Technical Documentation
for
ASAP Version 3.0

NOAA Fisheries Toolbox

September 2012

Table of Contents

Introduction

ASAP3 is an update to the program ASAP (Legault and Restrepo 1998), which was previously updated as ASAP2 in 2008. It contains a number of new features and options that are described in the ASAP3 User's Guide. This document provides the basic equations used in the program along with the approaches used to fit different components of the objective function. More importantly, it contains the actual ADMB code used to generate the executable, so that the exact calculations in the program can be followed. This document uses variable names in a number of places instead of symbols to facilitate understanding of the underlying code.

Basic Equations

The description of the model follows the steps in the code for ease of understanding. Calculation of the objective function is described in the next section.

Spawning Stock Biomass

The spawning stock biomass is calculated based on the population abundance at age (N), the fecundity (Φ), and the proportion of the total mortality (Z, see mortality section below) during the year prior to spawning (p_{SSB}) as

$$SSB_t = \sum_a N_{t,a}\Phi_{t,a}e^{-p_{SSB}Z_{t,a}} \quad (1)$$

Where the fecundity matrix is either input by the user or else derived as the element by element product of the weight at age matrix and the maturity matrix.

Stock Recruitment Relationship

The Beverton and Holt stock recruitment relationship is used to calculate the expected recruitment in year t+1 from the spawning stock biomass in year t as

$$\hat{R}_{t+1} = \frac{\alpha SSB_t}{\beta + SSB_t} \quad (2)$$

The equation is reparameterized following Mace and Doonan (ref) to use two parameters: the SR scaler and steepness (τ). The SR scaler can be either unexploited spawning stock biomass (SSB_0) or unexploited recruitment (R_0). These two values are related to each other based on the unexploited spawners per recruit (SPR_0) as $SPR_0 = SSB_0/R_0$. All three of these unexploited values are computed using the natural mortality, weights at age, and maturity (or fecundity) values in the terminal year of the assessment. The stock recruitment relationship is therefor fixed for all years using equation 2 with

$$\alpha = \frac{4\tau(SSB_0 / SPR_0)}{5\tau - 1} \quad and \quad \beta = \frac{SSB_0(1-\tau)}{5\tau - 1} \quad (3)$$

However, the program also produces the values of unexploited SSB, R, spawners per recruit, and steepness associated with the natural mortality rate, weights at age, and maturity (or fecundity) for each year in the time series. This allows the user to see the influence of these values on the stock recruitment parameters SSB_0, R_0, SPR_0, and τ over time.

Steepness for the Beverton and Holt stock recruitment relationship is only defined between 0.2 and 1.0. Fixing steepness at 1.0 makes expected recruitment constant. The actual recruitment estimated by the model is formed by multiplying the expected recruitment by a recruitment deviation. The recruitment deviations are assumed to follow a lognormal distribution, making the parameters log_Rdev_t. The parameters are estimated as a bounded vector, meaning their sum is zero, so that they are centered on the expected stock recruitment relationship. The population numbers at age 1, recruitment is always assumed to occur at age 1, are

$$N_{t,1} = R_t e^{\log\_Rdev_t} \tag{4}$$

Selectivity

The approach used to estimate fleet selectivity in ASAP3 is quite different from that in ASAP, but the same as in ASAP2. As before, there are selectivity blocks, but now they are defined independently for each fleet. Within each selectivity block, there are three options for estimating selectivity:

1. estimate parameters for each age (one parameter for each age, similar to ASAP in concept, but now each age is bounded by zero and one and at least one age should be fixed at 1.0 instead of estimated)
2. logistic function (2 parameters: α_1, β_1)

$$Sel_a = \frac{1}{1+e^{-(a-\alpha_1)/\beta_1}} \tag{5}$$

3. double logistic (4 parameters: α_1, β_1, α_2, β_2)

$$Sel_a = \left(\frac{1}{1+e^{-(a-\alpha_1)/\beta_1}}\right)\left(1-\frac{1}{1+e^{-(a-\alpha_2)/\beta_2}}\right) \tag{6}$$

The selectivity at age is then assigned to all fleet and year combinations within that block. Note that for options 2 and 3, the selectivity at age is divided by the maximum value over all ages, creating the final selectivity vector with maximum of 1.0 for that block.

Mortality

Natural mortality (M) is entered as a year by age matrix, as it was in ASAP2, instead of just a vector by age as it was in ASAP.

Fishing mortality (F) is assumed to be separable, meaning it is the product of a year effect ($Fmult$) and selectivity at age (described above). The $Fmult$ for a fleet and year is determined by two sets of parameters, log_Fmult1_{ifleet}, the parameter for first year for that fleet, and $log\_Fmultdev_{ifleet,t}$, where t=2 to the number of years, the deviation of the parameter from the value in the first year for that fleet. Both sets of parameters are estimated in log space and then exponentiated as

$$Fmult_{ifleet,1} = e^{\log\_Fmult1_{ifleet}}$$

$$Fmult_{ifleet,t} = Fmult_{ifleet,1} e^{\log\_Fmultdev_{ifleet,t}} \quad \forall t \geq 2 \tag{7}$$

Note that the *log_Fmultdev* parameters are not estimated as a dev_vector in the ADMB code, and so fishing intensity can increase continually, decrease continually, or fluctuate throughout the time series. The directed F for a fleet, year, and age, meaning that portion of the F that contributes to landings, is computed using the separable equation along with the proportion of catch released for that fleet, year, and age ($prop\_release_{ifleet,t,a}$) as

$$Fdir_{ifleet,t,a} = Fmult_{ifleet,t,a} Sel_{ifleet,t,a} (1 - prop\_release_{ifleet,t,a}) \tag{8}$$

The bycatch F contains an additional component, the proportion of released fish that die, which is fleet specific ($release\_mort_{ifleet}$)

$$Fbycatch_{ifleet,t,a} = Fmult_{ifleet,t,a} Sel_{ifleet,t,a} prop\_release_{ifleet,t,a} release\_mort_{ifleet} \tag{9}$$

The two parts are then added together to produce the fishing mortality for the fleet, year and age

$$F_{ifleet,t,a} = Fdir_{ifleet,t,a} + Fbycatch_{ifleet,t,a} \tag{10}$$

The total mortality (Z) is the sum of natural and fishing mortality at year and age over all fleets

$$Z_{t,a} = M_{t,a} + \sum_{ifleet} F_{ifleet,t,a} \tag{11}$$

Population Abundance

The population abundance in the first year for ages 2 through the maximum age are derived from either the initial guesses ($N1ini_a$) and the parameters $log\_Nyear1dev_a$ as

$$N_{1,a} = N1ini_a e^{\log\_Nyear1dev_a} \tag{12}$$

or as deviations from a population in equilibrium according to the total mortality at age vector in the first year. A partial spawning stock biomass for ages 2 through the maximum age is computed and used in the stock recruitment relationship (Eq. 2) to create an expected recruitment in the first year. The recruitment deviation for the first year is applied to form the population abundance at age 1 in the first year (Eq. 4). The full spawning stock biomass is computed for year 1 using all ages (Eq. 1) now that the first year is completely filled.

The population abundance for years 2 through the end year are then filled by first computing the expected recruitment (Eq. 2) and then applying the recruitment deviation to create the abundance at age 1 (Eq. 4). Ages 2 through the maximum age are filled using the following set of equations

$$\begin{aligned} N_{t,a} &= N_{t-1,a-1} e^{-Z_{t-1,a-1}} \qquad 2 \leq a < A \\ N_{t,A} &= N_{t-1,A-1} e^{-Z_{t-1,A-1}} + N_{t-1,A} e^{-Z_{t-1,A}} \end{aligned} \tag{13}$$

Each year the spawning stock biomass is computed (Eq. 1) and the cycle continued until the end year is reached.

F Report

The original ASAP simply output the Fmult for each fleet and year as an indicator of fishing intensity, along with the full F matrix by fleet and combined over all fleets. This approach for comparing fishing intensity is sufficient if selectivity does not change over time, but can be problematic when selectivity changes. A feature of ASAP2 that is continued in ASAP3 is the use of Freport, which averages the total fishing mortality over an input range of ages (a_{repmin} to a_{repmax}). The averaging is done unweighted ($\omega_{t,a}$=1), weighted by population abundance at age ($\omega_{t,a}=N_{t,a}$), and weighted by population biomass at age ($\omega_{t,a}=N_{t,a}W_{t,a}$ where $W_{t,a}$ denotes the January 1 weight at year and age) as

$$Freport_t = \frac{\sum_{a=a_{rep\min}}^{a_{rep\max}} \omega_{t,a} F_{t,a}}{\sum_{a=a_{rep\min}}^{a_{rep\max}} \omega_{t,a}} \tag{14}$$

Predicted Catch

The predicted landings (*Lpred*) and discards (*Dpred*) in units of numbers of fish for each fleet, year, and age are derived from the Baranov catch equation

$$Lpred_{ifleet,t,a} = N_{ifleet,t,a} Fdir_{ifleet,t,a} (1 - e^{-Z_{t,a}}) / Z_{t,a} \tag{15}$$

$$Dpred_{ifleet,t,a} = N_{ifleet,t,a} Fbycatch_{ifleet,t,a} (1 - e^{-Z_{t,a}}) / Z_{t,a} \tag{16}$$

These predictions are used in two ways, one to form the predicted total weight of landings or discards for a fleet and year, and the other to form the proportions at age for a fleet and year. Both calculations are limited by the starting and ending ages for the fleet. The predicted total catch in weight calculations use the catch weight at year and age ($Wc_{t,a}$)

$$\hat{L}tot_{ifleet,t} = \sum_{a=fleetstart}^{fleetend} Lpred_{ifleet,t,a} Wc_{t,a} \tag{17}$$

$$\hat{D}tot_{ifleet,t} = \sum_{a=fleetstart}^{fleetend} Dpred_{ifleet,t,a} Wc_{t,a} \tag{18}$$

Note that since Fbycatch is derived using the proportion of fish that die after release, the total observed discards in weight (*Dtot*) should only include those fish that die after capture and release.

The predicted landings and discards proportions at age for each fleet and year are only computed for ages within the starting and ending range

$$\hat{L}p_{ifleet,t,a} = \frac{Lpred_{ifleet,t,a}}{\sum_{a=fleetstart}^{fleetend} Lpred_{ifleet,t,a}} \tag{19}$$

$$\hat{D}p_{ifleet,t,a} = \frac{Dpred_{ifleet,t,a}}{\sum_{a=fleetstart}^{fleetend} Dpred_{ifleet,t,a}} \tag{20}$$

Any predicted proportion less than 1e-15 is replaced by the value 1e-15 to avoid division by zero problems in the calculation of the likelihood function.

Catchability

Catchability for each index (*ind*) over time is computed similarly to the Fmult, with one parameter for the catchability in the first year (log_q1_{ind}) and a number of deviation parameters for each additional year of index observations ($log\_q\_dev_{ind,t}$). These parameters are combined and exponentiated to form the catchability value for the fleet and year as

$$q_{ind,t} = e^{\log\_q1_{ind} + \log\_q\_dev_{ind,t}} \quad (21)$$

where the parameter for the deviation in the first year ($log\_q\_dev_{ind,1}$) is defined as zero.

Predicted Indices

The observed indices have two characteristics that are matched when predicted values are computed, the time of year of the index and the units (numbers or biomass). The estimated population numbers at age are modified to the time of the index according to

$$N*_{ind,t,a} = N_{t,a} \frac{1 - e^{-Z_{t,a}}}{Z_{t,a}} \quad (22)$$

if the index month is set to -1, corresponding to an average abundance, or

$$N*_{ind,t,a} = N_{t,a}(1 - e^{-(ind\_month/12)Z_{t,a}}) \quad (23)$$

for index month between 0 and 12. Note that the index month refers to the end of the month, so *ind_month*=0 is January 1 and *ind_month*=12 is December 31. If the units for an index are biomass, then the N* values are multiplied by the user defined weights at age matrix. The selectivity associated with each index is either matched to a fleet or else input. If the selectivity for a fleet is input, it can be either fixed or estimated in the same way as the fleet selectivities (age based, logistic, or double logistic). The final predicted index (*Ipred*) is formed by summing the product of N* and selectivity values over the appropriate ages and multiplying by the catchability for the index

$$Ipred_{ind,t} = q_{ind,t} \sum_{a=indstart}^{indend} N*_{ind,t,a} \, Sel_{ind,t,a} \quad (24)$$

If the user selects to estimate the proportions at age for an index, then the proportions at age are computed in the same manner as the landings and discards at age (equations 19 and 20). Note that the units used for the aggregate index and proportions at age are set by the user separately, so all four combinations of numbers and biomass are possible.

Reference Points

The program computes a number of common reference points based on the estimated F and biological characteristics of the final year in the assessment. The reference points derive a directed and discard selectivity pattern from all the fleets that were assigned to be directed by summing the F at age and dividing by the maximum directed F. The non-

directed F is summed over all fleets that were not assigned as directed, and these F values are fixed during the reference point calculations. The F reference points are computed through a bisection algorithm that is repeated 20 times (producing an accuracy of approximately 1E-05). The reference points computed are $F_{0.1}$, F_{MAX}, $F_{30\%SPR}$, $F_{40\%SPR}$, and F_{MSY}. The associated maximum sustainable yield and spawning stock biomass at F_{MSY} are also provided. The reference point values are averaged in the same manner as the Freport to allow direct comparison. Note, however, that if selectivity or biological characteristics change over time, these comparisons will not be accurate because the reference points are computed assuming the final year values. The program now computes the annual unexploited SSB, unexploited R, unexploited SSB per R, and steepness to demonstrate the potential for change in the F reference points.

Projections

The projections use the same basic calculations as the main assessment program, except that there is no fitting done. The recruitments for each projection year can either be entered by the user or else be derived from the stock recruitment curve (without deviations from the curve). The directed and discard selectivity as well as the bycatch F at age are the same as used in the reference point calculations. There are five options to define what is used to define the fishery in each projections year:

1. match an input directed catch in weight
2. fish at an input F%SPR
3. fish at F_{MSY}
4. fish at the current (terminal year) F
5. fish at an input F

Each year the bycatch F can be modified from the terminal year to examine either increases or decreases in this(these) fishery(ies).

Objective Function Calculation (Fitting the Model)

The objective function in ASAP3 is the sum of a number of model fits and two penalties. There are two types of error distributions in the calculation of the objective function: lognormal and multinomial. Both are converted to negative log likelihoods for use in the minimization conducted by ADMB. Both error distributions contain constant terms that do not change for any value of the parameters. These constants can be either included or excluded from the objective function. Note that since the weights for different components of the objective function multiply the constants, different solutions may result when the constants are included or not.

The lognormal model fits all contain a lambda value that allows emphasis of that particular part of the objective function along with an input coefficient of variation (CV) that is used to measure how strong a particular deviation is. The CV is converted to a variance (σ^2) and associated standard deviation (σ) using the equation

$$\sigma^2 = \ln(CV^2 + 1) \qquad (25)$$

The lognormal distribution has a negative log likelihood, *-ln(L)*, defined by

$$-\ln(L) = 0.5\ln(2\pi) + \sum \ln(obs_i) + \ln(\sigma) + 0.5\sum \frac{(\ln(obs_i) - \ln(pred_i))^2}{\sigma^2} \quad (26)$$

The first two terms on the right side of equation (26) are the constants that are optionally kept or set to zero. The objective function is calculated as

$$obj\ fxn = \lambda * (-\ln(L)) \quad (27)$$

So that any component of the objective function can be turned off by setting λ for that component to zero. Standardized residuals for each component are calculated as

$$std\ resid_i = \frac{\ln(obs_i) - \ln(pred_i)}{\sigma} \quad (28)$$

In a perfectly fit model, the standardized residuals would have mean zero and standard deviation one.

The multinomial distribution fits employ an input effective sample size to multiply the negative log likelihood when calculating the objective function. This distribution is made up of k bins each containing p_i proportion of the total (sum of p_i=1). The input effective sample size (ESS) is used to create the number of fish in each bin (n_i) as n_i=ESS*p_i. The multinomial distribution then has a negative log likelihood defined by

$$-\ln(L) = -\ln(ESS!) + \sum_{i=1}^{k} \ln(n_i!) - ESS\sum_{i=1}^{k} p_i \ln(predp_i) \quad (29)$$

where p_i denotes an observed proportion and $predp_i$ denotes the associated predicted proportion. The first two terms on the right side of equation (29) are the constants that are optionally kept or set to zero. The objective function is simply the negative log likelihood for the multinomial distribution because the effective sample size is an integral part of the calculation of the likelihood.

The lognormal error distribution is assumed for

- Total catch in weight
- Total discards in weight
- Indices
- Stock recruitment relationship
- Selectivity parameters (relative to initial guesses)
- The two stock recruitment parameters (relative to their initial guesses)
- Fmult in year 1 by fleet (relative to initial guesses)
- Fmult deviations
- Catchability in year 1 by fleet (relative to initial guesses)
- Catchability deviations
- Numbers at age in year 1 (relative to either initial guesses or a population in equilibrium)

Multinomial distribution is assumed for

- Catch at age
- Discards at age
- Index proportions at age

The two penalties are formed from estimated total fishing mortality rates. The first is a penalty associated with any total F greater than an input maximum value, calculated as $1000*(F-Fmax)^2$ for F>Fmax. The second penalty is for F different than M in the early phases, calculated as $100*10^{-phase}$ $(ln(avg(F)-ln(M))^2$. The second penalty is always set to zero in the final estimation phase, regardless of the number of phases.

Appendix 1: Source Code for ASAP3

(Note the code sometimes wraps around to the next line in the presentation here.)

```
// ASAP3 (Age Structured Assessment Program Version 3: August 2012)
//   by Christopher Legault with major contributions from Liz Brooks
//   modified from ASAP2 by Christopher Legault
//   modified from original ASAP by Christopher Legault and Victor Restrepo 1998

// Major changes from ASAP2
// user defines SR curve using steepness and either R0 or S0
// allow user to mix and match biomass and numbers for aggreagate indices and indices proportions at age
// user enters a number of weight at age matrices then defines which are used for catch, discards, SSB, Jan-1 B,
and indices
// compute annual SR curve estimates of R0, S0, steepness, and spawners per recruit to show how changes in M,
fecundity, WAA impact these estimates over time
// expected population at age in year 1 can be either an exponential decline or user initial guesses for
optional deviation calculations
// compute Francis (2011) stage 2 multiplier for multinomial to adjust input Neff

// update April 2012
// fix bug with which inconsistent year for M and WAA used in calculation of unexploited SSB per recruit
// (was first year when all other calculations were last year, now everything last year)
// also added trap for division by zero in Freport calculation to avoid crashes when pop size gets small
// incorporated Liz Brook's make-Rfile.cxx for ADMB2R to optionally create rdat file automatically
// created new output file asap2RMSE.dat for use with R script

// update April 2008
// fixed bug in get_log_factorial function - variable could be i used in two places (thanks to Tim Miller for
finding this one)
//
// Major changes from original ASAP
//
// Enter all available indices and then select which ones to use for tuning
// Change in selectivity estimation to reduce parameter correlations
//   Added option to use logistic or double logistic selectivity patterns
//   Selectivity blocks now independent with own initial starting guesses
// Added CVs and lambdas for many parameters
// Multiple matrices for weights at age at different times of the year
// M matrix instead of vector
// Freport feature to allow easier comparison among years with different selectivity patterns
// Echo input read to file for improved debugging
// MCMC capability added
//   One file for Freport, SSB, and MSY related variables
//   One file for use in AgePro software (.bsn file)
// Full likelihood calculations, including (optionally) constants
// Output of standardized residuals
// Modified year 1 recruitment deviation calculations to reduce probability of extremely large residual

TOP_OF_MAIN_SECTION
// set buffer sizes
  arrmblsize=5000000;
  gradient_structure::set_GRADSTACK_BUFFER_SIZE(10000000);
  gradient_structure::set_MAX_NVAR_OFFSET(50000);
  gradient_structure::set_NUM_DEPENDENT_VARIABLES(10000);
  time(&start); //this is to see how long it takes to run
  cout << endl << "Start time : " << ctime(&start) << endl;

GLOBALS_SECTION
  #include <admodel.h>
  #include <time.h>
  #include <C:\ADMB\admb2r-1.15\admb2r\admb2r.cpp>
  time_t start,finish;
  long hour,minute,second;
  double elapsed_time;
  ofstream ageproMCMC("asap3.bsn");
  ofstream basicMCMC("asap3MCMC.dat");
  ofstream inputlog("asap3input.log");
  //--- preprocessor macro from Larry Jacobson NMFS-Woods Hole
```

```
  #define ICHECK(object) inputlog << "#" #object "\n " << object << endl;

DATA_SECTION
  int debug
  int iyear
  int iage
  int ia
  int ifleet
  int ind
  int i
  int j
  int k
  int iloop
  int io
  number pi
 !!  pi=3.14159265358979;
  number CVfill
 !! CVfill=100.0;
// basic dimensions
  init_int nyears
 !! ICHECK(nyears);
  init_int year1
 !! ICHECK(year1);
  init_int nages
 !! ICHECK(nages);
  init_int nfleets
 !! ICHECK(nfleets);
  init_int nselblocks;
 !! ICHECK(nselblocks);
  init_int navailindices
 !! ICHECK(navailindices);

// biology
  init_matrix M(1,nyears,1,nages)
 !! ICHECK(M);
  init_number isfecund
 !! ICHECK(isfecund);
  init_number fracyearSSB
 !! ICHECK(fracyearSSB);
  init_matrix mature(1,nyears,1,nages)
 !! ICHECK(mature);
  init_int nWAAmatrices
 !! ICHECK(nWAAmatrices);
  int nrowsWAAini
 !! nrowsWAAini=nyears*nWAAmatrices;
  init_matrix WAA_ini(1,nrowsWAAini,1,nages)
 !! ICHECK(WAA_ini);
  int nWAApointbio
 !! nWAApointbio=nfleets*2+2+2;
  init_ivector WAApointbio(1,nWAApointbio) // pointers to WAA matrix for fleet catch and discards, catch all
fleets, discard all fleets, SSB, and Jan1B
 !! ICHECK(WAApointbio);
  matrix fecundity(1,nyears,1,nages)
  3darray WAAcatchfleet(1,nfleets,1,nyears,1,nages)
  3darray WAAdiscardfleet(1,nfleets,1,nyears,1,nages)
  matrix WAAcatchall(1,nyears,1,nages)
  matrix WAAdiscardall(1,nyears,1,nages)
  matrix WAAssb(1,nyears,1,nages)
  matrix WAAjan1b(1,nyears,1,nages)
 LOCAL_CALCS
  if ((max(WAApointbio) > nWAAmatrices) || (min(WAApointbio) < 1))
  {
     cout << "Problem with WAApointbio" << endl;
     ad_exit(1);
  }
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     int ipointcatchfleet=(WAApointbio((ifleet*2)-1)-1)*nyears;
     int ipointdiscardfleet=(WAApointbio(ifleet*2)-1)*nyears;
     for (iyear=1;iyear<=nyears;iyear++)
     {
```

```
        WAAcatchfleet(ifleet,iyear)=WAA_ini((ipointcatchfleet+iyear));
        WAAdiscardfleet(ifleet,iyear)=WAA_ini((ipointdiscardfleet+iyear));
     }
  }
  int ipointcatchall=(WAApointbio((nfleets*2)+1)-1)*nyears;
  int ipointdiscardall=(WAApointbio((nfleets*2)+2)-1)*nyears;
  int ipointssb=(WAApointbio((nfleets*2)+3)-1)*nyears;
  int ipointjan1b=(WAApointbio((nfleets*2)+4)-1)*nyears;
  for (iyear=1;iyear<=nyears;iyear++)
  {
     WAAcatchall(iyear)=WAA_ini((ipointcatchall+iyear));
     WAAdiscardall(iyear)=WAA_ini((ipointdiscardall+iyear));
     WAAssb(iyear)=WAA_ini((ipointssb+iyear));
     WAAjan1b(iyear)=WAA_ini((ipointjan1b+iyear));
  }
  if (isfecund==1)
     fecundity=mature;
  else
     fecundity=elem_prod(WAAssb,mature);
 END_CALCS

// fleet names here with $ in front of label

// Selectivity ************************************
// need to enter values for all options even though only one will be used for each block
  init_matrix sel_blocks(1,nfleets,1,nyears) // defines blocks for each fleet in successive order
 !! ICHECK(sel_blocks);
  int nsel_ini
 !! nsel_ini=nselblocks*(nages+6);
  init_ivector sel_option(1,nselblocks) // 1=by age, 2=logisitic, 3=double logistic
 !! ICHECK(sel_option);
  init_matrix sel_ini(1,nsel_ini,1,4) // 1st value is initial guess, 2nd is phase, 3rd is lambda, 4th is CV
 !! ICHECK(sel_ini);
  int nselparm
 LOCAL_CALCS
// first count number of selectivity parameters and replace CV=0 with CVfill
  nselparm=0;
  for (i=1;i<=nselblocks;i++)
  {
     if (sel_option(i)==1) nselparm+=nages;
     if (sel_option(i)==2) nselparm+=2;
     if (sel_option(i)==3) nselparm+=4;
  }
  for (i=1;i<=nsel_ini;i++)
  {
     if (sel_ini(i,4) <= 0.0)
         sel_ini(i,4) = CVfill;
  }
 END_CALCS
  vector sel_initial(1,nselparm)
  vector sel_lo(1,nselparm)
  vector sel_hi(1,nselparm)
  ivector sel_phase(1,nselparm)
  vector sel_lambda(1,nselparm)
  vector sel_CV(1,nselparm)
  vector sel_sigma2(1,nselparm)
  vector sel_sigma(1,nselparm)
  vector sel_like_const(1,nselparm)
 LOCAL_CALCS
// now assign bounds and phases for each selectivity parameter
  k=0;
  for (i=1;i<=nselblocks;i++){
     if (sel_option(i)==1) {
        for (iage=1;iage<=nages;iage++) {
           k+=1;
           j=(i-1)*(nages+6)+iage;
           sel_initial(k)=sel_ini(j,1);
           sel_lo(k)=0.0;
           sel_hi(k)=1.0;
           sel_phase(k)=sel_ini(j,2);
           sel_lambda(k)=sel_ini(j,3);
```

```
            sel_CV(k)=sel_ini(j,4);
            sel_sigma2(k)=log(sel_CV(k)*sel_CV(k)+1.0);
            sel_sigma(k)=sqrt(sel_sigma2(k));
         }
      }
      if (sel_option(i)==2) {
         for (ia=1;ia<=2;ia++) {
            k+=1;
            j=(i-1)*(nages+6)+nages+ia;
            sel_initial(k)=sel_ini(j,1);
            sel_lo(k)=0.0;
            sel_hi(k)=nages;
            sel_phase(k)=sel_ini(j,2);
            sel_lambda(k)=sel_ini(j,3);
            sel_CV(k)=sel_ini(j,4);
            sel_sigma2(k)=log(sel_CV(k)*sel_CV(k)+1.0);
            sel_sigma(k)=sqrt(sel_sigma2(k));
         }
      }
      if (sel_option(i)==3) {
         for (ia=1;ia<=4;ia++) {
            k+=1;
            j=(i-1)*(nages+6)+nages+2+ia;
            sel_initial(k)=sel_ini(j,1);
            sel_lo(k)=0.0;
            sel_hi(k)=nages;
            sel_phase(k)=sel_ini(j,2);
            sel_lambda(k)=sel_ini(j,3);
            sel_CV(k)=sel_ini(j,4);
            sel_sigma2(k)=log(sel_CV(k)*sel_CV(k)+1.0);
            sel_sigma(k)=sqrt(sel_sigma2(k));
         }
      }
   }
  END_CALCS
   init_ivector sel_start_age(1,nfleets)
  !! ICHECK(sel_start_age);
   init_ivector sel_end_age(1,nfleets)
  !! ICHECK(sel_end_age);

   init_int Freport_agemin
  !! ICHECK(Freport_agemin);
   init_int Freport_agemax
  !! ICHECK(Freport_agemax);
   init_int Freport_wtopt
  !! ICHECK(Freport_wtopt);

   init_int use_likelihood_constants
  !! ICHECK(use_likelihood_constants);
   init_vector release_mort(1,nfleets)
  !! ICHECK(release_mort);

 // Catch ***************************************
 // Includes both landed and discarded components
   init_matrix CAA_ini(1,nyears*nfleets,1,nages+1)
  !! ICHECK(CAA_ini);
   init_matrix Discard_ini(1,nyears*nfleets,1,nages+1)
  !! ICHECK(Discard_ini);
   init_matrix proportion_release_ini(1,nyears*nfleets,1,nages)
  !! ICHECK(proportion_release_ini);
   3darray CAA_obs(1,nfleets,1,nyears,1,nages)
   3darray Discard_obs(1,nfleets,1,nyears,1,nages)
   3darray proportion_release(1,nfleets,1,nyears,1,nages)
   3darray CAA_prop_obs(1,nfleets,1,nyears,sel_start_age,sel_end_age)
   3darray Discard_prop_obs(1,nfleets,1,nyears,sel_start_age,sel_end_age)
   number catch_prop_like_const
   number discard_prop_like_const
   matrix Catch_tot_fleet_obs(1,nfleets,1,nyears)
   matrix Discard_tot_fleet_obs(1,nfleets,1,nyears)
   matrix CAA_prop_obs_sum(1,nfleets,1,nyears)
   matrix Discard_prop_obs_sum(1,nfleets,1,nyears)
```

```
  vector catch_tot_like_const(1,nfleets)
  vector discard_tot_like_const(1,nfleets)
 LOCAL_CALCS
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     catch_tot_like_const(ifleet)=0.0;
     discard_tot_like_const(ifleet)=0.0;
     for (iyear=1;iyear<=nyears;iyear++)
     {
        CAA_obs(ifleet,iyear)(1,nages)=CAA_ini((ifleet-1)*nyears+iyear)(1,nages);
        Discard_obs(ifleet,iyear)(1,nages)=Discard_ini((ifleet-1)*nyears+iyear)(1,nages);
        proportion_release(ifleet,iyear)=proportion_release_ini((ifleet-1)*nyears+iyear)(1,nages);
        Catch_tot_fleet_obs(ifleet,iyear)=CAA_ini((ifleet-1)*nyears+iyear,nages+1);
        Discard_tot_fleet_obs(ifleet,iyear)=Discard_ini((ifleet-1)*nyears+iyear,nages+1);
        if (Catch_tot_fleet_obs(ifleet,iyear)>1.0e-15)
           catch_tot_like_const(ifleet)+=0.5*log(2.0*pi)+log(Catch_tot_fleet_obs(ifleet,iyear));
        if (Discard_tot_fleet_obs(ifleet,iyear)>1.0e-15)
           discard_tot_like_const(ifleet)=0.5*log(2.0*pi)+log(Discard_tot_fleet_obs(ifleet,iyear));
     }
  }
  if (use_likelihood_constants != 1)
  {
     catch_tot_like_const=0.0;
     discard_tot_like_const=0.0;
  }
  CAA_prop_obs=0.0;
  Discard_prop_obs=0.0;
  CAA_prop_obs_sum=0.0;
  Discard_prop_obs_sum=0.0;
  for (iyear=1;iyear<=nyears;iyear++)
  {
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
       if (Catch_tot_fleet_obs(ifleet,iyear)>0.0)
       {
          for (iage=sel_start_age(ifleet);iage<=sel_end_age(ifleet);iage++)
             CAA_prop_obs_sum(ifleet,iyear)+=CAA_obs(ifleet,iyear,iage);
          if (CAA_prop_obs_sum(ifleet,iyear)==0.0)
          {
             CAA_prop_obs(ifleet,iyear)=0.0;
          }
          else
          {

CAA_prop_obs(ifleet,iyear)=CAA_obs(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet))/CAA_prop_obs_sum(ifl
eet,iyear);
          }
       }
       if (Discard_tot_fleet_obs(ifleet,iyear)>0.0)
       {
          for (iage=sel_start_age(ifleet);iage<=sel_end_age(ifleet);iage++)
             Discard_prop_obs_sum(ifleet,iyear)+=Discard_obs(ifleet,iyear,iage);
          if (Discard_prop_obs_sum(ifleet,iyear)==0.0)
          {
             Discard_prop_obs(ifleet,iyear)=0.0;
          }
          else
          {

Discard_prop_obs(ifleet,iyear)=Discard_obs(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet))/Discard_prop
_obs_sum(ifleet,iyear);
          }
       }
    }
  }
 END_CALCS

// Indices ******************************
// Enter in all available indices and then pick the ones that are to be used in objective function
// navailindices is the number of indices entered
// nindices is the number of indices used (calculated by program)
```

```
  int indavail
// index names here with $ in front of label
  init_vector index_units_aggregate_ini(1,navailindices) // 1=biomass, 2=numbers
 !! ICHECK(index_units_aggregate_ini);
  init_vector index_units_proportions_ini(1,navailindices) // 1=biomass, 2=numbers
 !! ICHECK(index_units_proportions_ini);
  init_ivector index_WAApoint_ini(1,navailindices) // pointer for which WAA matrix to use for biomass
calculations for each index
 !! ICHECK(index_WAApoint_ini);
  init_vector index_month_ini(1,navailindices) // -1=average pop
 !! ICHECK(index_month_ini);
  init_ivector index_sel_choice_ini(1,navailindices) // -1=fixed
 !! ICHECK(index_sel_choice_ini);
  init_ivector index_sel_option_ini(1,navailindices) // 1=by age, 2=logisitic, 3=double logistic
 !! ICHECK(index_sel_option_ini);
  init_ivector index_start_age_ini(1,navailindices)
 !! ICHECK(index_start_age_ini);
  init_ivector index_end_age_ini(1,navailindices)
 !! ICHECK(index_end_age_ini);
  init_ivector index_estimate_proportions_ini(1,navailindices) // 1=yes
 !! ICHECK(index_estimate_proportions_ini);
  init_ivector use_index(1,navailindices) // 1=yes
 !! ICHECK(use_index);
  int nindexsel_ini
 !! nindexsel_ini=navailindices*(nages+6);
  init_matrix index_sel_ini(1,nindexsel_ini,1,4) // 1st value is initial guess, 2nd is phase, 3rd is lambda, 4th
is CV
 !! ICHECK(index_sel_ini);
  init_matrix index_ini(1,nyears*navailindices,1,3+nages+1) // year, index value, CV, proportions at age, input
effective sample size
 !! ICHECK(index_ini);
  int nindices
 !! nindices=sum(use_index);
  vector index_units_aggregate(1,nindices)
  vector index_units_proportions(1,nindices)
  ivector index_WAApoint(1,nindices)
  vector index_month(1,nindices)
  vector index_sel_option(1,nindices)
  vector index_start_age(1,nindices)
  vector index_end_age(1,nindices)
  vector index_sel_choice(1,nindices)
  ivector index_nobs(1,nindices)
  ivector index_estimate_proportions(1,nindices)
  int nindexselparms
 LOCAL_CALCS
  if ((max(index_WAApoint_ini) > nWAAmatrices) || (min(index_WAApoint_ini) < 1))
  {
     cout << "Problem with index_WAApoint_ini" << endl;
     ad_exit(1);
  }
  for (i=1;i<=nindexsel_ini;i++)
  {
     if (index_sel_ini(i,4) <= 0.0)
         index_sel_ini(i,4) = CVfill;
  }
  for (i=1;i<=nyears*navailindices;i++)
  {
     if (index_ini(i,3) <= 0.0)
         index_ini(i,3) = CVfill;
  }
  ind=0;
  nindexselparms=0;
  for (indavail=1;indavail<=navailindices;indavail++)
  {
     if (use_index(indavail)==1)
     {
       ind+=1;
       index_units_aggregate(ind)=index_units_aggregate_ini(indavail);
       index_units_proportions(ind)=index_units_proportions_ini(indavail);
       index_WAApoint(ind)=index_WAApoint_ini(indavail);
       index_month(ind)=index_month_ini(indavail);
```

```
      index_sel_option(ind)=index_sel_option_ini(indavail);
      if (index_sel_option(ind)==1) nindexselparms+=nages;
      if (index_sel_option(ind)==2) nindexselparms+=2;
      if (index_sel_option(ind)==3) nindexselparms+=4;
      index_start_age(ind)=index_start_age_ini(indavail);
      index_end_age(ind)=index_end_age_ini(indavail);
      index_sel_choice(ind)=index_sel_choice_ini(indavail);
      index_estimate_proportions(ind)=index_estimate_proportions_ini(indavail);
      j=0;
      for (iyear=1;iyear<=nyears;iyear++)
      {
        if (index_ini((indavail-1)*nyears+iyear,2)>0.0)  // zero or negative value for index means not included
            j+=1;
      }
      index_nobs(ind)=j;
    }
  }
 END_CALCS
  matrix index_time(1,nindices,1,index_nobs)
  matrix index_year(1,nindices,1,index_nobs)
  matrix index_obs(1,nindices,1,index_nobs)
  matrix index_cv(1,nindices,1,index_nobs)
  matrix index_sigma2(1,nindices,1,index_nobs)
  matrix index_sigma(1,nindices,1,index_nobs)
  matrix input_eff_samp_size_index(1,nindices,1,index_nobs)
  vector indexsel_initial(1,nindexselparms)
  vector indexsel_lo(1,nindexselparms)
  vector indexsel_hi(1,nindexselparms)
  ivector indexsel_phase(1,nindexselparms)
  vector indexsel_lambda(1,nindexselparms)
  vector indexsel_CV(1,nindexselparms)
  vector indexsel_sigma2(1,nindexselparms)
  vector indexsel_sigma(1,nindexselparms)
  vector indexsel_like_const(1,nindexselparms)
  number index_prop_like_const
  3darray index_sel_input(1,nindices,1,nyears,1,nages)
  3darray index_prop_obs(1,nindices,1,index_nobs,1,nages)
  3darray index_WAA(1,nindices,1,nyears,1,nages)
  vector index_like_const(1,nindices)
  number tempsum
 LOCAL_CALCS
  index_prop_obs=0.0;
  ind=0;
  k=0;
  for (indavail=1;indavail<=navailindices;indavail++)
  {
     if (use_index(indavail)==1)
     {
       ind+=1;
// get the index selectivity information
      if (index_sel_option(ind)==1)
      {
         for (iage=1;iage<=nages;iage++)
         {
            k+=1;
            j=(indavail-1)*(nages+6)+iage;
            indexsel_initial(k)=index_sel_ini(j,1);
            indexsel_lo(k)=0.0;
            indexsel_hi(k)=1.0;
            indexsel_phase(k)=index_sel_ini(j,2);
            indexsel_lambda(k)=index_sel_ini(j,3);
            indexsel_CV(k)=index_sel_ini(j,4);
            indexsel_sigma2(k)=log(indexsel_CV(k)*indexsel_CV(k)+1.0);
            indexsel_sigma(k)=sqrt(indexsel_sigma2(k));
         }
      }
      else if (index_sel_option(ind)==2)
      {
         for (ia=1;ia<=2;ia++)
         {
            k+=1;
```

```
            j=(indavail-1)*(nages+6)+nages+ia;
            indexsel_initial(k)=index_sel_ini(j,1);
            indexsel_lo(k)=0.0;
            indexsel_hi(k)=nages;
            indexsel_phase(k)=index_sel_ini(j,2);
            indexsel_lambda(k)=index_sel_ini(j,3);
            indexsel_CV(k)=index_sel_ini(j,4);
            indexsel_sigma2(k)=log(indexsel_CV(k)*indexsel_CV(k)+1.0);
            indexsel_sigma(k)=sqrt(indexsel_sigma2(k));
         }
      }
      else if (index_sel_option(ind)==3)
      {
         for (ia=1;ia<=4;ia++)
         {
            k+=1;
            j=(indavail-1)*(nages+6)+nages+2+ia;
            indexsel_initial(k)=index_sel_ini(j,1);
            indexsel_lo(k)=0.0;
            indexsel_hi(k)=nages;
            indexsel_phase(k)=index_sel_ini(j,2);
            indexsel_lambda(k)=index_sel_ini(j,3);
            indexsel_CV(k)=index_sel_ini(j,4);
            indexsel_sigma2(k)=log(indexsel_CV(k)*indexsel_CV(k)+1.0);
            indexsel_sigma(k)=sqrt(indexsel_sigma2(k));
         }
      }

// get the index and year specific information
       j=0;
       for (iyear=1;iyear<=nyears;iyear++)
       {
         i=(indavail-1)*nyears+iyear;
         index_sel_input(ind,iyear)=--(--(--index_ini(i)(4,3+nages)));
         if (index_ini(i,2)>0.0)
         {
             j+=1;
             index_time(ind,j)=index_ini(i,1)-year1+1;
             index_year(ind,j)=index_ini(i,1);
             index_obs(ind,j)=index_ini(i,2);
             index_cv(ind,j)=index_ini(i,3);
             index_sigma2(ind,j)=log(index_cv(ind,j)*index_cv(ind,j)+1.0);
             index_sigma(ind,j)=sqrt(index_sigma2(ind,j));
             input_eff_samp_size_index(ind,j)=index_ini(i,nages+4);
             tempsum=sum(index_sel_input(ind,iyear)(index_start_age(ind),index_end_age(ind)));
             if (tempsum > 0.0)
             {
                 for (iage=index_start_age(ind);iage<=index_end_age(ind);iage++)
                 {
                     index_prop_obs(ind,j,iage)=index_sel_input(ind,iyear,iage)/tempsum;
                 }
             }
         }
       }
     }
   }
   index_like_const=0.0;
   if (use_likelihood_constants==1)
   {
      for (ind=1;ind<=nindices;ind++)
      {
         index_like_const(ind)=0.5*double(index_nobs(ind))*log(2.0*pi)+sum(log(index_obs(ind)));
      }
   }

// set up the index_WAA matrices (indices in numbers only will have WAA set to 0)
   index_WAA=0.0;
   for (ind=1;ind<=nindices;ind++)
   {
      if (index_units_aggregate(ind)==1 || index_units_proportions(ind)==1)
      {
```

```
        int ipointindex=(index_WAApoint(ind)-1)*nyears;
        for (iyear=1;iyear<=nyears;iyear++)
        {
           index_WAA(ind,iyear)=WAA_ini((ipointindex+iyear));
        }
     }
  }
 END_CALCS

// Phase Controls (other than selectivity)
  init_int phase_Fmult_year1
 !! ICHECK(phase_Fmult_year1);
  init_int phase_Fmult_devs
 !! ICHECK(phase_Fmult_devs);
  init_int phase_recruit_devs
 !! ICHECK(phase_recruit_devs);
  init_int phase_N_year1_devs
 !! ICHECK(phase_N_year1_devs);
  init_int phase_q_year1
 !! ICHECK(phase_q_year1);
  init_int phase_q_devs
 !! ICHECK(phase_q_devs);
  init_int phase_SR_scaler
 !! ICHECK(phase_SR_scaler);
  init_int phase_steepness
 !! ICHECK(phase_steepness);
  init_vector recruit_CV(1,nyears)
 !! ICHECK(recruit_CV);
  vector recruit_sigma2(1,nyears)
  vector recruit_sigma(1,nyears)
  number SR_like_const
 LOCAL_CALCS
  for (iyear=1;iyear<=nyears;iyear++)
  {
    if (recruit_CV(iyear) <= 0.0)
        recruit_CV(iyear) = CVfill;
    recruit_sigma2(iyear)=log(recruit_CV(iyear)*recruit_CV(iyear)+1.0);
    recruit_sigma(iyear)=sqrt(recruit_sigma2(iyear));
  }
  SR_like_const=0.0;
  if (use_likelihood_constants == 1)
     SR_like_const=0.5*double(nyears)*log(2.0*pi);
 END_CALCS
  init_vector lambda_ind_ini(1,navailindices)
 !! ICHECK(lambda_ind_ini);
  init_vector lambda_catch_tot(1,nfleets)
 !! ICHECK(lambda_catch_tot);
  init_vector lambda_Discard_tot(1,nfleets)
 !! ICHECK(lambda_Discard_tot);
  init_matrix catch_tot_CV(1,nyears,1,nfleets)
 !! ICHECK(catch_tot_CV);
  init_matrix discard_tot_CV(1,nyears,1,nfleets)
 !! ICHECK(discard_tot_CV);
  matrix catch_tot_sigma2(1,nfleets,1,nyears)
  matrix catch_tot_sigma(1,nfleets,1,nyears)
  matrix discard_tot_sigma2(1,nfleets,1,nyears)
  matrix discard_tot_sigma(1,nfleets,1,nyears)
  init_matrix input_eff_samp_size_catch_ini(1,nyears,1,nfleets)
 !! ICHECK(input_eff_samp_size_catch_ini);
  init_matrix input_eff_samp_size_discard_ini(1,nyears,1,nfleets)
 !! ICHECK(input_eff_samp_size_discard_ini);
  matrix input_eff_samp_size_catch(1,nfleets,1,nyears)
  matrix input_eff_samp_size_discard(1,nfleets,1,nyears)
  number nfact_in
  number nfact_out
 LOCAL_CALCS
  for(iyear=1;iyear<=nyears;iyear++)
  {
   for(ifleet=1;ifleet<=nfleets;ifleet++)
   {
     if (catch_tot_CV(iyear,ifleet) <= 0.0)
```

```
        catch_tot_CV(iyear,ifleet) = CVfill;
    if (discard_tot_CV(iyear,ifleet) <= 0.0)
        discard_tot_CV(iyear,ifleet) = CVfill;
    catch_tot_sigma2(ifleet,iyear)=log(catch_tot_CV(iyear,ifleet)*catch_tot_CV(iyear,ifleet)+1.0);
    catch_tot_sigma(ifleet,iyear)=sqrt(catch_tot_sigma2(ifleet,iyear));
    discard_tot_sigma2(ifleet,iyear)=log(discard_tot_CV(iyear,ifleet)*discard_tot_CV(iyear,ifleet)+1.0);
    discard_tot_sigma(ifleet,iyear)=sqrt(discard_tot_sigma2(ifleet,iyear));
    input_eff_samp_size_catch(ifleet,iyear)=input_eff_samp_size_catch_ini(iyear,ifleet);
    input_eff_samp_size_discard(ifleet,iyear)=input_eff_samp_size_discard_ini(iyear,ifleet);
  }
 }
END_CALCS
 init_vector lambda_Fmult_year1(1,nfleets)
!! ICHECK(lambda_Fmult_year1);
 init_vector Fmult_year1_CV(1,nfleets)
!! ICHECK(Fmult_year1_CV);
 init_vector lambda_Fmult_devs(1,nfleets)
!! ICHECK(lambda_Fmult_devs);
 init_vector Fmult_devs_CV(1,nfleets)
!! ICHECK(Fmult_devs_CV);
 init_number lambda_N_year1_devs
!! ICHECK(lambda_N_year1_devs);
 init_number N_year1_CV
!! ICHECK(N_year1_CV);
 init_number lambda_recruit_devs
!! ICHECK(lambda_recruit_devs);
 init_vector lambda_q_year1_ini(1,navailindices)
!! ICHECK(lambda_q_year1_ini);
 init_vector q_year1_CV_ini(1,navailindices)
!! ICHECK(q_year1_CV_ini);
 init_vector lambda_q_devs_ini(1,navailindices)
!! ICHECK(lambda_q_devs_ini);
 init_vector q_devs_CV_ini(1,navailindices)
!! ICHECK(q_devs_CV_ini);
 init_number lambda_steepness
!! ICHECK(lambda_steepness);
 init_number steepness_CV
!! ICHECK(steepness_CV);
 init_number lambda_SR_scaler
!! ICHECK(lambda_SR_scaler);
 init_number SR_scaler_CV
!! ICHECK(SR_scaler_CV);
LOCAL_CALCS
 for (i=1;i<=nfleets;i++)
 {
    if (Fmult_year1_CV(i) <= 0.0)
        Fmult_year1_CV(i) = CVfill;
    if (Fmult_devs_CV(i) <= 0.0)
        Fmult_devs_CV(i) = CVfill;
 }
 if (N_year1_CV <= 0.0)
     N_year1_CV = CVfill;
 for (i=1;i<=navailindices;i++)
 {
    if (q_year1_CV_ini(i) <= 0.0)
        q_year1_CV_ini(i) = CVfill;
    if (q_devs_CV_ini(i) <= 0.0)
        q_devs_CV_ini(i) = CVfill;
 }
 if (steepness_CV <= 0.0)
     steepness_CV = CVfill;
 if (SR_scaler_CV <= 0.0)
     SR_scaler_CV = CVfill;
END_CALCS
 vector Fmult_year1_sigma2(1,nfleets)
 vector Fmult_year1_sigma(1,nfleets)
 vector Fmult_year1_like_const(1,nfleets)
 vector Fmult_devs_sigma2(1,nfleets)
 vector Fmult_devs_sigma(1,nfleets)
 vector Fmult_devs_like_const(1,nfleets)
 number N_year1_sigma2
```

```
  number N_year1_sigma
  number N_year1_like_const
  vector lambda_ind(1,nindices)
  vector lambda_q_year1(1,nindices)
  vector q_year1_CV(1,nindices)
  vector q_year1_sigma2(1,nindices)
  vector q_year1_sigma(1,nindices)
  vector q_year1_like_const(1,nindices)
  vector lambda_q_devs(1,nindices)
  vector q_devs_CV(1,nindices)
  vector q_devs_sigma2(1,nindices)
  vector q_devs_sigma(1,nindices)
  vector q_devs_like_const(1,nindices)
  number steepness_sigma2
  number steepness_sigma
  number steepness_like_const
  number SR_scaler_sigma2
  number SR_scaler_sigma
  number SR_scaler_like_const

// starting guesses
  init_int NAA_year1_flag // 1 for devs from exponential decline, 2 for devs from initial guesses
 !! ICHECK(NAA_year1_flag);
  init_vector NAA_year1_ini(1,nages)
 !! ICHECK(NAA_year1_ini);
  init_vector Fmult_year1_ini(1,nfleets)
 !! ICHECK(Fmult_year1_ini);
  init_vector q_year1_iniavail(1,navailindices)
 !! ICHECK(q_year1_iniavail);
  vector q_year1_ini(1,nindices)
  init_number is_SR_scaler_R // 1 for R0, 0 for SSB0
 !! ICHECK(is_SR_scaler_R);
  init_number SR_scaler_ini
 !! ICHECK(SR_scaler_ini);
  init_number SR_steepness_ini
 !! ICHECK(SR_steepness_ini);
  init_number Fmult_max_value
 !! ICHECK(Fmult_max_value);

  init_number ignore_guesses
 !! ICHECK(ignore_guesses);
  number delta

// Projection Info*********************
  init_int do_projections
 !! ICHECK(do_projections);
  init_ivector directed_fleet(1,nfleets)
 !! ICHECK(directed_fleet);
  init_number nfinalyear
 !! ICHECK(nfinalyear);
  int nprojyears
 !! nprojyears=nfinalyear-year1-nyears+1;
  init_matrix project_ini(1,nprojyears,1,5)
 !! ICHECK(project_ini);
  vector proj_recruit(1,nprojyears)
  ivector proj_what(1,nprojyears)
  vector proj_target(1,nprojyears)
  vector proj_F_nondir_mult(1,nprojyears)
 LOCAL_CALCS
  for (iyear=1;iyear<=nprojyears;iyear++)
  {
    proj_recruit(iyear)=project_ini(iyear,2);
    proj_what(iyear)=project_ini(iyear,3);
    proj_target(iyear)=project_ini(iyear,4);
    proj_F_nondir_mult(iyear)=project_ini(iyear,5);
  }
 END_CALCS

// MCMC Info*******************************
  init_int doMCMC
 !! ICHECK(doMCMC);
```

```
 LOCAL_CALCS
  if (doMCMC == 1)
  {
     basicMCMC << " ";
     for (iyear=1;iyear<=nyears;iyear++)
     {
        basicMCMC << "F" << iyear+year1-1 << " ";
     }
     for (iyear=1;iyear<=nyears;iyear++)
     {
        basicMCMC << "SSB" << iyear+year1-1 << " ";
     }
     // Liz added Fmult_in lastyear and totBjan1
     for (iyear=1;iyear<=nyears;iyear++)
     {
        basicMCMC << "Fmult_" << iyear+year1-1 << " ";
          }
     for (iyear=1;iyear<=nyears;iyear++)
     {
        basicMCMC << "totBjan1_" << iyear+year1-1 << " ";
     }

     // end stuff Liz added
     basicMCMC << "MSY  SSBmsy  Fmsy  SSBmsy_ratio  Fmsy_ratio  ";
     basicMCMC << endl; // end of header line
  }
 END_CALCS
  init_int MCMCnyear_opt // 0=output nyear NAA, 1=output nyear+1 NAA
 !! ICHECK(MCMCnyear_opt)
  init_int MCMCnboot  // final number of values for agepro bootstrap file
 !! ICHECK(MCMCnboot);
  init_int MCMCnthin  // thinning rate (1=use every value, 2=use every other value, 3=use every third value,
etc)
 !! ICHECK(MCMCnthin);
  init_int MCMCseed   // large positive integer to seed random number generator
 !! ICHECK(MCMCseed);
// To run MCMC do the following two steps:
// 1st type "asap2 -mcmc N1 -mcsave MCMCnthin -mcseed MCMCseed"
//   where N1 = MCMCnboot * MCMCnthin
// 2nd type "asap2 -mceval"
  init_int fillR_opt // option for filling recruitment in terminal year+1 - used in agepro.bsn file only (1=SR,
2=geomean)
 !! ICHECK(fillR_opt);
  init_int Ravg_start
 !! ICHECK(Ravg_start);
  init_int Ravg_end
 !! ICHECK(Ravg_end);

  init_int make_Rfile // option to create rdat file of input and output values, set to 1 to create the file, 0
to skip this feature
 !! ICHECK(make_Rfile);

  init_int test_value
 !! ICHECK(test_value)
 !! cout << "test value = " << test_value << endl; //CHECK
 !! cout << "input complete" << endl;

  number ntemp0
  number SR_spawners_per_recruit
  vector s_per_r_vec(1,nyears)
 LOCAL_CALCS
  for (iyear=1;iyear<=nyears;iyear++)
  {
    ntemp0=1.0;
    s_per_r_vec(iyear)=0.0;
    for (iage=1;iage<nages;iage++)
    {
      s_per_r_vec(iyear)+=ntemp0*fecundity(iyear,iage)*mfexp(-1.0*fracyearSSB*M(iyear,iage));
      ntemp0*=mfexp(-M(iyear,iage));
    }
    ntemp0/=(1.0-mfexp(-M(iyear,nages)));
```

```
    s_per_r_vec(iyear)+=ntemp0*fecundity(iyear,nages)*mfexp(-1.0*fracyearSSB*M(iyear,nages));
  }
  SR_spawners_per_recruit=s_per_r_vec(nyears); // use last year calculations for SR curve
 END_CALCS

//**************************************************************************
PARAMETER_SECTION
  init_bounded_number_vector sel_params(1,nselparm,sel_lo,sel_hi,sel_phase)
  init_bounded_vector log_Fmult_year1(1,nfleets,-15.,2.,phase_Fmult_year1)
  init_bounded_matrix log_Fmult_devs(1,nfleets,2,nyears,-15.,15.,phase_Fmult_devs)
  init_bounded_dev_vector log_recruit_devs(1,nyears,-15.,15.,phase_recruit_devs)
  init_bounded_vector log_N_year1_devs(2,nages,-15.,15.,phase_N_year1_devs)
  init_bounded_vector log_q_year1(1,nindices,-30,5,phase_q_year1)
  init_bounded_matrix log_q_devs(1,nindices,2,index_nobs,-15.,15.,phase_q_devs)
  init_bounded_number_vector index_sel_params(1,nindexselparms,indexsel_lo,indexsel_hi,indexsel_phase)
  init_bounded_number log_SR_scaler(-1.0,200,phase_SR_scaler)
  init_bounded_number SR_steepness(0.20001,1.0,phase_steepness)
  vector sel_likely(1,nselparm)
  vector sel_stdresid(1,nselparm)
  number sel_rmse
  number sel_rmse_nobs
  number sum_sel_lambda
  number sum_sel_lambda_likely
  matrix indexsel(1,nindices,1,nages)
  vector indexsel_likely(1,nindexselparms)
  vector indexsel_stdresid(1,nindexselparms)
  number indexsel_rmse
  number indexsel_rmse_nobs
  number sum_indexsel_lambda
  number sum_indexsel_lambda_likely
  matrix log_Fmult(1,nfleets,1,nyears)
  matrix Fmult(1,nfleets,1,nyears)
  matrix NAA(1,nyears,1,nages)
  matrix temp_NAA(1,nyears,1,nages)
  matrix temp_BAA(1,nyears,1,nages)
  matrix temp_PAA(1,nyears,1,nages)
  matrix FAA_tot(1,nyears,1,nages)
  matrix Z(1,nyears,1,nages)
  matrix S(1,nyears,1,nages)
  matrix Catch_stdresid(1,nfleets,1,nyears)
  matrix Discard_stdresid(1,nfleets,1,nyears)
  matrix Catch_tot_fleet_pred(1,nfleets,1,nyears)
  matrix Discard_tot_fleet_pred(1,nfleets,1,nyears)
  3darray CAA_pred(1,nfleets,1,nyears,1,nages)
  3darray Discard_pred(1,nfleets,1,nyears,1,nages)
  3darray CAA_prop_pred(1,nfleets,1,nyears,sel_start_age,sel_end_age)
  3darray Discard_prop_pred(1,nfleets,1,nyears,sel_start_age,sel_end_age)
  3darray FAA_by_fleet_dir(1,nfleets,1,nyears,1,nages)
  3darray FAA_by_fleet_Discard(1,nfleets,1,nyears,1,nages)
  matrix sel_by_block(1,nselblocks,1,nages)
  3darray sel_by_fleet(1,nfleets,1,nyears,1,nages)
  vector temp_sel_over_time(1,nyears)
  number temp_sel_fix
  number temp_Fmult_max
  number Fmult_max_pen
  matrix q_by_index(1,nindices,1,index_nobs)
  matrix temp_sel(1,nyears,1,nages)
  vector temp_sel2(1,nages)
  matrix index_pred(1,nindices,1,index_nobs)
  3darray output_index_prop_obs(1,nindices,1,nyears,1,nages)
  3darray output_index_prop_pred(1,nindices,1,nyears,1,nages)
  matrix index_Neff_init(1,nindices,1,nyears)
  matrix index_Neff_est(1,nindices,1,nyears)
  3darray index_prop_pred(1,nindices,1,index_nobs,1,nages)
  number new_Neff_catch
  number new_Neff_discard
  number ntemp
  number SR_S0
  number SR_R0
  number SR_alpha
  number SR_beta
```

```
vector S0_vec(1,nyears)
vector R0_vec(1,nyears)
vector steepness_vec(1,nyears)
vector SR_pred_recruits(1,nyears+1)
number likely_SR_sigma
vector SR_stdresid(1,nyears)
number SR_rmse
number SR_rmse_nobs
vector RSS_sel_devs(1,nfleets)
vector RSS_catch_tot_fleet(1,nfleets)
vector RSS_Discard_tot_fleet(1,nfleets)
vector catch_tot_likely(1,nfleets)
vector discard_tot_likely(1,nfleets)
number likely_catch
number likely_Discard
vector RSS_ind(1,nindices)
vector RSS_ind_sigma(1,nindices)
vector likely_ind(1,nindices)
matrix index_stdresid(1,nindices,1,index_nobs)
number likely_index_age_comp
number fpenalty
number fpenalty_lambda
vector Fmult_year1_stdresid(1,nfleets)
number Fmult_year1_rmse
number Fmult_year1_rmse_nobs
vector Fmult_year1_likely(1,nfleets)
vector Fmult_devs_likely(1,nfleets)
matrix Fmult_devs_stdresid(1,nfleets,1,nyears)
vector Fmult_devs_fleet_rmse(1,nfleets)
vector Fmult_devs_fleet_rmse_nobs(1,nfleets)
number Fmult_devs_rmse
number Fmult_devs_rmse_nobs
number N_year1_likely
vector N_year1_stdresid(2,nages)
number N_year1_rmse
number N_year1_rmse_nobs
vector nyear1temp(1,nages)
vector q_year1_likely(1,nindices)
vector q_year1_stdresid(1,nindices)
number q_year1_rmse
number q_year1_rmse_nobs
vector q_devs_likely(1,nindices)
matrix q_devs_stdresid(1,nindices,1,index_nobs)
number q_devs_rmse
number q_devs_rmse_nobs
number steepness_likely
number steepness_stdresid
number steepness_rmse
number steepness_rmse_nobs
number SR_scaler_likely
number SR_scaler_stdresid
number SR_scaler_rmse
number SR_scaler_rmse_nobs
matrix effective_sample_size(1,nfleets,1,nyears)
matrix effective_Discard_sample_size(1,nfleets,1,nyears)
vector Neff_stage2_mult_catch(1,nfleets)
vector Neff_stage2_mult_discard(1,nfleets)
vector Neff_stage2_mult_index(1,nindices)
vector mean_age_obs(1,nyears)
vector mean_age_pred(1,nyears)
vector mean_age_pred2(1,nyears)
vector mean_age_resid(1,nyears)
vector mean_age_sigma(1,nyears)
number mean_age_x
number mean_age_n
number mean_age_delta
number mean_age_mean
number mean_age_m2
vector temp_Fmult(1,nfleets)
number tempU
number tempN
```

```
number tempB
number tempUd
number tempNd
number tempBd
number trefU
number trefN
number trefB
number trefUd
number trefNd
number trefBd
number Fref_report
number Fref
vector freftemp(1,nages)
vector nreftemp(1,nages)
vector Freport_U(1,nyears)
vector Freport_N(1,nyears)
vector Freport_B(1,nyears)
sdreport_vector Freport(1,nyears)
sdreport_vector TotJan1B(1,nyears)
sdreport_vector SSB(1,nyears)
sdreport_vector ExploitableB(1,nyears)
sdreport_vector recruits(1,nyears)
matrix SSBfracZ(1,nyears,1,nages)
vector final_year_total_sel(1,nages)
vector dir_F(1,nages)
vector Discard_F(1,nages)
vector proj_nondir_F(1,nages)
vector proj_dir_sel(1,nages)
vector proj_Discard_sel(1,nages)
matrix proj_NAA(1,nprojyears,1,nages)
vector proj_Fmult(1,nprojyears)
vector Ftemp(1,nages)
vector Ztemp(1,nages)
vector proj_TotJan1B(1,nprojyears)
vector proj_SSB(1,nprojyears)
number SSBtemp
number denom
matrix proj_F_dir(1,nprojyears,1,nages)
matrix proj_F_Discard(1,nprojyears,1,nages)
matrix proj_F_nondir(1,nprojyears,1,nages)
matrix proj_Z(1,nprojyears,1,nages)
matrix proj_SSBfracZ(1,nprojyears,1,nages)
matrix proj_catch(1,nprojyears,1,nages)
matrix proj_Discard(1,nprojyears,1,nages)
matrix proj_yield(1,nprojyears,1,nages)
vector proj_total_yield(1,nprojyears)
vector proj_total_Discard(1,nprojyears)
vector output_prop_obs(1,nages)
vector output_prop_pred(1,nages)
vector output_Discard_prop_obs(1,nages)
vector output_Discard_prop_pred(1,nages)
vector NAAbsn(1,nages)
number temp_sum
number temp_sum2
number A
number B
number C
number f
number z
number SPR_Fmult
number YPR_Fmult
number SPR
number SPRatio
number YPR
number S_F
number R_F
number slope_origin
number slope
number F30SPR
number F40SPR
number Fmsy
```

```
  number F01
  number Fmax
  number F30SPR_report
  number F40SPR_report
  number F01_report
  number Fmax_report
  number Fcurrent
  number F30SPR_slope
  number F40SPR_slope
  number Fmsy_slope
  number F01_slope
  number Fmax_slope
  number Fcurrent_slope
  number SSmsy
  number tempR
  vector tempFmult(1,nyears) // Liz added
  sdreport_number MSY
  sdreport_number SSBmsy_report
  sdreport_number Fmsy_report
  sdreport_number SSBmsy_ratio
  sdreport_number Fmsy_ratio
  objective_function_value obj_fun

PRELIMINARY_CALCS_SECTION
// subset only used index information
  ind=0;
  for (indavail=1;indavail<=navailindices;indavail++)
  {
     if (use_index(indavail)==1)
     {
       ind+=1;
       lambda_ind(ind)=lambda_ind_ini(indavail);
       lambda_q_year1(ind)=lambda_q_year1_ini(indavail);
       q_year1_CV(ind)=q_year1_CV_ini(indavail);
       lambda_q_devs(ind)=lambda_q_devs_ini(indavail);
       q_devs_CV(ind)=q_devs_CV_ini(indavail);
       q_year1_ini(ind)=q_year1_iniavail(indavail);
     }
  }

  if (ignore_guesses==0)
  {
     NAA(1)=NAA_year1_ini;
     log_Fmult_year1=log(Fmult_year1_ini);
     log_q_year1=log(q_year1_ini);
     log_SR_scaler=log(SR_scaler_ini);
     SR_steepness=SR_steepness_ini;
     for (k=1;k<=nselparm;k++)
     {
        sel_params(k)=sel_initial(k);
     }
     for (k=1;k<=nindexselparms;k++)
     {
        index_sel_params(k)=indexsel_initial(k);
     }
  }

  delta=0.00001;

// convert remaining CVs to variances
  Fmult_year1_sigma2=log(elem_prod(Fmult_year1_CV,Fmult_year1_CV)+1.0);
  Fmult_year1_sigma=sqrt(Fmult_year1_sigma2);
  Fmult_devs_sigma2=log(elem_prod(Fmult_devs_CV,Fmult_devs_CV)+1.0);
  Fmult_devs_sigma=sqrt(Fmult_devs_sigma2);
  N_year1_sigma2=log(N_year1_CV*N_year1_CV+1.0);
  N_year1_sigma=sqrt(N_year1_sigma2);
  q_year1_sigma2=log(elem_prod(q_year1_CV,q_year1_CV)+1.0);
  q_year1_sigma=sqrt(q_year1_sigma2);
  q_devs_sigma2=log(elem_prod(q_devs_CV,q_devs_CV)+1.0);
  q_devs_sigma=sqrt(q_devs_sigma2);
  steepness_sigma2=log(steepness_CV*steepness_CV+1.0);
```

```
  steepness_sigma=sqrt(steepness_sigma2);
  SR_scaler_sigma2=log(SR_scaler_CV*SR_scaler_CV+1.0);
  SR_scaler_sigma=sqrt(SR_scaler_sigma2);

// compute multinomial constants for catch and discards at age, if requested
  catch_prop_like_const=0.0;
  discard_prop_like_const=0.0;
  if (use_likelihood_constants == 1)
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
     {
        for (iyear=1;iyear<=nyears;iyear++)
        {
          if (input_eff_samp_size_catch(ifleet,iyear) > 0)
          {
             nfact_in=input_eff_samp_size_catch(ifleet,iyear);
             get_log_factorial();
             catch_prop_like_const+=-1.0*nfact_out;  // negative for the total
             for (iage=sel_start_age(ifleet);iage<=sel_end_age(ifleet);iage++)
             {
                nfact_in=double(input_eff_samp_size_catch(ifleet,iyear))*CAA_prop_obs(ifleet,iyear,iage)+0.5;
// +0.5 to round instead of truncate nfact_in
                get_log_factorial();
                catch_prop_like_const+=nfact_out;   // positive for the parts
             }
          }
          if (input_eff_samp_size_discard(ifleet,iyear) > 0)
          {
             nfact_in=input_eff_samp_size_discard(ifleet,iyear);
             get_log_factorial();
             discard_prop_like_const+=-1.0*nfact_out; // negative for the total
             for (iage=sel_start_age(ifleet);iage<=sel_end_age(ifleet);iage++)
             {

nfact_in=double(input_eff_samp_size_discard(ifleet,iyear))*Discard_prop_obs(ifleet,iyear,iage)+0.5;
                get_log_factorial();
                discard_prop_like_const+=nfact_out;   // positive for the parts
             }
          }
        }
     }
  }

// compute multinomial constants for index, if requested
  index_prop_like_const=0.0;
  if (use_likelihood_constants == 1)
  {
     for (ind=1;ind<=nindices;ind++)
     {
        if (index_estimate_proportions(ind)==1)
        {
           for (i=1;i<=index_nobs(ind);i++)
           {
              if (input_eff_samp_size_index(ind,i) > 0)
              {
                 nfact_in=input_eff_samp_size_index(ind,i);
                 get_log_factorial();
                 index_prop_like_const+=-1.0*nfact_out;    // negative for total
                 for (iage=index_start_age(ind);iage<=index_end_age(ind);iage++)
                 {
                    nfact_in=double(input_eff_samp_size_index(ind,i))*index_prop_obs(ind,i,iage)+0.5;
                    get_log_factorial();
                    index_prop_like_const+=nfact_out;      // positive for the parts
                 }
              }
           }
        }
     }
  }

// selectivity likelihood constants
```

```
  sel_like_const=0.0;
  if (use_likelihood_constants == 1)
  {
     for (k=1;k<=nselparm;k++)
     {
        if (sel_phase(k) >= 1)
        {
           sel_like_const(k)=0.5*log(2.0*pi)+log(sel_initial(k));
        }
     }
  }

// index selectivity likelihood constants
  indexsel_like_const=0.0;
  if (use_likelihood_constants == 1)
  {
     for (k=1;k<=nindexselparms;k++)
     {
        if (indexsel_phase(k) >= 1)
        {
           indexsel_like_const(k)=0.5*log(2.0*pi)+log(indexsel_initial(k));
        }
     }
  }

// rest of likelihood constants
  if (use_likelihood_constants == 1)
  {
     Fmult_year1_like_const=0.5*log(2.0*pi)+log(Fmult_year1_ini);
     Fmult_devs_like_const=0.5*log(2.0*pi);
     N_year1_like_const=0.5*log(2.0*pi);
     q_year1_like_const=0.5*log(2.0*pi)+log(q_year1_ini);
     q_devs_like_const=0.5*log(2.0*pi);
     steepness_like_const=0.5*log(2.0*pi)+log(SR_steepness_ini);
     SR_scaler_like_const=0.5*log(2.0*pi)+log(SR_scaler_ini);
  }
  else
  {
     Fmult_year1_like_const=0.0;
     Fmult_devs_like_const=0.0;
     N_year1_like_const=0.0;
     q_year1_like_const=0.0;
     q_devs_like_const=0.0;
     steepness_like_const=0.0;
     SR_scaler_like_const=0.0;
  }

// set dev vectors to zero
  log_Fmult_devs.initialize();
  log_recruit_devs.initialize();
  log_N_year1_devs.initialize();
  log_q_devs.initialize();

// initialize MSY related sdreport variables
  MSY.initialize();
  SSBmsy_report.initialize();
  Fmsy_report.initialize();
  SSBmsy_ratio.initialize();
  Fmsy_ratio.initialize();

  debug=0; // debug checks commented out to speed calculations

//****************************************************************************************
PROCEDURE_SECTION
                                  //  if (debug==1) cout << "starting procedure section" << endl;
  get_SR();                      //  if (debug==1) cout << "got SR" << endl;
  get_selectivity();              //  if (debug==1) cout << "got selectivity" << endl;
  get_mortality_rates();          //  if (debug==1) cout << "got mortality rates" << endl;
  get_numbers_at_age();           //  if (debug==1) cout << "got numbers at age" << endl;
  get_Freport();                  //  if (debug==1) cout << "got Freport" << endl;
  get_predicted_catch();          //  if (debug==1) cout << "got predicted catch" << endl;
```

```
  get_q();                              //  if (debug==1) cout << "got q" << endl;
  get_predicted_indices();              //  if (debug==1) cout << "got predicted indices" << endl;
  compute_the_objective_function();     //  if (debug==1) cout << "computed objective function" << endl;
  if (last_phase() || mceval_phase())
  {
     get_proj_sel();                    //  if (debug==1) cout <<"got proj sel" << endl;
     get_Fref();                        //  if (debug==1) cout <<"got Fref" << endl;
     get_multinomial_multiplier();      //  if (debug==1) cout <<"got multinomial multiplier" << endl;
  }
  if (mceval_phase())
  {
     write_MCMC();
  }                                     //  if (debug==1) cout << "  . . . end of procedure section" << endl;
//**************************************************************************************************

FUNCTION get_SR
// converts stock recruitment scaler and steepness to alpha and beta for Beverton-Holt SR
// note use of is_SR_scaler_R variable to allow user to enter guess for either R0 or SSB0
  if(is_SR_scaler_R==1)
  {
    SR_R0=mfexp(log_SR_scaler);
    SR_S0=SR_spawners_per_recruit*SR_R0;
  }
  else
  {
    SR_S0=mfexp(log_SR_scaler);
    SR_R0=SR_S0/SR_spawners_per_recruit;
  }
  SR_alpha=4.0*SR_steepness*SR_R0/(5.0*SR_steepness-1.0);
  SR_beta=SR_S0*(1.0-SR_steepness)/(5.0*SR_steepness-1.0);
  // now compute year specific vectors of R0, S0, and steepness
  for (iyear=1;iyear<=nyears;iyear++)
  {
    steepness_vec(iyear)=0.2*SR_alpha*s_per_r_vec(iyear)/(0.8*SR_beta+0.2*SR_alpha*s_per_r_vec(iyear));
    R0_vec(iyear)=(SR_alpha*s_per_r_vec(iyear)-SR_beta)/s_per_r_vec(iyear);
    S0_vec(iyear)=s_per_r_vec(iyear)*R0_vec(iyear);
  }

FUNCTION get_selectivity
  dvariable sel_alpha1;
  dvariable sel_beta1;
  dvariable sel_alpha2;
  dvariable sel_beta2;
  dvariable sel_temp;
  dvariable sel1;
  dvariable sel2;
// start by computing selectivity for each block
  k=0;
  for (i=1;i<=nselblocks;i++) {
     if (sel_option(i)==1) {
        for (iage=1;iage<=nages;iage++){
           k+=1;
           sel_by_block(i,iage)=sel_params(k);
        }
     }
     if (sel_option(i)==2) {
        sel_alpha1=sel_params(k+1);
        sel_beta1=sel_params(k+2);
        k+=2;
        for (iage=1;iage<=nages;iage++) {
           sel_by_block(i,iage)=1.0/(1.0+mfexp((sel_alpha1-double(iage))/sel_beta1));
        }
        sel_temp=max(sel_by_block(i));
        sel_by_block(i)/=sel_temp;
     }
     if (sel_option(i)==3) {
        sel_alpha1=sel_params(k+1);
        sel_beta1=sel_params(k+2);
        sel_alpha2=sel_params(k+3);
        sel_beta2=sel_params(k+4);
        k+=4;
```

```
        for (iage=1;iage<=nages;iage++) {
           sel1=1.0/(1.0+mfexp((sel_alpha1-double(iage))/sel_beta1));
           sel2=1.0-1.0/(1.0+mfexp((sel_alpha2-double(iage))/sel_beta2));
           sel_by_block(i,iage)=sel1*sel2;
        }
        sel_temp=max(sel_by_block(i));
        sel_by_block(i)/=sel_temp;
     }
  }
// now fill in selectivity for each fleet and year according to block
  for (ifleet=1;ifleet<=nfleets;ifleet++) {
     for (iyear=1;iyear<=nyears;iyear++) {
        sel_by_fleet(ifleet,iyear)=sel_by_block(sel_blocks(ifleet,iyear));
     }
  }

FUNCTION get_mortality_rates
// compute directed and discard F by fleet then sum to form total F at age matrix
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     log_Fmult(ifleet,1)=log_Fmult_year1(ifleet);
     if (active(log_Fmult_devs))
     {
         for (iyear=2;iyear<=nyears;iyear++)
             log_Fmult(ifleet,iyear)=log_Fmult(ifleet,iyear-1)+log_Fmult_devs(ifleet,iyear);
     }
     else
     {
         for (iyear=2;iyear<=nyears;iyear++)
             log_Fmult(ifleet,iyear)=log_Fmult_year1(ifleet);
     }
  }
  FAA_tot=0.0;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     for (iyear=1;iyear<=nyears;iyear++)
     {
       for (iage=1;iage<=nages;iage++)
       {

FAA_by_fleet_dir(ifleet,iyear,iage)=(mfexp(log_Fmult(ifleet,iyear))*sel_by_fleet(ifleet,iyear,iage))*(1.0-
proportion_release(ifleet,iyear,iage));

FAA_by_fleet_Discard(ifleet,iyear,iage)=(mfexp(log_Fmult(ifleet,iyear))*sel_by_fleet(ifleet,iyear,iage))*(propor
tion_release(ifleet,iyear,iage)*release_mort(ifleet));
       }
     }
     FAA_tot+=FAA_by_fleet_dir(ifleet)+FAA_by_fleet_Discard(ifleet);
  }
// add fishing and natural mortality to get total mortality
  for (iyear=1;iyear<=nyears;iyear++)
     Z(iyear)=FAA_tot(iyear)+M(iyear);
  S=mfexp(-1.0*Z);
  SSBfracZ=mfexp(-1.0*fracyearSSB*Z); // for use in SSB calcuations

FUNCTION get_numbers_at_age
// get N at age in year 1
  if (phase_N_year1_devs>0)
  {
     for (iage=2;iage<=nages;iage++)
     {
        NAA(1,iage)=NAA_year1_ini(iage)*mfexp(log_N_year1_devs(iage));
     }
  }
// compute initial SSB to derive R in first year
  SSB(1)=0.0;
  for (iage=2;iage<=nages;iage++)
  {
     SSB(1)+=NAA(1,iage)*SSBfracZ(1,iage)*fecundity(1,iage);  // note SSB in year 1 does not include age 1 to
estimate pred_R in year 1
  }
```

```
  SR_pred_recruits(1)=SR_alpha*SSB(1)/(SR_beta+SSB(1));
  NAA(1,1)=SR_pred_recruits(1)*mfexp(log_recruit_devs(1));
  SSB(1)+=NAA(1,1)*SSBfracZ(1,1)*fecundity(1,1);   // now SSB in year 1 is complete and can be used for pred_R
in year 2
// fill out rest of matrix
  for (iyear=2;iyear<=nyears;iyear++)
  {
     SR_pred_recruits(iyear)=SR_alpha*SSB(iyear-1)/(SR_beta+SSB(iyear-1));
     NAA(iyear,1)=SR_pred_recruits(iyear)*mfexp(log_recruit_devs(iyear));
     for (iage=2;iage<=nages;iage++)
         NAA(iyear,iage)=NAA(iyear-1,iage-1)*S(iyear-1,iage-1);
     NAA(iyear,nages)+=NAA(iyear-1,nages)*S(iyear-1,nages);
     SSB(iyear)=elem_prod(NAA(iyear),SSBfracZ(iyear))*fecundity(iyear);
  }
  SR_pred_recruits(nyears+1)=SR_alpha*SSB(nyears)/(SR_beta+SSB(nyears));
  for (iyear=1;iyear<=nyears;iyear++)
  {
     recruits(iyear)=NAA(iyear,1);
  }
// compute two other biomass time series
  for (iyear=1;iyear<=nyears;iyear++)
  {
     TotJan1B(iyear)=NAA(iyear)*WAAjan1b(iyear);
     ExploitableB(iyear)=elem_prod(NAA(iyear),FAA_tot(iyear))*WAAcatchall(iyear)/max(FAA_tot(iyear));
  }

FUNCTION get_Freport
// calculates an average F for a range of ages in each year under three weighting schemes
  for (iyear=1;iyear<=nyears;iyear++){
     tempU=0.0;
     tempN=0.0;
     tempB=0.0;
     tempUd=0.0;
     tempNd=0.0;
     tempBd=0.0;
     for (iage=Freport_agemin;iage<=Freport_agemax;iage++)
     {
       tempU+=FAA_tot(iyear,iage);
       tempN+=FAA_tot(iyear,iage)*NAA(iyear,iage);
       tempB+=FAA_tot(iyear,iage)*NAA(iyear,iage)*WAAjan1b(iyear,iage);
       tempUd+=1.0;
       tempNd+=NAA(iyear,iage);
       tempBd+=NAA(iyear,iage)*WAAjan1b(iyear,iage);
     }
     // April 2012 error trap addition
     if (tempUd <= 0.) Freport_U(iyear)=0.0;
     else Freport_U(iyear)=tempU/tempUd;
     if (tempNd <= 0.) Freport_N(iyear)=Freport_U(iyear);
     else Freport_N(iyear)=tempN/tempNd;
     if (tempBd <= 0.) Freport_B(iyear)=Freport_U(iyear);
     else Freport_B(iyear)=tempB/tempBd;
  }
  if (Freport_wtopt==1) Freport=Freport_U;
  if (Freport_wtopt==2) Freport=Freport_N;
  if (Freport_wtopt==3) Freport=Freport_B;

FUNCTION get_predicted_catch
// assumes continuous F using Baranov equation
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     CAA_pred(ifleet)=elem_prod(elem_div(FAA_by_fleet_dir(ifleet),Z),elem_prod(1.0-S,NAA));
     Discard_pred(ifleet)=elem_prod(elem_div(FAA_by_fleet_Discard(ifleet),Z),elem_prod(1.0-S,NAA));
  }
// now compute proportions at age and total weight of catch
  for (iyear=1;iyear<=nyears;iyear++)
  {
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
       CAA_prop_pred(ifleet,iyear)=0.0;
       Discard_prop_pred(ifleet,iyear)=0.0;
```

```
Catch_tot_fleet_pred(ifleet,iyear)=sum(CAA_pred(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet)));

Discard_tot_fleet_pred(ifleet,iyear)=sum(Discard_pred(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet)));
       if (Catch_tot_fleet_pred(ifleet,iyear)>0.0)

CAA_prop_pred(ifleet,iyear)=CAA_pred(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet))/Catch_tot_fleet_pr
ed(ifleet,iyear);
       if (Discard_tot_fleet_pred(ifleet,iyear)>0.0)

Discard_prop_pred(ifleet,iyear)=Discard_pred(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet))/Discard_to
t_fleet_pred(ifleet,iyear);

Catch_tot_fleet_pred(ifleet,iyear)=CAA_pred(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet))*WAAcatchfle
et(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet));

Discard_tot_fleet_pred(ifleet,iyear)=Discard_pred(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet))*WAAdi
scardfleet(ifleet,iyear)(sel_start_age(ifleet),sel_end_age(ifleet));
       for (iage=1;iage<=nages;iage++)
       {
          if (CAA_prop_pred(ifleet,iyear,iage)<1.e-15)
             CAA_prop_pred(ifleet,iyear,iage)=1.0e-15;
          if (Discard_prop_pred(ifleet,iyear,iage)<1.e-15)
             Discard_prop_pred(ifleet,iyear,iage)=1.0e-15;
       }
    }
  }

FUNCTION get_q
// catchability for each index, can be a random walk if q_devs turned on
  for (ind=1;ind<=nindices;ind++)
  {
     q_by_index(ind,1)=mfexp(log_q_year1(ind));
     if (active(log_q_devs))
     {
         for (i=2;i<=index_nobs(ind);i++)
             q_by_index(ind,i)=q_by_index(ind,i-1)*mfexp(log_q_devs(ind,i));
     }
     else
     {
         for (i=2;i<=index_nobs(ind);i++)
             q_by_index(ind,i)=q_by_index(ind,1);
     }
  }

FUNCTION get_predicted_indices
  dvariable sel_alpha1;
  dvariable sel_beta1;
  dvariable sel_alpha2;
  dvariable sel_beta2;
  dvariable sel_temp;
  dvariable sel1;
  dvariable sel2;
// get selectivity for each index
  k=0;
  for (ind=1;ind<=nindices;ind++)
  {
     if (index_sel_choice(ind)>0)
     {
         temp_sel=sel_by_fleet(index_sel_choice(ind));
         if (index_sel_option(ind)==1) k+=nages;
         if (index_sel_option(ind)==2) k+=2;
         if (index_sel_option(ind)==3) k+=4;

     }
     else
     {
         if (index_sel_option(ind)==1)
         {
             for (iage=1;iage<=nages;iage++)
             {
```

```
                k+=1;
                temp_sel2(iage)=index_sel_params(k);
            }
        }
        if (index_sel_option(ind)==2)
        {
            sel_alpha1=index_sel_params(k+1);
            sel_beta1=index_sel_params(k+2);
            k+=2;
            for (iage=1;iage<=nages;iage++)
            {
               temp_sel2(iage)=1.0/(1.0+mfexp((sel_alpha1-double(iage))/sel_beta1));
            }
            sel_temp=max(temp_sel2);
            temp_sel2/=sel_temp;
        }
        if (index_sel_option(ind)==3)
        {
            sel_alpha1=index_sel_params(k+1);
            sel_beta1=index_sel_params(k+2);
            sel_alpha2=index_sel_params(k+3);
            sel_beta2=index_sel_params(k+4);
            k+=4;
            for (iage=1;iage<=nages;iage++)
            {
               sel1=1.0/(1.0+mfexp((sel_alpha1-double(iage))/sel_beta1));
               sel2=1.0-1.0/(1.0+mfexp((sel_alpha2-double(iage))/sel_beta2));
               temp_sel2(iage)=sel1*sel2;
            }
            sel_temp=max(temp_sel2);
            temp_sel2/=sel_temp;
        }
        for (iyear=1;iyear<=nyears;iyear++)
        {
            temp_sel(iyear)=temp_sel2;
        }
    }
    indexsel(ind)=temp_sel(1);
// determine when the index should be applied
    if (index_month(ind)==-1)
    {
        temp_NAA=elem_prod(NAA,elem_div(1.0-S,Z));
    }
    else
    {
        temp_NAA=elem_prod(NAA,mfexp(-1.0*((index_month(ind)-1.0)/12.0)*Z));
    }
    temp_BAA=elem_prod(temp_NAA,index_WAA(ind));
// compute the predicted index for each year where observed value > 0
    if (index_units_aggregate(ind)==1)
    {
        temp_PAA=temp_BAA;
    }
    else
    {
        temp_PAA=temp_NAA;
    }
    for (i=1;i<=index_nobs(ind);i++)
    {
        j=index_time(ind,i);
        index_pred(ind,i)=q_by_index(ind,i)*sum(elem_prod(
            temp_PAA(j)(index_start_age(ind),index_end_age(ind)) ,
            temp_sel(j)(index_start_age(ind),index_end_age(ind))));
    }
// compute index proportions at age if necessary
    if (index_units_proportions(ind)==1)
    {
        temp_PAA=temp_BAA;
    }
    else
    {
```

```
        temp_PAA=temp_NAA;
    }
    index_prop_pred(ind)=0.0;
    if (index_estimate_proportions(ind)==1)
    {
        for (i=1;i<=index_nobs(ind);i++)
        {
            j=index_time(ind,i);
            if (index_pred(ind,i)>0.0)
            {
                for (iage=index_start_age(ind);iage<=index_end_age(ind);iage++)
                {
                    index_prop_pred(ind,i,iage)=q_by_index(ind,i)*temp_PAA(j,iage)*temp_sel(j,iage);
                }
                if (sum(index_prop_pred(ind,i)) > 0)
                    index_prop_pred(ind,i)/=sum(index_prop_pred(ind,i));
                for (iage=index_start_age(ind);iage<=index_end_age(ind);iage++)
                {
                    if (index_prop_pred(ind,i,iage)<1.e-15)
                       index_prop_pred(ind,i,iage)=1.e-15;
                }
            }
        }
    }
  }

FUNCTION get_proj_sel
// creates overall directed and discard selectivity patterns and sets bycatch F at age
  dir_F=0.0;
  Discard_F=0.0;
  proj_nondir_F=0.0;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     if (directed_fleet(ifleet)==1)
     {
        dir_F+=FAA_by_fleet_dir(ifleet,nyears);
        Discard_F+=FAA_by_fleet_Discard(ifleet,nyears);
     }
     else
     {
        proj_nondir_F+=FAA_by_fleet_dir(ifleet,nyears);
     }
  }
  proj_dir_sel=dir_F/max(dir_F);
  proj_Discard_sel=Discard_F/max(dir_F);

FUNCTION get_Fref
// calculates a number of common F reference points using bisection algorithm
  A=0.0;
  B=5.0;
  for (iloop=1;iloop<=20;iloop++)
  {
     C=(A+B)/2.0;
     SPR_Fmult=C;
     get_SPR();
     if (SPR/SR_spawners_per_recruit<0.30)
     {
        B=C;
     }
     else
     {
        A=C;
     }
  }
  F30SPR=C;
  Fref=F30SPR;
  get_Freport_ref();
  F30SPR_report=Fref_report;
  F30SPR_slope=1.0/SPR;
  A=0.0;
  B=5.0;
```

```
  for (iloop=1;iloop<=20;iloop++)
  {
     C=(A+B)/2.0;
     SPR_Fmult=C;
     get_SPR();
     if (SPR/SR_spawners_per_recruit<0.40)
     {
        B=C;
     }
     else
     {
        A=C;
     }
  }
  F40SPR=C;
  Fref=F40SPR;
  get_Freport_ref();
  F40SPR_report=Fref_report;
  F40SPR_slope=1.0/SPR;
  A=0.0;
  B=3.0;
  for (iloop=1;iloop<=20;iloop++)
  {
    C=(A+B)/2.0;
    SPR_Fmult=C+delta;
    get_SPR();
    S_F=SR_alpha*SPR-SR_beta;
    R_F=S_F/SPR;
    YPR_Fmult=C+delta;
    get_YPR();
    slope=R_F*YPR;
    SPR_Fmult=C;
    get_SPR();
    S_F=SR_alpha*SPR-SR_beta;
    R_F=S_F/SPR;
    YPR_Fmult=C;
    get_YPR();
    slope-=R_F*YPR;
//    slope/=delta; only care pos or neg
    if(slope>0.0)
    {
       A=C;
    }
    else
    {
       B=C;
    }
  }
  Fmsy=C;
  Fref=Fmsy;
  get_Freport_ref();
  Fmsy_report=Fref_report;
  SSmsy=S_F;
  SSBmsy_report=SSmsy;
  if (SSmsy>0.0)
    SSBmsy_ratio=SSB(nyears)/SSmsy;
  MSY=YPR*R_F;
  SPR_Fmult=Fmsy;
  get_SPR();
  Fmsy_slope=1.0/SPR;
  YPR_Fmult=delta;
  get_YPR();
  slope_origin=YPR/delta;
  A=0.0;
  B=5.0;
  for (iloop=1;iloop<=20;iloop++)
  {
     C=(A+B)/2.0;
     YPR_Fmult=C+delta;
     get_YPR();
     slope=YPR;
```

```
      YPR_Fmult=C;
      get_YPR();
      slope-=YPR;
      slope/=delta;
      if (slope<0.10*slope_origin)
      {
         B=C;
      }
      else
      {
         A=C;
      }
   }
   F01=C;
   Fref=F01;
   get_Freport_ref();
   F01_report=Fref_report;
   SPR_Fmult=F01;
   get_SPR();
   F01_slope=1.0/SPR;
   A=0.0;
   B=10.0;
   for (iloop=1;iloop<=20;iloop++)
   {
      C=(A+B)/2.0;
      YPR_Fmult=C+delta;
      get_YPR();
      slope=YPR;
      YPR_Fmult=C;
      get_YPR();
      slope-=YPR;
      slope/=delta;
      if (slope<0.0)
      {
         B=C;
      }
      else
      {
         A=C;
      }
   }
   Fmax=C;
   Fref=Fmax;
   get_Freport_ref();
   Fmax_report=Fref_report;
   SPR_Fmult=Fmax;
   get_SPR();
   Fmax_slope=1.0/SPR;
   Fcurrent=max(FAA_tot(nyears)-proj_nondir_F-Discard_F);
   SPR_Fmult=Fcurrent;
   get_SPR();
   Fcurrent_slope=1.0/SPR;
   if (Fmsy>0.0)
     Fmsy_ratio=Fcurrent/Fmsy;

FUNCTION get_Freport_ref
// Freport calculations for each of the reference points
   trefU=0.0;
   trefN=0.0;
   trefB=0.0;
   trefUd=0.0;
   trefNd=0.0;
   trefBd=0.0;
   nreftemp(1)=1.0;
   for (iage=1;iage<nages;iage++)
   {
      freftemp(iage)=Fref*(proj_dir_sel(iage)+proj_Discard_sel(iage))+proj_nondir_F(iage);
      nreftemp(iage+1)=mfexp(-1.0*(M(nyears,iage)+freftemp(iage)));
   }
   freftemp(nages)=Fref*(proj_dir_sel(nages)+proj_Discard_sel(nages))+proj_nondir_F(nages);
   nreftemp(nages)/=(1.0-mfexp(-1.0*(M(nyears,nages)+freftemp(nages))));
```

```
  for (iage=Freport_agemin;iage<=Freport_agemax;iage++)
  {
     trefU+=freftemp(iage);
     trefN+=freftemp(iage)*nreftemp(iage);
     trefB+=freftemp(iage)*nreftemp(iage)*WAAjan1b(nyears,iage);
     trefUd+=1.0;
     trefNd+=nreftemp(iage);
     trefBd+=nreftemp(iage)*WAAjan1b(nyears,iage);
  }
  if (Freport_wtopt==1) Fref_report=trefU/trefUd;
  if (Freport_wtopt==2) Fref_report=trefN/trefNd;
  if (Freport_wtopt==3) Fref_report=trefB/trefBd;

FUNCTION get_YPR
// simple yield per recruit calculations
  YPR=0.0;
  ntemp=1.0;
  for (iage=1;iage<nages;iage++)
  {
    f=YPR_Fmult*proj_dir_sel(iage);
    z=M(nyears,iage)+f+proj_nondir_F(iage)+YPR_Fmult*proj_Discard_sel(iage);
    YPR+=ntemp*f*WAAcatchall(nyears,iage)*(1.0-mfexp(-1.0*z))/z;
    ntemp*=mfexp(-1.0*z);
  }
  f=YPR_Fmult*proj_dir_sel(nages);
  z=M(nyears,nages)+f+proj_nondir_F(nages)+YPR_Fmult*proj_Discard_sel(nages);
  ntemp/=(1.0-mfexp(-1.0*z));
  YPR+=ntemp*f*WAAcatchall(nyears,nages)*(1.0-mfexp(-1.0*z))/z;

FUNCTION project_into_future
// project population under five possible scenarios for each year
  for (iyear=1;iyear<=nprojyears;iyear++)
  {
    proj_F_nondir(iyear)=proj_nondir_F*proj_F_nondir_mult(iyear);
    if (proj_recruit(iyear)<0.0)  // use stock-recruit relationship
    {
       if (iyear==1)
       {
          proj_NAA(iyear,1)=SR_alpha*SSB(nyears)/(SR_beta+SSB(nyears));
       }
       else
       {
          proj_NAA(iyear,1)=SR_alpha*proj_SSB(iyear-1)/(SR_beta+proj_SSB(iyear-1));
       }
    }
    else
    {
       proj_NAA(iyear,1)=proj_recruit(iyear);
    }
    if (iyear==1)
    {
       for (iage=2;iage<=nages;iage++)
          proj_NAA(1,iage)=NAA(nyears,iage-1)*S(nyears,iage-1);
       proj_NAA(1,nages)+=NAA(nyears,nages)*S(nyears,nages);
    }
    else
    {
       for (iage=2;iage<=nages;iage++)
          proj_NAA(iyear,iage)=proj_NAA(iyear-1,iage-1)*mfexp(-1.0*proj_Z(iyear-1,iage-1));
       proj_NAA(iyear,nages)+=proj_NAA(iyear-1,nages)*mfexp(-1.0*proj_Z(iyear-1,nages));
    }
    if (proj_what(iyear)==1)           // match directed yield
    {
       proj_Fmult(iyear)=3.0;  // first see if catch possible
       proj_F_dir(iyear)=proj_Fmult(iyear)*proj_dir_sel;
       proj_F_Discard(iyear)=proj_Fmult(iyear)*proj_Discard_sel;
       proj_Z(iyear)=M(nyears)+proj_F_nondir(iyear)+proj_F_dir(iyear)+proj_F_Discard(iyear);
       proj_catch(iyear)=elem_prod(elem_div(proj_F_dir(iyear),proj_Z(iyear)),elem_prod(1.0-mfexp(-
1.0*proj_Z(iyear)),proj_NAA(iyear)));
       proj_Discard(iyear)=elem_prod(elem_div(proj_F_Discard(iyear),proj_Z(iyear)),elem_prod(1.0-mfexp(-
1.0*proj_Z(iyear)),proj_NAA(iyear)));
```

```
      proj_yield(iyear)=elem_prod(proj_catch(iyear),WAAcatchall(nyears));
      proj_total_yield(iyear)=sum(proj_yield(iyear));
      proj_total_Discard(iyear)=sum(elem_prod(proj_Discard(iyear),WAAdiscardall(nyears)));
      if (proj_total_yield(iyear)>proj_target(iyear))  // if catch possible, what F needed
      {
         proj_Fmult(iyear)=0.0;
         for (iloop=1;iloop<=20;iloop++)
         {
            Ftemp=proj_Fmult(iyear)*proj_dir_sel;
            denom=0.0;
            for (iage=1;iage<=nages;iage++)
            {

Ztemp(iage)=M(nyears,iage)+proj_F_nondir(iyear,iage)+proj_Fmult(iyear)*proj_Discard_sel(iage)+Ftemp(iage);
               denom+=proj_NAA(iyear,iage)*WAAcatchall(nyears,iage)*proj_dir_sel(iage)*(1.0-mfexp(-
1.0*Ztemp(iage)))/Ztemp(iage);
            }
            proj_Fmult(iyear)=proj_target(iyear)/denom;
         }
      }
   }
   else if (proj_what(iyear)==2)      // match F%SPR
   {
      A=0.0;
      B=5.0;
      for (iloop=1;iloop<=20;iloop++)
      {
         C=(A+B)/2.0;
         SPR_Fmult=C;
         get_SPR();
         SPRatio=SPR/SR_spawners_per_recruit;
         if (SPRatio<proj_target(iyear))
         {
            B=C;
         }
         else
         {
            A=C;
         }
      }
      proj_Fmult(iyear)=C;
   }
   else if (proj_what(iyear)==3)      // project Fmsy
   {
      proj_Fmult=Fmsy;
   }
   else if (proj_what(iyear)==4)      // project Fcurrent
   {
      proj_Fmult=Fcurrent;
   }
   else if (proj_what(iyear)==5)      // project input F
   {
      proj_Fmult=proj_target(iyear);
   }
   proj_F_dir(iyear)=proj_Fmult(iyear)*proj_dir_sel;
   proj_F_Discard(iyear)=proj_Fmult(iyear)*proj_Discard_sel;
   proj_Z(iyear)=M(nyears)+proj_F_nondir(iyear)+proj_F_dir(iyear)+proj_F_Discard(iyear);
   proj_SSBfracZ(iyear)=mfexp(-1.0*fracyearSSB*proj_Z(iyear));
   proj_catch(iyear)=elem_prod(elem_div(proj_F_dir(iyear),proj_Z(iyear)),elem_prod(1.0-mfexp(-
1.0*proj_Z(iyear)),proj_NAA(iyear)));
   proj_Discard(iyear)=elem_prod(elem_div(proj_F_Discard(iyear),proj_Z(iyear)),elem_prod(1.0-mfexp(-
1.0*proj_Z(iyear)),proj_NAA(iyear)));
   proj_yield(iyear)=elem_prod(proj_catch(iyear),WAAcatchall(nyears));
   proj_total_yield(iyear)=sum(proj_yield(iyear));
   proj_total_Discard(iyear)=sum(elem_prod(proj_Discard(iyear),WAAdiscardall(nyears)));
   proj_TotJan1B(iyear)=sum(elem_prod(proj_NAA(iyear),WAAjan1b(nyears)));
   proj_SSB(iyear)=elem_prod(proj_NAA(iyear),proj_SSBfracZ(iyear))*fecundity(nyears);
 }

FUNCTION get_SPR
// simple spawners per recruit calculations
```

```
  ntemp=1.0;
  SPR=0.0;
  for (iage=1;iage<nages;iage++)
  {
    z=M(nyears,iage)+proj_nondir_F(iage)+SPR_Fmult*proj_dir_sel(iage)+SPR_Fmult*proj_Discard_sel(iage);
    SPR+=ntemp*fecundity(nyears,iage)*mfexp(-1.0*fracyearSSB*z);
    ntemp*=mfexp(-1.0*z);
  }
  z=M(nyears,nages)+proj_nondir_F(nages)+SPR_Fmult*proj_dir_sel(nages)+SPR_Fmult*proj_Discard_sel(nages);
  ntemp/=(1.0-mfexp(-1.0*z));
  SPR+=ntemp*fecundity(nyears,nages)*mfexp(-1.0*fracyearSSB*z);

FUNCTION get_multinomial_multiplier
// compute Francis (2011) stage 2 multiplier for multinomial to adjust input Neff
// Francis, R.I.C.C. 2011. Data weighting in statistical fisheries stock assessment models. CJFAS 68: 1124-1138
  Neff_stage2_mult_catch=1;
  Neff_stage2_mult_discard=1;
  Neff_stage2_mult_index=1;
  // Catch
  for (ifleet=1;ifleet<=nfleets;ifleet++){
    mean_age_obs=0.0;
    mean_age_pred=0.0;
    mean_age_pred2=0.0;
    mean_age_resid=0.0;
    for (iyear=1;iyear<=nyears;iyear++){
      for (iage=sel_start_age(ifleet);iage<=sel_end_age(ifleet);iage++){
        mean_age_obs(iyear) += CAA_prop_obs(ifleet,iyear,iage)*iage;
        mean_age_pred(iyear) += CAA_prop_pred(ifleet,iyear,iage)*iage;
        mean_age_pred2(iyear) += CAA_prop_pred(ifleet,iyear,iage)*iage*iage;
      }
    }
    mean_age_resid=mean_age_obs-mean_age_pred;
    mean_age_sigma=sqrt(mean_age_pred2-elem_prod(mean_age_pred,mean_age_pred));
    mean_age_n=0.0;
    mean_age_mean=0.0;
    mean_age_m2=0.0;
    for (iyear=1;iyear<=nyears;iyear++){
      if (input_eff_samp_size_catch(ifleet,iyear)>0){
        mean_age_x=mean_age_resid(iyear)*sqrt(input_eff_samp_size_catch(ifleet,iyear))/mean_age_sigma(iyear);
        mean_age_n += 1.0;
        mean_age_delta=mean_age_x-mean_age_mean;
        mean_age_mean += mean_age_delta/mean_age_n;
        mean_age_m2 += mean_age_delta*(mean_age_x-mean_age_mean);
      }
    }
    if ((mean_age_n > 0) && (mean_age_m2 > 0)) Neff_stage2_mult_catch(ifleet)=1.0/(mean_age_m2/(mean_age_n-
1.0));
  }

  // Discards
  for (ifleet=1;ifleet<=nfleets;ifleet++){
    mean_age_obs=0.0;
    mean_age_pred=0.0;
    mean_age_pred2=0.0;
    mean_age_resid=0.0;
    for (iyear=1;iyear<=nyears;iyear++){
      for (iage=sel_start_age(ifleet);iage<=sel_end_age(ifleet);iage++){
        mean_age_obs(iyear) += Discard_prop_obs(ifleet,iyear,iage)*iage;
        mean_age_pred(iyear) += Discard_prop_pred(ifleet,iyear,iage)*iage;
        mean_age_pred2(iyear) += Discard_prop_pred(ifleet,iyear,iage)*iage*iage;
      }
    }
    mean_age_resid=mean_age_obs-mean_age_pred;
    mean_age_sigma=sqrt(mean_age_pred2-elem_prod(mean_age_pred,mean_age_pred));
    mean_age_n=0.0;
    mean_age_mean=0.0;
    mean_age_m2=0.0;
    for (iyear=1;iyear<=nyears;iyear++){
      if (input_eff_samp_size_discard(ifleet,iyear)>0){
        mean_age_x=mean_age_resid(iyear)*sqrt(input_eff_samp_size_discard(ifleet,iyear))/mean_age_sigma(iyear);
        mean_age_n += 1.0;
```

```
        mean_age_delta=mean_age_x-mean_age_mean;
        mean_age_mean += mean_age_delta/mean_age_n;
        mean_age_m2 += mean_age_delta*(mean_age_x-mean_age_mean);
      }
    }
    if ((mean_age_n > 0) && (mean_age_m2 > 0)) Neff_stage2_mult_discard(ifleet)=1.0/(mean_age_m2/(mean_age_n-
1.0));
  }
  // Indices
  for (ind=1;ind<=nindices;ind++){
    mean_age_obs=0.0;
    mean_age_pred=0.0;
    mean_age_pred2=0.0;
    mean_age_resid=0.0;
    for (i=1;i<=index_nobs(ind);i++){
      j=index_time(ind,i);
      for (iage=index_start_age(ind);iage<=index_end_age(ind);iage++){
        mean_age_obs(j) += index_prop_obs(ind,i,iage)*iage;
        mean_age_pred(j) += index_prop_pred(ind,i,iage)*iage;
        mean_age_pred2(j) += index_prop_pred(ind,i,iage)*iage*iage;
      }
    }
    mean_age_resid=mean_age_obs-mean_age_pred;
    mean_age_sigma=sqrt(mean_age_pred2-elem_prod(mean_age_pred,mean_age_pred));
    mean_age_n=0.0;
    mean_age_mean=0.0;
    mean_age_m2=0.0;
    for (iyear=1;iyear<=nyears;iyear++){
      if (index_Neff_init(ind,iyear)>0){
        mean_age_x=mean_age_resid(iyear)*sqrt(index_Neff_init(ind,iyear))/mean_age_sigma(iyear);
        mean_age_n += 1.0;
        mean_age_delta=mean_age_x-mean_age_mean;
        mean_age_mean += mean_age_delta/mean_age_n;
        mean_age_m2 += mean_age_delta*(mean_age_x-mean_age_mean);
      }
    }
    if ((mean_age_n > 0) && (mean_age_m2 > 0)) Neff_stage2_mult_index(ind)=1.0/(mean_age_m2/(mean_age_n-1.0));
  }



FUNCTION get_log_factorial
// compute sum of log factorial, used in multinomial likelihood constant
  nfact_out=0.0;
  if (nfact_in >= 2)
  {
     for (int ilogfact=2;ilogfact<=nfact_in;ilogfact++)
     {
        nfact_out+=log(ilogfact);
     }
  }

FUNCTION compute_the_objective_function
  obj_fun=0.0;
  io=0; // io if statements commented out to speed up program

// indices (lognormal)
  for (ind=1;ind<=nindices;ind++)
  {
     likely_ind(ind)=index_like_const(ind);
     RSS_ind(ind)=norm2(log(index_obs(ind))-log(index_pred(ind)));
     for (i=1;i<=index_nobs(ind);i++)
     {
         likely_ind(ind)+=log(index_sigma(ind,i));
         likely_ind(ind)+=0.5*square(log(index_obs(ind,i))-log(index_pred(ind,i)))/index_sigma2(ind,i);
         index_stdresid(ind,i)=(log(index_obs(ind,i))-log(index_pred(ind,i)))/index_sigma(ind,i);
     }
     obj_fun+=lambda_ind(ind)*likely_ind(ind);
  }
  // if (io==1) cout << "likely_ind " << likely_ind << endl;
```

```
// indices age comp (multinomial)
  likely_index_age_comp=index_prop_like_const;
  for (ind=1;ind<=nindices;ind++)
  {
     if (index_estimate_proportions(ind)==1)
     {
       for (i=1;i<=index_nobs(ind);i++)
       {
         temp_sum=0.0;
         for (iage=index_start_age(ind);iage<=index_end_age(ind);iage++)
         {
            temp_sum+=index_prop_obs(ind,i,iage)*log(index_prop_pred(ind,i,iage));
         }
         likely_index_age_comp+=-1.0*input_eff_samp_size_index(ind,i)*temp_sum;
       }
     }
  }
  obj_fun+=likely_index_age_comp;
  // if (io==1) cout << "likely_index_age_comp " << likely_index_age_comp << endl;

// total catch (lognormal)
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     catch_tot_likely(ifleet)=catch_tot_like_const(ifleet);
     discard_tot_likely(ifleet)=discard_tot_like_const(ifleet);
     RSS_catch_tot_fleet(ifleet)=norm2(log(Catch_tot_fleet_obs(ifleet)+0.00001)-
log(Catch_tot_fleet_pred(ifleet)+0.00001));
     RSS_Discard_tot_fleet(ifleet)=norm2(log(Discard_tot_fleet_obs(ifleet)+0.00001)-
log(Discard_tot_fleet_pred(ifleet)+0.00001));
     for (iyear=1;iyear<=nyears;iyear++)
     {
        catch_tot_likely(ifleet)+=log(catch_tot_sigma(ifleet,iyear));
        catch_tot_likely(ifleet)+=0.5*square(log(Catch_tot_fleet_obs(ifleet,iyear)+0.00001)-
log(Catch_tot_fleet_pred(ifleet,iyear)+0.00001))/catch_tot_sigma2(ifleet,iyear);
        discard_tot_likely(ifleet)+=log(discard_tot_sigma(ifleet,iyear));
        discard_tot_likely(ifleet)+=0.5*square(log(Discard_tot_fleet_obs(ifleet,iyear)+0.00001)-
log(Discard_tot_fleet_pred(ifleet,iyear)+0.00001))/discard_tot_sigma2(ifleet,iyear);
     }
     obj_fun+=lambda_catch_tot(ifleet)*catch_tot_likely(ifleet);
     obj_fun+=lambda_Discard_tot(ifleet)*discard_tot_likely(ifleet);
  }
  // if (io==1) cout << "catch_tot_likely " << catch_tot_likely << endl;

// catch age comp (multinomial)
  likely_catch=catch_prop_like_const;
  likely_Discard=discard_prop_like_const;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
    for (iyear=1;iyear<=nyears;iyear++)
    {
       temp_sum=0.0;
       temp_sum2=0.0;
       for (iage=sel_start_age(ifleet);iage<=sel_end_age(ifleet);iage++)
       {
          temp_sum+=CAA_prop_obs(ifleet,iyear,iage)*log(CAA_prop_pred(ifleet,iyear,iage));
          if(proportion_release(ifleet,iyear,iage)>0.0)
             temp_sum2+=Discard_prop_obs(ifleet,iyear,iage)*log(Discard_prop_pred(ifleet,iyear,iage));
       }
       likely_catch+=-1.0*input_eff_samp_size_catch(ifleet,iyear)*temp_sum;
       likely_Discard+=-1.0*input_eff_samp_size_discard(ifleet,iyear)*temp_sum2;
    }
  }
  obj_fun+=likely_catch;
  obj_fun+=likely_Discard;
  // if (io==1) cout << "likely_catch " << likely_catch << endl;

// stock-recruitment relationship (lognormal)
  likely_SR_sigma=SR_like_const;
  if (use_likelihood_constants==1)
  {
     likely_SR_sigma+=sum(log(SR_pred_recruits));
```

```
     likely_SR_sigma-=log(SR_pred_recruits(nyears+1));  // pred R in terminal year plus one does not have a
deviation
  }
  SR_stdresid=0.0;
  if (active(log_recruit_devs))
  {
     for (iyear=1;iyear<=nyears;iyear++)
     {
        likely_SR_sigma+=log(recruit_sigma(iyear));
        likely_SR_sigma+=0.5*square(log(recruits(iyear))-log(SR_pred_recruits(iyear)))/recruit_sigma2(iyear);
        SR_stdresid(iyear)=(log(recruits(iyear))-log(SR_pred_recruits(iyear)))/recruit_sigma(iyear);
     }
     obj_fun+=lambda_recruit_devs*likely_SR_sigma;
  }
  // if (io==1) cout << "likely_SR_sigma " << likely_SR_sigma << endl;

// selectivity parameters
  sel_likely=0.0;
  sel_stdresid=0.0;
  for (k=1;k<=nselparm;k++)
  {
     if (active(sel_params(k)))
     {
        sel_likely(k)+=sel_like_const(k);
        sel_likely(k)+=log(sel_sigma(k))+0.5*square(log(sel_initial(k))-log(sel_params(k)))/sel_sigma2(k);
        sel_stdresid(k)=(log(sel_initial(k))-log(sel_params(k)))/sel_sigma(k);
        obj_fun+=sel_lambda(k)*sel_likely(k);
     }
  }
  // if (io==1) cout << "sel_likely " << sel_likely << endl;

// index selectivity parameters
  indexsel_likely=0.0;
  indexsel_stdresid=0.0;
  for (k=1;k<=nindexselparms;k++)
  {
     if (active(index_sel_params(k)))
     {
        indexsel_likely(k)+=indexsel_like_const(k);
        indexsel_likely(k)+=log(indexsel_sigma(k))+0.5*square(log(indexsel_initial(k))-
log(index_sel_params(k)))/indexsel_sigma2(k);
        indexsel_stdresid(k)=(log(indexsel_initial(k))-log(index_sel_params(k)))/indexsel_sigma(k);
        obj_fun+=indexsel_lambda(k)*indexsel_likely(k);
     }
  }
  // if (io==1) cout << "indexsel_likely " << indexsel_likely << endl;

  steepness_likely=0.0;
  steepness_stdresid=0.0;
  if (active(SR_steepness))
  {
     steepness_likely=steepness_like_const;
     steepness_likely+=log(steepness_sigma)+0.5*square(log(SR_steepness_ini)-
log(SR_steepness))/steepness_sigma2;
     steepness_stdresid=(log(SR_steepness_ini)-log(SR_steepness))/steepness_sigma;
     obj_fun+=lambda_steepness*steepness_likely;
  }
  // if (io==1) cout << "steepness_likely " << steepness_likely << endl;

  SR_scaler_likely=0.0;
  SR_scaler_stdresid=0.0;
  if (active(log_SR_scaler))
  {
     SR_scaler_likely=SR_scaler_like_const;
     SR_scaler_likely+=log(SR_scaler_sigma)+0.5*(square(log(SR_scaler_ini)-log_SR_scaler))/SR_scaler_sigma2;
     SR_scaler_stdresid=(log(SR_scaler_ini)-log_SR_scaler)/SR_scaler_sigma;
     obj_fun+=lambda_SR_scaler*SR_scaler_likely;
  }
  // if (io==1) cout << "SR_scaler_likely " << SR_scaler_likely << endl;

  Fmult_year1_stdresid=0.0;
```

```
  if (active(log_Fmult_year1))
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
     {
        Fmult_year1_likely(ifleet)=Fmult_year1_like_const(ifleet);
        Fmult_year1_likely(ifleet)+=log(Fmult_year1_sigma(ifleet))+0.5*square(log_Fmult_year1(ifleet)-
log(Fmult_year1_ini(ifleet)))/Fmult_year1_sigma2(ifleet);
        Fmult_year1_stdresid(ifleet)=(log_Fmult_year1(ifleet)-
log(Fmult_year1_ini(ifleet)))/Fmult_year1_sigma(ifleet);
     }
     obj_fun+=lambda_Fmult_year1*Fmult_year1_likely;
  }
  // if (io==1) cout << "Fmult_year1_likely " << Fmult_year1_likely << endl;

  Fmult_devs_stdresid=0.0;
  if (active(log_Fmult_devs))
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
     {
        Fmult_devs_likely(ifleet)=Fmult_devs_like_const(ifleet);

Fmult_devs_likely(ifleet)+=log(Fmult_devs_sigma(ifleet))+0.5*norm2(log_Fmult_devs(ifleet))/Fmult_devs_sigma2(ifl
eet);
        for (iyear=2;iyear<=nyears;iyear++)
           Fmult_devs_stdresid(ifleet,iyear)=log_Fmult_devs(ifleet,iyear)/Fmult_devs_sigma(ifleet);
     }
     obj_fun+=lambda_Fmult_devs*Fmult_devs_likely;
  }
  // if (io==1) cout << "Fmult_devs_likely " << Fmult_devs_likely << endl;

  q_year1_stdresid=0.0;
  if (active(log_q_year1))
  {
     for (ind=1;ind<=nindices;ind++)
     {
        q_year1_likely(ind)=q_year1_like_const(ind);
        q_year1_likely(ind)+=log(q_year1_sigma(ind))+0.5*square(log_q_year1(ind)-
log(q_year1_ini(ind)))/q_year1_sigma2(ind);
        q_year1_stdresid(ind)=(log_q_year1(ind)-log(q_year1_ini(ind)))/q_year1_sigma(ind);
     }
     obj_fun+=lambda_q_year1*q_year1_likely;
  }
  // if (io==1) cout << "q_year1_likely " << q_year1_likely << endl;

  q_devs_stdresid=0.0;
  if (active(log_q_devs))
  {
     for (ind=1;ind<=nindices;ind++)
     {
       q_devs_likely(ind)=q_devs_like_const(ind);
       q_devs_likely(ind)+=log(q_devs_sigma(ind))+0.5*norm2(log_q_devs(ind))/q_devs_sigma2(ind);
       for (i=2;i<=index_nobs(ind);i++)
          q_devs_stdresid(ind,i)=log_q_devs(ind,i)/q_devs_sigma(ind);
     }
     obj_fun+=lambda_q_devs*q_devs_likely;
  }
  // if (io==1) cout << "q_devs_likely " << q_devs_likely << endl;

  if (NAA_year1_flag==1)
  {
     nyear1temp(1)=SR_pred_recruits(1);
     N_year1_stdresid=0.0;
     for (iage=2;iage<=nages;iage++)
     {
        nyear1temp(iage)=nyear1temp(iage-1)*S(1,iage-1);
     }
     nyear1temp(nages)/=(1.0-S(1,nages));
  }
  else if (NAA_year1_flag==2)
  {
     nyear1temp=NAA_year1_ini;
```

```
  }
  if (active(log_N_year1_devs))
  {
     if (N_year1_sigma>0.0)
     {
        for (iage=2;iage<=nages;iage++)
           N_year1_stdresid(iage)=(log(NAA(1,iage))-log(nyear1temp(iage)))/N_year1_sigma;
     }
     N_year1_likely=N_year1_like_const+sum(log(nyear1temp));
     N_year1_likely+=log(N_year1_sigma)+0.5*norm2(log(NAA(1))-log(nyear1temp))/N_year1_sigma2;
     obj_fun+=lambda_N_year1_devs*N_year1_likely;
  }
  // if (io==1) cout << "N_year1_likely " << N_year1_likely << endl;

  Fmult_max_pen=0.0;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
    for (iyear=1;iyear<=nyears;iyear++)
    {
       temp_Fmult_max=mfexp(log_Fmult(ifleet,iyear))*max(sel_by_fleet(ifleet,iyear));
       if(temp_Fmult_max>Fmult_max_value)
          Fmult_max_pen+=1000.*(temp_Fmult_max-Fmult_max_value)*(temp_Fmult_max-Fmult_max_value);
    }
  }
  obj_fun+=Fmult_max_pen;
  // if (io==1) cout << "Fmult_max_pen " << Fmult_max_pen << endl;

  fpenalty_lambda=100.0*pow(10.0,(-1.0*current_phase())); // decrease emphasis on F near M as phases increase
  if (last_phase())                                       // no penalty in final solution
     fpenalty_lambda=0.0;
  fpenalty=fpenalty_lambda*square(log(mean(FAA_tot))-log(mean(M)));
  obj_fun+=fpenalty;
  // if (io==1) cout << "fpenalty " << fpenalty << endl;

FUNCTION write_MCMC
// first the output file for AgePro
  if (MCMCnyear_opt == 0)    // use final year
  {
     if (fillR_opt == 0)
     {
        NAAbsn(1)=NAA(nyears,1);
     }
     else if (fillR_opt == 1)
     {
        NAAbsn(1)=SR_pred_recruits(nyears);
     }
     else if (fillR_opt == 2)
     {
        tempR=0.0;
        for (i=Ravg_start;i<=Ravg_end;i++)
        {
           iyear=i-year1+1;
           tempR+=log(NAA(iyear,1));
        }
        NAAbsn(1)=mfexp(tempR/(Ravg_end-Ravg_start+1.0));
     }
     for (iage=2;iage<=nages;iage++)
     {
        NAAbsn(iage)=NAA(nyears,iage);
     }
  }
  else                       // use final year + 1
  {
     if (fillR_opt == 1)
     {
        NAAbsn(1)=SR_pred_recruits(nyears+1);
     }
     else if (fillR_opt == 2)
     {
        tempR=0.0;
        for (i=Ravg_start;i<=Ravg_end;i++)
```

```
        {
           iyear=i-year1+1;
           tempR+=log(NAA(iyear,1));
        }
        NAAbsn(1)=mfexp(tempR/(Ravg_end-Ravg_start+1.0));
     }
     for (iage=2;iage<=nages;iage++)
     {
        NAAbsn(iage)=NAA(nyears,iage-1)*S(nyears,iage-1);
     }
     NAAbsn(nages)+=NAA(nyears,nages)*S(nyears,nages);
  }

  // Liz added
       for (iyear=1;iyear<=nyears;iyear++)
     {
                tempFmult(iyear) = max(extract_row(FAA_tot,iyear));
                }
  // end stuff Liz added


// output the NAAbsn values
  ageproMCMC << NAAbsn << endl;

// now the standard MCMC output file
  basicMCMC << Freport << " " <<
               SSB << " " <<

                ///  Liz added

               tempFmult << " " <<

               rowsum(elem_prod(WAAjan1b, NAA)) << " " <<

               /// end stuff Liz added

               MSY << " " <<
               SSmsy << " " <<
               Fmsy << " " <<
               SSBmsy_ratio << " " <<
               Fmsy_ratio << " " <<
               endl;

REPORT_SECTION
  report << "Age Structured Assessment Program (ASAP) Version 3.0" << endl;
  report << "Start time for run: " << ctime(&start) << endl;
  report << "obj_fun        = " << obj_fun << endl << endl;
  report << "Component                Lambda          obj_fun" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
     report << "__Catch_Fleet_" << ifleet << "               " << lambda_catch_tot(ifleet) << "          " <<
lambda_catch_tot(ifleet)*catch_tot_likely(ifleet) << endl;
  report << "Catch_Fleet_Total             " << sum(lambda_catch_tot) << "          " <<
lambda_catch_tot*catch_tot_likely << endl;
  if (lambda_Discard_tot*discard_tot_likely > 0.0)
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
        report << "__Discard_Fleet_" << ifleet << "             " << lambda_Discard_tot(ifleet) << "          "
<< lambda_Discard_tot(ifleet)*discard_tot_likely(ifleet) << endl;
  }
  report << "Discard_Fleet_Total           " << sum(lambda_Discard_tot) << "          " <<
lambda_Discard_tot*discard_tot_likely << endl;
  for (ind=1;ind<=nindices;ind++)
     report << "__Index_Fit_" << ind << "                " << lambda_ind(ind) << "          " <<
lambda_ind(ind)*likely_ind(ind) << endl;
  report << "Index_Fit_Total               " << sum(lambda_ind) << "          " << lambda_ind*likely_ind <<
endl;
  report << "Catch_Age_Comps       see_below          " << likely_catch << endl;
  report << "Discard_Age_Comps     see_below          " << likely_Discard << endl;
  report << "Index_Age_Comps       see_below          " << likely_index_age_comp << endl;
  sum_sel_lambda=0;
  sum_sel_lambda_likely=0.0;
```

```
  for (k=1;k<=nselparm;k++)
  {
     if (sel_phase(k) >= 1)
     {
        if      (k < 10  ) report << "__Sel_Param_" << k << "                 " << sel_lambda(k) << "          "
<< sel_lambda(k)*sel_likely(k) << endl;
        else if (k < 100 ) report << "__Sel_Param_" << k << "                " << sel_lambda(k) << "          "
<< sel_lambda(k)*sel_likely(k) << endl;
        else if (k < 1000) report << "__Sel_Param_" << k << "               " << sel_lambda(k) << "          "
<< sel_lambda(k)*sel_likely(k) << endl;
        sum_sel_lambda+=sel_lambda(k);
        sum_sel_lambda_likely+=sel_lambda(k)*sel_likely(k);
     }
  }
  report << "Sel_Params_Total             " << sum_sel_lambda << "          " << sum_sel_lambda_likely << endl;
  sum_indexsel_lambda=0;
  sum_indexsel_lambda_likely=0.0;
  for (k=1;k<=nindexselparms;k++)
  {
     if (indexsel_phase(k) >= 1)
     {
        if      (k <10  ) report << "__Index_Sel_Param_" << k << "           " << indexsel_lambda(k) << "
" << indexsel_lambda(k)*indexsel_likely(k) << endl;
        else if (k <100 ) report << "__Index_Sel_Param_" << k << "          " << indexsel_lambda(k) << "
" << indexsel_lambda(k)*indexsel_likely(k) << endl;
        else if (k <1000) report << "__Index_Sel_Param_" << k << "         " << indexsel_lambda(k) << "
" << indexsel_lambda(k)*indexsel_likely(k) << endl;
        sum_indexsel_lambda+=indexsel_lambda(k);
        sum_indexsel_lambda_likely+=indexsel_lambda(k)*indexsel_likely(k);
     }
  }
  report << "Index_Sel_Params_Total       " << sum_indexsel_lambda << "          " <<
sum_indexsel_lambda_likely << endl;
  if (lambda_q_year1*q_year1_likely > 0.0)
  {
     for (ind=1;ind<=nindices;ind++)
        report << "__q_year1_index_" << ind << "             "  << lambda_q_year1(ind) << "          " <<
lambda_q_year1(ind)*q_year1_likely(ind) << endl;
  }
  report << "q_year1_Total                " << sum(lambda_q_year1) << "          " <<
lambda_q_year1*q_year1_likely << endl;

  if (lambda_q_devs*q_devs_likely > 0.0)
  {
     for (ind=1;ind<=nindices;ind++)
        report << "__q_devs_index_" << ind << "              "  << lambda_q_devs(ind) << "          " <<
lambda_q_devs(ind)*q_devs_likely(ind) << endl;
  }
  report << "q_devs_Total                 " << sum(lambda_q_devs) << "          " <<
lambda_q_devs*q_devs_likely << endl;
  if (lambda_Fmult_year1*Fmult_year1_likely > 0.0);
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
        report << "__Fmult_year1_fleet_" << ifleet << "         " << lambda_Fmult_year1(ifleet) << "          "
<< lambda_Fmult_year1(ifleet)*Fmult_year1_likely(ifleet) << endl;
  }
  report << "Fmult_year1_fleet_Total      " << sum(lambda_Fmult_year1) << "          " <<
lambda_Fmult_year1*Fmult_year1_likely << endl;
  if (lambda_Fmult_devs*Fmult_devs_likely > 0.0)
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
        report << "__Fmult_devs_fleet_" << ifleet << "          " << lambda_Fmult_devs(ifleet) << "          "
<< lambda_Fmult_devs(ifleet)*Fmult_devs_likely(ifleet) << endl;
  }
  report << "Fmult_devs_fleet_Total       " << sum(lambda_Fmult_devs) << "          " <<
lambda_Fmult_devs*Fmult_devs_likely << endl;
  report << "N_year_1                     " << lambda_N_year1_devs << "          " <<
lambda_N_year1_devs*N_year1_likely << endl;
  report << "Recruit_devs                 " << lambda_recruit_devs << "          " <<
lambda_recruit_devs*likely_SR_sigma << endl;
```

```
  report << "SR_steepness                  " << lambda_steepness << "          " <<
lambda_steepness*steepness_likely << endl;
  report << "SR_scaler                     " << lambda_SR_scaler << "          " <<
lambda_SR_scaler*SR_scaler_likely << endl;
  report << "Fmult_Max_penalty          1000          " << Fmult_max_pen << endl;
  report << "F_penalty                     " << fpenalty_lambda << "          " << fpenalty << endl;
  report << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     for (iyear=1;iyear<=nyears;iyear++)
     {
       if (input_eff_samp_size_catch(ifleet,iyear)==0)
       {
         effective_sample_size(ifleet,iyear)=0;
       }
       else
       {
         effective_sample_size(ifleet,iyear)=CAA_prop_pred(ifleet,iyear)*(1.0-
CAA_prop_pred(ifleet,iyear))/norm2(CAA_prop_obs(ifleet,iyear)-CAA_prop_pred(ifleet,iyear));
       }
       if (input_eff_samp_size_discard(ifleet,iyear)==0)
       {
         effective_Discard_sample_size(ifleet,iyear)=0;
       }
       else
       {
         effective_Discard_sample_size(ifleet,iyear)=Discard_prop_pred(ifleet,iyear)*(1.0-
Discard_prop_pred(ifleet,iyear))/norm2(Discard_prop_obs(ifleet,iyear)-Discard_prop_pred(ifleet,iyear));
       }
     }
  }
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     report << " Input and Estimated effective sample sizes for fleet " << ifleet << endl;
     for (iyear=1;iyear<=nyears;iyear++)
        report << iyear+year1-1 << "  " << input_eff_samp_size_catch(ifleet,iyear) << "  " <<
effective_sample_size(ifleet,iyear) << endl;
     report << " Total  " << sum(input_eff_samp_size_catch(ifleet)) << "  " <<
sum(effective_sample_size(ifleet)) << endl;
  }
  report << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     report << " Input and Estimated effective Discard sample sizes for fleet " << ifleet << endl;
     for (iyear=1;iyear<=nyears;iyear++)
        report << iyear+year1-1 << "  " << input_eff_samp_size_discard(ifleet,iyear) << "  " <<
effective_Discard_sample_size(ifleet,iyear) << endl;
     report << " Total  " << sum(input_eff_samp_size_discard(ifleet)) << "  " <<
sum(effective_Discard_sample_size(ifleet)) << endl;
  }
  report << endl;
  report << "Observed and predicted total fleet catch by year and standardized residual" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
    report << " fleet " << ifleet << " total catches" << endl;
    for (iyear=1;iyear<=nyears;iyear++)
    {
      Catch_stdresid(ifleet,iyear)=(log(Catch_tot_fleet_obs(ifleet,iyear)+0.00001)-
log(Catch_tot_fleet_pred(ifleet,iyear)+0.00001))/catch_tot_sigma(ifleet,iyear);
      report << iyear+year1-1 << "  " << Catch_tot_fleet_obs(ifleet,iyear) << "  " <<
Catch_tot_fleet_pred(ifleet,iyear) << "  " << Catch_stdresid(ifleet,iyear) << endl;
    }
  }
  report << "Observed and predicted total fleet Discards by year and standardized residual" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
    report << " fleet " << ifleet << " total Discards" << endl;
    for (iyear=1;iyear<=nyears;iyear++)
    {
      Discard_stdresid(ifleet,iyear)=(log(Discard_tot_fleet_obs(ifleet,iyear)+0.00001)-
log(Discard_tot_fleet_pred(ifleet,iyear)+0.00001))/discard_tot_sigma(ifleet,iyear);
```

```
      report << iyear+year1-1 << "  " << Discard_tot_fleet_obs(ifleet,iyear) << "  " <<
Discard_tot_fleet_pred(ifleet,iyear) << "  " << Discard_stdresid(ifleet,iyear) << endl;
    }
  }
  report << endl << "Index data" << endl;
  for (ind=1;ind<=nindices;ind++) {
     report << "index number " << ind << endl;
     report << "aggregate units = " << index_units_aggregate(ind) << endl;
     report << "proportions units = " << index_units_proportions(ind) << endl;
     report << "month = " << index_month(ind) << endl;
     report << "starting and ending ages for selectivity = " << index_start_age(ind) << "  " <<
index_end_age(ind) << endl;
     report << "selectivity choice = " << index_sel_choice(ind) << endl;
     report << " year, obs index, pred index, standardized residual" << endl;
     for (j=1;j<=index_nobs(ind);j++)
         report << index_year(ind,j) << "  " << index_obs(ind,j) << "  " << index_pred(ind,j) << "  " <<
index_stdresid(ind,j) << endl;
  }
  report << endl;
  index_Neff_init=0.0;
  index_Neff_est=0.0;
  for (ind=1;ind<=nindices;ind++)
  {
     for (iyear=1;iyear<=nyears;iyear++)
     {
         for (i=1;i<=index_nobs(ind);i++)
         {
             if (index_time(ind,i)==iyear)
             {
                 index_Neff_init(ind,iyear)=input_eff_samp_size_index(ind,i);
                 if (input_eff_samp_size_index(ind,i)==0)
                 {
                     index_Neff_est(ind,iyear)=0.0;
                 }
                 else
                 {
                     index_Neff_est(ind,iyear)=index_prop_pred(ind,i)*(1.0-
index_prop_pred(ind,i))/norm2(index_prop_obs(ind,i)-index_prop_pred(ind,i));
                 }
             }
         }
     }
  }
  report << "Input effective sample sizes by index (row=index, column=year)" << endl;
  report << index_Neff_init << endl;
  report << "Estimated effective sample sizes by index (row=index, column=year)" << endl;
  report << index_Neff_est << endl;
  report << endl;
  report << "Index proportions at age by index" << endl;
  for (ind=1;ind<=nindices;ind++)
  {
     output_index_prop_obs(ind)=0.0;
     output_index_prop_pred(ind)=0.0;
     if (index_estimate_proportions(ind)==1)
     {
        report << " Index number " << ind << endl;
        for (iyear=1;iyear<=nyears;iyear++)
        {
           for (i=1;i<=index_nobs(ind);i++)
           {
               if (index_time(ind,i)==iyear)
               {
                   for (iage=index_start_age(ind);iage<=index_end_age(ind);iage++)
                   {
                       output_index_prop_obs(ind,iyear,iage)=index_prop_obs(ind,i,iage);
                       output_index_prop_pred(ind,iyear,iage)=index_prop_pred(ind,i,iage);
                   }
               }
           }
           report << "Year " << iyear+year1-1 << " Obs  = " << output_index_prop_obs(ind,iyear) << endl;
           report << "Year " << iyear+year1-1 << " Pred = " << output_index_prop_pred(ind,iyear) << endl;
```

```
      }
    }
  }
  report << endl;
  report << "Index Selectivity at Age" << endl;
  report << indexsel << endl;
  report << endl;

  report << "Deviations section: only applicable if associated lambda > 0" << endl;
  report << "Nyear1 observed, expected, standardized residual" << endl;
  if (lambda_N_year1_devs > 0.0)
  {
     for (iage=2;iage<=nages;iage++)
     {
        report << iage << "  " << NAA(1,iage) << "  " << nyear1temp(iage) << "  " << N_year1_stdresid(iage) <<
endl;
     }
  }
  else
  {
     report << "N/A" << endl;
  }
  report << endl;
  report << "Fleet Obs, Initial, and Stadardized Residual for Fmult" << endl;
  if (sum(lambda_Fmult_year1) > 0.0)
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
        report << ifleet << "  " << mfexp(log_Fmult_year1(ifleet)) << "  " << Fmult_year1_ini(ifleet) << "  " <<
Fmult_year1_stdresid(ifleet) << endl;
  }
  else
  {
     report << "N/A" << endl;
  }
  report << endl;
  report << "Standardized Residuals for Fmult_devs by fleet and year" << endl;
  if (sum(lambda_Fmult_devs) > 0.0)
  {
     for (ifleet=1;ifleet<=nfleets;ifleet++)
     {
        report << " fleet " << ifleet << " Fmult_devs standardized residuals" << endl;
        for (iyear=2;iyear<=nyears;iyear++)
           report << iyear << "  " << Fmult_devs_stdresid(ifleet,iyear) << endl;
     }
  }
  else
  {
     report << "N/A" << endl;
  }
  report << endl;
  report << "Index Obs, Initial, and Standardized Residual for q_year1" << endl;
  if (sum(lambda_q_year1) > 0.0)
  {
     for (ind=1;ind<=nindices;ind++)
        report << ind << "  " << mfexp(log_q_year1(ind)) << "  " << q_year1_ini(ind) << "  " <<
              (log_q_year1(ind)-log(q_year1_ini(ind)))/q_year1_sigma(ind) << endl;
  }
  else
  {
     report << "N/A" << endl;
  }
  report << endl;
  report << "Standardized Residuals for catchability deviations by index and year" << endl;
  if (sum(lambda_q_devs) > 0.0)
  {
     for (ind=1;ind<=nindices;ind++)
     {
        report << " index " << ind << " q_devs standardized residuals" << endl;
        for (i=2;i<=index_nobs(ind);i++)
           report << index_year(ind,i) << "  " << log_q_devs(ind,i)/q_devs_sigma(ind) << endl;
     }
```

```
  }
  else
  {
     report << "N/A" << endl;
  }
  report << endl;
  report << "Obs, Initial, and Stadardized Residual for SR steepness" << endl;
  if (lambda_steepness > 0.0)
  {
     report << SR_steepness << "  " << SR_steepness_ini << "  " << (log(SR_steepness)-
log(SR_steepness_ini))/steepness_sigma << endl;
  }
  else
  {
     report << "N/A" << endl;
  }
  report << endl;
  report << "Obs, Initial, and Stadardized Residual for SR scaler" << endl;
  if (lambda_SR_scaler > 0.0)
  {
     report << mfexp(log_SR_scaler) << "  " << SR_scaler_ini << "  " << (log_SR_scaler-
log(SR_scaler_ini))/SR_scaler_sigma << endl;
  }
  else
  {
     report << "N/A" << endl;
  }
  report << endl;
  report << "End of Deviations Section" << endl << endl;

  report << "Selectivity by age and year for each fleet" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++) {
     report << " fleet " << ifleet << " selectivity at age" << endl;
     for (iyear=1;iyear<=nyears;iyear++)
       report << sel_by_fleet(ifleet,iyear) << endl;
  }
  report << endl;
  report << "Fmult by year for each fleet" << endl;
  Fmult=mfexp(log_Fmult);
  for (iyear=1;iyear<=nyears;iyear++) {
     for (ifleet=1;ifleet<=nfleets;ifleet++){
         temp_Fmult(ifleet)=Fmult(ifleet,iyear);
     }
     report << iyear+year1-1 << "  " << temp_Fmult << endl;
  }
  report << endl;
  report << "Directed F by age and year for each fleet" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     report << " fleet " << ifleet << " directed F at age" << endl;
     for (iyear=1;iyear<=nyears;iyear++)
         report << FAA_by_fleet_dir(ifleet,iyear) << endl;
  }
  report << "Discard F by age and year for each fleet" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     report << " fleet " << ifleet << " Discard F at age" << endl;
     for (iyear=1;iyear<=nyears;iyear++)
         report << FAA_by_fleet_Discard(ifleet,iyear) << endl;
  }
  report << "Total F" << endl;
  for (iyear=1;iyear<=nyears;iyear++)
     report << FAA_tot(iyear) << endl;
  report << endl;
  report << "Average F for ages " << Freport_agemin << " to " << Freport_agemax << endl;
  if (Freport_wtopt==1) report << "Freport unweighted in .std and MCMC files" << endl;
  if (Freport_wtopt==2) report << "Freport N weighted in .std and MCMC files" << endl;
  if (Freport_wtopt==3) report << "Freport B weighted in .std and MCMC files" << endl;
  report << "year    unweighted   Nweighted    Bweighted" << endl;
  for (iyear=1;iyear<=nyears;iyear++){
```

```
     report << iyear+year1-1 << "  " << Freport_U(iyear) << "  " << Freport_N(iyear) << "  " << Freport_B(iyear)
<< endl;
  }
  report << endl;
  report << "Population Numbers at the Start of the Year" << endl;
  for (iyear=1;iyear<=nyears;iyear++)
     report << NAA(iyear) << endl;
  report << endl;
  report << "Biomass Time Series" << endl;
  report << "Year, TotJan1B, SSB, ExploitableB" << endl;
  for (iyear=1;iyear<=nyears;iyear++)
  {
     report << iyear+year1-1 << "  " << TotJan1B(iyear) << "  " << SSB(iyear) << "  " << ExploitableB(iyear) <<
endl;
  }
  report << endl;
  report << "q by index" << endl;
  for (ind=1;ind<=nindices;ind++)
  {
     report << " index " << ind << " q over time" << endl;
     for (i=1;i<=index_nobs(ind);i++)
     {
         report << index_year(ind,i) << "  " << q_by_index(ind,i) << endl;
     }
  }
  report << endl;
  report << "Proportions of catch at age by fleet" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
    report << " fleet " << ifleet << endl;
    for (iyear=1;iyear<=nyears;iyear++)
    {
       output_prop_obs=0.0;
       output_prop_pred=0.0;
       output_prop_obs(sel_start_age(ifleet),sel_end_age(ifleet))=CAA_prop_obs(ifleet,iyear);
       output_prop_pred(sel_start_age(ifleet),sel_end_age(ifleet))=CAA_prop_pred(ifleet,iyear);
       report << "Year " << iyear << " Obs  = " << output_prop_obs << endl;
       report << "Year " << iyear << " Pred = " << output_prop_pred << endl;
    }
  }
  report << endl;
  report << "Proportions of Discards at age by fleet" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
    report << " fleet " << ifleet << endl;
    for (iyear=1;iyear<=nyears;iyear++)
    {
       output_Discard_prop_obs=0.0;
       output_Discard_prop_pred=0.0;
       output_Discard_prop_obs(sel_start_age(ifleet),sel_end_age(ifleet))=Discard_prop_obs(ifleet,iyear);
       output_Discard_prop_pred(sel_start_age(ifleet),sel_end_age(ifleet))=Discard_prop_pred(ifleet,iyear);
       report << "Year " << iyear << " Obs  = " << output_Discard_prop_obs << endl;
       report << "Year " << iyear << " Pred = " << output_Discard_prop_pred << endl;
    }
  }
  report << endl;
  report << "F Reference Points Using Final Year Selectivity and Freport options" << endl;
  report << " refpt           F       slope to plot on SR" << endl;
  report << "  F0.1     " << F01_report << "     " << F01_slope << endl;
  report << "  Fmax     " << Fmax_report << "     " << Fmax_slope << endl;
  report << "  F30%SPR  " << F30SPR_report << "     " << F30SPR_slope << endl;
  report << "  F40%SPR  " << F40SPR_report << "     " << F40SPR_slope << endl;
  report << "  Fmsy     " << Fmsy_report << "     " << Fmsy_slope << "    SSBmsy    " << SSBmsy_report << "
MSY   " << MSY << endl;
  report << "  Fcurrent " << Freport(nyears) << "     " << Fcurrent_slope << endl;
  report << endl;
  report << "Stock-Recruitment Relationship Parameters" << endl;
  report << " alpha     = " << SR_alpha << endl;
  report << " beta      = " << SR_beta << endl;
  report << " R0        = " << SR_R0 << endl;
  report << " S0        = " << SR_S0 << endl;
```

```
  report << " steepness = " << SR_steepness << endl;
  report << "Spawning Stock, Obs Recruits(year+1), Pred Recruits(year+1), standardized residual" << endl;
  report << "init  xxxx  " << recruits(1) << "  " << SR_pred_recruits(1) << "  " <<
            (log(recruits(1))-log(SR_pred_recruits(1)))/recruit_sigma(1) << endl;
  for (iyear=1;iyear<nyears;iyear++)
    report << iyear+year1-1 << "  " << SSB(iyear) << "  " << recruits(iyear+1) << "  "  <<
SR_pred_recruits(iyear+1) << "  " <<
             (log(recruits(iyear+1))-log(SR_pred_recruits(iyear+1)))/recruit_sigma(iyear+1) << endl;
  report << nyears+year1-1 << "  " << SSB(nyears) << "       xxxx   " << SR_pred_recruits(nyears+1) << endl;
  report << endl;

  report << "Annual stock recruitment parameters" << endl;
  report << "Year, S0_vec, R0_vec, steepness_vec, s_per_r_vec" << endl;
  for (iyear=1;iyear<=nyears;iyear++)
    report << iyear+year1-1 << "  " << S0_vec(iyear) << "  " << R0_vec(iyear) << "  " << steepness_vec(iyear) <<
"  " << s_per_r_vec(iyear) << endl;
  report << endl;

  report << "Root Mean Square Error computed from Standardized Residuals" << endl;
  report << "Component                 #resids         RMSE" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     report << "_Catch_Fleet_" << ifleet << "               " << nyears << "           " <<
sqrt(mean(square(Catch_stdresid(ifleet)))) << endl;
  }
  report << "Catch_Fleet_Total            " << nyears*nfleets << "           " <<
sqrt(mean(square(Catch_stdresid))) << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     if (norm2(Discard_stdresid(ifleet)) > 0.0 )
     {
        report << "_Discard_Fleet_" << ifleet << "             " << nyears << "           " <<
sqrt(mean(square(Discard_stdresid(ifleet)))) << endl;
     }
     else
     {
        report << "_Discard_Fleet_" << ifleet << "             " << "0" << "            " << "0" << endl;
     }
  }
  if (norm2(Discard_stdresid) > 0.0)
  {
     report << "Discard_Fleet_Total          " << nyears*nfleets << "           " <<
sqrt(mean(square(Discard_stdresid))) << endl;
  }
  else
  {
     report << "Discard_Fleet_Total          " << "0" << "            " << "0" << endl;
  }
  for (ind=1;ind<=nindices;ind++)
  {
     report << "_Index_" << ind << "                     " << index_nobs(ind) << "           " <<
sqrt(mean(square(index_stdresid(ind)))) << endl;
  }
  report << "Index_Total                  " << sum(index_nobs) << "           " <<
sqrt(mean(square(index_stdresid))) << endl;
  N_year1_rmse=0.0;
  N_year1_rmse_nobs=0;
  if (lambda_N_year1_devs > 0.0 && norm2(N_year1_stdresid) > 0.0)
  {
     N_year1_rmse=sqrt(mean(square(N_year1_stdresid)));
     N_year1_rmse_nobs=nages-1;
  }
  report << "Nyear1                       " << N_year1_rmse_nobs << "           " << N_year1_rmse << endl;
  Fmult_year1_rmse=0.0;
  Fmult_year1_rmse_nobs=0;
  if (sum(lambda_Fmult_year1) > 0.0 && norm2(Fmult_year1_stdresid) > 0.0)
  {
     Fmult_year1_rmse=sqrt(mean(square(Fmult_year1_stdresid)));
     Fmult_year1_rmse_nobs=nfleets;
  }
```

```
  report << "Fmult_Year1                  " << Fmult_year1_rmse_nobs << "           " << Fmult_year1_rmse <<
endl;
  Fmult_devs_fleet_rmse=0.0;
  Fmult_devs_fleet_rmse_nobs=0;
  Fmult_devs_rmse=0.0;
  Fmult_devs_rmse_nobs=0;
  for (ifleet=1;ifleet<=nfleets;ifleet++)
  {
     if (sum(lambda_Fmult_devs) > 0.0 && norm2(Fmult_devs_stdresid(ifleet)) > 0.0)
     {
        Fmult_devs_fleet_rmse(ifleet)=sqrt(mean(square(Fmult_devs_stdresid(ifleet))));
        Fmult_devs_fleet_rmse_nobs(ifleet)=nyears-1;
     }
     report << "_Fmult_devs_Fleet_" << ifleet << "          " << Fmult_devs_fleet_rmse_nobs(ifleet) << "
" << Fmult_devs_fleet_rmse(ifleet) << endl;
  }
  if (sum(lambda_Fmult_devs) > 0.0 && norm2(Fmult_devs_stdresid) > 0.0)
  {
     Fmult_devs_rmse=sqrt(mean(square(Fmult_devs_stdresid)));
     Fmult_devs_rmse_nobs=nfleets*(nyears-1);
  }
  report << "Fmult_devs_Total             " << Fmult_devs_rmse_nobs << "           " << Fmult_devs_rmse << endl;
  SR_rmse=0.0;
  SR_rmse_nobs=0;
  if (lambda_recruit_devs > 0.0 && norm2(SR_stdresid) > 0.0)
  {
     SR_rmse=sqrt(mean(square(SR_stdresid)));
     SR_rmse_nobs=nyears;
  }
  report << "Recruit_devs                 " << SR_rmse_nobs << "           " << SR_rmse << endl;
  sel_rmse=0.0;
  sel_rmse_nobs=0;
  if (sum(sel_lambda) > 0.0 && norm2(sel_stdresid) > 0.0)
  {
     sel_rmse=sqrt(mean(square(sel_stdresid)));
     for (k=1;k<=nselparm;k++)
     {
        if (sel_lambda(k) > 0.0)
           sel_rmse_nobs+=1;
     }
  }
  report << "Fleet_Sel_params             " << sel_rmse_nobs << "           " << sel_rmse << endl;
  indexsel_rmse=0.0;
  indexsel_rmse_nobs=0;
  if (sum(indexsel_lambda) > 0.0 && norm2(indexsel_stdresid) > 0.0)
  {
     indexsel_rmse=sqrt(mean(square(indexsel_stdresid)));
     for (k=1;k<=nindexselparms;k++)
     {
        if (indexsel_lambda(k) > 0.0)
           indexsel_rmse_nobs+=1;
     }
  }
  report << "Index_Sel_params             " << indexsel_rmse_nobs << "           " << indexsel_rmse << endl;
  q_year1_rmse=0.0;
  q_year1_rmse_nobs=0;
  if (sum(lambda_q_year1) > 0.0 && norm2(q_year1_stdresid) > 0.0)
  {
     q_year1_rmse=sqrt(mean(square(q_year1_stdresid)));
     for (ind=1;ind<=nindices;ind++)
     {
        if (lambda_q_year1(ind) > 0.0)
           q_year1_rmse_nobs+=1;
     }
  }
  report << "q_year1                      " << q_year1_rmse_nobs << "           " << q_year1_rmse << endl;
  q_devs_rmse=0.0;
  q_devs_rmse_nobs=0;
  if (sum(lambda_q_devs) > 0.0 && norm2(q_devs_stdresid) > 0.0)
  {
     q_devs_rmse=sqrt(mean(square(q_devs_stdresid)));
```

```
     for (ind=1;ind<=nindices;ind++)
     {
        if (lambda_q_year1(ind) > 0.0)
           q_devs_rmse_nobs+=index_nobs(ind)-1;
     }
  }
  report << "q_devs                        " << q_devs_rmse_nobs << "           " << q_devs_rmse << endl;
  steepness_rmse=0.0;
  steepness_rmse_nobs=0;
  if (lambda_steepness > 0.0)
  {
     steepness_rmse=sfabs(steepness_stdresid);
     steepness_rmse_nobs=1;
  }
  report << "SR_steepness                 " << steepness_rmse_nobs << "           " << steepness_rmse << endl;
  SR_scaler_rmse=0.0;
  SR_scaler_rmse_nobs=0;
  if (lambda_SR_scaler > 0.0)
  {
     SR_scaler_rmse=sfabs(SR_scaler_stdresid);
     SR_scaler_rmse_nobs=1;
  }
  report << "SR_scaler                    " << SR_scaler_rmse_nobs << "           " << SR_scaler_rmse << endl;
  report << endl;

  report << "Stage2 Multipliers for Multinomials (Francis 2011)" << endl;
  report << "Catch by Fleet" << endl;
  report << Neff_stage2_mult_catch << endl;
  report << "Discards by Fleet" << endl;
  report << Neff_stage2_mult_discard << endl;
  report << "Indices" << endl;
  report << Neff_stage2_mult_index << endl;
  report << endl;
  report << "New Input ESS based on applying stage2 multipliers" << endl;
  report << "Catch (rows are fleets, columns are years)" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++){
    report << input_eff_samp_size_catch(ifleet) * Neff_stage2_mult_catch(ifleet) << endl;
  }
  report << "Discards (rows are fleets, columns are years)" << endl;
  for (ifleet=1;ifleet<=nfleets;ifleet++){
    report << input_eff_samp_size_discard(ifleet) * Neff_stage2_mult_discard(ifleet) << endl;
  }
  report << "Indices (rows are indices, columns are years)" << endl;
  for (ind=1;ind<=nindices;ind++){
    report << index_Neff_init(ind) * Neff_stage2_mult_index(ind) << endl;
  }
  report << endl;

  if (do_projections==1 && last_phase())
  {
     project_into_future();
     report << "Projection into Future" << endl;
     report << "Projected NAA" << endl;
     report << proj_NAA << endl;
     report << "Projected Directed FAA" << endl;
     report << proj_F_dir << endl;
     report << "Projected Discard FAA" << endl;
     report << proj_F_Discard << endl;
     report << "Projected Nondirected FAA" << endl;
     report << proj_F_nondir << endl;
     report << "Projected Catch at Age" << endl;
     report << proj_catch << endl;
     report << "Projected Discards at Age (in numbers)" << endl;
     report << proj_Discard << endl;
     report << "Projected Yield at Age" << endl;
     report << proj_yield << endl;
     report << "Year, Total Yield (in weight), Total Discards (in weight), TotJan1B, SSB, proj_what, SS/SSmsy"
<< endl;
     for (iyear=1;iyear<=nprojyears;iyear++)
```

```
       report << year1+nyears-1+iyear << "  " << proj_total_yield(iyear) << "  " << proj_total_Discard(iyear) <<
"  " << proj_TotJan1B(iyear) << "  " << proj_SSB(iyear) << "  " << proj_what(iyear) << "  " <<
proj_SSB(iyear)/SSmsy << endl;
     report << endl;
  }
  else
  {
     report << "Projections not requested" << endl;
     report << endl;
  }
  report << "that's all" << endl;

  if (make_Rfile==1 && last_phase())
  {
    #include "make-Rfile_asap3.cxx"  // ADMB2R code in this file
  }

RUNTIME_SECTION
  convergence_criteria 1.0e-4
  maximum_function_evaluations 1000,1600,10000

FINAL_SECTION
  //Calculates how long is taking to run
  // this code is based on the Widow Rockfish model (from Erik H. Williams, NMFS-Santa Cruz, now Beaufort)
  time(&finish);
  elapsed_time = difftime(finish,start);
  hour = long(elapsed_time)/3600;
  minute = long(elapsed_time)%3600/60;
  second = (long(elapsed_time)%3600)%60;
  cout<<endl<<endl<<"starting time: "<<ctime(&start);
  cout<<"finishing time: "<<ctime(&finish);
  cout<<"This run took: ";
  cout<<hour<<" hours, "<<minute<<" minutes, "<<second<<" seconds."<<endl<<endl<<endl;
```

Appendix 2: make-Rfile_asap3.cxx (to make rdat file)

```
// this is the file that creates the R data object

//========================================================================
// Open the output file using the AD Model Builder template name, and
// specify 6 digits of precision
// use periods in R variable names instead of underscore

// variables used for naming fleets and indices
adstring ifleetchar;
adstring indchar;
adstring onenum(4);
adstring onednm(4);
adstring twodnm(4);

open_r_file(adprogram_name + ".rdat", 6, -99999);

  // metadata
  open_r_info_list("info", true);
      wrt_r_item("program", "ASAP3");
  close_r_info_list();


  // basic parameter values
  open_r_info_list("parms", false);
      wrt_r_item("styr", year1);
      wrt_r_item("endyr", (year1+nyears-1));
      wrt_r_item("nyears", nyears);
      wrt_r_item("nages", nages);
      wrt_r_item("nfleets", nfleets);
      wrt_r_item("nselblocks", nselblocks);
      wrt_r_item("navailindices", navailindices);
```

```
    wrt_r_item("nindices", nindices);
close_r_info_list();

// run options
open_r_info_list("options", false);
    wrt_r_item("isfecund", isfecund);
    wrt_r_item("frac.yr.spawn", fracyearSSB);
    wrt_r_item("do.projections", do_projections);
    wrt_r_item("ignore.guesses", ignore_guesses);
    wrt_r_item("Freport.agemin", Freport_agemin);
    wrt_r_item("Freport.agemax", Freport_agemax);
    wrt_r_item("Freport.wtopt", Freport_wtopt);
    wrt_r_item("use.likelihood.constants", use_likelihood_constants);
    wrt_r_item("Fmult.max.value", Fmult_max_value);
    wrt_r_item("N.year1.flag",NAA_year1_flag);
    wrt_r_item("do.mcmc",doMCMC);
close_r_info_list();

// Likelihood contributions
open_r_info_list("like", false);
    wrt_r_item("lk.total", obj_fun);
    wrt_r_item("lk.catch.total", (lambda_catch_tot*catch_tot_likely));
    wrt_r_item("lk.discard.total", (lambda_Discard_tot*discard_tot_likely));
    wrt_r_item("lk.index.fit.total", (lambda_ind*likely_ind));
    wrt_r_item("lk.catch.age.comp", likely_catch);
    wrt_r_item("lk.discards.age.comp", likely_Discard);
    wrt_r_item("lk.index.age.comp", likely_index_age_comp);
    wrt_r_item("lk.sel.param.total", sum_sel_lambda_likely);
    wrt_r_item("lk.index.sel.param.total", sum_indexsel_lambda_likely);
    wrt_r_item("lk.q.year1", (lambda_q_year1*q_year1_likely));
    wrt_r_item("lk.q.devs", (lambda_q_devs*q_devs_likely));
    wrt_r_item("lk.Fmult.year1.total", (lambda_Fmult_year1*Fmult_year1_likely));
    wrt_r_item("lk.Fmult.devs.total", (lambda_Fmult_devs*Fmult_devs_likely));
    wrt_r_item("lk.N.year1", (lambda_N_year1_devs*N_year1_likely));
    wrt_r_item("lk.Recruit.devs", (lambda_recruit_devs*likely_SR_sigma));
    wrt_r_item("lk.SR.steepness", (lambda_steepness*steepness_likely));
    wrt_r_item("lk.SR.scaler", (lambda_SR_scaler*SR_scaler_likely));
    wrt_r_item("lk.Fmult.Max.penalty", Fmult_max_pen);
    wrt_r_item("lk.F.penalty", fpenalty);
close_r_info_list();

// fleet, block, and index specific likelihood contributions
open_r_info_list("like.additional", false);
    wrt_r_item("nfleets",nfleets);
    wrt_r_item("nindices",nindices);
    wrt_r_item("nselparms",nselparm);
    wrt_r_item("nindexselparms",nindexselparms);
    if (nfleets>1)
    {
        for (ifleet=1;ifleet<=nfleets;ifleet++)
        {
            if (nfleets < 10) itoa(ifleet, onenum, 10);
            else onenum="0";
            ifleetchar = "fleet" + onenum;
            adstring lk_catch_fleet = adstring("lk.catch.") + ifleetchar;
            wrt_r_item(lk_catch_fleet,(lambda_catch_tot(ifleet)*catch_tot_likely(ifleet)));
        }

        for (ifleet=1;ifleet<=nfleets;ifleet++)
        {
            if (nfleets < 10) itoa(ifleet, onenum, 10);
            else onenum="0";
            ifleetchar = "fleet" + onenum;
            adstring lk_discard_fleet = adstring("lk.discard.") + ifleetchar;
            wrt_r_item(lk_discard_fleet,(lambda_Discard_tot(ifleet)*discard_tot_likely(ifleet)));
        }

        for (ifleet=1;ifleet<=nfleets;ifleet++)
        {
            if (nfleets < 10) itoa(ifleet, onenum, 10);
            else onenum="0";
```

```
        ifleetchar = "fleet" + onenum;
        adstring lk_Fmult_year1_fleet = adstring("lk.Fmult.year1.") + ifleetchar;
        wrt_r_item(lk_Fmult_year1_fleet,(lambda_Fmult_year1(ifleet)*Fmult_year1_likely(ifleet)));
    }

    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
        ifleetchar = "fleet" + onenum;
        adstring lk_Fmult_devs_fleet = adstring("lk.Fmult.devs.") + ifleetchar;
        wrt_r_item(lk_Fmult_devs_fleet,(lambda_Fmult_devs(ifleet)*Fmult_devs_likely(ifleet)));
    }
}

if (nindices>1)
{
    for (ind=1;ind<=nindices;ind++)
    {
        if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
        {
            itoa(ind, onednm, 10);
            twodnm = "0" + onednm;
        }
        else if (ind <=99)
        {
            itoa(ind,twodnm, 10);
        }
        else
        {
            twodnm = "00";
        }
        indchar = "ind" + twodnm;
        adstring lk_index_fit_ind = adstring("lk.index.fit.") + indchar;
        wrt_r_item(lk_index_fit_ind,(lambda_ind(ind)*likely_ind(ind)));
    }

    for (ind=1;ind<=nindices;ind++)
    {
        if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
        {
            itoa(ind, onednm, 10);
            twodnm = "0" + onednm;
        }
        else if (ind <=99)
        {
            itoa(ind,twodnm, 10);
        }
        else
        {
            twodnm = "00";
        }
        indchar = "ind" + twodnm;
        adstring lk_q_year1_ind = adstring("lk.q.year1.") + indchar;
        wrt_r_item(lk_q_year1_ind,(lambda_q_year1(ind)*q_year1_likely(ind)));
    }

    for (ind=1;ind<=nindices;ind++)
    {
        if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
        {
            itoa(ind, onednm, 10);
            twodnm = "0" + onednm;
        }
        else if (ind <=99)
        {
            itoa(ind,twodnm, 10);
        }
        else
        {
            twodnm = "00";
```

```
            }
            indchar = "ind" + twodnm;
            adstring lk_q_devs_ind = adstring("lk.q.devs.") + indchar;
            wrt_r_item(lk_q_devs_ind,(lambda_q_devs(ind)*q_devs_likely(ind)));
        }
    }

    for (k=1;k<=nselparm;k++)
    {
        if (sel_phase(k) >=1)
        {
            if (k <= 9)  // note have to deal with one digit and two digit numbers separately
            {
                itoa(k, onednm, 10);
                twodnm = "0" + onednm;
            }
            else if (k <=99)
            {
                itoa(k, twodnm, 10);
            }
            else
            {
                twodnm = "00";
            }
            adstring lk_sel_param = adstring("lk.sel.param.") + twodnm;
            wrt_r_item(lk_sel_param,(sel_lambda(k)*sel_likely(k)));

        }
    }

    for (k=1;k<=nindexselparms;k++)
    {
        if (indexsel_phase(k) >=1)
        {
            if (k <= 9)  // note have to deal with one digit and two digit numbers separately
            {
                itoa(k, onednm, 10);
                twodnm = "0" + onednm;
            }
            else if (k <=99)
            {
                itoa(k, twodnm, 10);
            }
            else
            {
                twodnm = "00";
            }
            adstring lk_indexsel_param = adstring("lk.indexsel.param.") + twodnm;
            wrt_r_item(lk_indexsel_param,(indexsel_lambda(k)*indexsel_likely(k)));

        }
    }

close_r_info_list();

// initial guesses
open_r_list("initial.guesses");
    open_r_info_list("SR.inits", false);
        wrt_r_item("is.SR.scaler.R",is_SR_scaler_R);
        wrt_r_item("SR.scaler.init",SR_scaler_ini);
        wrt_r_item("SR_steepness.init",SR_steepness_ini);
    close_r_info_list();
    wrt_r_complete_vector("NAA.year1.init",NAA_year1_ini);
    wrt_r_complete_vector("Fmult.year1.init",Fmult_year1_ini);
    wrt_r_complete_vector("q.year1.init",q_year1_ini);
    wrt_r_complete_vector("release.mort", release_mort);
    wrt_r_complete_vector("index.use.flag",use_index);
close_r_list();

// control parameters
open_r_list("control.parms");
```

```
      open_r_info_list("phases", false);
          wrt_r_item("phase.Fmult.year1", phase_Fmult_year1);
          wrt_r_item("phase.Fmult.devs", phase_Fmult_devs);
          wrt_r_item("phase.recruit.devs", phase_recruit_devs);
          wrt_r_item("phase.N.year1.devs", phase_N_year1_devs);
          wrt_r_item("phase.q.year1", phase_q_year1);
          wrt_r_item("phase.q.devs", phase_q_devs);
          wrt_r_item("phase.SR.scaler", phase_SR_scaler);
          wrt_r_item("phase.steepness", phase_steepness);
      close_r_info_list();
      open_r_info_list("singles", false);
          wrt_r_item("lambda.N.year1.devs",lambda_N_year1_devs);
          wrt_r_item("N.year1.cv",N_year1_CV);
          wrt_r_item("lambda.recruit.devs",lambda_recruit_devs);
          wrt_r_item("lambda.steepness",lambda_steepness);
          wrt_r_item("steepness.cv",steepness_CV);
          wrt_r_item("lambda.SR.scaler",lambda_SR_scaler);
          wrt_r_item("SR.scaler.cv", SR_scaler_CV);
      close_r_info_list();
      open_r_info_list("mcmc", false);
          wrt_r_item("mcmc.nyear.opt",MCMCnyear_opt);
          wrt_r_item("mcmc.n.boot",MCMCnboot);
          wrt_r_item("mcmc.n.thin",MCMCnthin);
          wrt_r_item("mcmc.seed",MCMCseed);
          wrt_r_item("fillR.opt",fillR_opt);
          wrt_r_item("Ravg.start",Ravg_start);
          wrt_r_item("Ravg.end",Ravg_end);
      close_r_info_list();
      wrt_r_complete_vector("recruit.cv",recruit_CV);
      wrt_r_complete_vector("lambda.ind",lambda_ind);
      wrt_r_complete_vector("lambda.catch.tot",lambda_catch_tot);
      open_r_matrix("catch.tot.cv");
          wrt_r_matrix(catch_tot_CV, 2, 2);
          wrt_r_namevector(year1, (year1+nyears-1));
          wrt_r_namevector(1, nfleets);
      close_r_matrix();
      wrt_r_complete_vector("lambda.Discard.tot",lambda_Discard_tot);
      open_r_matrix("discard.tot.cv");
          wrt_r_matrix(discard_tot_CV, 2, 2);
          wrt_r_namevector(year1, (year1+nyears-1));
          wrt_r_namevector(1, nfleets);
      close_r_matrix();
      wrt_r_complete_vector("lambda.Fmult.year1",lambda_Fmult_year1);
      wrt_r_complete_vector("Fmult.year1.cv",Fmult_year1_CV);
      wrt_r_complete_vector("lambda.Fmult.devs",lambda_Fmult_devs);
      wrt_r_complete_vector("Fmult.devs.cv",Fmult_devs_CV);
      wrt_r_complete_vector("lambda.q.year1",lambda_q_year1);
      wrt_r_complete_vector("q.year1.cv",q_year1_CV);
      wrt_r_complete_vector("lambda.q.devs",lambda_q_devs);
      wrt_r_complete_vector("q.devs.cv",q_devs_CV);
      wrt_r_complete_vector("directed.fleet",directed_fleet);
      wrt_r_complete_vector("WAA.point.bio",WAApointbio);
      wrt_r_complete_vector("index.units.aggregate", index_units_aggregate);
      wrt_r_complete_vector("index.units.proportions", index_units_proportions);
      wrt_r_complete_vector("index.WAA.point", index_WAApoint);
      wrt_r_complete_vector("index.month", index_month);
      wrt_r_complete_vector("index.sel.start.age",index_start_age);
      wrt_r_complete_vector("index.sel.end.age",index_end_age);
      wrt_r_complete_vector("index.sel.choice",index_sel_choice);
      wrt_r_complete_vector("index.age.comp.flag",index_estimate_proportions);
  close_r_list();

  // selectivity input matrices for fleets and indices
  open_r_list("sel.input.mats");
      // input selectivity matrix, contains combinations of values not used, see fleet_sel_option to determine
which choice was made for each block
      open_r_matrix("fleet.sel.ini");
          wrt_r_matrix(sel_ini, 2, 2);
          wrt_r_namevector(1, (nselblocks*(nages+6)));
          wrt_r_namevector(1, 4);
      close_r_matrix();
```

```
    open_r_matrix("index.sel.ini");
        wrt_r_matrix(index_sel_ini, 2, 2);
        wrt_r_namevector(1, (navailindices*(nages+6)));
        wrt_r_namevector(1, 4);
    close_r_matrix();
close_r_list();

// Weight at Age matrices
open_r_list("WAA.mats");
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
        ifleetchar = "fleet" + onenum;
        adstring WAA_c_fleet = adstring("WAA.catch.") + ifleetchar;
        open_r_matrix(WAA_c_fleet);
            wrt_r_matrix(WAAcatchfleet(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(1,nages);
        close_r_matrix();
        adstring WAA_d_fleet = adstring("WAA.discard.") + ifleetchar;
        open_r_matrix(WAA_d_fleet);
            wrt_r_matrix(WAAdiscardfleet(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(1,nages);
        close_r_matrix();
    }
    open_r_matrix("WAA.catch.all");
        wrt_r_matrix(WAAcatchall, 2, 2);
        wrt_r_namevector(year1, (year1+nyears-1));
        wrt_r_namevector(1, nages);
    close_r_matrix();

    open_r_matrix("WAA.discard.all");
        wrt_r_matrix(WAAdiscardall, 2, 2);
        wrt_r_namevector(year1, (year1+nyears-1));
        wrt_r_namevector(1, nages);
    close_r_matrix();

    open_r_matrix("WAA.ssb");
        wrt_r_matrix(WAAssb, 2, 2);
        wrt_r_namevector(year1, (year1+nyears-1));
        wrt_r_namevector(1, nages);
    close_r_matrix();

    open_r_matrix("WAA.jan1");
        wrt_r_matrix(WAAjan1b, 2, 2);
        wrt_r_namevector(year1, (year1+nyears-1));
        wrt_r_namevector(1, nages);
    close_r_matrix();

    for (ind=1;ind<=nindices;ind++)
    {
        if (index_units_aggregate(ind)==1 || index_units_proportions(ind)==1)
        {
            if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
            {
                itoa(ind, onednm, 10);
                twodnm = "0" + onednm;
            }
            else if (ind <=99)
            {
                itoa(ind,twodnm, 10);
            }
            else
            {
                twodnm = "00";
            }
            indchar = "ind" + twodnm;
            adstring index_WAA_name = adstring("index.WAA.") + indchar;
```

```
            open_r_matrix(index_WAA_name);
                wrt_r_matrix(index_WAA(ind), 2, 2);
                wrt_r_namevector(year1, (year1+nyears-1));
                wrt_r_namevector(1,nages);
            close_r_matrix();
        }
    }

  close_r_list();

  // Year by Age Matrices (not fleet specific): M, maturity, fecundity, N, Z, F,
  open_r_matrix("M.age");
      wrt_r_matrix(M, 2, 2);
      wrt_r_namevector(year1, (year1+nyears-1));
      wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("maturity");
      wrt_r_matrix(mature, 2, 2);
      wrt_r_namevector(year1, (year1+nyears-1));
      wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("fecundity");
      wrt_r_matrix(fecundity, 2, 2);
      wrt_r_namevector(year1, (year1+nyears-1));
      wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("N.age");
      wrt_r_matrix(NAA, 2, 2);
      wrt_r_namevector(year1, (year1+nyears-1));
      wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("Z.age");
      wrt_r_matrix(Z, 2, 2);
      wrt_r_namevector(year1, (year1+nyears-1));
      wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("F.age");
      wrt_r_matrix(FAA_tot, 2, 2);
      wrt_r_namevector(year1, (year1+nyears-1));
      wrt_r_namevector(1, nages);
  close_r_matrix();

  // Fleet by Year Matrices: Catch.tot.obs, Catch.tot.pred, Catch.tot.resid), Discard.tot.obs, Discard.tot.pred,
Discard.tot.resid
  open_r_matrix("catch.obs");
      wrt_r_matrix(Catch_tot_fleet_obs, 2, 2);
      wrt_r_namevector(1, nfleets);
      wrt_r_namevector(year1, (year1+nyears-1));
  close_r_matrix();

  open_r_matrix("catch.pred");
      wrt_r_matrix(Catch_tot_fleet_pred, 2, 2);
      wrt_r_namevector(1, nfleets);
      wrt_r_namevector(year1, (year1+nyears-1));
  close_r_matrix();

  open_r_matrix("catch.std.resid");
      wrt_r_matrix(Catch_stdresid, 2, 2);
      wrt_r_namevector(1, nfleets);
      wrt_r_namevector(year1, (year1+nyears-1));
  close_r_matrix();

  open_r_matrix("discard.obs");
      wrt_r_matrix(Discard_tot_fleet_obs, 2, 2);
      wrt_r_namevector(1, nfleets);
      wrt_r_namevector(year1, (year1+nyears-1));
```

```
close_r_matrix();

open_r_matrix("discard.pred");
    wrt_r_matrix(Discard_tot_fleet_pred, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();

open_r_matrix("discard.std.resid");
    wrt_r_matrix(Discard_stdresid, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();


// Age Compositions: Catch and Discards observed and predicted by fleet
open_r_list("catch.comp.mats");
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
        ifleetchar = "fleet" + onenum;
        adstring ccomp_ob = adstring("catch.") + ifleetchar + adstring(".ob");
        open_r_matrix(ccomp_ob);
            wrt_r_matrix(CAA_prop_obs(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(sel_start_age(ifleet), sel_end_age(ifleet));
        close_r_matrix();

        adstring ccomp_pr = adstring("catch.") + ifleetchar + adstring(".pr");
        open_r_matrix(ccomp_pr);
            wrt_r_matrix(CAA_prop_pred(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(sel_start_age(ifleet), sel_end_age(ifleet));
        close_r_matrix();

        adstring dcomp_ob = adstring("discard.") + ifleetchar + adstring(".ob");
        open_r_matrix(dcomp_ob);
            wrt_r_matrix(Discard_prop_obs(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(sel_start_age(ifleet), sel_end_age(ifleet));
        close_r_matrix();

        adstring dcomp_pr = adstring("discard.") + ifleetchar + adstring(".pr");
        open_r_matrix(dcomp_pr);
            wrt_r_matrix(Discard_prop_pred(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(sel_start_age(ifleet), sel_end_age(ifleet));
        close_r_matrix();
    }
close_r_list();


// fleet selectivity blocks
open_r_matrix("fleet.sel.blocks");
    wrt_r_matrix(sel_blocks, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();

// vectors of fleet selectivity options
wrt_r_complete_vector("fleet.sel.start.age",sel_start_age);
wrt_r_complete_vector("fleet.sel.end.age",sel_end_age);
wrt_r_complete_vector("fleet.sel.option",sel_option);

// selecivity matrices for each fleet
open_r_list("fleet.sel.mats");
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
```

```
        ifleetchar = "fleet" + onenum;
        adstring sel_fleet_char = adstring("sel.m.") + ifleetchar;
        open_r_matrix(sel_fleet_char);
            wrt_r_matrix(sel_by_fleet(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(1, nages);
        close_r_matrix();
    }
close_r_list();

// Fmults by fleet
open_r_matrix("fleet.Fmult");
    wrt_r_matrix(Fmult, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();

// FAA by fleet directed and discarded
open_r_list("fleet.FAA");
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
        ifleetchar = "fleet" + onenum;

        adstring fleet_FAA_dir = adstring("FAA.directed.") + ifleetchar;
        open_r_matrix(fleet_FAA_dir);
            wrt_r_matrix(FAA_by_fleet_dir(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(1,nages);
        close_r_matrix();

        adstring fleet_FAA_discard = adstring("FAA.discarded.") + ifleetchar;
        open_r_matrix(fleet_FAA_discard);
            wrt_r_matrix(FAA_by_fleet_Discard(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(1,nages);
        close_r_matrix();
    }
close_r_list();

// proportion release year by age matrices by fleet
open_r_list("fleet.prop.release");
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
        ifleetchar = "fleet" + onenum;
        adstring fleet_prop_release = adstring("prop.release.") + ifleetchar;
        open_r_matrix(fleet_prop_release);
            wrt_r_matrix(proportion_release(ifleet), 2, 2);
            wrt_r_namevector(year1, (year1+nyears-1));
            wrt_r_namevector(1,nages);
        close_r_matrix();
    }
close_r_list();

// fleet specific annual effective sample sizes input and estimated for catch and discards
open_r_matrix("fleet.catch.Neff.init");
    wrt_r_matrix(input_eff_samp_size_catch, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();

open_r_matrix("fleet.catch.Neff.est");
    wrt_r_matrix(effective_sample_size, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();

open_r_matrix("fleet.discard.Neff.init");
```

```
    wrt_r_matrix(input_eff_samp_size_discard, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();

open_r_matrix("fleet.discard.Neff.est");
    wrt_r_matrix(effective_Discard_sample_size, 2, 2);
    wrt_r_namevector(1, nfleets);
    wrt_r_namevector(year1, (year1+nyears-1));
close_r_matrix();

// vector of q for each index if qdevs turned off, otherwise a list with vectors for each index
if (phase_q_devs <= 0)
{
    wrt_r_complete_vector("q.indices",  column(q_by_index,1));
}
else
{
    open_r_list("q.random.walk");
        for (ind=1;ind<=nindices;ind++)
        {
            if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
            {
                itoa(ind, onednm, 10);
                twodnm = "0" + onednm;
            }
            else if (ind <=99)
            {
                itoa(ind,twodnm, 10);
            }
            else
            {
                twodnm = "00";
            }
            indchar = "ind" + twodnm;
            adstring q_ind = adstring("q.") + indchar;
            wrt_r_complete_vector(q_ind,q_by_index(ind));
        }
    close_r_list();
}

// vectors for Freport and Biomasses (TotJan1B, SSB, ExploitableB)
wrt_r_complete_vector("F.report",Freport);
wrt_r_complete_vector("tot.jan1.B",TotJan1B);
wrt_r_complete_vector("SSB",SSB);
wrt_r_complete_vector("exploitable.B",ExploitableB);


// F reference values
open_r_info_list("Fref", false);
    wrt_r_item("Fmax", Fmax_report);
    wrt_r_item("F01", F01_report);
    wrt_r_item("F30", F30SPR_report);
    wrt_r_item("F40", F40SPR_report);
    wrt_r_item("Fcurrent", Freport(nyears));
close_r_info_list();

// SR curve parameters
open_r_info_list("SR.parms", false);
    wrt_r_item("SR.alpha", SR_alpha);
    wrt_r_item("SR.beta", SR_beta);
    wrt_r_item("SR.SPR0", SR_spawners_per_recruit);
    wrt_r_item("SR.S0", SR_S0);
    wrt_r_item("SR.R0", SR_R0);
    wrt_r_item("SR.steepness", SR_steepness);
close_r_info_list();

// SR obs, pred, devs, and standardized resids
// note year coresponds to age-1 recruitment, when plot SR curve have to offset SSB and R by one year
open_r_df("SR.resids", year1, (year1+nyears-1), 2);
    wrt_r_namevector(year1, (year1+nyears-1));
```

```
      wrt_r_df_col("year", year1, (year1+nyears-1));
      wrt_r_df_col("recruits", recruits, year1);
      wrt_r_df_col("R.no.devs", SR_pred_recruits, year1);
      wrt_r_df_col("logR.dev", log_recruit_devs, year1);
      wrt_r_df_col("SR.std.resid", SR_stdresid, year1);
  close_r_df();

  // annual values for S0_vec, R0_vec, steepness_vec, s_per_r_vec (last year values should match SR.parms
values)
  open_r_df("SR.annual.parms", year1, (year1+nyears-1), 2);
      wrt_r_namevector(year1, (year1+nyears-1));
      wrt_r_df_col("year", year1, (year1+nyears-1));
      wrt_r_df_col("S0.vec", S0_vec, year1);
      wrt_r_df_col("R0.vec", R0_vec, year1);
      wrt_r_df_col("steepness.vec", steepness_vec, year1);
      wrt_r_df_col("s.per.r.vec",s_per_r_vec, year1);
  close_r_df();



  // index stuff starts here

  // selectivity by index
  open_r_matrix("index.sel");
      wrt_r_matrix(indexsel, 2, 2);
      wrt_r_namevector(1, nindices);
      wrt_r_namevector(1, nages);
  close_r_matrix();

  wrt_r_complete_vector("index.nobs",index_nobs);

  // index year counter (sequential numbers starting at 1 for first year)
  open_r_list("index.year.counter");
      for (ind=1;ind<=nindices;ind++)
      {
          if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
          {
              itoa(ind, onednm, 10);
              twodnm = "0" + onednm;
          }
          else if (ind <=99)
          {
              itoa(ind,twodnm, 10);
          }
          else
          {
              twodnm = "00";
          }
          indchar = "ind" + twodnm;
          wrt_r_complete_vector(indchar,index_time(ind));
      }
  close_r_list();

  // index years
  open_r_list("index.year");
      for (ind=1;ind<=nindices;ind++)
      {
          if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
          {
              itoa(ind, onednm, 10);
              twodnm = "0" + onednm;
          }
          else if (ind <=99)
          {
              itoa(ind,twodnm, 10);
          }
          else
          {
              twodnm = "00";
          }
          indchar = "ind" + twodnm;
```

```
        wrt_r_complete_vector(indchar,index_year(ind));
    }
close_r_list();

// index CV
open_r_list("index.cv");
    for (ind=1;ind<=nindices;ind++)
    {
        if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
        {
            itoa(ind, onednm, 10);
            twodnm = "0" + onednm;
        }
        else if (ind <=99)
        {
            itoa(ind,twodnm, 10);
        }
        else
        {
            twodnm = "00";
        }
        indchar = "ind" + twodnm;
        wrt_r_complete_vector(indchar,index_cv(ind));
    }
close_r_list();

// index sigmas (derived from input CV)
open_r_list("index.sigma");
    for (ind=1;ind<=nindices;ind++)
    {
        if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
        {
            itoa(ind, onednm, 10);
            twodnm = "0" + onednm;
        }
        else if (ind <=99)
        {
            itoa(ind,twodnm, 10);
        }
        else
        {
            twodnm = "00";
        }
        indchar = "ind" + twodnm;
        wrt_r_complete_vector(indchar,index_sigma(ind));
    }
close_r_list();

// index observations
open_r_list("index.obs");
    for (ind=1;ind<=nindices;ind++)
    {
        if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
        {
            itoa(ind, onednm, 10);
            twodnm = "0" + onednm;
        }
        else if (ind <=99)
        {
            itoa(ind,twodnm, 10);
        }
        else
        {
            twodnm = "00";
        }
        indchar = "ind" + twodnm;
        wrt_r_complete_vector(indchar,index_obs(ind));
    }
close_r_list();

// predicted indices
```

```
  open_r_list("index.pred");
      for (ind=1;ind<=nindices;ind++)
      {
          if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
          {
              itoa(ind, onednm, 10);
              twodnm = "0" + onednm;
          }
          else if (ind <=99)
          {
              itoa(ind,twodnm, 10);
          }
          else
          {
              twodnm = "00";
          }
          indchar = "ind" + twodnm;
          wrt_r_complete_vector(indchar,index_pred(ind));
      }
  close_r_list();

  // index standardized residuals
  open_r_list("index.std.resid");
      for (ind=1;ind<=nindices;ind++)
      {
          if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
          {
              itoa(ind, onednm, 10);
              twodnm = "0" + onednm;
          }
          else if (ind <=99)
          {
              itoa(ind,twodnm, 10);
          }
          else
          {
              twodnm = "00";
          }
          indchar = "ind" + twodnm;
          wrt_r_complete_vector(indchar,index_stdresid(ind));
      }
  close_r_list();

  // index proportions at age related output
  if (max(index_estimate_proportions)>0)  // check to see if any West Coast style indices, skip this section if
all are East Coast style
  {
      // Index Age Comp
      open_r_list("index.comp.mats");
          for (ind=1;ind<=nindices;ind++)
          {
              if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
              {
                  itoa(ind, onednm, 10);
                  twodnm = "0" + onednm;
              }
              else if (ind <=99)
              {
                  itoa(ind,twodnm, 10);
              }
              else
              {
                  twodnm = "00";
              }
              indchar = "ind" + twodnm;

              adstring acomp_ob = indchar + adstring(".ob");
              open_r_matrix(acomp_ob);
                  wrt_r_matrix(output_index_prop_obs(ind), 2, 2);
                  wrt_r_namevector(year1, (year1+nyears-1));
                  wrt_r_namevector(1,nages);
```

```
            close_r_matrix();

            adstring acomp_pr = indchar + adstring(".pr");
            open_r_matrix(acomp_pr);
                wrt_r_matrix(output_index_prop_pred(ind), 2, 2);
                wrt_r_namevector(year1, (year1+nyears-1));
                wrt_r_namevector(1, nages);
            close_r_matrix();
        }
    close_r_list();

  // Neff for indices initial guess
  open_r_matrix("index.Neff.init");
      wrt_r_matrix(index_Neff_init, 2, 2);
      wrt_r_namevector(1, nindices);
      wrt_r_namevector(year1, (year1+nyears-1));
  close_r_matrix();

  // Neff for indices estimated
  open_r_matrix("index.Neff.est");
      wrt_r_matrix(index_Neff_est, 2, 2);
      wrt_r_namevector(1, nindices);
      wrt_r_namevector(year1, (year1+nyears-1));
  close_r_matrix();

}  // end if-statement to test for any index age comp


// deviations section: only reported if associated with lambda > 0
if (lambda_N_year1_devs > 0)
{
    // note: obs and pred include age 1 while std.resid does not - do not use age 1 when plotting
    open_r_list("deviations.N.year1");
        wrt_r_complete_vector("N.year1.obs",NAA(1));
        wrt_r_complete_vector("N.year1.pred",nyear1temp);
        wrt_r_complete_vector("N.year1.std.resid",N_year1_stdresid);
    close_r_list();
}


// RMSE number of observations section
open_r_info_list("RMSE.n", false);
    if (nfleets>1)
    {
        for (ifleet=1;ifleet<=nfleets;ifleet++)
        {
            if (nfleets < 10) itoa(ifleet, onenum, 10);
            else onenum="0";
            ifleetchar = "fleet" + onenum;
            adstring rmse_n_catch_fleet = adstring("rmse.n.catch.") + ifleetchar;
            wrt_r_item(rmse_n_catch_fleet,nyears);
        }
    }
    wrt_r_item("rmse.n.catch.tot",(nyears*nfleets));

    if (nfleets>1)
    {
        for (ifleet=1;ifleet<=nfleets;ifleet++)
        {
            if (nfleets < 10) itoa(ifleet, onenum, 10);
            else onenum="0";
            ifleetchar = "fleet" + onenum;
            adstring rmse_n_discard_fleet = adstring("rmse.n.discard.") + ifleetchar;
            if (sum(Discard_tot_fleet_obs(ifleet)) > 0)
            {
                wrt_r_item(rmse_n_discard_fleet,nyears);
            }
            else
            {
                wrt_r_item(rmse_n_discard_fleet,0);
            }
```

```
        }
    }
    if (sum(Discard_tot_fleet_obs) > 0)
    {
        wrt_r_item("rmse.n.discard.tot",(nyears*nfleets));
    }
    else
    {
        wrt_r_item("rmse.n.discard.tot",0);
    }

    if (nindices>1)
    {
        for (ind=1;ind<=nindices;ind++)
        {
            if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
            {
                itoa(ind, onednm, 10);
                twodnm = "0" + onednm;
            }
            else if (ind <=99)
            {
                itoa(ind,twodnm, 10);
            }
            else
            {
                twodnm = "00";
            }
            indchar = "ind" + twodnm;
            adstring rmse_n_ind = adstring("rmse.n.") + indchar;
            wrt_r_item(rmse_n_ind,index_nobs(ind));
        }
    }
    wrt_r_item("rmse.n.ind.total",sum(index_nobs));

    wrt_r_item("rmse.n.N.year1",N_year1_rmse_nobs);

    wrt_r_item("rmse.n.Fmult.year1",Fmult_year1_rmse_nobs);

    if (nfleets>1)
    {
        for (ifleet=1;ifleet<=nfleets;ifleet++)
        {
            if (nfleets < 10) itoa(ifleet, onenum, 10);
            else onenum="0";
            ifleetchar = "fleet" + onenum;
            adstring rmse_n_Fmult_devs_fleet = adstring("rmse.n.Fmult.devs.") + ifleetchar;
            wrt_r_item(rmse_n_Fmult_devs_fleet,Fmult_devs_fleet_rmse_nobs(ifleet));
        }
    }
    wrt_r_item("rmse.n.Fmult.devs.total",Fmult_devs_rmse_nobs);

    wrt_r_item("rmse.n.recruit.devs",SR_rmse_nobs);

    wrt_r_item("rmse.n.fleet.sel.params",sel_rmse_nobs);

    wrt_r_item("rmse.n.index.sel.params",indexsel_rmse_nobs);

    wrt_r_item("rmse.n.q.year1",q_year1_rmse_nobs);

    wrt_r_item("rmse.n.q.devs",q_devs_rmse_nobs);

    wrt_r_item("rmse.n.SR.steepness",steepness_rmse_nobs);

    wrt_r_item("rmse.n.SR.scaler",SR_scaler_rmse_nobs);

close_r_info_list();

// RMSE section
open_r_info_list("RMSE", false);
    if (nfleets>1)
```

```
{
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
        ifleetchar = "fleet" + onenum;
        adstring rmse_catch_fleet = adstring("rmse.catch.") + ifleetchar;
        wrt_r_item(rmse_catch_fleet,sqrt(mean(square(Catch_stdresid(ifleet)))));
    }
}
wrt_r_item("rmse.catch.tot",sqrt(mean(square(Catch_stdresid))));

if (nfleets>1)
{
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
        if (nfleets < 10) itoa(ifleet, onenum, 10);
        else onenum="0";
        ifleetchar = "fleet" + onenum;
        adstring rmse_discard_fleet = adstring("rmse.discard.") + ifleetchar;
        if (sum(Discard_tot_fleet_obs(ifleet)) > 0)
        {
            wrt_r_item(rmse_discard_fleet,sqrt(mean(square(Discard_stdresid(ifleet)))));
        }
        else
        {
            wrt_r_item(rmse_discard_fleet,0);
        }
    }
}
if (sum(Discard_tot_fleet_obs) > 0)
{
    wrt_r_item("rmse.discard.tot",sqrt(mean(square(Discard_stdresid))));
}
else
{
    wrt_r_item("rmse.discard.tot",0);
}

if (nindices>1)
{
    for (ind=1;ind<=nindices;ind++)
    {
        if (ind <= 9)  // note have to deal with one digit and two digit numbers separately
        {
            itoa(ind, onednm, 10);
            twodnm = "0" + onednm;
        }
        else if (ind <=99)
        {
            itoa(ind,twodnm, 10);
        }
        else
        {
            twodnm = "00";
        }
        indchar = "ind" + twodnm;
        adstring rmse_ind = adstring("rmse.") + indchar;
        wrt_r_item(rmse_ind,sqrt(mean(square(index_stdresid(ind)))));
    }
}
wrt_r_item("rmse.ind.total",sqrt(mean(square(index_stdresid))));

wrt_r_item("rmse.N.year1",N_year1_rmse);

wrt_r_item("rmse.Fmult.year1",Fmult_year1_rmse);

if (nfleets>1)
{
    for (ifleet=1;ifleet<=nfleets;ifleet++)
    {
```

```
            if (nfleets < 10) itoa(ifleet, onenum, 10);
            else onenum="0";
            ifleetchar = "fleet" + onenum;
            adstring rmse_Fmult_devs_fleet = adstring("rmse.Fmult.devs.") + ifleetchar;
            wrt_r_item(rmse_Fmult_devs_fleet,Fmult_devs_fleet_rmse(ifleet));
        }
    }
    wrt_r_item("rmse.Fmult.devs.total",Fmult_devs_rmse);

    wrt_r_item("rmse.recruit.devs",SR_rmse);

    wrt_r_item("rmse.fleet.sel.params",sel_rmse);

    wrt_r_item("rmse.index.sel.params",indexsel_rmse);

    wrt_r_item("rmse.q.year1",q_year1_rmse);

    wrt_r_item("rmse.q.devs",q_devs_rmse);

    wrt_r_item("rmse.SR.steepness",steepness_rmse);

    wrt_r_item("rmse.SR.scaler",SR_scaler_rmse);

 close_r_info_list();

 open_r_list("Neff.stage2.mult");
    wrt_r_complete_vector("Neff.stage2.mult.catch", Neff_stage2_mult_catch);
    wrt_r_complete_vector("Neff.stage2.mult.discard", Neff_stage2_mult_discard);
    wrt_r_complete_vector("Neff.stage2.mult.index", Neff_stage2_mult_index);
 close_r_list();

// close file
close_r_file();
```