Package 'ctmcmove'

April 20, 2018

Type Package

Title Modeling Animal Movement with Continuous-Time Discrete-Space Markov Chains

Version 1.2.9

Date 2018-04-20

Author Ephraim Hanks

Maintainer Ephraim Hanks <hanks@psu.edu>

Depends R (>= 2.10), raster, Matrix, fda, gdistance, sp

Suggests mgcv, dismo, crawl

Description Software to facilitates taking movement data in xyt format and pairing it with raster covariates within a continuous time Markov chain (CTMC) framework. As described in Hanks et al. (2015) <DOI:10.1214/14-AOAS803> , this allows flexible modeling of movement in response to covariates (or covariate gradients) with model fitting possible within a Poisson GLM framework.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2018-04-20 13:58:33 UTC

R topics documented:

`ctmcmove-package` *ctmcmove*

Description

Software to facilitates taking movement data in xyt format and pairing it with raster covariates within a continuous time Markov chain (CTMC) framework. As described in Hanks et al. (2015) <DOI:10.1214/14-AOAS803> , this allows flexible modeling of movement in response to covariates (or covariate gradients) with model fitting possible within a Poisson GLM framework.

Details

Typical work flow for analysis of telemetry / GPS movement data:

1. Fit a quasi-continuous path model to telemetry xyt data. The ctmcmove package facilitates this through the "mcmc.fmove" function.

2. Create or import raster layers (from package "raster") for each covariate.

3. Impute a quasi-continuous path (done jointly with model fitting in the "mcmc.fmove" function.

4. Turn the quasi-continuous path into a CTMC discrete-space path using the "path2ctmc" command.

5. Turn discrete-space path into Poisson GLM format using the "ctmc2glm" command.

6. Repeat #3 - #5 multiple times (M times). Stack together the response "z", model matrix "X", and offset "tau" elements from each imputed path.

7. Fit a Poisson GLM model to the stacked data with response "z", model matrix "X", offset "log(tau)", and weights for each row equal to "1/M".

7 (alternate). Alternately, multiple imputation could be used, as described in Hanks et al., (2015).

Author(s)

Ephraim M. Hanks

Maintainer: Ephraim M. Hanks

References

Hanks, E. M.; Hooten, M. B. & Alldredge, M. W. Continuous-time Discrete-space Models for Animal Movement The Annals of Applied Statistics, 2015, 9, 145-165

Hanks, E.; Hooten, M.; Johnson, D. & Sterling, J. Velocity-Based Movement Modeling for Individual and Population Level Inference PLoS ONE, Public Library of Science, 2011, 6, e22795

Hooten, M. B.; Johnson, D. S.; Hanks, E. M. & Lowry, J. H. Agent-Based Inference for Animal Movement and Selection Journal of Agricultural, Biological, and Environmental Statistics, 2010, 15, 523-538

Examples

```
## Not run:

##
## Example of using a CTMC model for movement
##
## Steps:
##  1. Fit Quasi-Continuous Path Model to telemetry data (done using Buderman et al 2015)
##  2. Create covariate raster objects (the CTMC will be on the raster
##     grid cells)
##  3. Impute a quasi-continuous path
##  4. Turn quasi-continuous path into a CTMC discrete-space path
##  5. Turn discrete-space path into latent Poisson GLM format
##  6. Fit a Poisson GLM model to the data
##

library(ctmcmove)
data(seal)
xyt=seal$locs[,3:1]
head(xyt)
plot(xyt[,1:2],type="b")
xy=xyt[,-3]
x=xyt[,1]
y=xyt[,2]
t=xyt[,3]


#########################


################################################################################
##
## 1. Fit functional movement model to telemetry data
##
################################################################################

library(fda)

## Define the knots of the spline expansion.
##
## Problems with fitting the functional movement model can often be fixed by
## varying the spacing of the knots.
knots = seq(min(t),max(t),by=1/4)
## create B-spline basis vectors used to approximate the path
b=create.bspline.basis(c(min(t),max(t)),breaks=knots,norder=3)
## define the sequence of times on which to sample the imputed path
tpred=seq(min(t),max(t),by=1/24/60)



## Fit latent Gaussian model using MCMC
out=mcmc.fmove(xy,t,b,tpred,QQ="CAR",n.mcmc=400,a=1,r=1,num.paths.save=30)
```

```
str(out)

## plot 3 imputed paths
plot(xy,type="b")
points(out$pathlist[[1]]$xy,col="red",type="l")
points(out$pathlist[[2]]$xy,col="blue",type="l")
points(out$pathlist[[3]]$xy,col="green",type="l")


#############################################################################
##
## 2. Creating rasters
##
#############################################################################

cov.df=seal$cov.df
str(cov.df)

NN=sqrt(nrow(cov.df$X))
sst=matrix(seal$cov.df$X$sst,NN,byrow=TRUE)
sst=sst[NN:1,]
sst=raster(sst,xmn=min(seal$cov.df$X$x),xmx=max(seal$cov.df$X$x),
           ymn=min(seal$cov.df$X$y),ymx=max(seal$cov.df$X$y))


crs(sst)="+proj=longlat +datum=WGS84"
plot(sst)

chA=matrix(seal$cov.df$X$chA,NN,byrow=TRUE)
chA=chA[NN:1,]
chA=raster(chA,xmn=min(seal$cov.df$X$x),xmx=max(seal$cov.df$X$x),
           ymn=min(seal$cov.df$X$y),ymx=max(seal$cov.df$X$y))
crs(chA)="+proj=longlat +datum=WGS84"

pro=matrix(seal$cov.df$X$pro,NN,byrow=TRUE)
pro=pro[NN:1,]
npp=raster(pro,xmn=min(seal$cov.df$X$x),xmx=max(seal$cov.df$X$x),
           ymn=min(seal$cov.df$X$y),ymx=max(seal$cov.df$X$y))
crs(npp)="+proj=longlat +datum=WGS84"


int=sst
values(int) <- 1

d2r=int
rookery.cell=cellFromXY(int,xyt[1,1:2])
values(d2r)=NA
values(d2r)[rookery.cell]=0
d2r=distance(d2r)

grad.stack=stack(sst,chA,npp,d2r)
names(grad.stack) <- c("sst","cha","npp","d2r")
```

```
plot(sst)
points(xyt[,1:2],type="b")

plot(grad.stack)


##############################################################################
##
## 3 Impute Quasi-Continuous Paths
##
##############################################################################

P=20

plot(sst,col=grey.colors(100))
for(i in 1:P){
    points(out$pathlist[[i]]$xy,col=i,type="l",lwd=2)
}
points(xyt[,1:2],type="b",pch=20,cex=2,lwd=2)

##############################################################################
##
## 4. Turn continuous space path into a CTMC discrete space path
##
##############################################################################

path=out$pathlist[[1]]
ctmc=path2ctmc(path$xy,path$t,int,method="LinearInterp")
## alternate method, useful if you have impassible barriers, but slower
## ctmc=path2ctmc(path$xy,path$t,int,method="ShortestPath")

str(ctmc)

##############################################################################
##
## 5. Turn CTMC discrete path into latent Poisson GLM data
##
##############################################################################


loc.stack=stack(int,sst)
names(loc.stack) <- c("Intercept","sst.loc")

glm.list=list()
glm.list[[1]]=ctmc2glm(ctmc,loc.stack,grad.stack)

str(glm.list)

for(i in 2:P){
    cat(i," ")
    path=out$pathlist[[i]]
    ctmc=path2ctmc(path$xy,path$t,int,method="LinearInterp")
    glm.list[[i]]=ctmc2glm(ctmc,loc.stack,grad.stack)
```

```
}

## remove transitions that are nearly instantaneous
##  (These are essentially outliers in the following regression analyses)
for(i in 1:P){
    idx.0=which(glm.list[[i]]$tau<10^-5)
    if(length(idx.0)>0){
        glm.list[[i]]=glm.list[[i]][-idx.0,]
    }
    glm.list[[i]]$t=glm.list[[i]]$t-min(glm.list[[i]]$t)
}


##
## Stack the P imputations together
##

glm.data=glm.list[[1]]
for(i in 2:P){
    glm.data=rbind(glm.data,glm.list[[i]])
}

str(glm.data)

#############################################################################
##
## 6. Fit Poisson GLM
##    (here we are fitting all "M" paths simultaneously,
##     giving each one a weight of "1/M")
##
#############################################################################

fit.SWL=glm(z~cha+npp+sst+crw+d2r+sst.loc,
        weights=rep(1/P,nrow(glm.data)),family="poisson",offset=log(tau),data=glm.data)
summary(fit.SWL)

beta.hat.SWL=coef(fit.SWL)
beta.se.SWL=summary(fit.SWL)$coef[,2]

#############################################################################
##
## 6. Fit Poisson GLM
##    (here we are fitting using Multiple Imputation
##
#############################################################################

## Fit each path individually
glm.fits=list()
for(i in 1:P){
    glm.fits[[i]]=glm(z~cha+npp+sst+crw+d2r+sst.loc,
        family="poisson",offset=log(tau),data=glm.list[[i]])
}
```

```
## get point estimates and sd estimates using Rubin's MI combining rules
beta.hat.mat=integer()
beta.se.mat=integer()
for(i in 1:P){
    beta.hat.mat=rbind(beta.hat.mat,coef(glm.fits[[i]]))
    beta.se.mat=rbind(beta.se.mat,summary(glm.fits[[i]])$coef[,2])
}

beta.hat.mat
beta.se.mat

## E(beta) = E_paths(E(beta|path))
beta.hat.MI=apply(beta.hat.mat,2,mean)
beta.hat.MI

## Var(beta) = E_paths(Var(beta|path))+Var_paths(E(beta|path))
beta.var.MI=apply(beta.se.mat^2,2,mean)+apply(beta.hat.mat,2,var)
beta.se.MI=sqrt(beta.var.MI)

cbind(beta.hat.MI,beta.se.MI)

##
## compare estimates from MI and Stacked Weighted Likelihood approach
##

## standardize regression coefficients by multiplying by the SE of the X matrix
sds=apply(model.matrix(fit.SWL),2,sd)
sds[1]=1

## plot MI and SWL regression coefficients
par(mfrow=c(1,2))
plot(beta.hat.MI*sds,beta.hat.SWL*sds,main="(a) Coefficient Estimates",
xlab="Weighted Likelihood Coefficient",
ylab="Multiple Imputation Coefficient",pch=20,cex=2)
abline(0,1,col="red")
plot(log(beta.se.MI),log(beta.se.SWL),
main="(b) Estimated log(Standard Errors)",xlab="Weighted Likelihood log(SE)",
 ylab="Multiple Imputation log(SE)",pch=20,cex=2)
abline(0,1,col="red")


#############################################################################
##
## 6. (Alternate) We can use any software which fits Poisson glm data.
##    The following uses "gam" in package "mgcv" to fit a time-varying
##    effect of "d2r" using penalized regression splines.  The result
##    is similar to that found in:
##
##    Hanks, E.; Hooten, M.; Johnson, D. & Sterling, J. Velocity-Based
##    Movement Modeling for Individual and Population Level Inference
##    PLoS ONE, Public Library of Science, 2011, 6, e22795
##
#############################################################################
```

```
library(mgcv)

fit=gam(z~cha+npp+crw+sst.loc+s(t,by=-d2r),
        weights=rep(1/P,nrow(glm.data)),family="poisson",offset=log(tau),data=glm.data)
summary(fit)

plot(fit)
abline(h=0,col="red")




############################################################
##
## Overview Plot
##
############################################################


## pdf("sealfig.pdf",width=8.5,height=8.85)
par(mfrow=c(3,3))
##
plot(sst,col=(terrain.colors(30)),main="(a) Sea Surface Temperature")
points(xyt[1,1:2]-c(0,.05),type="p",pch=17,cex=2,col="red")
points(xyt[,1:2],type="b",pch=20,cex=.75,lwd=1)
##
plot(d2r/1000,col=(terrain.colors(30)),main="(b) Distance to Rookery")
points(xyt[1,1:2]-c(0,.05),type="p",pch=17,cex=2,col="red")
points(xyt[,1:2],type="b",pch=20,cex=.75,lwd=1)
##
image(sst,col=rev(terrain.colors(30)),main="(c) Imputed Functional Paths",xlab="",ylab="")
for(i in 1:5){
    ## points(out$pathlist[[i]]$xy,col=i+1,type="l",lwd=3)
    points(out$pathlist[[i]]$xy,col=i+1,type="l",lwd=2)
}
points(xyt[,1:2],type="p",pch=20,cex=.75,lwd=1)
##
ee=extent(c(188.5,190.5,58.4,59.1))
sst.crop=crop(sst,ee)
bg=sst.crop
values(bg)=NA
for(i in c(2)){
    values(bg)[cellFromXY(bg,out$pathlist[[i]]$xy)] <- 1
}
image(sst.crop,col=(terrain.colors(30)),xlim=c(188.85,190.2),
ylim=c(58.5,59),main="(d) CTMC Path",xlab="",ylab="")
image(bg,col="blue",xlim=c(188.85,190.2),ylim=c(58.5,59),add=TRUE)
for(i in c(2)){
    points(out$pathlist[[i]]$xy,col=i,type="l",lwd=3)
}
points(xyt[,1:2],type="b",pch=20,cex=2,lwd=2)
```

```
##
image(sst.crop,col=(terrain.colors(30)),xlim=c(189.62,189.849),
ylim=c(58.785,58.895),main="(e) CTMC Model Detail",xlab="",ylab="")
abline(v=189.698+res(sst)[1]*c(-1,0,1,2))
abline(h=58.823+res(sst)[2]*c(-1,0,1,2))
##
plot(fit,main="(f) Time-Varying Response to Rookery",shade=TRUE,
shade.col="orange",lwd=3,rug=F,xlab="Day of Trip",
ylab="Coefficient of Distance To Rookery")
abline(h=0,col="red")
##



################################################
##
## Get UD (following Kenady et al 2017+)
##
################################################

RR=get.rate.matrix(fit.SWL,loc.stack,grad.stack)
UD=get.UD(RR,method="lu")
ud.rast=sst
values(ud.rast) <- as.numeric(UD)
plot(ud.rast)


################################################
##
## Get shortest path and current maps (following Brennan et al 2017+)
##
################################################

library(gdistance)

## create a dummy transition layer from a raster.
## make sure the "directions" argument matches that used in path2ctmc
## also make sure to add the "symm=FALSE" argument
trans=transition(sst,mean,directions=4,symm=FALSE)
## now replace the transition object with the "rate" matrix
## so "conductance" values are "transition rates"
transitionMatrix(trans) <- RR
str(trans)

##
## now calculate least cost paths using "shortestPath" from gdistance
##

## pick start and end locations
plot(sst)
st=c(185,59.5)
en=c(190,57.3)
```

```
st.cell=cellFromXY(sst,st)
en.cell=cellFromXY(sst,en)

## shortest path
sp=shortestPath(trans,st,en,output="SpatialLines")
plot(sst,main="Shortest Path (SST in background)")
lines(sp,col="brown",lwd=7)



##
## Now calculate "current maps" that show space use of random walkers
## moving between two given locations.
##
## gdistance's "passage" function allows for asymmetric transition rates
##

passage.gdist=passage(trans,st,en,theta=.001,totalNet="net")
plot((passage.gdist))



## End(Not run)
```

`ctmc.sim` *Code to simulate a continuous-time Markov chain.*

Description

Simulates a CTMC with given rate matrix (Q) for a time (T), or until it reaches a final absorbing state.

Usage

```
ctmc.sim(Q,start.state=1,T=1,final.state=NA)
```

Arguments

`Q`	A square matrix. Either a rate matrix or the infinitessimal generator of the CTMC.
`start.state`	An integer - the starting state for the simulation.
`T`	A numeric value greater than zero. The time window for simulating the CTMC will be [0,T].
`final.state`	Either NA or an integer. If an integer, the chain will be simulated until it enters the "final.state", at which time the simulation will be terminated.

Details

This code uses the Gillespie algorithm to simulate a CTMC path in continuous time.

Value

ec	A vector of the sequential grid cells (the embedded chain) in the CTMC movement path
rt	A vector of residence times in each sequential grid cell in the CTMC movement path

Author(s)

Ephraim M. Hanks

References

None

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

---

ctmc2glm *Convert a "ctmc" object into Poisson glm format.*

Description

Transforms a "ctmc" object and covariate rasters into data suitable for analysis using Poisson GLM software (like glm in R).

Usage

```
ctmc2glm(ctmc, stack.static, stack.grad, crw = TRUE,
    normalize.gradients = FALSE, grad.point.decreasing = TRUE,
    include.cell.locations=TRUE,directions=4,zero.idx=integer())
```

Arguments

ctmc	A "ctmc" object (output from "path2ctmc").
stack.static	A rasterStack object, where each layer in the stack is a location-based covariate.
stack.grad	A rasterStack object, where each layer in the stack is a directional gradient-based covariate

`crw` Logical. If TRUE (default), an autocovariate is created for each cell visited in the CTMC movement path. The autocovariate is a unit-length directional vector pointing from the center of the most recent grid cell to the center of the current grid cell.

`normalize.gradients` Logical. Default is FALSE. If TRUE, then all gradient covariates are normalized by dividing by the length of the gradient vector at each point.

`grad.point.decreasing` Logical. If TRUE, then the gradient covariates are positive in the direction of decreasing values of the covariate. If FALSE, then the gradient covariates are positive in the direction of increasing values of the covariate (like a true gradient).

`include.cell.locations` Logical. If TRUE, then the x and y locations of the centers of the (1) current and (2) neighboring raster cells will be returned for each row in the created data matrix.

`directions` Integer. Either 4 (indicating a "Rook's neighborhood" of 4 neighboring grid cells) or 8 (indicating a "King's neighborhood" of 8 neighboring grid cells).

`zero.idx` Integer vector of the indices of raster cells that are not passable and should be excluded. These are cells where movement should be impossible. Default is zero.idx=integer().

Details

This code creates one data row for each possible transition from each grid cell visited by the CTMC path.

Value

`z` Response variable (either zero or 1) for analysis using GLM software.

`X` Matrix of predictor variables for analysis using GLM software. Created from the location-based and gradient-based covariates.

`tau` Offset for each row in a Poisson GLM with log link.

`t` Vector of the time each raster grid cell was entered

Author(s)

Ephraim M. Hanks

References

Hanks, E. M.; Hooten, M. B. & Alldredge, M. W. Continuous-time Discrete-space Models for Animal Movement The Annals of Applied Statistics, 2015, 9, 145-165

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

`get.rate.matrix` *Create a CTMC rate matrix from rasters and parameter estimates.*

Description

Creates a CTMC rate matrix from rasters and parameter estimates (perhaps from a GLM analysis).

Usage

```
get.rate.matrix(object, stack.static, stack.grad,
    normalize.gradients = FALSE, grad.point.decreasing = TRUE,
    directions=4, zero.idx=integer(), coef)
```

Arguments

`object`	A fitted GLM or GAM object used to fit the CTMC movement model
`stack.static`	A rasterStack object, where each layer in the stack is a location-based covariate.
`stack.grad`	A rasterStack object, where each layer in the stack is a directional gradient-based covariate
`normalize.gradients`	Logical. Default is FALSE. If TRUE, then all gradient covariates are normalized by dividing by the length of the gradient vector at each point.
`grad.point.decreasing`	Logical. If TRUE, then the gradient covariates are positive in the direction of decreasing values of the covariate. If FALSE, then the gradient covariates are positive in the direction of increasing values of the covariate (like a true gradient).
`directions`	Integer. Either 4 (indicating a "Rook's neighborhood" of 4 neighboring grid cells) or 8 (indicating a "King's neighborhood" of 8 neighboring grid cells).
`zero.idx`	Integer vector of the indices of raster cells that are not passable and should be excluded. These are cells where movement should be impossible. Default is zero.idx=integer().
`coef`	A vector of coefficents to use in place of those in 'object'

Details

This function takes the covariate rasters in stack.static (motility covariates) and stack.grad (gradient covariates) and creates a CTMC rate matrix defining movement between all neighboring raster grid cells. It is NOT possible to include an autocovariate here ("crw" in ctmc2glm). If such was included in the original fitted model, then the crw term is set equal to zero.

Value

An n-by-n Matrix of CTMC rate values.

Author(s)

Ephraim M. Hanks

References

Hanks, E. M.; Hooten, M. B. & Alldredge, M. W. Continuous-time Discrete-space Models for Animal Movement. The Annals of Applied Statistics, 2015, 9, 145-165

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

---

get.UD *Find the stationary distribution of the CTMC.*

Description

Finds the stationary distribution (proportional utilization distribution) implied by a CTMC movement model with a given rate matrix.

Usage

```
get.UD(R,method="lu",maxiter, start, tol)
```

Arguments

R	Rate matrix with R[i,j] equal to the CTMC rate of movement from raster cell i to neighboring raster cell j. R[i,j]=0 implies that cells i and j are not first order neighbors.
method	Either "lu" (default) or "limit". See Details for a description of the two methods.
start	A value for the starting distribution for the 'limit' method. Defaults to 1/num. cells. Ignored for method='lu'.
maxiter	Total number of iterations for limit method if tolerance not reached first. Defaults to 100. Ignored for method='lu'.
tol	Value used to assess convergence for limit method. If max(abs(pi1-pi0))<tol, limit method has converged. Defaults to sqrt(.Machine$double.eps)

Details

This calculates the stationary distribution of the CTMC. If method="lu", then the method used is the method on pg. 455 of Harrod and Plemmons (1984). If method="limit", then the stationary distribution is approximated by brute-force simulation. If R is a sparse Matrix object, then sparse matrix methods are used, making this calculation extremely efficient.

Value

Vector of the stationary distribution at each raster grid cell

Author(s)

Ephraim M. Hanks

References

Harrod, W. J. & Plemmons, R. J. Comparison of some direct methods for computing stationary distributions of Markov chains. SIAM Journal on Scientific and Statistical Computing, 1984, 5, 453-469

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

`mcmc.fmove` *Fit continuous-time functional movement model to telemetry data.*

Description

Fits a functional movement model to telemetry data following Buderman et al., 2015.

Usage

```
mcmc.fmove(xy,t,fdabasis,tpred=t,QQ="CAR2",a=1,b=1,r=1,q=1,
           n.mcmc=100,num.paths.save=10,sigma.fixed=NA)
```

Arguments

`xy`	A two-column matrix with each row corresponding to the x,y locations of a telemetry location.
`t`	A numeric vector of length = nrow(xy), with the i-th entry corresponding to the time of the i-th telemetry location in xy.
`fdabasis`	A "basisfd" object, typically resulting from a call to "create.bspline.basis" in the fda package. Other basis functions can be used.
`tpred`	Numeric vector of times to impute the quasi-continuous path.
`QQ`	The precision matrix of the fda basis coefficients. This can either be a string, taking on values of "CAR1" or "CAR2", or can be a user specified matrix (or sparse matrix using the Matrix package) of dimension equal to the number of basis functions in fdabasis. Defaults to "CAR2". "CAR1" will result in less-smooth paths.
`a`	The shape parameter of the inverse gamma prior on the observation variance.

b	The scale parameter of the inverse gamma prior on the observation variance.
r	The shape parameter of the inverse gamma prior on the partial sill parameter of the spline basis coefficients.
q	The scale parameter of the inverse gamma prior on the partial sill parameter of the spline basis coefficients.
n.mcmc	Number of mcmc iterations to run.
num.paths.save	Number of quasi-continuous path realizations to save. Defaults to 10.
sigma.fixed	Numeric value (or the default NA). If NA, then the observation variance sigma^2 is estimated using MCMC. If a numeric value, this is the fixed standard deviation of the observation error.

Details

Fits the functional movement model of Buderman et al., 2015, and outputs quasi-continuous paths that stochastically interpolate between telemetry locations. The model fit is as follows (written out for 1-D):

y_t = observed location at time t

z_t = Sum_k beta_k*phi_k(t) = true location at time t, expressed using a linear combination of spline basis functions phi_k(t).

y_t ~ N(z_t , sigma^2)

beta ~ N(0 , tau^2 * QQ^-1)

sigma^2 ~ IG(a,b)

tau^2 ~ IG(r,q)

Value

s2.save	Numeric vector of the values of sigma^2 at each mcmc iteration
tau2.save	Numeric vector of the values of tau^2 at each mcmc iteration
pathlist	A list of length num.paths.save, with each item itself being a list with two entries: xy = a matrix with rows corresponding to x,y locations of the quasi-continuous path imputation t = a vector with entries corresponding to the times at which the quasi-continuous path was imputed

Author(s)

Ephraim M. Hanks

References

Buderman, F.E.; Hooten, M. B.; Ivan, J. S. and Shenk, T. M. A functional model for characterizing long-distance movement behavior. Methods in Ecology and Evolution, 2016, 7, 264-273.

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

`path2ctmc` *Function to turn a discrete-time continuous-space path into a CTMC path.*

Description

This function takes a movement path defined by xyt values (not necessarily equally spaced in time), and converts it into a CTMC path (a continuous-time discrete-space path on grid cells in a raster).

Usage

```
path2ctmc(xy, t, rast,directions=4,zero.idx=integer(),print.iter=FALSE,
method="ShortestPath")
```

Arguments

`xy`	A matrix of x,y locations at T time points.
`t`	A vector of T times associated with the T locations in "xy".
`rast`	A raster object or raster stack object that will define the discrete-space grid cells for the CTMC movement path.
`directions`	Integer. Either 4 (indicating a "Rook's neighborhood" of 4 neighboring grid cells) or 8 (indicating a "King's neighborhood" of 8 neighboring grid cells).
`zero.idx`	Integer vector of the indices of raster cells that are not passable and should be excluded. These are cells where movement should be impossible. Default is zero.idx=integer().
`print.iter`	Logical. If true, then the progress stepping through each observed location in "xy" and "t" will be output in the terminal.
`method`	Specifies interpolation method. Either "ShortestPath", which uses the shortest graphical path on the raster graph, or "LinearInterp", which linearly interpolates between observed locations. "ShortestPath" is slower, slightly more accurate, and allows for impassible barriers specified through "zero.idx". "LinearInterp" is faster but does not allow for impassible barriers.

Details

This takes a xyt path and turns it into a list of the embedded chain and residence times of a continuous time Markov chain walk on the graph. A "zero.idx" indicates impassible grid cells. When successive (x,y) locations are not in the same grid cell, a shortest path between locations is found using the "shortestPath" function from gdistance, and the time between (x,y) locations is then evenly divided between all grid cells in the shortest path.

Value

ec	A vector of the sequential grid cells (the embedded chain) in the CTMC movement path
rt	A vector of residence times in each sequential grid cell in the CTMC movement path
trans.times	A vector of times in which the movement path enters the grid cell in "ec".

Author(s)

Ephraim M. Hanks

References

Hanks, E. M.; Hooten, M. B. & Alldredge, M. W. Continuous-time Discrete-space Models for Animal Movement The Annals of Applied Statistics, 2015, 9, 145-165

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

Pctmc *Transition Matrix of a CTMC.*

Description

Computes the transition matrix P(t) of a CTMC with given rate matrix (Q) and time (t).

Usage

```
Pctmc(Q,t)
```

Arguments

Q	A square matrix. Either a rate matrix or the infinitessimal generator of the CTMC.
t	A numeric value - the time step.

Details

Uses the method of homogenization to compute the probability transition matrix given by exp(Q*t).

Value

A square matrix P with entries P[i,j]=Prob(X(t)=j|X(0)=i)

Author(s)

Ephraim M. Hanks

References

Hanks, E. M.; Hooten, M. B. & Alldredge, M. W. Continuous-time Discrete-space Models for Animal Movement The Annals of Applied Statistics, 2015, 9, 145-165

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

---

rast.grad *Creates gradient rasters from a raster object.*

Description

This function takes a raster stack or raster object and creates two matrices for each raster layer, one which contains the x coordinates of the gradient of the raster layer and one which contains the y coordinates of the gradient of the raster layer.

Usage

```
rast.grad(rasterstack)
```

Arguments

rasterstack	A raster layer or raster stack from package "raster".

Details

The gradient is computed using the "terrain" function in raster.

Value

xy	A matrix of x and y coordinates of each cell in the raster stack or raster layer. The order is the order of the cells in the raster object.
grad.x	a matrix where each column is the x-coordinates of the gradient for one raster layer
grad.y	a matrix where each column is the y-coordinates of the gradient for one raster layer
rast.grad.x	A raster stack where each raster layer is the x-coordinates of the gradient for one covariate
rast.grad.y	A raster stack where each raster layer is the x-coordinates of the gradient for one covariate

Author(s)

Ephraim M. Hanks

References

Hanks, E. M.; Hooten, M. B. & Alldredge, M. W. Continuous-time Discrete-space Models for Animal Movement The Annals of Applied Statistics, 2015, 9, 145-165

Examples

```
## For example code, do
##
## > help(ctmcMove)
```

---

seal *Data for one foraging trip by a male northern fur seal (NFS).*

Description

seal$locs contains xyt locations for ARGOS fixes on the seal's location in the "datetime", "latitude", and "longitude" columns.

seal$cov.df contains a data.frame of spatial covariate values for sea surface temperature (sst), chlorophyll A levels (chA) and net primary production (npp).

Usage

```
data("seal")
```

Format

The format is:

$ locs :'data.frame': 163 obs. of 6 variables:

..$ datetime : num [1:163] 36741 36741 36741 36742 36742 ...

..$ latitude : num [1:163] 57.2 57.3 57.3 57.2 57.5 ...

..$ longitude : num [1:163] 190 190 190 190 190 ...

..$ landseamig: int [1:163] 0 1 1 1 1 1 1 1 1 ...

..$ lqadjust : int [1:163] 5 1 0 -2 -2 1 -2 -2 -2 -2 ...

..$ lq : Factor w/ 8 levels "0","1","2","3",..: 5 2 1 7 7 2 7 7 7 7 ...

$ sex : Factor w/ 2 levels "female","male": 2

$ cov.df :List of 4

..$ X :'data.frame': 10000 obs. of 5 variables:

.. ..$ x : num [1:10000] 184 184 184 184 184 ...

.. ..$ y : num [1:10000] 56.7 56.7 56.7 56.7 56.7 ...

.. ..$ chA: num [1:10000] 1.19 0.924 0.744 0.709 0.733 ...

.. ..$ sst: num [1:10000] 9.07 10.35 10.27 10.43 9.98 ...

.. ..$ pro: num [1:10000] 853 821 823 849 886 ...

Details

Covariate Rasters and ARGOS telemetry data for one NFS near the Pribilof islands.

Source

Hanks, E.; Hooten, M.; Johnson, D. & Sterling, J. Velocity-Based Movement Modeling for Individual and Population Level Inference PLoS ONE, Public Library of Science, 2011, 6, e22795

Examples

```
## For example code, do
##
## > help(ctmcmove)
```

Index