Ecological Archives

Brian Dennis, José Miguel Ponciano. 2014. Density dependent state space model for population abundance data with unequal time intervals. *Ecology* VOL:pp-pp.

Supplement.

This supplement contains three R scripts: (1) RUNNING-ROUSS.R calculates maximum likelihood and restricted maximum likelihood estimates for the Ornstein Uhlenbeck state space model (stationary and non-stationary), using population abundance data having possibly unequal observation time intervals. (2) RUNNING-PBLRT.R performs parametric bootstrap likelihood ratio test of exponential growth state space model vs. Ornstein-Uhlenbeck state space model. (3) ROUSSE-1.0.R contains the functions used in statistical inferences for the Ornstein-Uhlenbeck state space model. ROUSSE-1.0.R is called by both RUNNING-ROUSS.R and RUNNING-PBLRT.R and must be present (as a file by that name) in the working directory of R.

```
#=========================================================================
#  RUNNING-ROUSS.R:  calculates maximum likelihood and restricted
#  maximum likelihood estimates for the Ornstein-Uhlenbeck state space
#  model (stationary and non-stationary), using population abundance
#  data having possibly unequal observation time intervals.  The script
#  ROUSSE-1.0.R must be present in the working directory of R.
#  Be patient;  R is slow.
#=========================================================================

#-------------------------------------------------------------------------
#  Load all the OUSS model functions and needed packages
#-------------------------------------------------------------------------
library("MASS")
source("ROUSSE-1.0.R")

#-------------------------------------------------------------------------
#        USER INPUT SECTION
#-------------------------------------------------------------------------
#  User supplies time series data here into the vector "Observed.t.
#  User can substitute R statements to read population abundance data
```

```
#  from a file into the vector "Observed.t". No zeros! Do not change
#  the object name "Observed.t". Times of observation are entered into
#  the vector "Time.t". Do not change the object name "Time.t".

#  First example data set is bobcat (Lynx rufus) in Idaho, data set
#  212 from the Global Population Dynamics Database. Various other data
#  sets are also included.  Pick any of these data sets and associated
#  sampling years by "commenting out" the Idaho bobcat data and
#  "uncommenting" the desired data.

# Lynx rufus, from Idaho, GPPD data set 212
Observed.t=c(346,675,802,1478,1173,756,861,972,854,1161,1318,901,901,
1173,608,811,903,584,1179,1020,1129,966)  #  No zeros!
Time.t=c(1956,1957,1958,1959,1960,1961,1962,1963,1964,1965,1970,1971,
1972,1973,1974,1975,1976,1977,1978,1979,1980,1981)

#  Linx rufus, from Florida, GPPD data set 211.
# Time.t=c(1946,1947,1948,1949,1950,1954,1955,1956,1957,1958,
# 1959,1960,1961,1963,1964,1965,1966,1967,1968,1975,1976,1977,
# 1978,1979,1980,1981)
# Observed.t=c(672,1028,538,566,300,400,400,400,400,300,250,
# 450,450,13,23,23,2,400,20,389,537,983,1698,1132,1702,1031)

# Lynx rufus, California, GPPD data set 208.
# Time.t=c(1934,1935,1936,1938,1940,1941,1942,1943,1944,1945,
# 1946,1947,1948,1949,1950,1951,1952,1954,1955,1956,1957,1958,
# 1959,1960,1961,1962,1963,1964,1965,1966,1967,1968,1969,1970,
# 1971,1972,1973,1974,1975,1976,1977,1978,1979,1980,1981)
# Observed.t=c(1994,1436,1290,2292,2776,3239,1923,2898,2063,1730,
# 1072,689,169,375,293,239,336,223,228,276,202,142,175,304,205,
# 295,361,221,221,241,244,381,588,319,588,686,1244,1393,2203,
# 3618,4445,6928,7809,9595,9337)

# Lynx rufus, Michigan, GPPD data set 218.
# Time.t=c(1936,1937,1939,1940,1941,1942,1943,1944,1945,1946,
# 1947,1948,1949,1950,1951,1952,1954,1955,1956,1957,1958,1962,
# 1963,1964,1966, 1969,1976,1977,1978,1979,1980,1981)
# Observed.t=c(1134,811,598,528,529,375,2538,2802,2910,2363,
# 2174,2063,1547,1753,1443,1836,696,847,880,762,200,588,494,265,
# 400,300,341,331,386,597,223,200)

# Lynx rufus, Maine, GPPD data set 216.
# Time.t=c(1934,1935,1936,1937,1942,1943,1944,1945,1946,1947,
# 1948,1949,1950,1951,1952,1953,1954,1956,1957,1958,1959,1961,
# 1962,1963,1964,1965,1966,1968,1970,1971,1972,1973,1974,1975,
# 1976,1977,1978,1979,1980,1981)
# Observed.t=c(644,911,687,400,133,105,184,1044,181,178,489,100,
# 263,83,106,795,667,695,263,198,221,278,231,588,269,152,233,153,
# 730,654,641,573,544,373,436,389,278,318,381,345)

# Lynx rufus, Wisconsin, GPPD data set 239.
# Time.t=c(1934,1935,1936,1937,1938,1940,1941,1942,1943,1944,
# 1945,1946,1947,1948,1949,1950,1951,1952,1954,1956,1959,1960,
# 1969,1970,1971,1974,1975,1976,1977,1978,1979,1980,1981)
# Observed.t=c(302,428,513,461,593,180,283,191,765,384,1048,577,
# 427,437,482,525,724,740,524,321,479,869,148,148,147,205,223,
# 275,163,223,131,81,168)
```

```
# Elk, central valley of Grand Teton National Park, cited in
# Dennis and Taper (1994 Ecological Monographs).
# Observed.t = c(1627,1527,824,891,1140,1322,1431,1733,1131,1611,
# 1644,1991,1762,1076,1442,1800,1667,1558,1396,1753,1453,1804)
# Time.t = c(1963,1964,1965,1966,1967,1968,1969,1970,1971,1972,
# 1973,1974,1975,1976,1977,1978,1979,1980,1981,1982,1984,1985)

# Rangeland grasshoppers, MT western mountain region, from
# Kemp and Dennis 1993 Oecologia)
# Observed.t=c(5.7981,7.7194,4.8022,3.9397,11.8806,10.7568,8.9586,
# 10.6619,6.5895,4.4905,3.0684,6.9973,5.3986,4.2777,6.1166,7.2989,
# 5.0850,4.8298,5.3997,4.7679,4.5073,1.9714,4.1007,5.6403,3.0492,
# 2.8144,4.4071,2.4121,3.2233,1.4236,2.3404,10.5283,7.6872,2.7305,
# 3.4570,5.4336,3.1487,3.8315,4.4805)
# Time.t=c(1948,1951,1952,1953,1954,1955,1956,1957,1958,1959,1960,
# 1961,1962,1963,1964,1965,1966,1967,1968,1969,1970,1971,1972,1973,
# 1974,1975,1977,1978,1979,1980,1981,1983,1984,1985,1986,1987,1988,
# 1989,1990)

# Log-transform the observations to carry out all the calculations
# in this program.
log.obs = log(Observed.t)
#-----------------------------------------------------------------------


#-----------------------------------------------------------------------
#        PARAMETER ESTIMATION, PARAMETRIC BOOTSTRAP AND PREDICTIONS
#-----------------------------------------------------------------------
# Before doing the calculations, the user has to specify ONLY the
# following 4 options:

# 1. Do you want to compute the ML estimates or the REML estimates?

method = "REML" # alternatively, set method = "ML"

# 2. Do you want to plot the predictions?
pred.plot = "TRUE" # Set it to "FALSE" if you do not want to plot the
                   # predictions

# 3. Do you want to plot the parametric bootstrap distribution of the
# estimates?
pboot.plot = "TRUE" # Set it to "FALSE" if you do not want to plot the
                    # bootstrap distribution of the estimates

# 4. How many bootstrap replicates?
NBoot = 1000


#-----------------------------------------------------------------------
#  5. THE FOLLOWING LINES OF CODE COMPUTE THE ESTIMATES, PREDICTIONS,
#  AND PARAMETRIC BOOTSTRAP CONFIDENCE INTERVALS. THE USER DOES NOT
#  NEED TO MODIFY THESE LINES OF CODE.
#
#  THE OUTPUT OF THE FUNCTION 'ROUSS.CALCS' IS A LIST AND THE USER
#  CAN RETRIEVE EACH OF THE LIST ELEMENTS PRINTED AND SAVED IN THE
#  OBJECT "all.results".
```

```
#
#  THE 95% PARAMETRIC BOOTSTRAP FOR BOTH, THE PARAMETERS AND THE
#  PREDICTIONS ARE COMPUTED BY THE FUNCTION "ROUSS.CALCS".
#
#-----------------------------------------------------------------

all.results = ROUSS.CALCS(Yobs=log.obs,Tvec=Time.t,pmethod=method,
               nboot=NBoot,plot.pred=pred.plot,
               plot.bootdists=pboot.plot)


#=================================================================
#  RUNNING-PBLRT.R:  performs parametric bootstrap likelihood ratio
#  test of exponential growth state space model vs. Ornstein-Uhlenbeck
#  state space model.  The script ROUSS-1.0.R must be in the working
#  directory of R.
#  Be patient;  R is slow.
#=================================================================

#-----------------------------------------------------------------
#  Load all the OUSS model functions and needed packages
#-----------------------------------------------------------------
library("MASS")
source("ROUSSE-1.0.R")

#-----------------------------------------------------------------
#        USER INPUT SECTION
#-----------------------------------------------------------------
#  User supplies time series data here into the vector "Observed.t.
#  User can substitute R statements to read population abundance data
#  from a file into the vector "Observed.t". Do not change the object
#  name "Observed.t".
#  Times of observation are entered into the vector "Time.t". Do not
#  change the object name "Time.t".
#  Example data from the North American Breeding Bird Survey, American
#  Redstart, record # 02014 3328 08636, 1966-95 (Table 1 in Dennis et
#  al. 2006 Ecological Monographs).

Observed.t = c(18,10,9,14,17,14,5,10,9,5,11,11,4,5,4,8,2,3,9,2,4,
7,4,1,2,4,11,11,9,6)
Time.t = c(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
21,22,23,24,25,26,27,28,29)

# Log-transform the observations to carry out all the calculations
# in this program.
log.obs = log(Observed.t)
#-----------------------------------------------------------------

#-----------------------------------------------------------------
#   PARAMETER ESTIMATION AND PARAMETRIC BOOTSTRAP LIKELIHOOD RATIO TEST
#-----------------------------------------------------------------
#  The output contains 6 objects:
#  Object 1:  "egssml": this is a list that contains the ml estimates
#  of the EGSS model, the maximized log likelihood and the AIC value
#  under that model.
#
#  Object 2:  "roussml": this is a list that contains the ml estimates
```

```
#  of the OUSS model, the maximized log likelihood and the AIC value
#  under that model.
#
#  Object 3:  "Lam.obs":  this is the observed log-likelihood ratio
#  statistic given by -2*ln[L(H0)/L(H1)].
#
#  Object 4:  "Lam.vec":  this is a vector of the bootstrapped values
#  of -2*ln[L(H0)/L(H1)].
#
#  Object 5:  "pvalue":  this is the proportion of the bootstrapped
#  values of -2*ln[L(H0)/L(H1)] that are more extreme than the observed
#  value of -2*ln[L(H0)/L(H1)].
#
#  Object 6:  "Decision.rule": this is a character object that prints
#  out the decision of the test (Fail to reject or reject the null
#  hypothesis of the EGSS density independent model).
#
#  Before doing the calculations, the user has to specify ONLY the
#  following 3 options:

# 1. How many bootstrap replicates do you want to use? (Default is
# one thousand)
NBoot = 1000

# 2. At what significance level alpha do you want to test the null
# hypothesis of density independence vs. the alternative of density
# dependence?
my.alpha = 0.05

# 3. Do you want to plot the sampling distribution of the log-
# likelihood ratio statistic -2*ln[L(H0)/L(H1)]?
plot.it = TRUE

# Performs the bootstrap analysis (be patient):
pblrt.trial = PBLRT.ddpcont(B=NBoot, Yobs=log.obs, Tvec=Time.t,
alpha=my.alpha, plot.PBLR=plot.it)

# Printing results:

#  Object 1:  "egssml": this is a list that contains the ml estimates
#  of the EGSS model, the maximized log likelihood and the AIC value
#  under that model.
pblrt.trial$egssml

#  Object 2:  "roussml": this is a list that contains the ml estimates
#  of the OUSS model, the maximized log likelihood and the AIC value
#  under that model.
pblrt.trial$roussml

#  Object 3:  "Lam.obs":  this is the observed value of the log-
#  likelihood ratio statistic -2*ln[L(H0)/L(H1)]
pblrt.trial$Lam.obs

#  Object 5:  "pvalue":  this is the proportion of the bootstrapped
#  values of -2*ln[L(H0)/L(H1)] that are more extreme than the observed
#  value of -2*ln[L(H0)/L(H1)].
pblrt.trial$pvalue
```

```
#  Object 6:  "Decision.rule": this is a character object that prints
#  out the decision of the test (Fail to reject or reject the null
#  hypothesis of density independence)
pblrt.trial$Decision.rule

#  To plot the sampling distribution of the likelihood ratio test, we
#  recommend producing a trial histogram first, and after a visual
#  inspection, eliminating from the plotting range the numerical
#  extremes, which should be very few.
#
#  The script lines below will display an example using the American
#  Redstart data set.

#  Histogram of the pboot LRT with the observed LRT in red.
hist(pblrt.trial$Lam.vec)
abline(v=pblrt.trial$Lam.obs,lwd=2,col="red")

#  After plotting, note that the American Redstart example has
#  numerical extremes around -600 and around +600.
#  Re-do the plot zooming into the bulk of the bootstrap distribution.
hist(pblrt.trial$Lam.vec[abs(pblrt.trial$Lam.vec)<100],
   xlab="Bootstrap values of -2*ln[L(H0)/L(H1)]",main="")
abline(v=pblrt.trial$Lam.obs,lwd=2,col="red")


#=========================================================================
#  ROUSSE-1.0.R:  Functions used in statistical inferences for the
#  Ornstein-Uhlenbeck state space model.  This script should be present
#  in the working directory of R before running either RUNNING-ROUSS.R
#  or RUNNING-PBLRT.R.
#=========================================================================

##  PART I)  OUSS model R functions
##  PART II) EGSS model R functions
##  PART III) Density dependence vs. Density independence
##  Parametric Bootstrap Likelihood Ratio Test (PBLRT) functions

##  All EGSS Functions originally published in
##  Humbert, J.-Y., Mills, S., Horne, J. S. and Dennis, B. 2009. A
##  better way to estimate population trend. Oikos 118:1940-1946.

# PART I):  The OUSS functions ------------------------------------

# 1. Negative log-likelihood score, for ML estimation or Model
# Selection.
#  ML objective function "negloglike.OU.ml" is negative of log-
#  likelihood.
#  The function uses a multivariate normal log-
#  likelihood (Eq. 19).  The three function arguments are:
#  fguess, vector of parameters (transformed to the real line),
#  yt, vector of time series of log observed abundances (cannot
#  have 0's ), tt, vector of observation times.
negloglike.OU.ml <- function(fguess,yt,tt){
   mu = fguess[1]
   guess = exp(fguess[2:4]) # Constrains parameters > 0
   theta = guess[1]
```

```
   betasq = guess[2]
   tausq = guess[3]
   q = length(yt) - 1
   qp1 = q+1
   Var.inf = betasq/(2*theta) # Stationary variance
   ss = tt[2:qp1] - tt[1:q]
   t.cols = matrix(rep(tt,each=qp1),nrow=qp1,ncol=qp1, byrow=FALSE)
   t.rows = t(t.cols)
   abs.diffs = abs(t.rows-t.cols)
   V = Var.inf*exp(-theta*abs.diffs)
   diag(V) = diag(V) + rep(tausq,qp1)
   mu.vec = rep(mu,qp1)
   neglogl = (qp1/2)*log(2*pi)+(1/2)*log(det(V))+
      (1/2)*(yt-mu.vec)%*%ginv(V)%*%(yt-mu.vec)
   return(neglogl)
}

#  2. Negative log-likelihood for REML estimation or Model
#  Selection (Eq. 22).
#  REML objective function "negloglike.OU.reml" is negative of
#  log-likelihood for first differences of the log-scale observations.
#  The three function arguments are:
#  phi, vector of parameters (transformed to the real line),
#  yt, vector of time series observations (log scale),
#  tt, vector of observation times.
#  The function performs the differencing.
negloglike.OU.reml=function(phi,yt,tt) {
   theta = exp(phi[1])          #  Constrains theta > 0.
   betasq = exp(phi[2])         #  Constrains betasq > 0.
   tausq = exp(phi[3])          #  Constrains tausq > 0.
   Var.inf = betasq/(2*theta)   #  Recurring quantity.
   q = length(yt)-1
   qp1 = q+1
   ss = tt[2:qp1]-tt[1:q]       #  Time intervals.
   t.cols = matrix(rep(tt,each=qp1),nrow=qp1,ncol=qp1, byrow=FALSE)
   t.rows = t(t.cols)
   abs.diffs = abs(t.rows-t.cols)
   Sigma.mat = Var.inf*exp(-theta*abs.diffs)
   Itausq = matrix(0,qp1,qp1)
   diag(Itausq) = rep(tausq,qp1)
   V = Sigma.mat+Itausq
   Dmat = cbind(-diag(1,q),matrix(0,q,1)) +
      cbind(matrix(0,q,1),diag(1,q))        #  Differencing matrix.
   Phi.mat = Dmat%*%V%*%t(Dmat)             #  REML var-cov mattrix.
   wt = yt[2:qp1]-yt[1:q]
   ofn = (q/2)*log(2*pi)+0.5*log(det(Phi.mat)) +
      0.5*wt%*%ginv(Phi.mat)%*%wt         #  ginv() is numerically
                                          #  more robust than solve().
   return(ofn)
}

# 3. rand.MVN:  Multivariate Normal random number generator:
#    n = number of random samples of a MVN vector,
#    mu = mean vector of the MVN distribution,
#    cov.mat = Variance-covariance matrix of the MVN distribution.
randmvn = function(n,mu.vec, cov.mat) {
   p = length(mu.vec)
```

```
   Tau = chol(cov.mat)
   Zmat = matrix(rnorm(n=p*n,mean=0,sd=1),nrow=p,ncol=n)
          #  generate normal deviates outside loop.
   out = matrix(0,nrow=p,ncol=n)
   for(i in 1:n) {
      Z = Zmat[,i]
      out[,i] = t(Tau)%*%Z + mu.vec
   }
   return(out)
}

# 4. Simulation function: requires as input parameter values and
#  a vector of observation times. The multivariate Normal model (see
#  eq. 19) is used to simulate the data.
#    nsims = number of bootstrap replicates to simulate.
#    parms = vector of parameter values = c(mu_hat, theta_hat,
#       betasq_hat,tausq_hat), where 'hat' denotes neither the
#       ML or the REML estimates.
#    Tvec = vector of ORIGINAL observation times (t_0, t_1, t_2, ...,
#       t_q)
ROUSS.sim = function(nsims,parms,Tvec) {
   tt = Tvec-Tvec[1]
   mu = parms[1]
   theta = parms[2]
   betasq = parms[3]
   tausq = parms[4]
   q = length(tt)-1
   qp1 = q+1
   Var.inf = betasq/(2*theta)
   ss = tt[2:qp1]-tt[1:q]
   t.cols = matrix(rep(tt,each=qp1),nrow=qp1,ncol=qp1,byrow=FALSE)
   t.rows = t(t.cols)
   abs.diffs = abs(t.rows-t.cols)
   V = Var.inf*exp(-theta*abs.diffs)
   diag(V) = diag(V)+rep(tausq,qp1)
   m.vec = rep(mu,qp1)
   out = randmvn(n=nsims,mu.vec=m.vec,cov.mat=V)
   return(out)
}

# 5. Computing rough initial guesses for ML estimation.
#     Yobs = log(Observed time series of abundances),
#     Tvec = vector of sampling times.
guess.calc = function(Yobs,Tvec) {
   T.t = Tvec-Tvec[1]       #  For calculations, time starts at zero.
   q = length(Yobs)-1       #  Number of time series transitions, q.
   qp1 = q+1                #  q+1 gets used a lot, too.
   S.t = T.t[2:qp1]-T.t[1:q]  #  Time intervals.
   Ybar = mean(Yobs)
   Yvar = sum((Yobs-Ybar)*(Yobs-Ybar))/q
   mu1 = Ybar             # Kludge an initial value for theta based
                          #  on mean of Y(t+s) given Y(t).
   th1 = -mean(log(abs((Yobs[2:qp1]-mu1)/(Yobs[1:q]-mu1)))/S.t
   bsq1 = 2*th1*Yvar/(1+2*th1)       # Moment estimate using stationary
   tsq1 = bsq1                       # variance, with betasq=tausq.
   out = c(mu1,th1,bsq1,tsq1)
   return(abs(out))
```

```
}

# 6.  Computing the ML estimates of the OUSS model, maximized log-
# likelihood and the AIC score:
#     Yobs = log(Observed time series of abundances),
#     Tvec = vector of sampling times,
#     parms.guess = vector of initial guesses for the parameters
#       = c(mu_hat,theta_hat,betasq_hat,tausq_hat)
ROUSS.ML = function(Yobs,Tvec,parms.guess) {
   tt = Tvec-Tvec[1]
   q = length(tt)-1
   qp1 = q+1
   guess.optim = c(parms.guess[1],log(parms.guess[2:4]))
   optim.out = optim(par=guess.optim,fn=negloglike.OU.ml,
      method="Nelder-Mead",yt=Yobs,tt=tt)
   mles = c(optim.out$par[1],exp(optim.out$par[2:4]))
   lnL.hat = - optim.out$value[1]
   AIC = -2*lnL.hat + 2*4       # where 4 = length(mles)...
   out = list(mles=mles, lnL.hat = lnL.hat, AIC=AIC)
   return(out)
}

# 7.  Computing the REML estimates of the OUSS model, maximized log-
# likelihood and the AIC score:
#     Yobs = log(Observed time series of abundances),
#     Tvec = vector of sampling times,
#     parms.guess = vector of initial guesses for the parameters
#      = c(mu_hat,theta_hat,betasq_hat,tausq_hat).
ROUSS.REML = function(Yobs,Tvec,parms.guess) {
   tt = Tvec-Tvec[1]
   q = length(tt)-1
   qp1 = q+1
   ss = tt[2:qp1]-tt[1:q]
   guess.optim = log(parms.guess[2:4])
   optim.out = optim(par = guess.optim,fn=negloglike.OU.reml,
      method="Nelder-Mead",yt=Yobs,tt=tt)
   remls = exp(optim.out$par)
   lnL.hat = -optim.out$value[1]
   theta.reml = remls[1]
   betasq.reml = remls[2]
   tausq.reml = remls[3]
   Var.inf = betasq.reml/(2*theta.reml)
   vx = matrix(1,qp1,qp1)
   for (ti in 1:q) {
      vx[(ti+1):qp1,ti] = exp(-theta.reml*cumsum(ss[ti:q]))
      vx[ti,(ti+1):qp1] = vx[(ti+1):qp1,ti]
   }
   Sigma.mat = vx*Var.inf
   Itausq = matrix(0,qp1,qp1)
   diag(Itausq) = rep(tausq.reml,qp1)
   V.reml = Sigma.mat+Itausq
   j = matrix(1,qp1,1)
   Vinv = ginv(V.reml)
   mu.reml = (t(j)%*%Vinv%*%Yobs)/(t(j)%*%Vinv%*%j)  #  REML estimate
                                                     # of mu.
   out = list(remls=c(mu.reml,theta.reml,betasq.reml,tausq.reml),
      lnLhat = lnL.hat)
```

```
   return(out)
}

# 8. Parameteric Bootstrap function.
# B = number of bootstrap replicates,
# parms = c(mu.mle,theta.mle,betasq.mle,tausq.mle) = ML estimates of
# the parameters with the original time series,
# tt = vector of observation times (t_0, t_1, t_2, ..., t_q) from the
# original time series.
# If REML="TRUE", then the parametric bootsrap is computed for REML
# estimation.  In that case, 'parms' must contain the REML estimates
# for the original time series.
ROUSS.pboot = function(B=2,parms,Yobs,Tvec,REML="FALSE",
                  plot.it="FALSE") {
   tt = Tvec-Tvec[1]
   nparms = length(parms)
   preds.boot = matrix(0,nrow=B,ncol=length(tt))
   if(REML=="TRUE") {
      boot.remles = matrix(0,nrow=B,ncol=nparms+1)
      all.sims = ROUSS.sim(nsims=B,parms=parms,Tvec=Tvec)
      reml.preds = ROUSS.predict(parms=parms,Yobs=Yobs,Tvec=Tvec,
                     plot.it="FALSE")[,2]
      for(b in 1:B) {
         bth.timeseries = all.sims[,b]
         remles.out = ROUSS.REML(Yobs=bth.timeseries,Tvec=Tvec,
                        parms.guess=parms)
         boot.remles[b,] = c(remles.out$remls,remles.out$lnLhat)
         preds.boot[b,] = ROUSS.predict(parms=remles.out$remls,
                             Yobs=bth.timeseries,Tvec=Tvec,
                              plot.it="FALSE")[,2]
      }
      CIs.mat = apply(boot.remles,2,FUN=function(x){quantile(x,
                  probs=c(0.025,0.975))})
      CIs.mat = rbind(CIs.mat[1,1:4],parms,CIs.mat[2,1:4])
      rownames(CIs.mat) = c("2.5%","REMLE","97.5%")
      colnames(CIs.mat) = c("mu","theta","betasq","tausq")
      preds.CIs = apply(preds.boot,2,FUN=function(x){quantile(x,
                    probs=c(0.025,0.975))})
      preds.CIs = t(rbind(Tvec,preds.CIs[1,],reml.preds,
                    preds.CIs[2,]))
      colnames(preds.CIs) = c("Year","2.5%","REMLE","97.5%")
      boot.list = list(boot.remles=boot.remles,CIs.mat=CIs.mat,
                    preds.CIs=preds.CIs)
      if(plot.it=="TRUE") {
         X11()
         par(mfrow=c(2,2))
         hist(boot.remles[,1],main=expression(hat(mu)),xlab="")
         abline(v=parms[1],lwd=2,col="red")
         hist(boot.remles[,2],main=expression(hat(theta)),
            xlab="")
         abline(v=parms[2],lwd=2,col="red")
         hist(boot.remles[,3],main=expression(hat(beta^2)),
            xlab="")
         abline(v=parms[3],lwd=2,col="red")
         hist(boot.remles[,4],main=expression(hat(tau^2)),
            xlab="")
         abline(v=parms[4],lwd=2,col="red")
```

```
      }
      return(boot.list)
   } else {
      boot.mles = matrix(0,nrow=B,ncol=nparms+2)
      all.sims = ROUSS.sim(nsims=B,parms=parms,Tvec=Tvec)
      ml.preds = ROUSS.predict(parms=parms,Yobs=Yobs,
                    Tvec=Tvec,plot.it="FALSE")[,2]
      for(b in 1:B) {
         bth.timeseries = all.sims[,b]
         mles.out = ROUSS.ML(Yobs=bth.timeseries,Tvec=Tvec,
                       parms.guess=parms)
         boot.mles[b,] = c(mles.out$mles, mles.out$lnL.hat,
                           mles.out$AIC)
         preds.boot[b,] = ROUSS.predict(parms=mles.out$mles,
                             Yobs=bth.timeseries,Tvec=Tvec,
                             plot.it="FALSE")[,2]
      }
      CIs.mat = apply(boot.mles,2,FUN=function(x){quantile(x,
                   probs=c(0.025,0.975))})
      CIs.mat = rbind(CIs.mat[1,1:4],parms,CIs.mat[2,1:4])
      rownames(CIs.mat) = c("2.5%","MLE","97.5%")
      colnames(CIs.mat) = c("mu","theta","betasq","tausq")
      preds.CIs = apply(preds.boot,2,FUN=function(x){quantile(x,
                    probs=c(0.025,0.975))})
      preds.CIs = t(rbind(Tvec,preds.CIs[1,],ml.preds,preds.CIs[2,]))
      colnames(preds.CIs) = c("Year","2.5%","MLE","97.5%")
      boot.list = list(boot.mles=boot.mles,CIs.mat=CIs.mat,
                    preds.CIs=preds.CIs)
      if(plot.it=="TRUE") {
         X11()
         par(mfrow=c(2,2))
         hist(boot.mles[,1],main=expression(hat(mu)),
            xlab="")
         abline(v=parms[1],lwd=2,col="red")
         hist(boot.mles[,2],main=expression(hat(theta)),
            xlab="")
         abline(v=parms[2],lwd=2,col="red")
         hist(boot.mles[,3],main=expression(hat(beta^2)),
            xlab="")
         abline(v=parms[3],lwd=2,col="red")
         hist(boot.mles[,4],main=expression(hat(tau^2)),
            xlab="")
         abline(v=parms[4],lwd=2,col="red")
      }
      return(boot.list)
   }                    # End if/else
}

# ROUSS.pboot(B=20,parms=linx.remles,Yobs=log(Observed.t),
#    Tvec=Time.t,REML="TRUE",plot.it="FALSE")

# 9. Function to compute the predicted trajectory of the unobserved
# process.  The arguments are:
#    parms = ML or REML estimates of c(mu,theta,betasq,tausq),
#       whichever you prefer,
#    Yobs  = Log observations,
#    Tvec  = vector of original observation times (t_0,t_1,...,t_q).
```

```
ROUSS.predict = function(parms,Yobs,Tvec,plot.it="TRUE") {
   qp1 = length(Yobs)
   q = qp1-1
   tt = Tvec - Tvec[1]
   ss = tt[2:qp1] - tt[1:q]
   mu = parms[1]
   theta = parms[2]
   betasq = parms[3]
   tausq = parms[4]
   Var.inf = betasq/(2*theta)
   t.cols = matrix(rep(tt,each=qp1),nrow=qp1,ncol=qp1,byrow=FALSE)
   t.rows = t(t.cols)
   abs.diffs = abs(t.rows-t.cols)
   Sigma.mat = Var.inf*exp(-theta*abs.diffs)
   Itausq = matrix(0,qp1,qp1)
   diag(Itausq) = rep(tausq,qp1)
   V = Sigma.mat+Itausq
   Predict.t = rep(0,qp1)
   Muvec = rep(mu,q)
   for (tj in 1:qp1) {
      Y.omitj = Yobs[-tj]   #  Omit observation at time tj.
      V.omitj = V[-tj,-tj]  #  Omit row tj and col tj from var-cov
                            #    matrix.
      V12 = V[tj,-tj]       #  Submatrix:  row tj without col tj.
      Predict.t[tj] = mu+V12%*%ginv(V.omitj)%*%(Y.omitj-Muvec)
               #  Usual expression for conditional MV normal mean.
   }
   Predict.t = exp(Predict.t)
   if(plot.it=="TRUE") {
      #  Plot the data & model-fitted values
      X11()
      plot(Time.t,exp(Yobs),xlab="Time",ylab="Population
         abundance",type="b",cex=1.5, main="Predicted (--)
         and observed (-o-) abundances")
      #  Population data are circles.
      par(lty="dashed")    #  Predicted abundances are dashed line.
      points(Tvec,Predict.t,type="l",lwd=1)
   }
   return(cbind(Tvec,Predict.t))
}

# 10. Function to run the estimation, compute the predictions and
# run a parametric bootstrap.
ROUSS.CALCS = function(Yobs,Tvec,pmethod="ML",nboot,
                  plot.pred="TRUE",plot.bootdists="TRUE") {
   # 10.1 Compute a rough guess of the parameter estimates to
   #  initialize the search:
   guesscalc = guess.calc(Yobs=log.obs,Tvec=Tvec)
   # 10.2 Compute either the ML or the REML estimates, according
   #  to what was specified in point 1 above.
   if (pmethod=="ML") {
      best.guess = ROUSS.ML(Yobs=Yobs,Tvec=Tvec,
                     parms.guess=guesscalc)
      AIC = best.guess[[3]]
      reml.option = "FALSE"
   } else if (pmethod=="REML") {
      best.guess = ROUSS.REML(Yobs=Yobs,Tvec=Tvec,
```

```
                    parms.guess=guesscalc)
      reml.option = "TRUE"
   } else {
      print("Error: ML and REML are the only options allowed for
        'method' ")
   }
   # 10.3 Parameter estimates and maximized log-likelihood
   # (will be printed at the end).
   parms.est = best.guess[[1]]
   lnLhat = best.guess[[2]]

   # 10.4 Computing the predictions.
   # rouss.preds = ROUSS.predict(parms=parms.est,
   #               Yobs=Yobs,Tvec=Tvec,plot.it=plot.pred)
   # print("These are the predictions of what the true, unobserved
   #    population abundances were")
   # print(rouss.preds)

   # 10.5 Parametric bootstrap: computing both, parameters and
   # predictions 95 % CI's.
   pboot.calcs = ROUSS.pboot(B=nboot,parms=parms.est,Yobs=Yobs,
                   Tvec=Tvec,REML=reml.option,plot.it=plot.bootdists)
   print("These are the predictions of what the true, unobserved
      population abundances were, along with 95% CI's")
   print(pboot.calcs$preds.CIs)
   rouss.preds = ROUSS.predict(parms=parms.est,Yobs=Yobs,
                  Tvec=Tvec,plot.it=plot.pred)
   if(plot.pred=="TRUE") {
      points(Tvec,pboot.calcs$preds.CIs[,2],type="l",col="blue")
      points(Tvec,pboot.calcs$preds.CIs[,4],type="l",col="blue")
   }
   print("This is the matrix of Parametric Bootstrap 95% CI's
      along with the estimates")
   print(pboot.calcs$CIs.mat)
   print("Maximized log-likelihood score")
   print(lnLhat)
   if(pmethod=="ML") {
      print("AIC score")
      print(AIC)
      out = list(parms.est=parms.est,lnLhat=lnLhat,AIC=AIC,
               pbootmat=pboot.calcs[[1]],pboot.cis=pboot.calcs[[2]],
               pboot.preds=pboot.calcs$preds.CIs)
   } else {
      out = list(parms.est=parms.est,lnLhat=lnLhat,
               pbootmat=pboot.calcs[[1]],pboot.cis=pboot.calcs[[2]],
               pboot.preds=pboot.calcs$preds.CIs)
   }
   return(out)
}

# PART II):  The EGSS functions --------------------------------

# 11.  Negative Log-Likelihood function for the EGSS model.
negloglike.EGSS.ml = function(theta,yt,tt) {
   mu = theta[1]
   sigmasq = exp(theta[2])
   tausq = exp(theta[3])
```

```
   xo = theta[4]
   q = length(yt) - 1
   qp1 = q+1
   yt = matrix(yt,nrow=qp1,ncol=1)  # makes data a matrix object.
   vx = matrix(0,qp1,qp1)
   for(i in 1:q) {
      # Create the matrix with the correct dimensions
      # instead of relying on R's automatic recycling-
      # to-match-dimensions property.
      vx[((i+1):qp1),((i+1):qp1)] = matrix(1,(qp1-i),
                                     (qp1-i))*tt[(i+1)]
   }
   Sigma.mat = sigmasq*vx
   Itausq = matrix(rep(0,(qp1*qp1)),nrow=qp1,ncol=qp1)
   diag(Itausq) = rep(tausq,qp1)
   V = Sigma.mat + Itausq
   mu.vec = matrix((xo+mu*tt),nrow=qp1,ncol=1)
   return((qp1/2)*log(2*pi) + 0.5*log(det(V)) + 0.5*(t(yt-
            mu.vec)%*%ginv(V)%*%(yt-mu.vec)))
}

#  12. Function to propose initial values of the parameters
#  to feed the parameter estimation function under the EGSS model.
init.egss = function(Yobs,Tvec) {
   q = length(Yobs)-1       #  Number of time series transitions, q.
   qp1 = q+1                #  q+1 gets used a lot, too.
   T.t = Tvec-Tvec[1]       #  For calculations, time starts at zero.
   Ybar = mean(Yobs)
   Tbar = mean(T.t)
   mu.egoe = sum((T.t-Tbar)*(Yobs-Ybar))/sum((T.t-Tbar)*(T.t-Tbar))
   x0.egoe = Ybar-mu.egoe*Tbar
   ssq.egoe = 0
   Yhat.egoe = x0.egoe+mu.egoe*T.t
   tsq.egoe = sum((Yobs-Yhat.egoe)*(Yobs-Yhat.egoe))/(q-1)
   S.t = T.t[2:qp1]-T.t[1:q]   #  Time intervals.
   Ttr = sqrt(S.t)
   Ytr = (Yobs[2:qp1]- Yobs[1:q])/Ttr
   mu.egpn = sum(Ttr*Ytr)/sum(Ttr*Ttr)
   Ytrhat = mu.egpn*Ttr
   ssq.egpn = sum((Ytr-Ytrhat)*(Ytr-Ytrhat))/(q-1)
   tsq.egpn = 0
   x0.egpn = Yobs[1]
   mu0 = (mu.egoe+mu.egpn)/2
   ssq0 = ssq.egpn/2
   tsq0 = tsq.egoe/2
   xo.out = x0.egoe
   return(c(mu0,ssq0,tsq0,xo.out))
}

#  13. Function to optimize the negative log-likelihood and return
#  parameter estimates, likelihood score and AIC, BIC value.
EGSS.ML = function(Yobs,Tvec,parms.guess) {
   tt = Tvec-Tvec[1]
   q = length(tt)-1
   qp1 = q+1
   guess.optim = c(parms.guess[1],log(parms.guess[2:3]),
                   parms.guess[4])
```

```
   optim.out = optim(par=guess.optim,fn=negloglike.EGSS.ml,
                  method="Nelder-Mead",yt=Yobs,tt=tt)
   mles = c(optim.out$par[1],exp(optim.out$par[2:3]),
             optim.out$par[4])
   lnL.hat = - optim.out$value[1]
   AIC = -2*lnL.hat + 2*4  # where 4 is the number of parameters.
   out = list(mles=mles,lnL.hat=lnL.hat,AIC=AIC)
   return(out)
}

# guess.egss = init.egss(Yobs=yt,Tvec=Time.t)
# trial.egssml = EGSS.ML(Yobs=yt,Tvec=Time.t,parms.guess=guess.egss)


#  14. Simulation from the EGSS model for parametric bootstrap:
#  simulate data using the observed time intervals and the ML
#  estimates.  The simulation is greatly eased by the fact that
#  the log observations are multivariate normally distributed.

EGSS.sim = function(nsims,parms,Tvec) {
   tt = Tvec-Tvec[1]
   mu = parms[1]
   sigmasq = parms[2]
   tausq = parms[3]
   xo = parms[4]
   q = length(tt)-1
   qp1 = q+1
   vx = matrix(0,qp1,qp1)
   for(i in 1:q) {
      vx[((i+1):qp1),((i+1):qp1)] = matrix(1,(qp1-i),(qp1-
                                        i))*tt[(i+1)]
   }
   Sigma.mat = sigmasq*vx
   Itausq = matrix(rep(0,(qp1*qp1)),nrow=qp1,ncol=qp1)
   diag(Itausq) = rep(tausq,qp1)
   V = Sigma.mat + Itausq
   mu.vec = matrix((xo+mu*tt),nrow=qp1,ncol=1)
   out = randmvn(n=nsims,mu.vec=mu.vec,cov.mat=V)
   return(out)
}

# PART III):  PBLR function --------------------------------

# 15. Function to do a PBLRT between the EGSS model (null)
# and the OUSS model (alternative).
PBLRT.ddpcont = function(B=10,Yobs,Tvec,alpha=0.05,plot.PBLR=TRUE) {
   #  Estimation under the null Ho: the EGSS model.
   guess.egss = init.egss(Yobs=Yobs,Tvec=Tvec)
   egssml = EGSS.ML(Yobs=Yobs,Tvec=Tvec,parms.guess=guess.egss)
   lnL.Ho = egssml$lnL.hat
   #  Estimation under the alternative H1:  the OUSS model
   guess.rouss  = guess.calc(Yobs=Yobs,Tvec=Tvec)
   roussml = ROUSS.ML(Yobs=Yobs,Tvec=Tvec,parms.guess=guess.rouss)
   lnL.H1 = roussml$lnL.hat
   #  Computing the observed Lam.obs = -2*ln[L(H0)/L(H1)].
      Lam.obs = -2*(lnL.Ho-lnL.H1)
   #  Simulating under H0 using the ML estimates
```

```
   mlsforboot = egssml$mles
   sims.mat = EGSS.sim(nsims=B,parms=mlsforboot,Tvec=Tvec)
   #  Looping over the simulations and maximizing the likelihood
   #  for both models each time and computing the bootstrapped
   #  values of Lam.boot = -2*ln[L(H0)/L(H1)].
   Lam.vec = rep(0,B)
   for(b in 1:B) {
      # Estimating parameters and maximizing under the null.
      Yobs.b = sims.mat[,b]
      egssml.b = EGSS.ML(Yobs=Yobs.b,Tvec=Tvec,
                  parms.guess=mlsforboot)
      lnL.Ho.b = egssml.b$lnL.hat
      # Estimating parameters and maximizing under the alternative.
      guess.rouss.b = guess.calc(Yobs=Yobs.b,Tvec=Tvec)
      roussml.b = ROUSS.ML(Yobs=Yobs.b,Tvec=Tvec,
                     parms.guess=guess.rouss.b)
      lnL.H1.b = roussml.b$lnL.hat
      # Computing the boostrapped likelihood ratio.
      Lam.boot = -2*(lnL.Ho.b-lnL.H1.b)
      Lam.vec[b] = Lam.boot
   }
   #  Computing the proportion of the simulations that have a
   #  more extreme Lamb.boot value than the observed Lamb.obs.
   pval = sum(Lam.vec>Lam.obs)/B
   Decision.rule = "Fail to Reject Density Independence"
   if(pval<alpha) {
      Decision.rule = "Reject Density Independence"
   }
   #  Return the vector of Lam.boot values and
   #  a plot if plot.PBLR=TRUE
   out = list(egssml=egssml,roussml=roussml,Lam.obs=Lam.obs,
            Lam.vec=Lam.vec,pvalue=pval,Decision.rule=Decision.rule)
   if(plot.PBLR==TRUE) {
      hist(Lam.vec,main="P-boot distribution of -2*ln[L(H0)/L(H1)]")
      abline(v=Lam.obs,col="red")
   }
   return(out)
}
```