# Example R code for conducting a CTR analysis using generated data
# Hirsch, Ben T, Visser, Marco D, Kays, Roland W, Jansen, Patrick A.
# Nijmegen June 2011
# Revised August 2011

############################### load dependancies##############################
# Code requires package fdrtool & survival to be in library
# otherwise use e.g. install.packages("fdrtool") first
require(fdrtool);require(survival)

############################# Create data ####################################
# Next step is to generate example data "radiotagged distance", stored as object
x
# set random seed
set.seed(2011)
# generate data from lognormal distribution, 500 tracked seeds
# meanlog=log(50), sdlog=log(3)
x=rlnorm(500,log(50),log(3))
# truncate data after 20 units to create "tracked distances"
# with 20 m search radius
xtrunc=x[x<20]
# Prepare data for CTR
CTRdata=data.frame(
  # all seeds that went beyond 20 meters are treated as censored events
  # (distance > 20 meter)
d=c(xtrunc,rep(20,500-length(xtrunc))),
  # classify events, found seeds = 1, censored seeds = 0
  evnt=c(rep(1,length(xtrunc)),rep(0,500-length(xtrunc))))
# fitsurvival function
CTR_function=survfit(Surv(CTRdata$d, event=CTRdata$evnt) ~ 1)
# return survival probabilties (P) corresponding to distances (D)
P=summary(CTR_function)$surv;D=summary(CTR_function)$time

######################## Define dispersal kernels ##############################
# these kernels are then fit through OLS (Ordinary Least Squares) to objects P & D
# log normal
SSLN=function(param){
  Ex=1-plnorm(D,meanlog=param[1],sdlog=param[2])
  sum((Ex-P)^2)
}
# Weibull
SSW=function(param){
  Ex=1-pweibull(D,shape=param[1],scale=param[2])
  sum((Ex-P)^2)
}
# exponential
SSEX=function(param){
  Ex=1-pexp(D,rate=param)
  sum((Ex-P)^2)
}
### Obtain kernels with reconstructed tails

# Fit each model to the censored data and store

```r
fitLN=optim(c(1,1),SSLN)
LNpsave=c(fitLN$par[1],fitLN$par[2])

fitW=optim(c(1.2,55),SSW)
WBpsave=c(fitW$par[1],fitW$par[2])
```

# Note: above the OLS function was optimized with the Nelder-Mead algorithm
# however this algorithm is optimal for optimization problems of 2 Dimensions
# or greater. The quasi-Newton method 'BFGS' is better suited for 1 D (or 1
# parameter) problems. Alternatively the function 'optimize' can be used
# however result will be the same either way.

```r
fitN=optim(c(0.05),SSN,method="BFGS")
Npsave=c(fitN$par[1])

fitEX=optim(c(0.01),SSEX,method="BFGS")
EXpsave=c(fitEX$par[1])
```

# choose best model based on AIC score

```r
OLSscores=c(fitLN$value,fitW$value,fitN$value,fitEX$value)

# vector with number of parameters for each model
pars=c(2,2,1,1)
# calculating AIC from sum of squares
AICscores=(500*log(OLSscores/500) + 2*pars)
```

### FINAL

```r
bestfit=c("LN","WB","T","N","EX")[which(AICscores==min(AICscores))]
```

# checking difference in between estimated and generating kernel
par(cex.axis=0.9,cex.lab=1.1,las=1,mar=c(4,5,1,1),mfrow=c(2,1))
# density plots
```r
curve(dlnorm(x,log(50),log(3)),0,150,col='grey',xlab="distance",
ylab="probability density",lwd=2)
curve(dlnorm(x,fitLN$par[1],fitLN$par[2]),col='black',add=T,lty='dashed',lwd=2)
legend(100,0.010,legend=c('True', 'CTR derived'),lty=c('solid','dashed'),
col=c('grey','black'),bty='n',lwd=2)
```
# probability $P$ of dispersal beyond distance $D$

curve(1-plnorm(x,log(50),log(3)),0,150,col='grey',xlab="D",
ylab="P",lwd=2)
curve(1-
plnorm(x,fitLN$par[1],fitLN$par[2]),col='black',add=T,lty='dashed',lwd=2)
legend(100,0.90,legend=c('True', 'CTR derived'),lty=c('solid','dashed'),
col=c('grey','black'),bty='n',lwd=2)