```
#  Gompertz State Space model version date 120815.R:
#
#  Program for calculating maximum likelihood (ML) or restricted maximum
#  likelihood (REML) estimates of unknown parameters for the Gompertz State
#  Space (GSS) model of stochastic population growth.  The model is
#
#  X(t) = a + c*X(t-1) + E(t)
#               with E(t) ~ normal(0,ssq),
#  Y(t) = X(t) + F(t)
#               with F(t) ~ normal(0,tsq).
#
#  Here X(t) is log-population abundance, Y(t) is observed or estimated value
#  of X(t), a, c, ssq, tsq are parameters.  The parameter ssq is the variance
#  of the process noise, and tsq is the variance of the observation error.
#  The model takes population abundance N(t) = exp(X(t)) to be governed by a
#  stochastic, density dependent Gompertz model, with the observed abundances
#  O(t) = N(t)*exp(F(t)) arising from lognormal sampling error.
#
#  User provides time series of observed population abundances o(0), o(1),
#  ..., o(q), which are log-transformed by the program into y(0), y(1), ...,
#  y(q), assumed to be a time series realization of Y(t).  Likelihood
#  function of y(0), y(1), ..., y(q) is that of a multivariate normal
#  distribution, with initial value assumed to arise from the stationary
#  of Y(t).
#
#  User must provide initial parameter values as well.  The program should
#  be re-run for several sets of initial values, as the likelihood function
#  for the model frequently has multiple local maxima.
#
#  An alternative program, in SAS, is available as an online appendix to
#  Dennis et al. (2006).
#
#  This program written by Brian Dennis (Dept Fish and Wildlife Resources,
#  Univ Idaho, Moscow, Idaho, 83844-1136 USA) and is based on an R program
#  written by Jose Miguel Ponciano and an earlier MATLAB program written
#  by Brian Dennis.
#
#  Citation:  Dennis, B., J. M. Ponciano, S. R. Lele, M. L. Taper, and D. F.
#  Staples.  2006.  Estimating density dependence, process noise, and
#  observation error.  Ecological Monographs 76:323-341.
#
#------------------------------------------------------------------------
#        USER INPUT SECTION
#------------------------------------------------------------------------
#  User must supply initial parameter values here.
a0=.4             #  Initial value of a
c0=.8             #  Initial value of c
ssq0=.1           #  Initial value of ssq
tsq0=.2           #  Initial value of tsq

#  User supplies time series data here.  User can substitute R statements to
#    read population abundance data from a file into the vector "Observed.t".
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)

#  Example data are American Redstart counts from North American Breeding
#    Bird Survey, record # 02014 3328 08636, 1966-95 (Table 1 in Dennis
```

```
#    et al. 2006).

#-----------------------------------------------------------------------
#        PROGRAM INITIALIZATION SECTION
#-----------------------------------------------------------------------
library(MASS)  #  loads miscellaneous functions (ginv, etc.)

Y.t=log(Observed.t)  #  Log-transform the observations.
q=length(Y.t)-1      #  Number of time series transitions, q.
qp1=q+1              #  q+1 gets used a lot, too.
m=rep(1,qp1)  #  Will contain Kalman means for Kalman calculations.
v=rep(1,qp1)  #  Will contain variances for Kalman calculations.

#-----------------------------------------------------------------------
#        SECTION FOR DEFINING ML & REML LOG-LIKELIHOODS
#-----------------------------------------------------------------------

#  ML objective function "negloglike.ml" is negative of log-likelihood
#  the Nelder-Mead optimization routine in R, "optim", is a minimization
#  routine.  The ML objective function uses equations 14-17 from Dennis et
#  al. (2006).  The two function arguments are:  theta, vector of parameters
#  (transformed to the real line), and yt, vector of time series
observations.
negloglike.ml=function(theta,yt)
{
   aa=exp(theta[1])           #  Constrains a > 0.
   cc=2*exp(-exp(theta[2]))-1 #  Constrains -1 < c < 1.
   sigmasq=exp(theta[3])      #  Constrains ssq > 0.
   tausq=exp(theta[4])        #  Constrains tsq > 0.
   q=length(yt)-1
   qp1=q+1
   yt=matrix(yt,nrow=qp1,ncol=1)
   Sigma.mat=toeplitz(c(1,cumprod(rep(cc,q))))*(sigmasq/(1-(cc^2)))
   Itausq=matrix(rep(0,(qp1*qp1)),nrow=q+1,ncol=q+1)
   diag(Itausq)=rep(tausq,q+1)
   V=Sigma.mat+Itausq
   mu=matrix((aa/(1-cc))*rep(1,qp1),nrow=qp1,ncol=1)
   ofn=((qp1)/2)*log(2*pi)+(0.5*log(det(V)))+
      (0.5*(t(yt-mu)%*%ginv(V)%*%(yt-mu)))
   return(ofn)
}

#  REML objective function "negloglike.reml" is negative of log-likelihood
#  for first differences of the log-scale observations.  The REML objective
#  function uses equations 27-30 & 41) from Dennis et al. (2006).  The two
#  function arguments are:  theta, vector of parameters (transformed to the
#  real line), and wt, vector of differenced time series observations.
negloglike.reml=function(theta,wt)
{
   cc=2*exp(-exp(theta[1]))-1  #  Constrains -1 < c < 1.
   sigsq=exp(theta[2])         #  Constrains ssq > 0.
   tausq=exp(theta[3])         #  Constrains tsq < 0.
   q=length(wt)
   Sig.mat=toeplitz(c(1,cumprod(rep(cc,q))))*(sigsq/(1-(cc^2)))
   Dmat=toeplitz(c(-1,1,rep(0,q)))
   for(i in 1:q){Dmat[i+1,i]<-0}
   qp1=q+1
```

```
   Dmat=Dmat[1:q,1:qp1]
   Itausq=matrix(rep(0,(qp1*qp1)),nrow=q+1,ncol=q+1)
   diag(Itausq)=rep(tausq,q+1)
   Phi.mat=Dmat%*%(Sig.mat+Itausq)%*%t(Dmat)
   ofn=(q/2)*log(2*pi)+(0.5*log(det(Phi.mat)))+
      (0.5*(wt%*%ginv(Phi.mat)%*%wt))
   return(ofn)
}

#-------------------------------------------------------------------------
#        SECTION FOR CALCULATING ML & REML PARAMETER ESTIMATES
#-------------------------------------------------------------------------

# The ML estimates.
GSSml=optim(par=c(log(a0),log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
   negloglike.ml,NULL,method="Nelder-Mead",yt=Y.t)
GSSml=c(exp(GSSml$par[1]),2*exp(-exp(GSSml$par[2]))-1,exp(GSSml$par[3]),
   exp(GSSml$par[4]))
a.ml=GSSml[1]             # These are the ML estimates.
c.ml=GSSml[2]             #          --
ssq.ml=GSSml[3]           #          --
tsq.ml=GSSml[4]           #          --

# The REML estimates.
W.t=Y.t[2:(q+1)]-Y.t[1:q]  #  Calculate the first differences.
GSSreml=optim(par=c(log(-log((c0+1)/2)),log(ssq0),log(tsq0)),
   negloglike.reml,NULL,method="Nelder-Mead",wt=W.t)
GSSreml=c(2*exp(-exp(GSSreml$par[1]))-1,exp(GSSreml$par[2]),
   exp(GSSreml$par[3]))
c.reml=GSSreml[1]       #  These are the REML estimates.
ssq.reml=GSSreml[2]     #           --
tsq.reml=GSSreml[3]     #           --

#  Calculate the estimated var-cov matrix and then calculate
#    the REML estimate of 'a'
Sigmat.reml=toeplitz(c(1,cumprod(rep(c.reml,q))))*(ssq.reml/(1-(c.reml^2)))
IXtsq=matrix(rep(0,(qp1*qp1)),nrow=qp1,ncol=qp1)
diag(IXtsq)=rep(tsq.reml,qp1)
Psimat.reml=Sigmat.reml+IXtsq
Psiinv.reml=ginv(Psimat.reml)
jvec=rep(1,qp1)
a.reml=(jvec%*%Psiinv.reml%*%Y.t)/(jvec%*%Psiinv.reml%*%jvec)
a.reml=a.reml*(1-c.reml)  #  Equation 42 from Dennis et al. (2006).


#  Calculate estimated population sizes with Kalman filter, for plotting.
#  Choose ML or REML estimates here.
#  aa=a.ml; cc=c.ml; ssq=ssq.ml; tsq=tsq.ml;
aa=a.reml; cc=c.reml; ssq=ssq.reml; tsq=tsq.reml;

m[1]=aa/(1-cc)            #  Stationary mean of Y(t).
v[1]=ssq/(1-cc*cc)+tsq    #  Stationary variance of Y(t).

for (ti in 1:q)   #  Loop to generate estimated population abundances
{                 #    using Kalman filter (see equations 6 & 7, Dennis et
                  #    al. (2006).
   m[ti+1]=aa+cc*(m[ti]+((v[ti]-tsq)/v[ti])*(Y.t[ti]-m[ti]))
```

```
   v[ti+1]=cc*cc*tsq*((v[ti]-tsq)/v[ti])+ssq+tsq
}

#  The following statement calculates exp{E[X(t) | Y(t), Y(t-1),...,Y(0)]}
#    see equation 54 in Dennis et al. (2006).
Predict.t=exp(m+((v-tsq)/v)*(Y.t-m))

#  Plot the data & model-fitted values
plot(Observed.t,xlab="time",ylab="population abundance",
   type="o",pch=1,cex=1.5)      #  Population data are circles.
par(lty="dashed")               #  Estimated abundances are dashed line.
points(Predict.t, type="l", lwd=1)

#  Print the parameter estimates
parms.reml=c(a.reml,c.reml,ssq.reml,tsq.reml)  #  Gather stuff for printing
parms.ml=c(a.ml,c.ml,ssq.ml,tsq.ml)            #          --
names=c("a","c","ssq","tsq")                   #          --
types=c("ML","REML")                           #          --
matrix(cbind(parms.ml,parms.reml),nrow=2,ncol=4,byrow=TRUE,
   dimnames=list(types,names))                 #  Print the results
```