<- "+proj=utm +zone=12 +datum=WGS84" #assign the projection
378 extent(r)<-c(.5,4.5,.5,4.5)  #assign the extent of the raster
379 alpha2<-1 #assign the resistance coefficient
380 cost<- exp(alpha2*covariate) #calculate the resistance surface
381 #(i.e. the cost per pixel for each pixel of the covariate
382
383 #par(mfrow=c(1,1))
384 #plot(r)
385 r<-cost #rename the cost raster r
386 #identify neighbors and calculate distances as conductances (1/average distance between 2 pixels)
387 tr1<-transition(r,transitionFunction=function(x) 1/mean(x),directions=8)
388 #correct the diagonal distances
389 tr1CorrC<-geoCorrection(tr1,type="c",multpl=FALSE,scl=FALSE)
390
391 #Create the trap locations composed of locations xg, yg
392 xg<-seq(1,4,1)
393 yg<-4:1
394 traplocs<-cbind( sort(rep(xg,4)),rep(yg,4))
395 #graph the traplocations
396 points(traplocs,pch=20,col="red")
397 ntraps<-nrow(traplocs)
398
399
400 for(sim in 1:nsim){
401
402 #Create the activity centers, distributed uniformly in the state space= extent
403 #Note that the raster defines the state space because least cost distance
404 #can not be calculated where there are no covariate values.
405
406 S<-cbind(runif(N,.5,4.5),runif(N,.5,4.5))
407
408 #calculate the least cost distances between activity centers and traplocations
409
410 D<- costDistance(tr1CorrC,S,traplocs)  #N x ntraps matrix
411
412 #calculate the probability of capture based on the weighted distance distribution
413
414 probcap<-plogis(alpha0)*exp(-alpha1*D*D) #N X ntraps matrix
415
416 # now generate the number of encounters of every individual in every trap
417
418 Y<-matrix(NA,nrow=N,ncol=ntraps) #N X ntraps matrix
419 for(i in 1:nrow(Y)){
420  Y[i,]<-rbinom(ntraps,K,probcap[i,]) #simulate # of encounters from K occasions
421 }
422
423 #select only those individuals that were counted at least once for the capture history
424
425 Y<-Y[apply(Y,1,sum)>0,] #matrix of N actually captured X ntraps
426 n0<- N-nrow(Y)  #number of zero-encounter histories
427
428 #calculate the likelihood based solely on Euclidean distance
429 frog<-optim(c(alpha0,alpha1,log(n0)),intlik3ed,hessian=TRUE,y=Y,K=K,X=traplocs,
430                distmet="euclid",covariate=covariate,alpha2=1)
431 # if using nlm and sometimes with optim, warnings can be produced. This
432 # is due to parameters outside of parameter space or small number
433 # arithmetic.
434 # ignore these warnings.
```

```
435
436 simout1[sim,]<-c(frog$par,NA,nrow(Y))
437
438 #calculate the likelihood based on ecological distance with cost coefficient fixed
439 frog<-optim(c(alpha0,alpha1,log(n0)),intlik3ed,hessian=TRUE,y=Y,K=K,X=traplocs,
440                distmet="ecol",covariate=covariate,alpha2=1)
441 simout2[sim,]<-c(frog$par,NA,nrow(Y))
442
443 #calculate the likelihood based on ecological distance with cost coefficient estimated
444 frog<-optim(c(alpha0,alpha1,log(n0),-.3),intlik3ed,hessian=TRUE,y=Y,K=K,X=traplocs,
445                distmet="ecol",covariate=covariate,alpha2=NA)
446 simout3[sim,]<-c(frog$par,nrow(Y))
447 }
448 #output results of simulations, plus the inputs to the function (cl defined above)
449 list(simout1=simout1,simout2=simout2,simout3=simout3,call=cl)
450
451 }  #end of the simulation function
452
453
454
455 #####The next groups of code were used to specify and summarize the simulations reported in the paper.
456 ###
457 ### R commands to carry-out the simulations. MUST SOURCE likelihood function first -- SEE BELOW
458 ### This takes 1-2 days to run for nsims=100
459 ###
460 ###
461
462 nsims<-50
463 ###
464 ####Systematic covariate
465 ###
466 simout.low.N100<-sim.fn(N=100,nsim=nsims,alpha0=-2,sigma=.5,K=5,covariate=covariate.trend)
467 simout.low.N200<-sim.fn(N=200,nsim=nsims,alpha0= -2, sigma=.5,K=5,covariate=covariate.trend)
468 simout.reallylow.N100<-sim.fn(N=100,nsim=nsims,alpha0=-2,sigma=.5,K=3,covariate=covariate.trend)
469 simout.reallylow.N200<-sim.fn(N=200,nsim=nsims,alpha0= -2, sigma=.5,K=3,covariate=covariate.trend)
470 simout.high.N100<-sim.fn(N=100,nsim=nsims,alpha0=-2,sigma=.5,K=10,covariate=covariate.trend)
471 simout.high.N200<-sim.fn(N=200,nsim=nsims,alpha0= -2, sigma=.5,K=10,covariate=covariate.trend)
472 ###
473 ### R commands to summarize the simulation output
474 ###
475 mat<-matrix(NA,nrow=9,ncol=10)
476 for(i in 1:3){
477 mat[i,1:5]<- smy.fn(exp(simout.reallylow.N100[[i]][,3]) + simout.reallylow.N100[[i]][,5])
478 mat[i,6:10]<- smy.fn(exp(simout.reallylow.N200[[i]][,3]) + simout.reallylow.N200[[i]][,5])
479 mat[3+i,1:5]<- smy.fn(exp(simout.low.N100[[i]][,3]) + simout.low.N100[[i]][,5])
480 mat[3+i,6:10]<- smy.fn(exp(simout.low.N200[[i]][,3]) + simout.low.N200[[i]][,5])
481 mat[6+i,1:5]<- smy.fn(exp(simout.high.N100[[i]][,3]) + simout.high.N100[[i]][,5])
482 mat[6+i,6:10]<- smy.fn(exp(simout.high.N200[[i]][,3]) + simout.high.N200[[i]][,5])
483 }
484
485 ###
486 #### Patchy covariate
487 ###
488 simout.low.N100.k<-sim.fn(N=100,nsim=nsims,alpha0=-2,sigma=.5,K=5,covariate=covariate.patchy)
489 simout.low.N200.k<-sim.fn(N=200,nsim=nsims,alpha0= -2, sigma=.5,K=5,covariate=covariate.patchy)
490 simout.reallylow.N100.k<-sim.fn(N=100,nsim=nsims,alpha0=-2,sigma=.5,K=3,covariate=covariate.patchy)
491 simout.reallylow.N200.k<-sim.fn(N=200,nsim=nsims,alpha0= -2, sigma=.5,K=3,covariate=covariate.patchy)
492 simout.high.N100.k<-sim.fn(N=100,nsim=nsims,alpha0=-2,sigma=.5,K=10,covariate=covariate.patchy)
493 simout.high.N200.k<-sim.fn(N=200,nsim=nsims,alpha0= -2, sigma=.5,K=10,covariate=covariate.patchy)
494 ###
495 ### R commands to summarize the simulation output
496 ###
497 mat.k<-matrix(NA,nrow=9,ncol=10)
498 for(i in 1:3){
499 mat.k[i,1:5]<- smy.fn(exp(simout.reallylow.N100.k[[i]][,3]) + simout.reallylow.N100.k[[i]][,5])
500 mat.k[i,6:10]<- smy.fn(exp(simout.reallylow.N200.k[[i]][,3]) + simout.reallylow.N200.k[[i]][,5])
501 mat.k[3+i,1:5]<- smy.fn(exp(simout.low.N100.k[[i]][,3]) + simout.low.N100.k[[i]][,5])
502 mat.k[3+i,6:10]<- smy.fn(exp(simout.low.N200.k[[i]][,3]) + simout.low.N200.k[[i]][,5])
503 mat.k[6+i,1:5]<- smy.fn(exp(simout.high.N100.k[[i]][,3]) + simout.high.N100.k[[i]][,5])
504 mat.k[6+i,6:10]<- smy.fn(exp(simout.high.N200.k[[i]][,3]) + simout.high.N200.k[[i]][,5])
505 }
506
507
508
```