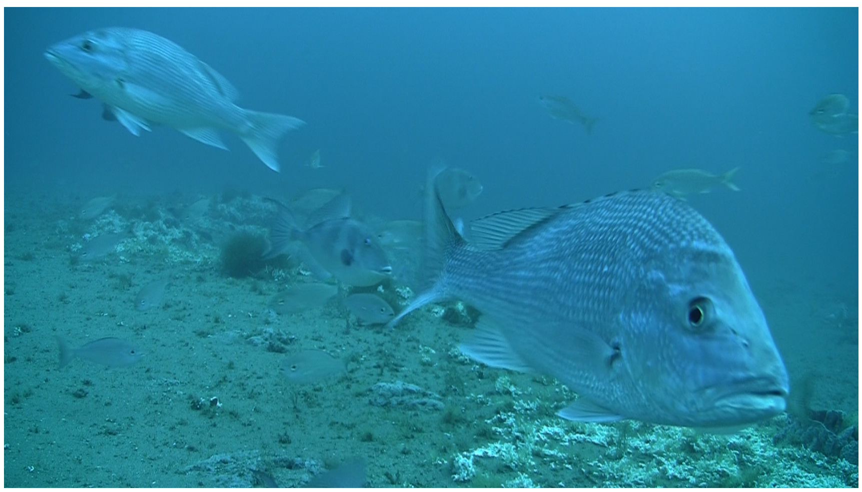
NOAA Technical Memorandum NMFS–SEFSC–671

doi:10.7289/V57M05W6

Technical documentation of the Beaufort Assessment Model (BAM)



Erik H. Williams
Kyle W. Shertzer

U.S. DEPARTMENT OF COMMERCE
National Oceanic and Atmospheric Administration
National Marine Fisheries Service
Southeast Fisheries Science Center
NOAA Beaufort Laboratory
101 Pivers Island Road
Beaufort, North Carolina 28516

February, 2015

NOAA Technical Memorandum NMFS–SEFSC–671
doi:10.7289/V57M05W6

Technical documentation of the Beaufort Assessment Model (BAM)

Erik H. Williams
Kyle W. Shertzer
Southeast Fisheries Science Center
Beaufort, North Carolina

U. S. DEPARTMENT OF COMMERCE
Penny Pritzker, Secretary

NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
Dr. Kathryn D. Sullivan, Undersecretary for Oceans and Atmosphere

NATIONAL MARINE FISHERIES SERVICE
Eileen Sobeck, Assistant Administrator for Fisheries

February, 2015

NOTICE

This report should be cited as follows:

Williams, E. H., and K. W. Shertzer. 2015. Technical documentation of the Beaufort Assessment Model (BAM). U.S. Department of Commerce, NOAA Technical Memorandum NMFS–SEFSC–671. 43 p. doi:10.7289/V57M05W6

Copies may be obtained from:

National Technical Information Center
5825 Port Royal Road
Springfield, VA 22161
(800) 553-6842 or
(703) 605-6000
http://www.ntis.gov/numbers.htm

or by contacting either author
Erik.Williams@noaa.gov or Kyle.Shertzer@noaa.gov

PDF version available at http://www.sefsc.noaa.gov/

Contents

This page intentionally left blank

1 Overview

The Beaufort Assessment Model (BAM) applies a statistical catch-age formulation implemented with the AD Model Builder software (Fournier et al. 2012) and fitted to multiple data sources simultaneously in a single integrated analysis (Maunder and Punt 2013). In essence, the model simulates a population forward in time while including fishing and biological processes (Quinn and Deriso 1999; Shertzer et al. 2014). Quantities to be estimated are systematically varied until characteristics of the simulated population match available data on the real population. Its basic structure is similar to that of other packages such as Stock Synthesis (Methot 2012; Methot and Wetzel 2013) and Age Structured Assessment Program (Legault 2008).

Simulation testing has shown that the BAM can recover estimated parameters accurately. Furthermore, the code and general model structure have been implemented by multiple analysts and have been through numerous independent reviews. Versions of BAM have been applied in peer-reviewed publications [e.g., Conn et al. (2010)] and in stock assessments of Atlantic menhaden (*Brevoortia tyrannus*), Gulf menhaden (*Brevoortia patronus*), Spanish mackerel (*Scomberomorus maculatus*), and numerous reef fishes off the southeast U.S. coast, such as black sea bass (*Centropristis striata*), blueline tilefish (*Caulolatilus microps*), gag (*Mycteroperca microlepis*), greater amberjack (*Seriola dumerili*), red grouper (*Epinephelus morio*), red porgy (*Pagrus pagrus*), red snapper (*Lutjanus campechanus*), snowy grouper (*Hyporthodus niveatus* or *Epinephelus niveatus*), tilefish (*Lopholatilus chamaeleonticeps*), and vermilion snapper (*Rhomboplites aurorubens*). Assessment reports are available at http://www.sefsc.noaa.gov/sedar.

2 Model description

BAM is fundamentally an age-structured population model with birth and death processes. New biomass is acquired through growth and recruitment, while abundance of existing cohorts experiences exponential decay from fishing and natural mortality. The population is assumed closed to immigration and emigration. The model follows an annual time step for n years, $y_1, ..., y_n$, and it includes A age classes $1 - A^+$, where the oldest age class A^+ allows for the accumulation of fish (i.e., plus group). The youngest age class (recruits) is typically age-1 fish produced by the previous year's spawners, but it could instead be age-0 fish produced by the current year's spawners (and consequently with A+1 age classes). Subsequent descriptions assume age-1 is the youngest age class.

Model notation and details are described below and in Table 1, and the basic flow of operations is illustrated in Fig. 1. Although some features of the source code are generalized, others are customized to each stock assessment. Thus, application of BAM requires some programming of the AD Model Builder template file (i.e., *filename*.tpl), as well as configuration of the data input file (i.e., *filename*.dat). This has its drawbacks, notably that application of BAM requires a working knowledge of AD Model Builder and user effort to code the tpl file. It also has its benefits, primarily that the model configuration is extremely flexible, which allows maximum customization for any particular stock assessment and relatively quick modification if needed. This latter ability can be quite useful during stock assessment workshops. In addition, BAM is not static but continues to evolve as the field of stock assessment advances. Thus, because of BAM's flexibility and continued development, the description below is intended as a general documentation of model structure, not an exhaustive account of all possible features. An example application using gag is provided in the Appendix.

2.1 Initialization

BAM has several options to compute initial abundance at age, i.e., abundance in the first modeled year. In all cases, the equilibrium age structure is computed based on natural and initial fishing mortality (F_{init}), where F_{init} is typically defined in one of three ways: 1) input as a fixed value, 2) assumed equal to the average F from the first few years (usually three) of the assessment, and 3) estimated, either freely or with a prior, as its own parameter or as a proportion of the average F from the first few years of the assessment. In some assessments, the equilibrium age structure is used to initialize the population. However, other assessments attempt to estimate the initial nonequilibrium age structure, if composition data are available to inform these estimates. If so, estimation follows a two-part procedure, where first the equilibrium age structure is computed as described above, and second, lognormal deviations ($\hat{\sigma}_a^{init}$) around the equilibrium are estimated for each age two and older. The deviations are penalized by the squared Euclidean norm function, i.e., sum of squares. Consequently, the initial abundance of each age can vary from equilibrium if suggested by early composition data, but remain estimable if data are uninformative. The initial spawning stock, computed from the initial abundance of ages 2^+, is used to generate the number of recruits (age-1 fish) in the first assessment year, using methods similar to those in subsequent years (described below).

2.2 Growth of individuals

Mean length (l_a, in units of mm) at age of the population is modeled with the von Bertalanffy equation,

$$l_a = L_\infty(1 - \exp[-K(a - t_0 + \tau)]) \tag{1}$$

where L_∞, K, and t_0 are parameters, and τ is a fixed value to represent a fraction of the year (typically $\tau = 0.5$). In some assessments, these parameters are estimated within the assessment model, often informed by prior distributions. In other assessments, the parameters are estimated from data before the assessment and treated by BAM as input. Variation in length at age is assumed to be normally distributed, with a CV or standard deviation that is typically estimated and assumed constant across ages, but can also be configured to vary with age.

Weight at age (w_a, in kg of whole weight, WW) is treated as input or else modeled as a function of mean length. If modeled, the functional form is specified by the user, commonly as a power function, $w_a = \theta_1 l_a^{\theta_2}$, where θ_1 and θ_2 are parameters. These parameters are typically estimated from data before the assessment and treated by BAM as input. Once whole weight at age in kg is computed, various conversions may be applied as needed, for example from kilograms to metric tons or to pounds, or from whole weight to gutted weight (GW).

In some cases, fishing fleets might target fish of different sizes than those in the population at large. If so, length at age would differ from l_a in Equation 1 (Schueller et al. 2014). The BAM accommodates this feature by allowing for fleet-specific growth curves, which would translate into fleet-specific weights at age.

2.3 Natural mortality rate

The natural mortality rate (M) is typically treated as input, but in some cases can be estimated. The form of M as a function of age is defined by the user. It could be specified as constant (i.e., age independent), but more commonly M is assumed to decrease with age or size. For assessments in the southeast U.S., age-specific M has typically been based on studies by Lorenzen (1996) or, more recently, Charnov et al. (2013). The Lorenzen (1996) approach inversely relates the natural mortality at age to mean weight at age W_a by the power function $M_a{=}\alpha W_a^\beta$, where α is a scale parameter and β is a shape parameter. Lorenzen (1996) provided point estimates of $\hat{\alpha} = 3.69$ and $\hat{\beta} = -0.305$ for oceanic fishes. Similarly, the Charnov et al. (2013) approach inversely relates the natural mortality at age to somatic growth, $M_a = K(l_a/L_\infty)^{-1.5}$. Whichever approach is taken, the age-dependent estimates of M_a are often rescaled for consistency with cumulative survival to maximum age (Hoenig 1983; Hewitt and Hoenig 2005; Then et al. 2014). In some assessments, M_a is assumed to vary across years.

2.4 Maturity and sex ratio

Maturity at age of females is treated by BAM as input, either as a vector (i.e., if time invariant) or an n$\times$A matrix (i.e., for year- and age-specific values). Sex ratio at age is treated in the same manner. Many stocks in the southeast U.S. are protogynous hermaphrodites, and so for those stocks, maturity at age of males is also modeled.

2.5 Reproductive output

BAM is flexible in how it computes reproductive output, often referred to as spawning stock (S). For gonochoristic species, reproductive output is typically computed as total fecundity (when that information is available), or else as mature female biomass (in units of mt). For protogynous species, reproductive output is typically modeled as total mature biomass (mt; males and females), following the advice of Brooks et al. (2008). Computations discount abundance to the time of peak spawning, thus accounting for a partial year of natural and fishing mortality.

2.6 Recruitment

Expected annual recruitment ($\bar{R}_y$) is computed from either the Beverton–Holt or Ricker spawner-recruit model. In BAM, the Beverton–Holt formulation is,

$$\bar{R}_{y+1} = \frac{0.8 R_0 h S_y}{0.2 R_0 \phi_0 (1-h) + S_y (h - 0.2)} \tag{2}$$

where R_0 is virgin recruitment, h is steepness, and ϕ_0 is the unfished spawners per recruit. The analogous Ricker formulation is,

$$\bar{R}_{y+1} = \frac{S_y}{\phi_0} \exp\left(h\left(1 - \frac{S_y}{R_0 \phi_0}\right)\right) \tag{3}$$

In years when data are considered to be informative on recruitment, multiplicative deviations are included assuming a lognormal distribution,

$$N_{1,y} = \bar{R}_y \exp(r_y) \tag{4}$$

Here r_y is assumed to follow a normal distribution with standard deviation σ_R.

In arithmetic space, expected recruitment is higher than that estimated directly from the spawner-recruit curve because of lognormal deviation in recruitment residuals. Thus, a bias correction is applied when computing equilibrium recruitment. The bias correction (ς) is computed from the variance (σ_R^2) of recruitment deviations in log space: $\varsigma = \exp(\sigma_R^2/2)$. Then, under Beverton–Holt, the expected equilibrium recruitment (R^{eq}) associated with any F is,

$$R^{eq} = \frac{R_0 \left[\varsigma 4 h \phi_F - (1-h)\phi_0\right]}{(5h-1)\phi_F} \tag{5}$$

and under Ricker,

$$R^{eq} = \frac{R_0}{\Phi_F}\left(1 + \frac{\log(\varsigma \Phi_F)}{h}\right) \tag{6}$$

where ϕ_F is spawners per recruit given F, and $\Phi_F = \phi_F/\phi_0$ is the spawning potential ratio.

In years when data are considered to be uninformative on recruitment, multiplicative deviations would not generally be estimated. Instead, $N_{1,y+1} = R_{eq}$. Computation of R_{eq}, along with the mortality schedule, implies an equilibrium age structure, which would apply to calculations of the initialization (described above) as well as calculations of biological reference points (described below).

2.7 Selectivities

In BAM, selectivity is modeled as a function of age. It may also vary over time, but to simplify the description below, the year subscript is not included. Selectivity at age ($s_{(f,d,u),a}$) ranges on the interval $[0,1]$ and can be modeled for three different types of data: landings (denoted by subscript f), discards (subscript d), and indices (subscript u). In any case, it may be estimated by using a free parameter ($x_{(f,d,u),a}$) for each age, or by using a parametric function. The free-parameter approach estimates selectivity in logit space, such that $s_{(f,d,u),a} = \frac{1}{1+\exp(-x_{(f,d,u),a})}$.

The parametric approach imposes theoretical structure on selectivity, and it can reduce the number of estimated parameters, particularly when the model includes many ages. Parametric models of selectivity in BAM impose one of two forms: flat-topped or dome-shaped. Flat-topped selectivity describes a pattern of fishing rates that increase across the younger ages and then saturate at a value of 1.0 for all older ages. In BAM, it is estimated using a two-parameter (x_1, x_2) logistic model:

$$s_{(f,d,u),a} = \frac{1}{1+\exp(-x_1(a-x_2))} \tag{7}$$

where x_1 controls the rate of increase, and x_2 is the age at 50% selection.

Dome-shaped selectivity describes a pattern of fishing rates that increase across the younger ages, peak at a value of 1.0, and then decrease across older ages. In BAM, four options are available for dome-shaped selectivity: double-logistic, joint-logistic, logistic-exponential, and double-Gaussian. The double-logistic model (four parameters) combines two logistic curves, one to describe the increasing portion and one to describe the decreasing portion:

$$s_{(f,d,u),a} = \left(\frac{1}{1+\exp(-x_1(a-x_2))}\right)\left(1-\frac{1}{1+\exp(-x_3(a-x_4))}\right) \tag{8}$$

The double-logistic model typically requires re-scaling to ensure that it peaks at one. As such, parameters may not be identifiable without the use of priors.

The joint-logistic model (five parameters) does not require re-scaling, but does require specifying *a priori* the age at full selection (a_f). In addition, this model allows the descending limb to saturate at a value (x_5) less than 1.0:

$$s_{(f,d,u),a} = \begin{cases} \frac{1}{1+\exp(-x_1(a-x_2))} & : \quad a < a_f \\ 1.0 & : \quad a = a_f \\ 1-\frac{1-x_5}{1+\exp(-x_3(a-x_4))} & : \quad a > a_f \end{cases} \tag{9}$$

Similarly, the logistic-exponential model (three parameters) requires specifying *a priori* the age at full selection. It describes the ascending limb with a logistic curve for ages prior to full selection (two parameters x_1, x_2), and the descending limb with a negative exponential curve (one parameter, x_3):

$$s_{(f,d,u),a} = \begin{cases} \frac{1}{1+\exp[-x_1(a-x_2)]} & : \quad a < a_f \\ 1.0 & : \quad a = a_f \\ \exp\left(-\left(\frac{(a-a_f)}{x_3}\right)^2\right) & : \quad a > a_f \end{cases} \tag{10}$$

The double-gaussian model (six parameters) is the most flexible option in BAM, but does require re-scaling. Parameters are loosely defined as follows: x_1' is the ascending inflection location, x_2' controls the width of the plateau, x_3' controls the ascent width, x_4' controls the descent width, x_5' controls the function value at the youngest age, and x_6' controls the function value at the oldest age. These parameters are transformed as follow:

$$
\begin{aligned}
x_1 &= x_1' \\
x_2 &= x_1' + 1.0 + \tfrac{(0.99A - x_1' - 1.0)}{1+\exp(-x_2')} \\
x_3 &= \exp(x_3') \\
x_4 &= \exp(x_4') \\
x_5 &= \tfrac{1.0}{1.0+\exp(-x_5')} \\
x_6 &= \tfrac{1.0}{1.0+\exp(-x_6')}
\end{aligned} \tag{11}
$$

Given the transformed parameters, several intermediate functions are defined:

$$
\begin{aligned}
f_1(a) &= \exp(\tfrac{-(a-x_1)^2}{x_3}) \\
f_2(a) &= x_5 + (1.0 + x_5)\tfrac{(f_1(a) - f_1(a_1))}{(1.0 - f_1(a_1))} \\
f_3(a) &= \exp\left(\tfrac{-(a-x_2)^2}{x_4}\right) \\
f_4(a) &= 1.0 + (x_6 - 1)\tfrac{(f_3(a) - 1.0)}{(f_3(A) - 1.0)} \\
f_5(a) &= \tfrac{1.0}{1.0+\exp\left(\frac{-20(a-x_1)}{(1.0+|a-x_1|)}\right)} \\
f_6(a) &= \tfrac{1.0}{1.0+\exp\left(\frac{-20(a-x_2)}{(1.0+|a-x_2|)}\right)}
\end{aligned} \tag{12}
$$

Here, a_1 is the youngest age (typically 0 or 1), and A is the oldest age. Then, using the intermediate functions, selectivity is computed as:

$$
s_{(f,d,u),a} = f_2(a)\,(1.0 + f_5(a)) + f_5(a)\,[1.0 - f_6(a) + f_4(a) f_6(a)] \tag{13}
$$

Whichever approach is used, selectivity functions may vary over time, and thus in practice have the additional subscript of year, $s_{(f,d,u),a,y}$. The variation could be annual or across blocks of years, for example, where blocks represent periods of consistent regulations. Age and/or length composition data are critical to estimating selectivity, but even with those data, parameters will not always be identifiable without the use of priors.

2.8 Fishing

For each fleet being modeled, the BAM estimates a separate full fishing mortality rate for each year of the time series ($F_{(f,d),y}$), with landings and discards treated as distinct fleets. Age-specific rates are computed as the product of full F and selectivity at age (i.e., $F_{(f,d),a,y} = s_{(f,d),a,y} F_{(f,d),y}$). Then, the across-fleet annual F_y is represented by apical F, computed as the maximum of F at age summed across fleets,

$$
F_{a,y} = \sum_{(f,d)} F_{(f,d),a,y} \tag{14a}
$$

$$
F_y = \max_a (F_{a,y}) \tag{14b}
$$

2.9 Landings and discard mortality

Landings at age in numbers for each fleet are predicted using the Baranov catch equation (Baranov 1918),

$$
l'_{f,a,y} = \frac{F_{f,a,y}}{Z_{a,y}} N_{a,y}[1 - \exp(-Z_{a,y})] \tag{15}
$$

where $Z_{a,y} = M_a + F_{a,y}$ is total mortality at age and $N_{a,y}$ is annual abundance at age. Then, landings at age in weight are calculated as,

$$l''_{f,a,y} = \mathcal{C} w^L_{f,a,y} l'_{f,a,y} \tag{16}$$

where $w^L_{f,a,y}$ is fleet-specific weight at age, which may differ from that of the population at large. The constant $\mathcal{C}$ converts units from those of w^L to those of observed removals (e.g., from mt to 1000 lb). Total landings in numbers and weight are computed as,

$$L'_{f,y} = \sum_a l'_{f,a,y} \tag{17a}$$

$$L''_{f,y} = \sum_a l''_{f,a,y} \tag{17b}$$

Similarly, dead discards at age in numbers from each discard fleet are computed,

$$d'_{d,a,y} = \frac{F_{d,a,y}}{Z_{a,y}} N_{a,y}[1 - \exp(-Z_{a,y})] \tag{18}$$

as are those in weight,

$$d''_{d,a,y} = \mathcal{C} w^D_{d,a,y} d'_{d,a,y} \tag{19}$$

Total discards in numbers and weight are computed as,

$$D'_{f,y} = \sum_a d'_{d,a,y} \tag{20a}$$

$$D''_{f,y} = \sum_a d''_{d,a,y} \tag{20b}$$

2.10 Stock dynamics

Abundance of recruits ($N_{1,y}$) is described above in the section titled Recruitment. Abundance of each subsequent age at the start of each year is computed assuming exponential decay,

$$N_{a+1,y+1} = N_{a,y} \exp(-Z_{a,y}) \quad \forall a \in (1 \ldots A-1) \tag{21a}$$

$$N_{A,y+1} = N_{A-1,y} \exp(-Z_{A-1,y}) + N_{A,y} \exp(-Z_{A,y}) \tag{21b}$$

In addition, BAM computes abundance later in the year, $N'_{a,y} = N_{a,y} \exp(-t_{\text{index}} Z_{a,y})$, for matching observed indices of abundance. In this calculation, t_{index} represents the fraction of the year over which to apply total mortality, most typically $t_{\text{index}} = 0.5$ for calculating mid-year abundance. Similarly, BAM computes abundance at the time of peak spawning, $N''_{a,y} = N_{a,y} \exp(-t_{\text{spawn}} Z_{a,y})$, to derive spawning stock. Here, t_{spawn} represents the fraction of the year when peak spawning occurs (e.g., $t_{\text{spawn}} = 0.25$ reflects peak spawning at the end of March).

2.11 Indices of abundance

Predicted indices ($U_{u,y}$) for each index (u) are computed from numbers at age, scaled to the relevant portion of the age structure by selectivity. A predicted index could additionally be computed in weight, if the observed index is measured in weight.

$$U_{u,y} = \begin{cases} \hat{q}_{u,y} \sum_a s_{u,a} N'_{a,y} & : \text{ if in numbers} \\ \hat{q}_{u,y} \sum_a s_{u,a} w_a N'_{a,y} & : \text{ if in weight} \end{cases} \tag{22}$$

Catchability ($q_{u,y}$) scales indices of abundance to the estimated population at large.

2.12 Catchability

Annual catchability associated with each index can be modeled as constant or variable through time. Constant catchability is often the default assumption, but when available data allow for meaningful estimation, modeling catchability as time-varying may be desirable (SEDAR Procedural Guidance 2009; Wilberg et al. 2010). In BAM, three types of time-varying catchability are included as options: 1) density dependent, 2) linearly increasing, and 3) penalized random walk. The three options operate multiplicatively, and can be applied in any combination.

Density dependence is applied via a function, $f^{density}(B'_y) = (B'_0)^{\widehat{\psi}}(B'_y)^{-\widehat{\psi}}$, where $\widehat{\psi}$ is a parameter to be estimated or fixed, $B'_y = \sum_{a=a'}^{A} B_{a,y}$ is annual biomass above some threshold age a', and B'_0 is unfished biomass for ages a' and older. In practice, a' should be set high enough to reflect the exploitable biomass.

A linearly increasing trend is applied via the function, $f_u^{trend}(y)$, which is set to 1.0 in year one $(y_{u,1})$ of the index, and increases thereafter according to the slope (B_q): $f_u^{trend}(y) = f_u^{trend}(y-1) * (y - y_{u,1})B_q$. Several applications of BAM have applied a slope of 2% per year to account for technological improvements in fishing efficiency. This increasing trend reflects the belief that catchability has generally increased over time as a result of improved technology (SEDAR Procedural Guidance 2009) and as estimated for reef fishes in the Gulf of Mexico (Thorson and Berkson 2010).

A random walk is applied assuming lognormal deviations, $f_u^{rw}(y) = \exp(\epsilon_{u,y})$. The values, $\epsilon_{u,y}$, are penalized for deviation from zero, as described below in §3. The amount of "tension" on the random walk is controlled by an input parameter, σ_u^q. As σ_u^q decreases, variation in the random walk is penalized more heavily.

Any of the time-varying functions not in use can simply be set to a value of 1.0. Then, annual catchability is computed as the product of the scaling constant, $\widehat{q'_u}$, and each of the functions,

$$q_{u,y} = \widehat{q'_u} \times f^{density}(B'_y) \times f_u^{trend}(y) \times f_u^{rw}(y) \quad (23)$$

If time-varying catchability is not modeled, all of the functions are set to 1.0, such that $q_{u,y} = \widehat{q'_u}$.

2.13 Ageing error

The BAM can accommodate ageing error through application of a $\mathcal{B}^\alpha \times \mathcal{B}^\alpha$ matrix $\mathcal{E}$, where $\mathcal{B}^\alpha$ is the number of age classes. In this matrix, the columns sum to one and act to spread true ages across ages that would be observed given ageing error. Predicted age compositions incorporate $\mathcal{E}$ for matching observed age compositions, as described in Table 1. If ageing error is not included, the matrix is set equal to the identity matrix, $\mathcal{E} = I$.

3 Fitting criteria

The objective function minimized by AD Model Builder is a composite of negative likelihoods with some additional penalty terms. Observed landings ($\breve{L}$), discards ($\breve{D}$), and indices ($\breve{U}$) are fit using lognormal likelihoods. Observed age compositions ($\breve{p}^\alpha$) and length compositions ($\breve{p}^\lambda$) are fit using standard or robust multinomial likelihoods (Francis 2011). In addition, the objective function includes various penalties, applied for two reasons: 1) to include prior information on estimated parameters, as might be done in a Bayesian analysis, and 2) to constrain variability within estimated vectors, such as annual recruitment deviations, random walk in catchability, and initial age structure. Although BAM contains common formulations of likelihoods and penalties, the objective function can be customized by the user to include virtually any fitting criteria.

3.1 Data components

Observed landings can be supplied in numbers or in weight for any given fleet. For fitting landings data, BAM uses the corresponding prediction (L), computed such that units of predictions and observations match, i.e., $L = L'$ or $L = L''$. The landings contribution (Λ^L) to the total objective function is

$$\Lambda^L = \sum_f \sum_y \frac{\left[\log\left((L_{f,y} + \epsilon)/(\breve{L}_{f,y} + \epsilon)\right)\right]^2}{2(\sigma^L_{f,y})^2} \tag{24}$$

where $\epsilon = 1e-5$ to prevent the optimization procedure from attempting to compute the log of zero (an undefined value), and where $\sigma^L_{f,y}$ are standard deviations in log space. These standard deviations are computed as $\sigma^L_{f,y} = \sqrt{\log(1 + (CV^L_{f,y}/\omega^L_f)^2)}$, where $CV^L_{f,y}$ are user-supplied coefficients of variation in arithmetic space and ω^L_f are user-supplied weights. Analogous contributions to the total objective function are computed for discards (Λ^D) and indices of abundance (Λ^U).

Composition data are typically fit using a robust formulation of the multinomial likelihood (Francis 2011). In this formulation, predicted age compositions ($p^\alpha_{(f,u),a,y}$) of fleet f or index u are matched to the observed values ($\breve{p}^\alpha_{(f,u),a,y}$), with contribution ($\Lambda^\alpha$) to the total objective function computed as,

$$\Lambda^\alpha = \sum_{f,u} \sum_y 0.5 \log(E') - \log\left[\exp\left(-\frac{(\breve{p}^\alpha_{(f,u),a,y} - p^\alpha_{(f,u),a,y})^2}{2E'/(n^\alpha_{(f,u),y}\omega^\alpha_{(f,u)})}\right) + \epsilon\right] \tag{25}$$

where $E' = \left[(1 - \breve{p}^\alpha_{(f,u),a,y})(\breve{p}^\alpha_{(f,u),a,y}) + \frac{0.1}{\mathcal{B}^\alpha}\right]$, $\mathcal{B}^\alpha$ is the number of age bins, $n^\alpha_{(f,u),y}$ are sample sizes, $\omega^\alpha_{(f,u)}$ are user-supplied weights, and ϵ =1e-5 to avoid log zero. Analogous contributions to the total objective function are computed for length composition data (Λ^λ). The standard formulation of the multinomial likelihood (i.e., not the robust version) is also available as an option.

3.2 Penalty terms

Recruitment deviations are assumed to follow a lognormal distribution, with the option to allow first-order autocorrelation,

$$\Lambda^{R1} = \omega_{R1}\left[\frac{[r_{y'} + (\widehat{\sigma}^2_R/2)]^2}{2\widehat{\sigma}^2_R} + \sum_{y>y'}^{y''} \frac{[(r_y - \widehat{\rho} r_{y-1}) + (\widehat{\sigma}^2_R/2)]^2}{2\widehat{\sigma}^2_R} + n\log(\widehat{\sigma}_R)\right] \tag{26}$$

where r_y are recruitment deviations in log space, n is the number of years, ω_{R1} is a user-supplied weight (may be 1.0), $\widehat{\rho}$ is the autocorrelation term, and $\widehat{\sigma}^2_R$ is the estimated recruitment variance. The years y' and y'' are the first and last years for estimating recruitment deviations, which need not be the first and last years of the full assessment period. BAM includes the option for early recruitment deviations to receive additional constraint through a sum-of-squares penalty, $\Lambda^{R2} = \omega_{R2}\sum_y r^2_y$, applied over years y. This penalty can be turned off by setting $\omega_{R2} = 0$. Similarly, terminal recruitment deviations may receive additional constraint if desired, $\Lambda^{R3} = \omega_{R3}\sum_y r^2_y$, which can be turned off by setting $\omega_{R3} = 0$.

If a nonequilibrium initial age structure is estimated, the deviations ($\hat{\sigma}^{init}_a$) from equilibrium are assumed to be lognormally distributed. They are penalized for deviating from zero using a sum-of-squares term, $\Lambda^{init} = \omega_{init}\sum_a(\sigma^{init}_a)^2$. These deviations do not include the youngest age, because it is already accounted for by the first year of recruitment deviations.

Similarly, if a random walk is applied to the catchability of index u, a sum-of-squares penalty is applied, $\Lambda^q = \sum_u \sum_y (\epsilon^2_{u,y})/(2\sigma^q_u)$. Here, σ^q_u controls the amount of tension on each random walk.

BAM includes an option to penalize apical F_y if it exceeds a threshold value ϕ, which is set by the user. The penalty is zero if $F_y \leq \phi$ and otherwise grows exponentially,

$$\Lambda^F = \omega_F \sum_y (\exp(F_y - \phi) - 1) \quad \forall\ F_y > \phi \tag{27}$$

This penalty is turned off when the user-defined weight ω_F is set to zero.

For any estimated parameter, a penalty can be applied for deviation from a user-supplied value. These penalties are similar in concept to prior distributions used in Bayesian approaches. Their purpose in BAM is to maintain parameter estimates near reasonable values and to prevent the optimization routine from drifting into parameter space with negligible gradient in the likelihood. This prior information on any given parameter is implemented as a negative log-likelihood term using one of three standard distributional forms that the user must specify: normal, lognormal, or beta. In addition, the user must specify the mean and variance of each distribution. The sum of all such penalty terms (i.e., negative log-likelihoods) is labeled Λ^P.

Given the data components and penalty terms, the total objective function value to be minimized is,

$$\Lambda = \Lambda^L + \Lambda^D + \Lambda^U + \Lambda^\alpha + \Lambda^\lambda + \Lambda^{R1} + \Lambda^{R2} + \Lambda^{R3} + \Lambda^{init} + \Lambda^q + \Lambda^F + \Lambda^P \tag{28}$$

4 Biological reference points

Biological reference points (benchmarks) are calculated based on maximum sustainable yield (MSY) estimates from the spawner-recruit model with bias correction (expected values in arithmetic space). These benchmarks include MSY, fishing mortality rate at MSY (F_{MSY}), dead discards at MSY (D_{MSY}), and spawning stock at MSY ($\mathrm{SSB}_{\mathrm{MSY}}$). The point of maximum yield is identified from the spawner-recruit curve and parameters describing growth, natural mortality, maturity, and selectivity. The value of F_{MSY} is the F that maximizes equilibrium landings (i.e., MSY). The values of D_{MSY} and $\mathrm{SSB}_{\mathrm{MSY}}$ are those that correspond to F_{MSY}.

In addition to the MSY-related benchmarks, the assessment considered proxies based on per recruit analyses (e.g., $F_{40\%}$). The values of $F_{X\%}$ are defined as those Fs corresponding to X% spawning potential ratio, i.e., spawners (population fecundity) per recruit relative to that at the unfished level. These quantities may serve as proxies for F_{MSY} if the spawner-recruit relationship cannot be estimated reliably. Mace (1994) recommended $F_{40\%}$ as a proxy; however, later studies have found that a fishing rate of $F_{40\%}$ is too high across many life-history strategies (Williams and Shertzer 2003; Brooks et al. 2009) and can lead to undesirably low levels of biomass and recruitment (Clark 2002).

The MSY-based benchmarks and proxies are conditional on the estimated selectivity functions. For computation of benchmarks, three composite selectivities are computed from the terminal year of the assessment: 1) selectivity associated with landings, 2) selectivity associated with dead discards, and 3) the sum of the previous two, which describes total fishing mortality and has a peak value of one. The composite selectivities are F-weighted average selectivities across fleets, with F from each fleet estimated as the full F averaged over the last X years of the assessment. Typically, $X = 3$ years.

5 Acknowledgments

The BAM has benefited from the analytical scrutiny of Lew Coggins, Paul Conn, Kevin Craig, Mike Prager, Amy Schueller, and Katie Siegfried. The authors are grateful for helpful comments on this memorandum by Kevin Craig, Amy Schueller, and Katie Siegfried.

6 References

References

Baranov, F. I. 1918. On the question of the biological basis of fisheries. Nauchnye Issledovaniya Ikhtiologicheskii Instituta Izvestiya **1**:81–128.

Brooks, E. N., J. E. Powers, and E. Cortes. 2009. Analytical reference points for age-structured models: application to data-poor fisheries. ICES Journal of Marine Science **67**:165–175.

Brooks, E. N., K. W. Shertzer, T. Gedamke, and D. S. Vaughan. 2008. Stock assessment of protogynous fish: evaluating measures of spawning biomass used to estimate biological reference points. Fishery Bulletin **106**:12–23.

Charnov, E. L., H. Gislason, and J. G. Pope. 2013. Evolutionary assembly rules for fish life histories. Fish and Fisheries **14**:213–224.

Clark, W. G. 2002. $F_{35\%}$ revisited ten years later. North American Journal of Fisheries Management **22**:251–257.

Conn, P. B., E. H. Williams, and K. W. Shertzer. 2010. When can we reliably estimate the productivity of fish stocks? Canadian Journal of Fisheries and Aquatic Sciences **67**:511–523.

Fournier, D. A., H. J. Skaug, J. Ancheta, J. Ianelli, A. Magnusson, M. N. Maunder, A. Nielsen, and J. Sibert. 2012. AD Model Builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models. Optimization Methods and Software **27**:233–249.

Francis, R. 2011. Data weighting in statistical fisheries stock assessment models. Canadian Journal of Fisheries and Aquatic Sciences **68**:1124–1138.

Hewitt, D. A., and J. M. Hoenig. 2005. Comparison of two approaches for estimating natural mortality based on longevity. Fishery Bulletin **103**:433–437.

Hoenig, J. M. 1983. Empirical use of longevity data to estimate mortality rates. Fishery Bulletin **81**:898–903.

Legault, C. M. 2008. Technical Documentation for ASAP Version 2.0. NOAA Fisheries Toolbox. URL `http://nft.nefsc.noaa.gov/`.

Lorenzen, K. 1996. The relationship between body weight and natural mortality in juvenile and adult fish: a comparison of natural ecosystems and aquaculture. Journal of Fish Biology **49**:627–642.

Mace, P. M. 1994. Relationships between common biological reference points used as thresholds and targets of fisheries management strategies. Canadian Journal of Fisheries and Aquatic Sciences **51**:110–122.

Maunder, M. N., and A. E. Punt. 2013. A review of integrated analysis in fisheries stock assessment. Fisheries Research **142**:61–74.

Methot, R. D., 2012. User Manual for Stock Synthesis, Model Version 3.24f. NOAA Fisheries, Seattle, WA.

Methot, R. D., and C. R. Wetzel. 2013. Stock synthesis: A biological and statistical framework for fish stock assessment and fishery management. Fisheries Research **142**:86–99.

Quinn, T. J., and R. B. Deriso. 1999. Quantitative Fish Dynamics. Oxford University Press, New York.

Schueller, A. M., E. H. Williams, and R. T. Cheshire. 2014. A proposed, tested, and applied adjustment to account for bias in growth parameter estimates due to selectivity. Fisheries Research **158**:26–39.

SEDAR Procedural Guidance, 2009. SEDAR Procedural Guidance Document 2: Addressing Time-Varying Catchability. North Charleton, SC. Available online http://www.sefsc.noaa.gov/sedar/.

Shertzer, K. W., E. H. Williams, M. H. Prager, and D. S. Vaughan, 2014. Fishery models. *in* Reference Module in Earth Systems and Environmental Sciences. Elsevier, http://dx.doi.org/10.1016/B978-0-12-409548-9.09406-9.

Then, A. Y., J. M. Hoenig, N. G. Hall, and D. A. Hewitt. 2014. Size, growth, temperature and the natural mortality of marine fish. ICES Journal of Marine Science **doi: 10.1093/icesjms/fsu136**.

Thorson, J. T., and J. Berkson. 2010. Multispecies estimation of Bayesian priors for catchability trends and density dependence in the US Gulf of Mexico. Canadian Journal of Fisheries and Aquatic Science **67**:936–954.

Wilberg, M. J., J. T. Thorson, B. C. Linton, and J. Berkson. 2010. Incorporating time-varying catchability into population dynamic stock assessment models. Reviews in Fisheries Science **18**:7–24.

Williams, E. H., and K. W. Shertzer. 2003. Implications of life-history invariants for biological reference points used in fishery management. Canadian Journal of Fisheries and Aquatic Science **60**:710–720.

Table 1. General definitions, input data, population model, and objective-function components of the BAM.

Quantity	Symbol	Description or definition
Labels for Indexing		
Years	y	$y \in \{y_1 \ldots y_n\}$.
Ages	a	$a \in \{a_1 \ldots A\}$, where a_1 is the recruitment age, typically 0 or 1, and A is the oldest age, treated as a plus group. The number of age bins is denoted $\mathcal{B}^{\alpha}$; typically $\mathcal{B}^{\alpha} = A$ or $A+1$.
Length bins	l	$l \in \{1 \ldots \mathcal{B}^{\lambda}\}$, where $\mathcal{B}^{\lambda}$ is the number of length bins.
Length bin boundaries	l'	$l' \in \{l'_1 \ldots l'_{\mathcal{B}^{\lambda}}\}$, with values representing the upper bound of each length bin. Largest length bin is treated as a plus group.
Fleets, landings	f	Various fleets from commercial and/or recreational sectors.
Fleets, discards	d	Various fleets from commercial and/or recreational sectors.
Indices of abundance	u	Fishery dependent and/or fishery independent sources.
Input Data		
Observed length compositions	$\breve{p}^{\lambda}_{(f,d,u),l,y}$	Proportional contribution of length bin l in year y to fleet f, d (landings, discards) or index u.
Observed age compositions	$\breve{p}^{\alpha}_{(f,d,u),a,y}$	Proportional contribution of age class a in year y to fleet f, d or index u.
Length comp. sample sizes	$n^{\lambda}_{(f,d,u),y}$	Effective number of length samples collected in year y from fleet f, d, or index u.
Age comp. sample sizes	$n^{\alpha}_{(f,d,u),y}$	Effective number of age samples collected in year y from fleet f, d or index u.
Observed landings	$\breve{L}_{f,y}$	Reported landings in year y from fleet f.
CVs of landings	$CV^{L}_{f,y}$	Annual CV in arithmetic space.
Observed abundance indices	$\breve{U}_{u,y}$	Relative abundance in year y from index u.
CVs of abundance indices	$CV^{U}_{u,y}$	Annual CV in arithmetic space.
Observed total discards	$\tilde{D}_{d,y}$	Reported total discards (live and dead) in year y from fleet d.
Discard mortality rate	δ_d	Proportion discards by fleet d that die.
Observed discard mortalities	$\breve{D}_{d,y}$	$D_{d,y} = \delta_d \tilde{D}_{d,y}$
CVs of dead discards	$CV^{D}_{d,y}$	Annual CV in arithmetic space.
Population Model		
Mean length at age	l_a	Total length (midyear, in units of mm); $l_a = L_{\infty}(1 - \exp[-K(a - t_0 + \tau)])$ where K, L_{∞}, and t_0 are parameters, and τ is a fixed value representing a fraction of the year (typically $\tau = 0.5$).
CV of l_a	CV^{λ}_a	Coefficient of variation of length at age.
SD of l_a	σ^{λ}_a	Standard deviation of length at age, $\sigma^{\lambda}_a = CV^{\lambda}_a l_a$.

Table 1. (continued)

Quantity	Symbol	Description or definition
Age–length conversion of population	$\psi_{a,l}$	$\psi_{a,l=i} = \int_{x=l'_{i-1}}^{x=l'_i} f(x; l_a, \sigma_a^\lambda)dx$, where f represents the normal density function and x is the variable of integration. The smallest length bin uses zero as the lower bound, and the largest uses infinity (i.e., is a plus group). Matrix ψ is rescaled to sum to one within each age, which may be necessary because of truncating the normal distribution at a lower bound of zero.
Mean length at age of landings and discards	$\ell_{(f,d),a,y}$	BAM contains the option for fleet-specific mean lengths of landings ($\ell_{f,a,y}$) and discards ($\ell_{d,a,y}$), modeled with von Bertalanffy growth as for the population (l_a), but with separately estimated parameters. If this option is not used, $\ell_{(f,d),a,y} = l_a$.
Age–length conversion of landings	$\psi^L_{f,a,l,y}$	Computation is analogous to that of $\psi_{a,l}$, but with mean lengths based on $\ell_{f,a,y}$.
Age–length conversion of discards	$\psi^D_{d,a,l,y}$	Computation is analogous to that of $\psi_{a,l}$, but with mean lengths based on $\ell_{d,a,y}$. In addition, if a regulatory size limit ($l_{\lim}$) is in effect, the distribution of sizes at age may be truncated, such that $\psi^D_{d,a,l>l_{\lim},y} = 0$. If so, annual matrices $\psi^D_{d,a,l,y}$ are rescaled to sum to one within each age.
Individual weight at age of population	w_a	Typically computed from mean length at age, for example using the equation $w_a = \theta_1 l_a^{\theta_2}$, where θ_1 and θ_2 are parameters.
Individual weight at age of landings and discards	$w^{L,D}_{(f,d),a,y}$	Computed from mean length at age, for example by $w^{L,D}_{(f,d),a,y} = \theta_1(\ell^{L,D}_{(f,d),a,y})^{\theta_2}$.
Natural mortality rate	M_a	Natural mortality rate at age, typically assumed constant across years.
Proportion female at age	η_a	A decreasing function for protogynous hermaphrodites; constant for gonochoristic stocks. Typically assumed constant across years.
Proportion male at age	$1-\eta_a$	Complement of proportion female.
Proportion females mature at age	m_a	Typically increases with age.
Proportion males mature at age	m'_a	For protogynous hermaphrodites, all males typically assumed mature.
Batch fecundity at age	E_a	Eggs spawned per batch.
Number annual batches at age	b_a	Number of batches spawned per year.
Annual fecundity at age	$\mathcal{F}_a$	$\mathcal{F}_a = b_a E_a$. If fecundity information is unavailable, spawning biomass is used as a proxy.
Index date	t_{index}	Fraction denoting the proportional time of year for computing abundance at age, as used to predict indices of abundance. Typically, $t_{\text{index}} = 0.5$ for mid-year calculations.
Spawning date	t_{spawn}	Fraction denoting the proportional time of year when spawning occurs. For example, if peak spawning occurs at the end of March, $t_{\text{spawn}} = 0.25$.
Reproductive capacity at age	ξ_a	Multiplier on abundance to define reproductive output (spawning stock), for example: $\xi_a = \begin{cases} \eta_a m_a \mathcal{F}_a & : \text{If based on fecundity} \\ \eta_a m_a w_a & : \text{If based on mature female body weight} \\ [\eta_a m_a + (1-\eta_a) m'_a]\, w_a & : \text{If based on mature body weight of both sexes.} \end{cases}$
Selectivities	$s_{(f,d,u),a,y}$	Modeled as a free parameter at each age or using a parametric approach, as described in the "Selectivities" section of the text.
Fishing mortality rate of landings	$F_{f,a,y}$	$F_{f,a,y} = s_{f,a,y} F_{f,y}$ where $F_{f,y}$ is the (estimated) fully selected fishing mortality rate of landings by fleet.

Table 1. (continued)

Quantity	Symbol	Description or definition
Fishing mortality rate of discards	$F_{d,a,y}$	$F_{d,a,y} = s'_{d,a,r} F_{d,y}$ where $F_{d,y}$ is the (estimated) fully selected fishing mortality rate of discards by fleet.
Total fishing mortality rate	$F_{a,y}$	$F_{a,y} = \sum_{(f,d)} F_{(f,d),a,y}$
Total mortality rate	$Z_{a,y}$	$Z_{a,y} = M_a + F_{a,y}$
Apical F	F_y	$F_y = \max_a(F_{a,y})$
Initialization mortality at age	Z_a^{init}	$Z_a^{init} = M_a + F_a^{init}$ where F_a^{init} is the initialization F at age.
Initial abundance at age per recruit at time of spawning	NPR_a^{init}	Same calculations as for NPR_a (described below), but based on Z_a^{init}.
Initial spawners per recruit	ϕ_{init}	$\phi_{init} = \sum_a NPR_a^{init} \xi_a$
Expected initial recruitment	$\bar{R}_{y1}$	$\bar{R}_{y1} = R^{eq}(\phi_F = \phi_{init})$ where R^{eq} is as described in the text. If recruitment deviations are included in initial year $y1$, bias correction would be excluded from the calculation.
Initial equilibrium abundance at age	N_a^{eq}	Equilibrium age structure given $\bar{R}_{y1}$ and Z_a^{init}.
Abundance at age	$N_{a,y}$	$N_{a,y_1} = N_a^{eq} \sigma_a^{init} \quad \forall a \in (2 \ldots A)$ $N_{1,y_1} = \begin{cases} \bar{R}_y & : \quad y < y' \\ \bar{R}_y \exp(r_y) & : \quad y \geq y' \end{cases}$ where y' is the first year of recruitment deviations. $N_{a+1,y} = N_{a,y} \exp(-Z_{a,y}) \quad \forall a \in (1 \ldots A-1)$ $N_{A,y} = N_{A-1,y-1} \frac{\exp(-Z_{A-1,y-1})}{1-\exp(-Z_{A,y-1})}$
Abundance at age (partial year)	$N'_{a,y}$	Used to match indices of abundance, $N'_{a,y} = N_{a,y} \exp(-t_{\text{index}} Z_{a,y})$.
Abundance at age at time of spawning	$N''_{a,y}$	Assumed to correspond with peak spawning, $N''_{a,y} = \exp(-t_{\text{spawn}} Z_{a,y}) N_{a,y}$.
Unfished abundance at age per recruit at time of spawning	NPR_a	$NPR_1 = 1 \times \exp(-t_{\text{spawn}} M_1)$ $NPR_{a+1} = NPR_a \exp[-(M_a(1-t_{\text{spawn}}) + M_{a+1} t_{\text{spawn}})] \quad \forall a \in (1 \ldots A-1)$ $NPR_A = \frac{NPR_{A-1} \exp[-(M_{A-1}(1-t_{\text{spawn}}) + M_A t_{\text{spawn}})]}{1-\exp(-M_A)}$
Unfished spawners per recruit	ϕ_0	$\phi_0 = \sum_{a=1}^{A} NPR_a \xi_a$
Reproductive output	S_y	$\sum_{a=1}^{A} N''_{a,y} \xi_a$, also called spawning stock.
Population biomass	B_y	$B_y = \sum_a N_{a,y} w_a$
Catchability	$q_{u,y}$	$q_{u,y} = q'_u f^{density}(B'_y) f_u^{trend}(y) f_u^{rw}(y)$ The functions $f^{density}$, f_u^{trend}, and f_u^{rw} are not required but allow for effects of density, time (e.g., technology creep), and random walk, as applied when fitting indices of abundance (see section "Catchability").
Landing at age in numbers	$l'_{f,a,y}$	$l'_{f,a,y} = \frac{F_{f,a,y}}{Z_{a,y}} N_{a,y}[1-\exp(-Z_{a,y})]$

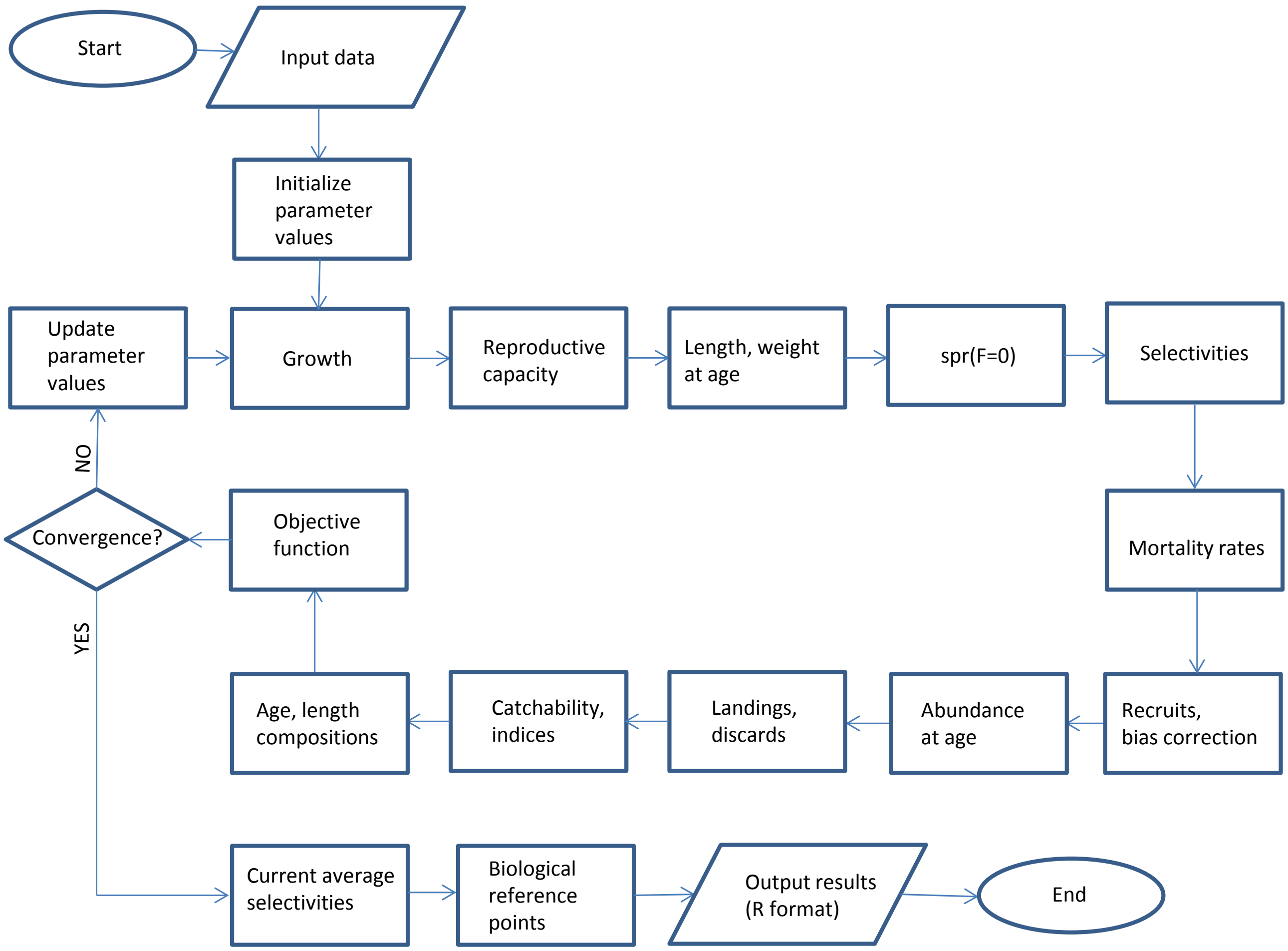
Table 1. (continued)

Quantity	**Symbol**	**Description or definition**
Landing at age in weight	$l''_{f,a,y}$	$l''_{f,a,y} = \mathcal{C} w^L_{f,a,y} l'_{f,a,y}$, where the constant $\mathcal{C}$ converts from units of w^L to those of observed removals.
Discard mortalities at age in numbers	$d'_{d,a,y}$	$d'_{d,a,y} = \frac{F_{d,a,y}}{Z_{a,y}} N_{a,y}[1 - \exp(-Z_{a,y})]$
Discard mortalities at age in weight	$d''_{d,a,y}$	$d''_{d,a,y} = \mathcal{C} w^D_{d,a,y} d'_{d,a,y}$, where the constant $\mathcal{C}$ converts from units of w^D to those of observed removals.
Predicted total landings in numbers	$L'_{f,y}$	$L'_{f,y} = \sum_a l'_{f,a,y}$
Predicted total landings in weight	$L''_{f,y}$	$L''_{f,y} = \sum_a l''_{f,a,y}$
Predicted discard mortalities in numbers	$D'_{d,y}$	$D'_{d,y} = \sum_a d'_{d,a,y}$
Predicted discard mortalities in weight	$D''_{d,y}$	$D''_{d,y} = \sum_a d''_{d,a,y}$
Predicted length compositions of fishery independent data	$p^\lambda_{u,l,y}$	$p^\lambda_{u,l,y} = \frac{\sum_a \psi_{a,l} s_{u,a,y} N'_{a,y}}{\sum_a s_{u,a,y} N'_{a,y}}$
Predicted length compositions of landings	$p^\lambda_{f,l,y}$	$p^\lambda_{f,l,y} = \frac{\sum_a \psi^L_{f,a,l,y} l'_{f,a,y}}{\sum_a l'_{f,a,y}}$
Predicted length compositions of discards	$p^\lambda_{d,l,y}$	$p^\lambda_{d,l,y} = \frac{\sum_a \psi^D_{d,a,l,y} d'_{d,a,y}}{\sum_a d'_{d,a,y}}$
Predicted age compositions of indices	$p^\alpha_{u,a,y}$	$p^\alpha_{u,a,y} = \frac{\mathcal{E} s_{u,a,y} N'_{a,y}}{\sum_a s_{u,a,y} N'_{a,y}}$ The $\mathcal{B}^\alpha \times \mathcal{B}^\alpha$ matrix $\mathcal{E}$ accounts for ageing error, which can be set to the identity matrix if ageing error is not applied.
Predicted age compositions of landings	$p^\alpha_{f,a,y}$	$p^\alpha_{f,a,y} = \frac{\mathcal{E} l'_{f,a,y}}{L'_{f,y}}$ The $\mathcal{B}^\alpha \times \mathcal{B}^\alpha$ matrix $\mathcal{E}$ accounts for ageing error, which can be set to the identity matrix if ageing error is not applied.
Predicted CPUE	$U_{u,y}$	$U_{u,y} = \begin{cases} q_{u,y} \sum_a w_{u,a,y} s_{u,a,y} N'_{a,y} & : \text{ if in weight} \\ q_{u,y} \sum_a s_{u,a,y} N'_{a,y} & : \text{ if in numbers} \end{cases}$ where $s_{u,a,y}$ is the selectivity of the relevant fleet or survey, and $w_{u,a,y}$ is the relevant weight at age (either w_a or $w^L_{f,a,y}$).
Objective Function (Equations in §3)		
Landings	Λ^L	Lognormal likelihood.
Discards	Λ^D	Lognormal likelihood.
Indices of abundance	Λ^U	Lognormal likelihood.
Age compositions	Λ^α	Robust multinomial likelihood.
Length compositions	Λ^λ	Robust multinomial likelihood.
Recruitment deviations	Λ^{R1}	Lognormal likelihood.
Early recruitment deviations	Λ^{R2}	Optional sum of squares penalty for deviations early in the time series.

Table 1. (continued)

Quantity	Symbol	Description or definition
Late recruitment deviations	Λ^{R3}	Optional sum of squares penalty for deviations late in the time series.
Initial age structure deviations	Λ^{init}	Optional sum of squares penalty.
Catchability random walk	Λ^{q}	Optional sum of squares penalty.
Fishing rate	Λ^{F}	Optional penalty on very large values of apical F.
Miscellaneous parameters	Λ^{P}	Sum of optional penalties (a.k.a., priors) applied to any parameter. Each penalty may take the form of a normal, lognormal, or beta likelihood.
Total objective function	Λ	$\Lambda = \Lambda^L + \Lambda^D + \Lambda^U + \Lambda^\alpha + \Lambda^\lambda + \Lambda^{R1} + \Lambda^{R2} + \Lambda^{R3} + \Lambda^{init} + \Lambda^q + \Lambda^F + \Lambda^P$, the overall objective function minimized by the assessment model. Likelihood components are negative likelihoods or negative log-likelihoods.

Figure 1. Flow of operations in the Beaufort Assessment Model.



9 AD Model Builder code to implement the Beaufort Assessment Model. Example application toward gag.

```
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##
//##  Gag Grouper assessment January 2014
//##  NMFS, Beaufort Lab, Sustainable Fisheries Branch
//##
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
DATA_SECTION

!!cout << "Starting Beaufort Assessment Model" << endl;
!!cout << endl;
!!cout << "                BAM!" << endl;
!!cout << endl;

//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//-- BAM DATA_SECTION: set-up section
//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>

// Starting and ending year of the model (year data starts)
init_int styr;
init_int endyr;

//Starting year to estimate recruitment deviation from S-R curve
init_int styr_rec_dev;
//Ending year to estimate recruitment deviation from S-R curve
init_int endyr_rec_dev;
//possible 3 phases of constraints on recruitment deviations
init_int endyr_rec_phase1;
init_int endyr_rec_phase2;

//ending years for selectivity blocks
init_int endyr_selex_phase1;
init_int endyr_selex_phase2;

//number assessment years
number nyrs;
number nyrs_rec;
//this section MUST BE INDENTED!!!
 LOCAL_CALCS
   nyrs=endyr-styr+1.;
   nyrs_rec=endyr_rec_dev-styr_rec_dev+1.;
 END_CALCS

//Total number of ages in population model
init_int nages;
// Vector of ages for age bins in population model
init_vector agebins(1,nages);

//Total number of ages used to match age comps: plus group may differ from popn, first age must not
init_int nages_agec;
//Vector of ages for age bins in age comps
init_vector agebins_agec(1,nages_agec);

//Total number of length bins for each matrix and width of bins)
init_int nlenbins;          //used to match data
init_number lenbins_width;  //width of length bins (mm)

//Vector of lengths for length bins (mm)(midpoint)
init_vector lenbins(1,nlenbins);

//Max F used in spr and msy calcs
init_number max_F_spr_msy;
//Total number of iterations for spr calcs
init_int n_iter_spr;
//Total number of iterations for msy calcs
init_int n_iter_msy;
//Number years at end of time series over which to average sector F's, for weighted selectivities
init_int selpar_n_yrs_wgted;
//bias correction (set to 1.0 for no bias correction or a negative value to compute from rec variance)
init_number set_BiasCor;

//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//-- BAM DATA_SECTION: observed data section
//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>

//###############Commercial handline fleet ######################################
// Comm HL CPUE
init_int styr_cH_cpue;
init_int endyr_cH_cpue;
init_vector obs_cH_cpue(styr_cH_cpue,endyr_cH_cpue); //Observed CPUE
init_vector cH_cpue_cv(styr_cH_cpue,endyr_cH_cpue);  //CV of cpue

// Comm HL  Landings (1000 lb gutted weight)
init_int styr_cH_L;
init_int endyr_cH_L;
init_vector obs_cH_L(styr_cH_L,endyr_cH_L);
init_vector cH_L_cv(styr_cH_L,endyr_cH_L);

// Comm HL  Discards (1000 fish)
init_int styr_cH_D;
init_int endyr_cH_D;
init_vector obs_cH_released(styr_cH_D,endyr_cH_D);
init_vector cH_D_cv(styr_cH_D,endyr_cH_D);

// Comm HL length Compositions (3 cm bins)
init_int nyr_cH_lenc;
```

```
init_ivector yrs_cH_lenc(1,nyr_cH_lenc);
init_vector nsamp_cH_lenc(1,nyr_cH_lenc);
init_vector nfish_cH_lenc(1,nyr_cH_lenc);
init_matrix obs_cH_lenc(1,nyr_cH_lenc,1,nlenbins);

// Comm HL age compositions
init_int nyr_cH_agec;
init_ivector yrs_cH_agec(1,nyr_cH_agec);
init_vector nsamp_cH_agec(1,nyr_cH_agec);
init_vector nfish_cH_agec(1,nyr_cH_agec);
init_matrix obs_cH_agec(1,nyr_cH_agec,1,nages_agec);


//################Commercial diving fleet #######################################
// Comm DV  Landings (1000 lb gutted weight)
init_int styr_cD_L;
init_int endyr_cD_L;
init_vector obs_cD_L(styr_cD_L,endyr_cD_L);
init_vector cD_L_cv(styr_cD_L,endyr_cD_L);

// Comm DV length Compositions (3 cm bins)
init_int nyr_cD_lenc;
init_ivector yrs_cD_lenc(1,nyr_cD_lenc);
init_vector nsamp_cD_lenc(1,nyr_cD_lenc);
init_vector nfish_cD_lenc(1,nyr_cD_lenc);
init_matrix obs_cD_lenc(1,nyr_cD_lenc,1,nlenbins);

// Comm DV age compositions
init_int nyr_cD_agec;
init_ivector yrs_cD_agec(1,nyr_cD_agec);
init_vector nsamp_cD_agec(1,nyr_cD_agec);
init_vector nfish_cD_agec(1,nyr_cD_agec);
init_matrix obs_cD_agec(1,nyr_cD_agec,1,nages_agec);


//##################Headboat fleet ##########################################
// HB CPUE
init_int styr_HB_cpue;
init_int endyr_HB_cpue;
init_vector obs_HB_cpue(styr_HB_cpue,endyr_HB_cpue);//Observed CPUE
init_vector HB_cpue_cv(styr_HB_cpue,endyr_HB_cpue); //CV of cpue

// HB Landings (1000s fish)
init_int styr_HB_L;
init_int endyr_HB_L;
init_vector obs_HB_L(styr_HB_L,endyr_HB_L);   //vector of observed landings by year
init_vector HB_L_cv(styr_HB_L,endyr_HB_L);    //vector of CV of landings by year

// HB  Discards (1000 fish)
init_int styr_HB_D;
init_int endyr_HB_D;
init_vector obs_HB_released(styr_HB_D,endyr_HB_D);
init_vector HB_D_cv(styr_HB_D,endyr_HB_D);

// HB length Compositions (3 cm bins)
init_int nyr_HB_lenc;
init_ivector yrs_HB_lenc(1,nyr_HB_lenc);
init_vector nsamp_HB_lenc(1,nyr_HB_lenc);
init_vector nfish_HB_lenc(1,nyr_HB_lenc);
init_matrix obs_HB_lenc(1,nyr_HB_lenc,1,nlenbins);

// HB age compositions
init_int nyr_HB_agec;
init_ivector yrs_HB_agec(1,nyr_HB_agec);
init_vector nsamp_HB_agec(1,nyr_HB_agec);
init_vector nfish_HB_agec(1,nyr_HB_agec);
init_matrix obs_HB_agec(1,nyr_HB_agec,1,nages_agec);


//##################General Recreational fleet #########################################
// MRFSS CPUE
init_int styr_GR_cpue;
init_int endyr_GR_cpue;
init_vector obs_GR_cpue(styr_GR_cpue,endyr_GR_cpue);//Observed CPUE
init_vector GR_cpue_cv(styr_GR_cpue,endyr_GR_cpue); //CV of cpue

// GR Landings (1000s fish)
init_int styr_GR_L;
init_int endyr_GR_L;
init_vector obs_GR_L(styr_GR_L,endyr_GR_L);   //vector of observed landings by year
init_vector GR_L_cv(styr_GR_L,endyr_GR_L);    //vector of CV of landings by year

// GR  Discards (1000 fish)
init_int styr_GR_D;
init_int endyr_GR_D;
init_vector obs_GR_released(styr_GR_D,endyr_GR_D);
init_vector GR_D_cv(styr_GR_D,endyr_GR_D);

//-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>
//-- BAM DATA_SECTION: parameter section
//-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>-->><>

//#################Single Parameter values and initial guesses #################################
// Von Bert parameters in TL mm all fish
init_vector set_Linf(1,7);
init_vector set_K(1,7);
init_vector set_t0(1,7);
init_vector set_len_cv(1,7); //CV of length at age and its standard error all fish
init_vector set_M_constant(1,7); //Scalar used only for computing MSST.
//Spawner-recruit parameters (Initial guesses or fixed values)
init_vector set_steep(1,7);         //recruitment steepness
init_vector set_log_R0(1,7);        //recruitment R0
init_vector set_R_autocorr(1,7);    //recruitment autocorrelation
init_vector set_rec_sigma(1,7);     //recruitment standard deviation in log space
```

```
//Initial guesses or fixed values of estimated selectivity parameters
init_vector set_selpar_L50_cH1(1,7);
init_vector set_selpar_slope_cH1(1,7);

init_vector set_selpar_L50_cH2(1,7);
init_vector set_selpar_slope_cH2(1,7);

init_vector set_selpar_L50_cH3(1,7);
init_vector set_selpar_slope_cH3(1,7);

init_vector set_selpar_L50_cD(1,7);
init_vector set_selpar_slope_cD(1,7);
init_vector set_selpar_afull_cD(1,7);
init_vector set_selpar_sigma_cD(1,7);

init_vector set_selpar_L50_HB1(1,7);
init_vector set_selpar_slope_HB1(1,7);

init_vector set_selpar_L50_HB2(1,7);
init_vector set_selpar_slope_HB2(1,7);

init_vector set_selpar_L50_HB3(1,7);
init_vector set_selpar_slope_HB3(1,7);

//--index catchability------------------------------------------------------------------------------------
init_vector set_log_q_cH(1,7);      //catchability coefficient (log) for comm handline index
init_vector set_log_q_HB(1,7);      //catchability coefficient (log) for headboat index
init_vector set_log_q_GR(1,7);      //catchability coefficient (log) for general rec index

//initial F
init_vector set_F_init(1,7);  //scales initial F
//--mean F's in log space --------------------------------
init_vector set_log_avg_F_cH(1,7);
init_vector set_log_avg_F_cD(1,7);
init_vector set_log_avg_F_HB(1,7);
init_vector set_log_avg_F_GR(1,7);
init_vector set_log_avg_F_cH_D(1,7);
init_vector set_log_avg_F_HB_D(1,7);
init_vector set_log_avg_F_GR_D(1,7);

//##################Dev Vector Parameter values (vals) and bounds #################################
//--F vectors---------------------------
init_vector set_log_F_dev_cH(1,3);
init_vector set_log_F_dev_cD(1,3);
init_vector set_log_F_dev_HB(1,3);
init_vector set_log_F_dev_GR(1,3);
init_vector set_log_F_dev_cH_D(1,3);
init_vector set_log_F_dev_HB_D(1,3);
init_vector set_log_F_dev_GR_D(1,3);
init_vector set_log_rec_dev(1,3);
init_vector set_log_Nage_dev(1,3);

init_vector set_log_F_dev_cH_vals(styr_cH_L,endyr_cH_L);
init_vector set_log_F_dev_cD_vals(styr_cD_L,endyr_cD_L);
init_vector set_log_F_dev_HB_vals(styr_HB_L,endyr_HB_L);
init_vector set_log_F_dev_GR_vals(styr_GR_L,endyr_GR_L);
init_vector set_log_F_dev_cH_D_vals(styr_cH_D,endyr_cH_D);
init_vector set_log_F_dev_HB_D_vals(styr_HB_D,endyr_HB_D);
init_vector set_log_F_dev_GR_D_vals(styr_GR_D,endyr_GR_D);
init_vector set_log_rec_dev_vals(styr_rec_dev,endyr_rec_dev);
init_vector set_log_Nage_dev_vals(2,nages);


//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//-- BAM DATA_SECTION: likelihood weights section
//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>

init_number set_w_L;            //weight for landings
init_number set_w_D;            //weight for discards
init_number set_w_I_cH;         //weight for comm handline index
init_number set_w_I_HB;         //weight for headboat index
init_number set_w_I_GR;         //weight for MRFSS index
init_number set_w_lc_cH;        //weight for comm handline len comps
init_number set_w_lc_cD;        //weight for comm diving len comps
init_number set_w_lc_HB;        //weight for headboat len comps
init_number set_w_ac_cH;        //weight for comm handline age comps
init_number set_w_ac_cD;        //weight for comm longline age comps
init_number set_w_ac_HB;        //weight for headboat age comps
init_number set_w_Nage_init;    //for fitting initial abundance at age (excluding first age)
init_number set_w_rec;          //for fitting S-R curve
init_number set_w_rec_early;    //additional constraint on early years recruitment
init_number set_w_rec_end;      //additional constraint on ending years recruitment
init_number set_w_fullF;        //penalty for any Fapex>3
init_number set_w_Ftune;        //weight applied to tuning F (removed in final phase of optimization)

//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//-- BAM DATA_SECTION: miscellaneous stuff section
//--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>

//TL(mm)-weight(whole weight in kg) relationship: W=aL^b
init_number wgtpar_a;
init_number wgtpar_b;
//whole weight to gutted weight conversion
init_number ww2gw;

//Maturity and proportion female at age
init_vector maturity_f_obs(1,nages);            //proportion females mature at age
init_vector maturity_m_obs(1,nages);            //proportion males mature at age
init_matrix prop_m_obs(styr,endyr,1,nages);     //proportion male at age
matrix prop_f_obs(styr,endyr,1,nages);          //proportion female at age
 LOCAL_CALCS
    prop_f_obs=1.0-prop_m_obs;
 END_CALCS
```

```
init_number spawn_time_frac; //time of year of peak spawning, as a fraction of the year

// Natural mortality
init_vector set_M(1,nages);     //age-dependent: used in model
init_number max_obs_age;        //max observed age, used to scale M, if estimated

//discard mortality constants
init_number set_Dmort_cH;
init_number set_Dmort_HB;
init_number set_Dmort_GR;

//Spawner-recruit parameters (Initial guesses or fixed values)
init_int SR_switch;

//rate of increase on q
init_int set_q_rate_phase;  //value sets estimation phase of rate increase, negative value turns it off
init_number set_q_rate;
//density dependence on fishery q's
init_int set_q_DD_phase;      //value sets estimation phase of random walk, negative value turns it off
init_number set_q_DD_beta;    //value of 0.0 is density independent
init_number set_q_DD_beta_se;
init_int set_q_DD_stage;      //age to begin counting biomass, should be near full exploitation

//random walk on fishery q's
init_int set_q_RW_phase;         //value sets estimation phase of random walk, negative value turns it off
init_number set_q_RW_rec_var;    //variance of RW q

//Tune Fapex (if applied, tuning removed in final phase of optimization)
init_number set_Ftune;
init_int set_Ftune_yr;

//threshold sample sizes for inclusion of length comps
init_number minSS_cH_lenc;
init_number minSS_cD_lenc;
init_number minSS_HB_lenc;

//threshold sample sizes for inclusion of age comps
init_number minSS_cH_agec;
init_number minSS_cD_agec;
init_number minSS_HB_agec;

//ageing error matrix (columns are true ages, rows are ages as read for age comps: columns must sum to one)
init_matrix age_error(1,nages,1,nages);

// #######Indexing integers for year(iyear), age(iage),length(ilen) ###############
int iyear;
int iage;
int ilen;
int ff;

number sqrt2pi;
number g2mt;                    //conversion of grams to metric tons
number g2kg;                    //conversion of grams to kg
number g2klb;                   //conversion of grams to 1000 lb
number mt2klb;                  //conversion of metric tons to 1000 lb
number mt2lb;                   //conversion of metric tons to lb
number dzero;                   //small additive constant to prevent division by zero
number huge_number;             //huge number, to avoid irregular parameter space

init_number end_of_data_file;
//this section MUST BE INDENTED!!!
 LOCAL_CALCS
   if(end_of_data_file!=999)
   {
       cout << "*** WARNING: Data File NOT READ CORRECTLY ****" << endl;
       exit(0);
   }
   else
   {cout << "Data File read correctly" << endl;}
 END_CALCS


PARAMETER_SECTION

 LOCAL_CALCS
  const double Linf_LO=set_Linf(2); const double Linf_HI=set_Linf(3); const double Linf_PH=set_Linf(4);
  const double K_LO=set_K(2); const double K_HI=set_K(3); const double K_PH=set_K(4);
  const double t0_LO=set_t0(2); const double t0_HI=set_t0(3); const double t0_PH=set_t0(4);
  const double len_cv_LO=set_len_cv(2); const double len_cv_HI=set_len_cv(3); const double len_cv_PH=set_len_cv(4);
  const double M_constant_LO=set_M_constant(2); const double M_constant_HI=set_M_constant(3); const double M_constant_PH=set_M_constant(4);
  const double steep_LO=set_steep(2); const double steep_HI=set_steep(3); const double steep_PH=set_steep(4);
  const double log_R0_LO=set_log_R0(2); const double log_R0_HI=set_log_R0(3); const double log_R0_PH=set_log_R0(4);
  const double R_autocorr_LO=set_R_autocorr(2); const double R_autocorr_HI=set_R_autocorr(3); const double R_autocorr_PH=set_R_autocorr(4);
  const double rec_sigma_LO=set_rec_sigma(2); const double rec_sigma_HI=set_rec_sigma(3); const double rec_sigma_PH=set_rec_sigma(4);

  const double selpar_L50_cH1_LO=set_selpar_L50_cH1(2); const double selpar_L50_cH1_HI=set_selpar_L50_cH1(3); const double selpar_L50_cH1_PH=set_selpar_L50_cH1(4);
  const double selpar_slope_cH1_LO=set_selpar_slope_cH1(2); const double selpar_slope_cH1_HI=set_selpar_slope_cH1(3); const double selpar_slope_cH1_PH=set_selpar_slope_cH1(4);
  const double selpar_L50_cH2_LO=set_selpar_L50_cH2(2); const double selpar_L50_cH2_HI=set_selpar_L50_cH2(3); const double selpar_L50_cH2_PH=set_selpar_L50_cH2(4);
  const double selpar_slope_cH2_LO=set_selpar_slope_cH2(2); const double selpar_slope_cH2_HI=set_selpar_slope_cH2(3); const double selpar_slope_cH2_PH=set_selpar_slope_cH2(4);
  const double selpar_L50_cH3_LO=set_selpar_L50_cH3(2); const double selpar_L50_cH3_HI=set_selpar_L50_cH3(3); const double selpar_L50_cH3_PH=set_selpar_L50_cH3(4);
  const double selpar_slope_cH3_LO=set_selpar_slope_cH3(2); const double selpar_slope_cH3_HI=set_selpar_slope_cH3(3); const double selpar_slope_cH3_PH=set_selpar_slope_cH3(4);
  const double selpar_L50_cD_LO=set_selpar_L50_cD(2); const double selpar_L50_cD_HI=set_selpar_L50_cD(3); const double selpar_L50_cD_PH=set_selpar_L50_cD(4);
  const double selpar_slope_cD_LO=set_selpar_slope_cD(2); const double selpar_slope_cD_HI=set_selpar_slope_cD(3); const double selpar_slope_cD_PH=set_selpar_slope_cD(4);
  const double selpar_afull_cD_LO=set_selpar_afull_cD(2); const double selpar_afull_cD_HI=set_selpar_afull_cD(3); const double selpar_afull_cD_PH=set_selpar_afull_cD(4);
  const double selpar_sigma_cD_LO=set_selpar_sigma_cD(2); const double selpar_sigma_cD_HI=set_selpar_sigma_cD(3); const double selpar_sigma_cD_PH=set_selpar_sigma_cD(4);

  const double selpar_L50_HB1_LO=set_selpar_L50_HB1(2); const double selpar_L50_HB1_HI=set_selpar_L50_HB1(3); const double selpar_L50_HB1_PH=set_selpar_L50_HB1(4);
  const double selpar_slope_HB1_LO=set_selpar_slope_HB1(2); const double selpar_slope_HB1_HI=set_selpar_slope_HB1(3); const double selpar_slope_HB1_PH=set_selpar_slope_HB1(4);
  const double selpar_L50_HB2_LO=set_selpar_L50_HB2(2); const double selpar_L50_HB2_HI=set_selpar_L50_HB2(3); const double selpar_L50_HB2_PH=set_selpar_L50_HB2(4);
  const double selpar_slope_HB2_LO=set_selpar_slope_HB2(2); const double selpar_slope_HB2_HI=set_selpar_slope_HB2(3); const double selpar_slope_HB2_PH=set_selpar_slope_HB2(4);
  const double selpar_L50_HB3_LO=set_selpar_L50_HB3(2); const double selpar_L50_HB3_HI=set_selpar_L50_HB3(3); const double selpar_L50_HB3_PH=set_selpar_L50_HB3(4);
  const double selpar_slope_HB3_LO=set_selpar_slope_HB3(2); const double selpar_slope_HB3_HI=set_selpar_slope_HB3(3); const double selpar_slope_HB3_PH=set_selpar_slope_HB3(4);
```

```
  const double log_q_cH_LO=set_log_q_cH(2); const double log_q_cH_HI=set_log_q_cH(3); const double log_q_cH_PH=set_log_q_cH(4);
  const double log_q_HB_LO=set_log_q_HB(2); const double log_q_HB_HI=set_log_q_HB(3); const double log_q_HB_PH=set_log_q_HB(4);
  const double log_q_GR_LO=set_log_q_GR(2); const double log_q_GR_HI=set_log_q_GR(3); const double log_q_GR_PH=set_log_q_GR(4);

  const double F_init_LO=set_F_init(2); const double F_init_HI=set_F_init(3); const double F_init_PH=set_F_init(4);
  const double log_avg_F_cH_LO=set_log_avg_F_cH(2); const double log_avg_F_cH_HI=set_log_avg_F_cH(3); const double log_avg_F_cH_PH=set_log_avg_F_cH(4);
  const double log_avg_F_cD_LO=set_log_avg_F_cD(2); const double log_avg_F_cD_HI=set_log_avg_F_cD(3); const double log_avg_F_cD_PH=set_log_avg_F_cD(4);
  const double log_avg_F_HB_LO=set_log_avg_F_HB(2); const double log_avg_F_HB_HI=set_log_avg_F_HB(3); const double log_avg_F_HB_PH=set_log_avg_F_HB(4);
  const double log_avg_F_GR_LO=set_log_avg_F_GR(2); const double log_avg_F_GR_HI=set_log_avg_F_GR(3); const double log_avg_F_GR_PH=set_log_avg_F_GR(4);
  const double log_avg_F_cH_D_LO=set_log_avg_F_cH_D(2); const double log_avg_F_cH_D_HI=set_log_avg_F_cH_D(3); const double log_avg_F_cH_D_PH=set_log_avg_F_cH_D(4);
  const double log_avg_F_HB_D_LO=set_log_avg_F_HB_D(2); const double log_avg_F_HB_D_HI=set_log_avg_F_HB_D(3); const double log_avg_F_HB_D_PH=set_log_avg_F_HB_D(4);
  const double log_avg_F_GR_D_LO=set_log_avg_F_GR_D(2); const double log_avg_F_GR_D_HI=set_log_avg_F_GR_D(3); const double log_avg_F_GR_D_PH=set_log_avg_F_GR_D(4);

  //-dev vectors-----------------------------------------------------------------------------------------------------------
  const double log_F_dev_cH_LO=set_log_F_dev_cH(1); const double log_F_dev_cH_HI=set_log_F_dev_cH(2); const double log_F_dev_cH_PH=set_log_F_dev_cH(3);
  const double log_F_dev_cD_LO=set_log_F_dev_cD(1); const double log_F_dev_cD_HI=set_log_F_dev_cD(2); const double log_F_dev_cD_PH=set_log_F_dev_cD(3);
  const double log_F_dev_HB_LO=set_log_F_dev_HB(1); const double log_F_dev_HB_HI=set_log_F_dev_HB(2); const double log_F_dev_HB_PH=set_log_F_dev_HB(3);
  const double log_F_dev_GR_LO=set_log_F_dev_GR(1); const double log_F_dev_GR_HI=set_log_F_dev_GR(2); const double log_F_dev_GR_PH=set_log_F_dev_GR(3);

  const double log_F_dev_cH_D_LO=set_log_F_dev_cH_D(1); const double log_F_dev_cH_D_HI=set_log_F_dev_cH_D(2); const double log_F_dev_cH_D_PH=set_log_F_dev_cH_D(3);
  const double log_F_dev_HB_D_LO=set_log_F_dev_HB_D(1); const double log_F_dev_HB_D_HI=set_log_F_dev_HB_D(2); const double log_F_dev_HB_D_PH=set_log_F_dev_HB_D(3);
  const double log_F_dev_GR_D_LO=set_log_F_dev_GR_D(1); const double log_F_dev_GR_D_HI=set_log_F_dev_GR_D(2); const double log_F_dev_GR_D_PH=set_log_F_dev_GR_D(3);

  const double log_rec_dev_LO=set_log_rec_dev(1); const double log_rec_dev_HI=set_log_rec_dev(2); const double log_rec_dev_PH=set_log_rec_dev(3);
  const double log_Nage_dev_LO=set_log_Nage_dev(1); const double log_Nage_dev_HI=set_log_Nage_dev(2); const double log_Nage_dev_PH=set_log_Nage_dev(3);

 END_CALCS

////--------------Growth---------------------------------------------------------------------------
  init_bounded_number Linf(Linf_LO,Linf_HI,Linf_PH);
  init_bounded_number K(K_LO,K_HI,K_PH);
  init_bounded_number t0(t0_LO,t0_HI,t0_PH);
  init_bounded_number len_cv_val(len_cv_LO,len_cv_HI,len_cv_PH);
  vector Linf_out(1,8);
  vector K_out(1,8);
  vector t0_out(1,8);
  vector len_cv_val_out(1,8);

  vector meanlen_TL(1,nages);   //mean total length (mm) at age all fish
  vector wgt_g(1,nages);        //whole wgt in g
  vector wgt_kg(1,nages);       //whole wgt in kg
  vector wgt_mt(1,nages);       //whole wgt in mt
  vector wgt_klb(1,nages);      //whole wgt in 1000 lb
  vector wgt_lb(1,nages);       //whole wgt in lb
  vector wgt_klb_gut(1,nages);  //gutted wgt in 1000 lb
  vector wgt_lb_gut(1,nages);   //gutted wgt in lb

  matrix len_cH_mm(styr,endyr,1,nages);          //mean length at age of commercial handline landings in mm (may differ from popn mean)
  matrix wholewgt_cH_klb(styr,endyr,1,nages);    //whole wgt of commercial handline landings in 1000 lb
  matrix gutwgt_cH_klb(styr,endyr,1,nages);      //gutted wgt of commercial handline landings in 1000 lb
  matrix len_cD_mm(styr,endyr,1,nages);          //mean length at age of commercial handline landings in mm (may differ from popn mean)
  matrix wholewgt_cD_klb(styr,endyr,1,nages);    //whole wgt of commercial diving landings in 1000 lb
  matrix gutwgt_cD_klb(styr,endyr,1,nages);      //gutted wgt of commercial diving landings in 1000 lb
  matrix len_HB_mm(styr,endyr,1,nages);          //mean length at age of HB landings in mm (may differ from popn mean)
  matrix wholewgt_HB_klb(styr,endyr,1,nages);    //whole wgt of HB landings in 1000 lb
  matrix gutwgt_HB_klb(styr,endyr,1,nages);      //gutted wgt of HB landings in 1000 lb
  matrix len_GR_mm(styr,endyr,1,nages);          //mean length at age of GR landings in mm (may differ from popn mean)
  matrix wholewgt_GR_klb(styr,endyr,1,nages);    //whole wgt of GR landings in 1000 lb
  matrix gutwgt_GR_klb(styr,endyr,1,nages);      //gutted wgt of GR landings in 1000 lb

  matrix len_cH_D_mm(styr,endyr,1,nages);          //mean length at age of commercial handline discards in mm (may differ from popn mean)
  matrix wholewgt_cH_D_klb(styr,endyr,1,nages);    //whole wgt of commercial handline discards in 1000 lb
  matrix gutwgt_cH_D_klb(styr,endyr,1,nages);      //gutted wgt of commercial handline discards in 1000 lb
  matrix len_HB_D_mm(styr,endyr,1,nages);          //mean length at age of HB discards in mm (may differ from popn mean)
  matrix wholewgt_HB_D_klb(styr,endyr,1,nages);    //whole wgt of HB discards in 1000 lb
  matrix gutwgt_HB_D_klb(styr,endyr,1,nages);      //gutted wgt of HB discards in 1000 lb
  matrix len_GR_D_mm(styr,endyr,1,nages);          //mean length at age of GR discards in mm (may differ from popn mean)
  matrix wholewgt_GR_D_klb(styr,endyr,1,nages);    //whole wgt of GR discards in 1000 lb
  matrix gutwgt_GR_D_klb(styr,endyr,1,nages);      //gutted wgt of GR discards in 1000 lb

  matrix lenprob(1,nages,1,nlenbins);           //distn of size at age (age-length key, 3 cm bins) in population
  number zscore_len;                            //standardized normal values used for computing lenprob
  vector cprob_lenvec(1,nlenbins);              //cumulative probabilities used for computing lenprob
  number zscore_lzero;                          //standardized normal values for length = 0
  number cprob_lzero;                           //length probability mass below zero, used for computing lenprob


  //matrices below are used to match length comps
  matrix lenprob_cH(1,nages,1,nlenbins);     //distn of size at age in cH
  matrix lenprob_cD(1,nages,1,nlenbins);     //distn of size at age in cD
  matrix lenprob_HB(1,nages,1,nlenbins);     //distn of size at age in HB (rec)

  vector len_sd(1,nages);
  vector len_cv(1,nages); //for fishgraph

//----Predicted length and age compositions
  matrix pred_cH_lenc(1,nyr_cH_lenc,1,nlenbins);
  matrix pred_cD_lenc(1,nyr_cD_lenc,1,nlenbins);
  matrix pred_HB_lenc(1,nyr_HB_lenc,1,nlenbins);

  matrix pred_cH_agec(1,nyr_cH_agec,1,nages_agec);
  matrix pred_cH_agec_allages(1,nyr_cH_agec,1,nages);
  matrix ErrorFree_cH_agec(1,nyr_cH_agec,1,nages);
  matrix pred_cD_agec(1,nyr_cD_agec,1,nages_agec);
  matrix pred_cD_agec_allages(1,nyr_cD_agec,1,nages);
  matrix ErrorFree_cD_agec(1,nyr_cD_agec,1,nages);
  matrix pred_HB_agec(1,nyr_HB_agec,1,nages_agec);
  matrix pred_HB_agec_allages(1,nyr_HB_agec,1,nages);
  matrix ErrorFree_HB_agec(1,nyr_HB_agec,1,nages);

//Effective sample size applied in multinomial distributions
  vector nsamp_cH_lenc_allyr(styr,endyr);
  vector nsamp_cD_lenc_allyr(styr,endyr);
  vector nsamp_HB_lenc_allyr(styr,endyr);
```

```
  vector nsamp_cH_agec_allyr(styr,endyr);
  vector nsamp_cD_agec_allyr(styr,endyr);
  vector nsamp_HB_agec_allyr(styr,endyr);

//Nfish used in MCB analysis (not used in fitting)
  vector nfish_cH_lenc_allyr(styr,endyr);
  vector nfish_cD_lenc_allyr(styr,endyr);
  vector nfish_HB_lenc_allyr(styr,endyr);
  vector nfish_cH_agec_allyr(styr,endyr);
  vector nfish_cD_agec_allyr(styr,endyr);
  vector nfish_HB_agec_allyr(styr,endyr);

//Computed effective sample size for output (not used in fitting)
  vector neff_cH_lenc_allyr_out(styr,endyr);
  vector neff_cD_lenc_allyr_out(styr,endyr);
  vector neff_HB_lenc_allyr_out(styr,endyr);
  vector neff_cH_agec_allyr_out(styr,endyr);
  vector neff_cD_agec_allyr_out(styr,endyr);
  vector neff_HB_agec_allyr_out(styr,endyr);

//-----Population-----------------------------------------------------------------------------------
  matrix N(styr,endyr+1,1,nages);           //Population numbers by year and age at start of yr
  matrix N_mdyr(styr,endyr,1,nages);        //Population numbers by year and age at mdpt of yr: used for comps and cpue
  matrix N_spawn(styr,endyr+1,1,nages);     //Population numbers by year and age at peaking spawning: used for SSB
  init_bounded_vector log_Nage_dev(2,nages,log_Nage_dev_LO,log_Nage_dev_HI,log_Nage_dev_PH);
  vector log_Nage_dev_output(1,nages);      //used in output. equals zero for first age
  matrix B(styr,endyr+1,1,nages);           //Population biomass by year and age at start of yr
  vector totB(styr,endyr+1);                //Total biomass by year
  vector totN(styr,endyr+1);                //Total abundance by year
  vector SSB(styr,endyr+1);                 //Total spawning biomass by year (female + male mature biomass)
  vector MatFemB(styr,endyr+1);             //Total spawning biomass by year (mature female biomass)
  vector rec(styr,endyr+1);                 //Recruits by year
  matrix prop_m(styr,endyr,1,nages);        //Year-dependent proportion male by age
  matrix prop_f(styr,endyr,1,nages);        //Year-dependent proportion female by age
  vector maturity_f(1,nages);               //Proportion of female mature at age
  vector maturity_m(1,nages);               //Proportion of male mature at age
  matrix reprod(styr,endyr,1,nages);        //vector used to compute spawning biomass (total mature biomass: males + females)
  matrix reprod2(styr,endyr,1,nages);       //vector used to compute mature female biomass

//---Stock-Recruit Function (Beverton-Holt, steepness parameterization)----------
  init_bounded_number log_R0(log_R0_LO,log_R0_HI,log_R0_PH);//log(virgin Recruitment)
  vector log_R0_out(1,8);
  number R0;                                  //virgin recruitment
  init_bounded_number steep(steep_LO,steep_HI,steep_PH);  //steepness
  vector steep_out(1,8);
  init_bounded_number rec_sigma(rec_sigma_LO,rec_sigma_HI,rec_sigma_PH);   //sd recruitment residuals
  vector rec_sigma_out(1,8);
  init_bounded_number R_autocorr(R_autocorr_LO,R_autocorr_HI,R_autocorr_PH);   //autocorrelation in SR
  vector R_autocorr_out(1,8);

  number rec_sigma_sq;                        //square of rec_sigma
  number rec_logL_add;                        //additive term in -logL term

  init_bounded_dev_vector log_rec_dev(styr_rec_dev,endyr_rec_dev,log_rec_dev_LO,log_rec_dev_HI,log_rec_dev_PH);
  vector log_rec_dev_output(styr,endyr+1);             //used in t.series output. equals zero except for yrs in log_rec_dev
  vector log_rec_dev_out(styr_rec_dev,endyr_rec_dev);   //used in output for bound checking

  number var_rec_dev;                                //variance of log recruitment deviations
  number sigma_rec_dev;                              //sample SD of log residuals (may not equal rec_sigma

  number BiasCor;                               //bias correction in equilibrium recruits
  number S0;                                    //equal to spr_F0*R0 = virgin SSB
  number B0;                                    //equal to bpr_F0*R0 = virgin B
  number R1;                                    //Recruits in styr
  number R_virgin;                              //unfished recruitment with bias correction
  vector SdS0(styr,endyr+1);                      //SSB / virgin SSB


//--------------------------------------------------------------------------------------------------------------------------------------------
////---Selectivity-------------------------------------------------------------------------

//Commercial handline------------------------------------------------
  matrix sel_cH(styr,endyr,1,nages);
  init_bounded_number selpar_L50_cH1(selpar_L50_cH1_LO,selpar_L50_cH1_HI,selpar_L50_cH1_PH);
  init_bounded_number selpar_slope_cH1(selpar_slope_cH1_LO,selpar_slope_cH1_HI,selpar_slope_cH1_PH);
  init_bounded_number selpar_L50_cH2(selpar_L50_cH2_LO,selpar_L50_cH2_HI,selpar_L50_cH2_PH);
  init_bounded_number selpar_slope_cH2(selpar_slope_cH2_LO,selpar_slope_cH2_HI,selpar_slope_cH2_PH);
  init_bounded_number selpar_L50_cH3(selpar_L50_cH3_LO,selpar_L50_cH3_HI,selpar_L50_cH3_PH);
  init_bounded_number selpar_slope_cH3(selpar_slope_cH3_LO,selpar_slope_cH3_HI,selpar_slope_cH3_PH);
  vector selpar_L50_cH1_out(1,8);
  vector selpar_slope_cH1_out(1,8);
  vector selpar_L50_cH2_out(1,8);
  vector selpar_slope_cH2_out(1,8);
  vector selpar_L50_cH3_out(1,8);
  vector selpar_slope_cH3_out(1,8);

//Commercial diving------------------------------------------------
  matrix sel_cD(styr,endyr,1,nages);
  init_bounded_number selpar_L50_cD(selpar_L50_cD_LO,selpar_L50_cD_HI,selpar_L50_cD_PH);
  init_bounded_number selpar_slope_cD(selpar_slope_cD_LO,selpar_slope_cD_HI,selpar_slope_cD_PH);
  init_bounded_number selpar_afull_cD(selpar_afull_cD_LO,selpar_afull_cD_HI,selpar_afull_cD_PH);
  init_bounded_number selpar_sigma_cD(selpar_sigma_cD_LO,selpar_sigma_cD_HI,selpar_sigma_cD_PH);
  vector selpar_L50_cD_out(1,8);
  vector selpar_slope_cD_out(1,8);
  vector selpar_afull_cD_out(1,8);
  vector selpar_sigma_cD_out(1,8);

//Recreational (HB and GR)------------------------------------------------
  matrix sel_HB(styr,endyr,1,nages);
  init_bounded_number selpar_L50_HB1(selpar_L50_HB1_LO,selpar_L50_HB1_HI,selpar_L50_HB1_PH);
  init_bounded_number selpar_slope_HB1(selpar_slope_HB1_LO,selpar_slope_HB1_HI,selpar_slope_HB1_PH);
  init_bounded_number selpar_L50_HB2(selpar_L50_HB2_LO,selpar_L50_HB2_HI,selpar_L50_HB2_PH);
  init_bounded_number selpar_slope_HB2(selpar_slope_HB2_LO,selpar_slope_HB2_HI,selpar_slope_HB2_PH);
  init_bounded_number selpar_L50_HB3(selpar_L50_HB3_LO,selpar_L50_HB3_HI,selpar_L50_HB3_PH);
```

```
  init_bounded_number selpar_slope_HB3(selpar_slope_HB3_LO,selpar_slope_HB3_HI,selpar_slope_HB3_PH);

  vector selpar_L50_HB1_out(1,8);
  vector selpar_slope_HB1_out(1,8);
  vector selpar_L50_HB2_out(1,8);
  vector selpar_slope_HB2_out(1,8);
  vector selpar_L50_HB3_out(1,8);
  vector selpar_slope_HB3_out(1,8);

//Discard selectivities
  matrix sel_cH_D(styr,endyr,1,nages);
  matrix sel_HB_D(styr,endyr,1,nages);


//Weighted total selectivity (effort-weighted, recent selectivities)--------------------------------------------
  vector sel_wgted_L(1,nages);  //toward landings
  vector sel_wgted_D(1,nages);  //toward discards
  vector sel_wgted_tot(1,nages);//toward Z, landings plus deads discards

//-----------------------------------------------------------------------------------------------------------------------------------------------------
//-------CPUE Predictions--------------------------------
  vector pred_cH_cpue(styr_cH_cpue,endyr_cH_cpue);                  //predicted cH index (weight fish per effort)
  matrix N_cH(styr_cH_cpue,endyr_cH_cpue,1,nages);                  //used to compute cH index
  vector pred_HB_cpue(styr_HB_cpue,endyr_HB_cpue);                  //predicted HB index (number fish per effort)
  matrix N_HB(styr_HB_cpue,endyr_HB_cpue,1,nages);                  //used to compute HB index
  vector pred_GR_cpue(styr_GR_cpue,endyr_GR_cpue);                  //predicted GR index (number fish per effort)
  matrix N_GR(styr_GR_cpue,endyr_GR_cpue,1,nages);                  //used to compute GR index


//---Catchability (CPUE q's)----------------------------------------------------------
  init_bounded_number log_q_cH(log_q_cH_LO,log_q_cH_HI,log_q_cH_PH);
  init_bounded_number log_q_HB(log_q_HB_LO,log_q_HB_HI,log_q_HB_PH);
  init_bounded_number log_q_GR(log_q_GR_LO,log_q_GR_HI,log_q_GR_PH);
  vector log_q_cH_out(1,8);
  vector log_q_HB_out(1,8);
  vector log_q_GR_out(1,8);

  init_bounded_number q_rate(0.001,0.1,set_q_rate_phase);   //not estimated in this model
  vector q_rate_fcn_cH(styr_cH_cpue,endyr_cH_cpue);         //increase due to technology creep (saturates in 2003)
  vector q_rate_fcn_HB(styr_HB_cpue,endyr_HB_cpue);         //increase due to technology creep (saturates in 2003)
  vector q_rate_fcn_GR(styr_GR_cpue,endyr_GR_cpue);         //increase due to technology creep (saturates in 2003)

  init_bounded_number q_DD_beta(0.1,0.9,set_q_DD_phase);   //not estimated in this model
  vector q_DD_fcn(styr,endyr);    //density dependent function as a multiple of q (scaled a la Katsukawa and Matsuda. 2003)
  number B0_q_DD;                 //B0 of ages q_DD_age plus
  vector B_q_DD(styr,endyr);      //annual biomass of ages q_DD_age plus

//Fishery dependent random walk catchability
//  init_bounded_vector q_RW_log_dev_HB(styr_HB_cpue,endyr_HB_cpue-1,-3.0,3.0,set_q_RW_phase); //NOT estimated in this model
 vector q_RW_log_dev_cH(styr_cH_cpue,endyr_cH_cpue-1);
 vector q_RW_log_dev_HB(styr_HB_cpue,endyr_HB_cpue-1);
 vector q_RW_log_dev_GR(styr_GR_cpue,endyr_GR_cpue-1);

//Fishery dependent catchability over time, may be constant
 vector q_cH(styr_cH_cpue,endyr_cH_cpue);
 vector q_HB(styr_HB_cpue,endyr_HB_cpue);
 vector q_GR(styr_GR_cpue,endyr_GR_cpue);


//-----------------------------------------------------------------------------------------------------------------------------------------------------
//---Landings in numbers (total or 1000 fish) and in wgt (1000 lb gutted)--------------------------------------------------
  matrix L_cH_num(styr,endyr,1,nages);   //landings (numbers) at age
  matrix L_cH_klb(styr,endyr,1,nages);   //landings (1000 lb gutted weight) at age
  vector pred_cH_L_knum(styr,endyr);     //yearly landings in 1000 fish summed over ages
  vector pred_cH_L_klb(styr,endyr);      //yearly landings in 1000 lb gutted summed over ages

  matrix L_cD_num(styr,endyr,1,nages);   //landings (numbers) at age
  matrix L_cD_klb(styr,endyr,1,nages);   //landings (1000 lb gutted weight) at age
  vector pred_cD_L_knum(styr,endyr);     //yearly landings in 1000 fish summed over ages
  vector pred_cD_L_klb(styr,endyr);      //yearly landings in 1000 lb gutted summed over ages

  matrix L_HB_num(styr,endyr,1,nages);   //landings (numbers) at age
  matrix L_HB_klb(styr,endyr,1,nages);   //landings (1000 lb gutted weight) at age
  vector pred_HB_L_knum(styr,endyr);     //yearly landings in 1000 fish summed over ages
  vector pred_HB_L_klb(styr,endyr);      //yearly landings in 1000 lb gutted summed over ages

  matrix L_GR_num(styr,endyr,1,nages);   //landings (numbers) at age
  matrix L_GR_klb(styr,endyr,1,nages);   //landings (1000 lb gutted weight) at age
  vector pred_GR_L_knum(styr,endyr);     //yearly landings in 1000 fish summed over ages
  vector pred_GR_L_klb(styr,endyr);      //yearly landings in 1000 lb gutted summed over ages

  matrix L_total_num(styr,endyr,1,nages);//total landings in number at age
  matrix L_total_klb(styr,endyr,1,nages);//landings in klb gutted wgt at age
  vector L_total_knum_yr(styr,endyr);    //total landings in 1000 fish by yr summed over ages
  vector L_total_klb_yr(styr,endyr);     //total landings (klb gutted wgt) by yr summed over ages

//---Discards (number dead fish) --------------------------------------------------
  matrix D_cH_num(styr,endyr,1,nages);             //discards (numbers) at age
  matrix D_cH_klb(styr,endyr,1,nages);             //discards (1000 lb gutted) at age
  vector pred_cH_D_knum(styr_cH_D,endyr_cH_D);     //yearly dead discards summed over ages
  vector obs_cH_D(styr_cH_D,endyr_cH_D);           //observed releases multiplied by discard mortality
  vector pred_cH_D_klb(styr_cH_D,endyr_cH_D);      //yearly dead discards in klb gutted wgt summed over ages

  matrix D_HB_num(styr,endyr,1,nages);             //discards (numbers) at age
  matrix D_HB_klb(styr,endyr,1,nages);             //discards (1000 lb gutted) at age
  vector pred_HB_D_knum(styr_HB_D,endyr_HB_D);     //yearly dead discards summed over ages
  vector obs_HB_D(styr_HB_D,endyr_HB_D);           //observed releases multiplied by discard mortality
  vector pred_HB_D_klb(styr_HB_D,endyr_HB_D);      //yearly dead discards in klb gutted wgt summed over ages

  matrix D_GR_num(styr,endyr,1,nages);             //discards (numbers) at age
  matrix D_GR_klb(styr,endyr,1,nages);             //discards (1000 lb gutted) at age
  vector pred_GR_D_knum(styr_GR_D,endyr_GR_D);     //yearly dead discards summed over ages
  vector obs_GR_D(styr_GR_D,endyr_GR_D);           //observed releases multiplied by discard mortality
  vector pred_GR_D_klb(styr_GR_D,endyr_GR_D);      //yearly dead discards in klb gutted wgt summed over ages
```

```
  matrix D_total_num(styr,endyr,1,nages);          //total discards in number at age
  matrix D_total_klb(styr,endyr,1,nages);          //discards in klb gutted wgt at age
  vector D_total_knum_yr(styr,endyr);              //total discards in 1000 fish by yr summed over ages
  vector D_total_klb_yr(styr,endyr);               //total discards (klb gutted wgt) by yr summed over ages

  number Dmort_cH;
  number Dmort_HB;
  number Dmort_GR;

////---MSY calcs----------------------------------------------------------------------------
  number F_cH_prop;       //proportion of F_sum attributable to cH, last X=selpar_n_yrs_wgted yrs
  number F_cD_prop;       //proportion of F_sum attributable to cD, last X=selpar_n_yrs_wgted yrs
  number F_HB_prop;       //proportion of F_sum attributable to HB, last X=selpar_n_yrs_wgted yrs
  number F_GR_prop;       //proportion of F_sum attributable to GR, last X=selpar_n_yrs_wgted yrs
  number F_cH_D_prop;     //proportion of F_sum attributable to cH discards, last X=selpar_n_yrs_wgted yrs
  number F_HB_D_prop;     //proportion of F_sum attributable to HB, last X=selpar_n_yrs_wgted yrs
  number F_GR_D_prop;     //proportion of F_sum attributable to GR, last X=selpar_n_yrs_wgted yrs

  number F_init_cH_prop;  //proportion of F_init attributable to cH, first X yrs, No diving or discards in initial yrs
  number F_init_HB_prop;  //proportion of F_init attributable to HB, first X yrs
  number F_init_GR_prop;  //proportion of F_init attributable to GR, first X yrs

  number F_temp_sum;      //sum of geom mean Fsum's in last X yrs, used to compute F_fishery_prop

  vector F_end(1,nages);
  vector F_end_L(1,nages);
  vector F_end_D(1,nages);
  number F_end_apex;

  number SSB_msy_out;           //SSB (total mature biomass) at msy
  number F_msy_out;             //F at msy
  number msy_klb_out;           //max sustainable yield (1000 lb gutted wgt)
  number msy_knum_out;          //max sustainable yield (1000 fish)
  number D_msy_klb_out;         //discards associated with msy (1000 lb gutted wgt)
  number D_msy_knum_out;        //discards associated with msy (1000 fish)
  number B_msy_out;             //total biomass at MSY
  number R_msy_out;             //equilibrium recruitment at F=Fmsy
  number spr_msy_out;           //spr at F=Fmsy

  vector N_age_msy(1,nages);         //numbers at age for MSY calculations: beginning of yr
  vector N_age_msy_spawn(1,nages);   //numbers at age for MSY calculations: time of peak spawning
  vector L_age_msy(1,nages);         //landings at age for MSY calculations
  vector D_age_msy(1,nages);         //discard mortality (dead discards) at age for MSY calculations
  vector Z_age_msy(1,nages);         //total mortality at age for MSY calculations
  vector F_L_age_msy(1,nages);       //fishing mortality landings (not discards) at age for MSY calculations
  vector F_D_age_msy(1,nages);       //fishing mortality of discards at age for MSY calculations
  vector F_msy(1,n_iter_msy);        //values of full F to be used in equilibrium calculations
  vector spr_msy(1,n_iter_msy);      //reproductive capacity-per-recruit values corresponding to F values in F_msy
  vector R_eq(1,n_iter_msy);         //equilibrium recruitment values corresponding to F values in F_msy
  vector L_eq_klb(1,n_iter_msy);     //equilibrium landings(klb gutted wgt) values corresponding to F values in F_msy
  vector L_eq_knum(1,n_iter_msy);    //equilibrium landings(1000 fish) values corresponding to F values in F_msy
  vector D_eq_klb(1,n_iter_msy);     //equilibrium discards(klb gutted wgt) values corresponding to F values in F_msy
  vector D_eq_knum(1,n_iter_msy);    //equilibrium discards(1000 fish) values corresponding to F values in F_msy
  vector SSB_eq(1,n_iter_msy);       //equilibrium reproductive capacity values corresponding to F values in F_msy
  vector B_eq(1,n_iter_msy);         //equilibrium biomass values corresponding to F values in F_msy

  vector FdF_msy(styr,endyr);
  vector SdSSB_msy(styr,endyr+1);
  number SdSSB_msy_end;
  number FdF_msy_end;
  number FdF_msy_end_mean;           //geometric mean of last X yrs

  vector wgt_wgted_L_klb(1,nages);   //fishery-weighted average weight at age of landings in gutted weight
  vector wgt_wgted_D_klb(1,nages);   //fishery-weighted average weight at age of discards in gutted weight
  number wgt_wgted_L_denom;          //used in intermediate calculations
  number wgt_wgted_D_denom;          //used in intermediate calculations

  number iter_inc_msy;               //increments used to compute msy, equals 1/(n_iter_msy-1)

////--------Mortality------------------------------------------------------------------

  vector M(1,nages);                         //age-dependent natural mortality
  init_bounded_number M_constant(M_constant_LO,M_constant_HI,M_constant_PH);                        //age-indpendent: used only for MSST
  vector M_constant_out(1,8);
  number smsy2msst;                          //scales Smsy to get msst using (1-M). Used only in output.
  number smsy2msst75;                        //scales Smsy to get msst using 75%. Used only in output.

  matrix F(styr,endyr,1,nages);
  vector Fsum(styr,endyr);                   //Full fishing mortality rate by year
  vector Fapex(styr,endyr);                  //Max across ages, fishing mortality rate by year (may differ from Fsum bc of dome-shaped selex
  matrix Z(styr,endyr,1,nages);

  init_bounded_number log_avg_F_cH(log_avg_F_cH_LO,log_avg_F_cH_HI,log_avg_F_cH_PH);
  vector log_avg_F_cH_out(1,8);
  init_bounded_dev_vector log_F_dev_cH(styr_cH_L,endyr_cH_L,log_F_dev_cH_LO,log_F_dev_cH_HI,log_F_dev_cH_PH);
  vector log_F_dev_cH_out(styr_cH_L,endyr_cH_L);
  matrix F_cH(styr,endyr,1,nages);
  vector F_cH_out(styr,endyr);         //used for intermediate calculations in fcn get_mortality
  number log_F_dev_init_cH;
  number log_F_dev_end_cH;

  init_bounded_number log_avg_F_cD(log_avg_F_cD_LO,log_avg_F_cD_HI,log_avg_F_cD_PH);
  vector log_avg_F_cD_out(1,8);
  init_bounded_dev_vector log_F_dev_cD(styr_cD_L,endyr_cD_L,log_F_dev_cD_LO,log_F_dev_cD_HI,log_F_dev_cD_PH);
  vector log_F_dev_cD_out(styr_cD_L,endyr_cD_L);
  matrix F_cD(styr,endyr,1,nages);
  vector F_cD_out(styr,endyr);         //used for intermediate calculations in fcn get_mortality
  number log_F_dev_end_cD;

  init_bounded_number log_avg_F_HB(log_avg_F_HB_LO,log_avg_F_HB_HI,log_avg_F_HB_PH);
  vector log_avg_F_HB_out(1,8);
  init_bounded_dev_vector log_F_dev_HB(styr_HB_L,endyr_HB_L,log_F_dev_HB_LO,log_F_dev_HB_HI,log_F_dev_HB_PH);
  vector log_F_dev_HB_out(styr_HB_L,endyr_HB_L);
```

```
  matrix F_HB(styr,endyr,1,nages);
  vector F_HB_out(styr,endyr);  //used for intermediate calculations in fcn get_mortality
  number log_F_dev_init_HB;
  number log_F_dev_end_HB;

  init_bounded_number log_avg_F_GR(log_avg_F_GR_LO,log_avg_F_GR_HI,log_avg_F_GR_PH);
  vector log_avg_F_GR_out(1,8);
  init_bounded_dev_vector log_F_dev_GR(styr_GR_L,endyr_GR_L,log_F_dev_GR_LO,log_F_dev_GR_HI,log_F_dev_GR_PH);
  vector log_F_dev_GR_out(styr_GR_L,endyr_GR_L);
  matrix F_GR(styr,endyr,1,nages);
  vector F_GR_out(styr,endyr);  //used for intermediate calculations in fcn get_mortality
  number log_F_dev_init_GR;
  number log_F_dev_end_GR;

  init_bounded_number log_avg_F_cH_D(log_avg_F_cH_D_LO,log_avg_F_cH_D_HI,log_avg_F_cH_D_PH);
  vector log_avg_F_cH_D_out(1,8);
  init_bounded_dev_vector log_F_dev_cH_D(styr_cH_D,endyr_cH_D,log_F_dev_cH_D_LO,log_F_dev_cH_D_HI,log_F_dev_cH_D_PH);
  vector log_F_dev_cH_D_out(styr_cH_D,endyr_cH_D);
  matrix F_cH_D(styr,endyr,1,nages);
  vector F_cH_D_out(styr,endyr);         //used for intermediate calculations in fcn get_mortality
  number log_F_dev_end_cH_D;

  init_bounded_number log_avg_F_HB_D(log_avg_F_HB_D_LO,log_avg_F_HB_D_HI,log_avg_F_HB_D_PH);
  vector log_avg_F_HB_D_out(1,8);
  init_bounded_dev_vector log_F_dev_HB_D(styr_HB_D,endyr_HB_D,log_F_dev_HB_D_LO,log_F_dev_HB_D_HI,log_F_dev_HB_D_PH);
  vector log_F_dev_HB_D_out(styr_HB_D,endyr_HB_D);
  matrix F_HB_D(styr,endyr,1,nages);
  vector F_HB_D_out(styr,endyr);         //used for intermediate calculations in fcn get_mortality
  number log_F_dev_end_HB_D;

  init_bounded_number log_avg_F_GR_D(log_avg_F_GR_D_LO,log_avg_F_GR_D_HI,log_avg_F_GR_D_PH);
  vector log_avg_F_GR_D_out(1,8);
  init_bounded_dev_vector log_F_dev_GR_D(styr_GR_D,endyr_GR_D,log_F_dev_GR_D_LO,log_F_dev_GR_D_HI,log_F_dev_GR_D_PH);
  vector log_F_dev_GR_D_out(styr_GR_D,endyr_GR_D);
  matrix F_GR_D(styr,endyr,1,nages);
  vector F_GR_D_out(styr,endyr);         //used for intermediate calculations in fcn get_mortality
  number log_F_dev_end_GR_D;

  init_bounded_number F_init(F_init_LO,F_init_HI,F_init_PH); //scales early F for initialization
  vector F_init_out(1,8);
  number F_init_denom;   //interim calculation

////---Per-recruit stuff----------------------------------------------------------------------------------
  vector N_age_spr(1,nages);         //numbers at age for SPR calculations: beginning of year
  vector N_age_spr_spawn(1,nages);   //numbers at age for SPR calculations: time of peak spawning
  vector L_age_spr(1,nages);         //catch at age for SPR calculations
  vector Z_age_spr(1,nages);         //total mortality at age for SPR calculations
  vector spr_static(styr,endyr);     //vector of static SPR values by year
  vector F_L_age_spr(1,nages);       //fishing mortality of landings (not discards) at age for SPR calculations
  vector F_spr(1,n_iter_spr);        //values of full F to be used in per-recruit calculations
  vector spr_spr(1,n_iter_spr);      //reproductive capacity-per-recruit values corresponding to F values in F_spr
  vector L_spr(1,n_iter_spr);        //landings(lb gutted)-per-recruit (ypr) values corresponding to F values in F_spr

  vector N_spr_F0(1,nages);          //Used to compute spr at F=0: at time of peak spawning
  vector N_bpr_F0(1,nages);          //Used to compute bpr at F=0: at start of year
  vector N_spr_initial(1,nages);     //Initial spawners per recruit at age given initial F
  vector N_initial_eq(1,nages);      //Initial equilibrium abundance at age
  vector F_initial(1,nages);         //initial F at age
  vector Z_initial(1,nages);         //initial Z at age
  number spr_initial;                //initial spawners per recruit
  number spr_F0;                     //Spawning biomass per recruit at F=0
  number bpr_F0;                     //Biomass per recruit at F=0

  number iter_inc_spr;               //increments used to compute msy, equals max_F_spr_msy/(n_iter_spr-1)

////-------SDNR output-----------------------------------------------------------------------------
  number sdnr_lc_cH;
  number sdnr_lc_cD;
  number sdnr_lc_HB;

  number sdnr_ac_cH;
  number sdnr_ac_cD;
  number sdnr_ac_HB;

  number sdnr_I_cH;
  number sdnr_I_HB;
  number sdnr_I_GR;

////-------Objective function components-----------------------------------------------------------------------------
  number w_L;
  number w_D;

  number w_I_cH;
  number w_I_HB;
  number w_I_GR;

  number w_lc_cH;
  number w_lc_cD;
  number w_lc_HB;

  number w_ac_cH;
  number w_ac_cD;
  number w_ac_HB;

  number w_Nage_init;
  number w_rec;
  number w_rec_early;
  number w_rec_end;
  number w_fullF;
  number w_Ftune;

  number f_cH_L;
  number f_cD_L;
  number f_HB_L;
```

```
  number f_GR_L;

  number f_cH_D;
  number f_HB_D;
  number f_GR_D;

  number f_cH_cpue;
  number f_HB_cpue;
  number f_GR_cpue;

  number f_cH_lenc;
  number f_cD_lenc;
  number f_HB_lenc;

  number f_cH_agec;
  number f_cD_agec;
  number f_HB_agec;

// Penalties and constraints. Not all are used.
  number f_Nage_init;              //weight on log devs to estimate initial abundance (excluding first age)
  number f_rec_dev;                //weight on recruitment deviations to fit S-R curve
  number f_rec_dev_early;          //extra weight on deviations in first recruitment stanza
  number f_rec_dev_end;            //extra weight on deviations in ending recruitment stanza
  number f_fullF_constraint;       //penalty for Fapex>X
  number f_Ftune;                  //penalty for tuning F in Ftune yr.  Not applied in final optimization phase.
  number f_priors;                 //prior information on parameters

  objective_function_value fval;
  number fval_data;
  number grad_max;

//--Dummy variables ----
  number denom;                   //denominator used in some calculations
  number numer;                   //numerator used in some calculations

//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//INITIALIZATION_SECTION


//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
GLOBALS_SECTION
  #include "admodel.h"  // Include AD class definitions
  #include "admb2r.cpp" // Include R-compatible output functions. Comment line if not using admb2r.
  #include <time.h>
time_t start,finish;
long hour,minute,second;
double elapsed_time;

//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
RUNTIME_SECTION
 maximum_function_evaluations 1000, 2000,3000, 10000;
 convergence_criteria 1e-2, 1e-2,1e-3, 1e-4;

//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
PRELIMINARY_CALCS_SECTION

//Set values of fixed parameters or set initial guess of estimated parameters
  Dmort_cH=set_Dmort_cH;
  Dmort_HB=set_Dmort_HB;
  Dmort_GR=set_Dmort_GR;

//Dead discards
  obs_cH_D=Dmort_cH*obs_cH_released;
  obs_HB_D=Dmort_HB*obs_HB_released;
  obs_GR_D=Dmort_GR*obs_GR_released;

//Growth parameters
  Linf=set_Linf(1);
  K=set_K(1);
  t0=set_t0(1);
  len_cv_val=set_len_cv(1);

 //Maturity and sex transition
  maturity_f=maturity_f_obs;
  maturity_m=maturity_m_obs;
  prop_f=prop_f_obs;

//Natural mortality
  M=set_M;
  M_constant=set_M_constant(1);
  smsy2msst=1.0-M_constant;
  smsy2msst75=0.75;

//Recruitment
  log_R0=set_log_R0(1);
  steep=set_steep(1);
  R_autocorr=set_R_autocorr(1);
  rec_sigma=set_rec_sigma(1);

//Catchability
  log_q_cH=set_log_q_cH(1);
  log_q_HB=set_log_q_HB(1);
  log_q_GR=set_log_q_GR(1);

  q_rate=set_q_rate;
  q_rate_fcn_cH=1.0;
  q_rate_fcn_HB=1.0;
  q_rate_fcn_GR=1.0;
  q_DD_beta=set_q_DD_beta;
  q_DD_fcn=1.0;
```

```
  q_RW_log_dev_cH.initialize();
  q_RW_log_dev_HB.initialize();
  q_RW_log_dev_GR.initialize();

  if (set_q_rate_phase<0 & q_rate!=0.0)
  {
    for (iyear=styr_cH_cpue; iyear<=endyr_cH_cpue; iyear++)
      {   if (iyear>styr_cH_cpue & iyear <=2003)
          {//q_rate_fcn_cH(iyear)=(1.0+q_rate)*q_rate_fcn_cH(iyear-1); //compound
             q_rate_fcn_cH(iyear)=(1.0+(iyear-styr_cH_cpue)*q_rate)*q_rate_fcn_cH(styr_cH_cpue);  //linear
          }
          if (iyear>2003) {q_rate_fcn_cH(iyear)=q_rate_fcn_cH(iyear-1);}
      }
    for (iyear=styr_HB_cpue; iyear<=endyr_HB_cpue; iyear++)
      {   if (iyear>styr_HB_cpue & iyear <=2003)
          {//q_rate_fcn_HB(iyear)=(1.0+q_rate)*q_rate_fcn_HB(iyear-1); //compound
             q_rate_fcn_HB(iyear)=(1.0+(iyear-styr_HB_cpue)*q_rate)*q_rate_fcn_HB(styr_HB_cpue);  //linear
          }
          if (iyear>2003) {q_rate_fcn_HB(iyear)=q_rate_fcn_HB(iyear-1);}
      }
    for (iyear=styr_GR_cpue; iyear<=endyr_GR_cpue; iyear++)
      {   if (iyear>styr_GR_cpue & iyear <=2003)
          {//q_rate_fcn_GR(iyear)=(1.0+q_rate)*q_rate_fcn_GR(iyear-1); //compound
             q_rate_fcn_GR(iyear)=(1.0+(iyear-styr_GR_cpue)*q_rate)*q_rate_fcn_GR(styr_GR_cpue);  //linear
          }
          if (iyear>2003) {q_rate_fcn_GR(iyear)=q_rate_fcn_GR(iyear-1);}
      }
  } //end q_rate conditional

//Objective function weights
  w_L=set_w_L;
  w_D=set_w_D;

  w_I_cH=set_w_I_cH;
  w_I_HB=set_w_I_HB;
  w_I_GR=set_w_I_GR;

  w_lc_cH=set_w_lc_cH;
  w_lc_cD=set_w_lc_cD;
  w_lc_HB=set_w_lc_HB;

  w_ac_cH=set_w_ac_cH;
  w_ac_cD=set_w_ac_cD;
  w_ac_HB=set_w_ac_HB;

  w_Nage_init=set_w_Nage_init;
  w_rec=set_w_rec;
  w_rec_early=set_w_rec_early;
  w_rec_end=set_w_rec_end;
  w_fullF=set_w_fullF;
  w_Ftune=set_w_Ftune;

//Fishing rates
  F_init=set_F_init(1);

  log_avg_F_cH=set_log_avg_F_cH(1);
  log_avg_F_cD=set_log_avg_F_cD(1);
  log_avg_F_HB=set_log_avg_F_HB(1);
  log_avg_F_GR=set_log_avg_F_GR(1);
  log_avg_F_cH_D=set_log_avg_F_cH_D(1);
  log_avg_F_HB_D=set_log_avg_F_HB_D(1);
  log_avg_F_GR_D=set_log_avg_F_GR_D(1);

  log_F_dev_cH=set_log_F_dev_cH_vals;
  log_F_dev_cD=set_log_F_dev_cD_vals;
  log_F_dev_HB=set_log_F_dev_HB_vals;
  log_F_dev_GR=set_log_F_dev_GR_vals;
  log_F_dev_cH_D=set_log_F_dev_cH_D_vals;
  log_F_dev_HB_D=set_log_F_dev_HB_D_vals;
  log_F_dev_GR_D=set_log_F_dev_GR_D_vals;

 //Selectivity parameters
  selpar_L50_cH1=set_selpar_L50_cH1(1);
  selpar_slope_cH1=set_selpar_slope_cH1(1);
  selpar_L50_cH2=set_selpar_L50_cH2(1);
  selpar_slope_cH2=set_selpar_slope_cH2(1);
  selpar_L50_cH3=set_selpar_L50_cH3(1);
  selpar_slope_cH3=set_selpar_slope_cH3(1);

  selpar_L50_cD=set_selpar_L50_cD(1);
  selpar_slope_cD=set_selpar_slope_cD(1);
  selpar_afull_cD=set_selpar_afull_cD(1);
  selpar_sigma_cD=set_selpar_sigma_cD(1);

  selpar_L50_HB1=set_selpar_L50_HB1(1);
  selpar_slope_HB1=set_selpar_slope_HB1(1);
  selpar_L50_HB2=set_selpar_L50_HB2(1);
  selpar_slope_HB2=set_selpar_slope_HB2(1);
  selpar_L50_HB3=set_selpar_L50_HB3(1);
  selpar_slope_HB3=set_selpar_slope_HB3(1);

//Conversions and fixed quantities
  sqrt2pi=sqrt(2.*3.14159265);
  g2mt=0.000001;         //conversion of grams to metric tons
  g2kg=0.001;            //conversion of grams to kg
  mt2klb=2.20462;        //conversion of metric tons to 1000 lb
  mt2lb=mt2klb*1000.0;   //conversion of metric tons to lb
  g2klb=g2mt*mt2klb;     //conversion of grams to 1000 lb
  dzero=0.00001;
  huge_number=1.0e+10;

  iter_inc_msy=max_F_spr_msy/(n_iter_msy-1);
  iter_inc_spr=max_F_spr_msy/(n_iter_spr-1);
```

```
//Fill in sample sizes of comps, possibly sampled in nonconsec yrs
//Used primarily for output in R object

      nsamp_cH_lenc_allyr=missing;
      nsamp_cD_lenc_allyr=missing;
      nsamp_HB_lenc_allyr=missing;
      nsamp_cH_agec_allyr=missing;
      nsamp_cD_agec_allyr=missing;
      nsamp_HB_agec_allyr=missing;

      nfish_cH_lenc_allyr=missing;
      nfish_cD_lenc_allyr=missing;
      nfish_HB_lenc_allyr=missing;
      nfish_cH_agec_allyr=missing;
      nfish_cD_agec_allyr=missing;
      nfish_HB_agec_allyr=missing;

      for (iyear=1; iyear<=nyr_cH_lenc; iyear++)
         {if (nsamp_cH_lenc(iyear)>=minSS_cH_lenc)
           {nsamp_cH_lenc_allyr(yrs_cH_lenc(iyear))=nsamp_cH_lenc(iyear);
            nfish_cH_lenc_allyr(yrs_cH_lenc(iyear))=nfish_cH_lenc(iyear);}}

      for (iyear=1; iyear<=nyr_cD_lenc; iyear++)
         {if (nsamp_cD_lenc(iyear)>=minSS_cD_lenc)
           {nsamp_cD_lenc_allyr(yrs_cD_lenc(iyear))=nsamp_cD_lenc(iyear);
            nfish_cD_lenc_allyr(yrs_cD_lenc(iyear))=nfish_cD_lenc(iyear);}}

      for (iyear=1; iyear<=nyr_HB_lenc; iyear++)
         {if (nsamp_HB_lenc(iyear)>=minSS_HB_lenc)
            {nsamp_HB_lenc_allyr(yrs_HB_lenc(iyear))=nsamp_HB_lenc(iyear);
             nfish_HB_lenc_allyr(yrs_HB_lenc(iyear))=nfish_HB_lenc(iyear);}}

      for (iyear=1; iyear<=nyr_cH_agec; iyear++)
         {if (nsamp_cH_agec(iyear)>=minSS_cH_agec)
           {nsamp_cH_agec_allyr(yrs_cH_agec(iyear))=nsamp_cH_agec(iyear);
            nfish_cH_agec_allyr(yrs_cH_agec(iyear))=nfish_cH_agec(iyear);}}

      for (iyear=1; iyear<=nyr_cD_agec; iyear++)
         {if (nsamp_cD_agec(iyear)>=minSS_cD_agec)
           {nsamp_cD_agec_allyr(yrs_cD_agec(iyear))=nsamp_cD_agec(iyear);
            nfish_cD_agec_allyr(yrs_cD_agec(iyear))=nfish_cD_agec(iyear);}}

      for (iyear=1; iyear<=nyr_HB_agec; iyear++)
         {if (nsamp_HB_agec(iyear)>=minSS_HB_agec)
           {nsamp_HB_agec_allyr(yrs_HB_agec(iyear))=nsamp_HB_agec(iyear);
            nfish_HB_agec_allyr(yrs_HB_agec(iyear))=nfish_HB_agec(iyear);}}

//fill in Fs for msy and per-recruit analyses
  F_msy(1)=0.0;
  for (ff=2;ff<=n_iter_msy;ff++) {F_msy(ff)=F_msy(ff-1)+iter_inc_msy;}
  F_spr(1)=0.0;
  for (ff=2;ff<=n_iter_spr;ff++) {F_spr(ff)=F_spr(ff-1)+iter_inc_spr;}


//fill in F's, Catch matrices, and log rec dev with zero's
  F_cH.initialize(); L_cH_num.initialize();
  F_cD.initialize(); L_cD_num.initialize();
  F_HB.initialize(); L_HB_num.initialize();
  F_GR.initialize(); L_GR_num.initialize();
  F_cH_D.initialize(); D_cH_num.initialize();
  F_HB_D.initialize(); D_HB_num.initialize();
  F_GR_D.initialize(); D_GR_num.initialize();

  F_cH_out.initialize();
  F_cD_out.initialize();
  F_HB_out.initialize();
  F_GR_out.initialize();
  F_cH_D_out.initialize();
  F_HB_D_out.initialize();
  F_GR_D_out.initialize();


  sel_cH.initialize();
  sel_cD.initialize();
  sel_HB.initialize();
  sel_cH_D.initialize();
  sel_HB_D.initialize();

  log_rec_dev_output.initialize();
  log_rec_dev=set_log_rec_dev_vals;
  log_Nage_dev_output.initialize();
  log_Nage_dev=set_log_Nage_dev_vals;


//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
TOP_OF_MAIN_SECTION
  time(&start);
  arrmblsize=20000000;
  gradient_structure::set_MAX_NVAR_OFFSET(1600);
  gradient_structure::set_GRADSTACK_BUFFER_SIZE(2000000);
  gradient_structure::set_CMPDIF_BUFFER_SIZE(2000000);
  gradient_structure::set_NUM_DEPENDENT_VARIABLES(10000);

//>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
PROCEDURE_SECTION

 get_length_weight_at_age();
 get_reprod();
 get_length_at_age_dist();
 get_weight_at_age_landings();
 get_spr_F0();
```

```
 get_selectivity();
 get_mortality();
 get_bias_corr();
 get_numbers_at_age();
 get_landings_numbers();
 get_landings_wgt();
 get_dead_discards();
 get_catchability_fcns();
 get_indices();
 get_length_comps();
 get_age_comps();
 evaluate_objective_function();


FUNCTION get_length_weight_at_age
  //compute mean length (mm TL) and weight (whole) at age
    meanlen_TL=Linf*(1.0-mfexp(-K*(agebins-t0+0.5)));     //total length in mm
    wgt_kg=wgtpar_a*pow(meanlen_TL,wgtpar_b);             //whole wgt in kg
    wgt_g=wgt_kg/g2kg;                                    //convert wgt in kg to weight in g
    wgt_mt=wgt_g*g2mt;                                    //convert weight in g to weight in mt
    wgt_klb=mt2klb*wgt_mt;                                //1000 lb of whole wgt
    wgt_lb=mt2lb*wgt_mt;                                  //lb of whole wgt
    wgt_klb_gut=ww2gw*wgt_klb;                            //1000 lb of gutted wgt
    wgt_lb_gut=ww2gw*wgt_lb;                              //lb of gutted wgt


FUNCTION get_reprod
   //reprod is the product of stuff going into reproductive capacity calcs
  for (iyear=styr; iyear<=endyr; iyear++)
  {
   reprod(iyear)=elem_prod((elem_prod(prop_f(iyear),maturity_f)+elem_prod((prop_m(iyear)),maturity_m)),wgt_mt);  //both sexes
   reprod2(iyear)=elem_prod(elem_prod(prop_f(iyear),maturity_f),wgt_mt);  //females only
  }

FUNCTION get_length_at_age_dist
  //compute matrix of length at age, based on the normal distribution

  for (iage=1;iage<=nages;iage++)
  {
    len_cv(iage)=len_cv_val;
    len_sd(iage)=meanlen_TL(iage)*len_cv(iage);

    zscore_lzero=(0.0-meanlen_TL(iage))/len_sd(iage);
    cprob_lzero=cumd_norm(zscore_lzero);

    //first length bin
    zscore_len=((lenbins(1)+0.5*lenbins_width)-meanlen_TL(iage)) / len_sd(iage);
    cprob_lenvec(1)=cumd_norm(zscore_len);          //includes any probability mass below zero
    lenprob(iage,1)=cprob_lenvec(1)-cprob_lzero;    //removes any probability mass below zero

    //most other length bins
    for (ilen=2;ilen<nlenbins;ilen++)
      {
        zscore_len=((lenbins(ilen)+0.5*lenbins_width)-meanlen_TL(iage)) / len_sd(iage);
        cprob_lenvec(ilen)=cumd_norm(zscore_len);
        lenprob(iage,ilen)=cprob_lenvec(ilen)-cprob_lenvec(ilen-1);
      }
    //last length bin is a plus group
    zscore_len=((lenbins(nlenbins)-0.5*lenbins_width)-meanlen_TL(iage)) / len_sd(iage);
    lenprob(iage,nlenbins)=1.0-cumd_norm(zscore_len);

    lenprob(iage)=lenprob(iage)/(1.0-cprob_lzero);  //renormalize to account for any prob mass below size=0
  }
  //fleet and survey specific length probs, all assumed here to equal the popn
  lenprob_cH=lenprob;
  lenprob_cD=lenprob;
  lenprob_HB=lenprob;


FUNCTION get_weight_at_age_landings

  for (iyear=styr; iyear<=endyr; iyear++)
  {
    len_cH_mm(iyear)=meanlen_TL;
    gutwgt_cH_klb(iyear)=wgt_klb_gut; wholewgt_cH_klb(iyear)=wgt_klb; //whole weight used to match index
    len_cD_mm(iyear)=meanlen_TL;
    gutwgt_cD_klb(iyear)=wgt_klb_gut;
    len_HB_mm(iyear)=meanlen_TL;
    gutwgt_HB_klb(iyear)=wgt_klb_gut;
    len_GR_mm(iyear)=meanlen_TL;
    gutwgt_GR_klb(iyear)=wgt_klb_gut;

    len_cH_D_mm(iyear)=meanlen_TL;
    gutwgt_cH_D_klb(iyear)=wgt_klb_gut;
    len_HB_D_mm(iyear)=meanlen_TL;
    gutwgt_HB_D_klb(iyear)=wgt_klb_gut;
    len_GR_D_mm(iyear)=meanlen_TL;
    gutwgt_GR_D_klb(iyear)=wgt_klb_gut;
  }

FUNCTION get_spr_F0
  //at mdyr, apply half this yr's natural mortality, half next yr's
  N_spr_F0(1)=1.0*mfexp(-1.0*M(1)*spawn_time_frac); //at peak spawning time
  N_bpr_F0(1)=1.0;       //at start of year
  for (iage=2; iage<=nages; iage++)
  {
    N_spr_F0(iage)=N_spr_F0(iage-1)*mfexp(-1.0*(M(iage-1)*(1.0-spawn_time_frac) + M(iage)*spawn_time_frac));
    N_bpr_F0(iage)=N_bpr_F0(iage-1)*mfexp(-1.0*(M(iage-1)));
  }
  N_spr_F0(nages)=N_spr_F0(nages)/(1.0-mfexp(-1.0*M(nages))); //plus group (sum of geometric series)
  N_bpr_F0(nages)=N_bpr_F0(nages)/(1.0-mfexp(-1.0*M(nages)));

  spr_F0=sum(elem_prod(N_spr_F0,reprod(endyr)));
  bpr_F0=sum(elem_prod(N_bpr_F0,wgt_mt));
```

```
FUNCTION get_selectivity

  //BLOCK 1 for selex.
  for (iyear=styr; iyear<=endyr_selex_phase1; iyear++)
  {
     sel_cH(iyear)=logistic(agebins, selpar_L50_cH1, selpar_slope_cH1);
     sel_cD(iyear)=logistic_exponential(agebins, selpar_L50_cD, selpar_slope_cD, selpar_sigma_cD, selpar_afull_cD);
     sel_HB(iyear)=logistic(agebins, selpar_L50_HB1, selpar_slope_HB1);

     sel_cD(iyear)(13,nages)=sel_cD(iyear)(12);
  }

  //BLOCK 2 for selex.
  for (iyear=(endyr_selex_phase1+1); iyear<=endyr_selex_phase2; iyear++)
   {
     sel_cH(iyear)=logistic(agebins, selpar_L50_cH2, selpar_slope_cH2);
     sel_cD(iyear)=logistic_exponential(agebins, selpar_L50_cD, selpar_slope_cD, selpar_sigma_cD, selpar_afull_cD);
     sel_HB(iyear)=logistic(agebins, selpar_L50_HB2, selpar_slope_HB2);

     sel_cD(iyear)(13,nages)=sel_cD(iyear)(12);

   }

  //BLOCK 3 for selex.
  for (iyear=(endyr_selex_phase2+1); iyear<=endyr; iyear++)
   {
     sel_cH(iyear)=logistic(agebins, selpar_L50_cH3, selpar_slope_cH3);
     sel_cD(iyear)=logistic_exponential(agebins, selpar_L50_cD, selpar_slope_cD, selpar_sigma_cD, selpar_afull_cD);
     sel_HB(iyear)=logistic(agebins, selpar_L50_HB3, selpar_slope_HB3);

     sel_cD(iyear)(13,nages)=sel_cD(iyear)(12);
    }

   //Discard selex, uses a 2 age shift
   for (iyear=styr_HB_D;iyear<=endyr_HB_D;iyear++)
   {
      for (iage=1; iage<=(nages-2); iage++)
      {
        sel_HB_D(iyear,iage)=(sel_HB(endyr,iage+2)-sel_HB(endyr,iage));
        if(sel_HB_D(iyear,iage)<0.0) {sel_HB_D(iyear,iage)=0.0;}
      }
      sel_HB_D(iyear,(nages-1))=0.0;
      sel_HB_D(iyear,nages)=0.0;
      sel_HB_D(iyear)=sel_HB_D(iyear)/max(sel_HB_D(iyear));
   }

   for (iyear=styr_cH_D;iyear<=endyr_cH_D;iyear++)
   {
      for (iage=1; iage<=(nages-2); iage++)
      {
        sel_cH_D(iyear,iage)=(sel_cH(endyr,iage+2)-sel_cH(endyr,iage));
        if(sel_cH_D(iyear,iage)<0.0) {sel_cH_D(iyear,iage)=0.0;}
      }
      sel_cH_D(iyear,(nages-1))=0.0;
      sel_cH_D(iyear,nages)=0.0;
      sel_cH_D(iyear)=sel_cH_D(iyear)/max(sel_cH_D(iyear));
   }


FUNCTION get_mortality
  Fsum.initialize();
  Fapex.initialize();
  F.initialize();
  //initialization F is avg from first 3 yrs of observed landings: cD excluded bc it begins later than styr
  log_F_dev_init_cH=sum(log_F_dev_cH(styr_cH_L,(styr_cH_L+2)))/3.0;
  log_F_dev_init_HB=sum(log_F_dev_HB(styr_HB_L,(styr_HB_L+2)))/3.0;
  log_F_dev_init_GR=sum(log_F_dev_GR(styr_GR_L,(styr_GR_L+2)))/3.0;

  for (iyear=styr; iyear<=endyr; iyear++)
  {
    if(iyear>=styr_cH_L & iyear<=endyr_cH_L)
    {  F_cH_out(iyear)=mfexp(log_avg_F_cH+log_F_dev_cH(iyear));
       F_cH(iyear)=sel_cH(iyear)*F_cH_out(iyear);
       Fsum(iyear)+=F_cH_out(iyear);
    }

    if(iyear>=styr_cD_L & iyear<=endyr_cD_L)
    {  F_cD_out(iyear)=mfexp(log_avg_F_cD+log_F_dev_cD(iyear));
       F_cD(iyear)=sel_cD(iyear)*F_cD_out(iyear);
       Fsum(iyear)+=F_cD_out(iyear);
    }

    if(iyear>=styr_HB_L & iyear<=endyr_HB_L)
    {  F_HB_out(iyear)=mfexp(log_avg_F_HB+log_F_dev_HB(iyear));
       F_HB(iyear)=sel_HB(iyear)*F_HB_out(iyear);
       Fsum(iyear)+=F_HB_out(iyear);
    }

    if(iyear>=styr_GR_L & iyear<=endyr_GR_L)
    {  F_GR_out(iyear)=mfexp(log_avg_F_GR+log_F_dev_GR(iyear));
       F_GR(iyear)=sel_HB(iyear)*F_GR_out(iyear); //general rec shares headboat selex
       Fsum(iyear)+=F_GR_out(iyear);
    }

    if(iyear>=styr_cH_D & iyear<=endyr_cH_D)
    {  F_cH_D_out(iyear)=mfexp(log_avg_F_cH_D+log_F_dev_cH_D(iyear));
       F_cH_D(iyear)=sel_cH_D(iyear)*F_cH_D_out(iyear);
       Fsum(iyear)+=F_cH_D_out(iyear);
    }

    if(iyear>=styr_HB_D & iyear<=endyr_HB_D)
    {  F_HB_D_out(iyear)=mfexp(log_avg_F_HB_D+log_F_dev_HB_D(iyear));
       F_HB_D(iyear)=sel_HB_D(iyear)*F_HB_D_out(iyear);
```

```
      Fsum(iyear)+=F_HB_D_out(iyear);
    }

    if(iyear>=styr_GR_D & iyear<=endyr_GR_D)
    {  F_GR_D_out(iyear)=mfexp(log_avg_F_GR_D+log_F_dev_GR_D(iyear));
       F_GR_D(iyear)=sel_HB_D(iyear)*F_GR_D_out(iyear); //general rec shares headboat selex
       Fsum(iyear)+=F_GR_D_out(iyear);
    }

    //Total F at age
    F(iyear)=F_cH(iyear);  //first in additive series (NO +=)
    F(iyear)+=F_cD(iyear);
    F(iyear)+=F_HB(iyear);
    F(iyear)+=F_GR(iyear);
    F(iyear)+=F_cH_D(iyear);
    F(iyear)+=F_HB_D(iyear);
    F(iyear)+=F_GR_D(iyear);

    Fapex(iyear)=max(F(iyear));
    Z(iyear)=M+F(iyear);

  }  //end iyear


FUNCTION get_bias_corr
  var_rec_dev=norm2(log_rec_dev(styr_rec_dev,endyr_rec_dev)-
              sum(log_rec_dev(styr_rec_dev,endyr_rec_dev))/nyrs_rec)
              /(nyrs_rec-1.0);
  rec_sigma_sq=square(rec_sigma);
  if (set_BiasCor <= 0.0) {BiasCor=mfexp(rec_sigma_sq/2.0);}   //bias correction based on Rsigma
  else {BiasCor=set_BiasCor;}

FUNCTION get_numbers_at_age
//Initialization
  R0=mfexp(log_R0);
  S0=spr_F0*R0;

  R_virgin=SR_eq_func(R0, steep, spr_F0, spr_F0, BiasCor, SR_switch);

  B0=bpr_F0*R_virgin;
  B0_q_DD=R_virgin*sum(elem_prod(N_bpr_F0(set_q_DD_stage,nages),wgt_mt(set_q_DD_stage,nages)));

  F_init_denom=mfexp(log_avg_F_cH+log_F_dev_init_cH)+mfexp(log_avg_F_HB+log_F_dev_init_HB)+mfexp(log_avg_F_GR+log_F_dev_init_GR);
  F_init_cH_prop=mfexp(log_avg_F_cH+log_F_dev_init_cH)/F_init_denom;
  F_init_HB_prop=mfexp(log_avg_F_HB+log_F_dev_init_HB)/F_init_denom;
  F_init_GR_prop=mfexp(log_avg_F_GR+log_F_dev_init_GR)/F_init_denom;

  F_initial=sel_cH(styr)*F_init*F_init_cH_prop+
            sel_HB(styr)*F_init*F_init_HB_prop+
            sel_HB(styr)*F_init*F_init_GR_prop; //GR uses HB selex
  Z_initial=M+F_initial;

//Initial equilibrium age structure
  N_spr_initial(1)=1.0*mfexp(-1.0*Z_initial(1)*spawn_time_frac); //at peak spawning time;
  for (iage=2; iage<=nages; iage++)
    {
      N_spr_initial(iage)=N_spr_initial(iage-1)*
                   mfexp(-1.0*(Z_initial(iage-1)*(1.0-spawn_time_frac) + Z_initial(iage)*spawn_time_frac));
    }
  N_spr_initial(nages)=N_spr_initial(nages)/(1.0-mfexp(-1.0*Z_initial(nages))); //plus group
  spr_initial=sum(elem_prod(N_spr_initial,reprod(styr)));

  if (styr==styr_rec_dev) {R1=SR_eq_func(R0, steep, spr_F0, spr_initial, 1.0, SR_switch);} //without bias correction (deviation added later)
  else {R1=SR_eq_func(R0, steep, spr_F0, spr_initial, BiasCor, SR_switch);} //with bias correction
  if(R1<10.0) {R1=10.0;} //Avoid unrealistically low popn sizes during optimization procedure

//Compute equilibrium age structure for first year
  N_initial_eq(1)=R1;
  for (iage=2; iage<=nages; iage++)
  {
    N_initial_eq(iage)=N_initial_eq(iage-1)*
        mfexp(-1.0*(Z_initial(iage-1)));
  }
  //plus group calculation
  N_initial_eq(nages)=N_initial_eq(nages)/(1.0-mfexp(-1.0*Z_initial(nages)));

//Add deviations to initial equilibrium N
  N(styr)(2,nages)=elem_prod(N_initial_eq(2,nages),mfexp(log_Nage_dev));

  if (styr==styr_rec_dev) {N(styr,1)=N_initial_eq(1)*mfexp(log_rec_dev(styr_rec_dev));}
  else {N(styr,1)=N_initial_eq(1);}

  N_mdyr(styr)(1,nages)=elem_prod(N(styr)(1,nages),(mfexp(-1.*(Z_initial(1,nages))*0.5))); //mid year
  N_spawn(styr)(1,nages)=elem_prod(N(styr)(1,nages),(mfexp(-1.*(Z_initial(1,nages))*spawn_time_frac))); //peak spawning time

  SSB(styr)=sum(elem_prod(N_spawn(styr),reprod(styr)));
  MatFemB(styr)=sum(elem_prod(N_spawn(styr),reprod2(styr)));
  B_q_DD(styr)=sum(elem_prod(N(styr)(set_q_DD_stage,nages),wgt_mt(set_q_DD_stage,nages)));

//Rest of years
  for (iyear=styr; iyear<endyr; iyear++)
  {
    if(iyear<(styr_rec_dev-1)||iyear>(endyr_rec_dev-1)) //recruitment follows S-R curve (with bias correction) exactly
    {
        N(iyear+1,1)=BiasCor*SR_func(R0, steep, spr_F0, SSB(iyear),SR_switch);
        N(iyear+1)(2,nages)=++elem_prod(N(iyear)(1,nages-1),(mfexp(-1.*Z(iyear)(1,nages-1))));
        N(iyear+1,nages)+=N(iyear,nages)*mfexp(-1.*Z(iyear,nages)); //plus group
        N_mdyr(iyear+1)(1,nages)=elem_prod(N(iyear+1)(1,nages),(mfexp(-1.*(Z(iyear+1)(1,nages))*0.5))); //mid year
        N_spawn(iyear+1)(1,nages)=elem_prod(N(iyear+1)(1,nages),(mfexp(-1.*(Z(iyear+1)(1,nages))*spawn_time_frac))); //peak spawning time
        SSB(iyear+1)=sum(elem_prod(N_spawn(iyear+1),reprod(iyear+1)));
        MatFemB(iyear+1)=sum(elem_prod(N_spawn(iyear+1),reprod2(iyear+1)));
        B_q_DD(iyear+1)=sum(elem_prod(N(iyear+1)(set_q_DD_stage,nages),wgt_mt(set_q_DD_stage,nages)));
    }
    else   //recruitment follows S-R curve with lognormal deviation
```

```
    {
        N(iyear+1,1)=SR_func(R0, steep, spr_F0, SSB(iyear),SR_switch)*mfexp(log_rec_dev(iyear+1));
        N(iyear+1)(2,nages)=++elem_prod(N(iyear)(1,nages-1),(mfexp(-1.*Z(iyear)(1,nages-1))));
        N(iyear+1,nages)+=N(iyear,nages)*mfexp(-1.*Z(iyear,nages)); //plus group
        N_mdyr(iyear+1)(1,nages)=elem_prod(N(iyear+1)(1,nages),(mfexp(-1.*(Z(iyear+1)(1,nages))*0.5))); //mid year
        N_spawn(iyear+1)(1,nages)=elem_prod(N(iyear+1)(1,nages),(mfexp(-1.*(Z(iyear+1)(1,nages))*spawn_time_frac))); //peak spawning time
        SSB(iyear+1)=sum(elem_prod(N_spawn(iyear+1),reprod(iyear+1)));
        MatFemB(iyear+1)=sum(elem_prod(N_spawn(iyear+1),reprod2(iyear+1)));
        B_q_DD(iyear+1)=sum(elem_prod(N(iyear+1)(set_q_DD_stage,nages),wgt_mt(set_q_DD_stage,nages)));
    }
  }

  //last year (projection) has no recruitment variability
  N(endyr+1,1)=BiasCor*SR_func(R0, steep, spr_F0, SSB(endyr),SR_switch);
  N(endyr+1)(2,nages)=++elem_prod(N(endyr)(1,nages-1),(mfexp(-1.*Z(endyr)(1,nages-1))));
  N(endyr+1,nages)+=N(endyr,nages)*mfexp(-1.*Z(endyr,nages)); //plus group


FUNCTION get_landings_numbers //Baranov catch eqn
  for (iyear=styr; iyear<=endyr; iyear++)
  {
    for (iage=1; iage<=nages; iage++)
    {
      L_cH_num(iyear,iage)=N(iyear,iage)*F_cH(iyear,iage)*
        (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
      L_cD_num(iyear,iage)=N(iyear,iage)*F_cD(iyear,iage)*
        (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
      L_HB_num(iyear,iage)=N(iyear,iage)*F_HB(iyear,iage)*
        (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
      L_GR_num(iyear,iage)=N(iyear,iage)*F_GR(iyear,iage)*
        (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
    }
    pred_cH_L_knum(iyear)=sum(L_cH_num(iyear))/1000.0;
    pred_cD_L_knum(iyear)=sum(L_cD_num(iyear))/1000.0;
    pred_HB_L_knum(iyear)=sum(L_HB_num(iyear))/1000.0;
    pred_GR_L_knum(iyear)=sum(L_GR_num(iyear))/1000.0;
  }


FUNCTION get_landings_wgt
  for (iyear=styr; iyear<=endyr; iyear++)
  {
    L_cH_klb(iyear)=elem_prod(L_cH_num(iyear),gutwgt_cH_klb(iyear));     //in 1000 lb gutted weight
    L_cD_klb(iyear)=elem_prod(L_cD_num(iyear),gutwgt_cD_klb(iyear));     //in 1000 lb gutted weight
    L_HB_klb(iyear)=elem_prod(L_HB_num(iyear),gutwgt_HB_klb(iyear));     //in 1000 lb gutted weight
    L_GR_klb(iyear)=elem_prod(L_GR_num(iyear),gutwgt_GR_klb(iyear));     //in 1000 lb gutted weight

    pred_cH_L_klb(iyear)=sum(L_cH_klb(iyear));
    pred_cD_L_klb(iyear)=sum(L_cD_klb(iyear));
    pred_HB_L_klb(iyear)=sum(L_HB_klb(iyear));
    pred_GR_L_klb(iyear)=sum(L_GR_klb(iyear));
  }

FUNCTION get_dead_discards
  //dead discards at age (number fish)
  for (iyear=styr_cH_D; iyear<=endyr_cH_D; iyear++)
  {
    for (iage=1; iage<=nages; iage++)
    {
      D_cH_num(iyear,iage)=N(iyear,iage)*F_cH_D(iyear,iage)*
        (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
    }
    pred_cH_D_knum(iyear)=sum(D_cH_num(iyear))/1000.0;            //pred annual dead discards in 1000s (for matching data)
    pred_cH_D_klb(iyear)=sum(elem_prod(D_cH_num(iyear),gutwgt_cH_D_klb(iyear))); //annual dead discards in 1000 lb gutted (for output only)
  }

  for (iyear=styr_HB_D; iyear<=endyr_HB_D; iyear++)
  {
    for (iage=1; iage<=nages; iage++)
    {
      D_HB_num(iyear,iage)=N(iyear,iage)*F_HB_D(iyear,iage)*
        (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
    }
    pred_HB_D_knum(iyear)=sum(D_HB_num(iyear))/1000.0;            //pred annual dead discards in 1000s (for matHBing data)
    pred_HB_D_klb(iyear)=sum(elem_prod(D_HB_num(iyear),gutwgt_HB_D_klb(iyear))); //annual dead discards in 1000 lb gutted (for output only)
  }

  for (iyear=styr_GR_D; iyear<=endyr_GR_D; iyear++)
  {
    for (iage=1; iage<=nages; iage++)
    {
      D_GR_num(iyear,iage)=N(iyear,iage)*F_GR_D(iyear,iage)*
        (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
    }
    pred_GR_D_knum(iyear)=sum(D_GR_num(iyear))/1000.0;            //pred annual dead discards in 1000s (for matGRing data)
    pred_GR_D_klb(iyear)=sum(elem_prod(D_GR_num(iyear),gutwgt_GR_D_klb(iyear))); //annual dead discards in 1000 lb gutted (for output only)
  }

FUNCTION get_catchability_fcns
 //Get rate increase if estimated, otherwise fixed above
  if (set_q_rate_phase>0.0)
  {

      for (iyear=styr_cH_cpue; iyear<=endyr_cH_cpue; iyear++)
      {   if (iyear>styr_cH_cpue & iyear <=2003)
          {//q_rate_fcn_cH(iyear)=(1.0+q_rate)*q_rate_fcn_cH(iyear-1); //compound
             q_rate_fcn_cH(iyear)=(1.0+(iyear-styr_cH_cpue)*q_rate)*q_rate_fcn_cH(styr_cH_cpue);  //linear
          }
          if (iyear>2003) {q_rate_fcn_cH(iyear)=q_rate_fcn_cH(iyear-1);}
      }
      for (iyear=styr_HB_cpue; iyear<=endyr_HB_cpue; iyear++)
      {   if (iyear>styr_HB_cpue & iyear <=2003)
          {//q_rate_fcn_HB(iyear)=(1.0+q_rate)*q_rate_fcn_HB(iyear-1); //compound
             q_rate_fcn_HB(iyear)=(1.0+(iyear-styr_HB_cpue)*q_rate)*q_rate_fcn_HB(styr_HB_cpue);  //linear
```

```
        }
        if (iyear>2003) {q_rate_fcn_HB(iyear)=q_rate_fcn_HB(iyear-1);}
    }
    for (iyear=styr_GR_cpue; iyear<=endyr_GR_cpue; iyear++)
    {   if (iyear>styr_GR_cpue & iyear <=2003)
        {//q_rate_fcn_GR(iyear)=(1.0+q_rate)*q_rate_fcn_GR(iyear-1); //compound
           q_rate_fcn_GR(iyear)=(1.0+(iyear-styr_GR_cpue)*q_rate)*q_rate_fcn_GR(styr_GR_cpue);  //linear
        }
        if (iyear>2003) {q_rate_fcn_GR(iyear)=q_rate_fcn_GR(iyear-1);}
    }

  } //end q_rate conditional

 //Get density dependence scalar (=1.0 if density independent model is used)
  if (q_DD_beta>0.0)
  {
    B_q_DD+=dzero;
    for (iyear=styr;iyear<=endyr;iyear++)
        {q_DD_fcn(iyear)=pow(B0_q_DD,q_DD_beta)*pow(B_q_DD(iyear),-q_DD_beta);}
  }


FUNCTION get_indices
//---Predicted CPUEs------------------------

 //cH  cpue
  q_cH(styr_cH_cpue)=mfexp(log_q_cH);
  for (iyear=styr_cH_cpue; iyear<=endyr_cH_cpue; iyear++)
  {//index in weight units. original index in lb and re-scaled. predicted in klb whole weight, but difference in lb and klb is absorbed by q
      N_cH(iyear)=elem_prod(elem_prod(N_mdyr(iyear),sel_cH(iyear)),wholewgt_cH_klb(iyear));
      pred_cH_cpue(iyear)=q_cH(iyear)*q_rate_fcn_cH(iyear)*q_DD_fcn(iyear)*sum(N_cH(iyear));
      if (iyear<endyr_cH_cpue){q_cH(iyear+1)=q_cH(iyear)*mfexp(q_RW_log_dev_cH(iyear));}
  }

 //HB  cpue
  q_HB(styr_HB_cpue)=mfexp(log_q_HB);
  for (iyear=styr_HB_cpue; iyear<=endyr_HB_cpue; iyear++)
  {
      N_HB(iyear)=elem_prod(N_mdyr(iyear),sel_HB(iyear));
      pred_HB_cpue(iyear)=q_HB(iyear)*q_rate_fcn_HB(iyear)*q_DD_fcn(iyear)*sum(N_HB(iyear));
      if (iyear<endyr_HB_cpue){q_HB(iyear+1)=q_HB(iyear)*mfexp(q_RW_log_dev_HB(iyear));}
  }

 //GR  cpue
  q_GR(styr_GR_cpue)=mfexp(log_q_GR);
  for (iyear=styr_GR_cpue; iyear<=endyr_GR_cpue; iyear++)
  {
      N_GR(iyear)=elem_prod(N_mdyr(iyear),sel_HB(iyear)); //GR uses HB selex
      pred_GR_cpue(iyear)=q_GR(iyear)*q_rate_fcn_GR(iyear)*q_DD_fcn(iyear)*sum(N_GR(iyear));
      if (iyear<endyr_GR_cpue){q_GR(iyear+1)=q_GR(iyear)*mfexp(q_RW_log_dev_GR(iyear));}
  }

FUNCTION get_length_comps

 //comm handline
  for (iyear=1;iyear<=nyr_cH_lenc;iyear++)
  {pred_cH_lenc(iyear)=(L_cH_num(yrs_cH_lenc(iyear))*lenprob_cH)/sum(L_cH_num(yrs_cH_lenc(iyear)));}

 //comm diving
  for (iyear=1;iyear<=nyr_cD_lenc;iyear++)
  {pred_cD_lenc(iyear)=(L_cD_num(yrs_cD_lenc(iyear))*lenprob_cD)/sum(L_cD_num(yrs_cD_lenc(iyear)));}

 //headboat
  for (iyear=1;iyear<=nyr_HB_lenc;iyear++)
  {pred_HB_lenc(iyear)=(L_HB_num(yrs_HB_lenc(iyear))*lenprob_HB)/sum(L_HB_num(yrs_HB_lenc(iyear)));}



FUNCTION get_age_comps

  //Commercial handline
  for (iyear=1;iyear<=nyr_cH_agec;iyear++)
  {
    ErrorFree_cH_agec(iyear)=L_cH_num(yrs_cH_agec(iyear))/sum(L_cH_num(yrs_cH_agec(iyear)));
    pred_cH_agec_allages(iyear)=age_error*(ErrorFree_cH_agec(iyear)/sum(ErrorFree_cH_agec(iyear)));
    for (iage=1; iage<=nages_agec; iage++) {pred_cH_agec(iyear,iage)=pred_cH_agec_allages(iyear,iage);}
    for (iage=(nages_agec+1); iage<=nages; iage++) {pred_cH_agec(iyear,nages_agec)+=pred_cH_agec_allages(iyear,iage);} //plus group
  }

  //Commercial diving
  for (iyear=1;iyear<=nyr_cD_agec;iyear++)
  {
    ErrorFree_cD_agec(iyear)=L_cD_num(yrs_cD_agec(iyear))/sum(L_cD_num(yrs_cD_agec(iyear)));
    pred_cD_agec_allages(iyear)=age_error*(ErrorFree_cD_agec(iyear)/sum(ErrorFree_cD_agec(iyear)));
    for (iage=1; iage<=nages_agec; iage++) {pred_cD_agec(iyear,iage)=pred_cD_agec_allages(iyear,iage);}
    for (iage=(nages_agec+1); iage<=nages; iage++) {pred_cD_agec(iyear,nages_agec)+=pred_cD_agec_allages(iyear,iage);} //plus group
  }

 //Headboat
 for (iyear=1;iyear<=nyr_HB_agec;iyear++)
  {
    ErrorFree_HB_agec(iyear)=L_HB_num(yrs_HB_agec(iyear))/sum(L_HB_num(yrs_HB_agec(iyear)));
    pred_HB_agec_allages(iyear)=age_error*ErrorFree_HB_agec(iyear);
    for (iage=1; iage<=nages_agec; iage++) {pred_HB_agec(iyear,iage)=pred_HB_agec_allages(iyear,iage);}
    for (iage=(nages_agec+1); iage<=nages; iage++) {pred_HB_agec(iyear,nages_agec)+=pred_HB_agec_allages(iyear,iage);} //plus group
  }


////----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
FUNCTION get_weighted_current
  F_temp_sum=0.0;
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_cH+
        sum(log_F_dev_cH((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_cD+
```

```
       sum(log_F_dev_cD((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_HB+
       sum(log_F_dev_HB((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_GR+
       sum(log_F_dev_GR((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_cH_D+
       sum(log_F_dev_cH_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_HB_D+
       sum(log_F_dev_HB_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_GR_D+
       sum(log_F_dev_GR_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);

  F_cH_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_cH+
       sum(log_F_dev_cH((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_cD_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_cD+
       sum(log_F_dev_cD((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_HB_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_HB+
       sum(log_F_dev_HB((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_GR_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_GR+
       sum(log_F_dev_GR((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_cH_D_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_cH_D+
       sum(log_F_dev_cH_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_HB_D_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_HB_D+
       sum(log_F_dev_HB_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_GR_D_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_GR_D+
       sum(log_F_dev_GR_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;

  log_F_dev_end_cH=sum(log_F_dev_cH((endyr-selpar_n_yrs_wgted+1),endyr))/selpar_n_yrs_wgted;
  log_F_dev_end_cD=sum(log_F_dev_cD((endyr-selpar_n_yrs_wgted+1),endyr))/selpar_n_yrs_wgted;
  log_F_dev_end_HB=sum(log_F_dev_HB((endyr-selpar_n_yrs_wgted+1),endyr))/selpar_n_yrs_wgted;
  log_F_dev_end_GR=sum(log_F_dev_GR((endyr-selpar_n_yrs_wgted+1),endyr))/selpar_n_yrs_wgted;

  log_F_dev_end_cH_D=sum(log_F_dev_cH_D((endyr-selpar_n_yrs_wgted+1),endyr))/selpar_n_yrs_wgted;
  log_F_dev_end_HB_D=sum(log_F_dev_HB_D((endyr-selpar_n_yrs_wgted+1),endyr))/selpar_n_yrs_wgted;
  log_F_dev_end_GR_D=sum(log_F_dev_GR_D((endyr-selpar_n_yrs_wgted+1),endyr))/selpar_n_yrs_wgted;

  F_end_L=sel_cH(endyr)*mfexp(log_avg_F_cH+log_F_dev_end_cH)+
         sel_cD(endyr)*mfexp(log_avg_F_cD+log_F_dev_end_cD)+
         sel_HB(endyr)*mfexp(log_avg_F_HB+log_F_dev_end_HB)+
         sel_HB(endyr)*mfexp(log_avg_F_GR+log_F_dev_end_GR);   //GR uses HB selex

  F_end_D=sel_cH_D(endyr)*mfexp(log_avg_F_cH_D+log_F_dev_end_cH_D)+
         sel_HB_D(endyr)*mfexp(log_avg_F_HB_D+log_F_dev_end_HB_D)+
         sel_HB_D(endyr)*mfexp(log_avg_F_GR_D+log_F_dev_end_GR_D);    //GR uses HB selex

  F_end=F_end_L+F_end_D;
  F_end_apex=max(F_end);

  sel_wgted_tot=F_end/F_end_apex;
  sel_wgted_L=elem_prod(sel_wgted_tot, elem_div(F_end_L,F_end));
  sel_wgted_D=elem_prod(sel_wgted_tot, elem_div(F_end_D,F_end));

  wgt_wgted_L_denom=F_cH_prop+F_cD_prop+F_HB_prop+F_GR_prop;
  wgt_wgted_L_klb=F_cH_prop/wgt_wgted_L_denom*gutwgt_cH_klb(endyr)+
                 F_cD_prop/wgt_wgted_L_denom*gutwgt_cD_klb(endyr)+
                 F_HB_prop/wgt_wgted_L_denom*gutwgt_HB_klb(endyr)+
                 F_GR_prop/wgt_wgted_L_denom*gutwgt_GR_klb(endyr);

  wgt_wgted_D_denom=F_cH_D_prop+F_HB_D_prop+F_GR_D_prop;
  wgt_wgted_D_klb=F_cH_D_prop/wgt_wgted_D_denom*gutwgt_cH_D_klb(endyr)+
                 F_HB_D_prop/wgt_wgted_D_denom*gutwgt_HB_D_klb(endyr)+
                 F_GR_D_prop/wgt_wgted_D_denom*gutwgt_GR_D_klb(endyr);

FUNCTION get_msy

  SSB_msy_out=0.0;
  //compute values as functions of F
  for(ff=1; ff<=n_iter_msy; ff++)
  {
    //uses fishery-weighted F's
    Z_age_msy=0.0;
    F_L_age_msy=0.0;
    F_D_age_msy=0.0;

    F_L_age_msy=F_msy(ff)*sel_wgted_L;
    F_D_age_msy=F_msy(ff)*sel_wgted_D;
    Z_age_msy=M+F_L_age_msy+F_D_age_msy;

    N_age_msy(1)=1.0;
    for (iage=2; iage<=nages; iage++)
      {N_age_msy(iage)=N_age_msy(iage-1)*mfexp(-1.*Z_age_msy(iage-1));}
    N_age_msy(nages)=N_age_msy(nages)/(1.0-mfexp(-1.*Z_age_msy(nages)));
    N_age_msy_spawn(1,(nages-1))=elem_prod(N_age_msy(1,(nages-1)),
                                mfexp((-1.*Z_age_msy(1,(nages-1)))*spawn_time_frac));
    N_age_msy_spawn(nages)=(N_age_msy_spawn(nages-1)*(mfexp(-1.*(Z_age_msy(nages-1)*(1.0-spawn_time_frac) +
                          Z_age_msy(nages)*spawn_time_frac) )))/(1.0-mfexp(-1.*Z_age_msy(nages)));

    spr_msy(ff)=sum(elem_prod(N_age_msy_spawn,reprod(endyr)));

    R_eq(ff)=SR_eq_func(R0, steep, spr_msy(1), spr_msy(ff), BiasCor, SR_switch);

    if (R_eq(ff)<dzero) {R_eq(ff)=dzero;}
    N_age_msy*=R_eq(ff);
    N_age_msy_spawn*=R_eq(ff);

    for (iage=1; iage<=nages; iage++)
    {
      L_age_msy(iage)=N_age_msy(iage)*(F_L_age_msy(iage)/Z_age_msy(iage))*
                    (1.-mfexp(-1.*Z_age_msy(iage)));
      D_age_msy(iage)=N_age_msy(iage)*(F_D_age_msy(iage)/Z_age_msy(iage))*
                    (1.-mfexp(-1.0*Z_age_msy(iage)));
    }

    SSB_eq(ff)=sum(elem_prod(N_age_msy_spawn,reprod(endyr)));
```

```
    B_eq(ff)=sum(elem_prod(N_age_msy,wgt_mt));
    L_eq_klb(ff)=sum(elem_prod(L_age_msy,wgt_wgted_L_klb)); //in gutted weight
    L_eq_knum(ff)=sum(L_age_msy)/1000.0;
    D_eq_klb(ff)=sum(elem_prod(D_age_msy,wgt_wgted_D_klb)); //in gutted weight
    D_eq_knum(ff)=sum(D_age_msy)/1000.0;
  }

  msy_klb_out=max(L_eq_klb); //msy in gutted weight

  for(ff=1; ff<=n_iter_msy; ff++)
  {
   if(L_eq_klb(ff) == msy_klb_out)
      {
        SSB_msy_out=SSB_eq(ff);
        B_msy_out=B_eq(ff);
        R_msy_out=R_eq(ff);
        msy_knum_out=L_eq_knum(ff);
        D_msy_knum_out=D_eq_knum(ff);
        D_msy_klb_out=D_eq_klb(ff);
        F_msy_out=F_msy(ff);
        spr_msy_out=spr_msy(ff);
      }
  }

//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
FUNCTION get_miscellaneous_stuff

//switch here if var_rec_dev <=dzero
  if(var_rec_dev>0.0)
   {sigma_rec_dev=sqrt(var_rec_dev);}
   else{sigma_rec_dev=0.0;}

  len_cv=elem_div(len_sd,meanlen_TL);

  //compute total landings- and discards-at-age in 1000 fish and klb gutted weight
  L_total_num.initialize();
  L_total_klb.initialize();
  L_total_knum_yr.initialize();
  L_total_klb_yr.initialize();
  D_total_num.initialize();
  D_total_klb.initialize();
  D_total_knum_yr.initialize();
  D_total_klb_yr.initialize();
  D_cH_klb.initialize();D_HB_klb.initialize();D_GR_klb.initialize();

  for(iyear=styr; iyear<=endyr; iyear++)
  {
        L_total_klb_yr(iyear)=pred_cH_L_klb(iyear)+pred_cD_L_klb(iyear)+pred_HB_L_klb(iyear)+pred_GR_L_klb(iyear);
        L_total_knum_yr(iyear)=pred_cH_L_knum(iyear)+pred_cD_L_knum(iyear)+pred_HB_L_knum(iyear)+pred_GR_L_knum(iyear);

        B(iyear)=elem_prod(N(iyear),wgt_mt);
        totN(iyear)=sum(N(iyear));
        totB(iyear)=sum(B(iyear));

        if (iyear>=styr_cH_D && iyear<=endyr_cH_D)
        {
         D_total_knum_yr(iyear)+=pred_cH_D_knum(iyear);
         D_total_klb_yr(iyear)+=pred_cH_D_klb(iyear);
         D_cH_klb(iyear)=elem_prod(D_cH_num(iyear),gutwgt_cH_D_klb(iyear));     //in 1000 lb gutted
        }

        if (iyear>=styr_HB_D && iyear<=endyr_HB_D)
        {
         D_total_knum_yr(iyear)+=pred_HB_D_knum(iyear);
         D_total_klb_yr(iyear)+=pred_HB_D_klb(iyear);
         D_HB_klb(iyear)=elem_prod(D_HB_num(iyear),gutwgt_HB_D_klb(iyear));     //in 1000 lb gutted
        }

        if (iyear>=styr_GR_D && iyear<=endyr_GR_D)
        {
         D_total_knum_yr(iyear)+=pred_GR_D_knum(iyear);
         D_total_klb_yr(iyear)+=pred_GR_D_klb(iyear);
         D_GR_klb(iyear)=elem_prod(D_GR_num(iyear),gutwgt_GR_D_klb(iyear));     //in 1000 lb gutted
        }
  }

  L_total_num=L_cH_num+L_cD_num+L_HB_num+L_GR_num;   //landings at age in number fish
  L_total_klb=L_cH_klb+L_cD_klb+L_HB_klb+L_GR_klb;   //landings at age in klb gutted weight

  D_total_num=(D_cH_num+D_HB_num+D_GR_num);          //discards at age in number fish
  D_total_klb=D_cH_klb+D_HB_klb+D_GR_klb;            //discards at age in klb gutted weight

  //Time series of interest
  B(endyr+1)=elem_prod(N(endyr+1),wgt_mt);
  totN(endyr+1)=sum(N(endyr+1));
  totB(endyr+1)=sum(B(endyr+1));
  N_spawn(endyr+1)=N(endyr+1);
  SSB(endyr+1)=sum(elem_prod(N_spawn(endyr+1),reprod(endyr)));
  MatFemB(endyr+1)=sum(elem_prod(N_spawn(endyr+1),reprod2(endyr)));
  rec=column(N,1);
  SdS0=SSB/S0;

  if(F_msy_out>0)
    {
      FdF_msy=Fapex/F_msy_out;
      FdF_msy_end=FdF_msy(endyr);
      FdF_msy_end_mean=pow((FdF_msy(endyr)*FdF_msy(endyr-1)*FdF_msy(endyr-2)),(1.0/3.0));
    }
  if(SSB_msy_out>0)
    {
      SdSSB_msy=SSB/SSB_msy_out;
      SdSSB_msy_end=SdSSB_msy(endyr);
    }
```

```
  //fill in log recruitment deviations for yrs they are nonzero
  for(iyear=styr_rec_dev; iyear<=endyr_rec_dev; iyear++)
    {log_rec_dev_output(iyear)=log_rec_dev(iyear);}
  //fill in log Nage deviations for ages they are nonzero (ages2+)
  for(iage=2; iage<=nages; iage++)
    {log_Nage_dev_output(iage)=log_Nage_dev(iage);}


//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
FUNCTION get_per_recruit_stuff

  //static per-recruit stuff
  for(iyear=styr; iyear<=endyr; iyear++)
  {
    N_age_spr(1)=1.0;
    for(iage=2; iage<=nages; iage++)
     {N_age_spr(iage)=N_age_spr(iage-1)*mfexp(-1.*Z(iyear,iage-1));}
    N_age_spr(nages)=N_age_spr(nages)/(1.0-mfexp(-1.*Z(iyear,nages)));
    N_age_spr_spawn(1,(nages-1))=elem_prod(N_age_spr(1,(nages-1)),
                                mfexp(-1.*Z(iyear)(1,(nages-1))*spawn_time_frac));
    N_age_spr_spawn(nages)=(N_age_spr_spawn(nages-1)*
                          (mfexp(-1.*(Z(iyear)(nages-1)*(1.0-spawn_time_frac) + Z(iyear)(nages)*spawn_time_frac) )))
                          /(1.0-mfexp(-1.*Z(iyear)(nages)));
    spr_static(iyear)=sum(elem_prod(N_age_spr_spawn,reprod(iyear)))/spr_F0;
  }


  //compute SSB/R and YPR as functions of F
  for(ff=1; ff<=n_iter_spr; ff++)
  {
    //uses fishery-weighted F's, same as in MSY calculations
    Z_age_spr=0.0;
    F_L_age_spr=0.0;

    F_L_age_spr=F_spr(ff)*sel_wgted_L;

    Z_age_spr=M+F_L_age_spr+F_spr(ff)*sel_wgted_D;

    N_age_spr(1)=1.0;
    for (iage=2; iage<=nages; iage++)
     {N_age_spr(iage)=N_age_spr(iage-1)*mfexp(-1.*Z_age_spr(iage-1));}
    N_age_spr(nages)=N_age_spr(nages)/(1-mfexp(-1.*Z_age_spr(nages)));
    N_age_spr_spawn(1,(nages-1))=elem_prod(N_age_spr(1,(nages-1)),
                                   mfexp((-1.*Z_age_spr(1,(nages-1)))*spawn_time_frac));
    N_age_spr_spawn(nages)=(N_age_spr_spawn(nages-1)*
                          (mfexp(-1.*(Z_age_spr(nages-1)*(1.0-spawn_time_frac) + Z_age_spr(nages)*spawn_time_frac) )))
                          /(1.0-mfexp(-1.*Z_age_spr(nages)));

    spr_spr(ff)=sum(elem_prod(N_age_spr_spawn,reprod(endyr)));
    L_spr(ff)=0.0;
    for (iage=1; iage<=nages; iage++)
    {
      L_age_spr(iage)=N_age_spr(iage)*(F_L_age_spr(iage)/Z_age_spr(iage))*
                      (1.-mfexp(-1.*Z_age_spr(iage)));
      L_spr(ff)+=L_age_spr(iage)*wgt_wgted_L_klb(iage)*1000.0; //in lb gutted wgt
    }
  }

//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
FUNCTION get_effective_sample_sizes
      neff_cH_lenc_allyr_out=missing;
      neff_cD_lenc_allyr_out=missing;
      neff_HB_lenc_allyr_out=missing;

      neff_cH_agec_allyr_out=missing;
      neff_cD_agec_allyr_out=missing;
      neff_HB_agec_allyr_out=missing;

      for (iyear=1; iyear<=nyr_cH_lenc; iyear++)
         {if (nsamp_cH_lenc(iyear)>=minSS_cH_lenc)
            {neff_cH_lenc_allyr_out(yrs_cH_lenc(iyear))=multinom_eff_N(pred_cH_lenc(iyear),obs_cH_lenc(iyear));}
          else {neff_cH_lenc_allyr_out(yrs_cH_lenc(iyear))=-99;}
         }

      for (iyear=1; iyear<=nyr_cD_lenc; iyear++)
         {if (nsamp_cD_lenc(iyear)>=minSS_cD_lenc)
            {neff_cD_lenc_allyr_out(yrs_cD_lenc(iyear))=multinom_eff_N(pred_cD_lenc(iyear),obs_cD_lenc(iyear));}
          else {neff_cD_lenc_allyr_out(yrs_cD_lenc(iyear))=-99;}
         }

      for (iyear=1; iyear<=nyr_HB_lenc; iyear++)
         {if (nsamp_HB_lenc(iyear)>=minSS_HB_lenc)
            {neff_HB_lenc_allyr_out(yrs_HB_lenc(iyear))=multinom_eff_N(pred_HB_lenc(iyear),obs_HB_lenc(iyear));}
          else {neff_HB_lenc_allyr_out(yrs_HB_lenc(iyear))=-99;}
         }


      for (iyear=1; iyear<=nyr_cH_agec; iyear++)
         {if (nsamp_cH_agec(iyear)>=minSS_cH_agec)
            {neff_cH_agec_allyr_out(yrs_cH_agec(iyear))=multinom_eff_N(pred_cH_agec(iyear),obs_cH_agec(iyear));}
          else {neff_cH_agec_allyr_out(yrs_cH_agec(iyear))=-99;}
         }

      for (iyear=1; iyear<=nyr_cD_lenc; iyear++)
         {if (nsamp_cD_lenc(iyear)>=minSS_cD_lenc)
            {neff_cD_lenc_allyr_out(yrs_cD_lenc(iyear))=multinom_eff_N(pred_cD_lenc(iyear),obs_cD_lenc(iyear));}
          else {neff_cD_lenc_allyr_out(yrs_cD_lenc(iyear))=-99;}
         }

      for (iyear=1; iyear<=nyr_HB_agec; iyear++)
         {if (nsamp_HB_agec(iyear)>=minSS_HB_agec)
            {neff_HB_agec_allyr_out(yrs_HB_agec(iyear))=multinom_eff_N(pred_HB_agec(iyear),obs_HB_agec(iyear));}
          else {neff_HB_agec_allyr_out(yrs_HB_agec(iyear))=-99;}
         }
```

```
//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

FUNCTION evaluate_objective_function

  fval=0.0;
  fval_data=0.0;

//---Indices-------------------------------

  f_cH_cpue=0.0;
  f_cH_cpue=lk_lognormal(pred_cH_cpue, obs_cH_cpue, cH_cpue_cv, w_I_cH);
  fval+=f_cH_cpue;
  fval_data+=f_cH_cpue;

  f_HB_cpue=0.0;
  f_HB_cpue=lk_lognormal(pred_HB_cpue, obs_HB_cpue, HB_cpue_cv, w_I_HB);
  fval+=f_HB_cpue;
  fval_data+=f_HB_cpue;

  f_GR_cpue=0.0;
  f_GR_cpue=lk_lognormal(pred_GR_cpue, obs_GR_cpue, GR_cpue_cv, w_I_GR);
  fval+=f_GR_cpue;
  fval_data+=f_GR_cpue;

//---Landings-------------------------------

  //f_cH_L in 1000 lb gutted wgt
  f_cH_L=lk_lognormal(pred_cH_L_klb(styr_cH_L,endyr_cH_L), obs_cH_L(styr_cH_L,endyr_cH_L),
                    cH_L_cv(styr_cH_L,endyr_cH_L), w_L);
  fval+=f_cH_L;
  fval_data+=f_cH_L;

  //f_cD_L in 1000 lb gutted wgt
  f_cD_L=lk_lognormal(pred_cD_L_klb(styr_cD_L,endyr_cD_L), obs_cD_L(styr_cD_L,endyr_cD_L),
                    cD_L_cv(styr_cD_L,endyr_cD_L), w_L);
  fval+=f_cD_L;
  fval_data+=f_cD_L;

  //f_HB_L in 1000 fish
  f_HB_L=lk_lognormal(pred_HB_L_knum(styr_HB_L,endyr_HB_L), obs_HB_L(styr_HB_L,endyr_HB_L),
                    HB_L_cv(styr_HB_L,endyr_HB_L), w_L);
  fval+=f_HB_L;
  fval_data+=f_HB_L;

  //f_GR_L in 1000 fish
  f_GR_L=lk_lognormal(pred_GR_L_knum(styr_GR_L,endyr_GR_L), obs_GR_L(styr_GR_L,endyr_GR_L),
                    GR_L_cv(styr_GR_L,endyr_GR_L), w_L);
  fval+=f_GR_L;
  fval_data+=f_GR_L;

//---Discards-------------------------------

  //f_cH_D in 1000 fish
  f_cH_D=lk_lognormal(pred_cH_D_knum(styr_cH_D,endyr_cH_D), obs_cH_D(styr_cH_D,endyr_cH_D),
                    cH_D_cv(styr_cH_D,endyr_cH_D), w_D);
  fval+=f_cH_D;
  fval_data+=f_cH_D;

  //f_HB_D in 1000 fish
  f_HB_D=lk_lognormal(pred_HB_D_knum(styr_HB_D,endyr_HB_D), obs_HB_D(styr_HB_D,endyr_HB_D),
                    HB_D_cv(styr_HB_D,endyr_HB_D), w_D);
  fval+=f_HB_D;
  fval_data+=f_HB_D;

  //f_GR_D in 1000 fish
  f_GR_D=lk_lognormal(pred_GR_D_knum(styr_GR_D,endyr_GR_D), obs_GR_D(styr_GR_D,endyr_GR_D),
                    GR_D_cv(styr_GR_D,endyr_GR_D), w_D);
  fval+=f_GR_D;
  fval_data+=f_GR_D;

//---Length comps-------------------------------

  //f_cH_lenc
  f_cH_lenc=lk_robust_multinomial(nsamp_cH_lenc, pred_cH_lenc, obs_cH_lenc, nyr_cH_lenc, double(nlenbins), minSS_cH_lenc, w_lc_cH);
  fval+=f_cH_lenc;
  fval_data+=f_cH_lenc;

  //f_cD_lenc
  f_cD_lenc=lk_robust_multinomial(nsamp_cD_lenc, pred_cD_lenc, obs_cD_lenc, nyr_cD_lenc, double(nlenbins), minSS_cD_lenc, w_lc_cD);
  fval+=f_cD_lenc;
  fval_data+=f_cD_lenc;

  //f_HB_lenc
  f_HB_lenc=lk_robust_multinomial(nsamp_HB_lenc, pred_HB_lenc, obs_HB_lenc, nyr_HB_lenc, double(nlenbins), minSS_HB_lenc, w_lc_HB);
  fval+=f_HB_lenc;
  fval_data+=f_HB_lenc;


//---Age comps-------------------------------

  //f_cH_agec
  f_cH_agec=lk_robust_multinomial(nsamp_cH_agec, pred_cH_agec, obs_cH_agec, nyr_cH_agec, double(nages_agec), minSS_cH_agec, w_ac_cH);
  fval+=f_cH_agec;
  fval_data+=f_cH_agec;

  //f_cD_agec
  f_cD_agec=lk_robust_multinomial(nsamp_cD_agec, pred_cD_agec, obs_cD_agec, nyr_cD_agec, double(nages_agec), minSS_cD_agec, w_ac_cD);
  fval+=f_cD_agec;
  fval_data+=f_cD_agec;

  //f_HB_agec
```

```
  f_HB_agec=lk_robust_multinomial(nsamp_HB_agec, pred_HB_agec, obs_HB_agec, nyr_HB_agec, double(nages_agec), minSS_HB_agec, w_ac_HB);
  fval+=f_HB_agec;
  fval_data+=f_HB_agec;


//----------Constraints and penalties-------------------------------

  //Light penalty applied to log_Nage_dev for deviation from zero. If not estimated, this penalty equals zero.
  f_Nage_init=norm2(log_Nage_dev);
  fval+=w_Nage_init*f_Nage_init;

  f_rec_dev=0.0;
  rec_logL_add=nyrs_rec*log(rec_sigma);
  f_rec_dev=(square(log_rec_dev(styr_rec_dev) + rec_sigma_sq/2.0)/(2.0*rec_sigma_sq));
  for(iyear=(styr_rec_dev+1); iyear<=endyr; iyear++)
  {f_rec_dev+=(square(log_rec_dev(iyear)-R_autocorr*log_rec_dev(iyear-1) + rec_sigma_sq/2.0)/
             (2.0*rec_sigma_sq));}
  f_rec_dev+=rec_logL_add;
  fval+=w_rec*f_rec_dev;


  f_rec_dev_early=0.0; //possible extra constraint on early rec deviations
  if (w_rec_early>0.0)
    { if (styr_rec_dev<endyr_rec_phase1)
        {
          for(iyear=styr_rec_dev; iyear<=endyr_rec_phase1; iyear++)
          {f_rec_dev_early+=square(log_rec_dev(iyear));}
        }
  fval+=w_rec_early*f_rec_dev_early;
  }

  f_rec_dev_end=0.0; //possible extra constraint on ending rec deviations
  if (w_rec_end>0.0)
  { if (endyr_rec_phase2<endyr_rec_dev)
        {
          for(iyear=(endyr_rec_phase2+1); iyear<=endyr_rec_dev; iyear++)
          {f_rec_dev_end+=square(log_rec_dev(iyear));}
        }
      fval+=w_rec_end*f_rec_dev_end;
   }

  //Ftune penalty: does not apply in last phase
  f_Ftune=0.0;
  if (w_Ftune>0.0)
  {if (set_Ftune>0.0 && !last_phase()) {f_Ftune=square(Fapex(set_Ftune_yr)-set_Ftune);}
   fval+=w_Ftune*f_Ftune;
  }

  //Penalty if apical F exceeds 3.0
  f_fullF_constraint=0.0;
  if (w_fullF>0.0)
  {for (iyear=styr; iyear<=endyr; iyear++)
        {if(Fapex(iyear)>3.0) {f_fullF_constraint+=(mfexp(Fapex(iyear)-3.0)-1.0);}}
   fval+=w_fullF*f_fullF_constraint;
  }

//  //Random walk components of fishery dependent indices
//  f_HB_RW_cpue=0.0;
//  for (iyear=styr_HB_cpue; iyear<endyr_HB_cpue; iyear++)
//      {f_HB_RW_cpue+=square(q_RW_log_dev_HB(iyear))/(2.0*set_q_RW_HB_var);}
//  fval+=f_HB_RW_cpue;


//---Priors---------------------------------------------------
//neg_log_prior arguments: estimate, prior mean, prior var/-CV, pdf type
//Variance input as a negative value is considered to be CV in arithmetic space (CV=-1 implies loose prior)
//pdf type 1=none, 2=lognormal, 3=normal, 4=beta
  f_priors=0.0;
  f_priors+=neg_log_prior(Linf,set_Linf(5),set_Linf(6),set_Linf(7));
  f_priors+=neg_log_prior(K,set_K(5),set_K(6),set_K(7));
  f_priors+=neg_log_prior(t0,set_t0(5),set_t0(6),set_t0(7));
  f_priors+=neg_log_prior(len_cv_val,set_len_cv(5),set_len_cv(6),set_len_cv(7));
  f_priors+=neg_log_prior(M_constant,set_M_constant(5),set_M_constant(6),set_M_constant(7));

  f_priors+=neg_log_prior(steep,set_steep(5),set_log_R0(6),set_log_R0(7));
  f_priors+=neg_log_prior(log_R0,set_log_R0(5),set_log_R0(6),set_log_R0(7));
  f_priors+=neg_log_prior(R_autocorr,set_R_autocorr(5),set_R_autocorr(6),set_R_autocorr(7));
  f_priors+=neg_log_prior(rec_sigma,set_rec_sigma(5),set_rec_sigma(6),set_rec_sigma(7));

  f_priors+=neg_log_prior(selpar_L50_cH1,set_selpar_L50_cH1(5), set_selpar_L50_cH1(6), set_selpar_L50_cH1(7));
  f_priors+=neg_log_prior(selpar_slope_cH1,set_selpar_slope_cH1(5), set_selpar_slope_cH1(6), set_selpar_slope_cH1(7));
  f_priors+=neg_log_prior(selpar_L50_cH2,set_selpar_L50_cH2(5), set_selpar_L50_cH2(6), set_selpar_L50_cH2(7));
  f_priors+=neg_log_prior(selpar_slope_cH2,set_selpar_slope_cH2(5), set_selpar_slope_cH2(6), set_selpar_slope_cH2(7));
  f_priors+=neg_log_prior(selpar_L50_cH3,set_selpar_L50_cH3(5), set_selpar_L50_cH3(6), set_selpar_L50_cH3(7));
  f_priors+=neg_log_prior(selpar_slope_cH3,set_selpar_slope_cH3(5), set_selpar_slope_cH3(6), set_selpar_slope_cH3(7));

  f_priors+=neg_log_prior(selpar_L50_cD,set_selpar_L50_cD(5), set_selpar_L50_cD(6), set_selpar_L50_cD(7));
  f_priors+=neg_log_prior(selpar_slope_cD,set_selpar_slope_cD(5), set_selpar_slope_cD(6), set_selpar_slope_cD(7));
  f_priors+=neg_log_prior(selpar_afull_cD,set_selpar_afull_cD(5), set_selpar_afull_cD(6), set_selpar_afull_cD(7));
  f_priors+=neg_log_prior(selpar_sigma_cD,set_selpar_sigma_cD(5), set_selpar_sigma_cD(6), set_selpar_sigma_cD(7));

  f_priors+=neg_log_prior(selpar_L50_HB1,set_selpar_L50_HB1(5), set_selpar_L50_HB1(6), set_selpar_L50_HB1(7));
  f_priors+=neg_log_prior(selpar_slope_HB1,set_selpar_slope_HB1(5), set_selpar_slope_HB1(6), set_selpar_slope_HB1(7));
  f_priors+=neg_log_prior(selpar_L50_HB2,set_selpar_L50_HB2(5), set_selpar_L50_HB2(6), set_selpar_L50_HB2(7));
  f_priors+=neg_log_prior(selpar_slope_HB2,set_selpar_slope_HB2(5), set_selpar_slope_HB2(6), set_selpar_slope_HB2(7));
  f_priors+=neg_log_prior(selpar_L50_HB3,set_selpar_L50_HB3(5), set_selpar_L50_HB3(6), set_selpar_L50_HB3(7));
  f_priors+=neg_log_prior(selpar_slope_HB3,set_selpar_slope_HB3(5), set_selpar_slope_HB3(6), set_selpar_slope_HB3(7));

  f_priors+=neg_log_prior(log_q_cH,set_log_q_cH(5),set_log_q_cH(6),set_log_q_cH(7));
  f_priors+=neg_log_prior(log_q_HB,set_log_q_HB(5),set_log_q_HB(6),set_log_q_HB(7));
  f_priors+=neg_log_prior(log_q_GR,set_log_q_GR(5),set_log_q_GR(6),set_log_q_GR(7));

  f_priors+=neg_log_prior(F_init,set_F_init(5),set_F_init(6),set_F_init(7));
```

```
  fval+=f_priors;


//---------------------------------------------------------------------------------
//Logistic function: 2 parameters
FUNCTION dvar_vector logistic(const dvar_vector& ages, const dvariable& L50, const dvariable& slope)
  //ages=vector of ages, L50=age at 50% selectivity, slope=rate of increase
  RETURN_ARRAYS_INCREMENT();
  dvar_vector Sel_Tmp(ages.indexmin(),ages.indexmax());
  Sel_Tmp=1./(1.+mfexp(-1.*slope*(ages-L50))); //logistic;
  RETURN_ARRAYS_DECREMENT();
  return Sel_Tmp;

//---------------------------------------------------------------------------------
//Logistic-exponential: 4 parameters (but 1 is fixed)
FUNCTION dvar_vector logistic_exponential(const dvar_vector& ages, const dvariable& L50, const dvariable& slope, const dvariable& sigma, const dvariable& joint)
  //ages=vector of ages, L50=age at 50% sel (ascending limb), slope=rate of increase, sigma=controls rate of descent (descending)
  //joint=age to join curves
  RETURN_ARRAYS_INCREMENT();
  dvar_vector Sel_Tmp(ages.indexmin(),ages.indexmax());
  Sel_Tmp=1.0;
  for (iage=1; iage<=nages; iage++)
  {
   if (ages(iage)<joint) {Sel_Tmp(iage)=1./(1.+mfexp(-1.*slope*(ages(iage)-L50)));}
   if (ages(iage)>joint){Sel_Tmp(iage)=mfexp(-1.*square((ages(iage)-joint)/sigma));}
  }
  Sel_Tmp=Sel_Tmp/max(Sel_Tmp);
  RETURN_ARRAYS_DECREMENT();
  return Sel_Tmp;

//---------------------------------------------------------------------------------
//Logistic function: 4 parameters
FUNCTION dvar_vector logistic_double(const dvar_vector& ages, const dvariable& L501, const dvariable& slope1, const dvariable& L502, const dvariable& slope2)
  //ages=vector of ages, L50=age at 50% selectivity, slope=rate of increase, L502=age at 50% decrease additive to L501, slope2=slope of decrease
  RETURN_ARRAYS_INCREMENT();
  dvar_vector Sel_Tmp(ages.indexmin(),ages.indexmax());
  Sel_Tmp=elem_prod( (1./(1.+mfexp(-1.*slope1*(ages-L501)))),(1.-(1./(1.+mfexp(-1.*slope2*(ages-(L501+L502)))))) );
  Sel_Tmp=Sel_Tmp/max(Sel_Tmp);
  RETURN_ARRAYS_DECREMENT();
  return Sel_Tmp;

//---------------------------------------------------------------------------------
//Jointed logistic function: 6 parameters (increasing and decreasing logistics joined at peak selectivity)
FUNCTION dvar_vector logistic_joint(const dvar_vector& ages, const dvariable& L501, const dvariable& slope1, const dvariable& L502, const dvariable& slope2, const dvariable& satval, const dvariable& joint)
  //ages=vector of ages, L501=age at 50% sel (ascending limb), slope1=rate of increase,L502=age at 50% sel (descending), slope1=rate of increase (ascending),
  //satval=saturation value of descending limb, joint=location in age vector to join curves (may equal age or age + 1 if age-0 is included)
  RETURN_ARRAYS_INCREMENT();
  dvar_vector Sel_Tmp(ages.indexmin(),ages.indexmax());
  Sel_Tmp=1.0;
  for (iage=1; iage<=nages; iage++)
  {
   if (double(iage)<joint) {Sel_Tmp(iage)=1./(1.+mfexp(-1.*slope1*(ages(iage)-L501)));}
   if (double(iage)>joint){Sel_Tmp(iage)=1.0-(1.0-satval)/(1.+mfexp(-1.*slope2*(ages(iage)-L502)));}
  }
  Sel_Tmp=Sel_Tmp/max(Sel_Tmp);
  RETURN_ARRAYS_DECREMENT();
  return Sel_Tmp;

//---------------------------------------------------------------------------------
//Double Gaussian function: 6 parameters (as in SS3)
FUNCTION dvar_vector gaussian_double(const dvar_vector& ages, const dvariable& peak, const dvariable& top, const dvariable& ascwid, const dvariable& deswid, const dvariable& init, const dvariable& final)
  //ages=vector of ages, peak=ascending inflection location (as logistic), top=width of plateau, ascwid=ascent width (as log(width))
  //deswid=descent width (as log(width))
  RETURN_ARRAYS_INCREMENT();
  dvar_vector Sel_Tmp(ages.indexmin(),ages.indexmax());
  dvar_vector sel_step1(ages.indexmin(),ages.indexmax());
  dvar_vector sel_step2(ages.indexmin(),ages.indexmax());
  dvar_vector sel_step3(ages.indexmin(),ages.indexmax());
  dvar_vector sel_step4(ages.indexmin(),ages.indexmax());
  dvar_vector sel_step5(ages.indexmin(),ages.indexmax());
  dvar_vector sel_step6(ages.indexmin(),ages.indexmax());
  dvar_vector pars_tmp(1,6); dvar_vector sel_tmp_iq(1,2);

  pars_tmp(1)=peak;
  pars_tmp(2)=peak+1.0+(0.99*ages(nages)-peak-1.0)/(1.0+mfexp(-top));
  pars_tmp(3)=mfexp(ascwid);
  pars_tmp(4)=mfexp(deswid);
  pars_tmp(5)=1.0/(1.0+mfexp(-init));
  pars_tmp(6)=1.0/(1.0+mfexp(-final));

  sel_tmp_iq(1)=mfexp(-(square(ages(1)-pars_tmp(1))/pars_tmp(3)));
  sel_tmp_iq(2)=mfexp(-(square(ages(nages)-pars_tmp(2))/pars_tmp(4)));

  sel_step1=mfexp(-(square(ages-pars_tmp(1))/pars_tmp(3)));
  sel_step2=pars_tmp(5)+(1.0-pars_tmp(5))*(sel_step1-sel_tmp_iq(1))/(1.0-sel_tmp_iq(1));
  sel_step3=mfexp(-(square(ages-pars_tmp(2))/pars_tmp(4)));
  sel_step4=1.0+(pars_tmp(6)-1.0)*(sel_step3-1.0)/(sel_tmp_iq(2)-1.0);
  sel_step5=1.0/ (1.0+mfexp(-(20.0* elem_div((ages-pars_tmp(1)), (1.0+sfabs(ages-pars_tmp(1)))) )));
  sel_step6=1.0/(1.0+mfexp(-(20.0*elem_div((ages-pars_tmp(2)),(1.0+sfabs(ages-pars_tmp(2)))) )));

  Sel_Tmp=elem_prod(sel_step2,(1.0-sel_step5))+
          elem_prod(sel_step5,((1.0-sel_step6)+ elem_prod(sel_step4,sel_step6)) );

  Sel_Tmp=Sel_Tmp/max(Sel_Tmp);
  RETURN_ARRAYS_DECREMENT();
  return Sel_Tmp;

//---------------------------------------------------------------------------------
//Spawner-recruit function (Beverton-Holt or Ricker)
FUNCTION dvariable SR_func(const dvariable& R0, const dvariable& h, const dvariable& spr_F0, const dvariable& SSB, int func)
  //R0=virgin recruitment, h=steepness, spr_F0=spawners per recruit @ F=0, SSB=spawning biomass
  //func=1 for Beverton-Holt, 2 for Ricker
  RETURN_ARRAYS_INCREMENT();
  dvariable Recruits_Tmp;
```

```
  switch(func) {
    case 1: //Beverton-Holt
      Recruits_Tmp=((0.8*R0*h*SSB)/(0.2*R0*spr_F0*(1.0-h)+(h-0.2)*SSB));
    break;
    case 2: //Ricker
      Recruits_Tmp=((SSB/spr_F0)*mfexp(h*(1-SSB/(R0*spr_F0))));
    break;
  }
  RETURN_ARRAYS_DECREMENT();
  return Recruits_Tmp;

//----------------------------------------------------------------------------------
//Spawner-recruit equilibrium function (Beverton-Holt or Ricker)
FUNCTION dvariable SR_eq_func(const dvariable& R0, const dvariable& h, const dvariable& spr_F0, const dvariable& spr_F, const dvariable& BC, int func)
  //R0=virgin recruitment, h=steepness, spr_F0=spawners per recruit @ F=0, spr_F=spawners per recruit @ F, BC=bias correction
  //func=1 for Beverton-Holt, 2 for Ricker
  RETURN_ARRAYS_INCREMENT();
  dvariable Recruits_Tmp;
  switch(func) {
    case 1: //Beverton-Holt
      Recruits_Tmp=(R0/((5.0*h-1.0)*spr_F))*(BC*4.0*h*spr_F-spr_F0*(1.0-h));
    break;
    case 2: //Ricker
      Recruits_Tmp=R0/(spr_F/spr_F0)*(1.0+log(BC*spr_F/spr_F0)/h);
    break;
  }
  RETURN_ARRAYS_DECREMENT();
  return Recruits_Tmp;


//----------------------------------------------------------------------------------
//compute multinomial effective sample size for a single yr
FUNCTION dvariable multinom_eff_N(const dvar_vector& pred_comp, const dvar_vector& obs_comp)
  //pred_comp=vector of predicted comps, obscomp=vector of observed comps
  dvariable EffN_Tmp; dvariable numer; dvariable denom;
  RETURN_ARRAYS_INCREMENT();
  numer=sum( elem_prod(pred_comp,(1.0-pred_comp)) );
  denom=sum( square(obs_comp-pred_comp) );
  if (denom>0.0) {EffN_Tmp=numer/denom;}
  else {EffN_Tmp=-missing;}
  RETURN_ARRAYS_DECREMENT();
  return EffN_Tmp;

//----------------------------------------------------------------------------------
//Likelihood contribution: lognormal
FUNCTION dvariable lk_lognormal(const dvar_vector& pred, const dvar_vector& obs, const dvar_vector& cv, const dvariable& wgt_dat)
  //pred=vector of predicted vals, obs=vector of observed vals, cv=vector of CVs in arithmetic space, wgt_dat=constant scaling of CVs
  //small_number is small value to avoid log(0) during search
  RETURN_ARRAYS_INCREMENT();
  dvariable LkvalTmp;
  dvariable small_number=0.00001;
  dvar_vector var(cv.indexmin(),cv.indexmax()); //variance in log space
  var=log(1.0+square(cv/wgt_dat));   // convert cv in arithmetic space to variance in log space
  LkvalTmp=sum(0.5*elem_div(square(log(elem_div((pred+small_number),(obs+small_number)))),var) );
  RETURN_ARRAYS_DECREMENT();
  return LkvalTmp;

//----------------------------------------------------------------------------------
//Likelihood contribution: multinomial
FUNCTION dvariable lk_multinomial(const dvar_vector& nsamp, const dvar_matrix& pred_comp, const dvar_matrix& obs_comp, const double& ncomp, const double& minSS, const dvariable& wgt_dat)
  //nsamp=vector of N's, pred_comp=matrix of predicted comps, obs_comp=matrix of observed comps, ncomp = number of yrs in matrix, minSS=min N threshold, wgt_dat=scaling of N's
  RETURN_ARRAYS_INCREMENT();
  dvariable LkvalTmp;
  dvariable small_number=0.00001;
  LkvalTmp=0.0;
  for (int ii=1; ii<=ncomp; ii++)
  {if (nsamp(ii)>=minSS)
    {LkvalTmp-=wgt_dat*nsamp(ii)*sum(elem_prod((obs_comp(ii)+small_number),
               log(elem_div((pred_comp(ii)+small_number), (obs_comp(ii)+small_number)))));
    }
  }
  RETURN_ARRAYS_DECREMENT();
  return LkvalTmp;


//----------------------------------------------------------------------------------
//Likelihood contribution: multinomial
FUNCTION dvariable lk_robust_multinomial(const dvar_vector& nsamp, const dvar_matrix& pred_comp, const dvar_matrix& obs_comp, const double& ncomp, const dvariable& mbin, const double& minSS, const dvariable& wgt
  //nsamp=vector of N's, pred_comp=matrix of predicted comps, obs_comp=matrix of observed comps, ncomp = number of yrs in matrix, mbin=number of bins, minSS=min N threshold, wgt_dat=scaling of N's
  RETURN_ARRAYS_INCREMENT();
  dvariable LkvalTmp;
  dvariable small_number=0.00001;
  LkvalTmp=0.0;
  dvar_matrix Eprime=elem_prod((1.0-obs_comp), obs_comp)+0.1/mbin; //E' of Francis 2011, p.1131
  dvar_vector nsamp_wgt=nsamp*wgt_dat;
  //cout<<nsamp_wgt<<endl;
  for (int ii=1; ii<=ncomp; ii++)
  {if (nsamp(ii)>=minSS)
    {LkvalTmp+= sum(0.5*log(Eprime(ii))-log(small_number+mfexp(elem_div((-square(obs_comp(ii)-pred_comp(ii))) , (Eprime(ii)*2.0/nsamp_wgt(ii)) ))) );
    }
  }
  RETURN_ARRAYS_DECREMENT();
  return LkvalTmp;


//----------------------------------------------------------------------------------
//----------------------------------------------------------------------------------
//Likelihood contribution: priors
FUNCTION  dvariable neg_log_prior(dvariable pred, const double& prior, dvariable var, int pdf)
  //prior=prior point estimate, var=variance (if negative, treated as CV in arithmetic space), pred=predicted value, pdf=prior type (1=none, 2=lognormal, 3=normal, 4=beta)
    dvariable LkvalTmp;
    dvariable alpha, beta, ab_iq;
    dvariable big_number=1e10;
    LkvalTmp=0.0;
```

```
    // compute generic pdf's
    switch(pdf) {
        case 1: //option to turn off prior
          LkvalTmp=0.0;
          break;
        case 2: // lognormal
          if(prior<=0.0) cout << "YIKES: Don't use a lognormal distn for a negative prior" << endl;
          else if(pred<=0) LkvalTmp=big_number=1e10;
          else {
            if(var<0.0) var=log(1.0+var*var) ;      // convert cv to variance on log scale
            LkvalTmp= 0.5*( square(log(pred/prior))/var + log(var) );
          }
        break;
        case 3: // normal
          if(var<0.0 && prior!=0.0) var=square(var*prior);       // convert cv to variance on observation scale
          else if(var<0.0 && prior==0.0) var=-var;               // cv not really appropriate if prior value equals zero
          LkvalTmp= 0.5*( square(pred-prior)/var + log(var) );
          break;
        case 4: // beta
          if(var<0.0) var=square(var*prior);          // convert cv to variance on observation scale
          if(prior<=0.0 || prior>=1.0) cout << "YIKES: Don't use a beta distn for a prior outside (0,1)" << endl;
          ab_iq=prior*(1.0-prior)/var - 1.0; alpha=prior*ab_iq; beta=(1.0-prior)*ab_iq;
          if(pred>=0 && pred<=1) LkvalTmp= (1.0-alpha)*log(pred)+(1.0-beta)*log(1.0-pred)-gammln(alpha+beta)+gammln(alpha)+gammln(beta);
          else LkvalTmp=big_number;
          break;
        default: // no such prior pdf currently available
          cout << "The prior must be either 1(lognormal), 2(normal), or 3(beta)." << endl;
          cout << "Presently it is " << pdf << endl;
          exit(0);
    }
    return LkvalTmp;


//-------------------------------------------------------------------------------------
//SDNR: age comp likelihood (assumes fits are done with the robust multinomial function)
FUNCTION dvariable sdnr_multinomial(const double& ncomp, const dvar_vector& ages, const dvar_vector& nsamp,
                                    const dvar_matrix& pred_comp, const dvar_matrix& obs_comp, const dvariable& wgt_dat)
  //ncomp=number of years of data, ages=vector of ages, nsamp=vector of N's,
  //pred_comp=matrix of predicted comps, obs_comp=matrix of observed comps, wgt_dat=likelihood weight for data source
  RETURN_ARRAYS_INCREMENT();
  dvariable SdnrTmp;
  dvar_vector o(1,ncomp);
  dvar_vector p(1,ncomp);
  dvar_vector ose(1,ncomp);
  dvar_vector res(1,ncomp);
  SdnrTmp=0.0;
  for (int ii=1; ii<=ncomp; ii++)
  {
    o(ii)=sum(elem_prod(ages,obs_comp(ii)));
    p(ii)=sum(elem_prod(ages,pred_comp(ii)));
    ose(ii)=sqrt((sum(elem_prod(square(ages),pred_comp(ii)))-square(p(ii)))/(nsamp(ii)*wgt_dat));
  }
  res=elem_div((o-p),ose);
  SdnrTmp=sqrt(sum(square(res-(sum(res)/ncomp))/(ncomp-1.0)));
  RETURN_ARRAYS_DECREMENT();
  return SdnrTmp;



//-------------------------------------------------------------------------------------
//SDNR: lognormal likelihood
FUNCTION dvariable sdnr_lognormal(const dvar_vector& pred, const dvar_vector& obs, const dvar_vector& cv, const dvariable& wgt_dat)
  //nyr=number of years of data, pred=vector of predicted data, obs=vector of observed data, cv=vector of cv's, wgt_dat=likelihood weight for data source
  RETURN_ARRAYS_INCREMENT();
  dvariable SdnrTmp;
  dvariable small_number=0.00001;
  dvariable n;
  dvar_vector res(cv.indexmin(),cv.indexmax());
  SdnrTmp=0.0;
  res=elem_div(log(elem_div(obs+small_number,pred+small_number)),sqrt(log(1+square(cv/wgt_dat))));
  n=cv.indexmax()-cv.indexmin()+1;
  SdnrTmp=sqrt(sum(square(res-(sum(res)/n))/(n-1.0)));
  RETURN_ARRAYS_DECREMENT();
  return SdnrTmp;

//-------------------------------------------------------------------------------------
REPORT_SECTION

  if (last_phase())
  {
      cout<<"start report"<<endl;
      get_weighted_current();
      cout<<"got weighted"<<endl;
      get_msy();
      cout<<"got msy"<<endl;
      get_miscellaneous_stuff();
      cout<<"got misc stuff"<<endl;
      get_per_recruit_stuff();
      get_effective_sample_sizes();

      grad_max=objective_function_value::pobjfun->gmax;
      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<<"*******************************************"<<endl;
  cout<<"--Start time: "<<ctime(&start)<<endl;
  cout<<"--Finish time: "<<ctime(&finish)<<endl;
  cout<<"--Runtime: ";
  cout<<hour<<" hours, "<<minute<<" minutes, "<<second<<" seconds"<<endl;
  cout << "--TotalLikelihood: " << fval << endl;
      cout<<"--Final gradient: "<<objective_function_value::pobjfun->gmax << endl;
  cout<<"*******************************************"<<endl;
```

```
    cout <<endl;
    cout << "><>--><>--><>--><>--><>--><>--><>--><>--><>--><>"  <<endl;
    cout <<"F status="<<FdF_msy_end<<endl;
    cout <<"Pop status="<<SdSSB_msy_end<<endl;
    cout << "h="<<steep<<"   R0="<<R0<<endl;
    cout << "><>--><>--><>--><>--><>--><>--><>--><>--><>--><>"  <<endl;

    report << "TotalLikelihood " << fval << endl;
    report << "N" << endl;
    report << N<<endl;
    report << "F" << endl;
    report << F <<endl;

    sdnr_lc_cH=sdnr_multinomial(nyr_cH_lenc, lenbins, nsamp_cH_lenc, pred_cH_lenc, obs_cH_lenc, w_lc_cH);
    sdnr_lc_cD=sdnr_multinomial(nyr_cD_lenc, lenbins, nsamp_cD_lenc, pred_cD_lenc, obs_cD_lenc, w_lc_cD);
    sdnr_lc_HB=sdnr_multinomial(nyr_HB_lenc, lenbins, nsamp_HB_lenc, pred_HB_lenc, obs_HB_lenc, w_lc_HB);

    sdnr_ac_cH=sdnr_multinomial(nyr_cH_agec, agebins_agec, nsamp_cH_agec, pred_cH_agec, obs_cH_agec, w_ac_cH);
    sdnr_ac_cD=sdnr_multinomial(nyr_cD_agec, agebins_agec, nsamp_cD_agec, pred_cD_agec, obs_cD_agec, w_ac_cD);
    sdnr_ac_HB=sdnr_multinomial(nyr_HB_agec, agebins_agec, nsamp_HB_agec, pred_HB_agec, obs_HB_agec, w_ac_HB);

    sdnr_I_cH=sdnr_lognormal(pred_cH_cpue, obs_cH_cpue, cH_cpue_cv, w_I_cH);
    sdnr_I_HB=sdnr_lognormal(pred_HB_cpue, obs_HB_cpue, HB_cpue_cv, w_I_HB);
    sdnr_I_GR=sdnr_lognormal(pred_GR_cpue, obs_GR_cpue, GR_cpue_cv, w_I_GR);


    //#################################################################################################
    //##  Passing parameters to vector for bounds check plotting
    //#################################################################################################
     Linf_out(8)=Linf; Linf_out(1,7)=set_Linf;
     K_out(8)=K; K_out(1,7)=set_K;
     t0_out(8)=t0; t0_out(1,7)=set_t0;
     len_cv_val_out(8)=len_cv_val; len_cv_val_out(1,7)=set_len_cv;
     log_R0_out(8)=log_R0; log_R0_out(1,7)=set_log_R0;
     M_constant_out(8)=M_constant; M_constant_out(1,7)=set_M_constant;
     steep_out(8)=steep; steep_out(1,7)=set_steep;
     rec_sigma_out(8)=rec_sigma; rec_sigma_out(1,7)=set_rec_sigma;
     R_autocorr_out(8)=R_autocorr; R_autocorr_out(1,7)=set_R_autocorr;

     selpar_L50_cH1_out(8)=selpar_L50_cH1; selpar_L50_cH1_out(1,7)=set_selpar_L50_cH1;
     selpar_slope_cH1_out(8)=selpar_slope_cH1; selpar_slope_cH1_out(1,7)=set_selpar_slope_cH1;
     selpar_L50_cH2_out(8)=selpar_L50_cH2; selpar_L50_cH2_out(1,7)=set_selpar_L50_cH2;
     selpar_slope_cH2_out(8)=selpar_slope_cH2; selpar_slope_cH2_out(1,7)=set_selpar_slope_cH2;
     selpar_L50_cH3_out(8)=selpar_L50_cH3; selpar_L50_cH3_out(1,7)=set_selpar_L50_cH3;
     selpar_slope_cH3_out(8)=selpar_slope_cH3; selpar_slope_cH3_out(1,7)=set_selpar_slope_cH3;

     selpar_L50_cD_out(8)=selpar_L50_cD; selpar_L50_cD_out(1,7)=set_selpar_L50_cD;
     selpar_slope_cD_out(8)=selpar_slope_cD; selpar_slope_cD_out(1,7)=set_selpar_slope_cD;
     selpar_afull_cD_out(8)=selpar_afull_cD; selpar_afull_cD_out(1,7)=set_selpar_afull_cD;
     selpar_sigma_cD_out(8)=selpar_sigma_cD; selpar_sigma_cD_out(1,7)=set_selpar_sigma_cD;

     selpar_L50_HB1_out(8)=selpar_L50_HB1; selpar_L50_HB1_out(1,7)=set_selpar_L50_HB1;
     selpar_slope_HB1_out(8)=selpar_slope_HB1; selpar_slope_HB1_out(1,7)=set_selpar_slope_HB1;
     selpar_L50_HB2_out(8)=selpar_L50_HB2; selpar_L50_HB2_out(1,7)=set_selpar_L50_HB2;
     selpar_slope_HB2_out(8)=selpar_slope_HB2; selpar_slope_HB2_out(1,7)=set_selpar_slope_HB2;
     selpar_L50_HB3_out(8)=selpar_L50_HB3; selpar_L50_HB3_out(1,7)=set_selpar_L50_HB3;
     selpar_slope_HB3_out(8)=selpar_slope_HB3; selpar_slope_HB3_out(1,7)=set_selpar_slope_HB3;

     log_q_cH_out(8)=log_q_cH; log_q_cH_out(1,7)=set_log_q_cH;
     log_q_HB_out(8)=log_q_HB; log_q_HB_out(1,7)=set_log_q_HB;
     log_q_GR_out(8)=log_q_GR; log_q_GR_out(1,7)=set_log_q_GR;

     log_avg_F_cH_out(8)=log_avg_F_cH; log_avg_F_cH_out(1,7)=set_log_avg_F_cH;
     log_avg_F_cD_out(8)=log_avg_F_cD; log_avg_F_cD_out(1,7)=set_log_avg_F_cD;
     log_avg_F_HB_out(8)=log_avg_F_HB; log_avg_F_HB_out(1,7)=set_log_avg_F_HB;
     log_avg_F_GR_out(8)=log_avg_F_GR; log_avg_F_GR_out(1,7)=set_log_avg_F_GR;
     log_avg_F_cH_D_out(8)=log_avg_F_cH_D; log_avg_F_cH_D_out(1,7)=set_log_avg_F_cH_D;
     log_avg_F_HB_D_out(8)=log_avg_F_HB_D; log_avg_F_HB_D_out(1,7)=set_log_avg_F_HB_D;
     log_avg_F_GR_D_out(8)=log_avg_F_GR_D; log_avg_F_GR_D_out(1,7)=set_log_avg_F_GR_D;
     F_init_out(8)=F_init; F_init_out(1,7)=set_F_init;

     log_rec_dev_out(styr_rec_dev, endyr_rec_dev)=log_rec_dev;
     log_F_dev_cH_out(styr_cH_L,endyr_cH_L)=log_F_dev_cH;
     log_F_dev_cD_out(styr_cD_L,endyr_cD_L)=log_F_dev_cD;
     log_F_dev_HB_out(styr_HB_L,endyr_HB_L)=log_F_dev_HB;
     log_F_dev_GR_out(styr_GR_L,endyr_GR_L)=log_F_dev_GR;
     log_F_dev_cH_D_out(styr_cH_D,endyr_cH_D)=log_F_dev_cH_D;
     log_F_dev_HB_D_out(styr_HB_D,endyr_HB_D)=log_F_dev_HB_D;
     log_F_dev_GR_D_out(styr_GR_D,endyr_GR_D)=log_F_dev_GR_D;

  #include "gag_make_Robject.cxx"   //Write R-compatible output. Comment line if not using admb2r.

} //endl last phase loop
```