```
#  Dennis-Taper Bootstrap Likelihood Ratio, version date 101207.R:
#
#  Program to fit stochastic density dependent population model of Dennis and
#  Taper (1994) to time series observations of population abundances.  Program
#  calculates bootstrap likelihood ratio (LR) hypothesis test of
#       H0: stochastic exponential growth
#             vs.
#       H1: stochastic density dependent growth.
#
#  User provides time series data (population abundances), confidence interval
#  coverage rate, and number of bootstrap data sets.
#
#  This program was written by Brian Dennis, Dept Fish and Wildlife Resources,
#  University of Idaho, Moscow, Idaho, 83844-1136 USA.
#
#  Citation:
#  Dennis, B. and M. L. Taper.  1994.  Density dependence in time series
#  observations of natural populations:  estimation and testing.  Ecological
#  Monographs 64:205-224.
#
#---------------------------------------------------------------------
#       USER INPUT SECTION
#---------------------------------------------------------------------
#  User enters time series data here:
n=c(33, 36, 34, 39, 35, 34, 38, 36, 37, 41, 39, 51, 47, 57, 48, 60, 65,
   74, 69, 65, 57, 70, 81, 99, 99, 105, 112, 131, 132, 139, 118, 127);
#  User can substitute R statements to read a vector "n" of abundances
#  from a file.  These data are a 3-yr running sum of the numbers of
#  unduplicated grizzly bear females observed with cubs-of-the-year,
#  1973-2006, in the greater Yellowstone ecosystem (Table 4 from Schwartz,
#  C.C., M. A. Haroldson, and K. West, eds.  2006.  Yellowstone grizzly
#  bear investigations:  annual report of the Interagency Grizzly Bear
#  Study Team, 2006. US Geological Survey, Bozeman, Montana, USA).  First
```

```
#  observation is sum of 1973-74-75 counts, second is 1974-75-76, etc.
#  Compare with Table 3 of Dennis and Taper (1994), which ends at 1991 and
#  used an unofficial count for 1990 reported by IGBST via phone.
#
#  User sets these initial values:
alpha=0.05;   # CI coverage rate is 100(1 - alpha)%.
bootn=2000;   # Number of bootstrap samples.

#-----------------------------------------------------------------------
#       INITIALIZATION OF PROGRAM VARIABLES
#-----------------------------------------------------------------------
q=length(n)-1;
n1=n[1:q];         #  Lagged abundances.
n2=n[2:(q+1)];     #  Present abundances.
x0=log(n1[1]);     #  Log-initial abundance.
nb1.NP=matrix(0,bootn,(q+1));   #  Will contain nonparametric and
nb1.NP[,1]=nb1.NP[,1]+exp(x0);  #    parametric bootstrap data for
nb1.P=nb1.NP;                   #    DD model confidence intervals.

m=matrix(1,2,(q+1));  #  Will contain one-step predictions for plots.
m[,1]=n[1]*m[,1];     #  Initial size.

ggstar.NP=matrix(0,1,bootn);   #  Repositories for
ggstar.P=matrix(0,1,bootn);    #    bootstrap test
param.NP=matrix(0,3,bootn);    #    statistics and
param.P=matrix(0,3,bootn);     #    parameter estimates.
ttstar.P=matrix(0,3,bootn);
ttstar.NP=matrix(0,3,bootn);

#  Fit DI and DD models.  Calculate G-squared.
#
#  DI model:  log(N(t)) = log(N(t-1)) + a0 + sqrt(ss0)*Z(t)
#               with Z(t) ~ normal(0,1)
```

```
#  DI model:  log(N(t)) = log(N(t-1)) + a1 + b1*N(t-1) + sqrt(ss1)*Z(t)
#              with Z(t) ~ normal(0,1)
#

y=log(n2/n1);      #  Changes in log-abundance.
nm=sum(n1)/q;      #  Sample mean of the lagged abundnaces.
ym=sum(y)/q;       #  Sample mean of the log-abundance changes.
a0hat=ym;                  #  ML estimate of a0.
r0=y-a0hat;                #  Residuals for H0 model.
ss0hat=sum(r0*r0)/q;          #  ML estimate of ss0.
s0hat=sqrt(q*ss0hat/(q-1));  #  Estimate of sqrt(ss0) uses unbiased estimate
                      #    of ss0 for bootstrap simulations.
b1hat=sum((y-ym)*(n1-nm))/sum(((n1-nm)*(n1-nm)));  #  ML estimate of b1.
a1hat=ym-b1hat*nm;            #  ML estimate of a1.
r1=y-a1hat-b1hat*n1;          #  Residuals for H1 model.
ss1hat=sum(r1*r1)/q;          #  ML estimate of ss1.
s1hat=sqrt(ss1hat);           #  ML estimate of sqrt(ss1).
gg=q*log(ss0hat/ss1hat);       #  G-squared likelihood ratio statistic.
tt=b1hat*sqrt((q-2)*sum((n1-nm)*(n1-nm))/(q*ss1hat)); #  T statistic for b1
                                      #    (T-squared is
                                      #    monotone function
                                      #    of G-squared)

if (b1hat>0) gg=-1;  #  Setting G-squared equal to -1 when b1hat>0 will
                #    ensure that the alternative hypothesis is one-sided
                #    in the form H1:  b1<0.

#-----------------------------------------------------------------
#       BOOTSTRAPPING SECTION.
#-----------------------------------------------------------------
#  Generate bootstrap samples from fitted DI model (H0),
#  using nonparametric (NP) and parametric (P) methods.
#  Nonparametric (NP) method resamples residuals for "noise",
```

```
#  while parametric (P) method generates noise from parametrically
#  estimated normal distribution.
#
#  Fit both models to each bootstrap sample and calculate bootstrap
#  values of G-squared.

#  Generate all noise outside the "for loop" for speed.
eb0.NP=matrix(sample(r0,bootn*q,replace=TRUE),bootn,q);  #  NP noise from H0.
eb0.P=matrix(rnorm(bootn*q,0,s0hat),bootn,q);            #  P noise from H0.

eb1.NP=matrix(sample(r1,bootn*q,replace=TRUE),bootn,q);  #  NP noise from H1.
eb1.P=matrix(rnorm(bootn*q,0,s1hat),bootn,q);            #  P noise from H1.

#  Produce bootstrap samples for H0.  Looping is not needed, as
#  the calculations can be vectorized by using cumulative sums
#  of the growth rates.
yb0.NP=a0hat+eb0.NP;   #  Stochastic growth rates, NP noise.
yb0.P=a0hat+eb0.P;     #  Stochastic growth rates, P noise.

xb0.NP=cbind(x0*matrix(1,bootn,1), t(apply(yb0.NP,1,cumsum))+x0); #  Bootstrap
xb0.P=cbind(x0*matrix(1,bootn,1), t(apply(yb0.P,1,cumsum))+x0);   #    samples.

nb0.NP=exp(xb0.NP[,1:q]);  #  Bootstrap samples on original abundance scale.
nb0.P=exp(xb0.P[,1:q]);    #    Last column is not needed and is left off.

for (ti in 1:q)   #  Loop to recursively generate bootstrap
  {               #    data from H1 model, and deterministic
                  #    trajectories from H0 and H1 for plotting.
  nb1.NP[,ti+1]=nb1.NP[,ti]*exp(a1hat+b1hat*nb1.NP[,ti]+eb1.NP[,ti]);
  nb1.P[,ti+1]=nb1.P[,ti]*exp(a1hat+b1hat*nb1.P[,ti]+eb1.P[,ti]);
  m[1,ti+1]=n[ti]*exp(a0hat);              #  One-step predictions, H0.
  m[2,ti+1]=n[ti]*exp(a1hat+b1hat*n[ti]); #  One-step predictions, H1.
  }
```

```
nb12.NP=nb1.NP[,2:(q+1)];   #  Present NP bootstrap abundances.
nb12.P=nb1.P[,2:(q+1)];     #  Present P bootstrap abundances.

nb11.NP=nb1.NP[,1:q];       #  Lagged NP bootstrap abundances.
nb11.P=nb1.P[,1:q];         #  Lagged NP bootstrap abundances.

yb1.NP=log(nb12.NP/nb11.NP);   #  Changes in log-abundance, NP bootstrap data.
yb1.P=log(nb12.P/nb11.P);      #  Changes in log-abundance, P bootstrap data.

for (bi in 1:bootn)   #  Loop to fit H0 and H1 models to each row of
{                     #    bootstrap data.  (Can this part be further
                      #    vectorized?)
  #  Fit H0 model to data generated under H0.
  nb0m.NP=sum(nb0.NP[bi,])/q;   #  Sample mean of NP lagged abundances.
  nb0m.P=sum(nb0.P[bi,])/q;     #  Sample mean of P lagged abundances

  yb0m.NP=sum(yb0.NP[bi,])/q; #  Sample mean of the NP log-abundance changes.
  yb0m.P=sum(yb0.P[bi,])/q;   #  Sample mean of the P log-abundance changes.

  a0star.NP=yb0m.NP;    #  ML estimate of a0, NP bootstrap data.
  a0star.P=yb0m.P;      #  ML estimate of a0, P bootstrap data.

  ss0star.NP=sum((yb0.NP[bi,]-a0star.NP)*(yb0.NP[bi,]-a0star.NP))/q;  #  ss0
  ss0star.P=sum((yb0.P[bi,]-a0star.P)*(yb0.P[bi,]-a0star.P))/q;       #  ss0

  #  Fit H1 model to data generated under H0.
  b1star.NP=sum((yb0.NP[bi,]-yb0m.NP)*(nb0.NP[bi,]-nb0m.NP))/
    sum((nb0.NP[bi,]-nb0m.NP)*(nb0.NP[bi,]-nb0m.NP));         #  b1
  b1star.P=sum((yb0.P[bi,]-yb0m.P)*(nb0.P[bi,]-nb0m.P))/
    sum((nb0.P[bi,]-nb0m.P)*(nb0.P[bi,]-nb0m.P));             #  b1

  a1star.NP=yb0m.NP-b1star.NP*nb0m.NP;   #  a1
```

```
a1star.P=yb0m.P-b1star.P*nb0m.P;       #  a1

ss1star.NP=sum((yb0.NP[bi,]-a1star.NP-b1star.NP*nb0.NP[bi,])*
  (yb0.NP[bi,]-a1star.NP-b1star.NP*nb0.NP[bi,]))/q;          #  ss1
ss1star.P=sum((yb0.P[bi,]-a1star.P-b1star.P*nb0.P[bi,])*
  (yb0.P[bi,]-a1star.P-b1star.P*nb0.P[bi,]))/q;              #  ss1

ggstar.NP[1,bi]=q*log(ss0star.NP/ss1star.NP);    #  G-squared, NP.
ggstar.P[1,bi]=q*log(ss0star.P/ss1star.P);       #  G-squared, P.
ttstar.NP[1,bi]=b1star.NP*sqrt((q-2)*sum((nb0.NP[bi,]-nb0m.NP)*
  (nb0.NP[bi,]-nb0m.NP))/(q*ss1star.NP));                  #  T statistic
ttstar.P[1,bi]=b1star.P*sqrt((q-2)*sum((nb0.P[bi,]-nb0m.P)*
  (nb0.P[bi,]-nb0m.P))/(q*ss1star.P));                     #  T statistic

if (b1star.NP>0) ggstar.NP[1,bi]=-1;   #  One-sided test, NP.
if (b1star.P>0) ggstar.P[1,bi]=-1;     #  One-sided test, P.

#  Fit H1 model to data generated under H1, for confidence intervals.
nb1m.NP=sum(nb11.NP[bi,])/q;   #  Sample mean of lagged abundances
nb1m.P=sum(nb11.P[bi,])/q;     #  Sample mean of legged abundances

yb1m.NP=sum(yb1.NP[bi,])/q;  #  Sample mean of the NP log-abundance changes.
yb1m.P=sum(yb1.P[bi,])/q;    #  Sample mean of the P log-abundance changes.

b1boot.NP=sum((yb1.NP[bi,]-yb1m.NP)*(nb11.NP[bi,]-nb1m.NP))/
  sum((nb11.NP[bi,]-nb1m.NP)*(nb11.NP[bi,]-nb1m.NP));          #  b1
b1boot.P=sum((yb1.P[bi,]-yb1m.P)*(nb11.P[bi,]-nb1m.P))/
  sum((nb11.P[bi,]-nb1m.P)*(nb11.P[bi,]-nb1m.P));              #  b1

a1boot.NP=yb1m.NP-b1boot.NP*nb1m.NP;     #  a1
a1boot.P=yb1m.P-b1boot.P*nb1m.P;         #  a1

ss1boot.NP=sum((yb1.NP[bi,]-a1boot.NP-b1boot.NP*nb11.NP[bi,])*
```

```
    (yb1.NP[bi,]-a1boot.NP-b1boot.NP*nb11.NP[bi,]))/q;          #  ss1
  ss1boot.P=sum((yb1.P[bi,]-a1boot.P-b1boot.P*nb11.P[bi,])*
    (yb1.P[bi,]-a1boot.P-b1boot.P*nb11.P[bi,]))/q;              #  ss1

  param.NP[,bi]=matrix(c(a1boot.NP, b1boot.NP, ss1boot.NP),3,1); #  NP params.
  param.P[,bi]=matrix(c(a1boot.P, b1boot.P, ss1boot.P),3,1);    #  P params.
}

#---------------------------------------------------------------------
#       CALCULATE P-VALUES, CONFIDENCE INTERVALS.
#---------------------------------------------------------------------
pval.NP=sum(gg<=ggstar.NP)/bootn;   #  G-squared:  bootstrap P-value, NP.
pval.NPlo=pval.NP-1.96*sqrt(pval.NP*(1-pval.NP)/bootn);  #  Approx 95% CI for
pval.NPhi=pval.NP+1.96*sqrt(pval.NP*(1-pval.NP)/bootn);  #    the bootstrap
pval.NPci=c(pval.NPlo, pval.NPhi);                       #    P-value.

pval.P=sum(gg<=ggstar.P)/bootn;     #  G-squared:  bootstrap P-value, P.
pval.Plo=pval.P-1.96*sqrt(pval.P*(1-pval.P)/bootn);      #  Approx 95% CI for
pval.Phi=pval.P+1.96*sqrt(pval.P*(1-pval.P)/bootn);      #    the bootstrap
pval.Pci=c(pval.Plo, pval.Phi);                          #    P-value.

ttpval.NP=sum(tt>=ttstar.NP)/bootn;   #  T-statistic:  bootstrap P-value, NP.
ttpval.NPlo=ttpval.NP-1.96*sqrt(ttpval.NP*(1-ttpval.NP)/bootn);  #  Approx 95%
ttpval.NPhi=ttpval.NP+1.96*sqrt(ttpval.NP*(1-ttpval.NP)/bootn);  #    CI for
ttpval.NPci=c(ttpval.NPlo, ttpval.NPhi);                         #    P-value.

ttpval.P=sum(tt>=ttstar.P)/bootn;     #  T-statistic:  bootstrap P-value, P.
ttpval.Plo=ttpval.P-1.96*sqrt(ttpval.P*(1-ttpval.P)/bootn);      #  Approx 95%
ttpval.Phi=ttpval.P+1.96*sqrt(ttpval.P*(1-ttpval.P)/bootn);      #    CI for
ttpval.Pci=c(ttpval.Plo, ttpval.Phi);                            #    P-value.

a1.NP=sort(param.NP[1,]);   #  Sort the bootstrap NP parameters
b1.NP=sort(param.NP[2,]);   #    from smallest to largest, in
```

```
ss1.NP=sort(param.NP[3,]);  #    preparation for calculating CIs.

a1.P=sort(param.P[1,]);     #  Sort the bootstrap P parameters
b1.P=sort(param.P[2,]);     #    from smallest to largest, in
ss1.P=sort(param.P[3,]);    #    preparation for calculating CIs.

a1.NPlo=a1.NP[floor((alpha/2)*bootn)];     #  100*(alpha/2)th percentiles
b1.NPlo=b1.NP[floor((alpha/2)*bootn)];     #    for lower end of bootstrap
ss1.NPlo=ss1.NP[floor((alpha/2)*bootn)];   #    CIs, NP.

a1.NPhi=a1.NP[ceiling((1-alpha/2)*bootn)];    #  100*(1-alpha/2)th percentiles
b1.NPhi=b1.NP[ceiling((1-alpha/2)*bootn)];    #    for higher end of bootstrap
ss1.NPhi=ss1.NP[ceiling((1-alpha/2)*bootn)];  #    CIs, NP.

a1.Plo=a1.P[floor((alpha/2)*bootn)];       #  100*(alpha/2)th percentiles
b1.Plo=b1.P[floor((alpha/2)*bootn)];       #    for lower end of bootstrap
ss1.Plo=ss1.P[floor((alpha/2)*bootn)];     #    CIs, P.

a1.Phi=a1.P[ceiling((1-alpha/2)*bootn)];      #  100*(1-alpha/2)th percentiles
b1.Phi=b1.P[ceiling((1-alpha/2)*bootn)];      #    for higher end of bootstrap
ss1.Phi=ss1.P[ceiling((1-alpha/2)*bootn)];    #    percentiles, P.

a1.NPci=c(a1.NPlo, a1.NPhi);      #  Assemble the confidence intervals
a1.Pci=c(a1.Plo, a1.Phi);         #
b1.NPci=c(b1.NPlo, b1.NPhi);      #
b1.Pci=c(b1.Plo, b1.Phi);         #
ss1.NPci=c(ss1.NPlo, ss1.NPhi);   #
ss1.Pci=c(ss1.Plo, ss1.Phi);      #

#---------------------------------------------------------------------
#       PRINT THE RESULTS TO THE CONSOLE.
#       User can alter the program to send these quantities to a file.
#---------------------------------------------------------------------
```

```
alpha;
bootn;
"Parameter estimates for density independent model";
a0hat;
ss0hat;
"Parameter estimates for density dependent model";
"NP:  nonparametric bootstrap.  P:  parametric bootstrap."
a1hat;
a1.NPci;
a1.Pci
b1hat;
b1.NPci;
b1.Pci;
ss1hat;
ss1.NPci;
ss1.Pci;
"Estimated deterministic equilibrium abundance";
-a1hat/b1hat;
"Density dependence hypothesis test using g-squared";
gg;
pval.NP;
pval.NPci;
pval.P;
pval.Pci;

"Density dependence hypothesis test using t";
tt;
ttpval.NP;
ttpval.NPci;
ttpval.P;
ttpval.Pci;

#  Plot the data
```

```
plot(n,xlab="time",ylab="population abundance",type="o",pch=1,cex=1.5);
par(lty="dashed");              #  One-steps from H0 are dashed line
points(m[1,], type="l", lwd=1);
par(lty="solid");               #  One-steps from H1 are solid line
points(m[2,], type="l", lwd=1);
```