Northeast Fisheries Science Center Reference Document 15-17

doi:10.7289/V57W695G

Technical Details for ASAP version 4

by Timothy J. Miller and Christopher M. Legault

September 2015

Northeast Fisheries Science Center Reference Document 15-17

doi:10.7289/V57W695G

Technical Details for ASAP version 4

by Timothy J. Miller and Christopher M. Legault

NOAA Fisheries, Northeast Fisheries Science Center, 166 Water Street,
Woods Hole, MA 02543

U.S. DEPARTMENT OF COMMERCE
National Oceanic and Atmospheric Administration
National Marine Fisheries Service
Northeast Fisheries Science Center
Woods Hole, Massachusetts
September 2015

Northeast Fisheries Science Center Reference Documents

This series is a secondary scientific series designed to assure the long-term documentation and to enable the timely transmission of research results by Center and/or non-Center researchers, where such results bear upon the research mission of the Center (see the outside back cover for the mission statement). These documents receive internal scientific review, and most receive copy editing. The National Marine Fisheries Service does not endorse any proprietary material, process, or product mentioned in these documents.

All documents issued in this series since April 2001, and many documents issued prior to that date, are available online at *http://www.nefsc.noaa.gov/publications/*. The electronic version is available in PDF format to permit printing of a paper copy directly from the Internet. If you do not have Internet access, or if a desired document is one of the pre-April 2001 documents available only in the paper version, you can obtain a paper copy by contacting the senior Center author of the desired document. Refer to the title page of the document for the senior Center author's name and mailing address. If there is no Center author, or if there is corporate (*i.e.*, nonindividualized) authorship, then contact the Center's Woods Hole Laboratory Library (166 Water St., Woods Hole, MA 02543-1026).

Information Quality Act Compliance: In accordance with section 515 of Public Law 106-554, the Northeast Fisheries Science Center completed both technical and policy reviews for this report. These predissemination reviews are on file at the NEFSC Editorial Office.

This document may be cited as:

Miller TJ, Legault CM. 2015. Technical details for ASAP version 4. US Dept Commer, Northeast Fish Sci Cent Ref Doc. 15-17; 136 p. Available at: http://www.nefsc.noaa.gov/publications/
doi:10.7289/V57W695G

Contents

1 INTRODUCTION

ASAP (A Stock Assessment Program) is an age structure stock assessment modeling program originally developed by Chris Legault and Victor Restrepo while they were at the Southeast Fisheries Science Center (Legault and Restrepo 1998). Modifications made in subsequent iterations are described at the top of Appendix 1. ASAP is a variant of statistical catch-at-age models. This latest version can integrate annual catches and associated age compositions (by fleet), abundance indices and associated age compositions, annual maturity, fecundity, weight, and natural mortality at age, and annual environmental covariate effects on stock-recruit parameters or natural mortality (when estimated). It also can incorporate (size-based) calibration estimates that relate abundance index time series with periodic changes in gear. However, it is also flexible enough to handle data poor stocks without age data (dynamic pool models) or with only new and post-recruit age or size groups. Further information and instructions for new features and options are described in the ASAP version 4 User's Guide provided with the installation.

There is an extensive usage of design matrices in ASAP4 for estimating covariate effects on natural mortality, catchability, size-based calibration, and stock recruit relationships. Design matrices are commonly used in parameterizing generalized linear models and related models and they provide a natural way to analyze effects of both categorical and continuous covariates. In our adoption of the design matrix approach to parameterizing these different aspects of the assessment model, we envisioned the user considering no more than a few covariates. To reduce the chance that model parameters are confounded, the user should ensure there is no collinearity of the covariates and the rank of the matrices is full. Finally, although there are a great number of changes from ASAP version 3, the GUI will accept an ASAP version 3 input file and automatically create a default ASAP version 4 which then can be modified in various ways if the user would like to access the features available in the new version.

This document provides details on the basic equations used in the ASAP version 4. It also provides appendices containing the actual ADMB code used to generate the executable so that the exact calculations in the program are available. This document uses an extensive set of mathematical notation and some variable names in a number of places instead of symbols to facilitate understanding of the underlying code. All notation is defined in the text of the document, but it is also listed with definitions in Table 1. Two log files are produced when running the model. "ASAP4input.log" reports the data objects read into the model and "ASAP4fix.log" reports both minor modifications made to the input to allow the model to run and major problems with the input data that need to be attended to by the user before the model will run. Note that all logarithms are the natural type.

2 MODEL EQUATIONS

The description of the model generally follows the steps in the code for ease of understanding. Calculation of the objective function is described in the next section. The population has age classes $1, \dots, A$ where the last included all ages $\geq A$ (plus group) and the population is modeled from year 1 to Y. Catch and discards by fleet and abundance indices and corresponding age composition information can exist for any subset of those years. When catch does not exist, fishing mortality will be zero.

2.1 Selectivity

The approach used to estimate fleet and abundance index selectivity in ASAP version 4 is a bit different from that in version 3. Similar to fleet selectivity in ASAP version 3, there are selectivity blocks and three options for estimating selectivity within each block:

- age-specific selectivity s_a where the number of parameters is equal to the number of ages (one parameter for each age and at least one age should be fixed at 1.0 instead of estimated),

- logistic function (2 parameters: a_{50}, b)

$$s_a = \frac{1}{1 + \exp\left[-(a - a_{50})/b\right]}, \tag{1}$$

- and double-logistic (4 parameters: $a_{50,1}$, b_1, $a_{50,2}$, b_2)

$$s_a = \frac{1}{1 + \exp\left[-(a - a_{50,1})/b_1\right]} \frac{1}{1 + \exp\left[(a - a_{50,2})/b_2\right]}. \tag{2}$$

Note that when logistic or double-logistic selectivity is specified, the selectivity at age is divided by the maximum value over all ages, creating the final selectivity vector with a maximum of 1.0 for that block. This scaling provides interpretation of catchability and fishing mortality estimated by the model to be "fully selected" or the maximum value across age groups.

Unlike ASAP version 3, the user now defines selectivity blocks independently from fleets and abundance indices. Parameters can be used in multiple blocks to mirror parameters between fleets and/or abundance indices, or multiple times within the same block to share parameters within a fleet or abundance index selectivity block. The user specifies which block to use for each fleet and abundance index each year so that the same selectivity block could be used for multiple abundance indices and/or fleets. When using the GUI, the total number of selectivity parameters is determined by the software; otherwise the user must specify it. The user also specifies the upper and lower bounds, penalty weights, CVs, phases, and the position in any selectivity blocks to use the parameter.

2.2 Catchability for Abundance Indices

Catchability for each abundance index $q_{i,t,a}$ is separable into year $q_{i,t}$ and selectivity at age components $s_{t,a}$ (see Section 2.1 for details about the latter). To allow more variety of assumptions to the user, $q_{i,t}$ is modeled in ASAP4 as the product of availability of the population to the index $Avail_{i,t}$ and efficiency of the gear used to collect the data from with the index is derived $Eff_{i,t}$,

$$q_{i,t} = Avail_{i,t} Eff_{i,t}. \tag{3}$$

Availability for abundance index i is modeled as a function of a vector of $n_{Avail,i}$ coefficients β_{Avail} and a $Y \times n_{Avail,i}$ design matrix $\mathbf{X}_{Avail,i}$. For year t,

$$Avail_{i,t} = l_{Avail_i} + \frac{u_{Avail_i} - l_{Avail_i}}{1 + \exp\left[-\boldsymbol{X}^T_{Avail,i,t}\beta_{Avail,i}\right]} \tag{4}$$

where $\boldsymbol{X}_{Avail,i,t}$ is the row of the design matrix corresponding to year t, and l_{Avail_i} and u_{Avail_i} are the lower and upper bounds on availability for abundance index i. Autoregressive models of availability are possible when availability deviation parameters $Adev_{i,t}$ are active. In this case, availability is specified as

$$Avail_{i,t} = l_{Avail_i} + \frac{u_{Avail_i} - l_{Avail_i}}{1 + \exp\left[-A_{i,t}\right]} \tag{5}$$

where

$$A_{i,t_{min,i}} = \boldsymbol{X}^T_{Avail,i,t_{min,i}}\beta_{Avail,i} \tag{6}$$

and for $t_{min,i} < t \leq t_{max,i}$

$$A_{i,t} = A_{i,t-1} + \boldsymbol{X}^T_{Avail,i,t}\beta_{Avail} + Adev_{i,t} \tag{7}$$

where $t_{min,i}$ and $t_{max,i}$ are the first and last years where the abundance index is observed. Missing observations between $t_{min,i}$ and $t_{max,i}$ are allowed. To specify a random walk (as was possible for catchability in ASAP version 3), activate the deviations $Adev_{i,t}$ and specify $\mathbf{X}_{Avail,i}$ to be a single column with a 1 at the first observation (at $t_{min,i}$) and 0 elsewhere,

$$\mathbf{X}_{Avail,i} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{8}$$

Activating the deviations with covariates in the design matrix $\mathbf{X}_{Avail,i}$ would fit a more complicated ARMAX-type model for availability which may be difficult to interpret.

Similar to availability, gear efficiency is parameterized as a function of a vector of $n_{Eff,i}$ coefficients $\boldsymbol{\beta}_{Eff,i}$ and a $Y \times n_{Eff,i}$ design matrix $\mathbf{X}_{Eff,i}$

$$Eff_{i,t} = l_{Eff_i} + \frac{u_{Eff_i} - l_{Eff_i}}{1 + \exp\left[-\boldsymbol{X}^T_{Eff,i,t}\boldsymbol{\beta}_{Eff,i}\right]} \tag{9}$$

where $\boldsymbol{X}_{Eff,i,t}$ is the row of the design matrix corresponding to year t, and l_{Eff_i} and u_{Eff_i} are the lower and upper bounds on gear efficiency for abundance index i specified by the user. There are no gear efficiency deviations.

The coefficients may or may not be estimated depending on whether phases are > 0. Both availability and gear efficiency are essentially parameterized as a sort of generalized logistic model with an arbitrary scale and location determined by the upper and lower bounds. That is, a typical logistic model would be obtained if the lower and upper bounds were 0 and 1. This same method is also used in Section 2.4 for steepness of the Beverton-Holt stock-recruit relationship. To model a single catchability term as in ASAP version 3, treat availability as catchability by fixing $Eff_{i,t} = 1$, which is accomplished by specifying an appropriate $\mathbf{X}_{Eff,i}$ while fixing $\boldsymbol{\beta}_{Eff,i}$ with proper lower and upper bounds. For example, this can be accomplished by fixing $\boldsymbol{\beta}_{Eff,i} = 0$ with the lower and upper bounds of 0 and 2, and specifying $\boldsymbol{X}_{Eff,i}$ to be a single column of 1s. To model the random walk in catchability as was possible in ASAP version 3, follow the directions above for the random walk in availability (which is interpreted as catchability with gear efficiency fixed at 1).

Modeling catchability as these components allows the user to include different sources of information explicitly. For example, there maybe information about availability of the stock to a fishery-independent survey when that is the source of information for the abundance index, or there may be information about efficiency of the gear that is used for the abundance index.

2.3 Mortality Rates

Like catchability, fishing mortality for fleet f in year t at age a is assumed to be separable as a product of a year effect ($Fmult_{f,t}$) and selectivity at age $s_{f,t,a}$ (see Section 2.1). If the input catch for a fleet is ≤ 0 in year t, $Fmult_{f,t} = e^{-1000}$. This allows fleets to operate during different years or the population to be modeled for a specified number of years before any fishing occurs. For a fleet operating in years $t_{min,f}, t_2, \ldots, t_{max,f}$, the $Fmult$ in the first year is calculated as

$$Fmult_{f,t_{min,f}} = e^{\beta_f} \tag{10}$$

and for each subsequent year $t_{min,f} < t_k \leq t_{max,f}$ that the fleet is operating,

$$Fmult_{f,t_k} = Fmult_{f,t_{k-1}} e^{Fdev_{f,t_k}}. \tag{11}$$

Both $Fmult_{f,t_{min,f}}$ and the vector of deviations are estimated in log space (β_f and $Fdev_{f,t}$). Note that the $Fdev$ parameters are not estimated as a bounded deviations vector in the ADMB code, and so fishing intensity can increase or decrease continually or fluctuate throughout the time series. When the weight for fishing mortality deviations is $\lambda_{Fdev_f} > 0$, a random walk in the log of $Fmult_f$ is specified for this fleet. If there are gaps in the catch series for the fleet, the random walk will skip over these years.

The directed fishing mortality rate (portion of F that contributes to landings) for a fleet, year, and age is computed using the separable equation along with the proportion of catch released for that fleet, year, and age ($PR_{f,t,a}$) as

$$Fdir_{f,t,a} = F_{f,t,a}(1 - PR_{f,t,a}). \tag{12}$$

The discard mortality rate is

$$Fdisc_{f,t,a} = F_{f,t,a}PR_{f,t,a}RM_f \tag{13}$$

where RM_f is the fleet-specific proportion of released fish that die. The two parts are then added together to produce the fishing mortality for the fleet, year, and age

$$F_{f,t,a} = Fdir_{f,t,a} + Fdisc_{f,t,a}. \tag{14}$$

If the user does not choose the option to estimate natural mortality (M), it is specified as a year by age matrix just as in ASAP version 3. If the user chooses to estimate M, initial values are specified as

$$M_{t,a} = \exp\left(\boldsymbol{X}_{M,1,t}^T \boldsymbol{\beta}_{M,1} + \boldsymbol{X}_{M,2,a}^T \boldsymbol{\beta}_{M,2}\right) \tag{15}$$

where $\boldsymbol{\beta}_{M,1}$ is a vector of m_1 coefficients with initial guesses which the user specifies, and $\boldsymbol{X}_{M,1,t}$ is the row corresponding to year t of a $Y \times m_1$ design matrix of annual covariates $\mathbf{X}_{M,1}$ which the user specifies. Similarly, $\boldsymbol{\beta}_{M,2}$ is a vector of m_2 coefficients with initial guesses which the user specifies, and $\boldsymbol{X}_{2,a}$ is the row corresponding to age a of a $A \times m_2$ design matrix of age-specific covariates $\mathbf{X}_{M,2}$ which the user specifies. Usually, the matrix $\mathbf{X}_{M,2}$ would be composed of 1s and 0s to specify different mortality rates for subsets of age classes. Phases for some or all of the coefficients $\boldsymbol{\beta}_{M,1}$ and $\boldsymbol{\beta}_{M,2}$ may be set to ≤ 0 to fix natural mortality for subsets of ages or years. For example, if there are 5 age classes, age-specific natural mortality constant over time is desired with initial guesses of 0.5,0.4,0.3,0.2,0.1, and M for age 5 is assumed known and phases for other ages is 2, then set the phases of the coefficients to 2,2,2,2,-1 with design matrix

$$\mathbf{X}_{M,2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{16}$$

and $\boldsymbol{\beta}_{M,2} = \{-0.693, -0.916, -1.204, -1.609, -2.303\}$. Also set 1 time-specific coefficient to $\boldsymbol{\beta}_{M,1} = 0$ and set the phase to -1. $\mathbf{X}_{M,1}$ must be 1 column, but the values are irrelevant.

The total mortality ($Z_{t,a}$) is the sum of natural and fishing mortality at year and age over all $\mathcal{F}$ fleets

$$Z_{t,a} = M_{t,a} + F_{t,a} = M_{t,a} + \sum_{f=1}^{\mathcal{F}} F_{f,t,a} \tag{17}$$

where $F_{f,t,a}$ is defined in Eq. 14.

2.4 Stock-Recruitment Relationship

If the user wants to include a Beverton-Holt stock-recruitment relationship, expected recruitment is parameterized following Mace and Doonan (1988) as

$$\widetilde{R}_{t+1} = \frac{4\tau_t R_{0,t} SSB_t}{SPR_{0,t} R_{0,t} + (5\tau_t - 1) SSB_t} = \frac{4\tau_t SSB_t SSB_{0,t} / SPR_{0,t}}{SSB_{0,t} + (5\tau_t - 1) SSB_t} \tag{18}$$

where $\widetilde{R}_{t+1}$ is the expected recruitment in year $t+1$ and SSB_t is defined in Eq. 25 (Section 2.5). Steepness (τ_t) and either unexploited recruitment or SSB ($R_{0,t}$ or $SSB_{0,t}$, which the user specifies) are related to each other by the unexploited SSB per recruit $SPR_{0,t} = SSB_{0,t}/R_{0,t}$. The unexploited spawning biomass per recruit is a potentially year-specific quantity calculated as

$$SPR_{0,t} = \sum_{a=1}^{A-1} \left\{ \exp\left[-\sum_{k=0}^{a-1} M_{t,k-1}\right] \Phi_{t,a} \exp\left[-p_{SSB} M_{t,a}\right] \right\} + \exp\left[-\sum_{k=0}^{A-1} M_{t,k-1}\right] \Phi_{t,A} \frac{\exp\left[-p_{SSB} M_{t,A}\right]}{1 - \exp[-M_{t,A}]} \tag{19}$$

where $M_{t,a}$ is the natural mortality rate at age a in year t with $M_{t,0} = 0$, $\Phi_{t,a}$ is the fecundity at age a in year t, and p_{SSB} is the fraction of the year at which spawning occurs. There are three options for the user on how to use $SPR_{0,t}$ in the stock-recruitment relationship: (1) use value from the first year, $SPR_{0,1}$; (2) use the value from the last year, $SPR_{0,Y}$ (used in ASAP version 3); or (3) use yearly values. The unexploited stock biomass per recruitment is therefore fixed using the first two options, but time-varying using the last option. It is also possible to model effects of annual covariates on τ and either SSB_0 or R_0 by including corresponding design matrices. Steepness is modeled as a function of a vector of n_τ coefficients $\boldsymbol{\beta}_\tau$ and a $Y \times n_\tau$ design matrix $\boldsymbol{X}_\tau$, so steepness in year t is

$$\tau_t = 0.2 + \frac{0.8}{1 + \exp\left(-\boldsymbol{X}_{\tau,t}^T \beta_\tau\right)} \tag{20}$$

where $\boldsymbol{X}_{\tau,t}$ is the row of $\mathbf{X}_\tau$ corresponding to year t. Similarly, either R_0 or SSB_0, are also modeled as a function of a vector of n_0 coefficients $\boldsymbol{\beta}_0$ and a $Y \times n_0$ design matrix $\boldsymbol{X}_0$,

$$R_{0,t} = \exp\left(\boldsymbol{X}_{0,t}^T \beta_{\mathbf{0}}\right) \tag{21}$$

or

$$SSB_{0,t} = \exp\left(\mathbf{X}_{0,t}^T \beta_{\mathbf{0}}\right) \tag{22}$$

where $\boldsymbol{X}_{0,t}$ is the row corresponding to year t of $\mathbf{X}_0$. The default for steepness and unexploited recruitment or spawning biomass is to use a single column of ones for the design matrices so that the parameters are constant over time. For example, the user can specify a constant value of $R_0 = 10^6$ with the design matrix

$$\mathbf{X}_0 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{23}$$

and starting value $\beta_0 = 13.81551$ which gives the vector of unexploited recruitment

$$\boldsymbol{R}_0(\mathbf{X}_0) = \exp\left(\mathbf{X}_0 \beta_{\mathbf{0}}\right) = \begin{bmatrix} 10^6 \\ 10^6 \\ \vdots \\ 10^6 \end{bmatrix}. \tag{24}$$

If the user specifies to not use the stock-recruit relationship to model recruitment or if steepness is fixed at 1 with no covariate effects on R_0 and using SPR_0 from either year 1 or Y, then expected recruitment will be constant, $\widetilde{R}_t = R_0$. Regardless of the what the user specifies, the program produces the annual values of SSB_0, R_0, SPR_0, and τ, so that the user may inspect trends or variation in the these values over time and any influence on other estimated quantities.

2.5 Spawning Stock Biomass

The spawning stock biomass in year t is a function of the population abundance which occurred at age $N_{t,a}$, the fecundity at age $\Phi_{t,a}$, and the total mortality at age $Z_{t,a}$ (see Section 2.3) which occurs during the fraction of the year prior to spawning p_{SSB},

$$SSB_t = \sum_{a=1}^{A} N_{t,a} \Phi_{t,a} \exp\left\{-p_{SSB} Z_{t,a}\right\} \tag{25}$$

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

2.6 Abundance at Age

For predicted recruitment $N_{t,1}$, the parameters estimated in the model are the log-recruitment deviations,

$$Rdev_t = \log(N_{t,1}) - \log(\widetilde{R}_t) \tag{26}$$

which are elements of a bounded deviations vector (sums to zero) and predicted recruitment is calculated as

$$N_{t,1} = \widetilde{R}_t e^{Rdev_t} \tag{27}$$

where $\widetilde{R}_t$ is generated using methods described in Section 2.4.

The user provides initial values for the population abundance at ages 2 through A in the first year (The initial value provided for age 1 is not used). If the phase for these parameters is > 0 then the model will estimate them, otherwise they will be fixed at the initial values. If the user specifies to use the stock-recruit relationship to model recruits, a partial SSB for ages 2 through the maximum age is computed and used in the stock recruitment relationship (Eq. 18) to create an expected recruitment in the first year $\widetilde{R}_1$. Otherwise, $\widetilde{R}_1 = R_0$. In either case, the predicted recruitment $N_{1,1}$ is specified using Eq. 27, and SSB_1 is then completed using Eq. 25 for the first age class.

For each subsequent year, abundances for age classes 2 to $A - 1$ are

$$N_{t,a} = N_{t-1,a-1} e^{-Z_{t-1,a-1}}, \tag{28}$$

the plus group abundance is

$$N_{t,A} = N_{t-1,A-1} e^{-Z_{t-1,A-1}} + N_{t-1,A} e^{-Z_{t-1,A}}, \tag{29}$$

and the spawning stock biomass is computed (Eq. 25) so that $\widetilde{R}_{t+1}$ can be computed if the stock-recruit function is used.

2.7 Predicted Landings, Discards and Proportions at Age

The predicted numbers of fish landed $\widehat{L}_{f,t,a}$ and discarded $\widehat{D}_{f,t,a}$ at age a in year t for fleet f are derived from the Baranov catch equation:

$$\widehat{L}_{f,t,a} = N_{t,a} \frac{Fdir_{f,t,a}}{Z_{t,a}} \left(1 - e^{-Z_{t,a}}\right) \tag{30}$$

and

$$\widehat{D}_{f,t,a} = N_{t,a} \frac{Fdisc_{f,t,a}}{Z_{t,a}} \left(1 - e^{-Z_{t,a}}\right). \tag{31}$$

These predictions are components of predicted total weight of landings and discards and respective proportions at age. The predicted total landings in weight is a function of $\widehat{L}_{f,t,a}$ and the weight at age for landed fish in the fleet $W_{L,f,t,a}$,

$$\widehat{L}_{W,f,t} = \sum_{a=1}^{A} \widehat{L}_{f,t,a} W_{L,f,t,a}. \tag{32}$$

Similarly, the total discards in weight is a function of $\widehat{D}_{f,t,a}$ and the weight at age for discarded fish in the fleet $W_{D,f,t,a}$,

$$\widehat{D}_{W,f,t} = \sum_{a=1}^{A} \widehat{D}_{f,t,a} W_{D,f,t,a} \tag{33}$$

Note that different weights at age can be specified for landings and discards for each fleet. Since $Fdisc_{f,t,a}$ is derived using the proportion of fish that die after release, the total observed discards in weight ($\widehat{D}_{f,t}$) should only include those fish that die after capture and release.

The predicted landings and discarded proportions at age for each fleet and year are

$$\widehat{p}_{L,f,t,a} = \frac{\widehat{L}_{f,t,a}}{\sum_{a=1}^{A} \widehat{L}_{f,t,a}} \tag{34}$$

and

$$\widehat{p}_{D,f,t,a} = \frac{\widehat{D}_{f,t,a}}{\sum_{a=1}^{A} \widehat{D}_{f,t,a}}. \tag{35}$$

Any predicted proportion less than 10^{-15} is replaced by the value 10^{-15} to avoid division by zero in the objective function.

2.8 Calibrated Abundance Indices and Age Composition

We provided this new feature in ASAP version 4 to allow users to include estimates of relative catch efficiency (calibration coefficients), potentially by length, more directly in an assessment. For example, the length-based relative catch efficiency of the Henry B. Bigelow and Albatross IV estimated by Miller (2013) for Atlantic butterfish was used in its most recent assessment (Adams et al. 2015). However, the user can specify multiple (length-based) relative catch efficiencies that can be used to calibrate specific portions of abundance index time series. The input data for each abundance index observation to be calibrated includes the uncalibrated numbers at length for each of L length classes and the age-length key for year t. The input data for relative catch efficiency k includes the p_k coefficient estimates $\widetilde{\boldsymbol{\beta}}_k$, the corresponding $p_k \times p_k$ variance-covariance matrix $\boldsymbol{\Sigma}_k$, and the design matrix $\mathbf{X}_k$ ($L \times p_k$) for calculating the relative catch efficiency at length. Calibrated abundance index i in year t is

$$I_{C,i,t} = \sum_{l=1}^{L} I_{U,i,t,l}\rho_{k,l} \tag{36}$$

where $I_{U,i,t,l}$ is the uncalibrated numbers-at length l,

$$\rho_{k,l} = e^{-\boldsymbol{X}_{k,l}^T \widehat{\boldsymbol{\beta}}_k} \tag{37}$$

is the relative catch efficiency at length l, $\boldsymbol{X}_{k,l}$ is the row of the design matrix corresponding to length class l, and $\widehat{\boldsymbol{\beta}}_k = \boldsymbol{REdev}_k + \widetilde{\boldsymbol{\beta}}_k$. The vector of p_k deviations $\boldsymbol{REdev}_k$ are parameters that are initially 0, but estimated when the phase for relative catch efficiency k is > 0. The deviations allow the calibration to depart from that provided using just the input coefficient $\widetilde{\boldsymbol{\beta}}_k$ when the other data components in the assessment model provide information about the change in efficiency. When the deviations are estimated they are penalized by an objective function component described in Section 3.2.11 which uses the variance-covariance matrix $\boldsymbol{\Sigma}_k$. The more precise the estimates of the coefficients, the less the deviations will be allowed to differ from 0. This approach to dealing with changes in efficiency for a given abundance index time series can be thought of as an intermediate between external calibration of the index observations and splitting the series with separate catchabilities and selectivities.

The calibrated number at age a is

$$I_{C,i,t,a} = \sum_{l=1}^{L} p_{i,t}(a|l) I_{U,i,t,l}\rho_{l,k} \tag{38}$$

where $p_{i,t}(a|l)$ is the proportion at age a given length l from the age length key for abundance index i in year t and the calibrated proportion at age a is

$$p_{C,i,t,a} = \frac{I_{C,i,t,a}}{\sum_{a=1}^{A} I_{C,i,t,a}} \tag{39}$$

Note that this option will only work correctly for numbers-based (not biomass) indices and age composition. The calibrated abundance indices $I_{C,i,t}$ are used in the calculations of abundance index objective function components using the CVs supplied with the abundance index series. Therefore, we implicitly assume that the CVs of the abundance indices and effective sample sizes for the proportions-at-age do not depend on the gear used to collect the abundance index data. The calibrated abundance indices and proportions at age also replace the normal abundance index data for the calibrated years in the reported results. Note also, there will be p_k more parameters estimated for each relative catch efficiency k for which the phase is 0.

2.9 Predicted Abundance Indices and Proportions at Age

Proper predictions of the abundance indices depend on correct specification of the time of year when the abundance index data are collected and the units of measure of the abundance indices (numbers or biomass). If the month for abundance index in year t is set to $m_{i,t} = -1$, the population numbers at age available to the survey are assumed to be the average annual abundance at age,

$$N_{i,t,a} = \frac{N_{t,a}}{Z_{t,a}}\left(1 - e^{-Z_{t,a}}\right). \tag{40}$$

If the month is $1 \leq m_{i,t} < 13$ the numbers at age are decremented based on the time of year when index occurs

$$N_{i,t,a} = N_{t,a}e^{-Z_{t,a}\delta_{i,t}} \tag{41}$$

where $\delta_{i,t} = (m_{i,t} - 1)/12$. Note that the time of the observation refers to the beginning of the month specified, so $m_{i,t} = 1$ is January 1 and $m_{i,t} = 12.5$ is December 15. If the abundance index is measured in numbers, the predicted abundance index $(\widehat{I}_{i,t})$ is

$$\widehat{I}_{i,t} = \sum_{a=1}^{A}\widehat{I}_{i,t,a} = q_{i,t}\sum_{a=1}^{A}N_{i,t,a}s_{i,t,a} \tag{42}$$

and if the abundance index is measured in biomass, then

$$\widehat{I}_{i,t} = \sum_{a=1}^{A}\widehat{I}_{i,t,a} = q_{i,t}\sum_{a=1}^{A}N_{i,t,a}W_{i,t,a}s_{i,t,a} \tag{43}$$

where $W_{i,t,a}$ are the user-defined weights at age for abundance index i. If the user selects to estimate the proportions at age for an abundance index, then the proportions at age are computed in the same manner as the landings and discards at age (Eqs. 34 and 35),

$$\widehat{p}_{i,t,a} = \frac{\widehat{I}_{i,t,a}}{\widehat{I}_{i,t}} \tag{44}$$

Note that the user specifies the unit of measure for the abundance index and proportions at age separately, so all four combinations of numbers and biomass are possible.

2.10 Reported Fishing Mortality

A feature of ASAP version 3 that is continued in ASAP version 4 is the use of a reported fishing mortality *Frep*, which averages the total fishing mortality over an input range of ages, a_{min} to a_{max}. The calculation of $Frep_i$ n a given year is done with 1 of 3 different types of weighting that the user chooses: equal weighting ($\omega_{t,a} = 1$), weighting by population abundance at age ($\omega_{t,a} = N_{t,a}$), or weighting by population biomass at age ($\omega_{t,a} = N_{t,a}W_{t,a}$ where $W_{t,a}$ denotes the January 1 weight at age a in year t). The weighted average is

$$Frep_t = \frac{\sum_{a=a_{min}}^{a_{max}}\omega_{t,a}F_{t,a}}{\sum_{a=a_{min}}^{a_{max}}\omega_{t,a}} \tag{45}$$

where $F_{t,a}$ is defined in Eq. 17.

2.11 Reference Points

As in ASAP version 3, there are a number of common reference points based on the estimated fishing mortality at age and biological characteristics in the model. The reference points are based on directed and discard selectivity at age from all the fleets that were assigned to be directed. The directed selectivity at age is the ratio of total directed fishing mortality at age to the maximum of the age-specific values

$$sdir_{t,a} = \frac{\sum_{f=1}^{\mathcal{F}}Fdir_{f,t,a}}{\max_a\left(\sum_{f=1}^{\mathcal{F}}Fdir_{f,t,a}\right)}. \tag{46}$$

The non-directed selectivity at age is obtained analogously from fishing mortality at age of fleets that were not assigned as directed. These selectivities are fixed during the reference point calculations. The fishing mortality reference points are computed for each year through a bisection algorithm that is repeated 20 times (producing an accuracy of approximately 10^{-5}). The reference points computed are $F_{0.1}$, F_{MAX}, F_{MSY}, and $F_{X\%}$ where the user specifies any number of values between 0 and 100 for the percentage of spawning potential ratio. The associated maximum sustainable yield and spawning stock biomass at F_{MSY} are also provided. The annual reference point values are averaged in the same manner as *Frep* to allow direct comparison. If selectivity or biological characteristics change over time, care must be taken in interpreting the reference points and for the MSY-based reference points, the option chosen for usage of SPR_0 in the stock-recruit relationship is very important. The program computes annual values using year-specific natural mortality, weights at age, fecundity, and selectivity to demonstrate the potential for change in the reference points.

2.12 Projections

The projections in year beyond the terminal year of the model use the same basic calculations, except that there are no data to which the estimates are fitted. The recruitments for each projection year can either be provided in the input data or be derived from the stock recruitment curve (without deviations from the curve). The directed and discard selectivity as well as the non-directed F at age are the same as used in the reference point calculations. There are five options to define harvesting in each projections year:

- match an input directed catch in weight
- fish at an input $F_{X\%}$
- fish at F_{MSY}
- fish at the current (terminal year) *Frep*
- fish at an input *Frep*

Each year the non-directed F can be modified from the terminal year to examine either increases or decreases in the fishery.

3 OBJECTIVE FUNCTION COMPONENTS

The objective function in ASAP version 4 is the sum of components for abundance indices, catch, discards, any respective age composition data, and any of a number of penalties. The components are the negative log of probability distributions, and the objective function is minimized using AD Model Builder (Fournier et al. 2012).

3.1 Data

The logarithm of observations for aggregate catch and abundance index data component d are treated as normally distributed,

$$f(X_{d,t}|\mu_{d,t},\sigma_{d,t}) = \frac{1}{\sqrt{2\pi}\sigma_{d,t}} \exp\left\{-\frac{1}{2\sigma^2_{d,t}}\left[\log(X_{d,t}) - \log(\mu_{d,t})\right]^2\right\} \tag{47}$$

where $X_{d,t}$ is the observed catch or abundance index in year t, $\mu_{d,t}$ is the predicted value for the observation (see Sections 2.7 and 2.9), and $\sigma_{d,t} = \sqrt{\log\left(CV_{d,t}^2 + 1\right)}$ where the user specifies $CV_{d,t}$ in the input. The negative logarithm of Eq. 47 multiplied by a weight λ_d

$$-\lambda_d \log\left[f(X_{d,t}|\mu_{d,t}, \sigma_{d,t})\right] = \lambda_d \left\{ \log(\sigma_{d,t}) + \frac{1}{2}\left\{ \log(2\pi) + \frac{\left[\log(X_{d,t}) - \log(\mu_{d,t})\right]^2}{\sigma_{d,t}^2} \right\}\right\} \tag{48}$$

is added to the objective function. The user specifies the weights and allows emphasis of one or more component of the objective function. Components can have no influence on the objective function by setting $\lambda_d = 0$.

Age composition observations for any fleet or abundance index d are treated as multinomially distributed,

$$\begin{aligned} f(p_{d,t,1}, \dots, p_{d,t,A}|\mu_{d,t,1}, \dots, \mu_{d,t,A}, ESS_{d,t}) &= \frac{ESS_{d,t}!}{\prod_{a=1}^{A}(ESS_{d,t}p_{d,t,a})!} \prod_{a=1}^{A} \mu_{d,t,a}^{ESS_{d,t}p_{d,t,a}} \\ &= \frac{\Gamma\left(ESS_{d,t} + 1\right)}{\prod_{a=1}^{A} \Gamma\left(ESS_{d,t}p_{d,t,a} + 1\right)} \prod_{a=1}^{A} \mu_{d,t,a}^{ESS_{d,t}p_{d,t,a}} \end{aligned} \tag{49}$$

where $ESS_{d,t}$ is the effective sample size which the user specifies, $p_{d,t,a}$ and $\mu_{d,t,a}$ are the observed and predicted proportion at age (see Sections 2.7 and 2.9), and $\Gamma(\cdot)$ is the gamma function. Similar to the aggregate observations the negative logarithm of Eq. 49 is added to the likelihood

$$\begin{aligned} &-\log\left[f(p_{d,t,1}, \dots, p_{d,t,A}|\mu_{d,t,1}, \dots, \mu_{d,t,A}, ESS_{d,t})\right] = \\ &\quad \sum_{a=1}^{A}\left\{\log\left[\Gamma\left(ESS_{d,t}p_{d,t,a} + 1\right)\right] - \left(ESS_{d,t}p_{d,t,a}\right)\log\left(\mu_{d,t,a}\right)\right\} - \log\left[\Gamma\left(ESS_{d,t} + 1\right)\right]. \end{aligned} \tag{50}$$

There are no weighting multipliers provided for the age compositiond data, but the emphasizing particular components can be achieved by increasing the effective sample sizes.

3.2 Penalties

Penalties are components of the objective function that allow the user to constrain how much a parameter deviates from some value. Most of the penalties constrain deviation from the initial values of the respctive parameters which the user provides, but in some cases the penalties are for deviation from some other expected value derived from other parameters (e.g., expected recruitment from the stock-recruitment relationship). The parameters that can be penalized for deviating from initial values are

- stock-recruitment function steepness value,
- stock-recruitment function scalar (R_0 or SSB_0) value,
- year 1 numbers at age,
- year 1 fully selected fishing mortality,
- selectivity parameters,
- relative catch efficiency coefficients,
- abundance index availability, and
- abundance index gear efficiency.

There are also penalites available for deviation of estimated recruitment from that expected from the stock-recruitment relationship, and to constrain inter-annual variability of fully selected fishing mortality and availability through autoregressive objective function components. Finally, there are penalties the user can specify to stabilize estimation in early phases of the minimization and for ensuring estimated fishing mortality is below a maximum value.

There are three distributions used for nearly all penalties in the objective function depending on the range of the parameter. For a strictly positive parameter θ, a normal distribution on the log-transformed parameter is used

$$f\left(\theta|\widetilde{\theta}\right) = \frac{1}{\sqrt{2\pi}\sigma_\theta} \exp\left\{-\frac{1}{2\sigma_\theta^2}\left[\log(\theta) - \log\left(\widetilde{\theta}\right)\right]^2\right\} \tag{51}$$

where θ is the estimated parameter or average of annual estimated parameters and $\widetilde{\theta}$ is usually the initial value of the parameter, but is specified for each penalty in subsections below. The standard deviation on log-scale σ_θ is derived from the user-provided CV, $\sigma_\theta = \sqrt{\log\left(CV_\theta^2 + 1\right)}$. Similar to data components, there is a weight λ_θ which the user specifies that can be used to adjust the emphasis of particular penalties. The product of the weight and the negative of the logarithm of the probability distribution function

$$-\lambda_\theta \log\left[f\left(\theta|\widetilde{\theta}\right)\right] = \lambda_\theta\left\{\log(\sigma_\theta) + \frac{1}{2}\left\{\log(2\pi) + \frac{\left[\log(\theta) - \log\left(\widetilde{\theta}\right)\right]^2}{\sigma_\theta^2}\right\}\right\} \tag{52}$$

is added to the objective function.

For parameters with lower and upper bounds, l_θ and u_θ, there is an option to use a truncated normal distribution

$$f\left(\theta|\widetilde{\theta}\right) = \frac{1}{\sqrt{2\pi}\sigma_\theta} \frac{\exp\left\{-\frac{1}{2\sigma_\theta^2}\left[\log(\theta) - \log\left(\widetilde{\theta}\right)\right]^2\right\}}{\Phi\left[\frac{\log(u_\theta) - \log\left(\widetilde{\theta}\right)}{\sigma_\theta}\right] - \Phi\left[\frac{\log(l_\theta) - \log\left(\widetilde{\theta}\right))}{\sigma_\theta}\right]} \tag{53}$$

where $\Phi(\cdot)$ is the cumulative standard normal distribution (the default) or a 4-parameter beta distribution

$$f\left(\theta|\widetilde{\theta}\right) = \frac{\Gamma(\phi_\theta)}{\Gamma(\phi_\theta\mu_\theta)\Gamma(\phi_\theta(1-\mu_\theta))} \frac{(\theta - l_\theta)^{\phi_\theta\mu_\theta - 1}(u_\theta - \theta)^{\phi_\theta(1-\mu_\theta)-1}}{(u_\theta - l_\theta)^{\phi_\theta - 1}} \tag{54}$$

where

$$\mu_\theta = \frac{\widetilde{\theta} - l_\theta}{u_\theta - l_\theta} \tag{55}$$

and ϕ_θ is a variance parameter. The variance for the 4-parameter beta distribution is

$$V\left(\theta|\widetilde{\theta}\right) = \widetilde{\theta}^2 CV_\theta^2 = (u_\theta - l_\theta)^2 \frac{\mu_\theta(1-\mu_\theta)}{\phi_\theta + 1} \tag{56}$$

so that

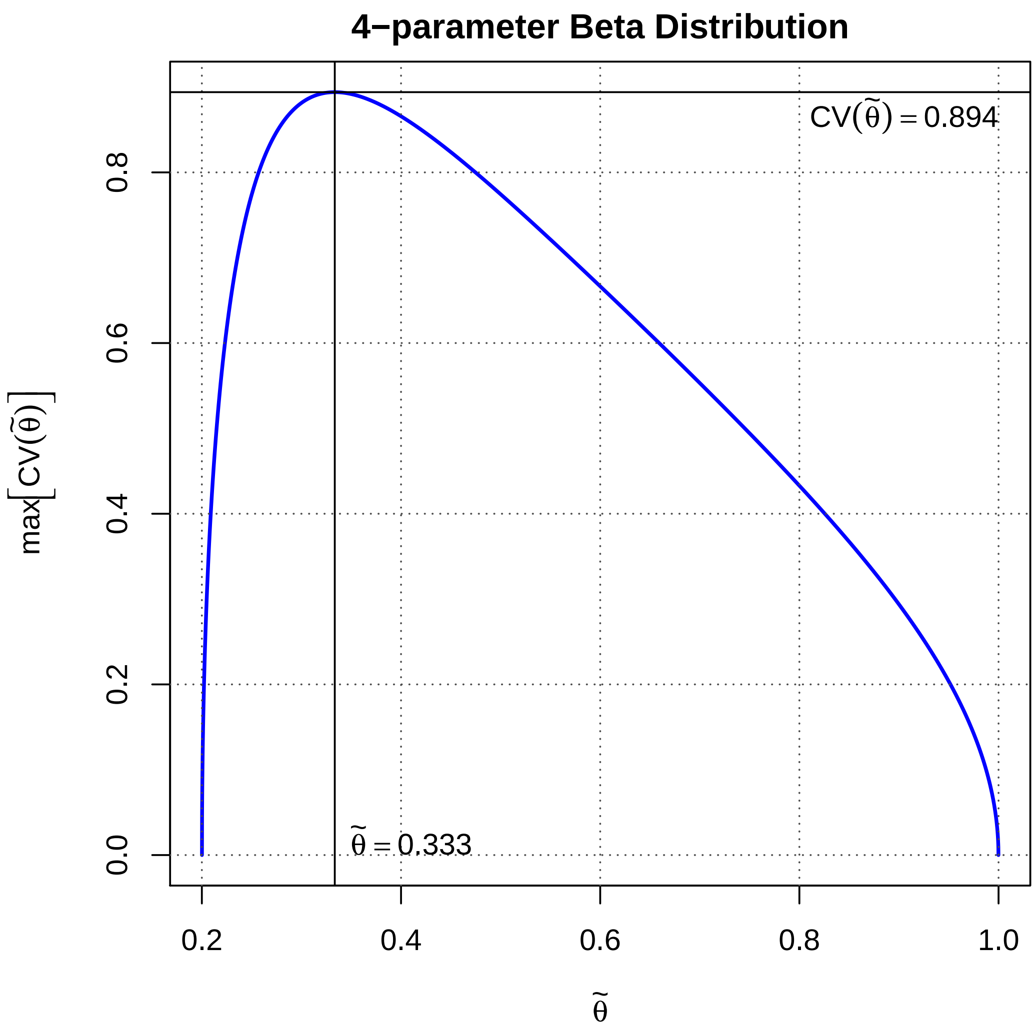
$$\phi_\theta = (u_\theta - l_\theta)^2 \frac{\mu_\theta(1-\mu_\theta)}{\widetilde{\theta}^2 CV_\theta^2} - 1 \tag{57}$$

where CV_θ is specified by the user in the input. Since $\phi_\theta > 0$,

$$0 < CV_\theta < \frac{\sqrt{\left(u_\theta - \widetilde{\theta}\right)\left(\widetilde{\theta} - l_\theta\right)}}{\widetilde{\theta}} \tag{58}$$

so if the user specifies a value larger than the maximum possible CV, the maximum is used instead (Figure 1).

Figure 1. The maximum CV for the 4-parameter beta distribution is a function of the initial parameter value and the lower and upper bounds of the parameter. For example when the Beverton-Holt steepness parameter, which is bounded by 0.2 and 1.0, is penalized using this distribution the maximum CV is greatest when the initial value is 0.333.



Similar to Eq. 52, the objective function component for a parameter with a truncated normal penalty is

$$
\begin{aligned}
-\lambda_\theta \log\left[f\left(\theta|\widetilde{\theta}\right)\right] =& \lambda_\theta \left\{ \log(\sigma_\theta) + \frac{1}{2}\left\{ \log(2\pi) + \frac{\left[\log(\theta) - \log\left(\widetilde{\theta}\right)\right]^2}{\sigma_\theta^2} \right\} \right. \\
& \left. + \log\left\{ \Phi\left[\frac{\log(u_\theta) - \log\left(\widetilde{\theta}\right)}{\sigma_\theta}\right] - \Phi\left[\frac{\log(l_\theta) - \log\left(\widetilde{\theta}\right)}{\sigma_\theta}\right] \right\} \right\}
\end{aligned} \tag{59}
$$

and for a parameter with a the 4-parameter beta penalty is

$$
\begin{aligned}
-\lambda_\theta \log\left[f\left(\theta|\widetilde{\theta}\right)\right] =& \lambda_\theta \left\{ \log\left[\Gamma(\phi_\theta \mu_\theta)\right] + \log\left[\Gamma(\phi_\theta(1-\mu_\theta))\right] - \log\left[\Gamma(\phi_\theta)\right] + (\phi_\theta - 1)\log(u_\theta - l_\theta) \right. \\
& \left. - (\phi_\theta \mu_\theta - 1)\log(\theta - l_\theta) - (\phi_\theta(1-\mu_\theta) - 1)\log(u_\theta - \theta) \right\}.
\end{aligned} \tag{60}
$$

3.2.1 Stock-Recruitment Relationship

When the weight for the recruitment penalty is $\lambda_R > 0$, Eq. 52 is added to the objective function for each annual estimated recruitment where $\widetilde{\theta} = \widetilde{R}_t$ and $\theta = N_{t,1}$ as defined in Sections 2.4 and 2.6.

3.2.2 Stock-Recruitment Scalar Parameter (R_0 or SSB_0)

When the weight for the stock-recruitment scalar ($SR = R_0$ or $SR = SSB_0$) penalty is $\lambda_{SR} > 0$, Eq. 52 is added to the objective function where $\widetilde{\theta} = \widetilde{SR}$ is the initial value which the user specifies, and $\theta = \overline{SR}$ is the average of the annual estimates which may vary if annual SPR_0 is specified or covariate effects are modeled (see Section 2.4).

3.2.3 Stock-Recruitment Steepness

When the weight for the steepness penalty is $\lambda_\tau > 0$, either Eq. 59 or Eq. 60 is added to the objective function where $\widetilde{\theta} = \widetilde{\tau}$ is the initial value provided by the user, and $\theta = \overline{\tau}$ is the average of the annual estimates which may vary if covariate effects are modeled (see Section 2.4).

3.2.4 Year 1 Abundance Parameters

When the weight for the year 1 numbers at age penalty is $\lambda_N > 0$, Eq. 52 is added to the objective function for each age class where $\theta = N_{1,a}$, and $\widetilde{\theta} = \widetilde{N}_{1,a}$ is either the initial value which the user specifies or determined by mortality rates in the first year.

$$
\widetilde{N}_{1,a} = \widetilde{N}_{1,1} \exp\left(-\sum_{k=1}^{a-1} Z_{1,k}\right) \tag{61}
$$

for ages 2 through $A-1$ and

$$
\widetilde{N}_{1,A} = \widetilde{N}_{1,1} \frac{\exp\left(-\sum_{k=1}^{A} Z_{1,k}\right)}{1 - \exp\left(-Z_{1,A}\right)}. \tag{62}
$$

Note that all ages are included in the penalty whereas in ASAP version 3 the penalty only included age classes $2, \ldots, A$, and if $\lambda_R > 0$ and $\lambda_N > 0$, $N_{1,1}$ also occurs in the stock-recruitment relationship penalty (see Section 3.2.1).

3.2.5 Year 1 Fishing Mortality Rate

When the weight for the penalty on year 1 fishing mortality for fleet f is $\lambda_{Fmult_{f,1}} > 0$, Eq. 52 is added to the objective function where $\widetilde{\theta} = \widetilde{Fmult}_{f,1}$ is the initial value provided by the user, and $\theta = Fmult_{f,1}$ is defined in Section 2.3.

3.2.6 Selectivity Parameters

When the weight of the penalty for selectivity parameter v, is $\lambda_{s_v} > 0$, either Eq. 59 or Eq. 60 is added to the objective function where $\widetilde{\theta} = \widetilde{s}_v$ is the initial value which the user specifies, and $\theta = s_v$ is the selectivity parameter (see Section 2.1).

3.2.7 Availability of Population to Abundance Indices

When the weight for the penalty on the availability component of catchability for abundance index i is $\lambda_{Avail,i} > 0$, either Eq. 59 or Eq. 60 is added to the objective function where $\widetilde{\theta} = \widetilde{Avail}_i$ is the initial value which the user specifies, and $\theta = \overline{Avail}_i$ is the average of the annual estimates which may vary if covariate effects are modeled. This would be a penalty on catchability if the user specifies the the attributes of the gear efficiency component properly. See Section 2.2 for more details.

3.2.8 Gear Efficiency of Abundance Indices

When the weight for the penalty on the gear efficiency component of catchability for abundance index i is $\lambda_{Eff,i} > 0$, either Eq. 59 or Eq. 60 is added to the objective function where $\widetilde{\theta} = \widetilde{Eff}_i$ is the initial value which the user specifies, and $\theta = \overline{Eff}_i$ is the average of the annual estimates which may vary if covariate effects are modeled (see Section 2.2).

3.2.9 Fishing Mortality Random Walk

When the weight for the penalty on fishing mortality deviations for fleet f is $\lambda_{Fdev_f} > 0$, Eq. 52 is added to the objective function for each annual devation where $\widetilde{\theta} = 0$, and $\theta = Fdev_{f,t_k}$ for $t_{min,f} < t_k \leq t_{max,f}$ (see Section 2.3).

3.2.10 AR process for Availability/Catchability of Abundance Indices

When the weight for the penalty on the deviations of the availability component of catchability for abundance index i is $\lambda_{Adev_i} > 0$, Eq. 52 is added to the objective function for each annual devation where $\widetilde{\theta} = 0$ and $\theta = Adev_{i,t_k}$ for $t_{min,i} < t_k \leq t_{max,i}$ (see Section 2.2).

3.2.11 Relative Catch Efficiency Coefficiencts

When the weight for the penalty on relative catch efficiency k is $\lambda_{RE,k} > 0$, the product of $\lambda_{RE,k}$ and the negative log of a multivariate normal penalty

$$f(\widehat{\boldsymbol{\beta}}_k|\widetilde{\boldsymbol{\beta}}_k) = (2\pi)^{-\frac{p_k}{2}}|\boldsymbol{\Sigma}_k|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\widehat{\boldsymbol{\beta}}_k - \widetilde{\boldsymbol{\beta}}_k)^T \boldsymbol{\Sigma}_k^{-1}(\widehat{\boldsymbol{\beta}}_k - \widetilde{\boldsymbol{\beta}}_k)\right] \tag{63}$$

is added to the objective function where $\widetilde{\boldsymbol{\beta}}_k$ are the input estimates of the p_k relative catch efficiency coefficients, $\boldsymbol{\Sigma}_k$ is the estimated variance-covariance matrix for the coefficients and $\widehat{\boldsymbol{\beta}}_k$ are the estimates within the assessment model. Note that the differences $\widehat{\boldsymbol{\beta}}_k - \widetilde{\boldsymbol{\beta}}_k = \boldsymbol{REdev}_k$ are the parameters actually estimated (see Section 2.8).

3.2.12 Deviations between F and M

This penalty which helps stabilize estimation in early phases was used in ASAP version 3 and is an option to the user in ASAP version 4. When specified and the current phase is not the final phase of estimation, the negative log of the penalty

$$f(\overline{F}, \overline{M}) = \exp\left\{-\lambda_{F,M}\left[\log\left(\overline{F}\right) - \log\left(\overline{M}\right)\right]^2\right\} \tag{64}$$

is added to the objective function where $\overline{F}$ and $\overline{M}$ are the average total fishing and natural mortalities over all ages and years of the model and

$$\lambda_{F,M} = 10^{2-\text{PHASE}} \tag{65}$$

where PHASE is the current phase of estimation. When there are multiple phases, the influence of the penalty decreases as the phase increases.

3.2.13 Maximum F

This penalty was also used in ASAP version 3 and is an option to the user in ASAP version 4. When specified and any of the fishing mortalities at age for fleet f in year t is greater than the maximum fishing mortality MAX_F (which the user specifies), the negative log of the penalty,

$$f(\max(F_{f,t,a})) = \exp\left[-1000\left(\max(F_{f,t,a}) - \text{MAX}_F\right)^2\right] \tag{66}$$

is added to the objective function.

4 STANDARDIZED RESIDUALS, RMSE, AND EFFECTIVE SAMPLE SIZE

For any log-normally distributed observation x the standardized residual is calculated as

$$Res = \frac{\log(x) - \log(\widetilde{x})}{\sigma} \tag{67}$$

where $\widetilde{x}$ is the predicted value and σ is the standard deviation of the log-observation or parameter which is a function of the CV that the user specifies in the input as described previously in Section 3.1. For log-normal or truncated log-normal parameter penalties, a standardized residual is provided where $\widetilde{x} = \widetilde{\theta}$, $x = \theta$, and $\sigma = \sigma_\theta$ are defined in Section 3.2. For penalties that use the 4-parameter beta distribution, standardized residuals are calculated as

$$Res = \frac{p_\theta - \mu_\theta}{\sqrt{V(\mu_\theta)}} \tag{68}$$

where

$$p_\theta = \frac{\theta - l_\theta}{u_\theta - l_\theta}, \tag{69}$$

$$V(\mu_\theta) = \frac{\mu_\theta(1 - \mu_\theta)}{\phi_\theta + 1} \tag{70}$$

and μ_θ, ϕ_θ, l_θ, and u_θ are defined in Section 3.2.

The root mean-squared error for data component or parameter penalty d is calculated as

$$RMSE_d = \sqrt{\frac{1}{n_d}\sum_{t=1}^{n_d} Res_{d,t}^2} \tag{71}$$

where n_d is the number of annual observations. For parameter penalties $n_d = 1$.

There are two types of effective sample sizes calculated for the age composition data components. The first uses the method described by McAllister and Ianelli (1997). For each yearly age composition data component, an estimated effective sample size is calculated as

$$\widehat{ESS}_{1,d,t} = \frac{\sum_{a=1}^{A} \widehat{p}_{d,t,a}(1 - \widehat{p}_{d,t,a})}{\sum_{a=1}^{A} (p_{d,t,a} - \widehat{p}_{d,t,a})^2}. \tag{72}$$

The second uses a method (TA1.8) described by Francis (2011),

$$\widehat{ESS}_{2,d,t} = n_{d,t} \frac{n_d - 1}{\sum_{t=1}^{n_d} (W_{d,t} - \overline{W}_d)^2} \tag{73}$$

where n_d is the number of years of age composition data in component d (e.g., a fleet or abundance index),

$$W_{d,t} = \frac{R_{d,t}\sqrt{n_{d,t}}}{S_{d,t}}, \tag{74}$$

$$R_{d,t} = \sum_{a=1}^{A} a(p_{d,t,a} - \widehat{p}_{d,t,a}), \tag{75}$$

$$S_{d,t} = \sqrt{\sum_{a=1}^{A} a^2 \widehat{p}_{d,t,a} - \left(\sum_{a=1}^{A} a\widehat{p}_{d,t,a} \right)^2}, \tag{76}$$

and

$$\overline{W}_d = \frac{1}{n_d} \sum_{t=1}^{n_d} W_{d,t} \tag{77}$$

For either method, the predicted proportion $\widehat{p}_{d,t,a}$ is given by Eqs. 34, 35, or 44, depending on the data component, and $p_{d,t,a}$ is the corresponding observation.

5 Bibliography

Adams CF, Miller TJ, Manderson JP, Richardson DE, Smith BE. 2015. Atlantic butterfish 2014 stock assessment. NEFSC Ref. Doc. 15-06; 110 p. Available from: National Marine Fisheries Service, 166 Water Street, Woods Hole, MA 02543-1026, or online at http://www.nefsc.noaa.gov/publications/

Fournier DA, Skaug HJ, Ancheta J, Ianelli J, Magnusson A, Maunder M, Nielsen A, Sibert J. 2012. AD Model Builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models. Optim Method Softw. **27**(2): 233–249.

Francis RICC. 2011. Data weighting in statistical fisheries stock assessment models. Can J Fish Aquat Sci. **68**(6): 1124–1138.

Legault CM, Restrepo VR. 1998. A flexible forward age-structured assessment program. ICCAT Working Document SCRS/98/58. 15p.

Mace PM, Doonan IJ. 1988. A generalised bioeconomic simulation model for fish population dynamics. New Zealand Fishery Assessment Research Document 88/4. Fisheries Research Centre, MAFFish, POB 297, Wellington, NZ.

McAllister MK, Ianelli JN. 1997. Bayesian stock assessment using catch-age data and the sampling-importance resampling algorithm. Can J Fish Aquat Sci. **54**(2): 284–300.

Miller TJ. 2013. A comparison of hierarchical models for relative catch efficiency based on paired-gear data for U.S. Northwest Atlantic fish stocks. Can J Fish Aquat Sci. **70**(9): 1306–1316.

Table 1. Definitions of all notation used in equations throughout this document.

a	a particular age class
A	number of age classes
t	a particular year
Y	number of years in the model
f	a particular fleet
$\mathcal{F}$	number of fleets in the model
i	a particular abundance index
u	a particular fleet or abundance index
k	a particular relative catch efficiency
l	a particular length class
L	number of length classes
v	a particular selectivity parameter
s_a	selectivity at age a
a_{50}	age at the inflection point of a logistic selectivity ogive
b	inverse of the slope parameter of a logistic selectivity ogive
$a_{50,1}$	age at the inflection point of the ascending limb in a double-logistic selectivity ogive
b_1	inverse of the slope parameter of the ascending limb in a double-logistic selectivity ogive
$a_{50,2}$	age at the inflection point of the decending limb in a double-logistic selectivity ogive
b_2	inverse of the slope parameter of the decending limb in a double-logistic selectivity ogive
$q_{i,t}$	catchability of abundance index i in year t
$t_{min,i}$	first year that abundance index i is observed
$t_{max,i}$	last year that abundance index i is observed
$Avail_{i,t}$	availability of population to abundance index i in year t
$n_{Avail,i}$	number of coefficients modeling effects on availability of population to abundance index i
$\boldsymbol{\beta}_{Avail}$	vector of $n_{Avail,i}$ coefficients modeling effects on availability of population to abundance index i
$\mathbf{X}_{Avail,i}$	$Y \times n_{Avail,i}$ design matrix modeling effects on availability of population to abundance index i
$\boldsymbol{X}_{Avail,i,t}$	row of the design matrix $\mathbf{X}_{Avail,i}$ corresponding to year t modeling effects on availability of population to abundance index i
l_{Avail_i}	lower bound on availability for abundance index i
u_{Avail_i}	upper bound on availability for abundance index i
$Adev_{i,t}$	availability deviation parameter for abundance index i in year t
$Eff_{i,t}$	efficiency of the gear used for abundance index i in year t
$n_{Eff,i}$	number of coefficients modeling effects on gear efficiency of abundance index i
$\boldsymbol{\beta}_{Eff}$	vector of $n_{Eff,i}$ coefficients modeling effects on gear efficiency of abundance index i
$\mathbf{X}_{Eff,i}$	$Y \times n_{Eff,i}$ design matrix modeling effects on gear efficiency of abundance index i
$\boldsymbol{X}_{Eff,i,t}$	row of the design matrix $\mathbf{X}_{Eff,i}$ corresponding to year t modeling effects on gear efficiency of abundance index i
l_{Eff_i}	lower bound on gear efficiency for abundance index i
u_{Eff_i}	upper bound on gear efficiency for abundance index i
$F_{f,t,a}$	fishing mortality for fleet f in year t at age a
$Fmult_{f,t}$	fully selected fishing mortality for fleet f in year t
$t_{min,f}$	first year that fleet f is operating
$t_{max,f}$	last year that fleet f is operating
β_f	natural logarithm of fully selected fishing mortality for fleet f in year $t_{min,f}$
$Fdev_{f,t_k}$	deviation parameter for fully selected fishing mortality for fleet f in year t_k
λ_{Fdev_f}	weight for the fishing mortality deviations penalty in the objective function
$Fdir_{f,t,a}$	directed fishing mortality rate at age a for fleet f in year t attributed to landings
$sdir_{t,a}$	directed selectivity at age a in year t attributed to landings

Table 1. *(continued)*

$Fdisc_{f,t,a}$	discard fishing mortality rate at age a for fleet f in year t attributed to catch not landed
RM_f	discard mortality rate for fleet f
$PR_{f,t,a}$	proportion of catch at age a released by fleet f in year t
$M_{t,a}$	natural mortality rate at age a in year t
m_1	number of coefficients modeling anual effects on natural mortality
$\beta_{M,1}$	vector of m_1 coefficients modeling anual effects on natural mortality
$\mathbf{X}_{M,1}$	$Y \times m_1$ design matrix modeling anual effects on natural mortality
$\boldsymbol{X}_{M,1,t}$	row of the design matrix $\mathbf{X}_{M,1}$ corresponding to year t modeling anual effects on natural mortality
m_2	number of coefficients modeling age effects on natural mortality
$\beta_{M,2}$	vector of m_2 coefficients modeling age effects on natural mortality
$\mathbf{X}_{M,2}$	$Y \times m_2$ design matrix modeling age effects on natural mortality
$\boldsymbol{X}_{M,2,a}$	row of the design matrix $\mathbf{X}_{M,2}$ corresponding to age class a modeling age effects on natural mortality
$F_{t,a}$	total fishing mortality rate, the sum of fishing mortality of all fleets, at age a in year t
$Z_{t,a}$	total mortality rate, the sum of fishing and natural mortality, at age a in year t
$\widetilde{R}_t$	expected recruitment in year t based on the stock-recruitment relationship
SSB_t	spawning stock biomass in year t
τ	steepness parameter of the stock-recruitment relationship
$R_{0,t}$	unexploited recruitment parameter of the stock-recruitment relationship in year t
$SSB_{0,t}$	unexploited spawning stock biomass parameter of the stock-recruitment relationship in year t
$SPR_{0,t}$	ratio of unexploited spawning stock biomass and recruitment in year t
$\Phi_{t,a}$	fecundity at age a in year t
p_{SSB}	fraction of the year elapsed at time of spawning
n_τ	number of coefficients modeling annual covariate effects on steepness
β_τ	vector of n_τ coefficients modeling annual covariate effects on steepness
$\mathbf{X}_\tau$	$Y \times n_\tau$ design matrix modeling annual covariate effects on steepness
$\boldsymbol{X}_{\tau,t}$	row of the design matrix $\mathbf{X}_\tau$ corresponding to year t modeling annual covariate effects on steepness
n_0	number of coefficients modeling annual covariate effects on either unexploited recruitment or *SSB*
β_0	vector of n_0 coefficients modeling annual covariate effects on unexploited recruitment or *SSB*
$\mathbf{X}_0$	$Y \times n_0$ design matrix modeling annual covariate effects on either unexploited recruitment or *SSB*
$\boldsymbol{X}_{0,t}$	row of the design matrix $\mathbf{X}_0$ corresponding to year t modeling annual covariate effects on either unexploited recruitment or *SSB*
$N_{t,a}$	population abundance (numbers) at age a in year t
$Rdev_t$	annual deviation parameter for recruitment in year t
a_{min}	minimum of age range used for calculating $Frep_t$
a_{max}	maximum of age range used for calculating $Frep_t$
$\omega_{t,a}$	one of three different types of weighting for age a and year t used for calculating $Frep_t$
$W_{t,a}$	weight (mass) of a fish at age a in year t on January 1
$Frep_t$	one of three different types of weighted averages of total fishing mortality over ages a_{min} to a_{max}
$\widehat{L}_{f,t,a}$	predicted landings in numbers by fleet f in year t at age a
$\widehat{L}_{W,t}$	predicted landings in weight by fleet f in year t
$\widehat{D}_{f,t,a}$	predicted discards in numbers by fleet f in year t at age a
$\widehat{D}_{W,t}$	predicted discards in weight by fleet f in year t
$W_{L,f,t,a}$	weight (mass) of a fish landed by fleet f at age a in year t
$\widehat{p}_{L,f,t,a}$	predicted proportion of landings by fleet f in year t at age a
$\widehat{p}_{D,f,t,a}$	predicted proportion of discards by fleet f in year t at age a
p_k	number of coefficients modeling length effects on relative catch efficiency (calibration) k
$\widetilde{\beta}_k$	vector of p_k input coefficients modeling length effects on relative catch efficiency (calibration) k
$\widehat{\beta}_k$	vector of p_k estimated coefficients modeling length effects on relative catch efficiency (calibration) k

Table 1. *(continued)*

$\boldsymbol{REdev}_k$	$\widehat{\boldsymbol{\beta}}_k - \widetilde{\boldsymbol{\beta}}_k$
$\boldsymbol{\Sigma}_k$	$p_k \times p_k$ variance-covariance matrix for estimates of $\boldsymbol{\beta}_k$
$\mathbf{X}_k$	$L \times p_k$ design matrix modeling length effects on relative catch efficiency (calibration) k
$\boldsymbol{X}_{k,l}$	row of the design matrix $\mathbf{X}_k$ corresponding to length class l modeling length effects on relative catch efficiency (calibration) k
$I_{C,i,t}$	calibrated observation for abundance index i in year t
$I_{C,i,t,a}$	calibrated observation for abundance index i in year t at age a
$p_{C,i,t,a}$	calibrated observation for proportion of abundance index i in year t at age a
$I_{U,i,t,l}$	uncalibrated observation for abundance index i in year t at length l
$\rho_{k,l}$	relative catch efficiency k at length l
$p_{i,t}(a\|l)$	proportion at age a given length l from the age length key for abundance index i in year t
$m_{i,t}$	month when abundance index i occurs in year t
$\delta_{i,t}$	fraction of the year elapsed at time of observation of abundance index i in year t
$\widehat{I}_{i,t}$	prediction from model for abundance index i in year t
$W_{i,t,a}$	weight (mass) of a fish at age a in year t for index i
$\widehat{p}_{i,t,a}$	predicted proportion at age a for abundance index i in year t
$F_{0.1}$	fully selected fishing mortality when slope of yield per recruit curve is 10%
F_{MAX}	fully selected fishing mortality that maximizes yield per recruit
F_{MSY}	fully selected fishing mortality that maximizes sustainable yield defined by stock-recruit relationship
$F_{X\%}$	fully selected fishing mortality that provides X% of unexploited SSB per recruit
$CV_{d,t}$	coefficient of variation for observation in year t for either fleet or abundance index d
$\sigma_{d,t}$	standard devation of the natural logarithm of the observation in year t for either fleet or abundance index d
λ_d	weight for fleet or abundance index d in the objective function
$ESS_{d,t}$	effective sample size of multinomial distribution for age composition observations in year t for either fleet or abundance index d
$p_{d,t,a}$	observed proportion at age a in year t for either fleet or abundance index d
$\mu_{d,t,a}$	predicted proportion at age a in year t for either fleet or abundance index d
λ_R	weight for recruitment penalty in the objective function
$\overline{SR}$	average value of R_0 or SSB_0 over all years in the model
$\widetilde{SR}$	initial value of R_0 or SSB_0
λ_R	weight for log-normal penalty on deviation of average R_0 or SSB_0 from initial value in the objective function
$\overline{\tau}$	average value of steepness over all years in the model
$\widetilde{\tau}$	initial value of steepness
λ_τ	weight for penalty on deviation of average steepness from initial value in the objective function
$\widetilde{N}_{1,a}$	initial value for abundance at age a in the first year
λ_N	weight for penalty on deviation of abundance at age a in the first year from initial value in the objective function
$\widetilde{Fmult}_{f,1}$	initial value for fully selected fishing mortality of fleet f in the first year
$\lambda_{Fmult_{f,1}}$	weight for penalty on deviation of fully selected fishing mortality of fleet f in the first year from initial value in the objective function
s_v	selectivity parameter v
$\widetilde{s}_v$	initial value of selectivity parameter
λ_{s_v}	weight for penalty on deviation of selectivity parameter v from initial value in the objective function
$\widehat{\boldsymbol{\beta}}_k$	estimates of coefficients for size effects on relative catch efficiency k
$\widetilde{\boldsymbol{\beta}}_k$	initial values of coefficients for size effects on relative catch efficiency k

Table 1. *(continued)*

$\lambda_{RE,k}$	weight for penalty on devation of estimated coefficients for size effects on relative catch efficiency k from initial values
$\overline{Avail_i}$	average availability of population to abundance index i over all years observed
$\widetilde{Avail_i}$	initial value of availability of population to abundance index i
λ_{Avail_i}	weight for penalty on deviation of average availability of population to abundance index i from initial value in the objective function
$\overline{Eff_i}$	average gear efficiency of abundance index i over all years observed
$\widetilde{Eff_i}$	initial value of gear efficiency of abundance index i
λ_{Eff_i}	weight for penalty on deviation of average gear efficiency of abundance index i from initial value in the objective function
λ_{Fdev_f}	weight for penalty on fully selected fishing mortality deviations in the objective function
λ_{Adev_i}	weight for penalty on annual deviations of availability of population to abundance index i in the objective function
$\overline{F}$	average fishing mortality of all ages and years in the model
$\overline{M}$	average natural mortality of all ages and years in the model
$\lambda_{F,M}$	weight for penalty on deviation of $\overline{F}$ from $\overline{M}$
MAX_F	maximum fishing mortality allowed in the model
Res	standardised residual for a particular observation or a penalized parameter estimate
$RMSE_d$	Root-mean squared error for data component or penalized parameter estimate d
x	a particular observation, parameter estimate, or average of annual parameter estimates
$\widetilde{x}$	a particular predicted observation or initial parameter value
$V(\mu_\theta)$	component of the standardized residual for a parameter with a 4-parameter beta distribution penalty
$\widehat{ESS}_{1,d,t}$	estimate type 1 of effective sample size for age composition data component d in year t
$\widehat{ESS}_{2,d,t}$	estimate type 2 of effective sample size for age composition data component d in year t
$R_{d,t}$	a component of $\widehat{n}_{2,d,t}$
$S_{d,t}$	a component of $\widehat{n}_{2,d,t}$
$W_{d,t}$	a component of $\widehat{n}_{2,d,t}$

6 APPENDICES

6.1 AD Model Builder Code for ASAP4

```
// ASAP4 (Age Structured Assessment Program Version 4: November 2014)
//   modified from ASAP3 by Timothy Miller
//   ASAP3 code by Christopher Legault with major contributions from Liz Brooks
//   modified from original ASAP by Christopher Legault and Victor Restrepo 1998

// Major changes from ASAP3
// 1) Restructure selectivitiy specification so that blocks and/or parameters can be
   used in multiple fleets and/or surveys.
// 2) Reparameterizes catchability as a product of availability and efficiency
//    as functions of annual covariates and flexibility of phases for each survey,
   single catchability possible by turning off one of the components.
//    Reparameterizing q required changing catchability random walk to be AR(1) for
   availability and phases specified for each index.
//    Random walk can still be done using a column of a single 1 with the rest zeros for
    covariates.  Also changed the process to be with respect to
//    the time span of the survey so that if there are years when the survey is not
   carried out, the random walk still occurs.
// 3) Estimation of natural mortality allowed, possibly as a function of annual
   covariates or age.
// 4) Internal length-based calibration of HBB:AIV series possible by providing curve
   coefficients and associated penalties,
//    covariate design matrix, and annual indices at length and age-length keys for
   calibrated years.
// 5) Fleets can operate for subsets of the entire model time span, fishing mortality in
    non-operating years will be zero. Random walks and phases
//    are fleet-specific spanning corresponding periods of operation
// 6) Changed way user determines which information is included in calculations and
   objective function to be less confusing.
//    Catch, discard, and index observations are included if observations are > 0. An
   entire index or fleet may be omitted
//    from likelihood if lambda is set to zero (and appropriate parameter phases are set
    to <= 0). Age composition will be included
//    only for years where the aggregate is > 0 AND input Neff is > 0. So, number of age
    comp years <= number of aggregate years.
// 7) Parameterized steepness and R0 (or S0) as functions of annual covariates.
// 8) Penalty for numbers at age in the first year now also always includes the first
   age class.

// Note: for indices, age composition in year y is not used if the aggregate index in
   year y is not used.

// Minor changes from ASAP3
// 1) removed option to use likelihood constants. Just always include them.
// 2) log-normal penalties are now normal or truncated normal (based on lower and upper
   bounds) by default. Priors/penalties for age-specific selectivity
//    parameters and steepness can also be shifted and scaled beta distributed instead
   of log-normal  to accommodate bounds provided (for selectivity) or 0.2-1.0 for
   steepness.
```

```
// 4) made objective function components sdreport numbers/vectors so that correlation
    with other components and parameters can more easily observed.
// 5) Timing of indices can now vary from year to year.

// 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 <admb2r.cpp>
  time_t start,finish;
  long hour,minute,second;
  double elapsed_time;
  ofstream ageproMCMC("asap4.bsn");
  ofstream basicMCMC("asap4MCMC.dat");
  ofstream inputlog("asap4input.log");
  ofstream fixlog("asap4fix.log");
  #define see(object) cout << #object ":\n" << object << endl;
  //--- preprocessor macro from Larry Jacobson NMFS-Woods Hole
  #define ICHECK(object) inputlog << "#" #object "\n" << object << endl;
  #define easy(object) cout << #object ":\n" << object << endl;

DATA_SECTION
  int debug
  int io
  number CVfill
  !! CVfill=100.0;

  // basic dimensions
  init_int n_years
  !! ICHECK(n_years);
  init_int year1
  !! ICHECK(year1);
  init_int n_ages
  !! ICHECK(n_ages);
  vector double_ages(1,n_ages)
  !! for(int i=1; i<=n_ages; i++) double_ages(i) = double(i);
  init_int n_fleets
  !! ICHECK(n_fleets);
  init_int n_indices
  !! ICHECK(n_indices);

  // fleet names here with $ in front of label
  // index names here with $ in front of label

  // biology
  // option now to estimate year and age effects on natural mortality
  init_matrix M_ini(1,n_years,1,n_ages)
  !! ICHECK(M_ini);
  init_int estimate_M //0 = no, 1 = yes
  !! ICHECK(estimate_M);
```

```
 init_int n_M_year_cov
 !! ICHECK(n_M_year_cov);
 init_matrix M_X_year(1,n_years,1,n_M_year_cov)
 !! ICHECK(M_X_year);
 init_ivector phase_M_year_pars(1,n_M_year_cov)
 !! ICHECK(phase_M_year_pars);
 init_vector M_year_pars_ini(1,n_M_year_cov)
 !! ICHECK(M_year_pars_ini);
 init_int n_M_age_cov
 !! ICHECK(n_M_age_cov);
 init_matrix M_X_age(1,n_ages,1,n_M_age_cov)
 !! ICHECK(M_X_age);
 init_ivector phase_M_age_pars(1,n_M_age_cov)
 !! ICHECK(phase_M_age_pars);
 init_vector M_age_pars_ini(1,n_M_age_cov)
 !! ICHECK(M_age_pars_ini);
LOCAL_CALCS
 if(estimate_M == 0) //use M at age  matrix input rather than estimate
 {
   M_year_pars_ini = 0.0;
   for(int i=1;i<=n_M_year_cov;i++) phase_M_year_pars(i) = -1;
   M_age_pars_ini = 0.0;
   for(int i=1;i<=n_M_age_cov;i++) phase_M_age_pars(i) = -1;
 }
END_CALCS
 init_number isfecund
 !! ICHECK(isfecund);
 init_number fracyearSSB
 !! ICHECK(fracyearSSB);
 init_matrix mature(1,n_years,1,n_ages)
 !! ICHECK(mature);
 init_int n_WAA_matrices
 !! ICHECK(n_WAA_matrices);
 int nrowsWAAini
 !! nrowsWAAini=n_years*n_WAA_matrices;
 init_matrix WAA_ini(1,nrowsWAAini,1,n_ages)
 !! ICHECK(WAA_ini);
 3darray WAA(1,n_WAA_matrices,1,n_years,1,n_ages)
 int nWAApointbio
 !! nWAApointbio=n_fleets*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,n_years,1,n_ages)
 3darray WAAcatchfleet(1,n_fleets,1,n_years,1,n_ages)
 3darray WAAdiscardfleet(1,n_fleets,1,n_years,1,n_ages)
 matrix WAAcatchall(1,n_years,1,n_ages)
 matrix WAAdiscardall(1,n_years,1,n_ages)
 matrix WAAssb(1,n_years,1,n_ages)
 matrix WAAjan1b(1,n_years,1,n_ages)
LOCAL_CALCS
 for (int i=1; i<=n_WAA_matrices; i++)
```

```
{
  for(int y=1; y<=n_years; y++) WAA(i,y) = WAA_ini((i-1)*n_years+y);
}

if ((max(WAApointbio) > n_WAA_matrices) || (min(WAApointbio) < 1))
{
  for (int i=1; i<=n_WAA_matrices; i++)
  {
  if(WAApointbio(i) > n_WAA_matrices || WAApointbio(i) < 1)
        fixlog << "WAApointbio(" << i <<") is " << WAApointbio(i) << " but it needs
           to be between 1 and " << n_WAA_matrices << endl;
  }
  ad_exit(1);
}
for (int i=1; i<=n_fleets; i++)
{
  WAAcatchfleet(i) = WAA(WAApointbio(i*2-1));
  WAAdiscardfleet(i) = WAA(WAApointbio(i*2));
}
ICHECK(WAAcatchfleet);
ICHECK(WAAdiscardfleet);
WAAcatchall=WAA(WAApointbio((n_fleets*2)+1));
WAAdiscardall=WAA(WAApointbio((n_fleets*2)+2));
WAAssb = WAA(WAApointbio((n_fleets*2)+3));
ICHECK(WAAssb);
WAAjan1b = WAA(WAApointbio((n_fleets*2)+4));
ICHECK(WAAjan1b);

if (isfecund==1) fecundity=mature;
else fecundity=elem_prod(WAAssb,mature);

END_CALCS

 // Catch *******************************************
 // Includes both landed and discarded components
 init_matrix CAA_ini(1,n_years*n_fleets,1,n_ages+1)
 !! ICHECK(CAA_ini);
 init_matrix DAA_ini(1,n_years*n_fleets,1,n_ages+1)
 !! ICHECK(DAA_ini);
 init_matrix proportion_release_ini(1,n_years*n_fleets,1,n_ages)
 !! ICHECK(proportion_release_ini);
 init_vector release_mort(1,n_fleets)
 !! ICHECK(release_mort);
 init_matrix catch_tot_CV(1,n_years,1,n_fleets)
 !! ICHECK(catch_tot_CV);
 init_matrix discard_tot_CV(1,n_years,1,n_fleets)
 !! ICHECK(discard_tot_CV);
 init_matrix input_Neff_catch_ini(1,n_years,1,n_fleets)
 !! ICHECK(input_Neff_catch_ini);
 init_matrix input_Neff_size_discard_ini(1,n_years,1,n_fleets)
 !! ICHECK(input_Neff_size_discard_ini);
```

```
3darray proportion_release(1,n_fleets,1,n_years,1,n_ages)
3darray catch_paa_obs(1,n_fleets,1,n_years,1,n_ages)
3darray discard_paa_obs(1,n_fleets,1,n_years,1,n_ages)
vector catch_age_comp_like_const(1,n_fleets)
vector discard_age_comp_like_const(1,n_fleets)
matrix catch_tot_fleet_obs(1,n_fleets,1,n_years)
matrix discard_tot_fleet_obs(1,n_fleets,1,n_years)
vector catch_tot_like_const(1,n_fleets)
vector discard_tot_like_const(1,n_fleets)
matrix catch_tot_sigma(1,n_fleets,1,n_years)
matrix discard_tot_sigma(1,n_fleets,1,n_years)
matrix input_Neff_catch(1,n_fleets,1,n_years)
matrix input_Neff_discard(1,n_fleets,1,n_years)

//fleets can be operating at different times
ivector n_catch_years(1,n_fleets)
ivector n_catch_age_comp_years(1,n_fleets)
ivector catch_time_span(1,n_fleets)
ivector catch_min_time(1,n_fleets)
ivector catch_max_time(1,n_fleets)

ivector n_discard_years(1,n_fleets)
ivector n_discard_age_comp_years(1,n_fleets)
ivector discard_time_span(1,n_fleets)
ivector discard_min_time(1,n_fleets)
ivector discard_max_time(1,n_fleets)

LOCAL_CALCS
catch_paa_obs=0.0;
discard_paa_obs=0.0;
catch_tot_like_const=0.0;
discard_tot_like_const=0.0;
catch_age_comp_like_const=0.0;
discard_age_comp_like_const=0.0;
n_catch_years = 0;
n_catch_age_comp_years = 0;
catch_min_time = 0;
catch_max_time = 0;
catch_time_span = 0;
n_discard_years = 0;
n_discard_age_comp_years = 0;
discard_min_time = 0;
discard_max_time = 0;
discard_time_span = 0;
dvector temp(1,n_ages);

for (int i=1;i<=n_fleets;i++)
{
  for (int y=1;y<=n_years;y++)
  {
    temp =CAA_ini((i-1)*n_years+y)(1,n_ages);
```

```
for(int a = 1; a<=n_ages; a++) if(temp(a)<0.0) temp(a) = 0.0;
catch_tot_fleet_obs(i,y)=CAA_ini((i-1)*n_years+y,n_ages+1);
input_Neff_catch(i,y)=input_Neff_catch_ini(y,i);
if (catch_tot_CV(y,i) < 1.0e-15)
{
  fixlog << "Changed catch_tot_CV(" << i << "," << y << ") to 100" << endl;
  catch_tot_CV(y,i) = CVfill;
}
catch_tot_sigma(i,y)=sqrt(log(catch_tot_CV(y,i)*catch_tot_CV(y,i)+1.0));
if (catch_tot_fleet_obs(i,y)>1.0e-15)
{
  if(catch_min_time(i) == 0) catch_min_time(i) = y;
  if(catch_max_time(i) < y) catch_max_time(i) = y;
  n_catch_years(i)++;
  catch_tot_like_const(i)+=0.5*log(2.0*PI) + log(catch_tot_sigma(i,y));
  if (sum(temp)>1.0e-15 && input_Neff_catch(i,y) > 1.0e-15)
  { //both requirements as well as total catch > 0 to include age comp in
      objective function
    n_catch_age_comp_years(i)++;
    catch_paa_obs(i,y)=temp/sum(temp);
    // compute multinomial constants for catch at age, if requested
    catch_age_comp_like_const(i) -= gammln(input_Neff_catch(i,y) + 1.0);
    catch_age_comp_like_const(i) += sum(gammln(input_Neff_catch(i,y)*catch_paa_obs
        (i,y) + 1.0));
  }
  else  catch_paa_obs(i,y)=0.0;
}

temp =DAA_ini((i-1)*n_years+y)(1,n_ages);
for(int a = 1; a<=n_ages; a++) if(temp(a)<0.0) temp(a) = 0.0;
discard_tot_fleet_obs(i,y)=DAA_ini((i-1)*n_years+y,n_ages+1);
proportion_release(i,y)=proportion_release_ini((i-1)*n_years+y)(1,n_ages);
input_Neff_discard(i,y)=input_Neff_size_discard_ini(y,i);
if (discard_tot_CV(y,i) < 1.0e-15)
{
  fixlog << "Changed discard_tot_CV(" << y << "," << i << ") to 100" << endl;
  discard_tot_CV(y,i) = CVfill;
}
discard_tot_sigma(i,y)=sqrt(log(discard_tot_CV(y,i)*discard_tot_CV(y,i)+1.0));
if (discard_tot_fleet_obs(i,y)>1.0e-15)
{
  if(discard_min_time(i) == 0) discard_min_time(i) = y;
  if(discard_max_time(i) < y) discard_max_time(i) = y;
  n_discard_years(i)++;
  discard_tot_like_const(i)+=0.5*log(2.0*PI)+log(discard_tot_sigma(i,y));
  //discard_tot_like_const(i)+=0.5*log(2.0*PI)+log(discard_tot_fleet_obs(i,y)) +
      log(discard_tot_sigma(i,y));
  if(sum(temp)>1.0e-15 && input_Neff_discard(i,y) > 1.0e-15)
  { //both requirements as well as total discards > 0 to include age comp in
      objective function
    n_discard_age_comp_years(i)++;
    discard_paa_obs(i,y)=temp/sum(temp);
```

```
        // compute multinomial constants for discards at age, if requested
        discard_age_comp_like_const(i) -= gammln(input_Neff_discard(i,y) + 1.0);
        discard_age_comp_like_const(i) += sum(gammln(input_Neff_discard(i,y)*
            discard_paa_obs(i,y) + 1.0));
      }
      else discard_paa_obs(i,y)=0.0;
    }
  }
  catch_time_span(i) = catch_max_time(i) - catch_min_time(i);
  discard_time_span(i) = discard_max_time(i) - discard_min_time(i);
}
END_CALCS

imatrix catch_years(1,n_fleets,1,n_catch_years)
imatrix catch_times(1,n_fleets,1,n_catch_years)
imatrix catch_age_comp_years(1,n_fleets,1,n_catch_age_comp_years)
imatrix catch_age_comp_times(1,n_fleets,1,n_catch_age_comp_years)

imatrix discard_years(1,n_fleets,1,n_discard_years)
imatrix discard_times(1,n_fleets,1,n_discard_years)
imatrix discard_age_comp_years(1,n_fleets,1,n_discard_age_comp_years)
imatrix discard_age_comp_times(1,n_fleets,1,n_discard_age_comp_years)

LOCAL_CALCS
catch_times = 0;
catch_years = 0;
catch_age_comp_times = 0;
catch_age_comp_years = 0;
discard_times = 0;
discard_years = 0;
discard_age_comp_times = 0;
discard_age_comp_years = 0;

for (int i=1;i<=n_fleets;i++)
{
  int catch_counter = 0;
  int discard_counter = 0;
  int catch_age_comp_counter = 0;
  int discard_age_comp_counter = 0;
  for (int y=1;y<=n_years;y++)
  {
    if(n_catch_years(i)>0)
    {
      if (catch_tot_fleet_obs(i,y)>1.0e-15)
      {
        catch_counter++;
        catch_times(i,catch_counter) = y;
        catch_years(i,catch_counter) = y + year1 -1;
        if(n_catch_age_comp_years(i)>0)
        {
          if (sum(catch_paa_obs(i,y))>1.0e-15 && input_Neff_catch(i,y) > 1.0e-15)
          {
```

```
          catch_age_comp_counter++;
          catch_age_comp_times(i,catch_age_comp_counter) = y;
          catch_age_comp_years(i,catch_age_comp_counter) = y + year1 -1;
        }
      }
    }
  }
  if(n_discard_years(i)>0)
  {
    if (discard_tot_fleet_obs(i,y)>1.0e-15)
    {
      discard_counter++;
      discard_times(i,discard_counter) = y;
      discard_years(i,discard_counter) = y + year1 -1;
      if(n_discard_age_comp_years(i)>0)
      {
        if (sum(discard_paa_obs(i,y))>1.0e-15 && input_Neff_discard(i,y) > 1.0e-15)
        {
          discard_age_comp_counter++;
          discard_age_comp_times(i,discard_age_comp_counter) = y;
          discard_age_comp_years(i,discard_age_comp_counter) = y + year1 -1;
        }
      }
    }
  }
 }
}
END_CALCS
 // Indices ******************************
 // Enter in all available indices and then pick the ones that are to be used in
    objective function
 // n_indices is the number of indices entered
 // nindices is the number of indices used (calculated by program)
 int indavail
 init_vector index_units_aggregate(1,n_indices) // 1=biomass, 2=numbers
 !! ICHECK(index_units_aggregate);
 init_vector index_units_proportions(1,n_indices) // 1=biomass, 2=numbers
 !! ICHECK(index_units_proportions);
 init_ivector index_WAApoint(1,n_indices) // pointer for which WAA matrix to use for
    biomass calculations for each index
 !! ICHECK(index_WAApoint);
 init_matrix index_month_ini(1,n_years,1,n_indices) // -1=average pop
 !! ICHECK(index_month_ini);
 init_ivector use_index_age_comp(1,n_indices) // 1=yes
 !! ICHECK(use_index_age_comp);
 init_ivector use_index(1,n_indices) // 1=yes
 !! ICHECK(use_index);
 init_matrix index_ini(1,n_years*n_indices,1,3+n_ages+1) // year, index value, CV,
    proportions at age, input effective sample size
 !! ICHECK(index_ini);

 ivector n_index_years(1,n_indices)
```

```
  int n_tot_index_years
  ivector n_index_age_comp_years(1,n_indices)
  ivector index_min_time(1,n_indices)
  ivector index_max_time(1,n_indices)
  ivector index_time_span(1,n_indices)
  matrix index_obs(1,n_indices,1,n_years)
  matrix index_month(1,n_indices,1,n_years)
  matrix index_cv(1,n_indices,1,n_years)
  matrix index_sigma(1,n_indices,1,n_years)
  matrix input_Neff_index(1,n_indices,1,n_years)
  matrix index_age_comp_like_const(1,n_indices,1,n_years)
  3darray index_paa_obs(1,n_indices,1,n_years,1,n_ages)
  3darray index_WAA(1,n_indices,1,n_years,1,n_ages)
  vector index_like_const(1,n_indices)

LOCAL_CALCS
  if ((max(index_WAApoint) > n_WAA_matrices) || (min(index_WAApoint) < 1))
  {
    for (int i=1; i<=n_WAA_matrices; i++)
    {
      if(index_WAApoint(i) > n_WAA_matrices || index_WAApoint(i) < 1)
        fixlog << "index_WAApoint(" << i <<") is " << index_WAApoint(i) << " but it
           needs to be between 1 and " << n_WAA_matrices << endl;
    }
    ad_exit(1);
  }
  for (int i=1; i<=n_years*n_indices; i++)
  {
    if (index_ini(i,3) <= 1.0e-15)
    {
      index_ini(i,3) = CVfill;
      fixlog << "Changed index_ini(" << i << ",3) (the CV) to 100" << endl;
    }
  }
  index_paa_obs=0.0;
  index_min_time = 0;
  index_max_time = 0;
  index_time_span = 0;
  index_obs = 0.0;
  index_cv = 0.0;
  index_sigma = 0.0;
  input_Neff_index = 0.0;
  index_like_const=0.0;
  index_WAA=0.0;
  index_age_comp_like_const=0.0;
  n_index_years = 0;
  n_index_age_comp_years = 0;

  for (int ii=1; ii<=n_indices; ii++)
  {
    // get the index and year specific information
    for (int y=1; y<=n_years; y++)
```

```
    {
      int i=(ii-1)*n_years+y;
      if (index_ini(i,2)>1.0e-15) //this year is used
      {
        if(index_min_time(ii) == 0) index_min_time(ii) = y;
        if(index_max_time(ii) < y) index_max_time(ii) = y;
        if(use_index(ii) == 1) n_index_years(ii)++;
        index_paa_obs(ii,y)=--(--(--index_ini(i)(4,3+n_ages)));
        for(int a = 1; a<=n_ages; a++) if(index_paa_obs(ii,y)(a)<0.0) index_paa_obs(ii,y
            )(a) = 0.0;
        index_obs(ii,y)=index_ini(i,2);
        index_cv(ii,y)=index_ini(i,3);
        index_sigma(ii,y)=sqrt(log(index_cv(ii,y)*index_cv(ii,y)+1.0));
        index_month(ii,y)=index_month_ini(y,ii);
        if(use_index_age_comp(ii) == 1) input_Neff_index(ii,y)=index_ini(i,n_ages+4);
        //add in log-normal likelihood constants only for years used
        index_like_const(ii)+=0.5*log(2.0*PI)+log(index_sigma(ii,y));
        if (sum(index_paa_obs(ii,y)) > 1.0e-15 && input_Neff_index(ii,y)>1.0e-15)
        {
          n_index_age_comp_years(ii)++;
          index_paa_obs(ii,y) = index_paa_obs(ii,y)/sum(index_paa_obs(ii,y));
          // compute multinomial constants for index
          index_age_comp_like_const(ii,y) -= gammln(input_Neff_index(ii,y) + 1.0);
          index_age_comp_like_const(ii,y) += sum(gammln(input_Neff_index(ii,y)*
              index_paa_obs(ii,y) + 1.0));
        }
        else index_paa_obs(ii,y) = 0.0;
      }
    }
    index_time_span(ii) = index_max_time(ii) - index_min_time(ii);
    // set up the index_WAA matrices (indices in numbers only will have WAA set to 0)
    if (index_units_aggregate(ii)==1 || index_units_proportions(ii)==1)
    {
      index_WAA(ii) = WAA(index_WAApoint(ii));
    }
  }
  n_tot_index_years = sum(n_index_years);
END_CALCS
  imatrix index_years(1,n_indices,1,n_index_years)
  imatrix index_times(1,n_indices,1,n_index_years)
  imatrix index_age_comp_years(1,n_indices,1,n_index_age_comp_years)
  imatrix index_age_comp_times(1,n_indices,1,n_index_age_comp_years)
LOCAL_CALCS
  index_times = 0;
  index_years = 0;
  index_age_comp_times = 0;
  index_age_comp_years = 0;
  for (int i=1;i<=n_indices;i++)
  {
    int index_counter = 0;
    int index_age_comp_counter = 0;
    if(n_index_years(i) > 0)
```

```
    {
      for (int y=1;y<=n_years;y++)
      {
        if (index_obs(i,y)>1.0e-15)
        {
          index_counter++;
          index_times(i,index_counter) = y;
          index_years(i,index_counter) = y+year1-1;
          if (sum(index_paa_obs(i,y))>1.0e-15 && input_Neff_index(i,y) > 1.0e-15)
          {
            index_age_comp_counter++;
            index_age_comp_times(i,index_age_comp_counter) = y;
            index_age_comp_years(i,index_age_comp_counter) = y+year1-1;
          }
        }
      }
    }
  }
END_CALCS

  //General internal calibration of indices, intended for accounting for length effects
      on HBB tows, but calibration without length effects could also be done
  //by specifying a single coefficienct and a column of ones for X, or the inverse for
      AIV tows, or other gear changes for other surveys.
  init_int calibrate_indices //flag to convert (HBB) indices internally with length-
      based relative catch efficiency
  !! ICHECK(calibrate_indices);
  init_int n_rel_efficiency_penalties
  !! ICHECK(n_rel_efficiency_penalties);
  init_int n_lengths
  !! ICHECK(n_lengths);
  init_vector lambda_rel_efficiency(1,n_rel_efficiency_penalties)
  !! ICHECK(lambda_rel_efficiency);
  init_ivector phase_rel_efficiency(1,n_rel_efficiency_penalties)
  !! ICHECK(phase_rel_efficiency);
  init_ivector n_rel_efficiency_coef(1,n_rel_efficiency_penalties)
  !! ICHECK(n_rel_efficiency_coef);
  //estimates of relative efficiency coefficients
  init_matrix rel_efficiency_coef_ini(1,n_rel_efficiency_penalties,1,
      n_rel_efficiency_coef) //ragged array
  !! ICHECK(rel_efficiency_coef_ini);
  int n_tot_rel_efficiency_coefs
  !! n_tot_rel_efficiency_coefs = sum(n_rel_efficiency_coef);
  ivector n_var_relefficiency_coef_cols(1,n_tot_rel_efficiency_coefs)
  ivector n_rel_efficiency_X_cols(1,n_lengths*n_rel_efficiency_penalties)
LOCAL_CALCS
  int count = 0;
  for(int i=1; i<=n_rel_efficiency_penalties; i++)
  {
    for(int j=1; j<= n_rel_efficiency_coef(i); j++)
    {
      count++;
```

```
      n_var_relefficiency_coef_cols(count) = n_rel_efficiency_coef(i);
    }
  }
  count = 0;
  for(int i=1; i<=n_rel_efficiency_penalties; i++)
  {
    for(int j=1; j<= n_lengths; j++)
    {
      count++;
      n_rel_efficiency_X_cols(count) = n_rel_efficiency_coef(i);
    }
  }
END_CALCS
  //estimate of var-cov matrix for relative efficiency coefficienct
  init_matrix var_rel_efficiency_coef_ini(1,n_tot_rel_efficiency_coefs,1,
      n_var_relefficiency_coef_cols) // ragged matrix
  !! ICHECK(var_rel_efficiency_coef_ini);
  3darray var_rel_efficiency_coef(1,n_rel_efficiency_penalties,1,n_rel_efficiency_coef
      ,1,n_rel_efficiency_coef)
  //if constant calibration just make this a column of 1s
  init_matrix rel_efficiency_X_ini(1,n_lengths*n_rel_efficiency_penalties,1,
      n_rel_efficiency_X_cols) //ragged matrix
  !! ICHECK(rel_efficiency_X_ini);
  3darray rel_efficiency_X(1,n_rel_efficiency_penalties,1,n_lengths,1,
      n_rel_efficiency_coef)

  vector rel_efficiency_penalty_const(1,n_rel_efficiency_penalties)
  //matrix of 0s and 1,...,n_rel_efficiency_penalties telling which observations for
      each index to be calibrated and which calibration to use
  init_imatrix calibrate_this_obs(1,n_indices,1,n_years)
  !! ICHECK(calibrate_this_obs);
  ivector n_calibrated_obs(1,n_indices)
  !! for(int i=1;i<=n_indices;i++)
  !! {
  !!   n_calibrated_obs(i)=0;
  !!   for(int y=1;y<=n_years;y++) if(calibrate_this_obs(i,y)>0) n_calibrated_obs(i)++;
  !! }
  !! ICHECK(n_calibrated_obs);
  int total_calibrated_obs
  !!total_calibrated_obs=sum(n_calibrated_obs);

  init_matrix uncalibrated_index_at_len_obs_ini(1,total_calibrated_obs,1,n_lengths)
  !! ICHECK(uncalibrated_index_at_len_obs_ini);
  3darray uncalibrated_index_at_len_obs(1,n_indices,1,n_years,1,n_lengths)
  //proportions at age given length for each length in each survey and year
  init_matrix age_length_keys_ini(1,total_calibrated_obs,1,n_ages*n_lengths)
  !! ICHECK(age_length_keys_ini);
  4darray age_length_keys(1,n_indices,1,n_years,1,n_lengths,1,n_ages) //proportions at
      age given length for each length in each survey and year
LOCAL_CALCS
  int count1 = 0;
  int count2 = 0;
```

```
  for(int i = 1;i <= n_rel_efficiency_penalties; i ++)
  {
    for(int j = 1; j <= n_rel_efficiency_coef(i); j++)
    {
      count1++;
      var_rel_efficiency_coef(i,j) = var_rel_efficiency_coef_ini(count1);
    }
    rel_efficiency_penalty_const(i) = 0.5* (n_rel_efficiency_coef(i) * log(2.0*PI) + log
        (det(var_rel_efficiency_coef(i))));
    for(int j = 1; j <= n_lengths; j++)
    {
      count2++;
      rel_efficiency_X(i,j) = rel_efficiency_X_ini(count2);
    }
  }
  count1 = 0;
  for(int i=1; i<=n_indices; i++)
  {
    for(int y=1; y<=n_years; y++)
    {
      age_length_keys(i,y) = 0.0;
      if(calibrate_this_obs(i,y) > 0)
      {
        count1++;
        uncalibrated_index_at_len_obs(i,y) = uncalibrated_index_at_len_obs_ini(count1);
        count2 = 0;
        for(int a=1; a<=n_ages; a++)
        {
          for(int l=1; l<=n_lengths; l++)
          {
            count2++;
            age_length_keys(i,y,l,a) = age_length_keys_ini(count1,count2);
          }
        }
      }
    }
  }
END_CALCS

  //new q parameterization as a product of availability and efficiency.
  //covariates for availability and efficiency are allowed.
  init_ivector n_availability_pars(1,n_indices)
  !! ICHECK(n_availability_pars);
  int total_availability_pars
  !! total_availability_pars = sum(n_availability_pars);
  init_matrix availability_X_ini(1,n_years,1,total_availability_pars)
  !! ICHECK(availability_X_ini);
  3darray availability_X(1,n_indices,1,n_years,1,n_availability_pars)
  init_vector availability_pars_ini(1,total_availability_pars)
  !! ICHECK(availability_pars_ini);
  init_ivector phase_availability_pars(1,total_availability_pars)
  !! ICHECK(phase_availability_pars);
```

```
init_vector availability_ini(1,n_indices)
!! ICHECK(availability_ini);
init_vector availability_penalty_CV(1,n_indices)
!! ICHECK(availability_penalty_CV);
init_ivector availability_penalty_type(1,n_indices)
!! ICHECK(availability_penalty_type);
init_vector availability_lower(1,n_indices)
!! ICHECK(availability_lower);
init_vector availability_upper(1,n_indices)
!! ICHECK(availability_upper);
init_vector lambda_availability(1,n_indices)
!! ICHECK(lambda_availability);
vector availability_penalty_sigma(1,n_indices)
vector availability_penalty_lnorm_scale(1,n_indices)
vector availability_penalty_phi(1,n_indices)
vector availability_penalty_a(1,n_indices)
vector availability_penalty_b(1,n_indices)
vector availability_penalty_mu(1,n_indices)
vector availability_penalty_const(1,n_indices)
ivector first_availability_phase(1,n_indices)

init_vector lambda_availability_AR1(1,n_indices)
!! ICHECK(lambda_availability_AR1);
init_vector availability_AR1_sd(1,n_indices)
!! ICHECK(availability_AR1_sd);
init_ivector phase_availability_AR1(1,n_indices)
!! ICHECK(phase_availability_AR1);
int total_availability_AR1_devs
!! total_availability_AR1_devs = sum(index_time_span);
vector availability_AR1_penalty_const(1,n_indices)

init_ivector n_efficiency_pars(1,n_indices)
!! ICHECK(n_efficiency_pars);
int total_efficiency_pars
!! total_efficiency_pars = sum(n_efficiency_pars);
init_matrix efficiency_X_ini(1,n_years,1,total_efficiency_pars)
!! ICHECK(efficiency_X_ini);
3darray efficiency_X(1,n_indices,1,n_years,1,n_efficiency_pars)
init_vector efficiency_pars_ini(1,total_efficiency_pars)
!! ICHECK(efficiency_pars_ini);
init_ivector phase_efficiency_pars(1,total_efficiency_pars)
!! ICHECK(phase_efficiency_pars);
init_vector efficiency_ini(1,n_indices)
!! ICHECK(efficiency_ini);
init_vector efficiency_penalty_CV(1,n_indices)
!! ICHECK(efficiency_penalty_CV);
init_ivector efficiency_penalty_type(1,n_indices)
!! ICHECK(efficiency_penalty_type);
init_vector efficiency_lower(1,n_indices)
!! ICHECK(efficiency_lower);
init_vector efficiency_upper(1,n_indices)
!! ICHECK(efficiency_upper);
```

```
  init_vector lambda_efficiency(1,n_indices)
  !! ICHECK(lambda_efficiency);
  vector efficiency_penalty_sigma(1,n_indices)
  vector efficiency_penalty_lnorm_scale(1,n_indices)
  vector efficiency_penalty_phi(1,n_indices)
  vector efficiency_penalty_a(1,n_indices)
  vector efficiency_penalty_b(1,n_indices)
  vector efficiency_penalty_mu(1,n_indices)
  vector efficiency_penalty_const(1,n_indices)
  ivector first_efficiency_phase(1,n_indices)
LOCAL_CALCS
  availability_X = 0.0;
  efficiency_X = 0.0;
  count = 0;
  double max_CV = 0.0;
  for(int i = 1; i <= n_indices; i++)
  {
    for(int j=1; j<=n_availability_pars(i); j++)
    {
      count++;
      //make sure no availability parameters are trying to be estimated when the index
         is not used
      if(n_index_years(i) == 0 && phase_availability_pars(count) > 0)
      {
        phase_availability_pars(count) = -1;
        fixlog << "Changed phase for availability parameter " << count << " to -1
           because it pertains to index " << i << " which is not used." << endl;
      }
      //fill out the array for the design matrices
      for(int y=1; y<=n_years; y++)
      {
        availability_X(i,y,j) = availability_X_ini(y,count);
      }
    }
  }
  for(int i = 1; i <= n_indices; i++)
  {
    if(availability_lower(i) < 0.0)
    {
      availability_lower(i) = 0.0;
      fixlog << "Changed lower bound for availability for index " << i << " to 0 because
          it was less than 0" << endl;
    }
    if(availability_upper(i) < availability_lower(i))
    {
      availability_upper(i) = 1.0 + availability_lower(i);
      fixlog << "Changed upper bound for availability for index " << i << " to 1 + lower
          bound because it was less than lower bound" << endl;
    }
    if(availability_ini(i) < availability_lower(i) || availability_ini(i) >
       availability_upper(i))
    {
```

```
    if(availability_ini(i) < availability_lower(i))
    {
      availability_ini(i) = availability_lower(i) + 1.0e-15;
      fixlog << "changed initial value for availability parameter " << i << " to " <<
          availability_lower(i) + 1.0e-15 <<
        " because it was < the lower bound, " << availability_lower(i) << endl;
    }
    else
    {
      availability_ini(i) = availability_upper(i) - 1.0e-15;
      fixlog << "changed initial value for availability parameter " << i << " to " <<
          availability_upper(i) - 1.0e-15<<
        " because it was > the upper bound, " << availability_upper(i) << endl;
    }
  }
  if(availability_penalty_type(i) == 1)
  {
    max_CV = sqrt((availability_upper(i)-availability_ini(i))*(availability_ini(i) -
        availability_lower(i))/square(availability_ini(i)));
    if(availability_penalty_CV(i) >= max_CV)
    { //CV(par) must be < (E(par)-lower)(upper-E(par))/E(par)^2
      availability_penalty_CV(i) = max_CV*0.9999;
      fixlog << "Changed penalty CV for availability of index " << i << " to " <<
          max_CV*0.9999 << " because it was >= than maximum" << endl;
    }
    if(availability_penalty_CV(i) < 1.0e-15)
    {
      availability_penalty_CV(i) = max_CV*0.0001;
      fixlog << "Changed penalty CV for availability of index " << i << " to " <<
          max_CV*0.0001 << " because it was <= 0" << endl;
    }

    //beta distribution for penalties: phi = alpha + beta = mu/(1-mu)/CV^2 -1
    availability_penalty_mu(i) = (availability_ini(i) - availability_lower(i))/(
        availability_upper(i) - availability_lower(i));
    availability_penalty_phi(i) = (1-availability_penalty_mu(i))*
        availability_penalty_mu(i)*square(availability_upper(i) - availability_lower(i)
        )/square(availability_ini(i)*availability_penalty_CV(i)) - 1.0;
    availability_penalty_a(i) = availability_penalty_phi(i)*availability_penalty_mu(i)
        ;
    availability_penalty_b(i) = availability_penalty_phi(i)*(1-
        availability_penalty_mu(i));
    availability_penalty_const(i) = (availability_penalty_phi(i)-1.0)*log(
        availability_upper(i)-availability_lower(i) + 1.0e-15) - gammln(
        availability_penalty_phi(i)) +
    gammln(availability_penalty_a(i)) + gammln(availability_penalty_b(i));
  }
  else
  {
    if(availability_penalty_CV(i) < 1.0e-15)
    {
      fixlog << "Changed availability_penalty_CV(" << i << ") to 100" << endl;
```

```
    availability_penalty_CV(i) = CVfill;
  }
  availability_penalty_sigma(i) = sqrt(log(availability_penalty_CV(i)*
      availability_penalty_CV(i)+1.0));
  availability_penalty_lnorm_scale(i) = cumd_norm((log(availability_upper(i))-log(
      availability_ini(i)+1.0e-15))/availability_penalty_sigma(i)) -
  cumd_norm((log(availability_lower(i)+1.0e-15)-log(availability_ini(i)+1.0e-15))/
      availability_penalty_sigma(i));
  availability_penalty_const(i) = 0.5*log(2.0*PI) + log(availability_penalty_sigma(i
      )) + log(availability_penalty_lnorm_scale(i));
}

//availabity AR1 process
if(availability_AR1_sd(i) < 1.0e-15)
{
  availability_AR1_sd(i) = 0.0001;
  fixlog << "Changed penalty sd for availability AR1 process of index " << i << " to
       0.0001 because it was <= 0" << endl;
}
if(n_index_years(i) == 0 && phase_availability_AR1(i) > 0)
{
  phase_availability_AR1(i) = -1;
  fixlog << "Changed phase for availability AR1 process of index " << i << " to -1
      because this index is not used." << endl;
}
availability_AR1_penalty_const(i) = double(index_time_span(i))*(0.5*log(2.0*PI) +
    log(availability_AR1_sd(i)));

//efficiency
count = 0;
for(int j=1; j<=n_efficiency_pars(i); j++)
{
  count++;
  if(n_index_years(i) == 0 && phase_efficiency_pars(count) > 0)
  {
    phase_efficiency_pars(count) = -1;
    fixlog << "Changed phase for efficiency parameter " << count << " to -1 because
        it pertains to index " << i << " which is not used." << endl;
  }
  for(int y=1; y<=n_years; y++)
  {
    efficiency_X(i,y,j) = efficiency_X_ini(y,count);
  }
}
if(efficiency_lower(i) < 0.0)
{
  efficiency_lower(i) = 0.0;
  fixlog << "Changed lower bound for efficiency for index " << i << " to 0 because
      it was less than 0" << endl;
}
if(efficiency_upper(i) < efficiency_lower(i))
{
```

```
  efficiency_upper(i) = 1.0 + efficiency_lower(i);
  fixlog << "Changed upper bound for efficiency for index " << i << " to 1 + lower
     bound because it was less than lower bound" << endl;
}
if(efficiency_ini(i) < efficiency_lower(i) || efficiency_ini(i) > efficiency_upper(i
   ))
{
  if(efficiency_ini(i) < efficiency_lower(i))
  {
    efficiency_ini(i) = efficiency_lower(i) + 1.0e-15;
    fixlog << "changed initial value for efficiency parameter " << i << " to " <<
       efficiency_lower(i) + 1.0e-15 <<
     " because it was < the lower bound, " << efficiency_lower(i) << endl;
  }
  else
  {
    efficiency_ini(i) = efficiency_upper(i) - 1.0e-15;
    fixlog << "changed initial value for efficiency parameter " << i << " to " <<
       efficiency_upper(i) - 1.0e-15 <<
     " because it was > the upper bound, " << efficiency_upper(i) << endl;
  }
}
if(efficiency_penalty_type(i) == 1)
{
  max_CV = sqrt((efficiency_upper(i)-efficiency_ini(i))*(efficiency_ini(i) -
     efficiency_lower(i))/square(efficiency_ini(i)));
  if(efficiency_penalty_CV(i) >= max_CV)
  { //CV(par) must be < (E(par)-lower)(upper-E(par))/E(par)^2
    efficiency_penalty_CV(i) = max_CV*0.9999;
    fixlog << "Changed penalty CV for efficiency of index " << i << " to " << max_CV
       *0.9999 << " because it was >= than maximum" << endl;
  }
  if(efficiency_penalty_CV(i) < 1.0e-15)
  {
    efficiency_penalty_CV(i) = max_CV*0.0001;
    fixlog << "Changed penalty CV for efficiency of index " << i << " to " << max_CV
       *0.0001 << " because it was <= 0" << endl;
  }
  //beta distribution for penalties: phi = alpha + beta = mu/(1-mu)/CV^2 -1
  efficiency_penalty_mu(i) = (efficiency_ini(i) - efficiency_lower(i))/(
     efficiency_upper(i) - efficiency_lower(i));
  efficiency_penalty_phi(i) = (1-efficiency_penalty_mu(i))*efficiency_penalty_mu(i)*
     square(efficiency_upper(i) - efficiency_lower(i))/square(efficiency_ini(i)*
     efficiency_penalty_CV(i)) - 1.0;
  efficiency_penalty_a(i) = efficiency_penalty_phi(i)*efficiency_penalty_mu(i);
  efficiency_penalty_b(i) = efficiency_penalty_phi(i)*(1- efficiency_penalty_mu(i));
  efficiency_penalty_const(i) = (efficiency_penalty_phi(i)-1.0)*log(efficiency_upper
     (i)-efficiency_lower(i) + 1.0e-15) - gammln(efficiency_penalty_phi(i)) +
  gammln(efficiency_penalty_a(i)) + gammln(efficiency_penalty_b(i));
}
else
{
```

```
      if(efficiency_penalty_CV(i) < 1.0e-15)
      {
        fixlog << "Changed efficiency_penalty_CV(" << i << ") to 100" << endl;
        efficiency_penalty_CV(i) = CVfill;
      }
      efficiency_penalty_sigma(i) = sqrt(log(efficiency_penalty_CV(i)*
         efficiency_penalty_CV(i)+1.0));
      efficiency_penalty_lnorm_scale(i) = cumd_norm((log(efficiency_upper(i))-log(
         efficiency_ini(i)+1.0e-15))/efficiency_penalty_sigma(i)) -
      cumd_norm((log(efficiency_lower(i)+1.0e-15)-log(efficiency_ini(i)+1.0e-15))/
         efficiency_penalty_sigma(i));
      efficiency_penalty_const(i) = 0.5*log(2.0*PI) + log(efficiency_penalty_sigma(i)) +
          log(efficiency_penalty_lnorm_scale(i));
    }
  }
  count = 0;
  for(int i =1; i <= n_indices; i++)
  {
    first_availability_phase(i) = max(phase_availability_pars(count+1,count+
       n_availability_pars(i)));
    if(first_availability_phase(i)>0)
    {
      for(int j=1;j<= n_availability_pars(i);j++)
      {
        if(phase_availability_pars(count+j) > 0 && phase_availability_pars(count+j) <
           first_availability_phase(i))
        first_availability_phase(i) = phase_availability_pars(count+j);
      }
    }
    count = sum(n_availability_pars(1,i));
  }
  count = 0;
  for(int i =1; i <= n_indices; i++)
  {
    first_efficiency_phase(i) = max(phase_efficiency_pars(count+1,count+
       n_efficiency_pars(i)));
    if(first_efficiency_phase(i)>0)
    {
      for(int j=1;j<= n_efficiency_pars(i);j++)
      {
        if(phase_efficiency_pars(count+j) > 0 && phase_efficiency_pars(count+j) <
           first_efficiency_phase(i))
        first_efficiency_phase(i) = phase_efficiency_pars(count+j);
      }
    }
    count = sum(n_efficiency_pars(1,i));
  }
END_CALCS
  !! for(int i=1;i<=n_indices; i++) ICHECK(availability_X(i));
  !! for(int i=1;i<=n_indices; i++) ICHECK(efficiency_X(i));

  // 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,n_ages)
  !! ICHECK(NAA_year1_ini);
  init_vector Fmult_year1_ini(1,n_fleets)
  !! ICHECK(Fmult_year1_ini);
  init_number is_SR_scalar_R // 1 for R0, 0 for SSB0
  !! ICHECK(is_SR_scalar_R);
  init_number SR_scalar_ini
  !! ICHECK(SR_scalar_ini);
  init_number steepness_ini
  !! ICHECK(steepness_ini);

  // Phase Controls (other than selectivity and availability,efficiency)
  init_ivector phase_Fmult_year1(1,n_fleets)
  !! ICHECK(phase_Fmult_year1);
  init_ivector phase_Fmult_devs(1,n_fleets)
  !! ICHECK(phase_Fmult_devs);
  int n_tot_Fmult_devs;
  !! n_tot_Fmult_devs = sum(n_catch_years) - n_fleets; //no dev in first year
  ivector phase_Fmult_devs_ini(1,n_tot_Fmult_devs)
LOCAL_CALCS
  int counter = 0;
  for(int i=1;i<=n_fleets;i++)
  {
    phase_Fmult_devs_ini(1 + counter, sum(n_catch_years(1,i))-i) = phase_Fmult_devs(i);
    counter = sum(n_catch_years(1,i))-i;
  }
  ICHECK(phase_Fmult_devs_ini);
END_CALCS
  init_int phase_recruit_devs
  !! ICHECK(phase_recruit_devs);
  init_int phase_N_year1_devs
  !! ICHECK(phase_N_year1_devs);
  init_int phase_SR_scalar
  !! ICHECK(phase_SR_scalar);
  init_int phase_steepness
  !! ICHECK(phase_steepness);

  //weights for data components and priors/penalties
  init_vector lambda_catch_tot(1,n_fleets)
  !! ICHECK(lambda_catch_tot);
  init_vector lambda_discard_tot(1,n_fleets)
  !! ICHECK(lambda_discard_tot);
  init_vector lambda_index(1,n_indices)
  !! ICHECK(lambda_index);
  init_vector lambda_Fmult_year1(1,n_fleets)
  !! ICHECK(lambda_Fmult_year1);
  init_vector lambda_Fmult_devs(1,n_fleets)
  !! ICHECK(lambda_Fmult_devs);
  init_number lambda_N_year1_devs
```

```
 !! ICHECK(lambda_N_year1_devs);
 init_number lambda_recruit_devs
 !! ICHECK(lambda_recruit_devs);
 init_number lambda_steepness
 !! ICHECK(lambda_steepness);
 init_number lambda_SR_scalar
 !! ICHECK(lambda_SR_scalar);

 // CVs for priors/penalties
 init_vector recruit_CV(1,n_years)
 !! ICHECK(recruit_CV);
 init_vector Fmult_year1_CV(1,n_fleets)
 !! ICHECK(Fmult_year1_CV);
 init_vector Fmult_devs_CV(1,n_fleets)
 !! ICHECK(Fmult_devs_CV);
 init_number N_year1_CV
 !! ICHECK(N_year1_CV);
 init_number steepness_penalty_CV
 !! ICHECK(steepness_penalty_CV);
 init_int steepness_penalty_type
 !! ICHECK(steepness_penalty_type);
 init_number SR_scalar_CV
 !! ICHECK(SR_scalar_CV);
 vector recruit_sigma(1,n_years)
 number SR_penalty_const
 vector Fmult_year1_sigma(1,n_fleets)
 vector Fmult_year1_penalty_const(1,n_fleets)
 vector Fmult_devs_sigma(1,n_fleets)
 vector Fmult_devs_penalty_const(1,n_fleets)
 number N_year1_sigma
 number N_year1_penalty_const
 number steepness_penalty_const
 number SR_scalar_sigma
 number SR_scalar_penalty_const
 number steepness_penalty_phi
 number steepness_penalty_lnorm_scale
 number steepness_penalty_sigma
 number steepness_penalty_a
 number steepness_penalty_b
 number steepness_penalty_mu
LOCAL_CALCS
 for (int y=1; y<=n_years; y++)
 {
   if (recruit_CV(y) < 1.0e-15)
   {
     fixlog << "Changed recruit_CV(" << y << ") to 100" << endl;
     recruit_CV(y) = CVfill;
   }
 }
 for (int i=1; i<=n_fleets; i++)
 {
   if (Fmult_year1_CV(i) < 1.0e-15)
```

```
    {
      fixlog << "Changed Fmult_year1_CV(" << i << ") to 100" << endl;
      Fmult_year1_CV(i) = CVfill;
    }
    if (Fmult_devs_CV(i) < 1.0e-15)
    {
      fixlog << "Changed Fmult_devs_CV(" << i << ") to 100" << endl;
      Fmult_devs_CV(i) = CVfill;
    }
  }
  if (N_year1_CV < 1.0e-15)
  {
    fixlog << "Changed N_year1_CV to 100" << endl;
    N_year1_CV = CVfill;
  }
  //steepness is bounded between 0.2 and 1.0 so use a scaled beta prior/penalty
  if(steepness_ini < 0.2 || steepness_ini > 1.0)
  {
    if(steepness_ini < 0.2)
    {
      steepness_ini = 0.2 + 1.0e-15;
      fixlog << "changed initial value for steepness to " << 0.2 + 1.0e-15 << " because
          it was < the lower bound, 0.2" << endl;
    }
    else
    {
      steepness_ini = 1.0 - 1.0e-15;
      fixlog << "changed initial value for steepness to " << 1.0 - 1.0e-15 << " because
          it was > the upper bound, 1.0" << endl;
    }
  }

  if(steepness_penalty_type == 1)
  {
    double max_CV = sqrt((1.0-steepness_ini)*(steepness_ini - 0.2)/square(steepness_ini)
       );
    if(steepness_penalty_CV >= max_CV)
    { //CV(steepness) must be < sqrt((E(par)-0.2)(1.0-E(par))/E(par)^2)
      steepness_penalty_CV = max_CV*0.9999;
      fixlog << "Change penalty CV for steepness to " << max_CV*0.9999 << " because it
         was >= maximum" << endl;
    }
    if(steepness_penalty_CV < 1.0e-15)
    {
      steepness_penalty_CV = max_CV*0.0001;
      fixlog << "Changed penalty CV for steepness to " << max_CV*0.0001 << " because it
         was <= 0" << endl;
    }
    steepness_penalty_mu = (steepness_ini - 0.2)/(1.0 - 0.2);
    steepness_penalty_phi = (1-steepness_penalty_mu)*steepness_penalty_mu*square(1.0 -
       0.2)/square(steepness_ini*steepness_penalty_CV) - 1.0;
    steepness_penalty_a = steepness_penalty_phi*steepness_penalty_mu;
```

```
    steepness_penalty_b = steepness_penalty_phi*(1- steepness_penalty_mu);
    steepness_penalty_const = (steepness_penalty_phi -1.0)*log(1.0-0.2) - gammln(
        steepness_penalty_phi) +
      gammln(steepness_penalty_a) + gammln(steepness_penalty_b);
  }
  else
  {
    if(steepness_penalty_CV < 1.0e-15)
    {
      fixlog << "Changed steepness_penalty_CV to 100" << endl;
      steepness_penalty_CV = CVfill;
    }
    steepness_penalty_sigma = sqrt(log(steepness_penalty_CV*steepness_penalty_CV+1.0));
    steepness_penalty_lnorm_scale = cumd_norm((log(1.0)-log(steepness_ini))/
        steepness_penalty_sigma) -
          cumd_norm((log(0.2)-log(steepness_ini))/steepness_penalty_sigma);
    steepness_penalty_const = 0.5*log(2.0*PI) + log(steepness_penalty_sigma) + log(
        steepness_penalty_lnorm_scale);
  }
  if (SR_scalar_CV < 1.0e-15)
  {
    fixlog << "Changed SR_scalar_CV to 100" << endl;
    SR_scalar_CV = CVfill;
  }
  // convert CVs to variances
  recruit_sigma=sqrt(log(elem_prod(recruit_CV,recruit_CV)+1.0));
  Fmult_year1_sigma=sqrt(log(elem_prod(Fmult_year1_CV,Fmult_year1_CV)+1.0));
  Fmult_devs_sigma=sqrt(log(elem_prod(Fmult_devs_CV,Fmult_devs_CV)+1.0));
  N_year1_sigma=sqrt(log(N_year1_CV*N_year1_CV+1.0));
  SR_scalar_sigma=sqrt(log(SR_scalar_CV*SR_scalar_CV+1.0));

  // calculate penalty constants
  SR_penalty_const=0.5*double(n_years)*log(2.0*PI) + sum(log(recruit_sigma));
  SR_scalar_penalty_const=0.5*log(2.0*PI) + log(SR_scalar_sigma);
  Fmult_year1_penalty_const=0.5*log(2.0*PI) + log(Fmult_year1_sigma);
  for(int i=1;i<=n_fleets;i++) Fmult_devs_penalty_const(i)=0.5*double(catch_time_span(i)
     )*(log(2.0*PI) + log(Fmult_devs_sigma(i)));
  N_year1_penalty_const=0.5*double(n_ages-1)*log(2.0*PI) + double(n_ages-1)*log(
     N_year1_sigma);

END_CALCS

  // Selectivity ****************************************
  // Selectivity is defined for all fleets/surveys so that selectivities can be mixed
     and matched.
  // Also, a selectivity parameter locator allows parameters to be used for multiple
     ages in a block
  // or some parameters can be used in multiple selectivity blocks

  init_int n_selblocks; //now for both fleets and indices. can be as small as one if all
      fleets and surveys have the same selectivity
  !! ICHECK(n_selblocks);
```

```
  init_imatrix fleet_selblock_pointer_ini(1,n_years,1,n_fleets)
  !! ICHECK(fleet_selblock_pointer_ini);
  imatrix fleet_selblock_pointer(1,n_fleets,1,n_years)
  init_imatrix index_selblock_pointer_ini(1,n_years,1,n_indices)
  !! ICHECK(index_selblock_pointer_ini);
  imatrix index_selblock_pointer(1,n_indices,1,n_years)
  ivector n_selpars_by_block(1,n_selblocks)
  init_ivector selblock_type(1,n_selblocks)
  !! ICHECK(selblock_type);

LOCAL_CALCS
  for(int i = 1; i<=n_selblocks; i++)
  {
    if(selblock_type(i) == 1) n_selpars_by_block(i) = n_ages; //by age, estimated
        selectivity parameters are log(p/(1-p))
    else if(selblock_type(i) == 2) n_selpars_by_block(i) = 2; //logistic
    else if(selblock_type(i) == 3) n_selpars_by_block(i) = 4; //double logistic
    else
    {
      fixlog << "selblock_type(" << i << ")  = " << selblock_type(i) << " is not valid.
          Must be 1, 2, or 3" << endl;
      ad_exit(1);
    }
  }
  for(int i=1; i<=n_fleets; i++)
  {
    for(int y = 1; y<=n_years; y++)
    {
      fleet_selblock_pointer(i,y) = fleet_selblock_pointer_ini(y,i);
      if(fleet_selblock_pointer(i,y) < 1 || fleet_selblock_pointer(i,y)> n_selblocks)
      {
        fixlog << "Selectivity block for fleet " << i << " in year " << y << " is "  <<
            fleet_selblock_pointer(i,y) << ", but it must be between 1 and " <<
            n_selblocks << endl;
        ad_exit(1);
      }
    }
  }
  for(int i=1; i<=n_indices; i++)
  {
    for(int y=1; y<=n_years; y++)
    {
      index_selblock_pointer(i,y) = index_selblock_pointer_ini(y,i);
      if(index_selblock_pointer(i,y) < 1 || index_selblock_pointer(i,y)> n_selblocks)
      {
        fixlog << "Selectivity block for index " << i << " in year " << y << " is "  <<
            index_selblock_pointer(i,y) << ", but it must be between 1 and " <<
            n_selblocks << endl;
        ad_exit(1);
      }
    }
  }
```

```
END_CALCS

  init_int n_selpars
  init_vector selpars_ini(1,n_selpars) //input initial values
  !! ICHECK(selpars_ini);
  init_ivector phase_selpars(1,n_selpars)
  !! ICHECK(phase_selpars);
  init_vector selpars_upper(1,n_selpars)
  !! ICHECK(selpars_upper);
  init_vector selpars_lower(1,n_selpars)
  !! ICHECK(selpars_lower);
  init_vector lambda_selpars(1,n_selpars)
  !! ICHECK(lambda_selpars);
  init_vector selpars_penalty_CV(1,n_selpars)
  !! ICHECK(selpars_penalty_CV);
  init_ivector selpars_penalty_type(1,n_selpars) //1 is scaled,shifted beta, else
     truncated log-normal
  !! ICHECK(selpars_penalty_type);
  vector selpars_penalty_lnorm_scale(1,n_selpars)
  vector selpars_penalty_sigma(1,n_selpars)
  vector selpars_penalty_const(1,n_selpars)
  vector selpars_penalty_a(1,n_selpars)
  vector selpars_penalty_b(1,n_selpars)
  vector selpars_penalty_mu(1,n_selpars)
  vector selpars_penalty_phi(1,n_selpars)
  init_imatrix selpars_pointer(1,n_selblocks,1,n_selpars_by_block) //ragged matrix
  !! ICHECK(selpars_pointer);
  ivector selpar_use_count(1,n_selpars)
  !! selpar_use_count = 0;
  !! for(int i=1; i<=n_selpars; i++)
  !!   for(int j=1; j<=n_selblocks; j++)
  !!     for(int k=1; k<=n_selpars_by_block(j); k++)
  !!       if(selpars_pointer(j,k) == i) selpar_use_count(i)++;
  !! ICHECK(selpar_use_count);
  imatrix selpar_selblocks(1,n_selpars,1,selpar_use_count)//ragged  matrix
  imatrix selpar_seltypes(1,n_selpars,1,selpar_use_count)//ragged  matrix
  imatrix selpar_selpositions(1,n_selpars,1,selpar_use_count)//ragged  matrix

LOCAL_CALCS
  if(max(selpars_pointer) > n_selpars)
  {
    fixlog << "number of selectivity parameters indicated in locator, " << max(
       selpars_pointer) <<
      " is greater than the number of parameters specified," << n_selpars << endl;
    ad_exit(1);
  }
  if(min(selpar_use_count) == 0)
  {
    for(int i = 1; i <= n_selpars; i++)
    {
      if(selpar_use_count(i) == 0 && phase_selpars(i) > 0)
      {
```

```
      fixlog << "selectivity par " << i << " is not used in any selectivity blocks and
          its phase is " << phase_selpars(i) <<
        ", so setting it to -1" << endl;
      phase_selpars(i) = -1;
    }
  }
}
selpar_selblocks = 0;
selpar_seltypes = 0;
selpar_selpositions = 0;
for(int i=1; i<=n_selpars; i++)
{
  int count = 0;
  for(int j=1; j<=n_selblocks; j++)
  {
    for(int k=1; k<=n_selpars_by_block(j); k++)
    {
      if(selpars_pointer(j,k) == i) //found where the parameter is used
      {
        count++;
        selpar_selblocks(i,count) = j;
        selpar_seltypes(i,count) = selblock_type(j);
        selpar_selpositions(i,count) = k;
      }
    }
  }
  if(selpar_use_count(i) > 1)
  {
    int block = selpar_selblocks(i,1);
    int type = selpar_seltypes(i,1);
    int position = selpar_selpositions(i,1);
    for(int j=2; j<=selpar_use_count(i); j++)
    {
      if(selpar_seltypes(i,j) != type) // parameter is used multiple times in
          different selectivity types?
      {
        fixlog << "selectivity parameter " << i << " is specified in selectivity block
            " << block << " of type " << type <<
          " and also in selectivity block " << selpar_selblocks(i,j) << " of type " <<
              selblock_type(i,j) << endl;
        ad_exit(1);
      }
      else //sel types are the same, but now check to make sure they are in the right
          position
      {
        type = selpar_seltypes(i,j);
        if(selpar_selpositions(i,j) != position && type != 1) // only matters if
            selectivity type is not age-based
        {
          fixlog << "selectivity parameter " << i << " is specified in selectivity
              block " << block << " of type " << type <<
            " at position " << position << " and also in selectivity block " <<
```

```
              selpar_selblocks(i,j) << " of type " << selpar_seltypes(i,j) <<
            " at position " << selpar_selpositions(i,j) << endl;
          ad_exit(1);
        }
        else position = selpar_selpositions(i,j);
      }
    }
  }
}
for(int i=1; i<=n_selpars; i++)
{
  if(selpar_seltypes(i,1) == 1) //estimating proportion by age
  {
    if(selpars_lower(i) < 0.0)
    {
      selpars_lower(i) = 0.0;
      fixlog << "Changed lower bound for selectivity parameter " << i << " to 0
          because type is 1 (proportion) and it was less than 0" << endl;
    }
    if(selpars_upper(i) > 1.0 || selpars_upper(i) < selpars_lower(i))
    {
      selpars_upper(i) = 1.0;
      fixlog << "Changed upper bound for selectivity parameter " << i <<
        " to 1 because type is 1 (proportion) and it was greater than 1 or less than
           lower bound" << endl;
    }
  }
  else //either logistic or double logistic or incorrect type
  {
    //lower bound must be >= 0 for all these parameters
    if(selpars_lower(i) < 0.0)
    {
      selpars_lower(i) = 0.0;
      fixlog << "Changed lower bound for selectivity parameter " << i <<
       " to 0 because type is 2 (logistic) and this parameter was less than 0 (a50 and
           slope must be greater than 0)" << endl;
    }
    if(selpar_seltypes(i,1) == 2)// || selpar_seltypes(i,1) == 3) //logistic parameter
        a50 and increasing slope
    {
      if(selpar_selpositions(i,1) == 1)//  0<a50<n_ages
      {
        if(selpars_upper(i) > double(n_ages) || selpars_upper(i) < selpars_lower(i))
        {
        selpars_upper(i) = double(n_ages);
        fixlog << "Changed upper bound for selectivity parameter " << i <<
          " to n_ages because type is 2 (logistic) and it was greater than n_ages or
              less than lower bound and this is an a50 parameter" << endl;
        }
      }
      else //  0<slope
      {
```

```
      if(selpars_upper(i) < selpars_lower(i))
      {
        selpars_upper(i) = selpars_lower(i)+1.0;
        fixlog << "Changed upper bound for selectivity parameter " << i << " to " <<
            selpars_lower(i)+1.0 <<
          " because type is 2 (logistic) and it was less than lower bound and this
             is slope parameter" << endl;
      }
    }
  }
  else
  {
    if(selpar_seltypes(i,1) == 3) //double logistic
    {
      if(selpar_selpositions(i,1) == 1 || selpar_selpositions(i,1) == 3)//  0<a50<
          n_ages
      {
        if(selpars_upper(i) > double(n_ages) || selpars_upper(i) < selpars_lower(i))
        {
          selpars_upper(i) = double(n_ages);
          fixlog << "Changed upper bound for selectivity parameter " << i <<
            " to n_ages because type is 3 (double logistic) and it was greater than
                n_ages or less than lower bound and this is an a50 parameter" << endl
                ;
        }
      }
      else //  0<slope
      {
        if(selpars_upper(i) < selpars_lower(i))
        {
          selpars_upper(i) = selpars_lower(i)+1.0;
          fixlog << "Changed upper bound for selectivity parameter " << i << " to "
              << selpars_lower(i)+1.0 <<
            " because type is 2 (logistic) and it was less than lower bound and this
                is slope parameter" << endl;
        }
      }
    }
    else // incorrect selectivity type
    {
      fixlog << "selectivity type for parameter " << i << " is " << selpar_seltypes(
          i,1) << " but it must be 1, 2, or 3" << endl;
      ad_exit(1);
    }
  }
}
if(selpars_ini(i) < selpars_lower(i) || selpars_ini(i) > selpars_upper(i))
{
  if(selpars_ini(i) < selpars_lower(i))
  {
    selpars_ini(i) = selpars_lower(i) + 1.0e-15;
    fixlog << "changed initial value for selectivity parameter " << i << " to " <<
```

```
        selpars_lower(i) + 1.0e-15 <<
      " because it was < the lower bound, " << selpars_lower(i) << endl;
  }
  else
  {
    selpars_ini(i) = selpars_upper(i) - 1.0e-15;
    fixlog << "changed initial value for selectivity parameter " << i << " to " <<
        selpars_upper(i) - 1.0e-15 <<
      " because it was > the upper bound, " << selpars_upper(i) << endl;
  }
}
if(selpars_penalty_type(i) == 1)
{
  //because all parameters are bounded we use a shifted and scaled beta prior/
      penalty
  double max_CV = sqrt((selpars_upper(i)-selpars_ini(i))*(selpars_ini(i) -
      selpars_lower(i))/square(selpars_ini(i)));
  if(selpars_penalty_CV(i) >= max_CV)
  { //CV(par) must be < sqrt((E(par)-lower)(upper-E(par))/E(par)^2)
    selpars_penalty_CV(i) = max_CV*0.9999;
    fixlog << "Changed penalty CV for selectivity parameter " << i << " to " <<
        max_CV*0.9999 << " because it was >= maximum and penalty type is beta" <<
        endl;
  }
  if(selpars_penalty_CV(i) < 1.0e-15)
  {
    selpars_penalty_CV(i) = max_CV*0.0001;
    fixlog << "Changed penalty CV for selectivity parameter " << i << " to " <<
        max_CV*0.0001 << " because it was <= 0 and penalty type is beta" << endl;
  }
  //beta distribution for penalties: phi = alpha + beta = mu/(1-mu)/CV^2 -1
  selpars_penalty_mu(i) = (selpars_ini(i) - selpars_lower(i))/(selpars_upper(i) -
      selpars_lower(i));
  selpars_penalty_phi(i) = (1-selpars_penalty_mu(i))*selpars_penalty_mu(i)*square(
      selpars_upper(i) - selpars_lower(i))/square(selpars_ini(i)*selpars_penalty_CV(i
      )) - 1.0;
  selpars_penalty_a(i) = selpars_penalty_phi(i)*selpars_penalty_mu(i);
  selpars_penalty_b(i) = selpars_penalty_phi(i)*(1- selpars_penalty_mu(i));
  selpars_penalty_const(i) = (selpars_penalty_phi(i)-1.0)*log(selpars_upper(i)-
      selpars_lower(i) + 1.0e-15) -
        gammln(selpars_penalty_phi(i)) + gammln(selpars_penalty_a(i)) + gammln(
            selpars_penalty_b(i));
}
else
{
  //we use a truncated log-normal essentially the same as asap3 (just add a constant
      )
  if(selpars_penalty_CV(i) < 1.0e-15)
  {
    fixlog << "Changed selpars_penalty_CV(" << i << ") to 100" << endl;
    selpars_penalty_CV(i) = CVfill;
  }
```

```
      selpars_penalty_sigma(i) = sqrt(log(selpars_penalty_CV(i)*selpars_penalty_CV(i)
          +1.0));
      selpars_penalty_lnorm_scale(i) = cumd_norm((log(selpars_upper(i))-log(selpars_ini(
          i)+1.0e-15))/selpars_penalty_sigma(i)) -
        cumd_norm((log(selpars_lower(i)+1.0e-15)-log(selpars_ini(i)+1.0e-15))/
            selpars_penalty_sigma(i));
      selpars_penalty_const(i) = 0.5*log(2.0*PI) + log(selpars_penalty_sigma(i)) + log(
          selpars_penalty_lnorm_scale(i));
    }
  }
END_CALCS

  init_int Freport_agemin
  !! ICHECK(Freport_agemin);
  init_int Freport_agemax
  !! ICHECK(Freport_agemax);
  init_int Freport_wtopt
  !! ICHECK(Freport_wtopt);
  init_number Fmult_max_value_ini
  !! ICHECK(Fmult_max_value_ini);
  init_int use_Fmult_max_penalty; //1=yes, 0 = no
  !! ICHECK(use_Fmult_max_penalty);
  init_int use_F_penalty; //1=yes, 0 = no
  !! ICHECK(use_F_penalty);
  init_int nXSPR
  !! ICHECK(nXSPR);
  init_vector XSPR(1,nXSPR) //percentage(s) of SPR to use for reference point(s) must be
       between 0 and 100
  !! ICHECK(XSPR);
  init_int n_SR_scalar_pars // same approach as steepness, generally don't want to do
     both at the same time
  !! ICHECK(n_SR_scalar_pars);
  init_vector SR_scalar_pars_ini(1,n_SR_scalar_pars)
  !! ICHECK(SR_scalar_pars_ini);
  init_matrix SR_scalar_X(1,n_years,1,n_SR_scalar_pars)
  !! ICHECK(SR_scalar_X);
  init_int n_steepness_pars // need at least one parameter for mean, this has a column
     of 1's in design matrix
  !! ICHECK(n_steepness_pars);
  init_vector steepness_pars_ini(1,n_steepness_pars)
  !! ICHECK(steepness_pars_ini);
  init_matrix steepness_X(1,n_years,1,n_steepness_pars)
  !! ICHECK(steepness_X);
  init_int SR_ratio_0_type //1 for first year, 2 for last year, 3 for annual
  !! ICHECK(SR_ratio_0_type);
  init_int SR_model_type //1 for ASAP3 methods, else use just use R0 without using
     SR_ration_0 for average recruitment and estimate deviations
  !! ICHECK(SR_model_type);

  init_number ignore_guesses
  !! ICHECK(ignore_guesses);
```

```
 // used in calculation of slope for F_01 reference points
 number delta
 !! delta=0.00001;

 // Projection Info *********************
 init_int do_projections
 !! ICHECK(do_projections);
 init_ivector directed_fleet(1,n_fleets)
 !! ICHECK(directed_fleet);
 init_number nfinalyear
 !! ICHECK(nfinalyear);
 int nprojyears
 !! nprojyears=nfinalyear-year1-n_years+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 (int y=1; y<=nprojyears; y++)
 {
   proj_recruit(y)=project_ini(y,2);
   proj_what(y)=project_ini(y,3);
   proj_target(y)=project_ini(y,4);
   proj_F_nondir_mult(y)=project_ini(y,5);
 }
END_CALCS

 // MCMC Info *******************************
 init_int doMCMC
 !! ICHECK(doMCMC);
LOCAL_CALCS
 if (doMCMC == 1)
 {
   basicMCMC << " ";
   for (int y=1; y<=n_years; y++)
   {
     basicMCMC << "F_" << y+year1-1 << " ";
   }
   for (int y=1; y<=n_years; y++)
   {
     for (int a=1; a<=n_ages; a++)
     {
       basicMCMC << "M_" << a << "_" << y << " ";
     }
   }
   for (int y=1; y<=n_years; y++)
   {
     basicMCMC << "SSB_" << y+year1-1 << " ";
   }
   // Liz added Fmult_in lastyear and totBjan1
```

```
    for (int y=1; y<=n_years; y++)
    {
      basicMCMC << "Fmult_" << y+year1-1 << " ";
    }
    for (int y=1; y<=n_years; y++)
    {
      basicMCMC << "totBjan1_" << y+year1-1 << " ";
    }
    for(int i=1; i<=nXSPR; i++)
    {
      for (int y=1; y<=n_years; y++)
      {
        basicMCMC << "F_" << XSPR(i) << "_" << y+year1-1 << " ";
      }
    }
    for (int y=1; y<=n_years; y++)
    {
      basicMCMC << "MSY_" << y+year1-1 << " ";
    }
    for (int y=1; y<=n_years; y++)
    {
      basicMCMC << "SSB_MSY_" << y+year1-1 << " ";
    }
    for (int y=1; y<=n_years; y++)
    {
      basicMCMC << "F_MSY_" << y+year1-1 << " ";
    }
    for (int y=1; y<=n_years; y++)
    {
      basicMCMC << "SSB_MSY_ratio_" << y+year1-1 << " ";
    }
    for (int y=1; y<=n_years; y++)
    {
      basicMCMC << "F_MSY_ratio_" << y+year1-1 << " ";
    }
    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;

//************************************************************************
PARAMETER_SECTION

  !! cout << "begin parameter section" << endl;
  init_bounded_number_vector selpars(1,n_selpars,selpars_lower,selpars_upper,
     phase_selpars)
  init_number_vector log_Fmult_year1(1,n_fleets,phase_Fmult_year1)
  init_number_vector log_Fmult_devs_ini(1,n_tot_Fmult_devs,phase_Fmult_devs_ini)
  init_bounded_dev_vector log_recruit_devs(1,n_years,-15.,15.,phase_recruit_devs)
  init_bounded_vector log_N_year1_devs(2,n_ages,-15.,15.,phase_N_year1_devs)
  init_number_vector M_year_pars(1,n_M_year_cov,phase_M_year_pars)
  init_number_vector M_age_pars(1,n_M_age_cov,phase_M_age_pars)
  init_number_vector availability_pars_estimate(1,total_availability_pars,
     phase_availability_pars) //ragged matrix
  init_number_vector efficiency_pars_estimate(1,total_efficiency_pars,
     phase_efficiency_pars) //ragged matrix
  init_vector_vector availability_AR1_devs(1,n_indices,index_min_time+1,index_max_time,
     phase_availability_AR1)
  init_vector SR_scalar_pars(1,n_SR_scalar_pars,phase_SR_scalar)
  init_vector steepness_pars(1,n_steepness_pars,phase_steepness)
  init_vector_vector rel_efficiency_coef_devs(1,n_rel_efficiency_penalties,1,
     n_rel_efficiency_coef,phase_rel_efficiency)
  matrix log_Fmult_devs(1,n_fleets,catch_min_time+1,catch_max_time)
  vector log_SR_scalar(1,n_years)
  vector steepness(1,n_years)
  matrix log_Fmult(1,n_fleets,1,n_years)
  matrix Fmult(1,n_fleets,1,n_years)
  matrix M_use(1,n_years,1,n_ages)
  matrix NAA(1,n_years,1,n_ages)
  matrix FAA_tot(1,n_years,1,n_ages)
  matrix Z(1,n_years,1,n_ages)
  matrix S(1,n_years,1,n_ages)
  matrix catch_tot_stdresid(1,n_fleets,1,n_years)
  matrix discard_tot_stdresid(1,n_fleets,1,n_years)
  matrix catch_tot_fleet_pred(1,n_fleets,1,n_years)
```

```
matrix discard_tot_fleet_pred(1,n_fleets,1,n_years)
3darray CAA_pred(1,n_fleets,1,n_years,1,n_ages)
3darray discard_pred(1,n_fleets,1,n_years,1,n_ages)
3darray catch_paa_pred(1,n_fleets,1,n_years,1,n_ages)
3darray discard_paa_pred(1,n_fleets,1,n_years,1,n_ages)
3darray directed_FAA_by_fleet(1,n_fleets,1,n_years,1,n_ages)
3darray FAA_by_fleet_discard(1,n_fleets,1,n_years,1,n_ages)
matrix selectivity_blocks(1,n_selblocks,1,n_ages)
3darray sel_by_fleet(1,n_fleets,1,n_years,1,n_ages)
3darray sel_by_index(1,n_indices,1,n_years,1,n_ages)
matrix availability_pars(1,n_indices,1,n_availability_pars)
matrix availability(1,n_indices,1,n_years)
vector availability_stdresid(1,n_indices)
vector availability_rmse(1,n_indices)
matrix efficiency_pars(1,n_indices,1,n_efficiency_pars)
matrix efficiency(1,n_indices,1,n_years)
vector efficiency_stdresid(1,n_indices)
vector efficiency_rmse(1,n_indices)
matrix q_by_index(1,n_indices,1,n_years)
matrix rel_efficiency_coef(1,n_rel_efficiency_penalties,1,n_rel_efficiency_coef) //
   ragged matrix
3darray calibrated_naa_obs(1,n_indices,1,n_years,1,n_ages);
3darray calibrated_paa_obs(1,n_indices,1,n_years,1,n_ages);
3darray index_paa_obs_use(1,n_indices,1,n_years,1,n_ages);
matrix calibrated_index_obs(1,n_indices,1,n_years);

matrix index_pred(1,n_indices,1,n_years)
matrix index_stdresid(1,n_indices,1,n_years)
matrix output_Neff_index(1,n_indices,1,n_years)
3darray index_paa_pred(1,n_indices,1,n_years,1,n_ages)
vector SR_S0(1,n_years)
vector SR_R0(1,n_years)
vector SR_alpha(1,n_years)
vector SR_beta(1,n_years)
vector SR_pred_recruits(1,n_years+1)
vector SR_stdresid(1,n_years)
vector selpars_stdresid(1,n_selpars)
number lambda_Fmult_max_penalty
number Fmult_max_penalty
number fpenalty
number lambda_fpenalty
vector Fmult_year1_stdresid(1,n_fleets)
matrix Fmult_devs_stdresid(1,n_fleets,catch_min_time+1,catch_max_time)
vector N_year1_stdresid(1,n_ages)
matrix availability_AR1_stdresid(1,n_indices,1,index_time_span)//ragged matrix
number steepness_stdresid
number SR_scalar_stdresid
matrix output_Neff_catch(1,n_fleets,1,n_years)
matrix output_Neff_discard(1,n_fleets,1,n_years)
vector Neff_stage2_mult_catch(1,n_fleets)
vector Neff_stage2_mult_discard(1,n_fleets)
vector Neff_stage2_mult_index(1,n_indices)
```

```
vector mean_age_obs(1,n_years)
vector mean_age_pred(1,n_years)
vector mean_age_pred2(1,n_years)
vector mean_age_resid(1,n_years)
vector mean_age_sigma(1,n_years)
number mean_age_x
number mean_age_n
number mean_age_delta
number mean_age_mean
number mean_age_m2
number Fref_report
number Fref
vector Freport_U(1,n_years)
vector Freport_N(1,n_years)
vector Freport_B(1,n_years)
matrix SSBfracZ(1,n_years,1,n_ages)
vector final_year_total_sel(1,n_ages)
matrix directed_F(1,n_years,1,n_ages)
matrix nondirected_F(1,n_years,1,n_ages)
matrix discard_F(1,n_years,1,n_ages)
matrix directed_sel(1,n_years,1,n_ages)
matrix discard_sel(1,n_years,1,n_ages)
vector proj_directed_F(1,n_ages)
vector proj_nondirected_F(1,n_ages)
vector proj_discard_F(1,n_ages)
vector proj_directed_sel(1,n_ages)
vector proj_discard_sel(1,n_ages)
matrix proj_NAA(1,nprojyears,1,n_ages)
vector proj_Fmult(1,nprojyears)
vector proj_TotJan1B(1,nprojyears)
vector proj_SSB(1,nprojyears)
matrix proj_F_dir(1,nprojyears,1,n_ages)
matrix proj_F_discard(1,nprojyears,1,n_ages)
matrix proj_F_nondir(1,nprojyears,1,n_ages)
matrix proj_Z(1,nprojyears,1,n_ages)
matrix proj_SSBfracZ(1,nprojyears,1,n_ages)
matrix proj_catch(1,nprojyears,1,n_ages)
matrix proj_discard(1,nprojyears,1,n_ages)
matrix proj_yield(1,nprojyears,1,n_ages)
vector proj_total_yield(1,nprojyears)
vector proj_total_discard(1,nprojyears)
vector NAAbsn(1,n_ages)
number SPR_Fmult
number YPR_Fmult
number SPR
number SPRatio
number YPR
number S_F
number R_F
number Fmult_max_value
number slope_origin
number slope
```

```
vector SR_ratio_0(1,n_years)
//This allows user to specify what percentage(s) of SPR to make calculations for.
matrix FXSPR(1,nXSPR,1,n_years)
vector Fmsy(1,n_years)
vector F01(1,n_years)
vector Fmax(1,n_years)
//This allows user to specify what percentage(s) of SPR to make calculations for.
matrix FXSPR_report(1,nXSPR,1,n_years)
vector F01_report(1,n_years)
vector Fmax_report(1,n_years)
vector Fcurrent(1,n_years)
//This allows user to specify what percentage(s) of SPR to make calculations for.
matrix FXSPR_slope(1,nXSPR,1,n_years)
vector Fmsy_slope(1,n_years)
vector F01_slope(1,n_years)
vector Fmax_slope(1,n_years)
vector F_slope(1,n_years)
vector SSBmsy(1,n_years)

sdreport_vector logit_q(1,n_tot_index_years)
sdreport_vector Freport(1,n_years)
sdreport_vector TotJan1B(1,n_years)
sdreport_vector SSB(1,n_years)
sdreport_vector ExploitableB(1,n_years)
sdreport_vector recruits(1,n_years)
sdreport_vector MSY(1,n_years)
sdreport_vector SSBmsy_report(1,n_years)
sdreport_vector Fmsy_report(1,n_years)
sdreport_vector SSBmsy_ratio(1,n_years)
sdreport_vector Fmsy_ratio(1,n_years)
sdreport_vector catch_tot_likely(1,n_fleets)
sdreport_vector discard_tot_likely(1,n_fleets)
sdreport_vector catch_age_comp_likely(1,n_fleets)
sdreport_vector discard_age_comp_likely(1,n_fleets)
sdreport_vector index_likely(1,n_indices)
sdreport_vector index_age_comp_likely(1,n_indices)
sdreport_vector selpars_penalty(1,n_selpars)
sdreport_number SR_penalty
sdreport_vector Fmult_year1_penalty(1,n_fleets)
sdreport_vector Fmult_devs_penalty(1,n_fleets)
sdreport_number N_year1_penalty
sdreport_vector availability_penalty(1,n_indices)
sdreport_vector efficiency_penalty(1,n_indices)
sdreport_vector rel_efficiency_penalty(1,n_rel_efficiency_penalties)
sdreport_vector availability_AR1_penalty(1,n_indices)
sdreport_number steepness_penalty
sdreport_number SR_scalar_penalty

vector NAA_year1_pred(1,n_ages)

objective_function_value obj_fun
!! cout << "end parameter section" << endl;
```

```
PRELIMINARY_CALCS_SECTION
  // subset only used index information
  cout << "begin preliminary calcs" << endl;
  lambda_fpenalty = 0.0;
  fpenalty = 0.0;
  Fmult_max_penalty = 0.0;
  Fmult_max_value = Fmult_max_value_ini;
  if (ignore_guesses==0)
  {
    NAA(1)=NAA_year1_ini;
    for(int i=1; i<=n_SR_scalar_pars; i++) SR_scalar_pars(i) = SR_scalar_pars_ini(i);
    for(int i=1; i<=n_steepness_pars; i++) steepness_pars(i) = steepness_pars_ini(i);
    for (int i=1; i<=n_selpars; i++)
    {
      selpars(i) = selpars_ini(i);
    }
    for(int i=1;i<=n_fleets;i++) log_Fmult_year1(i)=log(Fmult_year1_ini(i));
    for(int i=1;i<=total_efficiency_pars;i++) efficiency_pars_estimate(i) =
        efficiency_pars_ini(i);
    for(int i=1;i<=total_availability_pars;i++) availability_pars_estimate(i) =
        availability_pars_ini(i);
    for(int i=1;i<=n_M_year_cov;i++) M_year_pars(i) = M_year_pars_ini(i);
    for(int i=1;i<=n_M_age_cov;i++) M_age_pars(i) = M_age_pars_ini(i);
  }

  // set dev vectors to zero
  for(int i=1;i<=n_rel_efficiency_penalties;i++) rel_efficiency_coef_devs(i).initialize
      ();
  for(int i=1;i<=n_tot_Fmult_devs;i++) log_Fmult_devs_ini(i).initialize();
  log_recruit_devs.initialize();
  log_N_year1_devs.initialize();
  for(int i=1;i<=n_indices;i++) availability_AR1_devs(i).initialize();
  for(int i=1;i<=n_indices;i++) index_paa_obs_use(i) = index_paa_obs(i);
  // 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
  cout << "end preliminary calcs" << endl;

//************************************************************************************************

PROCEDURE_SECTION
                                                              if (debug==1) cout << "
                                                                 starting procedure
                                                                 section" << endl;
  get_selectivity();                                          if (debug==1) cout << "
      got selectivity" << endl;
```

```
  get_catchability();                                          if (debug==1) cout << "
     got catchability" << endl;
  get_mortality_rates();                                       if (debug==1) cout << "
     got mortality rates" << endl;
  get_SR_ratio_0();                                            if (debug==1) cout << "
     got unexploited spawners per recruit" << endl;
  get_SR();                                                    if (debug==1) cout << "
     got SR" << 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;
  if(calibrate_indices == 1)
  {
    get_calibrated_indices();                                  if (debug==1) cout << "
       got calibrated indices" << 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_directed_and_discard_sel();                            if (debug==1) cout <<"
       got proj sel" << endl;
    get_Fref();                                                if (debug==1) cout <<"
       got Fref" << endl;
    get_multinomial_multiplier_catch_discards();               if (debug==1) cout <<"
       got multinomial multiplier for catch/discards" << endl;
    if(calibrate_indices == 1)
    {
      get_multinomial_multiplier_indices_with_calibration();   if (debug==1) cout <<"
         got multinomial multiplier for index with calibration" << endl;
    }
    else
    {
      get_multinomial_multiplier_indices();                    if (debug==1) cout <<"
         got multinomial multiplier for index" << endl;
    }
  }
  if (mceval_phase())
  {
    write_MCMC();
  }                                                            if (debug==1) cout << "
      . . . end of procedure section" << endl;
//*****************************************************************************************************

FUNCTION get_selectivity
  dvariable a50_1 = 0.0;
```

```
dvariable k_1 = 0.0;
dvariable a50_2 = 0.0;
dvariable k_2 = 0.0;

for (int i=1; i<=n_selblocks; i++)
{
  if (selblock_type(i)==1)
  { //proportions at age
    for (int a=1; a<=n_ages; a++)
    {
      selectivity_blocks(i,a)=selpars(selpars_pointer(i,a));
    }
  }
  else
  { //logistic or double-logistic
    a50_1 = selpars(selpars_pointer(i,1)); // a50 parameter
    k_1 = selpars(selpars_pointer(i,2)); //  1/slope
    if (selblock_type(i)==2)
    { //increasing logistic
      for (int a=1; a<=n_ages; a++)
      {
        selectivity_blocks(i,a) = 1.0/(1.0 + mfexp(-(double(a) - a50_1)/k_1));
      }
      selectivity_blocks(i) /= max(selectivity_blocks(i));
    }
    else
    {  //double logistic
      a50_2 = selpars(selpars_pointer(i,3));
      k_2 = selpars(selpars_pointer(i,4));
      for (int a=1; a<=n_ages; a++)
      {
        selectivity_blocks(i,a) = 1.0/(1.0 + mfexp(-(double(a) - a50_1)/k_1));
        selectivity_blocks(i,a) *= 1.0/(1.0 + mfexp((double(a) - a50_2)/k_2)); //1-p
      }
      selectivity_blocks(i) /= max(selectivity_blocks(i));
    }
  }
}

// now fill in selectivity for each fleet and year according to block
for (int i=1; i<=n_fleets; i++)
{
  for (int y=1; y<=n_years; y++)
  {
    sel_by_fleet(i,y)=selectivity_blocks(fleet_selblock_pointer(i,y));
  }
}
// now fill in selectivity for each index and year according to block
for (int i=1; i<=n_indices; i++)
{
  for (int y=1; y<=n_years; y++)
  {
```

```
      sel_by_index(i,y)=selectivity_blocks(index_selblock_pointer(i,y));
    }
  }

FUNCTION get_mortality_rates
  // compute directed and discard F by fleet then sum to form total F at age matrix
  FAA_tot=0.0;
  log_Fmult = -1000.0; //so F is ~0 in years where no catch exists
  int counter = 1;
  for (int i=1; i<=n_fleets; i++)
  {
    log_Fmult(i,catch_min_time(i))=log_Fmult_year1(i);
    if (current_phase() >= phase_Fmult_devs(i))
    {
      for (int y=2; y<=n_catch_years(i); y++)
      {
        log_Fmult_devs(i,catch_times(i,y)) = log_Fmult_devs_ini(counter);
        counter++;
        log_Fmult(i,catch_times(i,y)) = log_Fmult(i,catch_times(i,y-1))+log_Fmult_devs(i
            ,catch_times(i,y));
      }
    }
    else
    {
      for(int y = 2; y <= n_catch_years(i); y++)
      log_Fmult(i,catch_times(i,y)) = log_Fmult_year1(i);
    }
    for (int y=1; y<=n_years; y++)
    {
      directed_FAA_by_fleet(i,y)=elem_prod(mfexp(log_Fmult(i,y))*sel_by_fleet(i,y),1.0-
          proportion_release(i,y));
      FAA_by_fleet_discard(i,y)=elem_prod(mfexp(log_Fmult(i,y))*sel_by_fleet(i,y),
          proportion_release(i,y)*release_mort(i));
    }
    FAA_tot += directed_FAA_by_fleet(i)+FAA_by_fleet_discard(i);
  }
  // add fishing and natural mortality to get total mortality
  if(estimate_M == 1)
  {
    for (int y=1; y<=n_years; y++)
    {
      M_use(y) = mfexp(M_X_year(y)*M_year_pars);
      for(int a=1; a<=n_ages; a++) M_use(y,a) *= mfexp(M_X_age(a)*M_age_pars);
    }
  }
  else M_use = M_ini;

  Z=FAA_tot+M_use;

  S=mfexp(-1.0*Z);
  SSBfracZ=mfexp(-1.0*fracyearSSB*Z); // for use in SSB calcuations
```

```
FUNCTION get_SR_ratio_0
  // need to do this inside model because M might be estimated
  SR_ratio_0=0.0;
  for (int y=1; y<=n_years; y++)
  {
    dvariable ntemp0=1.0;
    for (int a=1; a<n_ages; a++)
    {
      SR_ratio_0(y)+=ntemp0*fecundity(y,a)*mfexp(-1.0*fracyearSSB*M_use(y,a));
      ntemp0*=mfexp(-M_use(y,a));
    }
    ntemp0/=(1.0-mfexp(-M_use(y,n_ages)));
    SR_ratio_0(y)+=ntemp0*fecundity(y,n_ages)*mfexp(-1.0*fracyearSSB*M_use(y,n_ages));
  }

FUNCTION get_SR
  // converts stock recruitment scalar and steepness to alpha and beta for Beverton-Holt
      SR
  // note use of is_SR_scalar_R variable to allow user to enter guess for either R0 or
     SSB0
  steepness = 0.2 + (1.0-0.2)/(1 + exp(-steepness_X*steepness_pars));
  log_SR_scalar = SR_scalar_X * SR_scalar_pars;
  // now compute year specific vectors of R0, S0, and steepness
  if(is_SR_scalar_R==1)
  {
    SR_R0=mfexp(log_SR_scalar);
    if(SR_ratio_0_type == 1) SR_S0 = SR_ratio_0(1)*SR_R0;
    else if(SR_ratio_0_type == 2) SR_S0 = SR_ratio_0(n_years)*SR_R0;
    else SR_S0 = elem_prod(SR_ratio_0,SR_R0);
  }
  else
  {
    SR_S0=mfexp(log_SR_scalar);
    if(SR_ratio_0_type == 1) SR_R0 = SR_S0/SR_ratio_0(1);
    else if(SR_ratio_0_type == 2) SR_R0 = SR_S0/SR_ratio_0(n_years);
    else SR_R0=elem_div(SR_S0,SR_ratio_0);
  }
  SR_alpha=4.0*elem_div(elem_prod(steepness,SR_R0),5.0*steepness-1.0);
  SR_beta=elem_div(elem_prod(SR_S0,1.0-steepness),5.0*steepness-1.0);

FUNCTION get_numbers_at_age

  // get N at age in year 1 other than recruits
  for (int a=2; a<=n_ages; a++)
    NAA(1,a)=NAA_year1_ini(a)*mfexp(log_N_year1_devs(a));
  // compute initial SSB to derive R in first year
  SSB(1)=0.0;
  if(SR_model_type == 1)
  { //Using the stock recruit parameters to determine annual recruitment, even when
      steepness is 1.
    for (int a=2; a<=n_ages; a++)
    {
```

```
      SSB(1)+=NAA(1,a)*SSBfracZ(1,a)*fecundity(1,a);   // note SSB in year 1 does not
          include age 1 to estimate pred_R in year 1
    }
    SR_pred_recruits(1)=SR_alpha(1)*SSB(1)/(SR_beta(1)+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
    for (int y=2; y<=n_years; y++)
    {
      SR_pred_recruits(y)=SR_alpha(y-1)*SSB(y-1)/(SR_beta(y-1)+SSB(y-1));
      NAA(y,1)=SR_pred_recruits(y)*mfexp(log_recruit_devs(y));
      for (int a=2; a<=n_ages; a++) NAA(y,a)=NAA(y-1,a-1)*S(y-1,a-1);
      NAA(y,n_ages)+=NAA(y-1,n_ages)*S(y-1,n_ages);
             // compute spawning biomass time series
      SSB(y)=elem_prod(NAA(y),SSBfracZ(y))*fecundity(y);
    }
  }
  else
  { // annual recruits are just devations from average recruitment determined by
      SR_scalar (possibly with annual covariates).
    NAA(1,1) = mfexp(log_SR_scalar(1)+log_recruit_devs(1));
    SSB(1) = elem_prod(NAA(1),SSBfracZ(1)) * fecundity(1);
    for (int y=2; y<=n_years; y++)
    {
      NAA(y,1) = mfexp(log_SR_scalar(y)+log_recruit_devs(y));
      for (int a=2; a<=n_ages; a++) NAA(y,a)=NAA(y-1,a-1)*S(y-1,a-1);
      NAA(y,n_ages)+=NAA(y-1,n_ages)*S(y-1,n_ages);
             // compute spawning biomass time series
      SSB(y)=elem_prod(NAA(y),SSBfracZ(y))*fecundity(y);
    }
  }
  // compute the other two biomass time series
  for (int y=1; y<=n_years; y++)
  {
    TotJan1B(y)=NAA(y)*WAAjan1b(y);
    ExploitableB(y)=elem_prod(NAA(y),FAA_tot(y))*WAAcatchall(y)/max(FAA_tot(y));
  }
  SR_pred_recruits(n_years+1)=SR_alpha(n_years)*SSB(n_years)/(SR_beta(n_years)+SSB(
      n_years));
  recruits=column(NAA,1);

FUNCTION get_Freport
  // calculates an average F for a range of ages in each year under three weighting
      schemes
  for (int y=1; y<=n_years; y++)
  {
    dvariable temp = 0.0;
    if (Freport_wtopt==1) Freport(y)=mean(FAA_tot(y)(Freport_agemin,Freport_agemax));
    else
    {
      if (Freport_wtopt==2)
      {
```

```
        temp = sum(NAA(y)(Freport_agemin,Freport_agemax));
        Freport(y) = sum(elem_prod(FAA_tot(y),NAA(y))(Freport_agemin,Freport_agemax));
      }
      else if (Freport_wtopt==3)
      {
        temp = sum(elem_prod(NAA(y),WAAjan1b(y))(Freport_agemin,Freport_agemax));
        Freport(y) = sum(elem_prod(elem_prod(FAA_tot(y),NAA(y)),WAAjan1b(y)));
      }
      if(temp <= 1.0e-15) Freport(y)=mean(FAA_tot(y)(Freport_agemin,Freport_agemax));
      else Freport(y) /= temp;
    }
  }

FUNCTION get_predicted_catch
  // assumes continuous F using Baranov equation
  catch_tot_fleet_pred=0.0;
  for (int i=1; i<=n_fleets; i++)
  {
    CAA_pred(i)=elem_prod(elem_div(directed_FAA_by_fleet(i),Z),elem_prod(1.0-S,NAA));
    discard_pred(i)=elem_prod(elem_div(FAA_by_fleet_discard(i),Z),elem_prod(1.0-S,NAA));
    catch_tot_fleet_pred(i)=rowsum(CAA_pred(i));
    discard_tot_fleet_pred(i)=rowsum(discard_pred(i));
    catch_paa_pred(i)=0.0;
    discard_paa_pred(i)=0.0;
    // now compute proportions at age and total weight of catch
    for (int y=1; y<=n_years;y++)
    {
      if (catch_tot_fleet_pred(i,y)>0.0) catch_paa_pred(i,y)=CAA_pred(i,y)/
          catch_tot_fleet_pred(i,y);
      if (discard_tot_fleet_pred(i,y)>0.0) discard_paa_pred(i,y)=discard_pred(i,y)/
          discard_tot_fleet_pred(i,y);
      catch_tot_fleet_pred(i,y)=CAA_pred(i,y)*WAAcatchfleet(i,y);
      discard_tot_fleet_pred(i,y)=discard_pred(i,y)*WAAdiscardfleet(i,y);
    }
  }

FUNCTION get_catchability

  //set up q's using availability and efficiency models
        q_by_index = 0.0;
        int counter1 = 0;
        int counter2 = 0;
        availability_pars = 0.0;
        availability = 0.0;
        efficiency = 0.0;
        q_by_index = 0.0;
        dvariable AR1 = 0.0;
  int kk = 0;
        logit_q = 0.0;
        for(int i = 1; i <= n_indices; i++)
        {
    for(int j = 1; j <= n_availability_pars(i); j++)
```

```
{
  counter1++;
  availability_pars(i,j) = availability_pars_estimate(counter1);
}
// catchability for each index, can be a log-AR(1) process. random walk can also be
    achieved with X(i) = 1,0,0,...
// when using this care must be taken to limit process to span of years observed in
    the given index.
if (active(availability_AR1_devs(i)))
{
  AR1 = availability_X(i,index_min_time(i))*availability_pars(i);
  availability(i,index_min_time(i)) = availability_lower(i) + (availability_upper(i)
      - availability_lower(i))/(1.0 + mfexp(-AR1));
  for (int y=index_min_time(i)+1; y<=index_max_time(i); y++)
  {
    AR1 += availability_X(i,y)*availability_pars(i) + availability_AR1_devs(i,y);
    availability(i,y) = availability_lower(i) + (availability_upper(i) -
        availability_lower(i))/(1.0 + mfexp(-AR1));
  }
}
else
{
  if(n_index_years(i) > 0)
  {
    for(int y = 1; y <= n_index_years(i); y++)
    {
      int this_y = index_times(i,y);
      availability(i,this_y) = (availability_lower(i) +
        (availability_upper(i) - availability_lower(i))/(1.0 + mfexp(-availability_X
            (i,this_y)*availability_pars(i))));
    }
  }
}

for(int j = 1; j <= n_efficiency_pars(i); j++)
{
  counter2++;
  efficiency_pars(i,j) = efficiency_pars_estimate(counter2);
}
if(n_index_years(i) > 0)
{
  for(int y = 1; y <= n_index_years(i); y++)
  {
    int this_y = index_times(i,y);
    efficiency(i,this_y) = (efficiency_lower(i) +
      (efficiency_upper(i) - efficiency_lower(i))/(1.0 + mfexp(-efficiency_X(i,
          this_y)*efficiency_pars(i))));
  }
}
q_by_index(i)= elem_prod(availability(i), efficiency(i));
if(sd_phase() && n_index_years(i) > 0)
{
```

```
      for(int y = 1; y <= n_index_years(i); y++)
      {
        logit_q(kk + y) = log(q_by_index(i,index_times(i,y)) - availability_lower(i)*
            efficiency_lower(i)) -
        log(availability_upper(i)*efficiency_upper(i) - q_by_index(i,index_times(i,y)));
      }
      kk += n_index_years(i);
    }
        }

FUNCTION get_calibrated_indices

        //for (AIV:HBB) length-based relative catch efficiency
        calibrated_index_obs = 0.0;
        dvar_matrix log_rel_efficiency(1,n_rel_efficiency_penalties,1,n_lengths);
        for(int i=1; i<= n_rel_efficiency_penalties; i++)
        {
    rel_efficiency_coef(i) = rel_efficiency_coef_ini(i) + rel_efficiency_coef_devs(i);
    log_rel_efficiency(i) = rel_efficiency_X(i) * rel_efficiency_coef(i);
        }
        dvar_vector calibrated_nal_obs(1,n_lengths);
  for(int i=1; i<=n_indices; i++)
  {
    calibrated_naa_obs(i) = 0.0;
    calibrated_paa_obs(i) = 0.0;
    if(n_index_age_comp_years(i)>0)
    {
      for(int y=1; y<=n_index_age_comp_years(i); y++)
      {
        int this_y = index_age_comp_times(i,y);
        if(calibrate_this_obs(i,this_y) > 0)
        {
          calibrated_nal_obs = elem_prod(uncalibrated_index_at_len_obs(i,this_y),mfexp(
              log_rel_efficiency(calibrate_this_obs(i,this_y))));
          calibrated_index_obs(i,this_y) = sum(calibrated_nal_obs);
          calibrated_naa_obs(i,this_y) = calibrated_nal_obs * age_length_keys(i,this_y);
          calibrated_paa_obs(i,this_y) = calibrated_naa_obs(i,this_y)/sum(
              calibrated_naa_obs(i,this_y));
          index_paa_obs_use(i,this_y) = calibrated_paa_obs(i,this_y);
        }
      }
    }
  }

FUNCTION get_predicted_indices

  // get selectivity for each index
  dvar_matrix temp_NAA(1,n_years,1,n_ages);
  temp_NAA = 0.0;
  int this_y = 0;
  for (int i=1; i<=n_indices; i++)
  {
```

```
    index_pred(i) = 0.0;
    index_paa_pred(i) = 0.0;
    for(int y=1; y<=n_years; y++)
    {
      // determine when the index should be applied
      if (index_month(i,y)==-1) temp_NAA(y)=elem_prod(NAA(y),elem_div(1.0-S(y),Z(y)));
      else temp_NAA(y)=elem_prod(NAA(y),mfexp(-1.0*((index_month(i,y)-1.0)/12.0)*Z(y)));
    }

    // compute the predicted index for each year where observed value > 0
    if (index_units_aggregate(i)==1) //biomass
    {
      temp_NAA=elem_prod(temp_NAA,index_WAA(i));
    }
    if(n_index_years(i) > 0)
    {
      for (int y=1; y<=n_index_years(i); y++)
      {
        this_y = index_times(i,y);
        index_pred(i,this_y)=q_by_index(i,this_y)*sel_by_index(i,this_y)*temp_NAA(this_y
            );
      }
    }
    // compute index proportions at age if necessary
    if (index_units_proportions(i)!=index_units_aggregate(i))//need to adjust temp_NAA
    {
      if(index_units_proportions(i) == 1) //biomass, but aggregate was numbers
        temp_NAA=elem_prod(temp_NAA,index_WAA(i));
      else //numbers, but aggregate was biomass
        temp_NAA = elem_div(temp_NAA,index_WAA(i));
    }
    if (n_index_age_comp_years(i) > 0)
    {
      for(int y = 1; y<=n_index_age_comp_years(i); y++)
      {
        this_y = index_age_comp_times(i,y);
        index_paa_pred(i,this_y)=elem_prod(q_by_index(i,this_y)*sel_by_index(i,this_y),
            temp_NAA(this_y));
        index_paa_pred(i,this_y)/=sum(index_paa_pred(i,this_y));
      }
    }
  }

FUNCTION get_directed_and_discard_sel

  // creates overall directed and discard selectivity patterns and sets bycatch F at age
  nondirected_F = 0.0;
  directed_F = 0.0;
  discard_F = 0.0;
  directed_sel = 0.0;
  discard_sel = 0.0;
  proj_nondirected_F = 0.0;
```

```
  proj_directed_F = 0.0;
  proj_discard_F = 0.0;
  proj_directed_sel=0.0;
  proj_discard_sel=0.0;
  for (int y=1; y<=n_years; y++)
  {
    for (int i=1; i<=n_fleets; i++)
    {
      if (directed_fleet(i)==1)
      {
        directed_F(y)+=directed_FAA_by_fleet(i,y);
        discard_F(y)+=FAA_by_fleet_discard(i,y);
      }
      else nondirected_F(y)+=directed_FAA_by_fleet(i,y);
    }
    directed_sel(y)=directed_F(y)/max(directed_F(y));
    discard_sel(y)=discard_F(y)/max(directed_F(y));
  }
  proj_directed_F = directed_F(n_years);
  proj_discard_F = discard_F(n_years);
  proj_nondirected_F = nondirected_F(n_years);
  proj_directed_sel =directed_sel(n_years);
  proj_discard_sel = discard_sel(n_years);

FUNCTION void get_SPR(int year)

  // simple spawners per recruit calculations for year
  dvariable ntemp=1.0;
  SPR =0.0;
  dvariable z;

  for (int age=1; age<n_ages; age++)
  {
    z = M_use(year,age)+nondirected_F(year,age) + SPR_Fmult*(directed_sel(year,age)+
        discard_sel(year,age));
    SPR += ntemp*fecundity(year,age)*mfexp(-1.0*fracyearSSB*z);
    ntemp*=mfexp(-1.0*z);
  }
  z = M_use(year,n_ages)+nondirected_F(year,n_ages)+SPR_Fmult*(directed_sel(year,n_ages)
      +discard_sel(year,n_ages));
  ntemp /= (1.0-mfexp(-1.0*z));
  SPR += ntemp*fecundity(year,n_ages)*mfexp(-1.0*fracyearSSB*z);

FUNCTION void get_Freport_ref(int year)

  // Freport calculations for each of the reference points in year
  dvariable tref=0.0;
  dvariable trefd=0.0;
  dvar_vector freftemp(1,n_ages);
  dvar_vector nreftemp(1,n_ages);
  int amin = Freport_agemin;
  int amax = Freport_agemax;
```

```
  nreftemp(1)=1.0;
  freftemp(1,n_ages-1)=(Fref*(directed_sel(year)+discard_sel(year))+nondirected_F(year))
      (1,n_ages-1);
  nreftemp(2,n_ages)= ++(mfexp(-1.0*(M_use(n_years)+freftemp)(1,n_ages-1)));
  freftemp(n_ages)=(Fref*(directed_sel(year)+discard_sel(year))+nondirected_F(year))(
      n_ages);
  nreftemp(n_ages)/=(1.0-mfexp(-1.0*(M_use(year)+freftemp)(n_ages)));

  if(Freport_wtopt==1) Fref_report=mean(freftemp(amin,amax));
  if(Freport_wtopt==2) Fref_report=freftemp(amin,amax)*nreftemp(amin,amax)/sum(nreftemp(
      amin,amax));
  if(Freport_wtopt==3) Fref_report=(elem_prod(freftemp(amin,amax),nreftemp(amin,amax)) *
       WAAjan1b(year)(amin,amax))/
         (nreftemp(amin,amax)*WAAjan1b(year)(amin,amax));

FUNCTION void get_FXSPR(int year)
  // FX%SPR calculations in year
  for(int i = 1; i <= nXSPR; i++)
  {
    dvariable A = 0.0;
    dvariable B = Fmult_max_value;
    dvariable C = 0.0;
    Fref = 0.0;
    for (int j=1; j<=20; j++)
    {
      C=(A+B)/2.0;
      SPR_Fmult=C;
      get_SPR(year);
      if(SPR/SR_ratio_0(year)< 0.01*XSPR(i)) B = C;
      else A=C;
    }
    FXSPR(i,year)=C;
    Fref=FXSPR(i,year);
    get_Freport_ref(year);
    FXSPR_report(i,year)=Fref_report;
    FXSPR_slope(i,year)=1.0/SPR;
  }

FUNCTION void get_FMSY(int year)

  // Fmsy calculations in year
  dvariable A = 0.0;
  dvariable B = Fmult_max_value;
  dvariable C = 0.0;
  Fref = 0.0;

  for (int i=1; i<=20; i++)
  {
    C=(A+B)/2.0;
    SPR_Fmult=C+delta;
    get_SPR(year);
    S_F=SR_alpha(year)*SPR-SR_beta(year);
```

```
    R_F=S_F/SPR;
    YPR_Fmult=C+delta;
    get_YPR(year);
    slope=R_F*YPR;
    SPR_Fmult=C;
    get_SPR(year);
    S_F=SR_alpha(year)*SPR-SR_beta(year);
    R_F=S_F/SPR;
    YPR_Fmult=C;
    get_YPR(year);
    slope-=R_F*YPR;
    if(slope>0.0) A=C;
    else B=C;
  }
  Fmsy(year)=C;
  Fref=Fmsy(year);
  get_Freport_ref(year);
  Fmsy_report(year)=Fref_report;
  SSBmsy(year)=S_F;
  SSBmsy_report(year)=SSBmsy(year);
  if(SSBmsy(year)>0.0) SSBmsy_ratio(year)=SSB(year)/SSBmsy(year);
  MSY(year)=YPR*R_F;
  SPR_Fmult=Fmsy(year);
  get_SPR(year);
  Fmsy_slope(year)=1.0/SPR;
  SPR_Fmult=max(FAA_tot(year)-nondirected_F(year)-discard_F(year));
  get_SPR(year);
  F_slope(year)=1.0/SPR;
  if(Fmsy(year)>0.0) Fmsy_ratio(year)=SPR_Fmult/Fmsy(year);

FUNCTION void get_F01(int year)
  //F0.1 calculations in year
  dvariable A = 0.0;
  dvariable B = Fmult_max_value;
  dvariable C = 0.0;
  Fref = 0.0;

  YPR_Fmult=delta;
  get_YPR(year);
  slope_origin=YPR/delta;
  for (int i=1; i<=20; i++)
  {
    C=(A+B)/2.0;
    YPR_Fmult=C+delta;
    get_YPR(year);
    slope=YPR;
    YPR_Fmult=C;
    get_YPR(year);
    slope-=YPR;
    slope/=delta;
    if(slope<0.10*slope_origin) B=C;
    else A=C;
```

```
  }
  F01(year)=C;
  Fref=F01(year);
  get_Freport_ref(year);
  F01_report(year)=Fref_report;
  SPR_Fmult=F01(year);
  get_SPR(year);
  F01_slope(year)=1.0/SPR;


FUNCTION void get_Fmax(int year)
  // Fmax calculations in year
  dvariable A = 0.0;
  dvariable B = Fmult_max_value;
  dvariable C = 0.0;
  Fref = 0.0;
  for (int i=1; i<=20; i++)
  {
    C=(A+B)/2.0;
    YPR_Fmult=C+delta;
    get_YPR(year);
    slope=YPR;
    YPR_Fmult=C;
    get_YPR(year);
    slope-=YPR;
    slope/=delta;
    if(slope<0.0) B=C;
    else A=C;
  }
  Fmax(year)=C;
  Fref=Fmax(year);
  get_Freport_ref(year);
  Fmax_report(year)=Fref_report;
  SPR_Fmult=Fmax(year);
  get_SPR(year);
  Fmax_slope(year)=1.0/SPR;

FUNCTION get_Fref
  //Get all of the reference points for each year
  for(int y = 1; y <= n_years; y++)
  {
    get_FXSPR(y);
    get_FMSY(y);
    get_F01(y);
    get_Fmax(y);
  }

FUNCTION void get_YPR(int year)
  // simple yield per recruit calculations
  YPR=0.0;
  dvariable ntemp=1.0;
  dvariable f = 0.0;
```

```
  dvariable z = 0.0;
  for (int age=1; age<n_ages; age++)
  {
    f = YPR_Fmult*directed_sel(year,age);
    z = M_use(year,age) + f + nondirected_F(year,age) + YPR_Fmult * discard_sel(year,age
        );
    YPR += ntemp * f * WAAcatchall(year,age) * (1.0-mfexp(-1.0*z))/z;
    ntemp*=mfexp(-1.0*z);
  }
  f = YPR_Fmult * directed_sel(year,n_ages);
  z = M_use(year,n_ages)+ f + nondirected_F(year,n_ages) + YPR_Fmult * discard_sel(year,
     n_ages);
  ntemp /= (1.0-mfexp(-1.0*z));
  YPR += ntemp * f * WAAcatchall(year,n_ages) * (1.0-mfexp(-1.0*z))/z;

FUNCTION project_into_future
  // project population under five possible scenarios for each year
  dvar_vector Ftemp(1,n_ages);
  dvar_vector Ztemp(1,n_ages);
  dvariable denom;
  for (int y=1; y<=nprojyears; y++)
  {
    proj_F_nondir(y) = proj_nondirected_F*proj_F_nondir_mult(y);
    if (proj_recruit(y)<0.0)  // use stock-recruit relationship
    {
      if (y==1) proj_NAA(y,1)=SR_alpha(n_years)*SSB(n_years)/(SR_beta(n_years)+SSB(
         n_years));
      else proj_NAA(y,1)=SR_alpha(n_years)*proj_SSB(y-1)/(SR_beta(n_years)+proj_SSB(y-1)
         );
    }
    else proj_NAA(y,1)=proj_recruit(y);

    if (y==1)
    {
      for (int a=2; a<=n_ages; a++) proj_NAA(1,a)=NAA(n_years,a-1)*S(n_years,a-1);
      proj_NAA(1,n_ages)+=NAA(n_years,n_ages)*S(n_years,n_ages);
    }
    else
    {
      for (int a=2; a<=n_ages; a++) proj_NAA(y,a)=proj_NAA(y-1,a-1)*mfexp(-1.0*proj_Z(y
         -1,a-1));
      proj_NAA(y,n_ages)+=proj_NAA(y-1,n_ages)*mfexp(-1.0*proj_Z(y-1,n_ages));
    }
    if (proj_what(y)==1) // match directed yield
    {
      proj_Fmult(y)=3.0;  // first see if catch possible
      proj_F_dir(y)=proj_Fmult(y)*proj_directed_sel;
      proj_F_discard(y)=proj_Fmult(y)*proj_discard_sel;
      proj_Z(y)=M_use(n_years)+proj_F_nondir(y)+proj_F_dir(y)+proj_F_discard(y);
      proj_catch(y)=elem_prod(elem_div(proj_F_dir(y),proj_Z(y)),elem_prod(1.0-mfexp
         (-1.0*proj_Z(y)),proj_NAA(y)));
      proj_discard(y)=elem_prod(elem_div(proj_F_discard(y),proj_Z(y)),elem_prod(1.0-
```

```
        mfexp(-1.0*proj_Z(y)),proj_NAA(y)));
    proj_yield(y)=elem_prod(proj_catch(y),WAAcatchall(n_years));
    proj_total_yield(y)=sum(proj_yield(y));
    proj_total_discard(y)=sum(elem_prod(proj_discard(y),WAAdiscardall(n_years)));
    if (proj_total_yield(y)>proj_target(y))  // if catch possible, what F needed
    {
      proj_Fmult(y)=0.0;
      for (int i=1; i<=20; i++)
      {
        Ftemp=proj_Fmult(y)*proj_directed_sel;
        denom=0.0;
        for (int a=1; a<=n_ages; a++)
        {
          Ztemp(a)=M_use(n_years,a)+proj_F_nondir(y,a)+proj_Fmult(y)*proj_discard_sel(
              a)+Ftemp(a);
          denom+=proj_NAA(y,a)*WAAcatchall(n_years,a)*proj_directed_sel(a)*(1.0-mfexp
              (-1.0*Ztemp(a)))/Ztemp(a);
        }
        proj_Fmult(y)=proj_target(y)/denom;
      }
    }
  }
  else if (proj_what(y)==2) // match F%SPR
  {
    dvariable A=0.0;
    dvariable B=5.0;
    dvariable C = 0.0;
    for (int i=1; i<=20; i++)
    {
      C=(A+B)/2.0;
      SPR_Fmult=C;
      get_SPR(n_years);
      SPRatio=SPR/SR_ratio_0(n_years);
      if (SPRatio<proj_target(y)) B=C;
      else A=C;
    }
    proj_Fmult(y)=C;
  }
  else if (proj_what(y)==3) // project Fmsy from last year
  {
    proj_Fmult=Fmsy(n_years);
  }
  else if (proj_what(y)==4) // project Fcurrent
  {
    proj_Fmult=max(FAA_tot(n_years)-nondirected_F(n_years)-discard_F(n_years));
  }
  else if (proj_what(y)==5) // project input F
  {
    proj_Fmult=proj_target(y);
  }
  proj_F_dir(y)=proj_Fmult(y)*proj_directed_sel;
  proj_F_discard(y)=proj_Fmult(y)*proj_discard_sel;
```

```
    proj_Z(y)=M_use(n_years)+proj_F_nondir(y)+proj_F_dir(y)+proj_F_discard(y);
    proj_SSBfracZ(y)=mfexp(-1.0*fracyearSSB*proj_Z(y));
    proj_catch(y)=elem_prod(elem_div(proj_F_dir(y),proj_Z(y)),elem_prod(1.0-mfexp(-1.0*
        proj_Z(y)),proj_NAA(y)));
    proj_discard(y)=elem_prod(elem_div(proj_F_discard(y),proj_Z(y)),elem_prod(1.0-mfexp
        (-1.0*proj_Z(y)),proj_NAA(y)));
    proj_yield(y)=elem_prod(proj_catch(y),WAAcatchall(n_years));
    proj_total_yield(y)=sum(proj_yield(y));
    proj_total_discard(y)=sum(elem_prod(proj_discard(y),WAAdiscardall(n_years)));
    proj_TotJan1B(y)=sum(elem_prod(proj_NAA(y),WAAjan1b(n_years)));
    proj_SSB(y)=elem_prod(proj_NAA(y),proj_SSBfracZ(y))*fecundity(n_years);
  }

FUNCTION get_multinomial_multiplier_catch_discards

  // compute Francis (2011) stage 2 multiplier for multinomial to adjust input Neff (
     method TA1.8)
  // Francis, R.I.C.C. 2011. Data weighting in statistical fisheries stock assessment
     models. CJFAS 68: 1124-1138
  Neff_stage2_mult_catch=0.0;
  Neff_stage2_mult_discard=0.0;

  for (int i=1; i<=n_fleets; i++)
  {
    if(n_catch_age_comp_years(i)>1)
        {
                dvar_vector M_obs(1,n_catch_age_comp_years(i)), M_exp(1,
                    n_catch_age_comp_years(i));
                dvar_vector R(1,n_catch_age_comp_years(i)), S(1,n_catch_age_comp_years
                    (i)), W1(1,n_catch_age_comp_years(i));

                M_obs = (catch_paa_obs(i) * double_ages)(catch_age_comp_times(i));
                M_exp = (catch_paa_pred(i) * double_ages)(catch_age_comp_times(i));
                R = M_obs - M_exp;
                S = sqrt((catch_paa_pred(i) * square(double_ages))(
                    catch_age_comp_times(i)) - square(M_exp));
                W1 = elem_div(elem_prod(R,sqrt(input_Neff_catch(i)(
                    catch_age_comp_times(i)))), S);
                dvariable W = (sum(square(W1)) - double(n_catch_age_comp_years(i)) *
                    square(mean(W1)))/(double(n_catch_age_comp_years(i)) - 1.0);
                Neff_stage2_mult_catch(i) = 1.0/W;
        }

      if(n_discard_age_comp_years(i)>1)
        {
                dvar_vector M_obs(1,n_discard_age_comp_years(i)), M_exp(1,
                    n_discard_age_comp_years(i));
                dvar_vector R(1,n_discard_age_comp_years(i)), S(1,
                    n_discard_age_comp_years(i)), W1(1,n_discard_age_comp_years(i));

                M_obs = (discard_paa_obs(i) * double_ages)(discard_age_comp_times(i));
                M_exp = (discard_paa_pred(i) * double_ages)(discard_age_comp_times(i))
```

```
                ;
            R = M_obs - M_exp;
            S = sqrt((discard_paa_pred(i) * square(double_ages))(
                discard_age_comp_times(i)) - square(M_exp));
            W1 = elem_div(elem_prod(R,sqrt(input_Neff_discard(i)(
                discard_age_comp_times(i)))), S);
            dvariable W = (sum(square(W1)) - double(n_discard_age_comp_years(i)) *
                 square(mean(W1)))/(double(n_discard_age_comp_years(i)) - 1.0);
            Neff_stage2_mult_discard(i) = 1.0/W;
        }
  }

FUNCTION get_multinomial_multiplier_indices

  // Indices
  Neff_stage2_mult_index=0.0;
  for (int i=1; i<=n_indices; i++)
  {
    if(n_index_age_comp_years(i)>1)
        {
            dvar_vector M_obs(1,n_index_age_comp_years(i)), M_exp(1,
                n_index_age_comp_years(i));
            dvar_vector R(1,n_index_age_comp_years(i)), S(1,n_index_age_comp_years
                (i)), W1(1,n_index_age_comp_years(i));

            M_obs = (index_paa_obs(i) * double_ages)(index_age_comp_times(i));
            M_exp = (index_paa_pred(i) * double_ages)(index_age_comp_times(i));
            R = M_obs - M_exp;
            S = sqrt((index_paa_pred(i) * square(double_ages))(
                index_age_comp_times(i)) - square(M_exp));
            W1 = elem_div(elem_prod(R,sqrt(input_Neff_index(i)(
                index_age_comp_times(i)))), S);
            dvariable W = sum(square(W1 - mean(W1)))/(double(
                n_index_age_comp_years(i)) - 1.0);
            Neff_stage2_mult_index(i) = 1.0/W;
        }
  }

FUNCTION get_multinomial_multiplier_indices_with_calibration

  // Indices
  Neff_stage2_mult_index=0.0;
  int this_y = 0;
  for (int i=1; i<=n_indices; i++)
  {
    if(n_index_age_comp_years(i)>1)
        {
            dvar_vector M_obs(1,n_index_age_comp_years(i)), M_exp(1,
                n_index_age_comp_years(i));
            dvar_vector R(1,n_index_age_comp_years(i)), S(1,n_index_age_comp_years
                (i)), W1(1,n_index_age_comp_years(i));
            for(int y=1; y<=n_index_age_comp_years(i); y++)
```

```
                {
        this_y=index_age_comp_times(i,y);
        if(calibrate_this_obs(i,this_y) > 0) M_obs(y) = (calibrated_paa_obs(i,this_y) *
            double_ages);
        else M_obs(y) = (index_paa_obs(i,this_y) * double_ages);
                }
                M_exp = (index_paa_pred(i) * double_ages)(index_age_comp_times(i));
                R = M_obs - M_exp;
                S = sqrt((index_paa_pred(i) * square(double_ages))(
                    index_age_comp_times(i)) - square(M_exp));
                W1 = elem_div(elem_prod(R,sqrt(input_Neff_index(i)(
                    index_age_comp_times(i)))), S);
                dvariable W = sum(square(W1 - mean(W1)))/(double(
                    n_index_age_comp_years(i)) - 1.0);
                Neff_stage2_mult_index(i) = 1.0/W;
         }
  }

FUNCTION get_index_likely

  // indices (lognormal)
  index_likely = 0.0;
  index_stdresid = 0.0;
  int this_y = 0;
  for (int i=1; i<=n_indices; i++)
  {
    if(n_index_years(i)>0)
    {
      index_likely(i)=index_like_const(i);
      for(int y=1; y<=n_index_years(i); y++)
      {
        this_y = index_times(i,y);
        index_stdresid(i,this_y)=(log(index_obs(i,this_y) + 1.0e-15)-log(index_pred(i,
            this_y) + 1.0e-15))/index_sigma(i,this_y);
        index_likely(i)+=0.5*square(index_stdresid(i,this_y));
      }
    }
  }
  obj_fun+=lambda_index*index_likely;
  if (io==1) ICHECK(index_likely);

FUNCTION get_index_likely_with_calibration

  // indices (lognormal)
  index_likely = 0.0;
  index_stdresid = 0.0;
  int this_y = 0;
  for (int i=1; i<=n_indices; i++)
  {

    if(n_index_years(i)>0)
    {
```

```
      //have to adjust "index_like_const" because parts of it are not constant
      //this was not done in butterfish SARC 58 assessment.
      index_likely(i) = double(n_index_years(i))*(0.5*log(2.0*PI))+sum(log(index_sigma(i
         )(index_times(i))));
      for(int y=1; y<=n_index_years(i); y++)
      {
        this_y = index_times(i,y);
        if(calibrate_this_obs(i,this_y) > 0)
        {
          index_stdresid(i,this_y)=(log(calibrated_index_obs(i,this_y) + 1.0e-15)-log(
              index_pred(i,this_y) + 1.0e-15))/index_sigma(i,this_y);
        }
        else
        {
          index_stdresid(i,this_y)=(log(index_obs(i,this_y) + 1.0e-15)-log(index_pred(i,
              this_y) + 1.0e-15))/index_sigma(i,this_y);
        }
        index_likely(i)+=0.5*square(index_stdresid(i,this_y));
      }
    }
  }

  obj_fun+=lambda_index*index_likely;
  if (io==1) ICHECK(index_likely);

FUNCTION get_index_age_comp_likely

  // indices age comp (multinomial)
  index_age_comp_likely = 0.0;
  int this_y = 0;
  dvariable temp = 0.0;
  for (int i=1; i<=n_indices; i++)
  {
      if(n_index_age_comp_years(i)>0)
      {
        index_age_comp_likely(i)=sum(index_age_comp_like_const(i));
        for(int y=1; y<=n_index_age_comp_years(i); y++)
        {
          this_y = index_age_comp_times(i,y);
          temp=index_paa_obs(i,this_y)*log(index_paa_pred(i,this_y) + 1.0e-15);
          index_age_comp_likely(i) -= input_Neff_index(i,this_y)*temp;
        }
      }
  }
  obj_fun+=sum(index_age_comp_likely);
  if (io==1) ICHECK(index_age_comp_likely);

FUNCTION get_index_age_comp_likely_with_calibration

  // indices age comp (multinomial)
  index_age_comp_likely = 0.0;
  int this_y = 0;
```

```
  dvariable temp = 0.0;
  dvar_matrix calibrated_index_age_comp_like_const(1,n_indices,1,n_years);
  calibrated_index_age_comp_like_const = 0.0;
  for (int i=1; i<=n_indices; i++)
  {
    if(n_index_age_comp_years(i)>0)
    {
      for(int y=1; y<=n_index_age_comp_years(i); y++)
      {
        this_y = index_age_comp_times(i,y);
        if(calibrate_this_obs(i,this_y) > 0)
        {
          //have to adjust "index_age_comp_like_const" because parts of it are not
              constant
          calibrated_index_age_comp_like_const(i,this_y) -= gammln(input_Neff_index(i,
              this_y) + 1.0);
          calibrated_index_age_comp_like_const(i,this_y) += sum(gammln(input_Neff_index(
              i,this_y)*calibrated_paa_obs(i,this_y) + 1.0));
          index_age_comp_likely(i) += calibrated_index_age_comp_like_const(i,this_y);
          temp=calibrated_paa_obs(i,this_y)*log(index_paa_pred(i,this_y) + 1.0e-15);
        }
        else
        {
          index_age_comp_likely(i) += index_age_comp_like_const(i,this_y);
          temp=index_paa_obs(i,this_y)*log(index_paa_pred(i,this_y) + 1.0e-15);
        }
        index_age_comp_likely(i) -= input_Neff_index(i,this_y)*temp;
      }
    }
  }
  obj_fun+=sum(index_age_comp_likely);
  if (io==1) ICHECK(rowsum(index_age_comp_like_const));
  if (io==1) ICHECK(index_age_comp_likely);

FUNCTION get_catch_likely

  // total catch and discards (lognormal)
  catch_tot_likely = 0.0;
  catch_tot_stdresid = 0.0;
  discard_tot_likely = 0.0;
  discard_tot_stdresid = 0.0;
  int this_y = 0;
  for (int i=1; i<=n_fleets; i++)
  {
    if(n_catch_years(i) > 0)
    {
      catch_tot_likely(i) = catch_tot_like_const(i);
      for(int y = 1; y <= n_catch_years(i); y++)
      {
        this_y = catch_times(i,y);
        catch_tot_stdresid(i,this_y) = (log(catch_tot_fleet_obs(i,this_y)+1.0e-15)-log(
            catch_tot_fleet_pred(i,this_y)+1.0e-15))/catch_tot_sigma(i,this_y);
```

```
      catch_tot_likely(i) += 0.5*square(catch_tot_stdresid(i,this_y));
    }
  }
  if(n_discard_years(i) > 0)
  {
    discard_tot_likely(i) = discard_tot_like_const(i);
    for(int y = 1; y <= n_discard_years(i); y++)
    {
      this_y = discard_times(i,y);
      discard_tot_stdresid(i,this_y) = (log(discard_tot_fleet_obs(i,this_y)+1.0e-15)-
         log(discard_tot_fleet_pred(i,this_y)+1.0e-15))/discard_tot_sigma(i,this_y);
      discard_tot_likely(i)+=0.5*square(discard_tot_stdresid(i,this_y));
    }
  }
}
obj_fun+=lambda_catch_tot*catch_tot_likely;
obj_fun+=lambda_discard_tot*discard_tot_likely;
if (io==1) ICHECK(catch_tot_likely);
if (io==1) ICHECK(discard_tot_likely);

FUNCTION get_catch_age_comp_likely

// catch age comp (multinomial)
catch_age_comp_likely = 0.0;
discard_age_comp_likely = 0.0;
int this_y = 0;
dvariable temp;
for (int i=1; i<=n_fleets; i++)
{
  if(n_catch_age_comp_years(i) > 0)
  {
    catch_age_comp_likely(i) = catch_age_comp_like_const(i);
    for(int y = 1; y <= n_catch_age_comp_years(i); y++)
    {
      this_y = catch_age_comp_times(i,y);
      temp=catch_paa_obs(i,this_y)*log(catch_paa_pred(i,this_y) + 1.0e-15);
      catch_age_comp_likely(i) -= input_Neff_catch(i,this_y)*temp;
    }
  }
  if(n_discard_age_comp_years(i) > 0)
  {
    discard_age_comp_likely(i) = discard_age_comp_like_const(i);
    for(int y = 1; y <= n_discard_age_comp_years(i); y++)
    {
      this_y = discard_age_comp_times(i,y);
      //add in negligible amount to predicted in case release proportion is 0
      temp=discard_paa_obs(i,this_y)*log(discard_paa_pred(i,this_y)+1.0e-15);
      discard_age_comp_likely(i) -= input_Neff_discard(i,this_y)*temp;
    }
  }
}
obj_fun+=sum(catch_age_comp_likely)+sum(discard_age_comp_likely);
```

```
  if (io==1) ICHECK(catch_age_comp_likely);
  if (io==1) ICHECK(discard_age_comp_likely);

FUNCTION get_SR_penalty

  // stock-recruitment relationship (lognormal)
  SR_penalty = 0.0;
  SR_stdresid=0.0;
  //use the stock- recruit relationship
  if(SR_model_type == 1) SR_stdresid=elem_div(log(recruits)-log(SR_pred_recruits(1,
     n_years)),recruit_sigma);
  //No stock-recruit relationship, just average R and deviations
  else SR_stdresid=elem_div(log_recruit_devs,recruit_sigma);

  if (active(log_recruit_devs) && lambda_recruit_devs > 1.0e-15)
  {
    SR_penalty = SR_penalty_const + 0.5*norm2(SR_stdresid);
    obj_fun+=lambda_recruit_devs*SR_penalty;
  }
  if (io==1) ICHECK(SR_penalty);

FUNCTION get_selectivity_penalties

  // selectivity parameters have scaled,shifted beta distributed penalties
  selpars_penalty=0.0;
  selpars_stdresid=0.0;
  for (int i=1; i<=n_selpars; i++)
  {
    if (active(selpars(i)) && lambda_selpars(i) > 1.0e-15)
    {
      selpars_penalty(i)+=selpars_penalty_const(i);
      if(selpars_penalty_type(i) == 1)
      {
        dvariable p = (selpars(i)-selpars_lower(i))/(selpars_upper(i)-selpars_lower(i));
        selpars_penalty(i)-= (selpars_penalty_a(i)-1.0)*log(selpars(i) - selpars_lower(i
           ) + 1.0e-15) + (selpars_penalty_b(i)-1.0)*log(selpars_upper(i) - selpars(i) +
            1.0e-15);
        selpars_stdresid(i) = (p - selpars_penalty_mu(i))/sqrt(selpars_penalty_mu(i)*(1-
           selpars_penalty_mu(i))/(selpars_penalty_phi(i)+1));
      }
      else
      {
        selpars_stdresid(i) = (log(selpars(i)) - log(selpars_ini(i)))/
           selpars_penalty_sigma(i);
        selpars_penalty(i) += 0.5*square(selpars_stdresid(i));
      }
      obj_fun+=lambda_selpars(i)*selpars_penalty(i);
    }
  }
  if (io==1) ICHECK(selpars_penalty);

FUNCTION get_steepness_penalty
```

```
  steepness_penalty=0.0;
  steepness_stdresid=0.0;
  if (current_phase() >= phase_steepness && lambda_steepness > 1.0e-15)
  {
    steepness_penalty = steepness_penalty_const;
    dvariable mean_h = mean(steepness);
    if(steepness_penalty_type == 1)
    {
      //scaled beta penalty because steepness is greater than 0.2 and less than 1.0
      dvariable p = (mean_h - 0.2)/(1.0 - mean_h);
      steepness_penalty -= (steepness_penalty_a-1.0) * log(mean_h-0.2 + 1.0e-15) + (
          steepness_penalty_b-1.0) * log(1.0 - mean_h + 1.0e-15);
      steepness_stdresid = (p - steepness_penalty_mu)/sqrt(steepness_penalty_mu*(1-
          steepness_penalty_mu)/(steepness_penalty_phi+1));
    }
          else
          {
      steepness_stdresid = (log(mean_h) - log(steepness_ini))/steepness_penalty_sigma;
      steepness_penalty += 0.5*square(steepness_stdresid);
          }
    obj_fun+=lambda_steepness*steepness_penalty;
  }
  if (io==1) ICHECK(steepness_penalty);

FUNCTION get_SR_scalar_penalty
  SR_scalar_penalty=0.0;
  SR_scalar_stdresid=0.0;
  if (current_phase() >= phase_SR_scalar && lambda_SR_scalar > 1.0e-15)
  {
    dvariable mean_SR_scalar = mean(log_SR_scalar);
    SR_scalar_stdresid=(mean_SR_scalar-log(SR_scalar_ini))/SR_scalar_sigma;
    //SR_scalar_penalty=SR_scalar_penalty_const + mean_SR_scalar + 0.5*square(
        SR_scalar_stdresid);
    SR_scalar_penalty=SR_scalar_penalty_const + 0.5*square(SR_scalar_stdresid);
    obj_fun+=lambda_SR_scalar*SR_scalar_penalty;
  }
  if (io==1) ICHECK(SR_scalar_penalty);

FUNCTION get_Fmult_year1_penalty
  Fmult_year1_stdresid = 0.0;
  Fmult_year1_penalty = 0.0;
  for (int i=1; i<=n_fleets; i++)
  {
          if (current_phase() >= phase_Fmult_year1(i) && lambda_Fmult_year1(i) > 1.0e
              -15)
          {
      Fmult_year1_stdresid(i)=(log_Fmult_year1(i)-log(Fmult_year1_ini(i)))/
          Fmult_year1_sigma(i);
      Fmult_year1_penalty(i)=Fmult_year1_penalty_const(i) + 0.5*square(
          Fmult_year1_stdresid(i));;
      obj_fun+=lambda_Fmult_year1(i)*Fmult_year1_penalty(i);
    }
```

```
    }
    if (io==1) ICHECK(Fmult_year1_penalty);

FUNCTION get_Fmult_devs_penalty
    Fmult_devs_stdresid=0.0;
    Fmult_devs_penalty = 0.0;
    for (int i=1; i<=n_fleets; i++)
    {
      if (current_phase() >=  phase_Fmult_devs(i) && lambda_Fmult_devs(i) > 1.0e-15)
      {
        Fmult_devs_stdresid(i)=log_Fmult_devs(i)(catch_min_time(i)+1,catch_max_time(i))/
            Fmult_devs_sigma(i);
        Fmult_devs_penalty(i)+=Fmult_devs_penalty_const(i) + 0.5*norm2(Fmult_devs_stdresid
            (i));
        obj_fun+=lambda_Fmult_devs(i)*Fmult_devs_penalty(i);
      }
    }
    if (io==1) ICHECK(Fmult_devs_penalty);

FUNCTION get_N_year1_penalty
    N_year1_stdresid=0.0;
    N_year1_penalty = 0.0;
    if (NAA_year1_flag==1)
    {
      //note that S is estimated when M is estimated so the predicted NAA(1) varies
      NAA_year1_pred(1)=SR_pred_recruits(1);
      for (int a=2; a<=n_ages; a++) NAA_year1_pred(a)=NAA_year1_pred(a-1)*S(1,a-1);
      NAA_year1_pred(n_ages)/=(1.0-S(1,n_ages));
    }
    else if (NAA_year1_flag==2) NAA_year1_pred=NAA_year1_ini;

    if (active(log_N_year1_devs) && lambda_N_year1_devs > 1.0e-15)
    {
      N_year1_stdresid=(log(NAA(1))-log(NAA_year1_pred))/N_year1_sigma;
      N_year1_penalty = N_year1_penalty_const + 0.5*norm2(N_year1_stdresid);
      obj_fun+=lambda_N_year1_devs*N_year1_penalty;
    }
    if (io==1) ICHECK(N_year1_penalty);

FUNCTION get_availability_penalty
    availability_penalty=0.0;
    availability_stdresid=0.0;
    for(int i = 1; i <= n_indices; i++)
    {
            if (current_phase() >= first_availability_phase(i) && lambda_availability(i) >
                1.0e-15)
            {
        availability_penalty(i) = availability_penalty_const(i);
        dvariable mean_avail = mean(availability(i)(index_times(i)));
        if(availability_penalty_type(i) == 1)
        {
          //scaled beta penalty because availability is between availability_lower and
```

```
            availability_upper
        dvariable p = (mean_avail - availability_lower(i))/(availability_upper(i) -
            mean_avail);
        availability_penalty(i) -= (availability_penalty_a(i)-1.0) * log(mean_avail-
            availability_lower(i) + 1.0e-15) +
          (availability_penalty_b(i) - 1.0) * log(availability_upper(i) - mean_avail +
              1.0e-15);
        availability_stdresid(i) = (p - availability_penalty_mu(i))/sqrt(
            availability_penalty_mu(i)*(1-availability_penalty_mu(i))/(
            availability_penalty_phi(i)+1));
      }
      else
      {
        availability_stdresid(i) = (log(mean_avail) - log(availability_ini(i) + 1.0e-15)
            )/availability_penalty_sigma(i);
        availability_penalty(i) += 0.5*square(availability_stdresid(i));
      }
      obj_fun+=lambda_availability(i)*availability_penalty(i);
          }
  }
  if (io==1) ICHECK(availability_penalty);

FUNCTION get_availability_AR1_penalty

  availability_AR1_stdresid=0.0;
  availability_AR1_penalty = 0.0;
  for (int i=1; i<=n_indices; i++)
  {
    if (active(availability_AR1_devs(i)))
    {
      availability_AR1_stdresid(i)=availability_AR1_devs(i)/availability_AR1_sd(i);
      availability_AR1_penalty(i)=availability_AR1_penalty_const(i) + 0.5*norm2(
          availability_AR1_stdresid);
      obj_fun+=lambda_availability_AR1(i)*availability_AR1_penalty(i);
    }
  }
  if (io==1) ICHECK(availability_AR1_penalty);

FUNCTION get_efficiency_penalty
  efficiency_penalty=0.0;
  efficiency_stdresid=0.0;
  for(int i = 1; i <= n_indices; i++)
  {
          if (current_phase() >= first_efficiency_phase(i) && lambda_efficiency(i) > 1.0
              e-15)
          {
      efficiency_penalty(i) = efficiency_penalty_const(i);
      dvariable mean_efficiency = mean(efficiency(i)(index_times(i)));
      if(efficiency_penalty_type(i) == 1)
      {
        //scaled beta penalty because efficiency is between efficiency_lower and
            efficiency_upper
```

```
        dvariable p = (mean_efficiency - efficiency_lower(i))/(efficiency_upper(i) -
            mean_efficiency);
        efficiency_penalty(i) -= (efficiency_penalty_a(i)-1.0) * log(mean_efficiency-
            efficiency_lower(i) + 1.0e-15) +
          (efficiency_penalty_b(i) - 1.0) * log(efficiency_upper(i) - mean_efficiency +
              1.0e-15);
        efficiency_stdresid(i) = (p - efficiency_penalty_mu(i))/sqrt(
            efficiency_penalty_mu(i)*(1-efficiency_penalty_mu(i))/(efficiency_penalty_phi
            (i)+1));
      }
      else
      {
        efficiency_stdresid(i) = (log(mean_efficiency) - log(efficiency_ini(i) + 1.0e
            -15))/efficiency_penalty_sigma(i);
        efficiency_penalty(i) += 0.5*square(efficiency_stdresid(i));
      }
      obj_fun+=lambda_efficiency(i)*efficiency_penalty(i);
          }
  }
   if (io==1) ICHECK(efficiency_penalty);

FUNCTION get_rel_efficiency_penalty

        rel_efficiency_penalty = 0.0;
        for(int i = 1; i <= n_rel_efficiency_penalties; i++)
        {
    if(active(rel_efficiency_coef_devs(i)))
    {
      rel_efficiency_penalty(i) = rel_efficiency_penalty_const(i) + 0.5 *
          rel_efficiency_coef_devs(i) * inv(var_rel_efficiency_coef(i)) *
          rel_efficiency_coef_devs(i);
      obj_fun += lambda_rel_efficiency(i) * rel_efficiency_penalty(i);
    }
  }

  if (io==1) ICHECK(rel_efficiency_penalty);

FUNCTION get_Fmult_max_penalty
  dvariable temp_Fmult_max;
  Fmult_max_penalty=0.0;
  lambda_Fmult_max_penalty = 1000.0;
  for (int i=1; i<=n_fleets; i++)
  {
    for (int y=1; y<=n_years; y++)
    {
      temp_Fmult_max=mfexp(log_Fmult(i,y))*max(sel_by_fleet(i,y));
      if(temp_Fmult_max>Fmult_max_value) Fmult_max_penalty+=(temp_Fmult_max-
          Fmult_max_value)*(temp_Fmult_max-Fmult_max_value);
    }
  }
  obj_fun+=lambda_Fmult_max_penalty*Fmult_max_penalty;
  if (io==1) ICHECK(Fmult_max_penalty);
```

```
FUNCTION get_F_penalty

  // decrease emphasis on F near M as phases increase
  lambda_fpenalty=100.0*pow(10.0,(-1.0*current_phase()));
  // no penalty in final solution
  if (last_phase()) lambda_fpenalty=0.0;

  fpenalty=lambda_fpenalty*square(log(mean(FAA_tot))-log(mean(M_use)));
  obj_fun+=fpenalty;
  if (io==1) ICHECK(fpenalty);

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

  if(calibrate_indices == 1)
  {
    get_index_likely_with_calibration();
    get_index_age_comp_likely_with_calibration();
    get_rel_efficiency_penalty();
  }
  else
  {
    get_index_likely();
    get_index_age_comp_likely();
  }
  get_catch_likely();
  get_catch_age_comp_likely();

  get_SR_penalty();
  get_selectivity_penalties();
  get_steepness_penalty();
  get_SR_scalar_penalty();
  get_Fmult_year1_penalty();
  get_Fmult_devs_penalty();
  get_N_year1_penalty();
  get_availability_penalty();
  get_efficiency_penalty();
  get_availability_AR1_penalty();
  if(use_Fmult_max_penalty == 1) get_Fmult_max_penalty();
  if(use_F_penalty == 1) get_F_penalty();
  if(io == 1) ICHECK(obj_fun);
  if(io == 1 && current_phase() == 2) ad_exit(1);

FUNCTION write_MCMC
  // first the output file for AgePro
  dvariable tempR = 0.0;
  dvar_vector tempFmult(1,n_years);
  if (MCMCnyear_opt == 0) // use final year
  {
    if (fillR_opt == 0) NAAbsn(1)=NAA(n_years,1);
```

```
    else if (fillR_opt == 1)
    {
    NAAbsn(1)=SR_pred_recruits(n_years);
    }
    else if (fillR_opt == 2)
    {
      tempR=0.0;
      for (int i=Ravg_start; i<=Ravg_end; i++) tempR+=log(NAA(i-year1+1,1));
      NAAbsn(1)=mfexp(tempR/(Ravg_end-Ravg_start+1.0));
    }
    for (int a=2; a<=n_ages; a++) NAAbsn(a)=NAA(n_years,a);
  }
  else // use final year + 1
  {
    if (fillR_opt == 1) NAAbsn(1)=SR_pred_recruits(n_years+1);
    else if (fillR_opt == 2)
    {
      tempR=0.0;
      for (int i=Ravg_start; i<=Ravg_end; i++) tempR+=log(NAA(i-year1+1,1));
      NAAbsn(1)=mfexp(tempR/(Ravg_end-Ravg_start+1.0));
    }
    for (int a=2; a<=n_ages; a++) NAAbsn(a)=NAA(n_years,a-1)*S(n_years,a-1);
    NAAbsn(n_ages)+=NAA(n_years,n_ages)*S(n_years,n_ages);
  }

  for (int y=1; y<=n_years; y++) tempFmult(y) = max(FAA_tot(y));


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

  // now the standard MCMC output file
  basicMCMC <<
        Freport << " ";
  if(estimate_M ==1)
  {
         for(int y = 1; y<= n_years; y++) basicMCMC << M_use(y) << " ";
  }
  basicMCMC <<
    SSB << " " <<
    tempFmult << " " <<
    rowsum(elem_prod(WAAjan1b, NAA)) << " ";
    for(int i = 1; i <= nXSPR; i++) basicMCMC << FXSPR_report(i) << " ";
    basicMCMC <<
    MSY << " " <<
    SSBmsy << " " <<
    Fmsy << " " <<
    SSBmsy_ratio << " " <<
    Fmsy_ratio << " " <<
    endl;

REPORT_SECTION
```

```
int this_y = 0;
report << "Age Structured Assessment Program (ASAP) Version 4.0" << endl;
report << "Start time for run: " << ctime(&start) << endl;
report << "Objective_function: " << obj_fun << endl << endl;
report << "Component                Lambda          obj_fun" << endl;
for (int i=1; i<=n_fleets; i++) if(n_catch_years(i)>0)
{
   report << "Catch_Fleet_" << i << "                  " << lambda_catch_tot(i) << "
              " << lambda_catch_tot(i)*catch_tot_likely(i) << 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 (int i=1; i<=n_fleets; i++) if(n_discard_years(i)>0)
  {
    report << "Discard_Fleet_" << i << "              " << lambda_discard_tot(i) << "
                " << lambda_discard_tot(i)*discard_tot_likely(i) << endl;
  }
  report << "Discard_Fleet_Total            " << sum(lambda_discard_tot) << "
      " << lambda_discard_tot*discard_tot_likely << endl;
}
for (int i=1; i<=n_indices; i++) if(n_index_years(i)>0)
{
  report << "Index_Fit_" << i << "                    " << lambda_index(i) << "
              " << lambda_index(i)*index_likely(i) << endl;
}
report << "Index_Fit_Total                " << sum(lambda_index) << "           " <<
    lambda_index*index_likely << endl;
for (int i=1; i<=n_fleets; i++) if(n_catch_age_comp_years(i)>0)
{
  report << "Catch_Fleet_Age_Comps_" << i << "        see_below   " <<
      catch_age_comp_likely(i) << endl;
}
if(sum(n_catch_age_comp_years)>0)
{
  report << "Catch_Age_Comps_Total          see_below   " << sum(catch_age_comp_likely)
      << endl;
}
for (int i=1; i<=n_fleets; i++) if(n_discard_age_comp_years(i)>0)
{
  report << "Discard_Fleet_Age_Comps_" << i << "        see_below            " <<
      discard_age_comp_likely(i) << endl;
}
if(sum(n_discard_age_comp_years)>0)
{
  report << "Discard_Age_Comps_Total        see_below   " << sum(discard_age_comp_likely
      ) << endl;
}
for (int i=1; i<=n_indices; i++) if(n_index_age_comp_years(i)>0)
{
  report << "Index_Age_Comps_" << i << "              see_below   " <<
```

```
      index_age_comp_likely(i) << endl;
}
if(sum(n_index_age_comp_years)>0)
{
  report << "Index_Age_Comps_Total          see_below   " << sum(index_age_comp_likely)
      << endl;
}
dvariable sum_sel_lambda=0;
dvariable sum_sel_lambda_penalty=0.0;
for (int i=1; i<=n_selpars; i++)
{
  if(fabs(selpars_penalty(i))>1.0e-15)
  {
    report << "Selectivity_Parameter_" << i;
    if      (i < 10  ) report << "       ";
    else if (i < 100 ) report << "      ";
    else if (i < 1000) report << "     ";
    report << lambda_selpars(i) << "           " << lambda_selpars(i)*selpars_penalty(i
        ) << endl;
    sum_sel_lambda+=lambda_selpars(i);
    sum_sel_lambda_penalty+=lambda_selpars(i)*selpars_penalty(i);
  }
}
if(sum(fabs(selpars_penalty))>1.0e-15)
{
  report << "Selectivity_Parameter_Total    " << sum_sel_lambda << "           " <<
      sum_sel_lambda_penalty << endl;
}
for (int i=1; i<=n_indices; i++) if(fabs(availability_penalty(i))>1.0e-15)
{
  report << "Availability_" << i << "              "  << lambda_availability(i) << "
             " << lambda_availability(i)*availability_penalty(i) << endl;
}
if(sum(fabs(availability_penalty))>1.0e-15)
{
  report << "Availability_Total                  " << sum(lambda_availability) << "
             " << lambda_availability*availability_penalty << endl;
}
for (int i=1; i<=n_indices; i++) if(fabs(availability_AR1_penalty(i))>1.0e-15)
{
        report << "Availability_AR1_" << i << "              "  <<
           lambda_availability_AR1(i) << "           " << lambda_availability_AR1(i)*
           availability_AR1_penalty(i) << endl;
}
if(sum(availability_AR1_penalty)>1.0e-15)
{
  report << "Availability_AR1_Total                  " << sum(lambda_availability_AR1)
      << "           " << lambda_availability_AR1*availability_AR1_penalty << endl;
}
for (int i=1; i<=n_indices; i++) if(fabs(efficiency_penalty(i))>1.0e-15)
{
  report << "Efficiency_" << i << "          "  << lambda_efficiency(i) << "            "
```

```
      << lambda_efficiency(i)*efficiency_penalty(i) << endl;
}
if(sum(fabs(efficiency_penalty))>1.0e-15)
{
  report << "Efficiency_Total                    " << sum(lambda_efficiency) << "
            " << lambda_efficiency*efficiency_penalty << endl;
}
for (int i=1; i<=n_fleets; i++) if(fabs(Fmult_year1_penalty(i))>1.0e-15)
{
  report << "Fmult_year1_fleet_" << i << "          " << lambda_Fmult_year1(i) << "
            " << lambda_Fmult_year1(i)*Fmult_year1_penalty(i) << endl;
}
if(sum(fabs(Fmult_year1_penalty))>1.0e-15)
{
  report << "Fmult_year1_Total        " << sum(lambda_Fmult_year1) << "           " <<
      lambda_Fmult_year1*Fmult_year1_penalty << endl;
}
for (int i=1; i<=n_fleets; i++) if(fabs(Fmult_devs_penalty(i))>1.0e-15)
{
  report << "Fmult_devs_fleet_" << i << "           " << lambda_Fmult_devs(i) << "
            " << lambda_Fmult_devs(i)*Fmult_devs_penalty(i) << endl;
}
if(sum(fabs(Fmult_devs_penalty))>1.0e-15)
{
  report << "Fmult_devs_Total         " << sum(lambda_Fmult_devs) << "           " <<
      lambda_Fmult_devs*Fmult_devs_penalty << endl;
}
for (int i=1; i<=n_rel_efficiency_penalties; i++) if(fabs(rel_efficiency_penalty(i))
    >1.0e-15)
{
  report << "Rel_efficiency_penalty_" << i << "       " << lambda_rel_efficiency(i) <<
      "          " << lambda_rel_efficiency(i)*rel_efficiency_penalty(i) << endl;
}
if(sum(fabs(rel_efficiency_penalty))>1.0e-15)
{
  report << "Rel_efficency_Total            " << sum(lambda_rel_efficiency) << "
            " << lambda_rel_efficiency*rel_efficiency_penalty << endl;
}
if(fabs(N_year1_penalty) > 1.0e-15)
{
  report << "N_year_1                       " << lambda_N_year1_devs << "           " <<
       lambda_N_year1_devs*N_year1_penalty << endl;
}
if(fabs(SR_penalty) > 1.0e-15)
{
  report << "Recruit_devs                   " << lambda_recruit_devs << "           " <<
       lambda_recruit_devs*SR_penalty << endl;
}
if(fabs(steepness_penalty) > 1.0e-15)
{
  report << "steepness                      " << lambda_steepness << "           " <<
      lambda_steepness*steepness_penalty << endl;
```

```
}
if(fabs(SR_scalar_penalty) > 1.0e-15)
{
  report << "SR_scalar                    " << lambda_SR_scalar << "          " <<
     lambda_SR_scalar*SR_scalar_penalty << endl;
}
if(fabs(Fmult_max_penalty) > 1.0e-15)
{
  report << "Fmult_max_penalty            " << lambda_Fmult_max_penalty << "
            " << Fmult_max_penalty << endl;
}
if(fabs(fpenalty) > 0)
{
  report << "F_penalty                    " << lambda_fpenalty << "          " <<
     fpenalty << endl;
}
report << endl;

if(sum(n_catch_years) > 0)
{
  report << "Observed and predicted total fleet catch by year and standardized
     residual" << endl;
  for (int i=1; i<=n_fleets; i++)
  {
    if(n_catch_years(i) > 0)
    {
      report << "fleet " << i << endl;
      for (int y=1; y<=n_catch_years(i); y++)
      {
        this_y = catch_times(i,y);
        report << catch_years(i,y) << "  " << catch_tot_fleet_obs(i,y) << "  " <<
           catch_tot_fleet_pred(i,y) << "  " << catch_tot_stdresid(i,y) << endl;
      }
    }
  }
  report << endl;
}
if(sum(n_discard_years)>0)
{
  report << "Observed and predicted total fleet discards by year and standardized
     residual" << endl;
  for (int i=1; i<=n_fleets; i++)
  {
    if(n_discard_years(i) > 0)
    {
      report << "fleet " << i << " total discards" << endl;
      for (int y=1; y<=n_discard_years(i); y++)
      {
        this_y = discard_times(i,y);
        report << discard_years(i,y) << "  " << discard_tot_fleet_obs(i,this_y) << "
           " << discard_tot_fleet_pred(i,this_y) << "  " << discard_tot_stdresid(i,
           this_y) << endl;
```

```
      }
    }
  }
  report << endl;
}
if(sum(n_catch_age_comp_years)>0)
{
  report << "Proportions of catch at age by fleet" << endl;
  for (int i=1; i<=n_fleets; i++)
  {
    if(n_catch_age_comp_years(i) > 0)
    {
      report << "fleet " << i << endl;
      for (int y=1; y<=n_catch_age_comp_years(i); y++)
      {
        this_y = catch_age_comp_times(i,y);
        report << "Year " << catch_age_comp_years(i,y) << " Obs  = " << catch_paa_obs(
            i,this_y) << endl;
        report << "Year " << catch_age_comp_years(i,y) << " Pred = " << catch_paa_pred
            (i,this_y) << endl;
      }
    }
  }
  report << endl;
}
if(sum(n_discard_age_comp_years) > 0)
{
  report << "Proportions of discards at age by fleet" << endl;
  for (int i=1; i<=n_fleets; i++)
  {
    if(n_discard_age_comp_years(i) > 0)
    {
      report << "fleet " << i << endl;
      for (int y=1; y<=n_discard_age_comp_years(i); y++)
      {
        this_y = discard_age_comp_times(i,y);
        report << "Year " << discard_age_comp_years(i,y) << " Obs  = " <<
            discard_paa_obs(i,this_y) << endl;
        report << "Year " << discard_age_comp_years(i,y) << " Pred = " <<
            discard_paa_pred(i,this_y) << endl;
      }
    }
  }
  report << endl;
}

output_Neff_catch = 0;
output_Neff_discard = 0;
if(sum(n_catch_age_comp_years)>0)
{
  report << "Input and Estimated effective sample sizes for catch at age by fleet" <<
      endl;
```

```
  for (int i=1; i<=n_fleets; i++)
  {
    if(n_catch_age_comp_years(i)> 0)
    {
      report << "fleet " << i << endl;
      for(int y =1; y <= n_catch_age_comp_years(i)>0; y++)
      {
        this_y = catch_age_comp_times(i,y);
        output_Neff_catch(i,this_y)=catch_paa_pred(i,this_y)*(1.0-catch_paa_pred(i,
            this_y))/norm2(catch_paa_obs(i,this_y)-catch_paa_pred(i,this_y));
        report << catch_age_comp_years(i,y) << "   " << input_Neff_catch(i,this_y) << "
              " << output_Neff_catch(i,this_y) << endl;
      }
      report << "Total   " << sum(input_Neff_catch(i)) << "   " << sum(output_Neff_catch
          (i)) << endl;
    }
  }
  report << endl;
}
if(sum(n_discard_age_comp_years)>0)
{
  report << "Input and Estimated effective sample sizes for discards at age by fleet"
      << endl;
  for (int i=1; i<=n_fleets; i++)
  {
    if(n_discard_age_comp_years(i)>0)
    {
      report << "fleet " << i << endl;
      for(int y =1; y <= n_discard_age_comp_years(i)>0; y++)
      {
        this_y = discard_age_comp_times(i,y);
        output_Neff_discard(i,this_y)=discard_paa_pred(i,this_y)*(1.0-discard_paa_pred
            (i,this_y))/norm2(discard_paa_obs(i,this_y)-discard_paa_pred(i,this_y));
        report << discard_age_comp_years(i,y) << "   " << input_Neff_discard(i,this_y)
            << "   " << output_Neff_discard(i,this_y) << endl;
      }
      report << "Total   " << sum(input_Neff_discard(i)) << "   " << sum(
          output_Neff_discard(i)) << endl;
    }
  }
  report << endl;
}

if(sum(n_index_years) > 0)
{
  report << endl << "Index data" << endl;
  for (int i=1; i<=n_indices; i++)
  {
    if(n_index_years(i) > 0)
    {
      report << "index number " << i << endl;
      report << "aggregate units = " << index_units_aggregate(i) << endl;
```

```
      report << "proportions units = " << index_units_proportions(i) << endl;
      report << "year, month, obs index, pred index, standardized residual" << endl;
      for (int y=1; y<=n_index_years(i); y++)
      {
        this_y = index_times(i,y);
        report << index_years(i,y) << "  " << index_month(i,this_y) << "  ";
        if(calibrate_indices == 1 && calibrate_this_obs(i,this_y) > 0)
        {
          report << calibrated_index_obs(i,this_y);
        }
        else report << index_obs(i,this_y);
        report << "  " << index_pred(i,this_y) << "  " << index_stdresid(i,this_y) <<
            endl;
      }
    }
  }
  report << endl;
}
if(sum(n_index_age_comp_years)>0)
{
  report << "Index proportions at age by index" << endl;
  for (int i=1; i<=n_indices; i++)
  {
    if(n_index_age_comp_years(i)>0)
    {
      report << "index " << i << endl;
      for (int y=1; y<=n_index_age_comp_years(i); y++)
      {
        this_y = index_age_comp_times(i,y);
        report << "Year " << index_age_comp_years(i,y) << " Obs  = ";
        if(calibrate_indices ==1 && calibrate_this_obs(i,y) > 0)
        {
        report << calibrated_paa_obs(i,this_y) << endl;
        }
        else report << index_paa_obs(i,this_y) << endl;
        report << "Year " << index_age_comp_years(i,y) << " Pred = " << index_paa_pred
            (i,this_y) << endl;
      }
    }
  }
  report << endl;
}
output_Neff_index=0.0;
if(sum(n_index_age_comp_years) > 0)
{
  report << " Input and Estimated effective sample sizes for age composition by index"
       << endl;
  for (int i=1; i<=n_indices; i++)
  {
    if(n_index_age_comp_years(i) > 0)
    {
      report << "index " << i << endl;
```

```
      for (int y=1; y<=n_index_age_comp_years(i); y++)
      {
        this_y = index_age_comp_times(i,y);
        if(calibrate_indices == 1 && calibrate_this_obs(i,y) > 0)
        {
          output_Neff_index(i,this_y)=index_paa_pred(i,this_y)*(1.0-index_paa_pred(i,
              this_y))/norm2(calibrated_paa_obs(i,this_y)-index_paa_pred(i,this_y));
        }
        else output_Neff_index(i,this_y)=index_paa_pred(i,this_y)*(1.0-index_paa_pred(
            i,this_y))/norm2(index_paa_obs(i,this_y)-index_paa_pred(i,this_y));
        report << index_age_comp_years(i,y) << "  " << input_Neff_index(i,this_y) << "
              " << output_Neff_index(i,this_y) << endl;
      }
      report << "Total  " << sum(input_Neff_index(i)(index_times(i))) << "  " << sum(
          output_Neff_index(i)(index_times(i))) << endl;
    }
  }
  report << endl;
}
report << "Penalties section: only applicable for included penalties" << endl;
report << endl;
      if(sum(fabs(selpars_penalty))>1.0e-15)
      {
  report << "Standardized Residuals for selectivity parameters" << endl;
  for (int i=1; i<=n_selpars; i++)
  {
    if(fabs(selpars_penalty(i))>1.0e-15) report << i << "  " << selpars_stdresid(i) <<
        endl;
  }
}

if (fabs(N_year1_penalty) > 1.0e-15)
{
   report << "Nyear1 estimate, expected, standardized residual" << endl;
   for (int a=2; a<=n_ages; a++) report << a << "  " << NAA(1,a) << "  " <<
       NAA_year1_pred(a) << "  " << N_year1_stdresid(a) << endl;
   report << endl;
}
if (sum(fabs(Fmult_year1_penalty)) > 1.0e-15)
{
  report << "Fleet Obs, Initial, and Stadardized Residual for Fmult" << endl;
  for (int i=1; i<=n_fleets; i++) if(fabs(Fmult_year1_penalty(i)) > 1.0e-15)
  {
    report << i << "  " << mfexp(log_Fmult_year1(i)) << "  " << Fmult_year1_ini(i) <<
        "  " << Fmult_year1_stdresid(i) << endl;
  }
  report << endl;
}
if (sum(fabs(Fmult_devs_penalty)) > 1.0e-15)
{
  report << "Standardized Residuals for Fmult_devs by fleet and year" << endl;
  for (int i=1; i<=n_fleets; i++)
```

```
  {
    if(fabs(Fmult_devs_penalty(i)) > 1.0e-15)
    {
      report << "fleet " << i << " Fmult_devs standardized residuals" << endl;
      for (int y=catch_min_time(i)+1; y<=catch_max_time(i); y++) report << y << "  "
          << Fmult_devs_stdresid(i,y) << endl;
    }
  }
  report << endl;
}
if (sum(fabs(availability_penalty)) > 1.0e-15)
{
  report << "Index average, Initial, and Standardized Residual for availability" <<
      endl;
  for (int i=1; i<=n_indices; i++) if(fabs(availability_penalty(i)) > 1.0e-15)
  {
    report << i << "  " << mean(availability(i)(index_times(i))) << "  " <<
        availability_ini(i) << "  " <<  availability_stdresid(i) << endl;
  }
  report << endl;
}
if (sum(fabs(availability_AR1_penalty)) > 1.0e-15)
{
  report << "Standardized Residuals for availability_AR1 by index and year" << endl;
  for (int i=1; i<=n_indices; i++)
  {
    if(fabs(availability_AR1_penalty(i)) > 1.0e-15)
    {
      report << "index " << i << " availability_AR1_devs standardized residuals" <<
          endl;
      for (int y=index_min_time(i)+1; y<=index_max_time(i); y++) report << y << "  "
          << availability_AR1_stdresid(i,y) << endl;
    }
  }
  report << endl;
}
if (sum(fabs(efficiency_penalty)) > 1.0e-15)
{
  report << "Index average, Initial, and Standardized Residual for efficiency" << endl
      ;
  for (int i=1; i<=n_indices; i++) if(fabs(efficiency_penalty(i)) > 1.0e-15)
  {
    report << i << "  " << mean(efficiency(i)(index_times(i))) << "  " <<
        efficiency_ini(i) << "  " <<  efficiency_stdresid(i) << endl;
  }
  report << endl;
}
if (fabs(steepness_penalty) > 1.0e-15)
{
   report << "Obs, Initial, and Stadardized Residual for SR steepness" << endl;
   report << steepness << "  " << steepness_ini << "  " << steepness_stdresid << endl;
   report << endl;
```

```
}
if (fabs(SR_scalar_penalty) > 1.0e-15)
{
   report << "Obs, Initial , and Stadardized Residual for SR scalar" << endl;
   report << mfexp(mean(log_SR_scalar)) << "  " << SR_scalar_ini << "  " <<
      SR_scalar_stdresid << endl;
   report << endl;
}
report << "End of Penalties Section" << endl << endl;

report << "Natural mortality by year and age" << endl;
report << "Year";
for(int a=1; a<=n_ages; a++) report << "  age_" << a;
report << endl;
for(int y =1; y<=n_years; y++) report << y + year1 - 1 << "  " << M_use(y) << endl;
report << endl;

report << "Fmult and selectivity by fleet and year" << endl;
report << "fleet  year  Fmult";
for(int a=1; a<=n_ages; a++) report << "  age_" << a;
report << endl;
for (int i=1; i<=n_fleets; i++)
{
  for (int y=1; y<=n_years; y++)
  {
    Fmult(i,y) = mfexp(log_Fmult(i,y))*max(sel_by_fleet(i,y));
    report << i << "  " << y + year1 - 1  << "  " << Fmult(i,y) << "  " <<
       sel_by_fleet(i,y) << endl;
  }
}
report << endl;
report << "Directed F by age and year for each fleet" << endl;
for (int i=1; i<=n_fleets; i++)
{
  report << "fleet " << i << " directed F at age" << endl;
  for (int y=1; y<=n_years; y++) report << directed_FAA_by_fleet(i,y) << endl;
}
report << "Discard F by age and year for each fleet" << endl;
for (int i=1; i<=n_fleets; i++)
{
  report << "fleet " << i << " Discard F at age" << endl;
  for (int y=1; y<=n_years; y++) report << FAA_by_fleet_discard(i,y) << endl;
}
report << "Total F" << endl;
for (int y=1; y<=n_years; y++) report << FAA_tot(y) << 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;
  report << "year    unweighted" << endl;
}
```

```
if (Freport_wtopt==2)
{
  report << "Freport N weighted in .std and MCMC files" << endl;
  report << "year   Nweighted" << endl;
}
if (Freport_wtopt==3)
{
  report << "Freport B weighted in .std and MCMC files" << endl;
  report << "year    Bweighted" << endl;
}
for (int y=1; y<=n_years; y++)
{
   report << y+year1-1 << "  " << Freport(y) << endl;
}
report << endl;

report << "availability, efficiency, q, and selectivity by index and year" << endl;
report << "index  year  availability  efficiency  catchability";
for(int a=1; a<=n_ages; a++) report << "  age_" << a;
report << endl;
for (int i=1; i<=n_indices; i++) if(n_index_years(i) > 0)
{
  for (int y=1; y<=n_index_years(i); y++)
  {
    report << i << "  " << index_years(i,y) << "  " << availability(i,index_times(i,y)
       ) << "  "  << efficiency(i,index_times(i,y)) <<
      "  " << q_by_index(i,index_times(i,y)) << "  " << sel_by_index(i,index_times(i,y
         )) << endl;
  }
}
report << endl;

report << "Population Numbers at the Start of the Year" << endl;
for (int y=1; y<=n_years; y++) report << NAA(y) << endl;
report << endl;
report << "Biomass Time Series" << endl;
report << "Year  TotJan1B  SSB  ExploitableB" << endl;
for (int y=1; y<=n_years; y++) report << y+year1-1 << "  " << TotJan1B(y) << "  " <<
   SSB(y) << "  " << ExploitableB(y) << endl;
report << endl;

int SR_year = 1;
if(SR_ratio_0_type != 1) SR_year = n_years;
if(SR_year == 1) report << "F Reference Points in First Year" << endl;
else report << "F Reference Points in Final Year" << endl;
report << "refpt     F          slope to plot on SR" << endl;
report << "F0.1      " << F01_report(SR_year) << "  " << F01_slope(SR_year) << endl;
report << "Fmax      " << Fmax_report(SR_year) << "  " << Fmax_slope(SR_year) << endl;
for(int i = 1; i <= nXSPR; i++) report << "F" << XSPR(i) << "%SPR   " << FXSPR_report(
   i,SR_year) << "  " << FXSPR_slope(i,SR_year) << endl;
report << "Fmsy      " << Fmsy_report(SR_year) << "  " << Fmsy_slope(SR_year) << "
   SSBmsy    " << SSBmsy_report(SR_year) << "     MSY   " << MSY(SR_year) << endl;
```

```
report << "  F " << Freport(SR_year) << "   " << F_slope(SR_year) << endl;
report << endl;

if(SR_year == 1) report << "Stock-Recruitment Relationship Parameters in First Year"
    << endl;
else report << "Stock-Recruitment Relationship Parameters in Final Year" << endl;
report << " alpha     = " << SR_alpha(SR_year) << endl;
report << " beta      = " << SR_beta(SR_year) << endl;
report << " R0        = " << SR_R0(SR_year) << endl;
report << " S0        = " << SR_S0(SR_year) << endl;
report << " steepness = " << steepness(SR_year) << 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 (int y=1; y<n_years; y++)
{
  report << y+year1-1 << "   " << SSB(y) << "   " << recruits(y+1) << "   " <<
      SR_pred_recruits(y+1) << "   " << SR_stdresid(y+1) << endl;
}
report << n_years+year1-1 << "   " << SSB(n_years) << "       xxxx    " <<
    SR_pred_recruits(n_years+1) << endl;
report << endl;

report << "Annual stock recruitment parameters and those used in stock recruit
    function" << endl;
report << "Year, SR_S0, SR_R0, steepness, alpha, beta, SR_ratio_0, S0_used, R0_used,
    alpha_used, beta_used" << endl;
for (int y=1; y<=n_years; y++)
{
  report << y+year1-1 << "   ";
  if(is_SR_scalar_R==1) report << SR_R0(y)*SR_ratio_0(y) << "   " << SR_R0(y) << "   ";
  else report << SR_S0(y) << "   " << SR_S0(y)/SR_ratio_0(y) << "   ";
  report << steepness(y) << "   ";
  if(is_SR_scalar_R==1) report << SR_alpha(y) << "   " << SR_R0(y)*SR_ratio_0(y)*(1.0-
      steepness(y))/(5.0*steepness(y)-1.0) << "   ";
  else report << 4.0*(steepness(y)*SR_S0(y)/SR_ratio_0(y))/(5.0*steepness(y)-1.0) << "
        " << SR_beta(y) << "   ";
  report << SR_ratio_0(y) << "   " << SR_S0(y) << "   " << SR_R0(y) << "   " << SR_alpha(
      y) << "   " << SR_beta(y) << endl;
}
report << endl;

report << "Annual reference points and slopes" << endl;
report << "Year  Fmsy  Fmsy_slope  SSBmsy  MSY  F01  F01_slope  Fmax  Fmax_slope";
for(int i = 1; i <= nXSPR; i++) report << "  F" << XSPR(i) << "  F" << XSPR(i) << "
    _slope";
report << endl;

for (int y=1; y<=n_years; y++)
{
  report << y+year1-1 << "   " << Fmsy_report(y) << "   " << Fmsy_slope(y) << "   " <<
```

```
      SSBmsy(y) << "   " << MSY(y) << " " <<
  F01_report(y) << "   " << F01_slope(y) << "   " << Fmax(y) << "   " << Fmax_slope(y);
  for(int i = 1; i <= nXSPR; i++) report  << "   " << FXSPR(i,y) << "   " << FXSPR_slope
      (i,y);
  report << endl;
}
report << endl;

report << "Root Mean Square Error computed from Standardized Residuals" << endl;
report << "Component                    #resids          RMSE" << endl;
dvar_vector catch_rmse(1,n_fleets);
dvariable catch_tot_rmse = 0.0;
catch_rmse = 0.0;
for (int i=1; i<=n_fleets; i++) if(n_catch_years(i) > 0)
{
  catch_rmse(i) = sqrt(mean(square(catch_tot_stdresid(i)(catch_times(i)))));
  report << "Catch_Fleet_" << i << "               " << n_catch_years(i) << "
                  " << catch_rmse(i) << endl;
  catch_tot_rmse += norm2(catch_tot_stdresid(i)(catch_times(i)));
}
if(sum(n_catch_years)>0)
{
  catch_tot_rmse = sqrt(catch_tot_rmse/double(sum(n_catch_years)));
  report << "Catch_Fleet_Total          " << sum(n_catch_years) << "                " <<
      catch_tot_rmse << endl;
}
dvar_vector discard_rmse(1,n_fleets);
dvariable discard_tot_rmse = 0.0;
discard_rmse = 0.0;
for (int i=1; i<=n_fleets; i++)
{
  if(n_discard_years(i) > 0)
  {
    discard_rmse(i) = sqrt(mean(square(discard_tot_stdresid(i)(discard_times(i)))));
    report << "Discard_Fleet_" << i << "             " << n_discard_years(i) << "
                  " <<  discard_rmse(i) << endl;
    discard_tot_rmse += norm2(discard_tot_stdresid(i)(discard_times(i)));
  }
}
if (sum(n_discard_years)>0)
{
  discard_tot_rmse = sqrt(discard_tot_rmse/double(sum(n_discard_years)));
  report << "Discard_Fleet_Total        " << sum(n_discard_years) << "                "
      << discard_tot_rmse << endl;
}
dvar_vector index_rmse(1,n_indices);
dvariable index_tot_rmse = 0.0;
index_rmse = 0.0;
for (int i=1; i<=n_indices; i++)
{
  if(n_index_years(i)>0)
  {
```

```
    index_rmse(i) = sqrt(mean(square(index_stdresid(i)(index_times(i)))));
    report << "Index_" << i << "                    " << n_index_years(i) << "
                    " << index_rmse(i) << endl;
    index_tot_rmse += norm2(index_stdresid(i)(index_times(i)));
  }
}
if (sum(n_index_years)>0)
{
  index_tot_rmse = sqrt(index_tot_rmse/double(sum(n_index_years)));
  report << "Index_Total              " << sum(n_index_years) << "                " <<
      index_tot_rmse << endl;
}

dvar_vector selpars_rmse(1,n_selpars);
dvar_vector selpars_rmse_n(1,n_selpars);
dvariable selpars_tot_rmse = 0.0;
selpars_rmse_n = 0.0;
selpars_rmse = 0.0;
for(int i=1; i<= n_selpars; i++)
{
  if(fabs(selpars_penalty(i)) > 1.0e-15)
  {
    selpars_rmse(i) = sqrt(square(selpars_stdresid(i)));
    selpars_rmse_n(i) = 1.0;
    report << "Selpar_" << i << "          " << selpars_rmse_n(i) << "           " <<
        selpars_rmse(i) << endl;
    selpars_tot_rmse +=  square(selpars_stdresid(i));
  }
}
if(sum(fabs(selpars_penalty)) > 1.0e-15)
{
  selpars_tot_rmse = sqrt(selpars_tot_rmse/sum(selpars_rmse_n));
  report << "Selpar_Total              " << sum(selpars_rmse_n) << "             " <<
      selpars_tot_rmse << endl;
}
dvariable N_year1_rmse = 0.0;
if (fabs(N_year1_penalty) > 1.0e-15)
{
  N_year1_rmse = sqrt(mean(square(N_year1_stdresid)));
  report << "Nyear1                       " << n_ages << "                " <<
      N_year1_rmse << endl;
}

dvar_vector Fmult_year1_rmse(1,n_fleets);
dvar_vector Fmult_year1_rmse_n(1,n_fleets);
dvariable Fmult_year1_tot_rmse = 0.0;
Fmult_year1_rmse_n = 0.0;
Fmult_year1_rmse = 0.0;
for(int i = 1; i <= n_fleets; i++)
{
  if(fabs(Fmult_year1_penalty(i)) > 1.0e-15)
  {
```

```
    Fmult_year1_rmse(i) = sqrt(square(Fmult_year1_stdresid(i)));
    Fmult_year1_rmse_n(i) = 1.0;
    report << "Fmult_Year1_fleet_" << i << "                    " << Fmult_year1_rmse_n(
       i) << "            " << Fmult_year1_rmse(i) << endl;
    Fmult_year1_tot_rmse += square(Fmult_year1_stdresid(i));
  }
}
if(sum(fabs(Fmult_year1_penalty))  > 1.0e-15)
{
  Fmult_year1_tot_rmse  = sqrt(Fmult_year1_tot_rmse/sum(Fmult_year1_rmse_n));
  report << "Fmult_year1_Total              " << sum(Fmult_year1_rmse_n) << "
              " << Fmult_year1_tot_rmse << endl;
}

dvar_vector Fmult_devs_rmse(1,n_fleets);
dvar_vector Fmult_devs_rmse_n(1,n_fleets);
dvariable Fmult_devs_tot_rmse = 0.0;
Fmult_devs_rmse_n = 0.0;
Fmult_devs_rmse = 0.0;
for (int i=1; i<= n_fleets; i++) if (fabs(Fmult_devs_penalty(i)) > 1.0e-15)
{
  Fmult_devs_rmse(i) = sqrt(mean(square(Fmult_devs_stdresid(i))));
  Fmult_devs_rmse_n(i) = catch_time_span(i);
  report << "Fmult_devs_fleet_" << i << "           " << Fmult_devs_rmse_n(i) << "
              " << Fmult_devs_rmse(i) << endl;
  Fmult_devs_tot_rmse += norm2(Fmult_devs_stdresid(i));
}
if (sum(fabs(Fmult_devs_penalty)) > 1.0e-15)
{
  Fmult_devs_tot_rmse = sqrt(Fmult_devs_tot_rmse/sum(Fmult_devs_rmse_n));
  report << "Fmult_devs_Total              " << sum(Fmult_devs_rmse_n) << "             "
     << Fmult_devs_tot_rmse << endl;
}
dvar_vector availability_rmse(1,n_indices);
dvar_vector availability_rmse_n(1,n_indices);
dvariable availability_tot_rmse = 0.0;
availability_rmse_n = 0.0;
availability_rmse = 0.0;
for(int i=1; i<= n_indices; i++) if(fabs(availability_penalty(i)) > 1.0e-15)
{
  availability_rmse(i) = sqrt(square(availability_stdresid(i)));
  availability_rmse_n(i) = 1.0;
  report << "Availability_index_" << i << "           " << availability_rmse_n(i) << "
             " << availability_rmse(i) << endl;
  availability_tot_rmse +=  square(availability_stdresid(i));
}
if(sum(fabs(availability_penalty)) > 1.0e-15)
{
  availability_tot_rmse = sqrt(availability_tot_rmse/sum(availability_rmse_n));
  report << "Availability_Total              " << sum(availability_rmse_n) << "
              " << availability_tot_rmse << endl;
}
```

```
dvar_vector availability_AR1_rmse(1,n_indices);
dvar_vector availability_AR1_rmse_n(1,n_indices);
dvariable availability_AR1_tot_rmse = 0.0;
availability_AR1_rmse_n = 0.0;
availability_AR1_rmse = 0.0;
for(int i=1; i<= n_indices; i++) if(fabs(availability_AR1_penalty(i)) > 1.0e-15)
{
  availability_AR1_rmse(i) = sqrt(mean(square(availability_AR1_stdresid(i))));
  availability_AR1_rmse_n(i) = index_time_span(i);
  report << "Availability_AR1_index_" << i << "          " << availability_AR1_rmse_n(
     i) << "          " << availability_AR1_rmse(i) << endl;
  availability_AR1_tot_rmse += norm2(availability_AR1_stdresid(i));
}
if(sum(fabs(availability_AR1_penalty)) > 1.0e-15)
{
  availability_AR1_tot_rmse = sqrt(availability_AR1_tot_rmse/sum(
     availability_AR1_rmse_n));
  report << "Availability_AR1_Total              " << sum(availability_AR1_rmse_n) << "
               " << availability_AR1_tot_rmse << endl;
}
dvar_vector efficiency_rmse(1,n_indices);
dvar_vector efficiency_rmse_n(1,n_indices);
dvariable efficiency_tot_rmse = 0.0;
efficiency_rmse_n = 0.0;
efficiency_rmse = 0.0;
for(int i=1; i<= n_indices; i++) if(fabs(efficiency_penalty(i)) > 1.0e-15)
{
  efficiency_rmse(i) = sqrt(square(efficiency_stdresid(i)));
  efficiency_rmse_n(i) = 1.0;
  report << "Efficiency_index_" << i << "          " << efficiency_rmse_n(i) << "
             " << efficiency_rmse(i) << endl;
  efficiency_tot_rmse += square(efficiency_stdresid(i));
}
if(sum(fabs(efficiency_penalty)) > 1.0e-15)
{
  efficiency_tot_rmse = sqrt(efficiency_tot_rmse/sum(efficiency_rmse_n));
  report << "Efficiency_Total              " << sum(efficiency_rmse_n) << "            "
      << efficiency_tot_rmse << endl;
}

dvariable recruit_devs_rmse = 0.0;
dvariable SR_scalar_rmse = 0.0;
dvariable steepness_rmse = 0.0;
if (fabs(SR_penalty) > 1.0e-15)
{
  recruit_devs_rmse = sqrt(mean(square(SR_stdresid)));
  report << "Recruit_devs              " << n_years << "             " <<
     recruit_devs_rmse << endl;
}
if (fabs(SR_scalar_penalty) > 1.0e-15)
{
  SR_scalar_rmse = sqrt(square(SR_scalar_stdresid));
```

```
  report << "SR_scalar                  " << 1       << "              " <<
     SR_scalar_rmse << endl;
}
if (fabs(steepness_penalty) > 1.0e-15)
{
  steepness_rmse = sqrt(square(steepness_stdresid));
  report << "steepness                  " << 1       << "              " <<
     steepness_rmse << endl;
}
report << endl;

report << "Stage2 Multipliers for Multinomials and estimated effective sample sizes (
    Francis 2011)" << endl;
if(sum(Neff_stage2_mult_catch) > 1.0e-15)
{
  report << "Catch age composition" << endl;
  report << "fleet  multiplier";
  for(int y=1; y<=n_years;y++) report << "  " << year1 + y -1;
  report << endl;
  for(int i=1;i<=n_fleets;i++)
  {
    report << i << "  " << Neff_stage2_mult_catch(i);
    for(int y=1; y<=n_years;y++) report << "  " << Neff_stage2_mult_catch(i)*
       input_Neff_catch(i,y);
    report << endl;
  }
  report << endl;
}
if(sum(Neff_stage2_mult_discard) > 1.0e-15)
{
  report << "Discard age composition" << endl;
  report << "fleet  multiplier";
  for(int y=1; y<=n_years;y++) report << "  " << year1 + y -1;
  report << endl;
  for(int i=1;i<=n_fleets;i++)
  {
    report << i << "  " << Neff_stage2_mult_discard(i);
    for(int y=1; y<=n_years;y++) report << "  " << Neff_stage2_mult_discard(i)*
       input_Neff_discard(i,y);
    report << endl;
  }
  report << endl;
}
if(sum(Neff_stage2_mult_index) > 1.0e-15)
{
  report << "Index age composition" << endl;
  report << "index  multiplier";
  for(int y=1; y<=n_years;y++) report << "  " << year1 + y -1;
  report << endl;
  for(int i=1;i<=n_indices;i++)
  {
    report << i << "  " << Neff_stage2_mult_index(i);
```

```
    for(int y=1; y<=n_years;y++) report << "  " << Neff_stage2_mult_index(i)*
        input_Neff_index(i,y);
    report << endl;
  }
  report << endl;
}

dvar_matrix rel_efficiency_ini(1,n_rel_efficiency_penalties,1,n_lengths);
rel_efficiency_ini = 0.0;
dvar_matrix rel_efficiency(1,n_rel_efficiency_penalties,1,n_lengths);
rel_efficiency = 0.0;
if(calibrate_indices == 1)
{
  report << "Relative catch efficiencies" << endl;
  report << "Initial relative catch efficiencies" << endl;
  report << "Penalty";
  for(int j=1;j<=n_lengths; j++) report << "  " << j;
  report << endl;
  for(int i=1;i<=n_rel_efficiency_penalties; i++)
  {
    report << i;
    for(int j=1;j<=n_lengths; j++)
    {
      rel_efficiency_ini(i,j) = mfexp(rel_efficiency_X(i,j)*rel_efficiency_coef_ini(i)
          );
      report << "  " << rel_efficiency_ini(i,j);
    }
    report << endl;
  }

  report << "Estimated relative catch efficiencies" << endl;
  report << "Penalty";
  for(int j=1;j<=n_lengths; j++) report << "  " << j;
  report << endl;
  for(int i=1;i<=n_rel_efficiency_penalties; i++)
  {
    report << i;
    for(int j=1;j<=n_lengths; j++)
    {
      rel_efficiency(i,j) = mfexp(rel_efficiency_X(i,j)*rel_efficiency_coef(i));
      report << "  " << rel_efficiency(i,j);
    }
    report << 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/SSBmsy" << endl;
    for (int y=1; y<=nprojyears; y++)
    {
      report << year1+n_years-1+y << "  " << proj_total_yield(y) << "  " <<
         proj_total_discard(y) << "  " <<
        proj_TotJan1B(y) << "  " << proj_SSB(y) << "  " << proj_what(y) << "  " <<
          proj_SSB(y)/SSBmsy(y) << 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_asap4.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;
```

6.2 Auxiliary Code for Exporting Output to R

```
// 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 ichar;
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", "ASAP4");
  close_r_info_list();


  // basic parameter values
  open_r_info_list("dimensions", false);
          wrt_r_item("styr", year1);
          wrt_r_item("endyr", (year1+n_years-1));
          wrt_r_item("n_years", n_years);
          wrt_r_item("n_ages", n_ages);
          wrt_r_item("n_lengths", n_lengths);
          wrt_r_item("n_fleets", n_fleets);
          wrt_r_item("n_selblocks", n_selblocks);
          wrt_r_item("n_indices", n_indices);
  close_r_info_list();
  wrt_r_complete_vector("n_index_years",n_index_years);
  wrt_r_complete_vector("index_min_time",index_min_time);
  wrt_r_complete_vector("index_max_time",index_max_time);
  wrt_r_complete_vector("n_index_age_comp_years",n_index_age_comp_years);
  wrt_r_complete_vector("n_catch_years",n_catch_years);
  wrt_r_complete_vector("n_catch_age_comp_years",n_catch_age_comp_years);
  wrt_r_complete_vector("n_discard_years",n_discard_years);
  wrt_r_complete_vector("n_discard_age_comp_years",n_discard_age_comp_years);
  if(sum(n_index_years)>0)
  {
          open_r_list("index_times");
          for (int i=1; i<=n_indices; i++)
          {
              if (n_indices < 10) sprintf(onenum, "%d", i);
              else onenum="0";
              ichar = "index" + onenum;
              if(n_index_years(i)>0) wrt_r_complete_vector(ichar,index_times(i));
          }
```

```
        close_r_list();
}
if(sum(n_index_age_comp_years)>0)
{
        open_r_list("index_age_comp_times");
        for (int i=1; i<=n_indices; i++)
        {
            if (n_indices < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "index" + onenum;
            if(n_index_age_comp_years(i)>0) wrt_r_complete_vector(ichar,
               index_age_comp_times(i));
        }
        close_r_list();
}
if(sum(n_catch_years)>0)
{
        open_r_list("catch_times");
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            if(n_catch_years(i)>0) wrt_r_complete_vector(ichar,catch_times(i));
        }
        close_r_list();
}
if(sum(n_catch_age_comp_years)>0)
{
        open_r_list("catch_age_comp_times");
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            if(n_catch_age_comp_years(i)>0) wrt_r_complete_vector(ichar,
               catch_age_comp_times(i));
        }
        close_r_list();
}
if(sum(n_discard_years)>0)
{
        open_r_list("discard_times");
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            if(n_discard_years(i)>0) wrt_r_complete_vector(ichar,discard_times(i));
        }
        close_r_list();
}
```

```
if(sum(n_discard_age_comp_years)>0)
{
        open_r_list("discard_age_comp_times");
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            if(n_discard_age_comp_years(i)>0) wrt_r_complete_vector(ichar,
                discard_age_comp_times(i));
        }
        close_r_list();
}

// run options
open_r_info_list("options",false);
        wrt_r_item("isfecund", isfecund);
        wrt_r_item("frac_yr_spawn", fracyearSSB);
        wrt_r_item("estimate_M", estimate_M);
        wrt_r_item("calibrate_indices", calibrate_indices);
        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("Fmult_max_value", Fmult_max_value);
        wrt_r_item("use_Fmult_max_penalty", use_Fmult_max_penalty);
        wrt_r_item("use_F_penalty", use_F_penalty);
        wrt_r_item("N_year1_type",NAA_year1_flag);
        wrt_r_item("is_SR_scalar_R",is_SR_scalar_R);
        wrt_r_item("SR_ratio_0_type", SR_ratio_0_type);
        wrt_r_item("SR_model_type", SR_model_type);
        wrt_r_item("do_mcmc",doMCMC);
close_r_info_list();
wrt_r_complete_vector("release_mort", release_mort);
wrt_r_complete_vector("use_index",use_index);
wrt_r_complete_vector("use_index_age_comp",use_index_age_comp);
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);
open_r_matrix("index_month");
  wrt_r_matrix(trans(index_month), 2, 2);
  wrt_r_namevector(year1, year1+n_years-1);
  wrt_r_namevector(1, n_indices);
close_r_matrix();

// Likelihood contributions
open_r_info_list("obj_fun",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_index*index_likely));
wrt_r_item("lk_catch_age_comp_total", sum(catch_age_comp_likely));
wrt_r_item("lk_discards_age_comp_total", sum(discard_age_comp_likely));
wrt_r_item("lk_index_age_comp_total", sum(index_age_comp_likely));
wrt_r_item("pen_sel_pars_total", sum_sel_lambda_penalty);
wrt_r_item("pen_availability", (lambda_availability*availability_penalty));
wrt_r_item("pen_availability_AR1", (lambda_availability_AR1*
   availability_AR1_penalty));
wrt_r_item("pen_effiicency", (lambda_efficiency*efficiency_penalty));
wrt_r_item("pen_rel_effiicency", (lambda_rel_efficiency*rel_efficiency_penalty
   ));
wrt_r_item("pen_Fmult_year1_total", (lambda_Fmult_year1*Fmult_year1_penalty));
wrt_r_item("pen_Fmult_devs_total", (lambda_Fmult_devs*Fmult_devs_penalty));
wrt_r_item("pen_N_year1", (lambda_N_year1_devs*N_year1_penalty));
wrt_r_item("pen_recruit_devs", (lambda_recruit_devs*SR_penalty));
wrt_r_item("pen_steepness", (lambda_steepness*steepness_penalty));
wrt_r_item("pen_SR_scalar", (lambda_SR_scalar*SR_scalar_penalty));
wrt_r_item("pen_Fmult_max", Fmult_max_penalty);
wrt_r_item("pen_F", fpenalty);
// fleet, block, and index specific likelihood contributions
if (n_fleets>1)
{
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring lk_catch_fleet = adstring("lk_catch_") + ichar;
        wrt_r_item(lk_catch_fleet,(lambda_catch_tot(i)*catch_tot_likely(i)));
    }
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring lk_catch_age_comp = adstring("lk_catch_age_comp_") + ichar;
        wrt_r_item(lk_catch_age_comp,(catch_age_comp_likely(i)));
    }

    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring lk_discard_fleet = adstring("lk_discard_") + ichar;
        wrt_r_item(lk_discard_fleet,(lambda_discard_tot(i)*discard_tot_likely(
           i)));
    }
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
```

```
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring lk_discard_age_comp = adstring("lk_discard_age_comp_") +
           ichar;
        wrt_r_item(lk_discard_age_comp,(discard_age_comp_likely(i)));
    }

    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring lk_Fmult_year1_fleet = adstring("lk_Fmult_year1_") + ichar;
        wrt_r_item(lk_Fmult_year1_fleet,(lambda_Fmult_year1(i)*
           Fmult_year1_penalty(i)));
    }

    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring lk_Fmult_devs_fleet = adstring("lk_Fmult_devs_") + ichar;
        wrt_r_item(lk_Fmult_devs_fleet,(lambda_Fmult_devs(i)*
           Fmult_devs_penalty(i)));
    }
}

if (n_indices>1)
{
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
             separately
        {
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        adstring lk_index_fit_ind = adstring("lk_index_fit_") + twodnm;
        wrt_r_item(lk_index_fit_ind,(lambda_index(i)*index_likely(i)));
    }
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
```

```
        separately
    {
        sprintf(onednm, "%d", i);
        twodnm = "0" + onednm;
    }
    else if (i <=99)
    {
        sprintf(twodnm, "%d", i);
    }
    else
    {
        twodnm = "00";
    }
    adstring lk_index_age_comp = adstring("lk_index_age_comp_") + twodnm;
    wrt_r_item(lk_index_age_comp,(index_age_comp_likely(i)));
}

for (int i=1; i<=n_indices; i++)
{
    if (i <= 9)  // note have to deal with one digit and two digit numbers
        separately
    {
        sprintf(onednm, "%d", i);
        twodnm = "0" + onednm;
    }
    else if (i <=99)
    {
        sprintf(twodnm, "%d", i);
    }
    else
    {
        twodnm = "00";
    }
    adstring lk_avail_ind = adstring("pen_availability_") + twodnm;
    wrt_r_item(lk_avail_ind,(lambda_availability(i)*availability_penalty(i
       )));
}
for (int i=1; i<=n_indices; i++)
{
    if (i <= 9)  // note have to deal with one digit and two digit numbers
        separately
    {
        sprintf(onednm, "%d", i);
        twodnm = "0" + onednm;
    }
    else if (i <=99)
    {
        sprintf(twodnm, "%d", i);
    }
    else
    {
        twodnm = "00";
```

```
        }
        adstring lk_avail_AR1 = adstring("pen_availability_AR1_") + twodnm;
        wrt_r_item(lk_avail_AR1,(lambda_availability_AR1(i)*
            availability_AR1_penalty(i)));
    }
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
             separately
        {
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        adstring lk_eff_ind = adstring("pen_efficiency_") + twodnm;
        wrt_r_item(lk_eff_ind,(lambda_efficiency(i)*efficiency_penalty(i)));
    }
    for (int i=1; i<=n_rel_efficiency_penalties; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
             separately
        {
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        adstring lk_rel_eff = adstring("pen_rel_efficiency_") + twodnm;
        wrt_r_item(lk_rel_eff,(rel_efficiency_penalty(i)));
    }
}

for (int i=1; i<=n_selpars;i++)
{
    if (phase_selpars(i) >=1 && lambda_selpars(i) > 1.0e-15)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
             separately
        {
```

```
                sprintf(onednm, "%d", i);
                twodnm = "0" + onednm;
            }
            else if (i <=99)
            {
                sprintf(twodnm, "%d", i);
            }
            else
            {
                twodnm = "00";
            }
            adstring lk_sel_pars = adstring("pen_sel_par_") + twodnm;
            wrt_r_item(lk_sel_pars,(lambda_selpars(i)*selpars_penalty(i)));

        }
    }
close_r_info_list(); //close objective function components list

wrt_r_complete_vector("availability_penalty_type",availability_penalty_type);
wrt_r_complete_vector("efficiency_penalty_type",efficiency_penalty_type);
open_r_info_list("steepness_penalty_type",false);
    wrt_r_item("steepness",steepness_penalty_type);
close_r_info_list();
wrt_r_complete_vector("selpars_penalty_type",selpars_penalty_type);

wrt_r_complete_vector("M_year_pars_ini",M_year_pars_ini);
wrt_r_complete_vector("M_age_pars_ini",M_age_pars_ini);
wrt_r_complete_vector("NAA_year1_ini",NAA_year1_ini);
wrt_r_complete_vector("Fmult_year1_ini",Fmult_year1_ini);
wrt_r_complete_vector("availability_ini",availability_ini);
wrt_r_complete_vector("availability_pars_ini",availability_pars_ini);
wrt_r_complete_vector("efficiency_ini",efficiency_ini);
wrt_r_complete_vector("efficiency_pars_ini",efficiency_pars_ini);
open_r_list("rel_efficiency_coef_ini");
for (int i=1; i<=n_rel_efficiency_penalties;i++)
{
    if (i <= 9)  // note have to deal with one digit and two digit numbers separately
    {
        sprintf(onednm, "%d", i);
        twodnm = "0" + onednm;
    }
    else if (i <=99)
    {
        sprintf(twodnm, "%d", i);
    }
    else
    {
        twodnm = "00";
    }
    adstring rel_eff = adstring("rel_eff_") + twodnm;
        wrt_r_complete_vector(rel_eff,rel_efficiency_coef_ini(i));
}
```

```
close_r_list();
wrt_r_complete_vector("selpars_ini",selpars_ini);
open_r_info_list("SR_scalar_ini",false);
    wrt_r_item("SR_scalar",SR_scalar_ini);
close_r_info_list();
open_r_info_list("steepness_ini",false);
        wrt_r_item("steepness",steepness_ini);
close_r_info_list();
open_r_info_list("Fmult_max",false);
        wrt_r_item("Fmult_max",Fmult_max_value_ini);
close_r_info_list();
wrt_r_complete_vector("SR_scalar_pars_ini",SR_scalar_pars_ini);
wrt_r_complete_vector("steepness_pars_ini",steepness_pars_ini);

open_r_info_list("NAA_year1_CV",false);
        wrt_r_item("NAA_year1",N_year1_CV);
close_r_info_list();
open_r_info_list("SR_scalar_CV",false);
        wrt_r_item("SR_scalar",SR_scalar_CV);
close_r_info_list();
open_r_info_list("steepness_CV",false);
        wrt_r_item("steepness",steepness_penalty_CV);
close_r_info_list();
wrt_r_complete_vector("Fmult_year1_CV",Fmult_year1_CV);
wrt_r_complete_vector("Fmult_devs_CV",Fmult_devs_CV);
wrt_r_complete_vector("availability_CV",availability_penalty_CV);
wrt_r_complete_vector("efficiency_CV",efficiency_penalty_CV);
wrt_r_complete_vector("selpars_CV",selpars_penalty_CV);
wrt_r_complete_vector("recruit_CV",recruit_CV);

wrt_r_complete_vector("availability_AR1_sd",availability_AR1_sd);

open_r_list("rel_efficiency_coef_var");
for (int i=1; i<=n_rel_efficiency_penalties;i++)
{
    if (i <= 9)  // note have to deal with one digit and two digit numbers separately
    {
        sprintf(onednm, "%d", i);
        twodnm = "0" + onednm;
    }
    else if (i <=99)
    {
        sprintf(twodnm, "%d", i);
    }
    else
    {
        twodnm = "00";
    }
    adstring rel_eff_vars = adstring("rel_efficiency_var_") + twodnm;
    open_r_matrix(rel_eff_vars);
         wrt_r_matrix(var_rel_efficiency_coef(i), 2, 2);
     wrt_r_namevector(1,n_rel_efficiency_coef(i));
```

```
      wrt_r_namevector(1,n_rel_efficiency_coef(i));
     close_r_matrix();
}
close_r_list();

open_r_info_list("N_year1_devs_phase",false);
        wrt_r_item("N_year1_devs", phase_N_year1_devs);
close_r_info_list();
open_r_info_list("recruitment_phase",false);
        wrt_r_item("recruitment", phase_recruit_devs);
close_r_info_list();
open_r_info_list("SR_scalar_phase",false);
        wrt_r_item("SR_scalar", phase_SR_scalar);
close_r_info_list();
open_r_info_list("steepness_phase",false);
        wrt_r_item("steepness", phase_steepness);
close_r_info_list();
wrt_r_complete_vector("Fmult_year1_phase", phase_Fmult_year1);
wrt_r_complete_vector("M_year_pars_phase",phase_M_year_pars);
wrt_r_complete_vector("M_age_pars_phase",phase_M_age_pars);
wrt_r_complete_vector("Fmult_devs_phase", phase_Fmult_devs);
wrt_r_complete_vector("availability_pars_phase", phase_availability_pars);
wrt_r_complete_vector("availability_AR1_phase", phase_availability_AR1);
wrt_r_complete_vector("efficiency_pars_phase", phase_efficiency_pars);
wrt_r_complete_vector("rel_efficiency_phase", phase_rel_efficiency);
wrt_r_complete_vector("selpars_phase", phase_selpars);

open_r_info_list("N_year1_devs_lambda",false);
        wrt_r_item("N_year1_devs", lambda_N_year1_devs);
close_r_info_list();
open_r_info_list("recruitment_lambda",false);
        wrt_r_item("recruitment",lambda_recruit_devs);
close_r_info_list();
open_r_info_list("steepness_lambda",false);
        wrt_r_item("steepness",lambda_steepness);
close_r_info_list();
open_r_info_list("SR_scalar_lambda",false);
        wrt_r_item("SR_scalar",lambda_SR_scalar);
close_r_info_list();
open_r_info_list("Fmult_max_lambda",false);
        wrt_r_item("Fmult_max",lambda_Fmult_max_penalty);
close_r_info_list();
open_r_info_list("fpenalty_lambda",false);
        wrt_r_item("fpenalty",lambda_fpenalty);
close_r_info_list();
wrt_r_complete_vector("Fmult_year1_lambda", lambda_Fmult_year1);
wrt_r_complete_vector("Fmult_devs_lambda", lambda_Fmult_devs);
wrt_r_complete_vector("availability_lambda",lambda_availability);
wrt_r_complete_vector("availability_AR1_lambda",lambda_availability_AR1);
wrt_r_complete_vector("efficiency_lambda",lambda_efficiency);
wrt_r_complete_vector("rel_efficiency_lambda", lambda_rel_efficiency);
wrt_r_complete_vector("selpars_lambda", lambda_selpars);
```

```
open_r_list("parameter_info");
    // selectivity input matrices for fleets and indices
    // input selectivity specifications
    open_r_matrix("fleet_selblock_pointer");
       wrt_r_matrix(fleet_selblock_pointer_ini, 2, 2);
       wrt_r_namevector(year1, (year1+n_years-1));
       wrt_r_namevector(1, n_fleets);
    close_r_matrix();
    open_r_matrix("index_selblock_pointer_ini");
       wrt_r_matrix(index_selblock_pointer_ini, 2, 2);
       wrt_r_namevector(year1, (year1+n_years-1));
       wrt_r_namevector(1, n_indices);
    close_r_matrix();
    wrt_r_complete_vector("selblock_type",selblock_type);
    wrt_r_complete_vector("selpars_lower",selpars_lower);
    wrt_r_complete_vector("selpars_upper",selpars_upper);
    wrt_r_complete_vector("availability_lower",availability_lower);
    wrt_r_complete_vector("availability_upper",availability_upper);
    wrt_r_complete_vector("efficiency_lower",efficiency_lower);
    wrt_r_complete_vector("efficiency_upper",efficiency_upper);
close_r_list(); //close parameter_info list

open_r_info_list("mcmc_info", 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();

// Weight at Age matrices
open_r_list("WAA_mats");
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring WAA_c_fleet = adstring("WAA_catch_") + ichar;
        open_r_matrix(WAA_c_fleet);
            wrt_r_matrix(WAAcatchfleet(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1,n_ages);
        close_r_matrix();
        adstring WAA_d_fleet = adstring("WAA_discard_") + ichar;
        open_r_matrix(WAA_d_fleet);
            wrt_r_matrix(WAAdiscardfleet(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1,n_ages);
        close_r_matrix();
```

```
    }
    open_r_matrix("WAA_catch_all");
        wrt_r_matrix(WAAcatchall, 2, 2);
        wrt_r_namevector(year1, (year1+n_years-1));
        wrt_r_namevector(1, n_ages);
    close_r_matrix();

    open_r_matrix("WAA_discard_all");
        wrt_r_matrix(WAAdiscardall, 2, 2);
        wrt_r_namevector(year1, (year1+n_years-1));
        wrt_r_namevector(1, n_ages);
    close_r_matrix();

    open_r_matrix("WAA_ssb");
        wrt_r_matrix(WAAssb, 2, 2);
        wrt_r_namevector(year1, (year1+n_years-1));
        wrt_r_namevector(1, n_ages);
    close_r_matrix();

    open_r_matrix("WAA_jan1");
        wrt_r_matrix(WAAjan1b, 2, 2);
        wrt_r_namevector(year1, (year1+n_years-1));
        wrt_r_namevector(1, n_ages);
    close_r_matrix();

    for (int i=1; i<=n_indices; i++)
    {
        if (index_units_aggregate(i)==1 || index_units_proportions(i)==1)
        {
            if (i <= 9)  // note have to deal with one digit and two digit numbers
                separately
            {
                sprintf(onednm, "%d", i);
                twodnm = "0" + onednm;
            }
            else if (i <=99)
            {
                sprintf(twodnm, "%d", i);
            }
            else
            {
                twodnm = "00";
            }
            adstring index_WAA_name = adstring("index_WAA_") + twodnm;
            open_r_matrix(index_WAA_name);
                wrt_r_matrix(index_WAA(i), 2, 2);
                wrt_r_namevector(year1, (year1+n_years-1));
                wrt_r_namevector(1,n_ages);
            close_r_matrix();
        }
    }
close_r_list(); // close WAA_mats list
```

```
// Year by Age Matrices (not fleet specific): M, maturity, fecundity, N, Z, F,
open_r_matrix("M_age");
    wrt_r_matrix(M_use, 2, 2);
    wrt_r_namevector(year1, (year1+n_years-1));
    wrt_r_namevector(1, n_ages);
close_r_matrix();

open_r_matrix("M_age_X");
    wrt_r_matrix(M_X_age, 2, 2);
    wrt_r_namevector(1,n_ages);
    wrt_r_namevector(1,n_M_age_cov);
close_r_matrix();

open_r_matrix("M_year_X");
    wrt_r_matrix(M_X_year, 2, 2);
    wrt_r_namevector(1,n_years);
    wrt_r_namevector(1,n_M_year_cov);
close_r_matrix();

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

open_r_matrix("fecundity");
    wrt_r_matrix(fecundity, 2, 2);
    wrt_r_namevector(year1, (year1+n_years-1));
    wrt_r_namevector(1, n_ages);
close_r_matrix();

open_r_matrix("N_age");
    wrt_r_matrix(NAA, 2, 2);
    wrt_r_namevector(year1, (year1+n_years-1));
    wrt_r_namevector(1, n_ages);
close_r_matrix();

open_r_matrix("Z_age");
    wrt_r_matrix(Z, 2, 2);
    wrt_r_namevector(year1, (year1+n_years-1));
    wrt_r_namevector(1, n_ages);
close_r_matrix();

open_r_matrix("F_age");
    wrt_r_matrix(FAA_tot, 2, 2);
    wrt_r_namevector(year1, (year1+n_years-1));
    wrt_r_namevector(1, n_ages);
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, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("catch_pred");
    wrt_r_matrix(catch_tot_fleet_pred, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("catch_std_resid");
    wrt_r_matrix(catch_tot_stdresid, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("discard_obs");
    wrt_r_matrix(discard_tot_fleet_obs, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("discard_pred");
    wrt_r_matrix(discard_tot_fleet_pred, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("discard_std_resid");
    wrt_r_matrix(discard_tot_stdresid, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("catch_tot_cv");
      wrt_r_matrix(catch_tot_CV, 2, 2);
      wrt_r_namevector(year1, (year1+n_years-1));
      wrt_r_namevector(1, n_fleets);
close_r_matrix();

open_r_matrix("discard_tot_cv");
      wrt_r_matrix(discard_tot_CV, 2, 2);
      wrt_r_namevector(year1, (year1+n_years-1));
      wrt_r_namevector(1, n_fleets);
close_r_matrix();

wrt_r_complete_vector("lambda_catch_tot",lambda_catch_tot);
wrt_r_complete_vector("lambda_discard_tot",lambda_discard_tot);

// Age Compositions: Catch and Discards observed and predicted by fleet
```

```
open_r_list("catch_comp_mats");
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring ccomp_ob = ichar + adstring("_ob");
        open_r_matrix(ccomp_ob);
            wrt_r_matrix(catch_paa_obs(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1, n_ages);
        close_r_matrix();

        adstring ccomp_pr = ichar + adstring("_pr");
        open_r_matrix(ccomp_pr);
            wrt_r_matrix(catch_paa_pred(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1, n_ages);
        close_r_matrix();
    }
close_r_list();

open_r_list("discard_comp_mats");
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring dcomp_ob = ichar + adstring("_ob");
        open_r_matrix(dcomp_ob);
            wrt_r_matrix(discard_paa_obs(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1, n_ages);
        close_r_matrix();

        adstring dcomp_pr = ichar + adstring("_pr");
        open_r_matrix(dcomp_pr);
            wrt_r_matrix(discard_paa_pred(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1, n_ages);
        close_r_matrix();
    }
close_r_list();

// proportion release year by age matrices by fleet
open_r_list("fleet_prop_release");
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring fleet_prop_release = ichar;
```

```
        open_r_matrix(fleet_prop_release);
            wrt_r_matrix(proportion_release(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1,n_ages);
        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_Neff_catch, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("fleet_catch_Neff_est");
    wrt_r_matrix(output_Neff_catch, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("fleet_discard_Neff_init");
    wrt_r_matrix(input_Neff_discard, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("fleet_discard_Neff_est");
    wrt_r_matrix(output_Neff_discard, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

// fleet selectivity blocks
open_r_matrix("fleet_sel_blocks");
    wrt_r_matrix(fleet_selblock_pointer, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

// selecivity matrices for each fleet
open_r_list("fleet_sel_mats");
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring sel_fleet_char = ichar;
        open_r_matrix(sel_fleet_char);
            wrt_r_matrix(sel_by_fleet(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
```

```
            wrt_r_namevector(1, n_ages);
        close_r_matrix();
    }
close_r_list();

// Fmults by fleet
open_r_matrix("fleet_Fmult");
    wrt_r_matrix(Fmult, 2, 2);
    wrt_r_namevector(1, n_fleets);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

// FAA by fleet directed and discarded
open_r_list("fleet_FAA");
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring fleet_FAA_dir = adstring("directed_") + ichar;
        open_r_matrix(fleet_FAA_dir);
            wrt_r_matrix(directed_FAA_by_fleet(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1,n_ages);
        close_r_matrix();

        adstring fleet_FAA_discard = adstring("discarded_") + ichar;
        open_r_matrix(fleet_FAA_discard);
            wrt_r_matrix(FAA_by_fleet_discard(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1,n_ages);
        close_r_matrix();
    }
close_r_list();

// index stuff starts here
// index observations
open_r_matrix("index_obs");
      wrt_r_matrix(index_obs, 2, 2);
      wrt_r_namevector(1, n_indices);
      wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();
// index predictions
open_r_matrix("index_pred");
      wrt_r_matrix(index_pred, 2, 2);
      wrt_r_namevector(1, n_indices);
      wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();
// index CV
open_r_matrix("index_cv");
      wrt_r_matrix(index_cv, 2, 2);
      wrt_r_namevector(1, n_indices);
```

```
        wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();
// index sigma
open_r_matrix("index_sigma");
        wrt_r_matrix(index_sigma, 2, 2);
        wrt_r_namevector(1, n_indices);
        wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

// index standardized residuals
open_r_matrix("index_std_resid");
        wrt_r_matrix(index_stdresid, 2, 2);
        wrt_r_namevector(1, n_indices);
        wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

wrt_r_complete_vector("lambda_index",lambda_index);

// Index Age Comp
open_r_list("index_comp_mats");
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
            separately
        {
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        ichar = "index" + twodnm;
        adstring acomp_ob = ichar + adstring("_ob");
        open_r_matrix(acomp_ob);
            wrt_r_matrix(index_paa_obs_use(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1,n_ages);
        close_r_matrix();

        adstring acomp_pr = ichar + adstring("_pr");
        open_r_matrix(acomp_pr);
            wrt_r_matrix(index_paa_pred(i), 2, 2);
            wrt_r_namevector(year1, (year1+n_years-1));
            wrt_r_namevector(1, n_ages);
        close_r_matrix();
    }
close_r_list();
```

```
// Neff for indices initial guess
open_r_matrix("index_Neff_init");
    wrt_r_matrix(input_Neff_index, 2, 2);
    wrt_r_namevector(1, n_indices);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

// Neff for indices estimated
open_r_matrix("index_Neff_est");
    wrt_r_matrix(output_Neff_index, 2, 2);
    wrt_r_namevector(1, n_indices);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

// index selectivity blocks
open_r_matrix("index_sel_blocks");
    wrt_r_matrix(index_selblock_pointer, 2, 2);
    wrt_r_namevector(1, n_indices);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("availability");
    wrt_r_matrix(availability, 2, 2);
    wrt_r_namevector(1, n_indices);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("efficiency");
    wrt_r_matrix(efficiency, 2, 2);
    wrt_r_namevector(1, n_indices);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();

open_r_matrix("q");
    wrt_r_matrix(q_by_index, 2, 2);
    wrt_r_namevector(1, n_indices);
    wrt_r_namevector(year1, (year1+n_years-1));
close_r_matrix();


// availability_X by index
open_r_list("availability_X");
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
            separately
        {
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
```

```
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        ichar = "index" + twodnm;
        adstring avail_X = ichar;
                open_r_matrix(avail_X);
                    wrt_r_matrix(availability_X(i), 2, 2);
                    wrt_r_namevector(1, n_years);
                    wrt_r_namevector(1, n_availability_pars(i));
                close_r_matrix();
        }
close_r_list();

// efficiency_X by index
open_r_list("efficiency_X");
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
            separately
        {
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        ichar = "index" + twodnm;
        adstring eff_X = ichar;
                open_r_matrix(eff_X);
                    wrt_r_matrix(efficiency_X(i), 2, 2);
                    wrt_r_namevector(1, n_years);
                    wrt_r_namevector(1, n_efficiency_pars(i));
                close_r_matrix();
        }
close_r_list();

// selectivity by index
open_r_list("index_sel_mats");
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
            separately
        {
```

```
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        ichar = "index" + twodnm;
        adstring index_sel = ichar;
                open_r_matrix(index_sel);
                    wrt_r_matrix(sel_by_index(i), 2, 2);
                    wrt_r_namevector(1, n_years);
                    wrt_r_namevector(1, n_ages);
                close_r_matrix();
        }
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_list("Fref");
    wrt_r_complete_vector("Fmax",Fmax_report);
    wrt_r_complete_vector("F01",F01_report);
        open_r_matrix("FXSPR");
          wrt_r_matrix(FXSPR_report, 2, 2);
      wrt_r_namevector(XSPR);
      wrt_r_namevector(1, n_years);
        close_r_matrix();
    wrt_r_complete_vector("F",Freport);
close_r_list();

// SR curve parameters
open_r_list("SR_parms");
    wrt_r_complete_vector("alpha",SR_alpha);
    wrt_r_complete_vector("beta",SR_beta);
    wrt_r_complete_vector("SPR0",SR_ratio_0);
    wrt_r_complete_vector("S0",SR_S0);
    wrt_r_complete_vector("R0",SR_R0);
    wrt_r_complete_vector("steepness",steepness);
close_r_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+n_years-1), 2);
    wrt_r_namevector(year1, (year1+n_years-1));
    wrt_r_df_col("year", year1, (year1+n_years-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();

// 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",NAA_year1_pred);
        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 (n_fleets>1)
    {
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            adstring rmse_n_catch_fleet = adstring("catch_") + ichar;
            wrt_r_item(rmse_n_catch_fleet,n_catch_years(i));
        }
    }
    wrt_r_item("catch_tot",sum(n_catch_years));

    if (n_fleets>1)
    {
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            adstring rmse_n_discard_fleet = adstring("discard_") + ichar;
            wrt_r_item(rmse_n_discard_fleet,n_discard_years(i));
        }
    }
    wrt_r_item("discard_tot",sum(n_discard_years));

    if (n_indices>1)
    {
```

```
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
           separately
        {
            sprintf(onednm, "%d", i);
            twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
            sprintf(twodnm, "%d", i);
        }
        else
        {
            twodnm = "00";
        }
        ichar = "index" + twodnm;
        adstring rmse_n_ind = ichar;
        wrt_r_item(rmse_n_ind,n_index_years(i));
    }
}
wrt_r_item("index_total",sum(n_index_years));

for (int i=1; i<=n_selpars; i++)
{
        if (i <= 9)  // note have to deal with one digit and two digit numbers
             separately
        {
                sprintf(onednm, "%d", i);
                twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
                sprintf(twodnm, "%d", i);
        }
        else
        {
                twodnm = "00";
        }
        ichar = "selpar" + twodnm;
        adstring rmse_n_selpar = ichar;
        wrt_r_item(rmse_n_selpar,selpars_rmse_n(i));
}
wrt_r_item("selpars_total",sum(selpars_rmse_n));
    for (int i=1; i<=n_indices; i++)
    {
        if (i <= 9)  // note have to deal with one digit and two digit numbers
             separately
        {
                sprintf(onednm, "%d", i);
                twodnm = "0" + onednm;
        }
```

```
            else if (i <=99)
            {
                    sprintf(twodnm, "%d", i);
            }
            else
            {
                    twodnm = "00";
            }
            ichar = "availability" + twodnm;
            adstring rmse_n_avail = ichar;
            wrt_r_item(rmse_n_avail,availability_rmse_n(i));
    }
wrt_r_item("availability_total",sum(availability_rmse_n));
    for (int i=1; i<=n_indices; i++)
    {
            if (i <= 9)  // note have to deal with one digit and two digit numbers
                 separately
            {
                    sprintf(onednm, "%d", i);
                    twodnm = "0" + onednm;
            }
            else if (i <=99)
            {
                    sprintf(twodnm, "%d", i);
            }
            else
            {
                    twodnm = "00";
            }
            ichar = "availability_AR1" + twodnm;
            adstring rmse_n_avail_AR1 = ichar;
            wrt_r_item(rmse_n_avail_AR1,availability_AR1_rmse_n(i));
    }
wrt_r_item("availability_AR1_total",sum(availability_AR1_rmse_n));
    for (int i=1; i<=n_indices; i++)
    {
            if (i <= 9)  // note have to deal with one digit and two digit numbers
                 separately
            {
                    sprintf(onednm, "%d", i);
                    twodnm = "0" + onednm;
            }
            else if (i <=99)
            {
                    sprintf(twodnm, "%d", i);
            }
            else
            {
                    twodnm = "00";
            }
            ichar = "efficiency" + twodnm;
            adstring rmse_n_eff = ichar;
```

```
            wrt_r_item(rmse_n_eff,efficiency_rmse_n(i));
        }
    wrt_r_item("efficiency_total",sum(efficiency_rmse_n));

    wrt_r_item("N_year1",n_ages);
    wrt_r_item("recruit_devs",n_years);
    wrt_r_item("SR_steepness",1);
    wrt_r_item("SR_scalar",1);

    if (n_fleets>1)
    {
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            adstring rmse_n_Fmult_year1_fleet = adstring("Fmult_year1_") + ichar;
            wrt_r_item(rmse_n_Fmult_year1_fleet,Fmult_year1_rmse_n(i));
        }
    }
    wrt_r_item("Fmult_year1_total",sum(Fmult_year1_rmse_n));
    if (n_fleets>1)
    {
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            adstring rmse_n_Fmult_devs_fleet = adstring("Fmult_devs_") + ichar;
            wrt_r_item(rmse_n_Fmult_devs_fleet,Fmult_devs_rmse_n(i));
        }
    }
    wrt_r_item("Fmult_devs_total",sum(Fmult_devs_rmse_n));
close_r_info_list();

// RMSE section
open_r_info_list("RMSE", false);
    if (n_fleets>1)
    {
        for (int i=1; i<=n_fleets; i++)
        {
            if (n_fleets < 10) sprintf(onenum, "%d", i);
            else onenum="0";
            ichar = "fleet" + onenum;
            adstring rmse_catch_fleet = adstring("catch_") + ichar;
            wrt_r_item(rmse_catch_fleet,catch_rmse(i));
        }
    }
    wrt_r_item("catch_tot",catch_tot_rmse);

    if (n_fleets>1)
    {
```

```
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring rmse_discard_fleet = adstring("discard_") + ichar;
        wrt_r_item(rmse_discard_fleet,discard_rmse(i));
    }
}
wrt_r_item("discard_tot",discard_tot_rmse);

if (n_indices>1)
{
        for (int i=1; i<=n_indices; i++)
        {
                if (i <= 9)  // note have to deal with one digit and two digit
                    numbers separately
                {
                        sprintf(onednm, "%d", i);
                        twodnm = "0" + onednm;
                }
                else if (i <=99)
                {
                        sprintf(twodnm, "%d", i);
                }
                else
                {
                        twodnm = "00";
                }
                ichar = "index" + twodnm;
        adstring rmse_ind = ichar;
        wrt_r_item(rmse_ind,index_rmse(i));
    }
}
wrt_r_item("index_total",index_tot_rmse);

for (int i=1; i<=n_selpars; i++)
{
        if (i <= 9)  // note have to deal with one digit and two digit numbers
            separately
        {
                sprintf(onednm, "%d", i);
                twodnm = "0" + onednm;
        }
        else if (i <=99)
        {
                sprintf(twodnm, "%d", i);
        }
        else
        {
                twodnm = "00";
        }
```

```
            ichar = "selpar" + twodnm;
            adstring rmse_selpar = ichar;
            wrt_r_item(rmse_selpar,selpars_rmse(i));
}
wrt_r_item("selpars_total",selpars_tot_rmse);

wrt_r_item("N_year1",N_year1_rmse);

if (n_fleets>1)
{
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring rmse_Fmult_year1 = adstring("Fmult_year1_") + ichar;
        wrt_r_item(rmse_Fmult_year1,Fmult_year1_rmse(i));
    }
}
wrt_r_item("Fmult_year1_tot",Fmult_year1_tot_rmse);

if (n_fleets>1)
{
    for (int i=1; i<=n_fleets; i++)
    {
        if (n_fleets < 10) sprintf(onenum, "%d", i);
        else onenum="0";
        ichar = "fleet" + onenum;
        adstring rmse_Fmult_devs = adstring("Fmult_devs_") + ichar;
        wrt_r_item(rmse_Fmult_devs,Fmult_devs_rmse(i));
    }
}
wrt_r_item("Fmult_devs_total",Fmult_devs_tot_rmse);

    for (int i=1; i<=n_indices; i++)
    {
            if (i <= 9)  // note have to deal with one digit and two digit numbers
                separately
            {
                    sprintf(onednm, "%d", i);
                    twodnm = "0" + onednm;
            }
            else if (i <=99)
            {
                    sprintf(twodnm, "%d", i);
            }
            else
            {
                    twodnm = "00";
            }
            ichar = "availability" + twodnm;
            adstring rmse_avail = ichar;
```

```
            wrt_r_item(rmse_avail,availability_rmse(i));
        }
    wrt_r_item("availability_total",availability_tot_rmse);
        for (int i=1; i<=n_indices; i++)
        {
            if (i <= 9)  // note have to deal with one digit and two digit numbers
                separately
            {
                sprintf(onednm, "%d", i);
                twodnm = "0" + onednm;
            }
            else if (i <=99)
            {
                sprintf(twodnm, "%d", i);
            }
            else
            {
                twodnm = "00";
            }
            ichar = "availability_AR1" + twodnm;
            adstring rmse_avail_AR1 = ichar;
            wrt_r_item(rmse_avail_AR1,availability_AR1_rmse(i));
        }
    wrt_r_item("availability_AR1_total",availability_AR1_tot_rmse);
        for (int i=1; i<=n_indices; i++)
        {
            if (i <= 9)  // note have to deal with one digit and two digit numbers
                separately
            {
                sprintf(onednm, "%d", i);
                twodnm = "0" + onednm;
            }
            else if (i <=99)
            {
                sprintf(twodnm, "%d", i);
            }
            else
            {
                twodnm = "00";
            }
            ichar = "efficiency" + twodnm;
            adstring rmse_eff = ichar;
            wrt_r_item(rmse_eff,efficiency_rmse(i));
        }
    wrt_r_item("efficiency_total",efficiency_tot_rmse);

    wrt_r_item("recruit_devs",recruit_devs_rmse);
    wrt_r_item("SR_steepness",steepness_rmse);
    wrt_r_item("SR_scalar",SR_scalar_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();

open_r_matrix("rel_efficiency_ini");
    wrt_r_matrix(rel_efficiency_ini, 2, 2);
    wrt_r_namevector(1, n_rel_efficiency_penalties);
    wrt_r_namevector(1, n_lengths);
close_r_matrix();
open_r_matrix("rel_efficiency");
    wrt_r_matrix(rel_efficiency, 2, 2);
    wrt_r_namevector(1, n_rel_efficiency_penalties);
    wrt_r_namevector(1, n_lengths);
close_r_matrix();

open_r_list("rel_efficiency_coef_est");
for (int i=1; i<=n_rel_efficiency_penalties;i++)
{
    if (i <= 9)  // note have to deal with one digit and two digit numbers separately
    {
        sprintf(onednm, "%d", i);
        twodnm = "0" + onednm;
    }
    else if (i <=99)
    {
        sprintf(twodnm, "%d", i);
    }
    else
    {
        twodnm = "00";
    }
    adstring rel_eff_coef = adstring("rel_eff_") + twodnm;
        wrt_r_complete_vector(rel_eff_coef,rel_efficiency_coef(i));
}
close_r_list();
open_r_list("rel_efficiency_X");
for (int i=1; i<=n_rel_efficiency_penalties;i++)
{
    if (i <= 9)  // note have to deal with one digit and two digit numbers separately
    {
        sprintf(onednm, "%d", i);
        twodnm = "0" + onednm;
    }
    else if (i <=99)
    {
        sprintf(twodnm, "%d", i);
    }
    else
    {
        twodnm = "00";
    }
```

```
    adstring rel_eff_X = adstring("rel_eff_") + twodnm;
        open_r_matrix(rel_eff_X);
                wrt_r_matrix(rel_efficiency_X(i), 2, 2);
                wrt_r_namevector(1, n_lengths);
                wrt_r_namevector(1, n_rel_efficiency_coef(i));
        close_r_matrix();
  }
  close_r_list();

// close file
close_r_file();
```

Procedures for Issuing Manuscripts in the *Northeast Fisheries Science Center Reference Document* (CRD) Series

Clearance

All manuscripts submitted for issuance as CRDs must have cleared the NEFSC's manuscript/abstract/webpage review process. If any author is not a federal employee, he/she will be required to sign an "NEFSC Release-of-Copyright Form." If your manuscript includes material from another work which has been copyrighted, then you will need to work with the NEFSC's Editorial Office to arrange for permission to use that material by securing release signatures on the "NEFSC Use-of-Copyrighted-Work Permission Form."

For more information, NEFSC authors should see the NEFSC's online publication policy manual, "Manuscript/abstract/webpage preparation, review, and dissemination: NEFSC author's guide to policy, process, and procedure," located in the Publications/Manuscript Review section of the NEFSC intranet page.

Organization

Manuscripts must have an abstract and table of contents, and (if applicable) lists of figures and tables. As much as possible, use traditional scientific manuscript organization for sections: "Introduction," "Study Area" and/or "Experimental Apparatus," "Methods," "Results," "Discussion," "Conclusions," "Acknowledgments," and "References Cited."

Style

The CRD series is obligated to conform with the style contained in the current edition of the United States Government Printing Office Style Manual. That style manual is silent on many aspects of scientific manuscripts. The CRD series relies more on the CSE Style Manual. Manuscripts should be prepared to conform with these style manuals.

The CRD series uses the American Fisheries Society's guides to names of fishes, mollusks, and decapod crustaceans, the Society for Marine Mammalogy's guide to names of marine mammals, the Biosciences Information Service's guide to serial title abbreviations, and the ISO's (International Standardization Organization) guide to statistical terms.

For in-text citation, use the name-date system. A special effort should be made to ensure that all necessary bibliographic information is included in the list of cited works. Personal communications must include date, full name, and full mailing address of the contact.

Preparation

Once your document has cleared the review process, the Editorial Office will contact you with publication needs – for example, revised text (if necessary) and separate digital figures and tables if they are embedded in the document. Materials may be submitted to the Editorial Office as files on zip disks or CDs, email attachments, or intranet downloads. Text files should be in Microsoft Word, tables may be in Word or Excel, and graphics files may be in a variety of formats (JPG, GIF, Excel, PowerPoint, etc.).

Production and Distribution

The Editorial Office will perform a copy-edit of the document and may request further revisions. The Editorial Office will develop the inside and outside front covers, the inside and outside back covers, and the title and bibliographic control pages of the document.

Once both the PDF (print) and Web versions of the CRD are ready, the Editorial Office will contact you to review both versions and submit corrections or changes before the document is posted online.

A number of organizations and individuals in the Northeast Region will be notified by e-mail of the availability of the document online.

Research Communications Branch
Northeast Fisheries Science Center
National Marine Fisheries Service, NOAA
166 Water St.
Woods Hole, MA 02543-1026

MEDIA
MAIL

Publications and Reports of the Northeast Fisheries Science Center

The mission of NOAA's National Marine Fisheries Service (NMFS) is "stewardship of living marine resources for the benefit of the nation through their science-based conservation and management and promotion of the health of their environment." As the research arm of the NMFS's Northeast Region, the Northeast Fisheries Science Center (NEFSC) supports the NMFS mission by "conducting ecosystem-based research and assessments of living marine resources, with a focus on the Northeast Shelf, to promote the recovery and long-term sustainability of these resources and to generate social and economic opportunities and benefits from their use." Results of NEFSC research are largely reported in primary scientific media (*e.g.*, anonymously-peer-reviewed scientific journals). However, to assist itself in providing data, information, and advice to its constituents, the NEFSC occasionally releases its results in its own media. Currently, there are three such media:

NOAA Technical Memorandum NMFS-NE -- This series is issued irregularly. The series typically includes: data reports of long-term field or lab studies of important species or habitats; synthesis reports for important species or habitats; annual reports of overall assessment or monitoring programs; manuals describing program-wide surveying or experimental techniques; literature surveys of important species or habitat topics; proceedings and collected papers of scientific meetings; and indexed and/or annotated bibliographies. All issues receive internal scientific review and most issues receive technical and copy editing.

Northeast Fisheries Science Center Reference Document -- This series is issued irregularly. The series typically includes: data reports on field and lab studies; progress reports on experiments, monitoring, and assessments; background papers for, collected abstracts of, and/or summary reports of scientific meetings; and simple bibliographies. Issues receive internal scientific review and most issues receive copy editing.

Resource Survey Report (formerly *Fishermen's Report*) -- This information report is a regularly-issued, quick-turnaround report on the distribution and relative abundance of selected living marine resources as derived from each of the NEFSC's periodic research vessel surveys of the Northeast's continental shelf. This report undergoes internal review, but receives no technical or copy editing.

TO OBTAIN A COPY of a *NOAA Technical Memorandum NMFS-NE* or a *Northeast Fisheries Science Center Reference Document*, either contact the NEFSC Editorial Office (166 Water St., Woods Hole, MA 02543-1026; 508-495-2350) or consult the NEFSC webpage on "Reports and Publications" (http://www.nefsc.noaa.gov/nefsc/publications/). To access *Resource Survey Report*, consult the Ecosystem Surveys Branch webpage (http://www.nefsc.noaa.gov/femad/ecosurvey/mainpage/).