S E D A R

Southeast Data, Assessment, and Review

==========

SEDAR-17-AW-11

AD Model Builder code to implement catch-age assessment model of vermilion snapper

Sustainable Fisheries Branch, National Marine Fisheries Service,
101 Pivers Island Rd, Beaufort NC 28516

September 2008

SEDAR is a Cooperative Initiative of:
The Caribbean Fishery Management Council
The Gulf of Mexico Fishery Management Council
The South Atlantic Fishery Management Council
NOAA Fisheries Southeast Regional Office
NOAA Fisheries Southeast Fisheries Science Center
The Atlantic States Marine Fisheries Commission
The Gulf States Marine Fisheries Commission

SEDAR Headquarters
The South Atlantic Fishery Management Council
4055 Faber Place #201
North Charleston, SC 29405
(843) 571-4366

This document contains four separate files:

I. The AD Model Builder code (Otter Research, 2005, ADMB Ver. 7.7.1) used to implement the catch-age assessment model of vermilion snapper
II. The data input file
III. A program called by ADMB to create an R object with data and results
IV. Routines for writing the R object

Only the first two files are necessary for running the model. The latter two are used for creating an R object populated with model input and output. To turn this feature off, comment out the relevant lines from vs3.tpl (see Globals Section and the very last line of code).

I. AD Model Builder code (vs3.tpl)

```
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##
//##  SEDAR17 Assessment: Vermilion Snapper, August 2008
//##
//##  Kyle Shertzer, NMFS, Beaufort Lab
//##  Kyle.Shertzer@noaa.gov
//##
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>

DATA_SECTION
//Create ascii file for output
//!!CLASS ofstream report1("rsresults.rep",ios::out);  //create file for output

!!cout << "Starting Vermilion Snapper Assessment Model" << endl;

// 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;

//4 periods of rec size regs: styr-91 no restrictions, 1992-98 10-inch TL, 1999-06 11-in TL, 2006-endyr 12-in TL
//2 periods of comm size regs: styr-91 no restrictions, 1992-endyr 12-in TL
init_int endyr_period1;
init_int endyr_period2;
init_int endyr_period3;

init_number limit_10in;  //10 inch limit in mm
init_number limit_11in;  //11 inch limit in mm
init_number limit_12in; //12 inch limit in mm

//Total number of ages
init_int nages;

// Vector of ages for age bins
init_ivector agebins(1,nages);

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

//Total number of length bins for each matrix and length bins used to compute mass in largest bin (plus group)
init_int nlenbins;
init_int nlenbins_plus;
// Vector of lengths for length bins (mm)(midpoint) and bins used in computation of plus group
init_ivector lenbins(1,nlenbins);
```

```
init_ivector lenbins_plus(1,nlenbins_plus);

//discard mortality constants
init_number set_Dmort_cHAL;
init_number set_Dmort_HB;
init_number set_Dmort_MRFSS;

//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;
//exclude these years from end of time series for computing bias correction
init_number BiasCor_exclude_yrs;
// Von Bert parameters in TL
init_number set_Linf;
init_number set_K;
init_number set_t0;
//CV of length at age
init_number set_len_cv;
//TL(mm)-FL(mm) relationship: FL=a+b*TL
init_number lenpar_TL2FL_a;
init_number lenpar_TL2FL_b;
//TL(mm)-weight(whole weight in g) relationship: W=aL^b
init_number wgtpar_a;
init_number wgtpar_b;
//Female maturity and proportion female at age
init_vector maturity_f_obs(1,nages);           //total maturity of females
init_vector prop_f_obs(1,nages);               //proportion female at age
//Fecundity-length relationship: Eggs/batch=aFL^b (length in FL); batches in number batches per yr
init_number fecpar_batches;
init_number fecpar_a;
init_number fecpar_b;

init_number spawn_time_frac; //time of year of peak spawning, as a fraction of the year
init_number fecpar_scale;    //could be used for scaling annual egg production (10^X eggs)
// Natural mortality
init_vector set_M(1,nages); //age-dependent: used in model
init_number set_M_constant; //age-independent: used only for MSST
//Spawner-recruit parameters (Initial guesses or fixed values)
init_number set_steep;
init_number set_log_R0;
init_number set_R_autocorr;
//Set initial conditions, treated as either initial guesses or fixed
init_number set_R1_mult;
init_number set_B1dB0;

//#########################################################################
//###MARMAP FL snapper trap index#################################
//CPUE
init_int styr_FST_cpue;
init_int endyr_FST_cpue;
init_vector obs_FST_cpue(styr_FST_cpue,endyr_FST_cpue);   //Observed CPUE
init_vector FST_cpue_cv(styr_FST_cpue,endyr_FST_cpue);    //CV of cpue
//Length Compositions (1 cm bins)
init_int styr_FST_lenc;
init_int endyr_FST_lenc;
init_vector nsamp_FST_lenc(styr_FST_lenc,endyr_FST_lenc);
init_matrix obs_FST_lenc(styr_FST_lenc,endyr_FST_lenc,1,nlenbins);

//###############MARMAP Chevron trap index#################################
//CPUE
init_int styr_CVT_cpue;
init_int endyr_CVT_cpue;
init_vector obs_CVT_cpue(styr_CVT_cpue,endyr_CVT_cpue);   //Observed CPUE
init_vector CVT_cpue_cv(styr_CVT_cpue,endyr_CVT_cpue);    //cv of cpue
////Length Compositions (1cm bins)
//init_int styr_CVT_lenc;
//init_int endyr_CVT_lenc;
//init_vector nsamp_CVT_lenc(styr_CVT_lenc,endyr_CVT_lenc);
//init_matrix obs_CVT_lenc(styr_CVT_lenc,endyr_CVT_lenc,1,nlenbins);
//Age Compositions
init_int styr_CVT_agec;
init_int endyr_CVT_agec;
init_vector nsamp_CVT_agec(styr_CVT_agec,endyr_CVT_agec);
```

```
init_matrix obs_CVT_agec(styr_CVT_agec,endyr_CVT_agec,1,nages);

//###################Commercial Hook and Line fishery ########################
//CPUE
init_int styr_HAL_cpue;
init_int endyr_HAL_cpue;
init_vector obs_HAL_cpue(styr_HAL_cpue,endyr_HAL_cpue);//Observed CPUE
init_vector HAL_cpue_cv(styr_HAL_cpue,endyr_HAL_cpue); //CV of cpue

// Landings  (1000 lb whole weight)
init_int styr_cHAL_L;
init_int endyr_cHAL_L;
init_vector obs_cHAL_L(styr_cHAL_L,endyr_cHAL_L); //vector of observed landings by year
init_vector cHAL_L_cv(styr_cHAL_L,endyr_cHAL_L);    //vector of CV of landings by year

// Discards (1000 fish)
init_int styr_cHAL_D;
init_int endyr_cHAL_D;
init_vector obs_cHAL_released(styr_cHAL_D,endyr_cHAL_D); //vector of observed releases by year, multiplied by discard mortality for fitting
init_vector cHAL_D_cv(styr_cHAL_D,endyr_cHAL_D);    //vector of CV of discards by year

// Length Compositions (1 cm bins)
init_int styr_cHAL_lenc;
init_int endyr_cHAL_lenc;
init_vector nsamp_cHAL_lenc(styr_cHAL_lenc,endyr_cHAL_lenc);
init_matrix obs_cHAL_lenc(styr_cHAL_lenc,endyr_cHAL_lenc,1,nlenbins);
// Age Compositions
init_int nyr_cHAL_agec;
init_ivector yrs_cHAL_agec(1,nyr_cHAL_agec);
init_vector nsamp_cHAL_agec(1,nyr_cHAL_agec);
init_matrix obs_cHAL_agec(1,nyr_cHAL_agec,1,nages);

// Length Compositions of Discards (1 cm bins)
init_int nyr_cHAL_D_lenc;
init_ivector yrs_cHAL_D_lenc(1,nyr_cHAL_D_lenc);
init_vector nsamp_cHAL_D_lenc(1,nyr_cHAL_D_lenc);
init_matrix obs_cHAL_D_lenc(1,nyr_cHAL_D_lenc,1,nlenbins);

//#########################################################################
//##Commercial Combined gears fishery (mostly EEZ trawl 1976-1988, mostly traps or state-water trawl in other yrs
// Landings (1000 lb whole weight)
init_int styr_cCMB_L;
init_int endyr_cCMB_L;
init_vector obs_cCMB_L(styr_cCMB_L,endyr_cCMB_L);
init_vector cCMB_L_cv(styr_cCMB_L,endyr_cCMB_L);    //vector of CV of landings by year
// Length Compositions (1 cm bins, data from trawls)
init_int nyr_cCMB_lenc;
init_ivector yrs_cCMB_lenc(1,nyr_cCMB_lenc);
init_vector nsamp_cCMB_lenc(1,nyr_cCMB_lenc);
init_matrix obs_cCMB_lenc(1,nyr_cCMB_lenc,1,nlenbins);

//###################Commercial historic trawl fishery ########################
// Landings  (1000 lb whole weight)
init_int styr_cHTR_L;
init_int endyr_cHTR_L;
init_vector obs_cHTR_L(styr_cHTR_L,endyr_cHTR_L); //vector of observed landings by year
init_vector cHTR_L_cv(styr_cHTR_L,endyr_cHTR_L);    //vector of CV of landings by year


//#########################################################################
//#################################Headboat fishery ########################################
//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
// Landings (1000 lb whole weight)
init_int styr_HB_L;
init_int endyr_HB_L;
init_vector obs_HB_L(styr_HB_L,endyr_HB_L);
init_vector HB_L_cv(styr_HB_L,endyr_HB_L);
//// Discards (1000s)
init_int styr_HB_D;
init_int endyr_HB_D;
init_vector obs_HB_released(styr_HB_D,endyr_HB_D); //vector of observed releases by year, multiplied by discard mortality for fitting
init_vector HB_D_cv(styr_HB_D,endyr_HB_D);    //vector of CV of discards by year
// Length Compositions (1 cm bins) of landings
init_int styr_HB_lenc;
init_int endyr_HB_lenc;
init_vector nsamp_HB_lenc(styr_HB_lenc,endyr_HB_lenc);
```

```
init_matrix obs_HB_lenc(styr_HB_lenc,endyr_HB_lenc,1,nlenbins);
// Age compositions of landings
init_int nyr_HB_agec;
init_ivector yrs_HB_agec(1,nyr_HB_agec);
init_vector nsamp_HB_agec(1,nyr_HB_agec);
init_matrix obs_HB_agec(1,nyr_HB_agec,1,nages);
// Length Compositions (1 cm bins) of discards
init_int styr_HB_D_lenc;
init_int endyr_HB_D_lenc;
init_vector nsamp_HB_D_lenc(styr_HB_D_lenc,endyr_HB_D_lenc);
init_matrix obs_HB_D_lenc(styr_HB_D_lenc,endyr_HB_D_lenc,1,nlenbins);

//#########################################################################
//############################MRFSS landings ##############################
//CPUE
init_int styr_MRFSS_cpue;
init_int endyr_MRFSS_cpue;
init_vector obs_MRFSS_cpue(styr_MRFSS_cpue,endyr_MRFSS_cpue);//Observed CPUE
init_vector MRFSS_cpue_cv(styr_MRFSS_cpue,endyr_MRFSS_cpue); //CV of cpue
// Landings (1000 lb whole weight)
init_int styr_MRFSS_L;
init_int endyr_MRFSS_L;
init_vector obs_MRFSS_L(styr_MRFSS_L,endyr_MRFSS_L);
init_vector MRFSS_L_cv(styr_MRFSS_L,endyr_MRFSS_L);
// Discards (1000s)
init_int styr_MRFSS_D;
init_int endyr_MRFSS_D;
init_vector obs_MRFSS_released(styr_MRFSS_D,endyr_MRFSS_D); //vector of observed releases by year, multiplied by discard mortality for fitting
init_vector MRFSS_D_cv(styr_MRFSS_D,endyr_MRFSS_D);    //vector of CV of discards by year
// Length Compositions (1 cm bins)
init_int styr_MRFSS_lenc;
init_int endyr_MRFSS_lenc;
init_vector nsamp_MRFSS_lenc(styr_MRFSS_lenc,endyr_MRFSS_lenc);
init_matrix obs_MRFSS_lenc(styr_MRFSS_lenc,endyr_MRFSS_lenc,1,nlenbins);
// Age Compositions
init_int styr_MRFSS_agec;
init_int endyr_MRFSS_agec;
init_vector nsamp_MRFSS_agec(styr_MRFSS_agec,endyr_MRFSS_agec);
init_matrix obs_MRFSS_agec(styr_MRFSS_agec,endyr_MRFSS_agec,1,nages);

//#########################################################################
//###################Parameter values and initial guesses ##################################
//--weights for likelihood components----------------------------------------------------------------------
init_number set_w_L;
init_number set_w_D;
init_number set_w_lc;
init_number set_w_ac;
init_number set_w_I_FST;
init_number set_w_I_CVT;
init_number set_w_I_HAL;
init_number set_w_I_HB;
init_number set_w_I_MRFSS;
init_number set_w_R;            //for fitting S-R curve
init_number set_w_fullF;        //penalty for any fullF>3
init_number set_w_R_init;       //additional constraint on early years recruitment
init_number set_w_R_end;        //additional constraint on ending years recruitment
init_number set_w_F;            //additional constraint on F in last few years
init_number set_w_B1dB0;        // weight on B1/B0
init_number set_w_cvlen_dev;        //penalty on cv deviations at age
init_number set_w_cvlen_diff;       //penalty on first difference of cv deviations at age

//Initial guess for recreational (HB and MRFSS from Salt-Water Angling Report) landings multiplicative bias
init_number set_L_early_bias;
//Initial guess for rate of increase on q
init_number set_q_rate;

//--index catchability-------------------------------------------------------------------------------------------------
init_number set_logq_FST;     //catchability coefficient (log) for MARMAP FST
init_number set_logq_CVT;     //catchability coefficient (log) for MARMAP CVT
init_number set_logq_HAL;     //catchability coefficient (log) for commercial logbook CPUE index
init_number set_logq_HB;      //catchability coefficient (log) for the headboat index
init_number set_logq_MRFSS;   //catchability coefficient (log) for MRFSS CPUE index

//--F's-------------------------------
init_number set_log_avg_F_cHAL;
init_number set_log_avg_F_cHTR;
init_number set_log_avg_F_cCMB;
init_number set_log_avg_F_HB;
init_number set_log_avg_F_MRFSS;
```

```
//--discard F's----------------------
init_number set_log_avg_F_cHAL_D;
init_number set_log_avg_F_HB_D;
init_number set_log_avg_F_MRFSS_D;

//Initial guesses of estimated selectivity parameters
init_number set_selpar_L50_FST;
init_number set_selpar_slope_FST;
init_number set_selpar_L502_FST;
init_number set_selpar_slope2_FST;

init_number set_selpar_L50_CVT;
init_number set_selpar_slope_CVT;
init_number set_selpar_L502_CVT;
init_number set_selpar_slope2_CVT;

init_number set_selpar_L50_cHAL1;
init_number set_selpar_slope_cHAL1;
init_number set_selpar_L502_cHAL1;
init_number set_selpar_slope2_cHAL1;
init_number set_selpar_L50_cHAL2;
init_number set_selpar_slope_cHAL2;
init_number set_selpar_L502_cHAL2;
init_number set_selpar_slope2_cHAL2;

init_vector set_sel_cHAL_D_2(1,nages);
init_number set_selpar_Age1_cHAL_D2;
init_number set_selpar_Age2_cHAL_D2;
//init_number set_selpar_L50_cHAL_D2;
//init_number set_selpar_slope_cHAL_D2;
//init_number set_selpar_L502_cHAL_D2;
//init_number set_selpar_slope2_cHAL_D2;

init_number set_selpar_L50_cCMB1;
init_number set_selpar_slope_cCMB1;
init_number set_selpar_L502_cCMB1;
init_number set_selpar_slope2_cCMB1;
//init_number set_selpar_L50_cCMB2;
//init_number set_selpar_slope_cCMB2;
//init_number set_selpar_L502_cCMB2;
//init_number set_selpar_slope2_cCMB2;

init_number set_selpar_L50_HB1;
init_number set_selpar_slope_HB1;
init_number set_selpar_L502_HB1;
init_number set_selpar_slope2_HB1;
init_number set_selpar_L50_HB2;
init_number set_selpar_slope_HB2;
init_number set_selpar_L502_HB2;
init_number set_selpar_slope2_HB2;
init_number set_selpar_L50_HB3;
init_number set_selpar_slope_HB3;
init_number set_selpar_L502_HB3;
init_number set_selpar_slope2_HB3;
init_number set_selpar_L50_HB4;
init_number set_selpar_slope_HB4;
init_number set_selpar_L502_HB4;
init_number set_selpar_slope2_HB4;

init_vector set_sel_HB_D_2(1,nages);
init_vector set_sel_HB_D_3(1,nages);
init_number set_selpar_Age1_HB_D3;
init_number set_selpar_Age2_HB_D3;
init_vector set_sel_HB_D_4(1,nages);
init_number set_selpar_Age1_HB_D4;
init_number set_selpar_Age2_HB_D4;
//init_number set_selpar_L50_HB_D3;
//init_number set_selpar_slope_HB_D3;
//init_number set_selpar_L502_HB_D3;
//init_number set_selpar_slope2_HB_D3;
//init_number set_selpar_L50_HB_D4;
//init_number set_selpar_slope_HB_D4;
//init_number set_selpar_L502_HB_D4;
//init_number set_selpar_slope2_HB_D4;

init_number set_selpar_L50_MRFSS3;
init_number set_selpar_slope_MRFSS3;
init_number set_selpar_L502_MRFSS3;
init_number set_selpar_slope2_MRFSS3;
init_number set_selpar_L50_MRFSS4;
```

```
init_number set_selpar_slope_MRFSS4;
init_number set_selpar_L502_MRFSS4;
init_number set_selpar_slope2_MRFSS4;

//threshold sample sizes for length comps
init_number minSS_FST_lenc;
//init_number minSS_CVT_lenc;
init_number minSS_cHAL_lenc;
init_number minSS_cHAL_D_lenc;
init_number minSS_cCMB_lenc;
init_number minSS_HB_lenc;
init_number minSS_HB_D_lenc;
init_number minSS_MRFSS_lenc;

//threshold sample sizes for age comps
init_number minSS_CVT_agec;
init_number minSS_cHAL_agec;
init_number minSS_HB_agec;
init_number minSS_MRFSS_agec;

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

//proportion of length comp mass below size limit considered when matching length comp
//note: these need length comp and age comp data to be estimable
init_number set_p_lenc_cHAL;
init_number set_p_lenc_cCMB;
init_number set_p_lenc_HB2;
init_number set_p_lenc_HB3;
init_number set_p_lenc_HB4;
init_number set_p_lenc_MRFSS2;
init_number set_p_lenc_MRFSS3;
init_number set_p_lenc_MRFSS4;

init_number set_p_lenc_cHAL_D;
init_number set_p_lenc_HB_D2;
init_number set_p_lenc_HB_D3;
init_number set_p_lenc_HB_D4;

// #######Indices for year(iyear), age(iage),length(ilen) ###############
int iyear;
int iage;
int ilen;

init_number end_of_data_file;
//this section MUST BE INDENTED!!!
 LOCAL_CALCS
  if(end_of_data_file!=999)
  {
   for(iyear=1; iyear<=100; iyear++)
   {
    cout << "*** WARNING: Data File NOT READ CORRECTLY ****" << endl;
    cout << "" <<endl;
   }
  }
  else
  {
   cout << "Data File read correctly" << endl;
  }
 END_CALCS


//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
PARAMETER_SECTION
////--------------Growth--------------------------------------------------------------------------------

  //init_bounded_number Linf(400,1200,2);
  //init_bounded_number K(0.05,0.6,2);
  //init_bounded_number t0(-5.0,0.0,2);
  number Linf;
  number K;
  number t0;
  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

  matrix len_cHAL_mm(styr,endyr,1,nages);       //mean length at age of cHAL landings in mm (may differ from popn mean)
```

```
matrix wgt_cHAL_klb(styr,endyr,1,nages);      //whole wgt of cHAL landings in 1000 lb
matrix len_cHTR_mm(styr,endyr,1,nages);       //mean length at age of cHTR landings in mm (may differ from popn mean)
matrix wgt_cHTR_klb(styr,endyr,1,nages);      //whole wgt of cHTR landings in 1000 lb
matrix len_cCMB_mm(styr,endyr,1,nages);       //mean length at age of cCMB landings in mm (may differ from popn mean)
matrix wgt_cCMB_klb(styr,endyr,1,nages);      //whole wgt of cCMB 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 wgt_HB_klb(styr,endyr,1,nages);        //whole wgt of HB landings in 1000 lb
matrix len_MRFSS_mm(styr,endyr,1,nages);      //mean length at age of MRFSS landings in mm (may differ from popn mean)
matrix wgt_MRFSS_klb(styr,endyr,1,nages);     //whole wgt of MRFSS landings in 1000 lb

matrix len_cHAL_D_mm(styr,endyr,1,nages);     //mean length at age of cHAL discards in mm (may differ from popn mean)
matrix wgt_cHAL_D_klb(styr,endyr,1,nages);    //whole wgt of cHAL discards in 1000 lb
matrix len_HB_D_mm(styr,endyr,1,nages);       //mean length at age of cHAL discards in mm (may differ from popn mean)
matrix wgt_HB_D_klb(styr,endyr,1,nages);      //whole wgt of cHAL discards in 1000 lb
matrix len_MRFSS_D_mm(styr,endyr,1,nages);    //mean length at age of cHAL discards in mm (may differ from popn mean)
matrix wgt_MRFSS_D_klb(styr,endyr,1,nages);   //whole wgt of cHAL discards in 1000 lb


vector meanlen_TL(1,nages);       //mean Total length (mm) at age
vector meanlen_FL(1,nages);       //mean Fork length (mm) at age
vector fecundity(1,nages);        // annual fec at age per mat female: fecpar_batches*(fecpar_a*FL^fecpar_b)

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

matrix lenprob(1,nages,1,nlenbins);          //distn of size at age (age-length key, 1 cm bins) in population
matrix lenprob_plus(1,nages,1,nlenbins_plus); //used to compute mass in last length bin (a plus group)

matrix lenprob_cHAL1(1,nages,1,nlenbins);    //distn of size at age in cHAL period 1
matrix lenprob_cHAL2(1,nages,1,nlenbins);    //distn of size at age in cHAL period 2
matrix lenprob_cCMB1(1,nages,1,nlenbins);    //distn of size at age in cHAL period 1
matrix lenprob_cCMB2(1,nages,1,nlenbins);    //distn of size at age in cHAL period 2
matrix lenprob_HB1(1,nages,1,nlenbins);      //distn of size at age in HB period 1
matrix lenprob_HB2(1,nages,1,nlenbins);      //distn of size at age in HB period 2
matrix lenprob_HB3(1,nages,1,nlenbins);      //distn of size at age in HB period 3
matrix lenprob_HB4(1,nages,1,nlenbins);      //distn of size at age in HB period 4
matrix lenprob_MRFSS1(1,nages,1,nlenbins);   //distn of size at age in MRFSS period 1
matrix lenprob_MRFSS2(1,nages,1,nlenbins);   //distn of size at age in MRFSS period 2
matrix lenprob_MRFSS3(1,nages,1,nlenbins);   //distn of size at age in MRFSS period 3
matrix lenprob_MRFSS4(1,nages,1,nlenbins);   //distn of size at age in MRFSS period 4


matrix lenprob_cHAL_D2(1,nages,1,nlenbins);  //distn of size at age in cHAL discards comm period 2
matrix lenprob_HB_D2(1,nages,1,nlenbins);    //distn of size at age in HB discards rec period 2. not used for comp data, just avg wgt.
matrix lenprob_HB_D3(1,nages,1,nlenbins);    //distn of size at age in HB discards rec period 3
matrix lenprob_HB_D4(1,nages,1,nlenbins);    //distn of size at age in HB discards rec period 4


init_bounded_number log_len_cv(-5,-0.3,3);
//init_bounded_dev_vector log_len_cv_dev(1,nages,-2,2,3)
vector len_cv(1,nages);

number age_limit_10in; //age corresponding to 10-inch size limit, given mean growth curve
number age_limit_11in; //age corresponding to 11-inch size limit, given mean growth curve
number age_limit_12in; //age corresponding to 12-inch size limit, given mean growth curve

//----Predicted length and age compositions
matrix pred_FST_lenc(styr_FST_lenc,endyr_FST_lenc,1,nlenbins);
//matrix pred_CVT_lenc(styr_CVT_lenc,endyr_CVT_lenc,1,nlenbins);
matrix pred_cHAL_lenc(styr_cHAL_lenc,endyr_cHAL_lenc,1,nlenbins);
matrix pred_cHAL_D_lenc(1,nyr_cHAL_D_lenc,1,nlenbins);
matrix pred_cCMB_lenc(1,nyr_cCMB_lenc,1,nlenbins);
matrix pred_HB_lenc(styr_HB_lenc,endyr_HB_lenc,1,nlenbins);
matrix pred_HB_D_lenc(styr_HB_D_lenc,endyr_HB_D_lenc,1,nlenbins);
matrix pred_MRFSS_lenc(styr_MRFSS_lenc,endyr_MRFSS_lenc,1,nlenbins);

//p_lenc_fishery pars require age comp and length comp data for estimation
//init_bounded_number p_lenc_cHAL(0.0,1.0,3);
//init_bounded_number p_lenc_cCMB(0.0,1.0,3);
//init_bounded_number p_lenc_HB2(0.0,1.0,3);
//init_bounded_number p_lenc_HB3(0.0,1.0,3);
//init_bounded_number p_lenc_HB4(0.0,1.0,3);
//init_bounded_number p_lenc_MRFSS2(0.0,1.0,3);
//init_bounded_number p_lenc_MRFSS3(0.0,1.0,3);
//init_bounded_number p_lenc_MRFSS4(0.0,1.0,3);
number p_lenc_cHAL;
```

```
  number p_lenc_cCMB;
  number p_lenc_HB2;
  number p_lenc_HB3;
  number p_lenc_HB4;
  number p_lenc_MRFSS2;
  number p_lenc_MRFSS3;
  number p_lenc_MRFSS4;

  //init_bounded_number p_lenc_cHAL_D(0.0,1.0,3);
  //init_bounded_number p_lenc_HB_D3(0.0,1.0,3);
  //init_bounded_number p_lenc_HB_D4(0.0,1.0,3);
  number p_lenc_cHAL_D;
  number p_lenc_HB_D2; //no comp data in this period, this par only used for avg weight
  number p_lenc_HB_D3;
  number p_lenc_HB_D4;

  matrix pred_CVT_agec(styr_CVT_agec,endyr_CVT_agec,1,nages);
  matrix ErrorFree_CVT_agec(styr_CVT_agec,endyr_CVT_agec,1,nages); //age comps prior to applying ageing error matrix
  matrix pred_cHAL_agec(1,nyr_cHAL_agec,1,nages);
  matrix ErrorFree_cHAL_agec(1,nyr_cHAL_agec,1,nages);
  matrix pred_HB_agec(1,nyr_HB_agec,1,nages);
  matrix ErrorFree_HB_agec(1,nyr_HB_agec,1,nages);
  matrix pred_MRFSS_agec(styr_MRFSS_agec,endyr_MRFSS_agec,1,nages);
  matrix ErrorFree_MRFSS_agec(styr_MRFSS_agec,endyr_MRFSS_agec,1,nages);

  //nsamp_X_allyr vectors used only for R output of comps with nonconsecutive yrs, given sample size cutoffs
  vector nsamp_FST_lenc_allyr(styr,endyr);
  //vector nsamp_CVT_lenc_allyr(styr,endyr);
  vector nsamp_cHAL_lenc_allyr(styr,endyr);
  vector nsamp_cHAL_D_lenc_allyr(styr,endyr);
  vector nsamp_cCMB_lenc_allyr(styr,endyr);
  vector nsamp_HB_lenc_allyr(styr,endyr);
  vector nsamp_HB_D_lenc_allyr(styr,endyr);
  vector nsamp_MRFSS_lenc_allyr(styr,endyr);
  vector nsamp_CVT_agec_allyr(styr,endyr);
  vector nsamp_cHAL_agec_allyr(styr,endyr);
  vector nsamp_HB_agec_allyr(styr,endyr);
  vector nsamp_MRFSS_agec_allyr(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, SSB, and cpue
  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
  //init_bounded_number log_R1(5,20,1);       //log(Recruits) in styr
  sdreport_vector SSB(styr,endyr);         ///Total egg production by year
  sdreport_vector rec(styr,endyr+1);       //Recruits by year
  vector prop_f(1,nages);                  //Proportion female by age
  vector maturity_f(1,nages);              //Proportion of female mature at age
  vector reprod(1,nages);

//---Stock-Recruit Function (Beverton-Holt, steepness parameterization)----------
  init_bounded_number log_R0(5,20,1);       //log(virgin Recruitment)
  sdreport_number R0;
  //init_bounded_number steep(0.25,0.95,3);     //steepness
  number steep;  //uncomment to fix steepness, comment line directly above
  init_bounded_dev_vector log_dev_N_rec(styr_rec_dev,endyr,-3,3,2); //log recruitment deviations
  vector log_dev_R(styr,endyr+1);          //used in output. equals zero except for yrs in log_dev_N_rec
  number var_rec_dev;                      //variance of log recruitment deviations
                                  //Estimate from yrs with unconstrainted S-R(XXXX-XXXX)
  number BiasCor;                          //Bias correction in equilibrium recruits
  init_bounded_number R_autocorr(0.0,1.0,3);  //autocorrelation in SR
  //number R_autocorr
  sdreport_number R_autocorr_sd;
  sdreport_number steep_sd;                //steepness for stdev report
  number S0;                               //equal to spr_F0*R0 = virgin SSB
  number B0;                               //equal to bpr_F0*R0 = virgin B
  number B1dB0;                            //B1dB0 treated as input here (virgin stock)
  //init_bounded_number R1_mult(0.05,1.0,1);   //R(styr)=R1_mult*R0
  number R1_mult;                          //equals B1dB0 under assumption of virgin age structure
  number R1;                               //Recruits in styr
  number R_virgin;                         //unfished recruitment with bias correction
  sdreport_number S1S0;                    //SSB(styr) / virgin SSB
  sdreport_number popstatus;               //SSB(endyr) / virgin SSB

//---Selectivity-----------------------------------------------------------------------------
//MARMAP FST-------------------------------------------------
  matrix sel_FST(styr,endyr,1,nages);
  //init_bounded_number selpar_L50_FST(0.1,8.0,1);
  //init_bounded_number selpar_slope_FST(0.5,12.0,1);
```

```
  //init_bounded_number selpar_L502_FST(1.0,6.0,3); //additive with L50
  //init_bounded_number selpar_slope2_FST(0.0,12.0,3);
  number selpar_L50_FST;
  number selpar_slope_FST;
  number selpar_L502_FST; //additive with L50
  number selpar_slope2_FST;
  vector sel_FST_vec(1,nages);

 //MARMAP CVT----------------------------------------------------
  matrix sel_CVT(styr,endyr,1,nages);
  init_bounded_number selpar_L50_CVT(0.1,8.0,1);
  //init_bounded_number selpar_slope_CVT(0.5,12.0,1);
  number selpar_slope_CVT;
  init_bounded_number selpar_L502_CVT(1.0,6.0,3);
  init_bounded_number selpar_slope2_CVT(0.0,12.0,3);
  vector sel_CVT_vec(1,nages);

 //Commercial handline-------------------------------------------------
  matrix sel_cHAL(styr,endyr,1,nages);
  init_bounded_number selpar_L50_cHAL1(0.1,8.0,1);
  init_bounded_number selpar_slope_cHAL1(0.5,12.0,1); //period 1
  //number selpar_slope_cHAL1; //period 1
  //number selpar_L50_cHAL1;
  number selpar_slope2_cHAL1; //period 1
  number selpar_L502_cHAL1;

  init_bounded_number selpar_L50_cHAL2(0.1,8.0,1);
  //init_bounded_number selpar_slope_cHAL2(0.5,12.0,1); //period 2
  number selpar_slope_cHAL2; //period 2
  //number selpar_L50_cHAL2;
  number selpar_slope2_cHAL2; //period 2
  number selpar_L502_cHAL2;

  //init_bounded_dev_vector selpar_L50_cHAL_dev(styr_cHAL_lenc,endyr_period1,-5,5,3);
  vector sel_cHAL_1(1,nages); //sel in period 1
  vector sel_cHAL_2(1,nages); //sel in period 2

 //Commercial handline Discards----------------------------------------------------
  matrix sel_cHAL_D(styr,endyr,1,nages);
  //number selpar_slope_cHAL_D1; //period 1
  //number selpar_L50_cHAL_D1;
  //number selpar_slope2_cHAL_D1; //period 1
  //number selpar_L502_cHAL_D1;

  number selpar_Age1_cHAL_D2;
  //init_bounded_number selpar_Age1_cHAL_D2(0.0,1.0,1); //period 2
  init_bounded_number selpar_Age2_cHAL_D2(0.0,1.0,1); //period 2

  //init_bounded_number selpar_L50_cHAL_D2(0.1,8.0,1); //period 2
  //init_bounded_number selpar_slope_cHAL_D2(0.5,12.0,1);
  //init_bounded_number selpar_L502_cHAL_D2(1.0,6.0,3);
  //init_bounded_number selpar_slope2_cHAL_D2(0.0,12.0,3);
  //number selpar_slope_cHAL_D2; //period 2
  //number selpar_L50_cHAL_D2;
  //number selpar_slope2_cHAL_D2; //period 2
  //number selpar_L502_cHAL_D2;

  //init_bounded_dev_vector selpar_L50_cHAL_dev(styr_cHAL_lenc,endyr_period1,-5,5,3);
  //vector sel_cHAL_D_1(1,nages); //sel in period 1
  vector sel_cHAL_D_2(1,nages); //sel in period 2

 //Commercial combined gears-------------------------------------------------
  matrix sel_cCMB(styr,endyr,1,nages);
  //init_bounded_number selpar_L50_cCMB1(0.1,8.0,1);
  //init_bounded_number selpar_slope_cCMB1(0.5,12.0,1);
  //init_bounded_number selpar_L502_cCMB1(1.0,6.0,3);
  //init_bounded_number selpar_slope2_cCMB1(0.0,12.0,3);
  number selpar_L50_cCMB1;
  number selpar_slope_cCMB1;
  number selpar_L502_cCMB1;
  number selpar_slope2_cCMB1;

  number selpar_L50_cCMB2;
  number selpar_slope_cCMB2;
  number selpar_L502_cCMB2;
  number selpar_slope2_cCMB2;

  vector sel_cCMB_1(1,nages); //sel vector
  vector sel_cCMB_2(1,nages); //sel vector
```

```
//Commercial historic trawl assumed the same as CMB_1-----------------------------------------------
  matrix sel_cHTR(styr,endyr,1,nages);
  //vector sel_cHTR_vec(1,nages); //sel vector

//Headboat-----------------------------------------------
  matrix sel_HB(styr,endyr,1,nages);
  init_bounded_number selpar_L50_HB1(0.1,8.0,1);
  init_bounded_number selpar_slope_HB1(0.5,12.0,1); //period 1
  //number selpar_slope_HB1; //period 1
  //number selpar_L50_HB1;
  number selpar_slope2_HB1; //period 1
  number selpar_L502_HB1;

  init_bounded_number selpar_L50_HB2(0.1,8.0,1);
  //init_bounded_number selpar_slope_HB2(0.5,12.0,1); //period 2
  number selpar_slope_HB2; //period 2
  //number selpar_L50_HB2;
  number selpar_slope2_HB2; //period 2
  number selpar_L502_HB2;

  init_bounded_number selpar_L50_HB3(0.1,8.0,1);
  //init_bounded_number selpar_slope_HB3(0.5,12.0,1); //period 3
  number selpar_slope_HB3; //period 3
  //number selpar_L50_HB3;
  number selpar_slope2_HB3; //period 3
  number selpar_L502_HB3;

  init_bounded_number selpar_L50_HB4(0.1,8.0,1);
  //init_bounded_number selpar_slope_HB4(0.5,12.0,1); //period 4
  number selpar_slope_HB4; //period 4
  //number selpar_L50_HB4;
  number selpar_slope2_HB4; //period 4
  number selpar_L502_HB4;

  //init_bounded_dev_vector selpar_L50_HB_dev(styr_HB_lenc,endyr_period1,-5,5,3);
  vector sel_HB_1(1,nages); //sel in period 1
  vector sel_HB_2(1,nages); //sel in period 2
  vector sel_HB_3(1,nages); //sel in period 3
  vector sel_HB_4(1,nages); //sel in period 4

//Headboat  Discards selectivity-----------------------------------------------
  matrix sel_HB_D(styr,endyr,1,nages);

  //number selpar_L50_HB_D2;
  //number selpar_slope_HB_D2; //period 2
  //number selpar_slope2_HB_D2;
  //number selpar_L502_HB_D2;

  number selpar_Age1_HB_D3; //period 1,2
  number selpar_Age1_HB_D4; //period 1,2
  //init_bounded_number selpar_Age1_HB_D3(0.0,1.0,1); //period 3
  init_bounded_number selpar_Age2_HB_D3(0.0,1.0,1); //period 3
  //init_bounded_number selpar_Age1_HB_D4(0.0,1.0,1); //period 4
  init_bounded_number selpar_Age2_HB_D4(0.0,1.0,1); //period 4

//  init_bounded_number selpar_L50_HB_D3(0.1,8.0,1);
//  init_bounded_number selpar_slope_HB_D3(0.5,12.0,1); //period 3
//  init_bounded_number selpar_slope2_HB_D3(1.0,6.0,3);
//  init_bounded_number selpar_L502_HB_D3(0.0,12.0,3);

//  init_bounded_number selpar_L50_HB_D4(0.1,8.0,1);
//  init_bounded_number selpar_slope_HB_D4(0.5,12.0,1); //period 4
//  init_bounded_number selpar_slope2_HB_D4(1.0,6.0,3);
//  init_bounded_number selpar_L502_HB_D4(0.0,12.0,3);

  //vector sel_HB_D_1(1,nages); //sel in period 1
  vector sel_HB_D_2(1,nages); //sel in period 2
  vector sel_HB_D_3(1,nages); //sel in period 3
  vector sel_HB_D_4(1,nages); //sel in period 4



//MRFSS-----------------------------------------------
  matrix sel_MRFSS(styr,endyr,1,nages);
  //number selpar_slope_MRFSS1; //period 1
  //number selpar_L50_MRFSS1;
  //number selpar_slope2_MRFSS1; //period 1
  //number selpar_L502_MRFSS1;

  //number selpar_slope_MRFSS2; //period 2
```

```
  //number selpar_L50_MRFSS2;
  //number selpar_slope2_MRFSS2; //period 2
  //number selpar_L502_MRFSS2;

  init_bounded_number selpar_L50_MRFSS3(0.1,8.0,1);
  //init_bounded_number selpar_slope_MRFSS3(0.5,12.0,1); //period 3
  number selpar_slope_MRFSS3; //period 3
  //number selpar_L50_MRFSS3;
  number selpar_slope2_MRFSS3; //period 3
  number selpar_L502_MRFSS3;

  init_bounded_number selpar_L50_MRFSS4(0.1,8.0,1);
  init_bounded_number selpar_slope_MRFSS4(0.5,12.0,1); //period 4
  //number selpar_slope_MRFSS4; //period 4
  //number selpar_L50_MRFSS4;
  number selpar_slope2_MRFSS4; //period 4
  number selpar_L502_MRFSS4;

  //init_bounded_dev_vector selpar_L50_MRFSS_dev(styr_MRFSS_lenc,endyr_period1,-5,5,3);
  //vector sel_MRFSS_1(1,nages); //sel in period 1
  //vector sel_MRFSS_2(1,nages); //sel in period 2
  vector sel_MRFSS_3(1,nages); //sel in period 3
  vector sel_MRFSS_4(1,nages); //sel in period 4

  //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_FST_cpue(styr_FST_cpue,endyr_FST_cpue);         //predicted FST U (fish/trap-hour)
  matrix N_FST(styr_FST_cpue,endyr_FST_cpue,1,nages);         //used to compute FST index
  vector pred_CVT_cpue(styr_CVT_cpue,endyr_CVT_cpue);         //predicted CVT U (fish/trap-hour)
  matrix N_CVT(styr_CVT_cpue,endyr_CVT_cpue,1,nages);         //used to compute CVT index
  vector pred_HAL_cpue(styr_HAL_cpue,endyr_HAL_cpue);         //predicted HAL U (pounds/hook-hour)
  matrix N_HAL(styr_HAL_cpue,endyr_HAL_cpue,1,nages);         //used to compute HAL index
  vector pred_HB_cpue(styr_HB_cpue,endyr_HB_cpue);           //predicted HB U (number/angler-day)
  matrix N_HB(styr_HB_cpue,endyr_HB_cpue,1,nages);           //used to compute HB index
  vector pred_MRFSS_cpue(styr_MRFSS_cpue,endyr_MRFSS_cpue);    //predicted MRFSS U (number/1000 hook-hours)
  matrix N_MRFSS(styr_MRFSS_cpue,endyr_MRFSS_cpue,1,nages);    //used to compute MRFSS index

//---Catchability (CPUE q's)-----------------------------------------------------------
  init_bounded_number log_q_FST(-20,-5,1);
  init_bounded_number log_q_CVT(-20,-5,1);
  init_bounded_number log_q_HAL(-20,-5,1);
  init_bounded_number log_q_HB(-20,-5,1);
  init_bounded_number log_q_MRFSS(-20,-5,1);
  //init_bounded_number q_rate(-0.1,0.1,-3);
  number q_rate;

//---Landings Bias for early recreational landings-------------------------------------------------------------
  //init_bounded_number L_early_bias(0.1,10.0,3);
  number L_early_bias;

//---Landings in numbers (total or 1000 fish) and in wgt (klb)-------------------------------------------------
  matrix L_cHAL_num(styr,endyr,1,nages);            //landings (numbers) at age
  matrix L_cHAL_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_cHAL_L_knum(styr_cHAL_L,endyr_cHAL_L); //yearly landings in 1000 fish summed over ages
  vector pred_cHAL_L_klb(styr_cHAL_L,endyr_cHAL_L);  //yearly landings in 1000 lb summed over ages

  matrix L_cCMB_num(styr,endyr,1,nages);            //landings (numbers) at age
  matrix L_cCMB_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_cCMB_L_knum(styr_cCMB_L,endyr_cCMB_L);  //yearly landings in 1000 fish summed over ages
  vector pred_cCMB_L_klb(styr_cCMB_L,endyr_cCMB_L);   //yearly landings in 1000 lb summed over ages

  matrix L_cHTR_num(styr,endyr,1,nages);            //landings (numbers) at age
  matrix L_cHTR_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_cHTR_L_knum(styr_cHTR_L,endyr_cHTR_L);  //yearly landings in 1000 fish summed over ages
  vector pred_cHTR_L_klb(styr_cHTR_L,endyr_cHTR_L);   //yearly landings in 1000 lb 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 whole weight) at age
  vector pred_HB_L_knum(styr_HB_L,endyr_HB_L);       //yearly landings in 1000 fish summed over ages
  vector pred_HB_L_klb(styr_HB_L,endyr_HB_L);        //yearly landings in 1000 lb summed over ages

  matrix L_MRFSS_num(styr,endyr,1,nages);           //landings (numbers) at age
  matrix L_MRFSS_klb(styr,endyr,1,nages);          //landings (1000 lb whole weight) at age
  vector pred_MRFSS_L_knum(styr_HB_L,endyr_HB_L);    //yearly landings in 1000 fish summed over ages
  vector pred_MRFSS_L_klb(styr_MRFSS_L,endyr_MRFSS_L);//yearly landings in 1000 lb 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 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) by yr summed over ages

//---Discards (number dead fish) ----------------------------------------------------
  matrix D_cHAL_num(styr_cHAL_D,endyr_cHAL_D,1,nages);   //discards (numbers) at age
  vector pred_cHAL_D_knum(styr_cHAL_D,endyr_cHAL_D);     //yearly discards summed over ages
  vector obs_cHAL_D(styr_cHAL_D,endyr_cHAL_D);           //observed releases multiplied by discard mortality
  vector pred_cHAL_D_klb(styr_cHAL_D,endyr_cHAL_D);      //yearly discards in klb summed over ages

  matrix D_HB_num(styr_HB_D,endyr_HB_D,1,nages);         //discards (numbers) at age
  vector pred_HB_D_knum(styr_HB_D,endyr_HB_D);           //yearly 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 discards in klb summed over ages

  matrix D_MRFSS_num(styr_HB_D,endyr_MRFSS_D,1,nages);   //discards (numbers) at age
  vector pred_MRFSS_D_knum(styr_MRFSS_D,endyr_MRFSS_D);  //yearly discards summed over ages
  vector obs_MRFSS_D(styr_MRFSS_D,endyr_MRFSS_D);        //observed releases multiplied by discard mortality
  vector pred_MRFSS_D_klb(styr_MRFSS_D,endyr_MRFSS_D);   //yearly discards in klb summed over ages

  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) by yr summed over ages

//---MSY calcs-----------------------------------------------------------------------------
  number F_cHAL_prop;     //proportion of F_full attributable to hal, last X=selpar_n_yrs_wgted yrs
  number F_cCMB_prop;     //proportion of F_full attributable to diving, last X yrs
  number F_HB_prop;       //proportion of F_full attributable to headboat, last X yrs
  number F_MRFSS_prop;    //proportion of F_full attributable to MRFSS, last X yrs
  number F_cHAL_D_prop;   //proportion of F_full attributable to hal discards, last X yrs
  number F_HB_D_prop;     //proportion of F_full attributable to headboat discards, last X yrs
  number F_MRFSS_D_prop;  //proportion of F_full attributable to MRFSS discards, last X yrs
  number F_temp_sum;      //sum of geom mean full Fs in last X yrs, used to compute F_fishery_prop

  number SSB_msy_out;          //SSB (total egg production) at msy
  number F_msy_out;           //F at msy
  number msy_out;             //max sustainable yield
  number B_msy_out;           //total biomass at MSY
  number R_msy_out;           //equilibrium recruitment at F=Fmsy
  number D_msy_out;           //equilibrium dead discards 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_mdyr(1,nages);   //numbers at age for MSY calculations: mdpt of yr
  vector C_age_msy(1,nages);        //catch at age for MSY calculations
  vector Z_age_msy(1,nages);        //total mortality at age for MSY calculations
  vector D_age_msy(1,nages);        //discard mortality (dead discards) 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) 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 D_eq(1,n_iter_msy);        //equilibrium discards (1000s) corresponding to F values in F_msy

  vector FdF_msy(styr,endyr);
  vector SdSSB_msy(styr,endyr);
  number SdSSB_msy_end;
  number FdF_msy_end;

  vector wgt_wgted_L_klb(1,nages);  //fishery-weighted average weight at age of landings
  vector wgt_wgted_D_klb(1,nages);  //fishery-weighted average weight at age of discards
  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
  number M_constant;                    //age-indpendent: used only for MSST
  matrix F(styr,endyr,1,nages);
  vector fullF(styr,endyr);             //Fishing mortality rate by year
  sdreport_vector fullF_sd(styr,endyr);
  matrix Z(styr,endyr,1,nages);

  init_bounded_number log_avg_F_cHAL(-10,0,1);
  init_bounded_dev_vector log_F_dev_cHAL(styr_cHAL_L,endyr_cHAL_L,-10,5,1);
```

```
  matrix F_cHAL(styr,endyr,1,nages);
  vector F_cHAL_out(styr,endyr); //used for intermediate calculations in fcn get_mortality
  //number log_F_init_cHAL;

  init_bounded_number log_avg_F_cCMB(-10,0,1);
  init_bounded_dev_vector log_F_dev_cCMB(styr_cCMB_L,endyr_cCMB_L,-10,5,2);
  matrix F_cCMB(styr,endyr,1,nages);
  vector F_cCMB_out(styr,endyr); //used for intermediate calculations in fcn get_mortality
  //number log_F_init_cCMB;

  init_bounded_number log_avg_F_cHTR(-10,0,1);
  init_bounded_dev_vector log_F_dev_cHTR(styr_cHTR_L,endyr_cHTR_L,-10,5,2);
  matrix F_cHTR(styr,endyr,1,nages);
  vector F_cHTR_out(styr,endyr); //used for intermediate calculations in fcn get_mortality
  //number log_F_init_cHTR;

  init_bounded_number log_avg_F_HB(-10,0,1);
  //init_bounded_dev_vector log_F_dev_HB(styr_HB_L,endyr_HB_L,-10,5,2);
  init_bounded_dev_vector log_F_dev_HB(styr_HB_L,endyr_HB_L,-10,5,2);
  matrix F_HB(styr,endyr,1,nages);
  vector F_HB_out(styr,endyr);        //used for intermediate calculations in fcn get_mortality
  //number log_F_init_HB;

  init_bounded_number log_avg_F_MRFSS(-10,0,1);
  init_bounded_dev_vector log_F_dev_MRFSS(styr_MRFSS_L,endyr_MRFSS_L,-10,5,2);
  matrix F_MRFSS(styr,endyr,1,nages);
  vector F_MRFSS_out(styr,endyr); //used for intermediate calculations in fcn get_mortality
  //number log_F_init_MRFSS;


//--Discard mortality stuff-----------------------------------------------------------------------------
  init_bounded_number log_avg_F_cHAL_D(-10,0,1);
  init_bounded_dev_vector log_F_dev_cHAL_D(styr_cHAL_D,endyr_cHAL_D,-10,5,2);
  matrix F_cHAL_D(styr,endyr,1,nages);
  vector F_cHAL_D_out(styr,endyr); //used for intermediate calculations in fcn get_mortality

  init_bounded_number log_avg_F_HB_D(-10,0,1);
  init_bounded_dev_vector log_F_dev_HB_D(styr_HB_D,endyr_HB_D,-10,5,2);
  matrix F_HB_D(styr,endyr,1,nages);
  vector F_HB_D_out(styr,endyr); //used for intermediate calculations in fcn get_mortality

  matrix sel_MRFSS_D(styr,endyr,1,nages);
  init_bounded_number log_avg_F_MRFSS_D(-10,0,1);
  init_bounded_dev_vector log_F_dev_MRFSS_D(styr_MRFSS_D,endyr_MRFSS_D,-10,5,2);
  matrix F_MRFSS_D(styr,endyr,1,nages);
  vector F_MRFSS_D_out(styr,endyr); //used for intermediate calculations in fcn get_mortality

  number Dmort_cHAL;
  number Dmort_HB;
  number Dmort_MRFSS;

//---Per-recruit stuff-----------------------------------------------------------------------------------
  vector N_age_spr(1,nages);        //numbers at age for SPR calculations: beginning of year
  vector N_age_spr_mdyr(1,nages);   //numbers at age for SPR calculations: midyear
  vector C_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)-per-recruit (ypr) values corresponding to F values in F_spr

  vector N_spr_F0(1,nages);         //Used to compute spr at F=0: midpt of year
  vector N_bpr_F0(1,nages);         //Used to compute bpr at F=0: start of year
  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)
//-------Objective function components-------------------------------------------------------------------
  number w_L;
  number w_D;
  number w_lc;
  number w_ac;
  number w_I_FST;
  number w_I_CVT;
  number w_I_HAL;
  number w_I_HB;
  number w_I_MRFSS;
  number w_R;
  number w_fullF;
```

```
  number w_R_init;
  number w_R_end;
  number w_F;
  number w_B1dB0;
  number w_cvlen_dev;
  number w_cvlen_diff;

  number f_FST_cpue;
  number f_CVT_cpue;
  number f_HAL_cpue;
  number f_HB_cpue;
  number f_MRFSS_cpue;

  number f_cHTR_L;
  number f_cHAL_L;
  number f_cCMB_L;
  number f_HB_L;
  number f_MRFSS_L;

  number f_cHAL_D;
  number f_HB_D;
  number f_MRFSS_D;

  number f_FST_lenc;
  //number f_CVT_lenc;
  number f_cHAL_lenc;
  number f_cHAL_D_lenc;
  number f_cCMB_lenc;
  number f_HB_lenc;
  number f_HB_D_lenc;
  number f_MRFSS_lenc;

  number f_CVT_agec;
  number f_cHAL_agec;
  number f_HB_agec;
  number f_MRFSS_agec;

  //Penalties and constraints. Not all are used.
  number f_N_dev;               //weight on recruitment deviations to fit S-R curve
  number f_N_dev_early;         //extra weight against deviations before styr
  number f_N_dev_end;           //extra constraint on last 3 years of recruitment variability
  number f_Fend_constraint;     //penalty for F deviation in last 5 years
  number f_B1dB0_constraint;    //penalty to fix B(styr)/K
  number f_fullF_constraint;    //penalty for fullF>5
  number f_cvlen_dev_constraint; //deviation penalty on cv's of length at age
  number f_cvlen_diff_constraint;//first diff penalty on cv's of length at age

  objective_function_value fval;
  number fval_unwgt;


//--Dummy arrays for output convenience  -------------------------
  vector xdum(styr,endyr);
  vector xdum2(styr,endyr+1);
//--Other dummy variables ----
  //number sel_diff_dum;
  number zero_dum;
  number dzero_dum;

  //init_number x_dum(1); //used only during model development. can be removed.


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


//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
GLOBALS_SECTION
  #include "admodel.h"         // Include AD class definitions
  #include "admb2r.cpp"    // Include S-compatible output functions (needs preceding)

//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
RUNTIME_SECTION
 maximum_function_evaluations 500, 2000, 10000;
 convergence_criteria 1e-2, 1e-2, 1e-4;

//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
```

```
PRELIMINARY_CALCS_SECTION

// Set values of fixed parameters or set initial guess of estimated parameters
 Dmort_cHAL=set_Dmort_cHAL;
 Dmort_HB=set_Dmort_HB;
 Dmort_MRFSS=set_Dmort_MRFSS;

 obs_cHAL_D=Dmort_cHAL*obs_cHAL_released;
 obs_HB_D=Dmort_HB*obs_HB_released;
 obs_MRFSS_D=Dmort_MRFSS*obs_MRFSS_released;

 Linf=set_Linf;
 K=set_K;
 t0=set_t0;

 age_limit_10in=t0-log(1.0-limit_10in/Linf)/K; //age at size limit: 10" limit;
 age_limit_11in=t0-log(1.0-limit_11in/Linf)/K; //age at size limit: 11" limit;
 age_limit_12in=t0-log(1.0-limit_12in/Linf)/K; //age at size limit: 12" limit;

 M=set_M;
 M_constant=set_M_constant;
 steep=set_steep;
 log_dev_N_rec=0.0;
 R_autocorr=set_R_autocorr;

 log_q_FST=set_logq_FST;
 log_q_CVT=set_logq_CVT;
 log_q_HAL=set_logq_HAL;
 log_q_HB=set_logq_HB;
 log_q_MRFSS=set_logq_MRFSS;
 q_rate=set_q_rate;

 L_early_bias=set_L_early_bias;

 w_L=set_w_L;
 w_D=set_w_D;
 w_lc=set_w_lc;
 w_ac=set_w_ac;
 w_I_FST=set_w_I_FST;
 w_I_CVT=set_w_I_CVT;
 w_I_HAL=set_w_I_HAL;
 w_I_HB=set_w_I_HB;
 w_I_MRFSS=set_w_I_MRFSS;
 w_R=set_w_R;
 w_fullF=set_w_fullF;
 w_R_init=set_w_R_init;
 w_R_end=set_w_R_end;
 w_F=set_w_F;
 w_B1dB0=set_w_B1dB0;
 w_cvlen_dev=set_w_cvlen_dev;
 w_cvlen_diff=set_w_cvlen_diff;

 log_avg_F_cHTR=set_log_avg_F_cHTR;
 log_avg_F_cHAL=set_log_avg_F_cHAL;
 log_avg_F_cCMB=set_log_avg_F_cCMB;
 log_avg_F_HB=set_log_avg_F_HB;
 log_avg_F_MRFSS=set_log_avg_F_MRFSS;

 log_avg_F_cHAL_D=set_log_avg_F_cHAL_D;
 log_avg_F_HB_D=set_log_avg_F_HB_D;
 log_avg_F_MRFSS_D=set_log_avg_F_MRFSS_D;

 log_len_cv=log(set_len_cv);
 log_R0=set_log_R0;
 R1_mult=set_R1_mult;
 B1dB0=set_B1dB0;

 selpar_L50_FST=set_selpar_L50_FST;
 selpar_slope_FST=set_selpar_slope_FST;
 selpar_L502_FST=set_selpar_L502_FST;
 selpar_slope2_FST=set_selpar_slope2_FST;

 selpar_L50_CVT=set_selpar_L50_CVT;
 selpar_slope_CVT=set_selpar_slope_CVT;
 selpar_L502_CVT=set_selpar_L502_CVT;
 selpar_slope2_CVT=set_selpar_slope2_CVT;

 selpar_L50_cHAL1=set_selpar_L50_cHAL1;
 selpar_slope_cHAL1=set_selpar_slope_cHAL1;
 selpar_L502_cHAL1=set_selpar_L502_cHAL1;
```

```
 selpar_slope2_cHAL1=set_selpar_slope2_cHAL1;
 selpar_L50_cHAL2=set_selpar_L50_cHAL2;
 selpar_slope_cHAL2=set_selpar_slope_cHAL2;
 selpar_L502_cHAL2=set_selpar_L502_cHAL2;
 selpar_slope2_cHAL2=set_selpar_slope2_cHAL2;

 sel_cHAL_D_2=set_sel_cHAL_D_2;
 selpar_Age1_cHAL_D2=set_selpar_Age1_cHAL_D2;
 selpar_Age2_cHAL_D2=set_selpar_Age2_cHAL_D2;
 //selpar_L50_cHAL_D2=set_selpar_L50_cHAL_D2;
 //selpar_slope_cHAL_D2=set_selpar_slope_cHAL_D2;
 //selpar_L502_cHAL_D2=set_selpar_L502_cHAL_D2;
 //selpar_slope2_cHAL_D2=set_selpar_slope2_cHAL_D2;

 selpar_L50_cCMB1=set_selpar_L50_cCMB1;
 selpar_L502_cCMB1=set_selpar_L502_cCMB1;
 selpar_slope_cCMB1=set_selpar_slope_cCMB1;
 selpar_slope2_cCMB1=set_selpar_slope2_cCMB1;
 //selpar_L50_cCMB2=set_selpar_L50_cCMB2;
 //selpar_L502_cCMB2=set_selpar_L502_cCMB2; //set equal to cCMB1 par in sel function
 //selpar_slope_cCMB2=set_selpar_slope_cCMB2; //set equal to cCMB1 par in sel function
 //selpar_slope2_cCMB2=set_selpar_slope2_cCMB2;//set equal to cCMB1 par in sel function

 selpar_L50_HB1=set_selpar_L50_HB1;
 selpar_slope_HB1=set_selpar_slope_HB1;
 selpar_L502_HB1=set_selpar_L502_HB1;
 selpar_slope2_HB1=set_selpar_slope2_HB1;
 selpar_L50_HB2=set_selpar_L50_HB2;
 selpar_slope_HB2=set_selpar_slope_HB2;
 selpar_L502_HB2=set_selpar_L502_HB2;
 selpar_slope2_HB2=set_selpar_slope2_HB2;
 selpar_L50_HB3=set_selpar_L50_HB3;
 selpar_slope_HB3=set_selpar_slope_HB3;
 selpar_L502_HB3=set_selpar_L502_HB3;
 selpar_slope2_HB3=set_selpar_slope2_HB3;
 selpar_L50_HB4=set_selpar_L50_HB4;
 selpar_slope_HB4=set_selpar_slope_HB4;
 selpar_L502_HB4=set_selpar_L502_HB4;
 selpar_slope2_HB4=set_selpar_slope2_HB4;

 sel_HB_D_2=set_sel_HB_D_2;
 sel_HB_D_3=set_sel_HB_D_3;
 selpar_Age1_HB_D3=set_selpar_Age1_HB_D3;
 selpar_Age2_HB_D3=set_selpar_Age2_HB_D3;
 sel_HB_D_4=set_sel_HB_D_4;
 selpar_Age1_HB_D4=set_selpar_Age1_HB_D4;
 selpar_Age2_HB_D4=set_selpar_Age2_HB_D4;
 //selpar_L50_HB_D3=set_selpar_L50_HB_D3;
 //selpar_slope_HB_D3=set_selpar_slope_HB_D3; //period 3
 //selpar_slope2_HB_D3=set_selpar_slope2_HB_D3; //period 3
 //selpar_L502_HB_D3=set_selpar_L502_HB_D3;
 //selpar_L50_HB_D4=set_selpar_L50_HB_D4;
 //selpar_slope_HB_D4=set_selpar_slope_HB_D4; //period 4
 //selpar_slope2_HB_D4=set_selpar_slope2_HB_D4; //period 4
 //selpar_L502_HB_D4=set_selpar_L502_HB_D4;

 selpar_L50_MRFSS3=set_selpar_L50_MRFSS3;
 selpar_slope_MRFSS3=set_selpar_slope_MRFSS3;
 selpar_L502_MRFSS3=set_selpar_L502_MRFSS3;
 selpar_slope2_MRFSS3=set_selpar_slope2_MRFSS3;
 selpar_L50_MRFSS4=set_selpar_L50_MRFSS4;
 selpar_slope_MRFSS4=set_selpar_slope_MRFSS4;
 selpar_L502_MRFSS4=set_selpar_L502_MRFSS4;
 selpar_slope2_MRFSS4=set_selpar_slope2_MRFSS4;

//  selpar_cHAL_D_age1=set_selpar_cHAL_D_age1;
//  selpar_HB_D_age1=set_selpar_HB_D_age1;
//  selpar_MRFSS_D_age1=set_selpar_MRFSS_D_age1;

 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

 zero_dum=0.0;

 //additive constant to prevent division by zero
 dzero_dum=0.00001;
```

```
SSB_msy_out=0.0;

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

maturity_f=maturity_f_obs;
prop_f=prop_f_obs;

p_lenc_cHAL=set_p_lenc_cHAL;
p_lenc_cCMB=set_p_lenc_cCMB;
p_lenc_HB2=set_p_lenc_HB2;
p_lenc_HB3=set_p_lenc_HB3;
p_lenc_HB4=set_p_lenc_HB4;
p_lenc_MRFSS2=set_p_lenc_MRFSS2;
p_lenc_MRFSS3=set_p_lenc_MRFSS3;
p_lenc_MRFSS4=set_p_lenc_MRFSS4;

p_lenc_cHAL_D=set_p_lenc_cHAL_D;
p_lenc_HB_D2=set_p_lenc_HB_D2;
p_lenc_HB_D3=set_p_lenc_HB_D3;
p_lenc_HB_D4=set_p_lenc_HB_D4;

////Fill in maturity matrix for calculations for styr to styr
//    for(iyear=styr; iyear<=styr-1; iyear++)
//     {
//       maturity_f(iyear)=maturity_f_obs;
//       maturity_m(iyear)=maturity_m_obs;
//       prop_m(iyear)=prop_m_obs(styr);
//       prop_f(iyear)=1.0-prop_m_obs(styr);
//     }
//    for (iyear=styr;iyear<=endyr;iyear++)
//     {
//       maturity_f(iyear)=maturity_f_obs;
//       maturity_m(iyear)=maturity_m_obs;
//       prop_m(iyear)=prop_m_obs(iyear);
//       prop_f(iyear)=1.0-prop_m_obs(iyear);
//     }

//Fill in sample sizes of comps sampled in nonconsec yrs.
//Used only for output in R object

     nsamp_FST_lenc_allyr=missing;//"missing" defined in admb2r.cpp
     //nsamp_CVT_lenc_allyr=missing;
     nsamp_cHAL_lenc_allyr=missing;
     nsamp_cHAL_D_lenc_allyr=missing;
     nsamp_cCMB_lenc_allyr=missing;
     nsamp_HB_lenc_allyr=missing;
     nsamp_HB_D_lenc_allyr=missing;
     nsamp_MRFSS_lenc_allyr=missing;
     nsamp_CVT_agec_allyr=missing;
     nsamp_cHAL_agec_allyr=missing;
     nsamp_HB_agec_allyr=missing;
     nsamp_MRFSS_agec_allyr=missing;

     for (iyear=styr_FST_lenc; iyear<=endyr_FST_lenc; iyear++)
       {
        if (nsamp_FST_lenc(iyear)>=minSS_FST_lenc)
        {
         nsamp_FST_lenc_allyr(iyear)=nsamp_FST_lenc(iyear);
        }
       }
//     for (iyear=styr_CVT_lenc; iyear<=endyr_CVT_lenc; iyear++)
//       {
//        if (nsamp_CVT_lenc(iyear)>=minSS_CVT_lenc)
//        {
//         nsamp_CVT_lenc_allyr(iyear)=nsamp_CVT_lenc(iyear);
//        }
//       }
     for (iyear=styr_cHAL_lenc; iyear<=endyr_cHAL_lenc; iyear++)
       {
        if (nsamp_cHAL_lenc(iyear)>=minSS_cHAL_lenc)
        {
         nsamp_cHAL_lenc_allyr(iyear)=nsamp_cHAL_lenc(iyear);
        }
       }
     for (iyear=1; iyear<=nyr_cHAL_D_lenc; iyear++)
       {
        if (nsamp_cHAL_D_lenc(iyear)>=minSS_cHAL_D_lenc)
        {
```

```
            nsamp_cHAL_D_lenc_allyr(yrs_cHAL_D_lenc(iyear))=nsamp_cHAL_D_lenc(iyear);
          }
        }
      for (iyear=1; iyear<=nyr_cCMB_lenc; iyear++)
        {
         if (nsamp_cCMB_lenc(iyear)>=minSS_cCMB_lenc)
         {
          nsamp_cCMB_lenc_allyr(yrs_cCMB_lenc(iyear))=nsamp_cCMB_lenc(iyear);
         }
        }
      for (iyear=styr_HB_lenc; iyear<=endyr_HB_lenc; iyear++)
        {
         if (nsamp_HB_lenc(iyear)>=minSS_HB_lenc)
         {
          nsamp_HB_lenc_allyr(iyear)=nsamp_HB_lenc(iyear);
         }
        }
      for (iyear=styr_HB_D_lenc; iyear<=endyr_HB_D_lenc; iyear++)
        {
         if (nsamp_HB_D_lenc(iyear)>=minSS_HB_D_lenc)
         {
          nsamp_HB_D_lenc_allyr(iyear)=nsamp_HB_D_lenc(iyear);
         }
        }
      for (iyear=styr_MRFSS_lenc; iyear<=endyr_MRFSS_lenc; iyear++)
        {
         if (nsamp_MRFSS_lenc(iyear)>=minSS_MRFSS_lenc)
         {
          nsamp_MRFSS_lenc_allyr(iyear)=nsamp_MRFSS_lenc(iyear);
         }
        }

      for (iyear=styr_CVT_agec; iyear<=endyr_CVT_agec; iyear++)
        {
         if (nsamp_CVT_agec(iyear)>=minSS_CVT_agec)
         {
          nsamp_CVT_agec_allyr(iyear)=nsamp_CVT_agec(iyear);
         }
        }
      for (iyear=1; iyear<=nyr_cHAL_agec; iyear++)
        {
         if (nsamp_cHAL_agec(iyear)>=minSS_cHAL_agec)
         {
           nsamp_cHAL_agec_allyr(yrs_cHAL_agec(iyear))=nsamp_cHAL_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);
         }
        }
      for (iyear=styr_MRFSS_agec; iyear<=endyr_MRFSS_agec; iyear++)
        {
         if (nsamp_MRFSS_agec(iyear)>=minSS_MRFSS_agec)
         {
          nsamp_MRFSS_agec_allyr(iyear)=nsamp_MRFSS_agec(iyear);
         }
        }

//fill in Fs for msy and per-recruit analyses
  //F_msy.fill_seqadd(0,0.0001);
  //F_msy.fill_seqadd(0,iter_inc_msy);
  //F_spr.fill_seqadd(0,.0001);
  F_msy(1)=0.0;
  for (int 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_cHTR.initialize();
  L_cHTR_num.initialize();
```

```
  F_cHAL.initialize();
  L_cHAL_num.initialize();
  F_cCMB.initialize();
  L_cCMB_num.initialize();
  F_HB.initialize();
  L_HB_num.initialize();
  F_MRFSS.initialize();
  L_MRFSS_num.initialize();

  F_cHAL_out.initialize();
  F_cHTR_out.initialize();
  F_cCMB_out.initialize();
  F_HB_out.initialize();
  F_MRFSS_out.initialize();

  F_cHAL_D_out.initialize();
  F_HB_D_out.initialize();
  F_MRFSS_D_out.initialize();

  F_cHAL_D.initialize();
  D_cHAL_num.initialize();
  pred_cHAL_D_klb.initialize();
  F_HB_D.initialize();
  D_HB_num.initialize();
  pred_HB_D_klb.initialize();
  F_MRFSS_D.initialize();
  D_MRFSS_num.initialize();
  pred_MRFSS_D_klb.initialize();

  L_total_knum_yr.initialize();
  L_total_klb_yr.initialize();

  sel_cHAL_D.initialize();
  sel_HB_D.initialize();
  sel_MRFSS_D.initialize();

  log_dev_R.initialize();

//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
//##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
TOP_OF_MAIN_SECTION
  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(500);


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

 R0=mfexp(log_R0);

 //cout<<"start"<<endl;
 get_length_weight_fecundity_at_age();
 //cout << "got length, weight, fecundity transitions" <<endl;
 get_reprod();
 get_length_at_age_dist();
 //cout<< "got predicted length at age distribution"<<endl;
 get_weight_at_age_landings();
 //cout<< "got weight at age of landings"<<endl;
 get_spr_F0();
 //cout << "got F0 spr" << endl;
 get_selectivity();
 //cout << "got selectivity" << endl;
 get_mortality();
 //cout << "got mortalities" << endl;
 get_bias_corr();
 //cout<< "got recruitment bias correction" << endl;
 get_numbers_at_age();
 //cout << "got numbers at age" << endl;
 get_landings_numbers();
 //cout << "got catch at age" << endl;
 get_landings_wgt();
 //cout << "got landings" << endl;
 get_dead_discards();
 //cout << "got discards" << endl;
 get_indices();
 //cout << "got indices" << endl;
```

```
 get_length_comps();
 //cout<< "got length comps"<< endl;
 get_age_comps();
 //cout<< "got age comps"<< endl;
 evaluate_objective_function();
 //cout << "objective function calculations complete" << endl;
FUNCTION get_length_weight_fecundity_at_age
  //compute mean length (mm) and weight (whole) at age
    meanlen_TL=Linf*(1.0-mfexp(-K*(agebins-t0+0.5)));       //total length in mm
    meanlen_FL=lenpar_TL2FL_a+lenpar_TL2FL_b*meanlen_TL; //fork length in mm
    wgt_g=wgtpar_a*pow(meanlen_TL,wgtpar_b);            //wgt in grams
    wgt_kg=g2kg*wgt_g;                                  //wgt in kilograms
    wgt_mt=g2mt*wgt_g;                                  //mt of whole wgt: g2mt converts g to mt
    wgt_klb=mt2klb*wgt_mt;                                //1000 lb of whole wgt
    wgt_lb=mt2lb*wgt_mt;                                //1000 lb of whole wgt
    fecundity=fecpar_batches*fecpar_a*pow(meanlen_FL,fecpar_b)/fecpar_scale; //annual egg production of a mature female at age in units of
fecpar_scale

FUNCTION get_reprod

   //product of stuff going into reproductive capacity calcs
   reprod=elem_prod(elem_prod(prop_f,maturity_f),fecundity);

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)=mfexp(log_len_cv+log_len_cv_dev(iage));
    len_cv(iage)=mfexp(log_len_cv);
    for (ilen=1;ilen<=nlenbins;ilen++)
    {
      lenprob(iage,ilen)=(mfexp(-(square(lenbins(ilen)-meanlen_TL(iage))/
      (2.*square(len_cv(iage)*meanlen_TL(iage)))))/(sqrt2pi*len_cv(iage)*meanlen_TL(iage)));
    }
    for (ilen=1;ilen<=nlenbins_plus;ilen++)
    {
      lenprob_plus(iage,ilen)=(mfexp(-(square(lenbins_plus(ilen)-meanlen_TL(iage))/
      (2.*square(len_cv(iage)*meanlen_TL(iage)))))/(sqrt2pi*len_cv(iage)*meanlen_TL(iage)));
    }

    lenprob(iage)(nlenbins)=lenprob(iage)(nlenbins)+sum(lenprob_plus(iage)); //add mass to plus group
    lenprob(iage)/=sum(lenprob(iage)); //standardize to account for truncated normal (i.e., no sizes<smallest)
  }

  //fishery specific length probs, assumed normal prior to size limits

  lenprob_cHAL1=lenprob;
  lenprob_cHAL2=lenprob; //values will be adjusted based on size limit
  lenprob_cCMB1=lenprob;
  lenprob_cCMB2=lenprob; //values will be adjusted based on size limit
  lenprob_HB1=lenprob;
  lenprob_HB2=lenprob; //values will be adjusted based on size limit
  lenprob_HB3=lenprob; //values will be adjusted based on size limit
  lenprob_HB4=lenprob; //values will be adjusted based on size limit
  lenprob_MRFSS1=lenprob;
  lenprob_MRFSS2=lenprob; //values will be adjusted based on size limit
  lenprob_MRFSS3=lenprob; //values will be adjusted based on size limit
  lenprob_MRFSS4=lenprob; //values will be adjusted based on size limit

  lenprob_cHAL_D2=lenprob; //values will be adjusted based on size limit
  lenprob_HB_D2=lenprob; //values will be adjusted based on size limit
  lenprob_HB_D3=lenprob; //values will be adjusted based on size limit
  lenprob_HB_D4=lenprob; //values will be adjusted based on size limit

  for (iage=1;iage<=nages;iage++)
  {
    for (ilen=1;ilen<=nlenbins;ilen++)
    {
       if (lenbins(ilen) < limit_10in)
       {
         lenprob_HB2(iage,ilen)=p_lenc_HB2*lenprob(iage,ilen);
         lenprob_MRFSS2(iage,ilen)=p_lenc_MRFSS2*lenprob(iage,ilen);

       }
       if (lenbins(ilen) < limit_11in)
       {
         lenprob_HB3(iage,ilen)=p_lenc_HB3*lenprob(iage,ilen);
         lenprob_MRFSS3(iage,ilen)=p_lenc_MRFSS3*lenprob(iage,ilen);

       }
```

```
      if (lenbins(ilen) < limit_12in)
      {
       lenprob_cHAL2(iage,ilen)=p_lenc_cHAL*lenprob(iage,ilen);
       lenprob_cCMB2(iage,ilen)=p_lenc_cCMB*lenprob(iage,ilen);
       lenprob_HB4(iage,ilen)=p_lenc_HB4*lenprob(iage,ilen);
       lenprob_MRFSS4(iage,ilen)=p_lenc_MRFSS4*lenprob(iage,ilen);

      }

      if (lenbins(ilen) > limit_10in)
      {
       lenprob_HB_D2(iage,ilen)=p_lenc_HB_D2*lenprob(iage,ilen);
      }


      if (lenbins(ilen) > limit_11in)
      {
       lenprob_HB_D3(iage,ilen)=p_lenc_HB_D3*lenprob(iage,ilen);
      }


      if (lenbins(ilen) > limit_12in)
      {
       lenprob_cHAL_D2(iage,ilen)=p_lenc_cHAL_D*lenprob(iage,ilen);
       lenprob_HB_D4(iage,ilen)=p_lenc_HB_D4*lenprob(iage,ilen);
      }


    }


    lenprob_cHAL2(iage)/=sum(lenprob_cHAL2(iage));   //standardize
    lenprob_cCMB2(iage)/=sum(lenprob_cCMB2(iage));   //standardize
    lenprob_HB2(iage)/=sum(lenprob_HB2(iage));       //standardize
    lenprob_MRFSS2(iage)/=sum(lenprob_MRFSS2(iage)); //standardize
    lenprob_HB3(iage)/=sum(lenprob_HB3(iage));       //standardize
    lenprob_MRFSS3(iage)/=sum(lenprob_MRFSS3(iage)); //standardize
    lenprob_HB4(iage)/=sum(lenprob_HB4(iage));       //standardize
    lenprob_MRFSS4(iage)/=sum(lenprob_MRFSS4(iage)); //standardize

    lenprob_cHAL_D2(iage)/=sum(lenprob_cHAL_D2(iage)); //standardize
    lenprob_HB_D2(iage)/=sum(lenprob_HB_D2(iage)); //standardize
    lenprob_HB_D3(iage)/=sum(lenprob_HB_D3(iage)); //standardize
    lenprob_HB_D4(iage)/=sum(lenprob_HB_D4(iage)); //standardize

  }


FUNCTION get_weight_at_age_landings
  for (iyear=styr; iyear<=endyr_period1; iyear++)
  {
    len_cHAL_mm(iyear)=meanlen_TL;
    wgt_cHAL_klb(iyear)=wgt_klb;
    len_cHTR_mm(iyear)=meanlen_TL;
    wgt_cHTR_klb(iyear)=wgt_klb;
    len_cCMB_mm(iyear)=meanlen_TL;
    wgt_cCMB_klb(iyear)=wgt_klb;
    len_HB_mm(iyear)=meanlen_TL;
    wgt_HB_klb(iyear)=wgt_klb;
    len_MRFSS_mm(iyear)=meanlen_TL;
    wgt_MRFSS_klb(iyear)=wgt_klb;

    len_cHAL_D_mm(iyear)=meanlen_TL;
    wgt_cHAL_D_klb(iyear)=wgt_klb;
    //len_HB_D_mm(iyear)=meanlen_TL;
    //wgt_HB_D_klb(iyear)=wgt_klb;

    //HB and rec discards assumed same size as period one, consistent with selectivity assumptions
    for (iage=1;iage<=nages; iage++)
    {
      len_HB_D_mm(iyear,iage)=sum(elem_prod(lenprob_HB_D2(iage),lenbins));
    }
      wgt_HB_D_klb(iyear)=g2klb*wgtpar_a*pow(len_HB_D_mm(iyear),wgtpar_b);
  }


  for (iyear=(endyr_period1+1); iyear<=endyr_period2; iyear++)
  {
    for (iage=1;iage<=nages; iage++)
    {
```

```
  len_cHAL_mm(iyear,iage)=sum(elem_prod(lenprob_cHAL2(iage),lenbins));
  len_cCMB_mm(iyear,iage)=sum(elem_prod(lenprob_cCMB2(iage),lenbins));
  len_HB_mm(iyear,iage)=sum(elem_prod(lenprob_HB2(iage),lenbins));
  len_MRFSS_mm(iyear,iage)=sum(elem_prod(lenprob_MRFSS2(iage),lenbins));
  len_cHAL_D_mm(iyear,iage)=sum(elem_prod(lenprob_cHAL_D2(iage),lenbins));
  len_HB_D_mm(iyear,iage)=sum(elem_prod(lenprob_HB_D2(iage),lenbins));
  }
  wgt_cHAL_klb(iyear)=g2klb*wgtpar_a*pow(len_cHAL_mm(iyear),wgtpar_b);
  wgt_cCMB_klb(iyear)=g2klb*wgtpar_a*pow(len_cCMB_mm(iyear),wgtpar_b);
  wgt_HB_klb(iyear)=g2klb*wgtpar_a*pow(len_HB_mm(iyear),wgtpar_b);
  wgt_MRFSS_klb(iyear)=g2klb*wgtpar_a*pow(len_MRFSS_mm(iyear),wgtpar_b);
  wgt_cHAL_D_klb(iyear)=g2klb*wgtpar_a*pow(len_cHAL_D_mm(iyear),wgtpar_b);
  wgt_HB_D_klb(iyear)=g2klb*wgtpar_a*pow(len_HB_D_mm(iyear),wgtpar_b);
 }

 for (iyear=(endyr_period2+1); iyear<=endyr_period3; iyear++)
 {
  for (iage=1;iage<=nages; iage++)
  {
  len_cHAL_mm(iyear,iage)=sum(elem_prod(lenprob_cHAL2(iage),lenbins));
  len_cCMB_mm(iyear,iage)=sum(elem_prod(lenprob_cCMB2(iage),lenbins));
  len_HB_mm(iyear,iage)=sum(elem_prod(lenprob_HB3(iage),lenbins));
  len_MRFSS_mm(iyear,iage)=sum(elem_prod(lenprob_MRFSS3(iage),lenbins));
  len_cHAL_D_mm(iyear,iage)=sum(elem_prod(lenprob_cHAL_D2(iage),lenbins));
  len_HB_D_mm(iyear,iage)=sum(elem_prod(lenprob_HB_D3(iage),lenbins));
  }
  wgt_cHAL_klb(iyear)=g2klb*wgtpar_a*pow(len_cHAL_mm(iyear),wgtpar_b);
  wgt_cCMB_klb(iyear)=g2klb*wgtpar_a*pow(len_cCMB_mm(iyear),wgtpar_b);
  wgt_HB_klb(iyear)=g2klb*wgtpar_a*pow(len_HB_mm(iyear),wgtpar_b);
  wgt_MRFSS_klb(iyear)=g2klb*wgtpar_a*pow(len_MRFSS_mm(iyear),wgtpar_b);
  wgt_cHAL_D_klb(iyear)=g2klb*wgtpar_a*pow(len_cHAL_D_mm(iyear),wgtpar_b);
  wgt_HB_D_klb(iyear)=g2klb*wgtpar_a*pow(len_HB_D_mm(iyear),wgtpar_b);
 }

 for (iyear=(endyr_period3+1); iyear<=endyr; iyear++)
 {
  for (iage=1;iage<=nages; iage++)
  {
  len_cHAL_mm(iyear,iage)=sum(elem_prod(lenprob_cHAL2(iage),lenbins));
  len_cCMB_mm(iyear,iage)=sum(elem_prod(lenprob_cCMB2(iage),lenbins));
  len_HB_mm(iyear,iage)=sum(elem_prod(lenprob_HB4(iage),lenbins));
  len_MRFSS_mm(iyear,iage)=sum(elem_prod(lenprob_MRFSS4(iage),lenbins));
  len_cHAL_D_mm(iyear,iage)=sum(elem_prod(lenprob_cHAL_D2(iage),lenbins));
  len_HB_D_mm(iyear,iage)=sum(elem_prod(lenprob_HB_D4(iage),lenbins));
  }
  wgt_cHAL_klb(iyear)=g2klb*wgtpar_a*pow(len_cHAL_mm(iyear),wgtpar_b);
  wgt_cCMB_klb(iyear)=g2klb*wgtpar_a*pow(len_cCMB_mm(iyear),wgtpar_b);
  wgt_HB_klb(iyear)=g2klb*wgtpar_a*pow(len_HB_mm(iyear),wgtpar_b);
  wgt_MRFSS_klb(iyear)=g2klb*wgtpar_a*pow(len_MRFSS_mm(iyear),wgtpar_b);
  wgt_cHAL_D_klb(iyear)=g2klb*wgtpar_a*pow(len_cHAL_D_mm(iyear),wgtpar_b);
  wgt_HB_D_klb(iyear)=g2klb*wgtpar_a*pow(len_HB_D_mm(iyear),wgtpar_b);
 }

 //MRFSS_D assumed same as HB_D
 len_MRFSS_D_mm=len_HB_D_mm;
 wgt_MRFSS_D_klb=wgt_HB_D_klb;



FUNCTION get_spr_F0
 //at mdyr, apply half this yr's 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));
   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));
 bpr_F0=sum(elem_prod(N_bpr_F0,wgt_mt));



FUNCTION get_selectivity
```

```
// ------- compute landings selectivities by period
  selpar_L50_cCMB2=selpar_L50_cHAL2;
  selpar_slope_cCMB2=selpar_slope_cHAL2;
  selpar_slope2_cCMB2=selpar_slope2_cHAL2;
  selpar_L502_cCMB2=selpar_L502_cHAL2;


  for (iage=1; iage<=nages; iage++)
  {
    sel_FST_vec(iage)=(1./(1.+mfexp(-1.*selpar_slope_FST*(double(agebins(iage))-
                  selpar_L50_FST))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_FST*
                  (double(agebins(iage))-(selpar_L50_FST+selpar_L502_FST))))); //double logistic
    sel_CVT_vec(iage)=(1./(1.+mfexp(-1.*selpar_slope_CVT*(double(agebins(iage))-
                  selpar_L50_CVT))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_CVT*
                  (double(agebins(iage))-(selpar_L50_CVT+selpar_L502_CVT))))); //double logistic

    //sel_cHAL_1(iage)=1./(1.+mfexp(-1.*selpar_slope_cHAL1*(double(agebins(iage))-selpar_L50_cHAL1))); //logistic
    sel_cHAL_1(iage)=(1./(1.+mfexp(-1.*selpar_slope_cHAL1*(double(agebins(iage))-
                  selpar_L50_cHAL1))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_cHAL1*
                  (double(agebins(iage))-(selpar_L50_cHAL1+selpar_L502_cHAL1))))); //double logistic

    //sel_cHAL_2(iage)=1./(1.+mfexp(-1.*selpar_slope_cHAL2*(double(agebins(iage))-selpar_L50_cHAL2))); //logistic
    sel_cHAL_2(iage)=(1./(1.+mfexp(-1.*selpar_slope_cHAL2*(double(agebins(iage))-
                  selpar_L50_cHAL2))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_cHAL2*
                  (double(agebins(iage))-(selpar_L50_cHAL2+selpar_L502_cHAL2))))); //double logistic

    sel_cCMB_1(iage)=(1./(1.+mfexp(-1.*selpar_slope_cCMB1*(double(agebins(iage))-
                  selpar_L50_cCMB1))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_cCMB1*
                  (double(agebins(iage))-(selpar_L50_cCMB1+selpar_L502_cCMB1))))); //double logistic
    sel_cCMB_2(iage)=(1./(1.+mfexp(-1.*selpar_slope_cCMB2*(double(agebins(iage))-
                  selpar_L50_cCMB2))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_cCMB2*
                  (double(agebins(iage))-(selpar_L50_cCMB2+selpar_L502_cCMB2))))); //double logistic

    //sel_HB_1(iage)=1./(1.+mfexp(-1.*selpar_slope_HB1*(double(agebins(iage))-selpar_L50_HB1))); //logistic
    sel_HB_1(iage)=(1./(1.+mfexp(-1.*selpar_slope_HB1*(double(agebins(iage))-
                  selpar_L50_HB1))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_HB1*
                  (double(agebins(iage))-(selpar_L50_HB1+selpar_L502_HB1))))); //double logistic
    //sel_HB_2(iage)=1./(1.+mfexp(-1.*selpar_slope_HB2*(double(agebins(iage))-selpar_L50_HB2))); //logistic
    sel_HB_2(iage)=(1./(1.+mfexp(-1.*selpar_slope_HB2*(double(agebins(iage))-
                  selpar_L50_HB2))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_HB2*
                  (double(agebins(iage))-(selpar_L50_HB2+selpar_L502_HB2))))); //double logistic
    //sel_HB_3(iage)=1./(1.+mfexp(-1.*selpar_slope_HB3*(double(agebins(iage))-selpar_L50_HB3))); //logistic
    sel_HB_3(iage)=(1./(1.+mfexp(-1.*selpar_slope_HB3*(double(agebins(iage))-
                  selpar_L50_HB3))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_HB3*
                  (double(agebins(iage))-(selpar_L50_HB3+selpar_L502_HB3))))); //double logistic
    //sel_HB_4(iage)=1./(1.+mfexp(-1.*selpar_slope_HB4*(double(agebins(iage))-selpar_L50_HB4))); //logistic
    sel_HB_4(iage)=(1./(1.+mfexp(-1.*selpar_slope_HB4*(double(agebins(iage))-
                  selpar_L50_HB4))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_HB4*
                  (double(agebins(iage))-(selpar_L50_HB4+selpar_L502_HB4))))); //double logistic
    //sel_MRFSS_3(iage)=1./(1.+mfexp(-1.*selpar_slope_MRFSS3*(double(agebins(iage))-selpar_L50_MRFSS3))); //logistic
    sel_MRFSS_3(iage)=(1./(1.+mfexp(-1.*selpar_slope_MRFSS3*(double(agebins(iage))-
                  selpar_L50_MRFSS3))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_MRFSS3*
                  (double(agebins(iage))-(selpar_L50_MRFSS3+selpar_L502_MRFSS3))))); //double logistic
    //sel_MRFSS_4(iage)=1./(1.+mfexp(-1.*selpar_slope_MRFSS4*(double(agebins(iage))-selpar_L50_MRFSS4))); //logistic
    sel_MRFSS_4(iage)=(1./(1.+mfexp(-1.*selpar_slope_MRFSS4*(double(agebins(iage))-
                  selpar_L50_MRFSS4))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_MRFSS4*
                  (double(agebins(iage))-(selpar_L50_MRFSS4+selpar_L502_MRFSS4))))); //double logistic

  }
  sel_FST_vec=sel_FST_vec/max(sel_FST_vec); //re-normalize double logistic
  sel_CVT_vec=sel_CVT_vec/max(sel_CVT_vec); //re-normalize double logistic

  sel_cHAL_1=sel_cHAL_1/max(sel_cHAL_1);    //re-normalize double logistic
  sel_cHAL_2=sel_cHAL_2/max(sel_cHAL_2);    //re-normalize double logistic
  sel_cCMB_1=sel_cCMB_1/max(sel_cCMB_1);    //re-normalize double logistic
  sel_cCMB_2=sel_cCMB_2/max(sel_cCMB_2);    //re-normalize double logistic
  //sel_cHTR_vec=sel_cCMB_1;

  sel_HB_1=sel_HB_1/max(sel_HB_1); //re-normalize double logistic
  sel_HB_2=sel_HB_2/max(sel_HB_2); //re-normalize double logistic
  sel_HB_3=sel_HB_3/max(sel_HB_3); //re-normalize double logistic
  sel_HB_4=sel_HB_4/max(sel_HB_4); //re-normalize double logistic

  //sel_MRFSS_1=sel_HB_1;
  //sel_MRFSS_2=sel_HB_2;
  sel_MRFSS_3=sel_MRFSS_3/max(sel_MRFSS_3); //re-normalize double logistic
  sel_MRFSS_4=sel_MRFSS_4/max(sel_MRFSS_4); //re-normalize double logistic

//-----------fill in years---------------------------------------------
```

```
 for (iyear=styr; iyear<=endyr; iyear++)
 {  //time-invariant selectivities
   sel_FST(iyear)=sel_FST_vec;
   sel_CVT(iyear)=sel_CVT_vec;

 }
 //Period 1: MRFSS same as HB.  Historical trawl same as CMB.
 for (iyear=styr; iyear<=endyr_period1; iyear++)
 {
   sel_cHTR(iyear)=sel_cCMB_1;
   sel_cHAL(iyear)=sel_cHAL_1;
   sel_cCMB(iyear)=sel_cCMB_1;
   sel_HB(iyear)=sel_HB_1;
   sel_MRFSS(iyear)=sel_HB_1;

 }

 //Period 2: MRFSS same as HB
 for (iyear=endyr_period1+1; iyear<=endyr_period2; iyear++)
 {
   sel_cHAL(iyear)=sel_cHAL_2;
   sel_cCMB(iyear)=sel_cCMB_2;
   sel_HB(iyear)=sel_HB_2;
   sel_MRFSS(iyear)=sel_HB_2;
 }

 //Period 3 (commercial doesn't change from period 2)
 for (iyear=endyr_period2+1; iyear<=endyr_period3; iyear++)
 {
   sel_cHAL(iyear)=sel_cHAL_2;
   sel_cCMB(iyear)=sel_cCMB_2;
   sel_HB(iyear)=sel_HB_3;
   sel_MRFSS(iyear)=sel_MRFSS_3;
 }

 //Period 4 (commercial doesn't change from period 2)
 for (iyear=endyr_period3+1; iyear<=endyr; iyear++)
 {
   sel_cHAL(iyear)=sel_cHAL_2;
   sel_cCMB(iyear)=sel_cCMB_2;
   sel_HB(iyear)=sel_HB_4;
   sel_MRFSS(iyear)=sel_MRFSS_4;
 }

//---Discard selectivities----------------------------------
//-------------------------------------------------------------

 //selpar_slope_HB_D2=(selpar_slope_HB_D3+selpar_slope_HB_D4)/2.0;
 //selpar_L50_HB_D2=(selpar_L50_HB_D3+selpar_L50_HB_D4)/2.0;
 //selpar_slope2_HB_D2=(selpar_slope2_HB_D3+selpar_slope2_HB_D4)/2.0;
 //selpar_L502_HB_D2=age_limit_10in-selpar_L50_HB_D2;  //set to age at period 2 size limit - L50 (bc L50 is added back in)

 sel_cHAL_D_2(1)=selpar_Age1_cHAL_D2;
 sel_cHAL_D_2(2)=selpar_Age2_cHAL_D2;

 sel_HB_D_2(1)=selpar_Age1_HB_D3;
 sel_HB_D_2(2)=selpar_Age2_HB_D3;
 sel_HB_D_3(1)=selpar_Age1_HB_D3;
 sel_HB_D_3(2)=selpar_Age2_HB_D3;
 sel_HB_D_4(1)=selpar_Age1_HB_D4;
 sel_HB_D_4(2)=selpar_Age2_HB_D4;

//  for (iage=1; iage<=nages; iage++)
//  {
//    //sel_cHAL_D_2(iage)=1./(1.+mfexp(-1.*selpar_slope_cHAL_D2*(double(agebins(iage))-selpar_L50_cHAL_D2))); //logistic
//    sel_cHAL_D_2(iage)=(1./(1.+mfexp(-1.*selpar_slope_cHAL_D2*(double(agebins(iage))-
//                  selpar_L50_cHAL_D2))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_cHAL_D2*
//                  (double(agebins(iage))-(selpar_L50_cHAL_D2+selpar_L502_cHAL_D2)))))); //double logistic
//    //sel_HB_D_2(iage)=1./(1.+mfexp(-1.*selpar_slope_HB_D2*(double(agebins(iage))-selpar_L50_HB_D2))); //logistic
//    sel_HB_D_2(iage)=(1./(1.+mfexp(-1.*selpar_slope_HB_D2*(double(agebins(iage))-
//                  selpar_L50_HB_D2))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_HB_D2*
//                  (double(agebins(iage))-(selpar_L50_HB_D2+selpar_L502_HB_D2)))))); //double logistic
//    //sel_HB_D_3(iage)=1./(1.+mfexp(-1.*selpar_slope_HB_D3*(double(agebins(iage))-selpar_L50_HB_D3))); //logistic
//    sel_HB_D_3(iage)=(1./(1.+mfexp(-1.*selpar_slope_HB_D3*(double(agebins(iage))-
//                  selpar_L50_HB_D3))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_HB_D3*
//                  (double(agebins(iage))-(selpar_L50_HB_D3+selpar_L502_HB_D3)))))); //double logistic
//    //sel_HB_D_4(iage)=1./(1.+mfexp(-1.*selpar_slope_HB_D4*(double(agebins(iage))-selpar_L50_HB_D4))); //logistic
//    sel_HB_D_4(iage)=(1./(1.+mfexp(-1.*selpar_slope_HB_D4*(double(agebins(iage))-
//                  selpar_L50_HB_D4))))*(1.-(1./(1.+mfexp(-1.*selpar_slope2_HB_D4*
```

```
//                    (double(agebins(iage))-(selpar_L50_HB_D4+selpar_L502_HB_D4))))); //double logistic
//  }

  sel_cHAL_D_2=sel_cHAL_D_2/max(sel_cHAL_D_2); //re-normalize double logistic
  sel_HB_D_2=sel_HB_D_2/max(sel_HB_D_2); //re-normalize double logistic
  sel_HB_D_3=sel_HB_D_3/max(sel_HB_D_3); //re-normalize double logistic
  sel_HB_D_4=sel_HB_D_4/max(sel_HB_D_4); //re-normalize double logistic

  //Period 1: MRFSS same as HB, both same as period 2
  for (iyear=styr; iyear<=endyr_period1; iyear++)
  {
    sel_HB_D(iyear)=sel_HB_D_2;
    sel_MRFSS_D(iyear)=sel_HB_D_2;

  }

  //Period 2: MRFSS same as HB
  for (iyear=endyr_period1+1; iyear<=endyr_period2; iyear++)
  {
    sel_cHAL_D(iyear)=sel_cHAL_D_2;
    sel_HB_D(iyear)=sel_HB_D_2;
    sel_MRFSS_D(iyear)=sel_HB_D_2;
  }

  //Period 3: MRFSS same as HB, commercial doesn't change from period 2
  for (iyear=endyr_period2+1; iyear<=endyr_period3; iyear++)
  {
    sel_cHAL_D(iyear)=sel_cHAL_D_2;
    sel_HB_D(iyear)=sel_HB_D_3;
    sel_MRFSS_D(iyear)=sel_HB_D_3;
  }

  //Period 4: MRFSS same as HB, commercial doesn't change from period 2
  for (iyear=endyr_period3+1; iyear<=endyr; iyear++)
  {
    sel_cHAL_D(iyear)=sel_cHAL_D_2;
    sel_HB_D(iyear)=sel_HB_D_4;
    sel_MRFSS_D(iyear)=sel_HB_D_4;
  }


FUNCTION get_mortality
  fullF=0.0;
  ////initialization F is avg of first 3 yrs of observed landings
  //log_F_init_cHTR=sum(log_F_dev_cHTR(styr_cHTR_L,(styr_cHTR_L+2)))/3.0;
  //log_F_init_cHAL=sum(log_F_dev_cHAL(styr_cHAL_L,(styr_cHAL_L+2)))/3.0;
  //log_F_init_cCMB=sum(log_F_dev_cCMB(styr_cCMB_L,(styr_cCMB_L+2)))/3.0;
  //log_F_init_HB=sum(log_F_dev_HB(styr_HB_L,(styr_HB_L+2)))/3.0;
  //log_F_init_MRFSS=sum(log_F_dev_MRFSS(styr_MRFSS_L,(styr_MRFSS_L+2)))/3.0;


  for (iyear=styr; iyear<=endyr; iyear++)
  {

    //-------------
    if(iyear>=styr_cHTR_L & iyear<=endyr_cHTR_L)
    {
      F_cHTR_out(iyear)=mfexp(log_avg_F_cHTR+log_F_dev_cHTR(iyear));
    }
    F_cHTR(iyear)=sel_cHTR(iyear)*F_cHTR_out(iyear);
    fullF(iyear)+=F_cHTR_out(iyear);

    //-------------
    if(iyear>=styr_cHAL_L & iyear<=endyr_cHAL_L)
    {
      F_cHAL_out(iyear)=mfexp(log_avg_F_cHAL+log_F_dev_cHAL(iyear));
    }
    F_cHAL(iyear)=sel_cHAL(iyear)*F_cHAL_out(iyear);
    fullF(iyear)+=F_cHAL_out(iyear);

    //-------------
    if(iyear>=styr_cCMB_L & iyear<=endyr_cCMB_L)
    {
      F_cCMB_out(iyear)=mfexp(log_avg_F_cCMB+log_F_dev_cCMB(iyear));
    }
    F_cCMB(iyear)=sel_cCMB(iyear)*F_cCMB_out(iyear);
    fullF(iyear)+=F_cCMB_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);
  fullF(iyear)+=F_HB_out(iyear);



  //-------------
  if(iyear>=styr_MRFSS_L & iyear<=endyr_MRFSS_L)
  {
   F_MRFSS_out(iyear)=mfexp(log_avg_F_MRFSS+log_F_dev_MRFSS(iyear));
  }
  F_MRFSS(iyear)=sel_MRFSS(iyear)*F_MRFSS_out(iyear);
  fullF(iyear)+=F_MRFSS_out(iyear);

  //discards-------------
  if(iyear>=styr_cHAL_D)
  {
   F_cHAL_D_out(iyear)=mfexp(log_avg_F_cHAL_D+log_F_dev_cHAL_D(iyear));
   F_cHAL_D(iyear)=sel_cHAL_D(iyear)*F_cHAL_D_out(iyear);
   fullF(iyear)+=F_cHAL_D_out(iyear);
  }

  if(iyear>=styr_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);
   fullF(iyear)+=F_HB_D_out(iyear);
  }

  if(iyear>=styr_MRFSS_D)
  {
   F_MRFSS_D_out(iyear)=mfexp(log_avg_F_MRFSS_D+log_F_dev_MRFSS_D(iyear));
   F_MRFSS_D(iyear)=sel_MRFSS_D(iyear)*F_MRFSS_D_out(iyear);
   fullF(iyear)+=F_MRFSS_D_out(iyear);
  }

  //Total F at age
  F(iyear)=F_cHAL(iyear); //first in additive series (NO +=)
  F(iyear)+=F_cHTR(iyear);
  F(iyear)+=F_cCMB(iyear);
  F(iyear)+=F_HB(iyear);
  F(iyear)+=F_MRFSS(iyear);

  F(iyear)+=F_cHAL_D(iyear);
  F(iyear)+=F_HB_D(iyear);
  F(iyear)+=F_MRFSS_D(iyear);

  Z(iyear)=M+F(iyear);
 }

FUNCTION get_bias_corr
 //may exclude last BiasCor_exclude_yrs yrs bc constrained or lack info to estimate
 var_rec_dev=norm2(log_dev_N_rec(styr_rec_dev,(endyr-BiasCor_exclude_yrs))-
           sum(log_dev_N_rec(styr_rec_dev,(endyr-BiasCor_exclude_yrs)))
           /(nyrs_rec-BiasCor_exclude_yrs))/(nyrs_rec-BiasCor_exclude_yrs-1.0);
 if (set_BiasCor <= 0.0) {BiasCor=mfexp(var_rec_dev/2.0);}   //bias correction
 else {BiasCor=set_BiasCor;}


FUNCTION get_numbers_at_age
//Initial age
 S0=spr_F0*R0;
 //B0=BiasCor*bpr_F0*R0;
 R1_mult=B1dB0;
 //R1=BiasCor*R1_mult*mfexp(log(((0.8*R0*steep*S0)/
   //(0.2*R0*spr_F0*(1.0-steep)+(steep-0.2)*S0))+dzero_dum));
 R_virgin=(R0/((5.0*steep-1.0)*spr_F0))*
           (BiasCor*4.0*steep*spr_F0-spr_F0*(1.0-steep));
 R1=R1_mult*R_virgin;
 B0=bpr_F0*R_virgin;

 //Assume unfished equilibrium age structure for first year
 N(styr,1)=R1;
 N_mdyr(styr,1)=N(styr,1)*mfexp(-1.*M(1)*spawn_time_frac);
 for (iage=2; iage<=nages; iage++)
 {
  N(styr,iage)=N(styr,iage-1)*mfexp(-1.*M(iage-1));
  N_mdyr(styr,iage)=N(styr,iage)*mfexp(-1.*M(iage)*spawn_time_frac);
```

```
  }
  //plus group calculation
  N(styr,nages)=N(styr,nages)/(1.-mfexp(-1.*M(nages)));
  N_mdyr(styr,nages)=N_mdyr(styr,nages)/(1.-mfexp(-1.*M(nages)));

  SSB(styr)=sum(elem_prod(N_mdyr(styr),reprod));
  B(styr)=elem_prod(N(styr),wgt_mt);
  totB(styr)=sum(B(styr));

 //Rest of years
  for (iyear=styr; iyear<endyr; iyear++)
  {
   if(iyear<(styr_rec_dev-1)) //recruitment follows S-R curve exactly
   {
      //add 0.00001 to avoid log(zero)
      N(iyear+1,1)=BiasCor*mfexp(log(((0.8*R0*steep*SSB(iyear))/(0.2*R0*spr_F0*
         (1.0-steep)+(steep-0.2)*SSB(iyear)))+dzero_dum));
      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))*spawn_time_frac))); //mdyr
      SSB(iyear+1)=sum(elem_prod(N_mdyr(iyear+1),reprod));
      B(iyear+1)=elem_prod(N(iyear+1),wgt_mt);
      totB(iyear+1)=sum(B(iyear+1));
   }
   else   //recruitment follows S-R curve with lognormal deviation
   {
      //add 0.00001 to avoid log(zero)
      N(iyear+1,1)=mfexp(log(((0.8*R0*steep*SSB(iyear))/(0.2*R0*spr_F0*
         (1.0-steep)+(steep-0.2)*SSB(iyear)))+dzero_dum)+log_dev_N_rec(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))*spawn_time_frac))); //mdyr
      SSB(iyear+1)=sum(elem_prod(N_mdyr(iyear+1),reprod));
      B(iyear+1)=elem_prod(N(iyear+1),wgt_mt);
      totB(iyear+1)=sum(B(iyear+1));
   }
  }


   //last year (projection) has no recruitment variability
   N(endyr+1,1)=mfexp(log(((0.8*R0*steep*SSB(endyr))/(0.2*R0*spr_F0*
            (1.0-steep)+(steep-0.2)*SSB(endyr)))+dzero_dum));
   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
   //SSB(endyr+1)=sum(elem_prod(N(endyr+1),reprod));
   B(endyr+1)=elem_prod(N(endyr+1),wgt_mt);
   totB(endyr+1)=sum(B(endyr+1));

 //Recruitment time series
  rec=column(N,1);

 //Benchmark parameters
  S1S0=SSB(styr)/S0;
  popstatus=SSB(endyr)/S0;

 FUNCTION get_landings_numbers //Baranov catch eqn
  for (iyear=styr; iyear<=endyr; iyear++)
  {
   for (iage=1; iage<=nages; iage++)
   {
    L_cHTR_num(iyear,iage)=N(iyear,iage)*F_cHTR(iyear,iage)*
      (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
    L_cHAL_num(iyear,iage)=N(iyear,iage)*F_cHAL(iyear,iage)*
      (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
    L_cCMB_num(iyear,iage)=N(iyear,iage)*F_cCMB(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_MRFSS_num(iyear,iage)=N(iyear,iage)*F_MRFSS(iyear,iage)*
      (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
   }
  }

  for (iyear=styr_cHTR_L; iyear<=endyr_cHTR_L; iyear++)
   {
    pred_cHTR_L_knum(iyear)=sum(L_cHTR_num(iyear))/1000.0;
   }
  for (iyear=styr_cHAL_L; iyear<=endyr_cHAL_L; iyear++)
   {
    pred_cHAL_L_knum(iyear)=sum(L_cHAL_num(iyear))/1000.0;
```

```
  }
 for (iyear=styr_cCMB_L; iyear<=endyr_cCMB_L; iyear++)
  {
   pred_cCMB_L_knum(iyear)=sum(L_cCMB_num(iyear))/1000.0;
  }
 for (iyear=styr_HB_L; iyear<=endyr_HB_L; iyear++)
  {
   pred_HB_L_knum(iyear)=sum(L_HB_num(iyear))/1000.0;
  }
 for (iyear=styr_MRFSS_L; iyear<=endyr_MRFSS_L; iyear++)
  {
   pred_MRFSS_L_knum(iyear)=sum(L_MRFSS_num(iyear))/1000.0;
  }


FUNCTION get_landings_wgt

//---Predicted landings-----------------------
 for (iyear=styr; iyear<=endyr; iyear++)
 {

  L_cHTR_klb(iyear)=elem_prod(L_cHTR_num(iyear),wgt_cHTR_klb(iyear));     //in 1000 lb
  L_cHAL_klb(iyear)=elem_prod(L_cHAL_num(iyear),wgt_cHAL_klb(iyear));     //in 1000 lb
  L_cCMB_klb(iyear)=elem_prod(L_cCMB_num(iyear),wgt_cCMB_klb(iyear));     //in 1000 lb
  L_HB_klb(iyear)=elem_prod(L_HB_num(iyear),wgt_HB_klb(iyear));         //in 1000 lb
  L_MRFSS_klb(iyear)=elem_prod(L_MRFSS_num(iyear),wgt_MRFSS_klb(iyear));   //in 1000 lb
 }

 for (iyear=styr_cHTR_L; iyear<=endyr_cHTR_L; iyear++)
  {
   pred_cHTR_L_klb(iyear)=sum(L_cHTR_klb(iyear));
  }
 for (iyear=styr_cHAL_L; iyear<=endyr_cHAL_L; iyear++)
  {
   pred_cHAL_L_klb(iyear)=sum(L_cHAL_klb(iyear));
  }
 for (iyear=styr_cCMB_L; iyear<=endyr_cCMB_L; iyear++)
  {
   pred_cCMB_L_klb(iyear)=sum(L_cCMB_klb(iyear));
  }
 for (iyear=styr_HB_L; iyear<=endyr_HB_L; iyear++)
  {
   pred_HB_L_klb(iyear)=sum(L_HB_klb(iyear));
  }
 for (iyear=styr_MRFSS_L; iyear<=endyr_MRFSS_L; iyear++)
  {
   pred_MRFSS_L_klb(iyear)=sum(L_MRFSS_klb(iyear));
  }

FUNCTION get_dead_discards //Baranov catch eqn
 //dead discards at age (number fish)

 for (iyear=styr_cHAL_D; iyear<=endyr_cHAL_D; iyear++)
 {
  for (iage=1; iage<=nages; iage++)
  {
   D_cHAL_num(iyear,iage)=N(iyear,iage)*F_cHAL_D(iyear,iage)*
     (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
  }
  pred_cHAL_D_knum(iyear)=sum(D_cHAL_num(iyear))/1000.0;          //pred annual dead discards in 1000s (for matching data)
  pred_cHAL_D_klb(iyear)=sum(elem_prod(D_cHAL_num(iyear),wgt_cHAL_D_klb(iyear))); //annual dead discards in 1000 lb (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 matching data)
  pred_HB_D_klb(iyear)=sum(elem_prod(D_HB_num(iyear),wgt_HB_D_klb(iyear)));  //annual dead discards in 1000 lb (for output only)
 }

 for (iyear=styr_MRFSS_D; iyear<=endyr_MRFSS_D; iyear++)
 {
  for (iage=1; iage<=nages; iage++)
  {
   D_MRFSS_num(iyear,iage)=N(iyear,iage)*F_MRFSS_D(iyear,iage)*
     (1.-mfexp(-1.*Z(iyear,iage)))/Z(iyear,iage);
```

```
  }
  pred_MRFSS_D_knum(iyear)=sum(D_MRFSS_num(iyear))/1000.0;            //pred annual dead discards in 1000s (for matching data)
  pred_MRFSS_D_klb(iyear)=sum(elem_prod(D_MRFSS_num(iyear),wgt_MRFSS_klb(iyear)));  //annual dead discards in 1000 lb (for output only)
 }



FUNCTION get_indices
//---Predicted CPUEs------------------------
 //MARMAP FL snapper trap
  for (iyear=styr_FST_cpue; iyear<=endyr_FST_cpue; iyear++)
  {   //index in number units
     N_FST(iyear)=elem_prod(N_mdyr(iyear),sel_FST(iyear));
     pred_FST_cpue(iyear)=mfexp(log_q_FST)*sum(N_FST(iyear));
  }

 //MARMAP chevron trap
  for (iyear=styr_CVT_cpue; iyear<=endyr_CVT_cpue; iyear++)
  {   //index in number units
     N_CVT(iyear)=elem_prod(N_mdyr(iyear),sel_CVT(iyear));
     pred_CVT_cpue(iyear)=mfexp(log_q_CVT)*sum(N_CVT(iyear));
  }

 //Commercial handline cpue
  for (iyear=styr_HAL_cpue; iyear<=endyr_HAL_cpue; iyear++)
  {   //index in weight units. original index in lb and re-scaled. predicted in klb, but difference is absorbed by q
     N_HAL(iyear)=elem_prod(elem_prod(N_mdyr(iyear),sel_cHAL(iyear)),wgt_cHAL_klb(iyear));
     pred_HAL_cpue(iyear)=mfexp(log_q_HAL)*(1.0+(iyear-styr_HAL_cpue)*q_rate)*sum(N_HAL(iyear));
     //pred_HAL_cpue(iyear)=mfexp(log_q_HAL)*sum(N_HAL(iyear));
  }
 //Headboat cpue
  for (iyear=styr_HB_cpue; iyear<=endyr_HB_cpue; iyear++)
  {   //index in number units
     N_HB(iyear)=elem_prod(N_mdyr(iyear),sel_HB(iyear));
     pred_HB_cpue(iyear)=mfexp(log_q_HB)*(1+(iyear-styr_HB_cpue)*q_rate)*sum(N_HB(iyear));
     //pred_HB_cpue(iyear)=mfexp(log_q_HB)*sum(N_HB(iyear));
  }
  //MRFSS cpue
  for (iyear=styr_MRFSS_cpue; iyear<=endyr_MRFSS_cpue; iyear++)
  {   //index in number units
     N_MRFSS(iyear)=elem_prod(N_mdyr(iyear),sel_MRFSS(iyear));
     pred_MRFSS_cpue(iyear)=mfexp(log_q_MRFSS)*(1+(iyear-styr_MRFSS_cpue)*q_rate)*sum(N_MRFSS(iyear));
     //pred_MRFSS_cpue(iyear)=mfexp(log_q_MRFSS)*sum(N_MRFSS(iyear));
  }


FUNCTION get_length_comps
 //MARMAP fishery independent
  for (iyear=styr_FST_lenc;iyear<=endyr_FST_lenc;iyear++)
  {
    pred_FST_lenc(iyear)=(N_FST(iyear)*lenprob)/sum(N_FST(iyear));
  }

//  for (iyear=styr_CVT_lenc;iyear<=endyr_CVT_lenc;iyear++)
//  {
//    pred_CVT_lenc(iyear)=(N_CVT(iyear)*lenprob)/sum(N_CVT(iyear));
//  }

 //Commercial
  for (iyear=styr_cHAL_lenc; iyear<=endyr_period1; iyear++)
  {
    pred_cHAL_lenc(iyear)=(L_cHAL_num(iyear)*lenprob_cHAL1)/sum(L_cHAL_num(iyear));
  }

  for (iyear=(endyr_period1+1);iyear<=endyr_cHAL_lenc;iyear++)
  {
     pred_cHAL_lenc(iyear)=(L_cHAL_num(iyear)*lenprob_cHAL2)/sum(L_cHAL_num(iyear));
  }

  for (iyear=1;iyear<=nyr_cCMB_lenc;iyear++)
  {
     if (yrs_cCMB_lenc(iyear)<=endyr_period1)
     {
       pred_cCMB_lenc(iyear)=(L_cCMB_num(yrs_cCMB_lenc(iyear))*lenprob_cCMB1)
                        /sum(L_cCMB_num(yrs_cCMB_lenc(iyear)));
     }
     else
     {
       pred_cCMB_lenc(iyear)=(L_cCMB_num(yrs_cCMB_lenc(iyear))*lenprob_cCMB2)
                        /sum(L_cCMB_num(yrs_cCMB_lenc(iyear)));
```

```
    }

 }

 for (iyear=1;iyear<=nyr_cHAL_D_lenc;iyear++)
 {
  pred_cHAL_D_lenc(iyear)=(D_cHAL_num(yrs_cHAL_D_lenc(iyear))*lenprob_cHAL_D2)
                          /sum(D_cHAL_num(yrs_cHAL_D_lenc(iyear)));
 }

//Headboat
 for (iyear=styr_HB_lenc;iyear<=endyr_period1;iyear++)
 {
  pred_HB_lenc(iyear)=(L_HB_num(iyear)*lenprob_HB1)/sum(L_HB_num(iyear));
 }
 for (iyear=(endyr_period1+1);iyear<=endyr_period2;iyear++)
 {
  pred_HB_lenc(iyear)=(L_HB_num(iyear)*lenprob_HB2)/sum(L_HB_num(iyear));
 }
 for (iyear=(endyr_period2+1);iyear<=endyr_period3;iyear++)
 {
  pred_HB_lenc(iyear)=(L_HB_num(iyear)*lenprob_HB3)/sum(L_HB_num(iyear));
 }
 for (iyear=(endyr_period3+1);iyear<=endyr_HB_lenc;iyear++)
 {
  pred_HB_lenc(iyear)=(L_HB_num(iyear)*lenprob_HB4)/sum(L_HB_num(iyear));
 }

 for (iyear=styr_HB_D_lenc;iyear<=endyr_period3;iyear++)
 {
  pred_HB_D_lenc(iyear)=(D_HB_num(iyear)*lenprob_HB_D3)/sum(D_HB_num(iyear));
 }
 for (iyear=(endyr_period3+1);iyear<=endyr_HB_D_lenc;iyear++)
 {
  pred_HB_D_lenc(iyear)=(D_HB_num(iyear)*lenprob_HB_D4)/sum(D_HB_num(iyear));
 }


//MRFSS
 for (iyear=styr_MRFSS_lenc;iyear<=endyr_period1;iyear++)
 {
  pred_MRFSS_lenc(iyear)=(L_MRFSS_num(iyear)*lenprob_MRFSS1)/sum(L_MRFSS_num(iyear));
 }
 for (iyear=(endyr_period1+1);iyear<=endyr_period2;iyear++)
 {
  pred_MRFSS_lenc(iyear)=(L_MRFSS_num(iyear)*lenprob_MRFSS2)/sum(L_MRFSS_num(iyear));
 }
 for (iyear=(endyr_period2+1);iyear<=endyr_period3;iyear++)
 {
  pred_MRFSS_lenc(iyear)=(L_MRFSS_num(iyear)*lenprob_MRFSS3)/sum(L_MRFSS_num(iyear));
 }
 for (iyear=(endyr_period3+1);iyear<=endyr_MRFSS_lenc;iyear++)
 {
  pred_MRFSS_lenc(iyear)=(L_MRFSS_num(iyear)*lenprob_MRFSS4)/sum(L_MRFSS_num(iyear));
 }

FUNCTION get_age_comps

 //MARMAP CVT
  for (iyear=styr_CVT_agec;iyear<=endyr_CVT_agec;iyear++)
  {
   ErrorFree_CVT_agec(iyear)=N_CVT(iyear)/sum(N_CVT(iyear));
   pred_CVT_agec(iyear)=age_error*ErrorFree_CVT_agec(iyear);
  }

 //Commercial
  for (iyear=1;iyear<=nyr_cHAL_agec;iyear++)
  {
   ErrorFree_cHAL_agec(iyear)=L_cHAL_num(yrs_cHAL_agec(iyear))/
                       sum(L_cHAL_num(yrs_cHAL_agec(iyear)));
   pred_cHAL_agec(iyear)=age_error*ErrorFree_cHAL_agec(iyear);
  }

 //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(iyear)=age_error*ErrorFree_HB_agec(iyear);
```

```
 }

 //MRFSS
  for (iyear=styr_MRFSS_agec;iyear<=endyr_MRFSS_agec;iyear++)
  {
    ErrorFree_MRFSS_agec(iyear)=L_MRFSS_num(iyear)/sum(L_MRFSS_num(iyear));
    pred_MRFSS_agec(iyear)=age_error*ErrorFree_MRFSS_agec(iyear);
  }

//--------------------------------------------------------------------------------------------------------------------------------------------------------------
-----
FUNCTION get_weighted_current //should not and does not include historic trawl
  F_temp_sum=0.0;
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_cHAL+
       sum(log_F_dev_cHAL((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_cCMB+
       sum(log_F_dev_cCMB((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_MRFSS+
       sum(log_F_dev_MRFSS((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);
  F_temp_sum+=mfexp((selpar_n_yrs_wgted*log_avg_F_cHAL_D+
       sum(log_F_dev_cHAL_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_MRFSS_D+
       sum(log_F_dev_MRFSS_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted);

  F_cHAL_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_cHAL+
       sum(log_F_dev_cHAL((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_cCMB_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_cCMB+
       sum(log_F_dev_cCMB((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_MRFSS_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_MRFSS+
       sum(log_F_dev_MRFSS((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;
  F_cHAL_D_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_cHAL_D+
       sum(log_F_dev_cHAL_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_MRFSS_D_prop=mfexp((selpar_n_yrs_wgted*log_avg_F_MRFSS_D+
       sum(log_F_dev_MRFSS_D((endyr-selpar_n_yrs_wgted+1),endyr)))/selpar_n_yrs_wgted)/F_temp_sum;

  sel_wgted_L=F_cHAL_prop*sel_cHAL(endyr)+
            F_cCMB_prop*sel_cCMB(endyr)+
            F_HB_prop*sel_HB(endyr)+
            F_MRFSS_prop*sel_MRFSS(endyr);

  sel_wgted_D=F_cHAL_D_prop*sel_cHAL_D(endyr)+
             F_HB_D_prop*sel_HB_D(endyr)+
             F_MRFSS_D_prop*sel_MRFSS_D(endyr);

  sel_wgted_tot=sel_wgted_L+sel_wgted_D;


  wgt_wgted_L_denom=F_cHAL_prop+F_cCMB_prop+F_HB_prop+F_MRFSS_prop;
  wgt_wgted_L_klb=F_cHAL_prop/wgt_wgted_L_denom*wgt_cHAL_klb(endyr)+
            F_cCMB_prop/wgt_wgted_L_denom*wgt_cCMB_klb(endyr)+
            F_HB_prop/wgt_wgted_L_denom*wgt_HB_klb(endyr)+
            F_MRFSS_prop/wgt_wgted_L_denom*wgt_MRFSS_klb(endyr);

  wgt_wgted_D_denom=F_cHAL_D_prop+F_HB_D_prop+F_MRFSS_D_prop;
  wgt_wgted_D_klb=F_cHAL_D_prop/wgt_wgted_D_denom*wgt_cHAL_D_klb(endyr)+
            F_HB_D_prop/wgt_wgted_D_denom*wgt_HB_D_klb(endyr)+
            F_MRFSS_D_prop/wgt_wgted_D_denom*wgt_MRFSS_D_klb(endyr);

FUNCTION get_msy

  //compute values as functions of F
  for(int 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_mdyr(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_mdyr(nages)=(N_age_msy_mdyr(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_mdyr,reprod));


  //Compute equilibrium values of R (including bias correction), SSB and Yield at each F
  R_eq(ff)=(R0/((5.0*steep-1.0)*spr_msy(ff)))*
            (BiasCor*4.0*steep*spr_msy(ff)-spr_F0*(1.0-steep));
  if (R_eq(ff)<dzero_dum) {R_eq(ff)=dzero_dum;}
  N_age_msy*=R_eq(ff);
  N_age_msy_mdyr*=R_eq(ff);

  for (iage=1; iage<=nages; iage++)
  {
   C_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_mdyr,reprod));
  B_eq(ff)=sum(elem_prod(N_age_msy,wgt_mt));
  L_eq_klb(ff)=sum(elem_prod(C_age_msy,wgt_wgted_L_klb));
  D_eq(ff)=sum(D_age_msy)/1000.0;
 }

 msy_out=max(L_eq_klb);

 for(ff=1; ff<=n_iter_msy; ff++)
 {
  if(L_eq_klb(ff) == msy_out)
    {
     SSB_msy_out=SSB_eq(ff);
     B_msy_out=B_eq(ff);
     R_msy_out=R_eq(ff);
     D_msy_out=D_eq(ff);
     F_msy_out=F_msy(ff);
     spr_msy_out=spr_msy(ff);
    }
 }

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

 //compute total landings-at-age in number and klb
 L_total_num=(L_HB_num+L_MRFSS_num+L_cHAL_num+L_cCMB_num+L_cHTR_num); //catch in number fish
 L_total_klb=L_HB_klb+L_MRFSS_klb+L_cHAL_klb+L_cCMB_klb+L_cHTR_klb;   //landings in klb whole weight

 D_total_knum_yr.initialize();
 D_total_klb_yr.initialize();

 for(iyear=styr; iyear<=endyr; iyear++)
 {
     L_total_klb_yr(iyear)=sum(L_total_klb(iyear));
     L_total_knum_yr(iyear)=sum(L_total_num(iyear))/1000.0;

     if (iyear>=styr_cHAL_D && iyear<=endyr_cHAL_D)
     {
      D_total_knum_yr(iyear)+=pred_cHAL_D_knum(iyear);
      D_total_klb_yr(iyear)+=pred_cHAL_D_klb(iyear);
     }

     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);
     }
```

```
      if (iyear>=styr_MRFSS_D && iyear<=endyr_MRFSS_D)
      {
       D_total_knum_yr(iyear)+=pred_MRFSS_D_knum(iyear);
       D_total_klb_yr(iyear)+=pred_MRFSS_D_klb(iyear);
      }
 }

 steep_sd=steep;
 fullF_sd=fullF;

 if(F_msy_out>0)
  {
   FdF_msy=fullF/F_msy_out;
   FdF_msy_end=FdF_msy(endyr);
  }
 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; iyear++)
 {
  log_dev_R(iyear)=log_dev_N_rec(iyear);
 }

//--------------------------------------------------------------------------------------------------------------------------------------------------------
-----
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_mdyr(1,(nages-1))=elem_prod(N_age_spr(1,(nages-1)),
                          mfexp(-1.*Z(iyear)(1,(nages-1))*spawn_time_frac));
  N_age_spr_mdyr(nages)=(N_age_spr_mdyr(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_mdyr,reprod))/spr_F0;
 }


 //compute SSB/R and YPR as functions of F
 for(int 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_mdyr(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_mdyr(nages)=(N_age_spr_mdyr(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_mdyr,reprod));
  L_spr(ff)=0.0;
  for (iage=1; iage<=nages; iage++)
  {
   C_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)+=C_age_spr(iage)*wgt_wgted_L_klb(iage)*1000.0; //in lb
   }
  }

//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----


 FUNCTION evaluate_objective_function
  fval=0.0;
  fval_unwgt=0.0;

 //---likelihoods-----------------------------
  //fval=square(x_dum-5226.0);
  //cout << "fval="<<fval<<endl;

 //---Indices---------------------------------
  f_FST_cpue=0.0;
  for (iyear=styr_FST_cpue; iyear<=endyr_FST_cpue; iyear++)
  {
   f_FST_cpue+=square(log((pred_FST_cpue(iyear)+dzero_dum)/
      (obs_FST_cpue(iyear)+dzero_dum)))/(2.0*square(FST_cpue_cv(iyear)));
  }
  fval+=w_I_FST*f_FST_cpue;
  fval_unwgt+=f_FST_cpue;

  f_CVT_cpue=0.0;
  for (iyear=styr_CVT_cpue; iyear<=endyr_CVT_cpue; iyear++)
  {
   f_CVT_cpue+=square(log((pred_CVT_cpue(iyear)+dzero_dum)/
      (obs_CVT_cpue(iyear)+dzero_dum)))/(2.0*square(CVT_cpue_cv(iyear)));
  }
  fval+=w_I_CVT*f_CVT_cpue;
  fval_unwgt+=f_CVT_cpue;

  f_HAL_cpue=0.0;
  for (iyear=styr_HAL_cpue; iyear<=endyr_HAL_cpue; iyear++)
  {
   f_HAL_cpue+=square(log((pred_HAL_cpue(iyear)+dzero_dum)/
      (obs_HAL_cpue(iyear)+dzero_dum)))/(2.0*square(HAL_cpue_cv(iyear)));
  }
  fval+=w_I_HAL*f_HAL_cpue;
  fval_unwgt+=f_HAL_cpue;

  f_HB_cpue=0.0;
  for (iyear=styr_HB_cpue; iyear<=endyr_HB_cpue; iyear++)
  {
   f_HB_cpue+=square(log((pred_HB_cpue(iyear)+dzero_dum)/
      (obs_HB_cpue(iyear)+dzero_dum)))/(2.0*square(HB_cpue_cv(iyear)));
  }
  fval+=w_I_HB*f_HB_cpue;
  fval_unwgt+=f_HB_cpue;

  f_MRFSS_cpue=0.0;
  for (iyear=styr_MRFSS_cpue; iyear<=endyr_MRFSS_cpue; iyear++)
  {
   f_MRFSS_cpue+=square(log((pred_MRFSS_cpue(iyear)+dzero_dum)/
      (obs_MRFSS_cpue(iyear)+dzero_dum)))/(2.0*square(MRFSS_cpue_cv(iyear)));
  }
  fval+=w_I_MRFSS*f_MRFSS_cpue;
  fval_unwgt+=f_MRFSS_cpue;


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

  f_cHAL_L=0.0; //in 1000 lb ww
  for (iyear=styr_cHAL_L; iyear<=endyr_cHAL_L; iyear++)
  {
     f_cHAL_L+=square(log((pred_cHAL_L_klb(iyear)+dzero_dum)/
      (obs_cHAL_L(iyear)+dzero_dum)))/(2.0*square(cHAL_L_cv(iyear)));
  }
  fval+=w_L*f_cHAL_L;
  fval_unwgt+=f_cHAL_L;

  f_cCMB_L=0.0; //in 1000 lb ww
  for (iyear=styr_cCMB_L; iyear<=endyr_cCMB_L; iyear++)
  {
   f_cCMB_L+=square(log((pred_cCMB_L_klb(iyear)+dzero_dum)/
      (obs_cCMB_L(iyear)+dzero_dum)))/(2.0*square(cCMB_L_cv(iyear)));
  }
```

```
  fval+=w_L*f_cCMB_L;
  fval_unwgt+=f_cCMB_L;

  f_cHTR_L=0.0; //in 1000 lb ww
  for (iyear=styr_cHTR_L; iyear<=endyr_cHTR_L; iyear++)
  {
   f_cHTR_L+=square(log((pred_cHTR_L_klb(iyear)+dzero_dum)/
      (obs_cHTR_L(iyear)+dzero_dum)))/(2.0*square(cHTR_L_cv(iyear)));
  }
  fval+=w_L*f_cHTR_L;
  fval_unwgt+=f_cHTR_L;


  f_HB_L=0.0; //in 1000 fish
  for (iyear=styr_HB_L; iyear<=1971; iyear++)
  {
   f_HB_L+=square(log((pred_HB_L_knum(iyear)+dzero_dum)/
      (L_early_bias*obs_HB_L(iyear)+dzero_dum)))/(2.0*square(HB_L_cv(iyear)));
  }
  for (iyear=1972; iyear<=endyr_HB_L; iyear++)
  {
   f_HB_L+=square(log((pred_HB_L_knum(iyear)+dzero_dum)/
      (obs_HB_L(iyear)+dzero_dum)))/(2.0*square(HB_L_cv(iyear)));
  }
  fval+=w_L*f_HB_L;
  fval_unwgt+=f_HB_L;

  f_MRFSS_L=0.0; //in 1000 fish
  for (iyear=styr_MRFSS_L; iyear<=1980; iyear++)
  {
   f_MRFSS_L+=square(log((pred_MRFSS_L_knum(iyear)+dzero_dum)/
      (L_early_bias*obs_MRFSS_L(iyear)+dzero_dum)))/(2.0*square(MRFSS_L_cv(iyear)));
  }
  for (iyear=1981; iyear<=endyr_MRFSS_L; iyear++)
  {
   f_MRFSS_L+=square(log((pred_MRFSS_L_knum(iyear)+dzero_dum)/
      (obs_MRFSS_L(iyear)+dzero_dum)))/(2.0*square(MRFSS_L_cv(iyear)));
  }
  fval+=w_L*f_MRFSS_L;
  fval_unwgt+=f_MRFSS_L;


 //---Discards---------------------------------
  f_cHAL_D=0.0; //in 1000 fish
  for (iyear=styr_cHAL_D; iyear<=endyr_cHAL_D; iyear++)
  {
   f_cHAL_D+=square(log((pred_cHAL_D_knum(iyear)+dzero_dum)/
      (obs_cHAL_D(iyear)+dzero_dum)))/(2.0*square(cHAL_D_cv(iyear)));
  }
  fval+=w_D*f_cHAL_D;
  fval_unwgt+=f_cHAL_D;


  f_HB_D=0.0; //in 1000 fish
  for (iyear=styr_HB_D; iyear<=endyr_HB_D; iyear++)
  {
   f_HB_D+=square(log((pred_HB_D_knum(iyear)+dzero_dum)/
      (obs_HB_D(iyear)+dzero_dum)))/(2.0*square(HB_D_cv(iyear)));
  }
  fval+=w_D*f_HB_D;
  fval_unwgt+=f_HB_D;

  f_MRFSS_D=0.0; //in 1000 fish
  for (iyear=styr_MRFSS_D; iyear<=endyr_MRFSS_D; iyear++)
  {
   f_MRFSS_D+=square(log((pred_MRFSS_D_knum(iyear)+dzero_dum)/
      (obs_MRFSS_D(iyear)+dzero_dum)))/(2.0*square(MRFSS_D_cv(iyear)));
  }
  fval+=w_D*f_MRFSS_D;
  fval_unwgt+=f_MRFSS_D;


 //---Length comps-----------------------------
  f_FST_lenc=0.0;
  for (iyear=styr_FST_lenc; iyear<=endyr_FST_lenc; iyear++)
  {
   if (nsamp_FST_lenc(iyear)>=minSS_FST_lenc)
   {
   f_FST_lenc-=nsamp_FST_lenc(iyear)*
      sum( elem_prod((obs_FST_lenc(iyear)+dzero_dum),
```

```
          log(elem_div((pred_FST_lenc(iyear)+dzero_dum),
            (obs_FST_lenc(iyear)+dzero_dum)));
   }
  }
  fval+=w_lc*f_FST_lenc;
  fval_unwgt+=f_FST_lenc;


//  f_CVT_lenc=0.0;
//  for (iyear=styr_CVT_lenc; iyear<=endyr_CVT_lenc; iyear++)
//  {
//    if (nsamp_CVT_lenc(iyear)>=minSS_CVT_lenc)
//    {
//    f_CVT_lenc-=nsamp_CVT_lenc(iyear)*
//       sum( elem_prod((obs_CVT_lenc(iyear)+dzero_dum),
//          log(elem_div((pred_CVT_lenc(iyear)+dzero_dum),
//            (obs_CVT_lenc(iyear)+dzero_dum)));
//    }
//  }
//  fval+=w_lc*f_CVT_lenc;
//  fval_unwgt+=f_CVT_lenc;


  f_cHAL_lenc=0.0;
  for (iyear=styr_cHAL_lenc; iyear<=endyr_cHAL_lenc; iyear++)
  {
   if (nsamp_cHAL_lenc(iyear)>=minSS_cHAL_lenc)
   {
   f_cHAL_lenc-=nsamp_cHAL_lenc(iyear)*
      sum( elem_prod((obs_cHAL_lenc(iyear)+dzero_dum),
         log(elem_div((pred_cHAL_lenc(iyear)+dzero_dum),
           (obs_cHAL_lenc(iyear)+dzero_dum)));
   }
  }
  fval+=w_lc*f_cHAL_lenc;
  fval_unwgt+=f_cHAL_lenc;


  f_cHAL_D_lenc=0.;
  for (iyear=1; iyear<=nyr_cHAL_D_lenc; iyear++)
  {
   if (nsamp_cHAL_D_lenc(iyear)>=minSS_cHAL_D_lenc)
   {
   f_cHAL_D_lenc-=nsamp_cHAL_D_lenc(iyear)*
      sum(elem_prod((obs_cHAL_D_lenc(iyear)+dzero_dum),
         log(elem_div((pred_cHAL_D_lenc(iyear)+dzero_dum),
           (obs_cHAL_D_lenc(iyear)+dzero_dum)));
   }
  }
  fval+=w_lc*f_cHAL_D_lenc;
  fval_unwgt+=f_cHAL_D_lenc;


  f_cCMB_lenc=0.;
  for (iyear=1; iyear<=nyr_cCMB_lenc; iyear++)
  {
   if (nsamp_cCMB_lenc(iyear)>=minSS_cCMB_lenc)
   {
   f_cCMB_lenc-=nsamp_cCMB_lenc(iyear)*
      sum(elem_prod((obs_cCMB_lenc(iyear)+dzero_dum),
         log(elem_div((pred_cCMB_lenc(iyear)+dzero_dum),
           (obs_cCMB_lenc(iyear)+dzero_dum)));
   }
  }
  fval+=w_lc*f_cCMB_lenc;
  fval_unwgt+=f_cCMB_lenc;


  f_HB_lenc=0.0;
  for (iyear=styr_HB_lenc; iyear<=endyr_HB_lenc; iyear++)
  {
   if (nsamp_HB_lenc(iyear)>=minSS_HB_lenc)
   {
   f_HB_lenc-=nsamp_HB_lenc(iyear)*
      sum( elem_prod((obs_HB_lenc(iyear)+dzero_dum),
         log(elem_div((pred_HB_lenc(iyear)+dzero_dum),
           (obs_HB_lenc(iyear)+dzero_dum)));
   }
  }
  fval+=w_lc*f_HB_lenc;
```

```
fval_unwgt+=f_HB_lenc;


f_HB_D_lenc=0.0;
for (iyear=styr_HB_D_lenc; iyear<=endyr_HB_D_lenc; iyear++)
{
 if (nsamp_HB_D_lenc(iyear)>=minSS_HB_D_lenc)
 {
 f_HB_D_lenc-=nsamp_HB_D_lenc(iyear)*
    sum( elem_prod((obs_HB_D_lenc(iyear)+dzero_dum),
       log(elem_div((pred_HB_D_lenc(iyear)+dzero_dum),
         (obs_HB_D_lenc(iyear)+dzero_dum)))));
 }
}
fval+=w_lc*f_HB_D_lenc;
fval_unwgt+=f_HB_D_lenc;


f_MRFSS_lenc=0.0;
for (iyear=styr_MRFSS_lenc; iyear<=endyr_MRFSS_lenc; iyear++)
{
 if (nsamp_MRFSS_lenc(iyear)>=minSS_MRFSS_lenc)
 {
 f_MRFSS_lenc-=nsamp_MRFSS_lenc(iyear)*
    sum( elem_prod((obs_MRFSS_lenc(iyear)+dzero_dum),
       log(elem_div((pred_MRFSS_lenc(iyear)+dzero_dum),
         (obs_MRFSS_lenc(iyear)+dzero_dum)))));
 }
}
fval+=w_lc*f_MRFSS_lenc;
fval_unwgt+=f_MRFSS_lenc;


//---Age comps--------------------------------
 f_CVT_agec=0.0;
 for (iyear=styr_CVT_agec; iyear<=endyr_CVT_agec; iyear++)
 {
  if (nsamp_CVT_agec(iyear)>=minSS_CVT_agec)
  {
  f_CVT_agec-=nsamp_CVT_agec(iyear)*
     sum(elem_prod((obs_CVT_agec(iyear)+dzero_dum),
        log(elem_div((pred_CVT_agec(iyear)+dzero_dum),
          (obs_CVT_agec(iyear)+dzero_dum)))));
  }
 }
 fval+=w_ac*f_CVT_agec;
 fval_unwgt+=f_CVT_agec;


 f_cHAL_agec=0.0;
 for (iyear=1; iyear<=nyr_cHAL_agec; iyear++)
 {
  if (nsamp_cHAL_agec(iyear)>=minSS_cHAL_agec)
  {
  f_cHAL_agec-=nsamp_cHAL_agec(iyear)*
        sum(elem_prod((obs_cHAL_agec(iyear)+dzero_dum),
          log(elem_div((pred_cHAL_agec(iyear)+dzero_dum),
            (obs_cHAL_agec(iyear)+dzero_dum)))));
  }
 }
 fval+=w_ac*f_cHAL_agec;
 fval_unwgt+=f_cHAL_agec;


 f_HB_agec=0.0;
 for (iyear=1; iyear<=nyr_HB_agec; iyear++)
 {
  if (nsamp_HB_agec(iyear)>=minSS_HB_agec)
  {
  f_HB_agec-=nsamp_HB_agec(iyear)*
        sum(elem_prod((obs_HB_agec(iyear)+dzero_dum),
          log(elem_div((pred_HB_agec(iyear)+dzero_dum),
            (obs_HB_agec(iyear)+dzero_dum)))));
  }
 }
 fval+=w_ac*f_HB_agec;
 fval_unwgt+=f_HB_agec;


 f_MRFSS_agec=0.0;
```

```
 for (iyear=styr_MRFSS_agec; iyear<=endyr_MRFSS_agec; iyear++)
 {
  if (nsamp_MRFSS_agec(iyear)>=minSS_MRFSS_agec)
  {
  f_MRFSS_agec-=nsamp_MRFSS_agec(iyear)*
      sum(elem_prod((obs_MRFSS_agec(iyear)+dzero_dum),
          log(elem_div((pred_MRFSS_agec(iyear)+dzero_dum),
              (obs_MRFSS_agec(iyear)+dzero_dum)))));
  }
 }
 fval+=w_ac*f_MRFSS_agec;
 fval_unwgt+=f_MRFSS_agec;


//-----------Constraints and penalties---------------------------------
 f_N_dev=0.0;
 //f_N_dev=norm2(log_dev_N_rec);
 f_N_dev=pow(log_dev_N_rec(styr_rec_dev),2);
 for(iyear=(styr_rec_dev+1); iyear<=endyr; iyear++)
 {
  f_N_dev+=pow(log_dev_N_rec(iyear)-R_autocorr*log_dev_N_rec(iyear-1),2);
 }
 fval+=w_R*f_N_dev;

//  f_N_dev_early=0.0;
//  f_N_dev_early=norm2(log_dev_N_rec(styr_rec_dev,(styr_rec_dev+5)));
//  fval+=w_R_init*f_N_dev_early;

 f_N_dev_end=0.0; //last BiasCor_exclude_yrs yrs
 if (BiasCor_exclude_yrs>0)
 {
  f_N_dev_end=norm2(log_dev_N_rec((endyr-BiasCor_exclude_yrs+1),endyr));
 }
 fval+=w_R_end*f_N_dev_end;

//  f_B1dB0_constraint=0.0;
//    f_B1dB0_constraint=square(totB(styr)/B0-B1dB0);
//  fval+=w_B1dB0*f_B1dB0_constraint;

 f_Fend_constraint=0.0; //last 3 yrs
 f_Fend_constraint=norm2(first_difference(fullF(endyr-2,endyr)));
 fval+=w_F*f_Fend_constraint;

 f_fullF_constraint=0.0;
 for (iyear=styr; iyear<=endyr; iyear++)
  {
   if (fullF(iyear)>3.0)
   {
   f_fullF_constraint+=square(fullF(iyear)-3.0);
   }
  }
 fval+=w_fullF*f_fullF_constraint;

//  f_cvlen_diff_constraint=0.0;
//    f_cvlen_diff_constraint=norm2(first_difference(log_len_cv_dev));
//  fval+=w_cvlen_diff*f_cvlen_diff_constraint;
//
//  f_cvlen_dev_constraint=0.0;
//    f_cvlen_dev_constraint=norm2(log_len_cv_dev);
//  fval+=w_cvlen_dev*f_cvlen_dev_constraint;

 //cout << "fval = " << fval << "  fval_unwgt = " << fval_unwgt << endl;


REPORT_SECTION

 //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();
 //cout<<"got per recruit"<<endl;
 cout <<endl;
 cout << "><>--><>--><>--><>--><>--><>--><>--><>--><>--><>"  <<endl;
 cout << "BC Fmsy=" << F_msy_out<< "   BC SSBmsy=" << SSB_msy_out <<endl;
 cout <<"F status="<<FdF_msy_end<<endl;
 cout <<"Pop status="<<SdSSB_msy_end<<endl;
```

```
 cout << "h="<<steep<<"   R0="<<R0<<endl;
 cout << "><>--><>--><>--><>--><>--><>--><>--><>--><>--><>"  <<endl;

 //cout << "x_dum="<<x_dum<<endl;

 report << "TotalLikelihood " << fval << endl;
 report<<" "<<endl;

 report << "Bias-corrected (BC) MSY stuff" << endl;
 report << "BC Fmsy " << F_msy_out << endl;
 report << "BC SSBmsy " << SSB_msy_out << endl;
 report << "BC Rmsy " << R_msy_out << endl;
 report << "BC Bmsy " << B_msy_out << endl;
 report << "BC MSY " << msy_out << endl;
 report << "BC F/Fmsy " << fullF/F_msy_out << endl;
 report << "BC SSB/SSBmsy " << SSB/SSB_msy_out << endl;
 report << "BC B/Bmsy " << totB/B_msy_out << endl;
 report << "BC Yield/MSY " << L_total_klb_yr/msy_out <<endl;
 report << "BC F(2007)/Fmsy " << fullF(endyr)/F_msy_out << endl;
 report << "BC SSB(2007)/SSBmsy " << SSB(endyr)/SSB_msy_out << endl;
 report << "BC Predicted Landings(2007)/MSY " << L_total_klb_yr(endyr)/msy_out <<endl;
 report  << " "<<endl;

 report << "Stock-Recruit " << endl;
 report << "R0= " << R0 << endl;
 report << "Steepness= " << steep << endl;
 report << "spr_F0= " << spr_F0 << endl;
 report<<" "<<endl;

#include "vs_make_Robject4.cxx"   // write the S-compatible report
```

II. Data input file (vs3.dat). Be aware that the following contains wrap-around text that ADMB may require as single-line input.

```
##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
##  Data Input File
##  SEDAR17 Assessment: Vermilion Snapper
##
##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>

#starting and ending year of model
1946
2007
#Starting year to estimate recruitment deviation from S-R curve
1976
#4 periods of rec size regs: yr1-91 no restrictions, 1992-98 10-inch TL, 1999-06 11-in TL, 2006-07 12-in TL
#2 periods of comm size regs: yr1-91 no restrictions, 1992-2007 12-in TL
#ending years of regulation period
1991
1998
2006
#Size limits of periods 2, 3, 4 (in mm)
253.8071
279.1878
304.5685

#Number of ages (16 classes is 1,...,12+)
12
#vector of agebins, last is a plus group
1 2 3 4 5 6 7 8 9 10 11 12
#number length bins used to match length comps and number used to compute plus group
46
20
#Vector of length bins (mm)(midpoint of bin) used to match length comps and bins used to compute plus group
150 160 170 180 190 200 210 220 230 240 250 260
    270 280 290 300 310 320 330 340 350 360 370
    380 390 400 410 420 430 440 450 460 470 480
    490 500 510 520 530 540 550 560 570 580 590
    600
610 620 630 640 650 660 670 680 690 700 710 720
    730 740 750 760 770 780 790 800
#discard mortality constant
0.41  #comm HAL
0.38  #headboat
0.38  #MRFSS
#max value of F used in spr and msy calculations
3.0
#number of iterations in spr calculations
30001
#number of iterations in msy calculations
30001
#Number years at end of time series over which to average sector F's, for weighted selectivities
3
#multiplicative bias correction of recruitment (may set to 1.0 for none or negative to compute from recruitment variance)
-1.0
#number yrs to exclude at end of time series for computing bias correction (end rec devs may have extra constraint)
3
#starting values for VonBert params (Linf, K, t0), units in mm TL
506.0
0.12
-3.5
#starting value of constant cv of length at age
0.22
#length-length (FL-TL) coefficients a and b, FL=a+b*TL, (FL, TL in mm)
0.371
0.898
#length-weight (TL-whole wgt) coefficients a and b, W=aL^b, (W in g, TL in mm)
2.1E-05
2.907
#time-invariant vector of % maturity-at-age for females (ages 1-12)
0.8 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
#time-variant vector of proportion female (ages 1-12)
0.715 0.715 0.715 0.715 0.715 0.715 0.715 0.715 0.715 0.715 0.715 0.715
#Fecundity-length relationship: Eggs/batch=aFL^b (length in FL); batches in number batches per yr
31.0    #batches per year, in the range (27,35)
0.0438  #parameter a
2.508   #parameter b

0.5     #time of year (as fraction) for spawning
```

```
#scalar to divide actual annual egg production (may want to present in units 10^X eggs)
1.0E12
#1.0

#age-dependent natural mortality at age
0.341 0.304 0.278 0.258 0.243 0.231 0.222 0.214 0.208 0.202 0.198 0.194
#age-independent natural mortality (used only to compute MSST=(1-M)SSBmsy)
0.22

#Spawner-recruit parameters
#steepness (fixed or initial guess)
0.56
#log_R0 - log virgin recruitment (initial guess set near soln from exploratory runs)
15.0
# R autocorrelation
0.0

#R1_mult: R(styr)=R1_mult*R1. (This parameter is not used if initial stock is at virgin.)
1.0
#B1/B0 (1.0 sets initial biomass to virgin conditions)
1.0
#########################################################################################################
###MARMAP FL snapper trap index#############################################################################
#Starting and ending years of time series, respectively
1983
1987
#Observed CPUE (numbers) and CV vectors, respectively
1.426 0.721 1.175 1.179 0.499
0.300 0.283 0.272 0.238 0.247
#Starting and ending year of length composition data
1983
1987
#sample sizes of length comps by year
469 354 608 471 290
#length composition by year (year,lengthbin 1cm)
0.00000 0.00000 0.00426 0.00640 0.05544 0.13646 0.12367 0.12793 0.09808 0.09168 0.06397 0.07036
        0.05757 0.06397 0.02985 0.01706 0.01066 0.01066 0.00426 0.01066 0.01066 0.00213 0.00426
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000
0.00000 0.00000 0.00000 0.01695 0.05085 0.16667 0.16384 0.17232 0.13559 0.11017 0.08475 0.03955
        0.01412 0.01130 0.00847 0.00282 0.00000 0.00565 0.00565 0.00282 0.00000 0.00000 0.00000
        0.00000 0.00565 0.00000 0.00000 0.00000 0.00000 0.00282 0.00000 0.00000 0.00000 0.00000
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000
0.00000 0.00000 0.00164 0.01316 0.05592 0.14145 0.15954 0.16612 0.12171 0.09539 0.11349 0.05099
        0.02796 0.02632 0.00658 0.00493 0.00329 0.00329 0.00329 0.00000 0.00164 0.00000 0.00000
        0.00329 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000
0.00000 0.00000 0.00637 0.03397 0.09979 0.23142 0.17197 0.15711 0.09766 0.05096 0.05945 0.03185
        0.01699 0.01062 0.00425 0.01699 0.00425 0.00212 0.00000 0.00000 0.00425 0.00000 0.00000
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000
0.00000 0.00345 0.00690 0.06207 0.12069 0.19310 0.18621 0.13793 0.10345 0.06552 0.02414 0.02414
        0.02069 0.00690 0.01034 0.01724 0.00690 0.00000 0.00345 0.00000 0.00345 0.00345 0.00000
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
        0.00000
#########################################################################################################

################MARMAP Chevron trap index#################################################################
#Starting and ending years of time series, respectively
1990
2007
#Observed CPUE (numbers) and CV vectors, respectively
0.568 2.541 1.314 1.052 2.026 1.069 1.182 0.695 0.640 0.883 0.956 0.994
      1.301 0.605 0.507 0.532 0.368 0.769
0.234 0.200 0.227 0.198 0.189 0.203 0.202 0.224 0.215 0.242 0.223 0.235
      0.218 0.300 0.237 0.216 0.258 0.232
##Starting and ending year of length composition data
#1990
#2007
##sample sizes of length comps by year
#830 3066 1514 1326 3350 1786 3162 1805 1249 735 1712 1369
    1742 245 457 772 366 1240
##length composition samples (year,lengthbin 1cm)
#0.00000 0.00000 0.00000 0.00602 0.13373 0.27952 0.19518 0.13253 0.10120 0.05904 0.03133 0.01928
         0.01446 0.00964 0.00241 0.00482 0.00120 0.00482 0.00241 0.00120 0.00000 0.00000 0.00000
```

	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00120
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00261	0.02871	0.14910	0.27863	0.22904	0.15498	0.07537	0.03426	0.01436	0.00946
	0.00587	0.00392	0.00261	0.00196	0.00163	0.00228	0.00228	0.00163	0.00033	0.00033	0.00033
	0.00000	0.00000	0.00033	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00991	0.12285	0.24703	0.31638	0.14597	0.07662	0.03303	0.01453	0.00727
	0.01057	0.00661	0.00330	0.00198	0.00066	0.00132	0.00066	0.00000	0.00000	0.00000	0.00066
	0.00000	0.00000	0.00000	0.00066	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00075	0.03243	0.20513	0.22926	0.19532	0.12519	0.08069	0.04148	0.02866	0.01433
	0.01508	0.00830	0.00830	0.00528	0.00377	0.00302	0.00151	0.00075	0.00075	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00836	0.09134	0.20179	0.23642	0.14687	0.10537	0.06776	0.04179	0.02806
	0.02299	0.01284	0.01075	0.00836	0.00507	0.00358	0.00239	0.00209	0.00090	0.00119	0.00090
	0.00060	0.00030	0.00030	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00280	0.00616	0.06831	0.15789	0.17357	0.15733	0.14614	0.09071	0.05543	0.04087	0.03191
	0.02576	0.01456	0.01176	0.00560	0.00504	0.00280	0.00056	0.00168	0.00112	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00316	0.01486	0.03953	0.12872	0.17394	0.16034	0.13030	0.10879	0.07242	0.04554
	0.03036	0.02056	0.01708	0.01581	0.00949	0.00949	0.00380	0.00316	0.00380	0.00253	0.00190
	0.00221	0.00032	0.00095	0.00032	0.00032	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00032	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00055	0.01163	0.07812	0.13850	0.16842	0.17839	0.12299	0.08532	0.06260
	0.04100	0.03712	0.01994	0.01717	0.01551	0.00776	0.00443	0.00388	0.00388	0.00055	0.00055
	0.00055	0.00055	0.00055	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00080	0.00240	0.02322	0.07526	0.08327	0.12330	0.11849	0.13050	0.11209	0.08727
	0.07366	0.05845	0.04564	0.02082	0.01441	0.01361	0.00801	0.00320	0.00080	0.00320	0.00080
	0.00000	0.00080	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.01088	0.05034	0.10204	0.08980	0.11973	0.15782	0.12925	0.07619	0.08027
	0.05442	0.03129	0.03129	0.02177	0.01633	0.00680	0.00816	0.00136	0.00272	0.00136	0.00544
	0.00272	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00117	0.01519	0.03446	0.06834	0.09696	0.08294	0.08879	0.08703	0.09346
	0.07360	0.07126	0.05958	0.04614	0.05199	0.03680	0.03037	0.01752	0.01285	0.00818	0.00292
	0.00467	0.00467	0.00117	0.00292	0.00292	0.00234	0.00117	0.00058	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00000	0.00657	0.02557	0.05844	0.09058	0.09861	0.08766	0.09642	0.10080
	0.07670	0.07670	0.07889	0.04675	0.03652	0.02995	0.02776	0.01607	0.01607	0.00877	0.00730
	0.00438	0.00365	0.00073	0.00219	0.00146	0.00000	0.00073	0.00073	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00000	0.01091	0.06315	0.11481	0.11137	0.11481	0.08726	0.08955	0.09185
	0.08209	0.07635	0.04822	0.03961	0.02641	0.01435	0.01263	0.00689	0.00172	0.00057	0.00172
	0.00057	0.00287	0.00115	0.00000	0.00057	0.00000	0.00000	0.00000	0.00000	0.00000	0.00057
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00000	0.02041	0.04898	0.03265	0.11429	0.13061	0.07755	0.07755	0.06939
	0.05714	0.07347	0.08163	0.08571	0.03265	0.03673	0.02449	0.00816	0.00408	0.00000	0.00816
	0.00408	0.00408	0.00408	0.00000	0.00000	0.00408	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00219	0.00438	0.00219	0.00219	0.02188	0.02845	0.06127	0.10066	0.10503	0.12691	0.14661	0.14442
	0.05470	0.04595	0.03939	0.02407	0.01313	0.00875	0.02188	0.01313	0.01094	0.00219	0.00875
	0.00219	0.00438	0.00219	0.00000	0.00000	0.00219	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00000	0.00130	0.00000	0.02591	0.07254	0.10492	0.13083	0.11528	0.09326	0.07642
	0.05440	0.05181	0.06736	0.04663	0.02979	0.02720	0.02979	0.01943	0.01295	0.01036	0.00389
	0.00648	0.00648	0.00777	0.00259	0.00259	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
#0.00000	0.00000	0.00546	0.01093	0.03005	0.05738	0.04098	0.04645	0.05738	0.09836	0.13934	0.16393
	0.08470	0.06284	0.05191	0.03825	0.03005	0.02186	0.01913	0.01366	0.01093	0.00546	0.00273

0.00000 0.00000 0.00546 0.00273 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
#0.00000 0.00000 0.00081 0.00645 0.00726 0.02823 0.05403 0.09758 0.09516 0.09194 0.08065 0.08548
0.08226 0.07097 0.07903 0.05484 0.04516 0.02742 0.02419 0.01129 0.00887 0.01290 0.00968
0.00645 0.01048 0.00323 0.00000 0.00242 0.00323 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
#Number and vector of years of age composition data
2002
2007
#sample sizes of age comps by year
765 215 305 481 272 536
#1720 243 355 707 364 1151
#age composition samples (year,age)
0.0000 0.2302 0.2593 0.1738 0.2134 0.0558 0.0285 0.0233 0.0110 0.0047 0.0000 0.0000
0.0000 0.2757 0.3498 0.2016 0.0700 0.1029 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.1014 0.3606 0.1944 0.1577 0.0676 0.0845 0.0338 0.0000 0.0000 0.0000 0.0000
0.0000 0.1697 0.2405 0.3083 0.1372 0.0693 0.0311 0.0438 0.0000 0.0000 0.0000 0.0000
0.0714 0.1016 0.1978 0.1593 0.3407 0.0797 0.0385 0.0110 0.0000 0.0000 0.0000 0.0000
0.0043 0.5995 0.1077 0.0643 0.0617 0.1043 0.0374 0.0104 0.0026 0.0052 0.0026 0.0000

###

####################Commercial Hook and Line fishery landings########################
#Commercial Hook and Line CPUE Index from Logbook
#Starting and ending years of CPUE index
1993
2007
##Observed CPUE and assumed CVs
0.661 0.731 0.840 0.738 0.860 0.840 1.043 1.255 1.275 1.185 0.963 1.110
1.310 1.079 1.111
0.284 0.255 0.224 0.251 0.248 0.261 0.259 0.262 0.244 0.245 0.295 0.294
0.273 0.300 0.275
#Commercial Hook and Line fishery landings
#Starting and ending years of landings time series, respectively
1958
2007
#Observed landings (1000 lb whole weight) and assumed CVs
0.1941 1.2619 1.7472 19.3167 10.8222 20.9669 6.7919 22.0094 3.3974 14.1721 31.9357 31.3475
19.5109 66.3214 68.7938 86.1925 119.3868 218.6555 212.4096 273.3219 345.0756 430.8881 482.6361
500.8858 672.7962 645.7321 734.0766 920.5064 896.3793 697.9285 854.2274 1041.5094 1141.1896 1332.6934
764.9364 866.3612 948.4258 928.4970 743.6924 759.0050 708.1121 876.5837 1348.5193 1633.5940 1334.4184
727.8589 1086.2997 1100.9156 827.1596 1012.6122
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05
#Starting and ending years of discards time series, respectively
1992
2007
#Observed discards (1000 fish) and assumed CVs
75.0240 84.2710 105.6070 127.4570 164.3450 152.6960 119.8030 100.0770 104.7820 124.9080 237.0200 98.2840
49.7620 74.9450 47.1930 51.7740
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05
#Starting and ending years of commercial hook and line length composition sample data
1983
2007
#sample size of commercial length comp data by year
391 7976 9800 7888 7315 5577 5625 6007 10525 6161 8286 7488
13246 6718 6793 6644 11961 18712 18034 12024 11977 13458 8604
11124 6769
#commercial length composition samples (year,lengthbin 30mm)
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0051
0.0026 0.0077 0.0102 0.0230 0.0281 0.0384 0.0537 0.0358 0.0588 0.0716 0.0512
0.0409 0.0537 0.0358 0.0460 0.0460 0.0435 0.0409 0.0256 0.0512 0.0307 0.0281
0.0128 0.0384 0.0256 0.0256 0.0256 0.0128 0.0077 0.0102 0.0102 0.0000 0.0000
0.0026
0.0000 0.0000 0.0000 0.0000 0.0001 0.0001 0.0000 0.0009 0.0029 0.0090 0.0176 0.0311
0.0452 0.0572 0.0582 0.0617 0.0622 0.0647 0.0479 0.0549 0.0508 0.0492 0.0374
0.0384 0.0336 0.0293 0.0263 0.0239 0.0200 0.0211 0.0192 0.0183 0.0163 0.0142
0.0136 0.0126 0.0144 0.0124 0.0100 0.0068 0.0058 0.0042 0.0036 0.0018 0.0010
0.0018
0.0000 0.0000 0.0001 0.0000 0.0000 0.0000 0.0003 0.0008 0.0036 0.0090 0.0187 0.0313
0.0387 0.0493 0.0527 0.0486 0.0569 0.0555 0.0597 0.0540 0.0499 0.0521 0.0528
0.0492 0.0378 0.0351 0.0326 0.0298 0.0269 0.0201 0.0200 0.0192 0.0139 0.0137

	0.0125	0.0124	0.0090	0.0110	0.0067	0.0049	0.0043	0.0032	0.0021	0.0005	0.0005
	0.0005										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0004	0.0011	0.0013	0.0034	0.0108	0.0194	0.0324
	0.0429	0.0514	0.0759	0.0673	0.0793	0.0639	0.0595	0.0504	0.0532	0.0398	0.0474
	0.0447	0.0306	0.0353	0.0285	0.0224	0.0204	0.0170	0.0183	0.0118	0.0115	0.0112
	0.0080	0.0079	0.0060	0.0070	0.0069	0.0041	0.0033	0.0021	0.0017	0.0006	0.0000
	0.0010										
0.0000	0.0000	0.0000	0.0001	0.0000	0.0003	0.0011	0.0015	0.0056	0.0140	0.0207	0.0350
	0.0483	0.0559	0.0656	0.0683	0.0708	0.0665	0.0664	0.0633	0.0568	0.0539	0.0405
	0.0310	0.0299	0.0251	0.0223	0.0235	0.0213	0.0192	0.0165	0.0133	0.0136	0.0105
	0.0085	0.0063	0.0070	0.0062	0.0042	0.0023	0.0023	0.0014	0.0002	0.0003	0.0002
	0.0002										
0.0000	0.0000	0.0000	0.0001	0.0000	0.0002	0.0014	0.0045	0.0096	0.0182	0.0395	0.0517
	0.0716	0.0679	0.0794	0.0709	0.0676	0.0622	0.0586	0.0556	0.0413	0.0389	0.0365
	0.0310	0.0283	0.0213	0.0201	0.0177	0.0186	0.0148	0.0121	0.0103	0.0104	0.0081
	0.0078	0.0079	0.0057	0.0023	0.0026	0.0013	0.0017	0.0006	0.0015	0.0003	0.0000
	0.0000										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0006	0.0023	0.0052	0.0095	0.0157	0.0285
	0.0403	0.0640	0.0711	0.0607	0.0911	0.0880	0.0590	0.0719	0.0412	0.0430	0.0583
	0.0282	0.0298	0.0266	0.0239	0.0276	0.0210	0.0209	0.0143	0.0138	0.0104	0.0073
	0.0055	0.0049	0.0049	0.0030	0.0018	0.0018	0.0020	0.0012	0.0003	0.0004	0.0000
	0.0002										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0005	0.0020	0.0058	0.0113	0.0200	0.0343
	0.0400	0.0533	0.0612	0.0608	0.0717	0.0682	0.0587	0.0632	0.0604	0.0450	0.0417
	0.0441	0.0357	0.0315	0.0265	0.0284	0.0212	0.0241	0.0163	0.0176	0.0092	0.0170
	0.0096	0.0066	0.0040	0.0027	0.0031	0.0016	0.0007	0.0003	0.0002	0.0004	0.0002
	0.0008										
0.0000	0.0000	0.0000	0.0001	0.0000	0.0005	0.0024	0.0072	0.0250	0.0331	0.0422	0.0434
	0.0531	0.0595	0.0555	0.0619	0.0590	0.0495	0.0491	0.0504	0.0478	0.0312	0.0439
	0.0422	0.0362	0.0331	0.0284	0.0267	0.0263	0.0158	0.0143	0.0120	0.0081	0.0095
	0.0080	0.0058	0.0050	0.0038	0.0033	0.0026	0.0019	0.0010	0.0006	0.0002	0.0002
	0.0004										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0002	0.0012	0.0018	0.0017
	0.0036	0.0084	0.0226	0.0509	0.0886	0.0860	0.0954	0.0890	0.0803	0.0714	0.0698
	0.0582	0.0487	0.0414	0.0345	0.0312	0.0265	0.0216	0.0151	0.0120	0.0108	0.0090
	0.0049	0.0043	0.0041	0.0020	0.0020	0.0008	0.0006	0.0011	0.0001	0.0002	0.0000
	0.0000										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	0.0000	0.0001	0.0000	0.0000	0.0001
	0.0013	0.0047	0.0121	0.0608	0.1186	0.0793	0.0842	0.0771	0.0650	0.0445	0.0586
	0.0569	0.0422	0.0393	0.0402	0.0340	0.0340	0.0270	0.0226	0.0204	0.0182	0.0140
	0.0143	0.0090	0.0078	0.0047	0.0031	0.0032	0.0011	0.0008	0.0002	0.0001	0.0000
	0.0004										
0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0006	0.0001	0.0001	0.0001
	0.0004	0.0012	0.0086	0.0482	0.0981	0.0867	0.0862	0.0821	0.0715	0.0487	0.0585
	0.0586	0.0524	0.0479	0.0348	0.0376	0.0327	0.0259	0.0260	0.0196	0.0140	0.0144
	0.0130	0.0080	0.0063	0.0053	0.0031	0.0020	0.0032	0.0009	0.0010	0.0008	0.0001
	0.0010										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0001	0.0008	0.0020
	0.0032	0.0036	0.0115	0.0551	0.0856	0.0963	0.0992	0.0899	0.0757	0.0517	0.0674
	0.0529	0.0494	0.0435	0.0371	0.0297	0.0314	0.0261	0.0204	0.0156	0.0105	0.0118
	0.0088	0.0056	0.0050	0.0025	0.0013	0.0021	0.0016	0.0011	0.0008	0.0002	0.0001
	0.0006										
0.0000	0.0000	0.0000	0.0000	0.0004	0.0004	0.0016	0.0008	0.0012	0.0004	0.0004	0.0000
	0.0022	0.0018	0.0243	0.0649	0.0942	0.0934	0.1046	0.0934	0.0795	0.0586	0.0526
	0.0489	0.0324	0.0351	0.0362	0.0295	0.0295	0.0223	0.0213	0.0151	0.0108	0.0111
	0.0086	0.0076	0.0054	0.0029	0.0026	0.0013	0.0018	0.0008	0.0011	0.0004	0.0001
	0.0005										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0007
	0.0021	0.0085	0.0439	0.0764	0.1010	0.0851	0.0924	0.0724	0.0581	0.0483	0.0417
	0.0369	0.0333	0.0362	0.0394	0.0294	0.0359	0.0250	0.0250	0.0210	0.0172	0.0149
	0.0117	0.0111	0.0073	0.0083	0.0051	0.0049	0.0025	0.0016	0.0014	0.0003	0.0003
	0.0005										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	0.0001	0.0002	0.0002	0.0005
	0.0023	0.0073	0.0373	0.0812	0.0924	0.0927	0.0872	0.0841	0.0712	0.0517	0.0537
	0.0445	0.0401	0.0532	0.0334	0.0286	0.0308	0.0259	0.0193	0.0141	0.0101	0.0091
	0.0064	0.0049	0.0033	0.0042	0.0029	0.0028	0.0022	0.0008	0.0005	0.0000	0.0005
	0.0003										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0002	0.0003
	0.0032	0.0050	0.0174	0.0589	0.0789	0.0840	0.0752	0.0770	0.0718	0.0418	0.0588
	0.0550	0.0493	0.0459	0.0472	0.0423	0.0408	0.0338	0.0224	0.0196	0.0115	0.0122
	0.0097	0.0087	0.0080	0.0057	0.0045	0.0039	0.0023	0.0020	0.0012	0.0007	0.0003
	0.0004										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	0.0000	0.0000
	0.0005	0.0016	0.0095	0.0448	0.0688	0.0949	0.0867	0.0865	0.0700	0.0353	0.0600
	0.0563	0.0465	0.0474	0.0457	0.0469	0.0439	0.0336	0.0279	0.0238	0.0093	0.0161
	0.0112	0.0102	0.0059	0.0046	0.0039	0.0025	0.0019	0.0013	0.0006	0.0009	0.0003
	0.0005										
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0002	0.0005
	0.0010	0.0021	0.0126	0.0463	0.0649	0.0871	0.0790	0.0817	0.0728	0.0344	0.0677
	0.0552	0.0415	0.0419	0.0402	0.0397	0.0351	0.0311	0.0284	0.0268	0.0101	0.0240

0.0145 0.0157 0.0113 0.0115 0.0081 0.0075 0.0019 0.0017 0.0005 0.0019 0.0003 0.0009
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0000 0.0005
0.0080 0.0134 0.0286 0.0696 0.0822 0.0970 0.0793 0.0747 0.0573 0.0315 0.0483
0.0482 0.0418 0.0321 0.0424 0.0424 0.0360 0.0310 0.0278 0.0194 0.0098 0.0146
0.0125 0.0107 0.0102 0.0091 0.0086 0.0060 0.0018 0.0022 0.0004 0.0008 0.0006 0.0009
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0002 0.0006 0.0005 0.0000
0.0003 0.0019 0.0173 0.0593 0.0667 0.0812 0.0855 0.0730 0.0723 0.0373 0.0592
0.0721 0.0555 0.0541 0.0460 0.0487 0.0339 0.0319 0.0207 0.0197 0.0079 0.0127
0.0115 0.0082 0.0070 0.0044 0.0036 0.0032 0.0017 0.0009 0.0003 0.0003 0.0002 0.0002
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0021 0.0001
0.0001 0.0010 0.0081 0.0416 0.0522 0.0840 0.0898 0.0799 0.0835 0.0293 0.0745
0.0622 0.0526 0.0478 0.0416 0.0438 0.0346 0.0303 0.0250 0.0274 0.0105 0.0148
0.0162 0.0143 0.0092 0.0099 0.0051 0.0029 0.0033 0.0006 0.0004 0.0004 0.0001 0.0005
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0002 0.0007 0.0085 0.0371 0.0424 0.0621 0.0662 0.0758 0.0674 0.0323 0.0605
0.0738 0.0568 0.0531 0.0525 0.0562 0.0420 0.0434 0.0308 0.0263 0.0180 0.0192
0.0195 0.0143 0.0104 0.0105 0.0058 0.0051 0.0034 0.0022 0.0009 0.0011 0.0004 0.0011
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0000 0.0001 0.0003 0.0006
0.0026 0.0090 0.0141 0.0351 0.0591 0.0725 0.0848 0.0825 0.0821 0.0540 0.0961
0.0685 0.0490 0.0492 0.0410 0.0389 0.0277 0.0274 0.0210 0.0221 0.0101 0.0103
0.0115 0.0118 0.0063 0.0034 0.0034 0.0026 0.0008 0.0012 0.0001 0.0006 0.0000 0.0005
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0002
0.0009 0.0048 0.0088 0.0335 0.0374 0.0634 0.0655 0.0792 0.0842 0.0594 0.0680
0.0728 0.0602 0.0534 0.0427 0.0424 0.0377 0.0324 0.0278 0.0305 0.0150 0.0210
0.0180 0.0094 0.0092 0.0075 0.0045 0.0039 0.0024 0.0019 0.0007 0.0006 0.0003 0.0006

#Number and vector of years of age compositions for hook and line fishery
15
1992 1993 1994 1995 1997 1998 1999 2000 2001 2002 2003 2004
2005 2006 2007
#sample sizes of age comps by year
82 183 164 317 55 104 136 209 244 181 122 512
727 938 1011
#age composition samples (year,age)
0.0000 0.0015 0.0936 0.4183 0.3508 0.1204 0.0154 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0108 0.1931 0.2994 0.2360 0.1080 0.1174 0.0274 0.0068 0.0000 0.0007 0.0005
0.0000 0.0000 0.0226 0.3441 0.2832 0.1966 0.0821 0.0611 0.0092 0.0001 0.0008 0.0001
0.0000 0.0058 0.2303 0.3334 0.2853 0.0726 0.0453 0.0242 0.0017 0.0001 0.0001 0.0011
0.0000 0.0000 0.0000 0.1401 0.2649 0.3417 0.1936 0.0209 0.0000 0.0000 0.0388 0.0000
0.0000 0.1506 0.2194 0.2051 0.1315 0.1644 0.1156 0.0085 0.0000 0.0000 0.0049 0.0000
0.0000 0.1555 0.6672 0.1546 0.0109 0.0000 0.0119 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0922 0.3145 0.3615 0.1445 0.0668 0.0098 0.0051 0.0056 0.0000 0.0000 0.0000
0.0000 0.0806 0.4683 0.3494 0.0875 0.0030 0.0026 0.0018 0.0008 0.0060 0.0000 0.0000
0.0000 0.0854 0.2933 0.4501 0.0920 0.0322 0.0200 0.0130 0.0139 0.0000 0.0000 0.0000
0.0000 0.0362 0.1976 0.3248 0.1220 0.1714 0.1111 0.0283 0.0069 0.0017 0.0000 0.0000
0.0000 0.0286 0.2768 0.3524 0.1201 0.1030 0.0702 0.0351 0.0111 0.0027 0.0000 0.0000
0.0000 0.0138 0.0982 0.3549 0.2342 0.1129 0.0630 0.0822 0.0306 0.0018 0.0078 0.0006
0.0011 0.0376 0.0646 0.1667 0.3317 0.2111 0.0839 0.0366 0.0455 0.0117 0.0081 0.0014
0.0012 0.1021 0.2588 0.1253 0.1611 0.1786 0.0952 0.0403 0.0107 0.0154 0.0097 0.0016

#Number and vector of years of comm HAL discard length compositions
1
2007
#sample sizes of discard length comp data by year
395
#cHAL discard length comp samples (year,lengthbin)
0.0000 0.0000 0.0000 0.0000 0.0000 0.0025 0.0025 0.0203 0.0203 0.0380 0.0405 0.0557
0.0886 0.1291 0.2608 0.2430 0.0380 0.0127 0.0051 0.0051 0.0101 0.0076 0.0051
0.0000 0.0051 0.0000 0.0000 0.0000 0.0000 0.0025 0.0051 0.0025 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

##########Commercial Combined (trawl + other) fishery landings (mostly EEZ trawl in 1976-1988, mostly traps or state-water trawl in other yrs)##########
#Starting and ending years of landings time series, respectively
1971
2007
#Observed landings (1000 lb whole weight) and CV's
0.3955 11.7898 6.1121 2.7278 2.8250 7.5215 11.2970 1.0465 54.1611 268.6132 242.8936 215.6655
142.7823 117.9559 24.9836 23.9772 51.6314 131.5371 90.0653 148.7127 61.4177 0.2775 8.5520
9.7341 2.8768 1.3942 2.0116 2.3942 4.5096 1.5921 3.2303 1.3379 6.9700 2.6764
0.8709 1.4604 7.6927
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05

```
          0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05
          0.05      0.05      0.05
#Number and vector of years of length compositions (comm combined)
5
1984      1986      1987      1988      1997
#sample sizes of length comp data by year
196       650       616       692       64
#commercial combined gear (data from trawls) length comp samples (year,age) (1cm length bins)
0.0155    0.1546    0.1649    0.1701    0.1237    0.0722    0.0361    0.0309    0.0258    0.0103    0.0000    0.0000
          0.0000    0.0000    0.0000    0.0052    0.0000    0.0000    0.0155    0.0052    0.0052    0.0052    0.0206
          0.0052    0.0052    0.0206    0.0052    0.0000    0.0206    0.0103    0.0155    0.0103    0.0103    0.0206
          0.0103    0.0052    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000
0.0031    0.0139    0.0277    0.0616    0.1094    0.1171    0.1248    0.1525    0.1017    0.0832    0.0478    0.0539
          0.0324    0.0308    0.0108    0.0077    0.0046    0.0062    0.0015    0.0031    0.0000    0.0000    0.0031
          0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0015    0.0000
          0.0015    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000
0.0101    0.0679    0.1421    0.1685    0.1527    0.1179    0.0923    0.0650    0.0512    0.0316    0.0302    0.0121
          0.0216    0.0115    0.0104    0.0055    0.0023    0.0011    0.0040    0.0011    0.0000    0.0011    0.0000
          0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000
0.0000    0.0072    0.0405    0.0780    0.1040    0.1329    0.0867    0.0795    0.0694    0.0361    0.0332    0.0159
          0.0260    0.0116    0.0087    0.0116    0.0072    0.0087    0.0058    0.0116    0.0116    0.0014    0.0087
          0.0058    0.0116    0.0188    0.0217    0.0202    0.0289    0.0087    0.0202    0.0188    0.0000    0.0087
          0.0072    0.0130    0.0130    0.0014    0.0029    0.0029    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000
0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000    0.0000    0.1875    0.1719    0.3281    0.1406    0.1406    0.0313    0.0000    0.0000    0.0000
          0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
          0.0000

####################Commercial historic trawl fishery landings#########################
#Commercial historic trawl fishery landings
#Starting and ending years of landings time series, respectively
1961
1962
#Observed landings (1000 lb whole weight) and assumed CVs
24.0246  42.5823
0.05  0.05
#####################################Headboat landings#######################################
##Starting and ending years for CPUE index
1976
2007
##Observed CPUE values (numbers) and CVs,
1.252     1.064     1.639     1.569     0.899     1.027     0.882     1.319     1.090     1.319     1.081     1.340
          1.431     1.140     1.147     1.066     0.498     0.500     0.497     0.536     0.586     0.826     0.688
          0.803     0.991     1.063     1.171     0.744     1.033     0.941     0.990     0.866
0.211     0.226     0.176     0.200     0.231     0.240     0.228     0.171     0.192     0.177     0.163     0.162
          0.156     0.208     0.196     0.209     0.252     0.259     0.272     0.276     0.269     0.289     0.249
          0.248     0.252     0.237     0.234     0.300     0.213     0.271     0.238     0.254
#Starting and ending years for landings time series
1947
2007
#Headboat landings vector (1000 fish) and CV's
14.632    30.581    46.801    63.431    80.610    98.478    117.175   136.839   157.611   179.629   203.033   227.963
          254.558   282.957   312.783   341.589   366.412   384.289   392.256   388.652   377.023   362.218   349.083
          342.467   345.461   402.814   383.908   421.690   477.319   399.737   317.303   487.529   425.382   322.990
          270.987   362.321   399.040   324.429   529.803   533.101   731.007   740.891   661.251   655.859   600.501
          345.266   327.027   369.720   354.766   340.340   364.742   341.563   381.936   428.235   418.876   335.543
          251.796   329.081   275.450   344.724   507.970
0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05
          0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05
          0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05
          0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05
          0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05      0.05
          0.05      0.05      0.05      0.05      0.05
#Starting and ending years of discards time series, respectively
1947
2007
#Observed discards (1000s) and assumed CVs
0.903     1.887     2.888     3.915     4.975     6.078     7.231     8.445     9.727     11.086    12.530    14.069
          15.710    17.463    19.303    21.081    22.613    23.716    24.208    23.986    23.268    22.354    21.544
          21.135    21.320    24.860    23.693    26.025    29.458    24.670    19.582    30.088    26.252    19.933
          16.724    22.361    24.627    20.022    32.697    32.900    45.114    45.724    40.809    40.476    37.060
          67.499    63.934    72.280    69.357    66.536    71.307    66.775    86.492    96.977    94.858    75.986
          57.021    87.969    52.502    76.340    127.773
```

0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05
#Starting and ending year of headboat length composition data
1976
2007
#sample sizes of length comp data by year
1146 1036 1768 1389 1348 1335 2777 4482 4545 5894 6159 6327
4759 4768 5308 4029 2823 3323 5724 4799 3858 4133 4239
4306 4469 3387 3895 3824 3324 2206 3209 3995
#HB length comp samples (year,lengthbin)
0.00000 0.00000 0.00016 0.00789 0.02274 0.04706 0.05065 0.03797 0.06469 0.05685 0.08667 0.06599
0.09133 0.06150 0.06378 0.03161 0.04668 0.04142 0.03303 0.02653 0.01877 0.01729 0.01321
0.01219 0.01789 0.00767 0.00888 0.00737 0.00440 0.00331 0.00456 0.00533 0.00533 0.00331
0.00424 0.00205 0.00347 0.00299 0.00485 0.00362 0.00534 0.00251 0.00205 0.00251 0.00016
0.00016
0.00000 0.00035 0.00000 0.00163 0.00163 0.01047 0.02430 0.06465 0.11407 0.14679 0.13266 0.13266
0.08370 0.06224 0.03625 0.03419 0.01842 0.01174 0.00969 0.00824 0.00720 0.00435 0.00977
0.00440 0.00942 0.00307 0.00368 0.00695 0.00259 0.00445 0.00517 0.00760 0.00355 0.00385
0.00324 0.00454 0.00610 0.00381 0.00346 0.00346 0.00173 0.00147 0.00177 0.00035 0.00000
0.00035
0.00000 0.00000 0.00000 0.00160 0.00173 0.01947 0.04927 0.08142 0.11116 0.13590 0.11284 0.09041
0.06495 0.04898 0.03647 0.03510 0.02652 0.02667 0.02791 0.02599 0.01726 0.01741 0.01074
0.01058 0.00537 0.00383 0.00331 0.00211 0.00238 0.00206 0.00112 0.00132 0.00214 0.00149
0.00362 0.00147 0.00303 0.00354 0.00309 0.00176 0.00202 0.00162 0.00093 0.00067 0.00027
0.00053
0.00000 0.00000 0.00075 0.00000 0.00299 0.02517 0.06334 0.12413 0.15487 0.15520 0.12487 0.07168
0.06749 0.03993 0.03528 0.01933 0.01203 0.00771 0.01078 0.01293 0.00628 0.00859 0.00252
0.00589 0.00861 0.00314 0.00139 0.00142 0.00155 0.00304 0.00284 0.00181 0.00217 0.00194
0.00284 0.00485 0.00310 0.00297 0.00217 0.00155 0.00129 0.00013 0.00052 0.00026 0.00000
0.00065
0.00000 0.00000 0.00100 0.00401 0.00201 0.01341 0.05894 0.11886 0.16807 0.13365 0.10744 0.06289
0.04950 0.02641 0.03576 0.02706 0.02508 0.02455 0.01516 0.02023 0.00905 0.00627 0.00599
0.00178 0.00364 0.00456 0.00493 0.00477 0.00477 0.00369 0.00343 0.00568 0.00589 0.00474
0.00592 0.00296 0.00752 0.00390 0.00589 0.00335 0.00249 0.00118 0.00215 0.00097 0.00024
0.00024
0.00000 0.00000 0.00000 0.00078 0.00160 0.00480 0.01658 0.07290 0.10598 0.13777 0.12225 0.08604
0.05424 0.04320 0.03390 0.02634 0.02263 0.02354 0.01756 0.02436 0.02306 0.02794 0.01935
0.00982 0.01561 0.01003 0.01230 0.00798 0.00557 0.01286 0.00570 0.00997 0.00899 0.00442
0.00884 0.00223 0.00434 0.00385 0.00192 0.00405 0.00000 0.00197 0.00065 0.00307 0.00000
0.00105
0.00000 0.00000 0.00000 0.00000 0.00172 0.00528 0.01096 0.03209 0.06254 0.08308 0.09345 0.08711
0.07940 0.06237 0.05870 0.05520 0.04503 0.04260 0.03210 0.03097 0.02250 0.02293 0.02559
0.01581 0.01311 0.01374 0.01391 0.01114 0.00868 0.00983 0.00673 0.00849 0.00762 0.00614
0.00439 0.00479 0.00662 0.00332 0.00211 0.00239 0.00476 0.00125 0.00063 0.00050 0.00040
0.00000
0.00020 0.00000 0.00060 0.00096 0.00315 0.00699 0.02512 0.05588 0.08662 0.11100 0.12524 0.10158
0.07897 0.06480 0.05513 0.04274 0.03397 0.02652 0.02503 0.02236 0.01532 0.01273 0.01264
0.01122 0.01005 0.00883 0.00772 0.00841 0.00572 0.00509 0.00544 0.00515 0.00681 0.00380
0.00477 0.00162 0.00281 0.00081 0.00093 0.00126 0.00060 0.00045 0.00058 0.00000 0.00000
0.00040
0.00041 0.00000 0.00041 0.00137 0.00289 0.00700 0.01927 0.03965 0.06500 0.09407 0.10466 0.10137
0.09153 0.07543 0.06599 0.05504 0.04363 0.04822 0.02798 0.02602 0.01979 0.01667 0.01225
0.01202 0.01165 0.00819 0.00790 0.00809 0.00539 0.00620 0.00555 0.00356 0.00316 0.00122
0.00228 0.00061 0.00115 0.00042 0.00175 0.00028 0.00063 0.00048 0.00000 0.00049 0.00015
0.00018
0.00000 0.00000 0.00116 0.00084 0.00269 0.00883 0.02259 0.04275 0.07779 0.11154 0.11293 0.10796
0.10880 0.07962 0.05692 0.04613 0.03696 0.02486 0.02243 0.01875 0.01792 0.01621 0.01218
0.00985 0.00796 0.00883 0.00708 0.00715 0.00467 0.00432 0.00492 0.00404 0.00266 0.00121
0.00115 0.00132 0.00071 0.00089 0.00083 0.00031 0.00103 0.00021 0.00074 0.00000 0.00000
0.00024
0.00000 0.00000 0.00014 0.00219 0.00348 0.01970 0.03823 0.06471 0.09491 0.11094 0.10259 0.10533
0.07502 0.06331 0.05572 0.04409 0.03971 0.02976 0.02357 0.02285 0.01914 0.01463 0.01037
0.01224 0.00729 0.01006 0.00399 0.00476 0.00327 0.00331 0.00310 0.00131 0.00158 0.00185
0.00095 0.00097 0.00135 0.00184 0.00086 0.00009 0.00024 0.00000 0.00000 0.00008 0.00031
0.00014
0.00017 0.00010 0.00053 0.00335 0.01211 0.02004 0.04969 0.08670 0.10839 0.12465 0.11440 0.09486
0.06848 0.05668 0.04700 0.04202 0.03128 0.02728 0.02133 0.01765 0.01017 0.00887 0.00759
0.00533 0.00627 0.00468 0.00423 0.00422 0.00245 0.00412 0.00214 0.00283 0.00241 0.00257
0.00225 0.00119 0.00034 0.00022 0.00057 0.00023 0.00000 0.00016 0.00018 0.00010 0.00000
0.00017
0.00028 0.00019 0.00296 0.00700 0.03373 0.06949 0.09678 0.09646 0.09452 0.09255 0.07775 0.06929
0.06406 0.05096 0.04650 0.03181 0.03167 0.02511 0.02023 0.01595 0.01120 0.00864 0.00827
0.00680 0.00467 0.00491 0.00329 0.00246 0.00342 0.00305 0.00222 0.00355 0.00184 0.00131
0.00093 0.00156 0.00069 0.00085 0.00104 0.00080 0.00048 0.00000 0.00000 0.00000 0.00000
0.00072

0.00000	0.00000	0.00501	0.00820	0.02156	0.04658	0.07397	0.10523	0.10564	0.11197	0.09664	0.08753
	0.06506	0.06547	0.04937	0.04050	0.02180	0.02132	0.01775	0.01321	0.00827	0.00757	0.00537
	0.00466	0.00364	0.00145	0.00127	0.00123	0.00118	0.00159	0.00074	0.00047	0.00066	0.00122
	0.00153	0.00013	0.00041	0.00031	0.00033	0.00035	0.00052	0.00000	0.00028	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00162	0.01360	0.04808	0.08554	0.10656	0.09929	0.10350	0.09404
	0.08365	0.07338	0.06095	0.05228	0.03758	0.03977	0.01817	0.01112	0.01451	0.01368	0.00943
	0.00618	0.00541	0.00398	0.00321	0.00228	0.00211	0.00184	0.00132	0.00185	0.00070	0.00101
	0.00063	0.00132	0.00059	0.00000	0.00041	0.00020	0.00000	0.00000	0.00021	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00237	0.01191	0.05245	0.10548	0.13326	0.11327	0.10033	0.09484
	0.06044	0.06029	0.05323	0.04390	0.02896	0.02585	0.01774	0.02099	0.01698	0.01305	0.00562
	0.00820	0.00595	0.00545	0.00482	0.00184	0.00171	0.00291	0.00227	0.00238	0.00147	0.00078
	0.00083	0.00000	0.00018	0.00000	0.00026	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00137	0.00109	0.00953	0.04531	0.03905	0.08620	0.12939
	0.10474	0.10815	0.07663	0.07979	0.05945	0.05108	0.04256	0.02772	0.02100	0.02527	0.01725
	0.01078	0.01050	0.00867	0.00560	0.00759	0.00601	0.00526	0.00413	0.00401	0.00355	0.00218
	0.00130	0.00167	0.00184	0.00133	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00026	0.00050	0.00082	0.00648	0.01535	0.02201	0.05178	0.10193
	0.12180	0.12062	0.10436	0.09600	0.06281	0.06630	0.03608	0.03340	0.02329	0.02395	0.02359
	0.01119	0.01553	0.01169	0.00832	0.00822	0.00734	0.00415	0.00735	0.00272	0.00222	0.00181
	0.00226	0.00219	0.00145	0.00000	0.00065	0.00033	0.00093	0.00000	0.00000	0.00000	0.00000
	0.00032										
0.00000	0.00007	0.00000	0.00000	0.00012	0.00012	0.00026	0.00377	0.00932	0.01999	0.05389	0.08170
	0.11789	0.11393	0.09204	0.09453	0.08725	0.06847	0.05296	0.04054	0.03442	0.02889	0.02360
	0.01323	0.01497	0.00994	0.00945	0.00390	0.00319	0.00281	0.00378	0.00291	0.00171	0.00270
	0.00230	0.00143	0.00101	0.00087	0.00084	0.00038	0.00020	0.00000	0.00000	0.00000	0.00000
	0.00063										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00073	0.00504	0.01388	0.04619	0.08172
	0.12245	0.11666	0.10002	0.09772	0.08091	0.07003	0.06459	0.03596	0.02991	0.02681	0.01735
	0.01411	0.01125	0.00916	0.01317	0.00744	0.00677	0.00581	0.00545	0.00419	0.00224	0.00237
	0.00187	0.00144	0.00099	0.00051	0.00043	0.00144	0.00047	0.00063	0.00000	0.00000	0.00000
	0.00031										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00027	0.00076	0.00097	0.00478	0.01512	0.03638	0.10597
	0.08970	0.11559	0.10516	0.08396	0.08947	0.07659	0.05141	0.04466	0.04277	0.02438	0.02640
	0.02046	0.01529	0.01098	0.00824	0.00562	0.00498	0.00265	0.00329	0.00259	0.00330	0.00206
	0.00248	0.00178	0.00064	0.00000	0.00014	0.00014	0.00027	0.00000	0.00000	0.00014	0.00000
	0.00065										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00072	0.00106	0.00302	0.01194	0.03270	0.07373
	0.08554	0.09548	0.11029	0.08263	0.09856	0.07582	0.06587	0.04623	0.04370	0.03711	0.02950
	0.02528	0.01429	0.01509	0.00874	0.01340	0.00621	0.00326	0.00484	0.00376	0.00350	0.00184
	0.00102	0.00111	0.00140	0.00049	0.00019	0.00040	0.00039	0.00082	0.00000	0.00000	0.00000
	0.00010										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00026	0.00051	0.00340	0.00522	0.01506	0.04088	0.07552
	0.09293	0.11652	0.11161	0.08811	0.08013	0.07080	0.06454	0.04626	0.04210	0.02923	0.03139
	0.02051	0.01387	0.00986	0.01047	0.00711	0.00570	0.00365	0.00214	0.00231	0.00273	0.00094
	0.00217	0.00110	0.00178	0.00072	0.00000	0.00026	0.00000	0.00023	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00030	0.00205	0.00729	0.02641
	0.05511	0.10262	0.10651	0.13452	0.10381	0.10558	0.08301	0.05686	0.04744	0.04299	0.02734
	0.03066	0.01670	0.01395	0.01058	0.00500	0.00740	0.00455	0.00196	0.00283	0.00130	0.00096
	0.00040	0.00078	0.00015	0.00030	0.00000	0.00000	0.00015	0.00000	0.00015	0.00000	0.00000
	0.00033										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00039	0.00000	0.00020	0.00116	0.00583	0.04150
	0.08557	0.09987	0.10202	0.09504	0.08592	0.08161	0.08228	0.06546	0.06254	0.04716	0.03723
	0.02415	0.01923	0.01435	0.00938	0.01130	0.00854	0.00547	0.00640	0.00311	0.00204	0.00082
	0.00074	0.00043	0.00000	0.00025	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00079	0.00419	0.00943	0.04282
	0.08267	0.11090	0.12113	0.11615	0.11176	0.08246	0.07049	0.05662	0.04759	0.03514	0.02633
	0.02382	0.01735	0.01143	0.00816	0.00769	0.00421	0.00298	0.00254	0.00135	0.00073	0.00090
	0.00000	0.00000	0.00000	0.00000	0.00036	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00009	0.00155	0.00253	0.00730	0.02082
	0.04813	0.08112	0.08972	0.11326	0.12070	0.12019	0.08367	0.06621	0.04784	0.05210	0.03437
	0.01925	0.01932	0.02383	0.00792	0.01272	0.00505	0.00524	0.00462	0.00230	0.00185	0.00222
	0.00324	0.00013	0.00130	0.00000	0.00130	0.00000	0.00013	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00013	0.00177	0.00197	0.01198	0.01997
	0.03977	0.06794	0.09636	0.09000	0.11329	0.10009	0.10383	0.06005	0.06147	0.04833	0.04104
	0.03340	0.02454	0.02297	0.01529	0.01302	0.01116	0.00502	0.00503	0.00414	0.00157	0.00105
	0.00062	0.00068	0.00000	0.00047	0.00120	0.00092	0.00095	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00023	0.00023	0.00000	0.00000	0.00039	0.00168	0.00811	0.01866
	0.03816	0.07696	0.10906	0.10379	0.11532	0.07637	0.07491	0.05231	0.05399	0.05503	0.04591
	0.02684	0.03722	0.03122	0.02057	0.01602	0.01226	0.01196	0.00356	0.00446	0.00299	0.00077
	0.00000	0.00015	0.00038	0.00036	0.00000	0.00015	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										

0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00013 0.00288 0.00815
0.02315 0.03798 0.09439 0.09942 0.14326 0.09784 0.08003 0.07375 0.05726 0.04862 0.04378
0.03225 0.03637 0.03612 0.01500 0.01540 0.02593 0.01232 0.00682 0.00847 0.00061 0.00000
0.00009 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00024 0.00065 0.00327
0.01373 0.04329 0.08466 0.10017 0.13411 0.11369 0.09509 0.05914 0.05036 0.04211 0.03909
0.03223 0.03775 0.03512 0.02995 0.01806 0.01706 0.01093 0.01078 0.01145 0.00330 0.00535
0.00254 0.00126 0.00187 0.00069 0.00000 0.00137 0.00000 0.00069 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00010 0.00001 0.00011 0.00001 0.00002 0.00029
0.00090 0.00438 0.01868 0.04818 0.09743 0.12725 0.10395 0.08993 0.07930 0.08172 0.07124
0.04977 0.06034 0.03293 0.02726 0.03080 0.02062 0.01592 0.01014 0.01118 0.00797 0.00406
0.00225 0.00270 0.00058 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
#Number and vector of years of age compositions (headboat)
23
1975 1980 1981 1982 1983 1986 1987 1988 1991 1992 1993 1994
1995 1996 1997 1998 2001 2002 2003 2004 2005 2006 2007
#sample sizes of age comp data by year
1 12 112 38 2 89 8 2 166 46 48 252
192 73 14 2 22 10 103 331 486 597 721
#headboat age comp samples (year,age)
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.37841 0.33207 0.26242 0.00000 0.02711 0.00000 0.00000 0.00000 0.00000 0.00000
0.07788 0.26843 0.24144 0.27936 0.07902 0.05388 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.07967 0.37061 0.11363 0.32502 0.09459 0.01648 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.75935 0.00000 0.24065 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.13665 0.43246 0.31863 0.07700 0.01634 0.01339 0.00442 0.00058 0.00000 0.00000 0.00035 0.00018
0.38879 0.13853 0.47268 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00450 0.05666 0.38623 0.41871 0.08466 0.03886 0.00809 0.00199 0.00024 0.00007 0.00000 0.00000
0.00000 0.00000 0.25651 0.44385 0.12083 0.13650 0.04012 0.00219 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.33592 0.29281 0.10595 0.25224 0.01146 0.00070 0.00000 0.00062 0.00000 0.00030
0.00000 0.01441 0.22340 0.44935 0.17976 0.10257 0.02268 0.00623 0.00109 0.00040 0.00003 0.00007
0.00000 0.03094 0.33612 0.28221 0.23949 0.08306 0.01921 0.00772 0.00067 0.00042 0.00016 0.00000
0.00000 0.07431 0.11710 0.20214 0.34395 0.17822 0.08044 0.00249 0.00097 0.00038 0.00000 0.00000
0.00000 0.16150 0.09328 0.13470 0.28033 0.09328 0.23691 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.50000 0.00000 0.00000 0.50000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.17329 0.64460 0.18211 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.48621 0.25759 0.24646 0.00974 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.05964 0.46588 0.34998 0.07205 0.04261 0.00751 0.00221 0.00013 0.00000 0.00000 0.00000 0.00000
0.00127 0.45704 0.39357 0.08826 0.03478 0.02146 0.00214 0.00148 0.00000 0.00000 0.00000 0.00000
0.01966 0.20797 0.32992 0.27397 0.08546 0.02139 0.02476 0.02391 0.00590 0.00705 0.00000 0.00000
0.01140 0.11040 0.31305 0.35489 0.14892 0.02910 0.01283 0.01121 0.00245 0.00576 0.00000 0.00000
0.00039 0.18156 0.33197 0.22119 0.13316 0.07363 0.03393 0.01427 0.00316 0.00666 0.00008 0.00000
#Starting and ending year of headboat discard length composition data
2003
2007
#sample sizes of length comp data by year
23 176 652 514 853
#HB discard length comp samples (year,lengthbin)
0.00000 0.00000 0.04348 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.04348 0.04348 0.21739
0.21739 0.21739 0.13043 0.04348 0.04348 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00571 0.00000 0.00000 0.00000 0.00000 0.01714 0.01143 0.00571 0.04571 0.06286 0.15429 0.14286
0.14286 0.11429 0.08000 0.08000 0.04000 0.03429 0.02857 0.02286 0.00571 0.00000 0.00571
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00448 0.00127 0.00321 0.00193 0.00127 0.00255 0.01344 0.03268 0.06027 0.12111 0.17624 0.20502
0.19746 0.09919 0.03194 0.01660 0.01344 0.00448 0.00448 0.00637 0.00000 0.00000 0.00000
0.00127 0.00000 0.00000 0.00000 0.00127 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00485 0.00567 0.01458 0.01538 0.02188 0.02755 0.05512 0.09561 0.12237 0.16292
0.18480 0.13047 0.11020 0.03242 0.01214 0.00243 0.00000 0.00000 0.00162 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00133 0.00000 0.00200 0.00642 0.01791 0.00708 0.00333 0.01575 0.02416 0.03483 0.03907 0.09123
0.12373 0.16939 0.21488 0.13832 0.05574 0.01083 0.02300 0.01592 0.00000 0.00000 0.00000
0.00000 0.00000 0.00508 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000

####################################MRFSS landings ###################################
#Recreational MRFSS CPUE Index

#Starting and ending years of CPUE index
1987
2007
#Observed CPUE and assumed CVs
1.165 0.801 0.946 1.790 1.446 0.916 0.896 0.621 0.798 1.151 0.654 0.868
1.096 1.093 1.031 0.873 1.049 1.128 0.918 0.920 0.840
0.300 0.194 0.163 0.252 0.204 0.161 0.140 0.109 0.196 0.195 0.150 0.121
0.093 0.101 0.092 0.098 0.104 0.094 0.101 0.153 0.080
#Recreational landings
##Starting and ending years for landings time series
1947
2007
#MRFSS landings vector (1000 fish)
9.013 17.077 25.099 33.059 40.935 48.707 56.352 63.851 71.182 78.325 85.257 91.959
98.409 104.587 110.447 115.852 120.640 124.651 127.722 129.749 130.858 131.230 131.049
130.497 129.728 128.788 127.693 126.461 125.109 123.653 122.112 120.503 118.841 117.146
115.008 230.532 304.266 366.589 420.894 307.370 202.196 179.117 202.690 190.929 164.798
136.442 116.230 85.881 81.076 93.301 104.549 120.203 165.514 209.669 212.669 191.314
204.084 212.034 180.800 180.756 213.288
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05
#Starting and ending years of discards time series, respectively
1947
2007
#Observed discards (1000s) and assumed CVs
1.953 3.701 5.439 7.165 8.871 10.556 12.213 13.838 15.426 16.974 18.477 19.929
21.327 22.666 23.936 25.107 26.145 27.014 27.680 28.119 28.359 28.440 28.401
28.281 28.114 27.911 27.673 27.406 27.113 26.798 26.464 26.115 25.755 25.388
33.350 41.312 31.903 22.494 24.091 24.091 24.091 25.687 63.855 71.476 42.392
79.547 48.160 66.768 121.089 41.777 41.445 59.409 257.553 215.610 137.247 108.259
183.324 150.603 87.678 83.166 262.351
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05
#Starting and ending year of mrfss length composition data
1981
2007
#sample sizes of length comp data by year
10 90 124 581 165 19 36 32 80 66 50 114
75 77 74 16 68 76 194 261 398 393 578
888 1835 1209 1274
#MRFSS length comp samples (year,lengthbin)
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.05214 0.05214 0.18946
0.09473 0.00000 0.00000 0.24478 0.18338 0.18338 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.01380 0.00000 0.00000 0.03289 0.01779 0.04851
0.12390 0.12420 0.00000 0.08737 0.12420 0.05520 0.08280 0.05919 0.07828 0.04539 0.02890
0.00130 0.00000 0.02308 0.01197 0.00928 0.00000 0.01197 0.00399 0.01197 0.00399 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00210 0.00068 0.00000 0.00105 0.00678
0.02518 0.00555 0.08226 0.06562 0.06346 0.06125 0.01121 0.15202 0.04831 0.00413 0.04095
0.04696 0.00000 0.00000 0.04696 0.09392 0.04027 0.00000 0.04027 0.08054 0.00000 0.00000
0.00000 0.00000 0.00000 0.04027 0.00000 0.04027 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00865 0.00128 0.00000 0.00744 0.01926 0.02006 0.01766 0.03940
0.06407 0.07931 0.01187 0.10542 0.10756 0.07978 0.10317 0.04850 0.05667 0.03346 0.04835
0.03005 0.00365 0.01438 0.02366 0.00736 0.00756 0.00795 0.01512 0.00740 0.00736 0.01449
0.00000 0.00000 0.00089 0.00256 0.00000 0.00128 0.00439 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00077 0.02031
0.05456 0.01597 0.01698 0.52882 0.19838 0.03763 0.05644 0.01645 0.02192 0.01485 0.00623
0.00259 0.00000 0.00292 0.00000 0.00518 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.05523 0.16569 0.11046 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.05523 0.11046
0.07183 0.12706 0.00000 0.05523 0.05523 0.05523 0.05523 0.00000 0.08313 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000

0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00670	0.00603	0.00000
	0.02215	0.10106	0.02246	0.06591	0.07487	0.00000	0.00603	0.02760	0.18046	0.00000	0.01635
	0.00000	0.00584	0.13766	0.00000	0.00000	0.10852	0.00000	0.19774	0.00000	0.00000	0.01031
	0.00000	0.00000	0.00000	0.01031	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.01114	0.01114	0.00000	0.00000
	0.00000	0.11202	0.00963	0.00000	0.00963	0.00963	0.00963	0.02627	0.01926	0.00000	0.00963
	0.53236	0.00000	0.12408	0.01926	0.00963	0.00963	0.00963	0.01926	0.00963	0.00963	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00963	0.00000	0.00000	0.01926	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00316	0.00316	0.07569	0.03646	0.07292
	0.03404	0.04438	0.07279	0.10286	0.06336	0.03071	0.06833	0.07498	0.08192	0.06223	0.02391
	0.00000	0.00000	0.01196	0.07569	0.00000	0.00000	0.00000	0.01169	0.01169	0.00000	0.00000
	0.00000	0.00945	0.00000	0.00000	0.02863	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.01150	0.00000	0.01398
	0.05269	0.07789	0.03660	0.02300	0.02547	0.00119	0.00000	0.01150	0.05095	0.37612	0.05591
	0.05591	0.00000	0.04904	0.01635	0.04419	0.01150	0.00000	0.00000	0.05828	0.00000	0.00000
	0.00000	0.00000	0.00000	0.01398	0.01398	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00021	0.04626	0.00021	0.18863
	0.28690	0.16353	0.04626	0.09469	0.00000	0.09442	0.02800	0.00379	0.00358	0.00021	0.00000
	0.00379	0.01263	0.01197	0.00021	0.00041	0.00000	0.00000	0.00000	0.00358	0.00716	0.00000
	0.00358	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00606	0.03967	0.01511	0.01048
	0.06701	0.05372	0.02462	0.20253	0.09615	0.12591	0.00993	0.02842	0.06966	0.07467	0.02561
	0.01513	0.01557	0.02747	0.00755	0.01048	0.01557	0.02963	0.00746	0.00302	0.00000	0.00603
	0.00000	0.01255	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.02488	0.00982	0.00000
	0.00000	0.00000	0.02761	0.10205	0.04633	0.12734	0.07729	0.01662	0.03540	0.18468	0.11429
	0.15579	0.00000	0.00306	0.00611	0.00000	0.00000	0.02607	0.00491	0.01289	0.00000	0.02488
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.01019	0.06024	0.02020	0.04526	0.04329	0.03695	0.01057	0.04141	0.06638	0.08189	0.10991
	0.06728	0.06672	0.04554	0.00826	0.04380	0.05020	0.00074	0.03174	0.10027	0.02497	0.00000
	0.00000	0.00000	0.02497	0.00000	0.00230	0.00230	0.00461	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.02547	0.00000	0.00796	0.07947
	0.09689	0.10119	0.01501	0.12263	0.25379	0.05326	0.07940	0.05402	0.02766	0.03933	0.00000
	0.00000	0.00000	0.00229	0.00000	0.00457	0.00457	0.00457	0.00000	0.00000	0.00000	0.00000
	0.00000	0.02792	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.06279	0.23172	0.14040	0.06279	0.00000	0.06279	0.00000	0.00000	0.06279
	0.00000	0.00000	0.06279	0.00000	0.18836	0.00000	0.06279	0.00000	0.00000	0.00000	0.06279
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00322
	0.02795	0.00000	0.00133	0.12195	0.24157	0.20275	0.10913	0.02382	0.05351	0.05866	0.01519
	0.06439	0.00760	0.01652	0.00000	0.00322	0.00000	0.00000	0.04920	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00539	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00539	0.00435
	0.00000	0.01464	0.00000	0.04702	0.00932	0.09059	0.00828	0.03967	0.04765	0.05966	0.04869
	0.10760	0.06006	0.03363	0.00000	0.09476	0.10327	0.07581	0.04576	0.00900	0.00900	0.04501
	0.00900	0.00000	0.00000	0.02642	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00277
	0.01227	0.03460	0.03456	0.12813	0.09178	0.16546	0.10808	0.05378	0.06198	0.03614	0.05036
	0.02594	0.03206	0.03665	0.03924	0.01742	0.00077	0.03420	0.00692	0.00000	0.00000	0.00000
	0.00000	0.00352	0.00000	0.00113	0.00113	0.00704	0.00000	0.00000	0.00000	0.00000	0.00000
	0.01407										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.03198	0.05317	0.12238	0.03998	0.05301	0.04024	0.01367	0.01992	0.01832	0.02024
	0.00549	0.02547	0.01956	0.01315	0.01908	0.01316	0.00659	0.00125	0.00136	0.47211	0.00339
	0.00056	0.00000	0.00000	0.00174	0.00000	0.00174	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00243										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00142
	0.00484	0.02102	0.03623	0.04447	0.03650	0.06944	0.09933	0.06185	0.06299	0.07729	0.04529
	0.16628	0.06097	0.02357	0.01738	0.01160	0.00932	0.03152	0.00494	0.00552	0.00259	0.00259
	0.00432	0.00000	0.00109	0.00109	0.00000	0.09439	0.00109	0.00109	0.00000	0.00000	0.00000
	0.00000										
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00431	0.01292
	0.01141	0.02768	0.01251	0.06043	0.07598	0.09279	0.05203	0.06662	0.03116	0.10960	0.04136
	0.02981	0.02666	0.03275	0.02924	0.01371	0.21642	0.02182	0.00793	0.01518	0.00000	0.00592
	0.00000	0.00000	0.00177	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000										

0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00566 0.05520 0.04555 0.05960 0.08079 0.09794 0.08669 0.08695 0.05771 0.05273
0.06055 0.05472 0.03708 0.07171 0.03798 0.01497 0.02425 0.01216 0.00440 0.00438 0.00289
0.00561 0.00000 0.00340 0.00000 0.00000 0.03709 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00013 0.00864 0.00473
0.00339 0.00565 0.01872 0.04039 0.10763 0.08180 0.04978 0.06053 0.07006 0.05925 0.05456
0.09804 0.06460 0.05858 0.04248 0.05131 0.02780 0.02490 0.02288 0.01453 0.00277 0.01330
0.00712 0.00578 0.00064 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00053 0.00280 0.00584
0.00445 0.02732 0.02492 0.02471 0.04995 0.05530 0.06720 0.09560 0.07468 0.08701 0.07145
0.06576 0.05730 0.06264 0.03958 0.05278 0.03469 0.02385 0.02638 0.01082 0.00738 0.00554
0.00217 0.00164 0.01093 0.00000 0.00605 0.00000 0.00022 0.00051 0.00000 0.00000 0.00000
0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00031 0.00000
0.00857 0.00159 0.01099 0.05646 0.05375 0.07353 0.08523 0.12427 0.07134 0.05020 0.09169
0.05205 0.09568 0.05155 0.05316 0.02307 0.03890 0.01671 0.00714 0.00421 0.00487 0.00220
0.00004 0.00334 0.00087 0.00004 0.00000 0.00000 0.00015 0.00362 0.00483 0.00603 0.00241
0.00121
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00012 0.00000 0.00031
0.00633 0.01457 0.02375 0.01358 0.04638 0.04569 0.10339 0.09689 0.11311 0.14584 0.07397
0.04815 0.05412 0.03974 0.03992 0.03331 0.02928 0.02550 0.01150 0.01082 0.01662 0.00436
0.00250 0.00023 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000
##Starting and ending year of mrfss age composition data
2001
2007
#sample sizes of mrfss age comp data by year
83 217 366 102 299 230 31
#mrfss age comps (year,lengthbin)
0.00000 0.20580 0.42328 0.24969 0.05732 0.04999 0.00000 0.00000 0.01392 0.00000 0.00000 0.00000
0.00111 0.12222 0.45630 0.27741 0.06241 0.04139 0.02718 0.00546 0.00653 0.00000 0.00000 0.00000
0.00260 0.09799 0.45269 0.21794 0.15407 0.03633 0.02629 0.00747 0.00000 0.00000 0.00461 0.00000
0.00000 0.14556 0.56627 0.16492 0.04777 0.03051 0.01845 0.00000 0.02653 0.00000 0.00000 0.00000
0.00000 0.07038 0.28799 0.32470 0.20651 0.07206 0.02285 0.01254 0.00000 0.00297 0.00000 0.00000
0.00000 0.00560 0.15576 0.42517 0.26112 0.08868 0.03615 0.00996 0.01216 0.00541 0.00000 0.00000
0.00000 0.00000 0.00000 0.24605 0.57481 0.09352 0.06571 0.01992 0.00000 0.00000 0.00000 0.00000
################Likelihood Component
Weighting###
#Weights in objective fcn
1000.0 #landings
1000.0 #discards
0.01 #length comps
1.0 #age comps
100.0 #FST index
100.0 #CVT index
100.0 #HAL index
100.0 #HB index
100.0 #MRFSS index
1.0 #S-R residuals
0.0 #penalty if F exceeds 3.0
0.0 #constrain first several years of recruitment variability
1000.0 #additional constraint on variability of recruitment in last BiasCor_exclude_yrs yrs
0.0 #constrain variability of F in last three years
0.0 #constraint B1/B0
0.0 #penalty on deviation in CV at age
0.0 #penalty on first difference in CV at age

################Parameter values and initial
guesses###
#bias adjustment (multiplier) for rec landings (HB and MRFSS) from Salt-Water Angling report
1.0
#annual rate of increase on all fishery dependent q's
0.02
#log catchabilities (initial guesses)
-15.0 #Marmap FST
-15.0 #Marmap CVT
-8.0 #commHAL (index in weight)
-15.0 #HB
-15.0 #MRFSS
#log mean F's (initial guesses)
-3.0 #commHAL
-4.0 #commHTR
-4.0 #commCMB
-3.0 #HB
-3.0 #MRFSS
#log mean F's for discards (initial guesses)
-6.0 #commHAL discards
-6.0 #HB discards

```
-6.0                    #MRFSS discards

##Selectivity parameters.
##Initial guess must be within boundaries.
#  init_bounded_number selpar_L50_commHAL1(0.1,8.0,1);
#  init_bounded_number selpar_slope_commHAL1(0.5,12.0,1); //period 1
#  init_bounded_number selpar_L502_commHAL1(1.0,6.0,3);  additive to L50
#  init_bounded_number selpar_slope2_commHAL1(0.0,12.0,3); //period 1

# age at size limits (10,11,12 inches)= 2.3, 3.2, 4.2
# zero in slope2 provides logistic selectivity

#values below fixed to provide selectivity of only age one fish
0.01 #selpar_L50_FST
10.0 #selpar_slope_FST
1.0 #selpar_L502_FST
30.0 #selpar_slope2_FST

2.0 #selpar_L50_CVT
12.0 #selpar_slope_CVT (fixed)
1.0 #selpar_L502_CVT
1.0 #selpar_slope2_CVT

2.0 #selpar_L50_commHAL1
6.0 #selpar_slope_commHAL1
4.0 #selpar_L502_commHAL1
0.0 #selpar_slope2_commHAL1
4.2 #selpar_L50_commHAL2
12.0 #selpar_slope_commHAL2 (fixed)
4.0 #selpar_L502_commHAL2
0.0 #selpar_slope2_commHAL2

#ages 1,2 are estimated, 3 is fixed at one, others are from probability of undersized give 12-inch limit
#0.937  0.785  0.606  0.455  0.343  0.264  0.207  0.168  0.139  0.117  0.101  0.089
0.2 0.2  1.0  0.455  0.343  0.264  0.207  0.168  0.139  0.117  0.101  0.089
0.0 #selpar_Age1_commHAL_D2
0.01 #selpar_Age2_commHAL_D2
#0.5 #selpar_L50_commHAL_D2
#2.0 #selpar_slope_commHAL_D2
#4.2 #selpar_L502_commHAL_D2
#3.0 #selpar_slope2_commHAL_D2

#values below fixed to provide selectivity of only age one fish
0.01 #selpar_L50_commCMB1
10.0 #selpar_slope_commCMB1
1.0 #selpar_L502_commCMB1
30.0 #selpar_slope2_commCMB1
#4.2 #selpar_L50_commCMB2
#2.0 #selpar_slope_commCMB2
#4.0 #selpar_L502_commCMB2
#2.0 #selpar_slope2_commCMB2

2.0 #selpar_L50_HB1
5.0 #selpar_slope_HB1
4.0 #selpar_L502_HB1
0.0 #selpar_slope2_HB1
2.3 #selpar_L50_HB2
12.0 #selpar_slope_HB2 (fixed)
4.0 #selpar_L502_HB2
0.0 #selpar_slope2_HB2
3.2 #selpar_L50_HB3
12.0 #selpar_slope_HB3 (fixed)
4.0 #selpar_L502_HB3
0.0 #selpar_slope2_HB3
4.2 #selpar_L50_HB4
12.0 #selpar_slope_HB4 (fixed)
4.0 #selpar_L502_HB4
0.0 #selpar_slope2_HB4

#ages 1,2 are estimated, 3 is fixed at one, others are from probability of undersized give 10-inch limit
#0.695 0.459 0.296 0.197 0.136 0.099 0.075 0.058 0.047 0.039 0.034 0.029
0.2 0.2  1.0  0.197 0.136 0.099 0.075 0.058 0.047 0.039 0.034 0.029
#ages 1,2 are estimated, 3 is fixed at one, others are from probability of undersized give 11-inch limit
#0.845 0.634 0.447 0.315 0.227 0.169 0.130 0.103 0.084 0.071 0.060 0.053
0.2 0.2  1.0  0.315 0.227 0.169 0.130 0.103 0.084 0.071 0.060 0.053
0.0 #selpar_Age1_HB_D3
0.01 #selpar_Age2_HB_D3
#ages 1,2 are estimated, 3 is fixed at one, others are from probability of undersized give 12-inch limit
#0.937  0.785  0.606  0.455  0.343  0.264  0.207  0.168  0.139  0.117  0.101  0.089
0.2 0.2  1.0  0.455  0.343  0.264  0.207  0.168  0.139  0.117  0.101  0.089
```

0.0 #selpar_Age1_HB_D4
0.01 #selpar_Age2_HB_D4

#0.5 #selpar_L50_HB_D3
#2.0 #selpar_slope_HB_D3
#2.2 #selpar_L502_HB_D3
#3.0 #selpar_slope2_HB_D3
#0.5 #selpar_L50_HB_D4
#2.0 #selpar_slope_HB_D4
#3.2 #selpar_L502_HB_D4
#3.0 #selpar_slope2_HB_D4

3.2 #selpar_L50_MRFSS3
12.0 #selpar_slope_MRFSS3 (fixed)
4.0 #selpar_L502_MRFSS3
0.0 #selpar_slope2_MRFSS3
4.2 #selpar_L50_MRFSS4
2.0 #selpar_slope_MRFSS4
4.0 #selpar_L502_MRFSS4
0.0 #selpar_slope2_MRFSS4

#threshold sample sizes for length comps
9999.0 #FLT This value is set above all observed sample sizes, bc selectivity of this fishery is fixed
#170.0 #CVT
400.0 #HAL
170.0 #HAL discards
9999.0 #TWL/CMB This value is set above all observed sample sizes, bc selectivity of this fishery is fixed
400.0 #HB
170.0 #HB discards
400.0 #MRFSS

#threshold sample sizes for age comps
45.0 #CVT
45.0 #HAL
45.0 #HB
45.0 #MRFSS

#Ageing error matrix (columns are true age 1-12, rows are ages as read for age comps)
0.67108374 0.23147254 0.02049724 0.00037312 0.00000137 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.30118969 0.51574628 0.22672800 0.02048959 0.00037312 0.00000137 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.02722891 0.23147254 0.50517490 0.22664340 0.02048956 0.00037312 0.00000137 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00049585 0.02092616 0.22672800 0.50498641 0.22664309 0.02048956 0.00037312 0.00000137 0.00000000 0.00000000 0.00000000 0.00000000
0.00000182 0.00038107 0.02049724 0.22664340 0.50498572 0.22664309 0.02048956 0.00037312 0.00000137 0.00000000 0.00000000 0.00000000
0.00000000 0.00000140 0.00037326 0.02048959 0.22664309 0.50498572 0.22664309 0.02048956 0.00037312 0.00000137 0.00000000 0.00000000
0.00000000 0.00000000 0.00000137 0.00037312 0.02048956 0.22664309 0.50498572 0.22664309 0.02048950 0.00037298 0.00000136 0.00000000
0.00000000 0.00000000 0.00000000 0.00000137 0.00037312 0.02048956 0.22664309 0.50498572 0.22664240 0.02048200 0.00037049 0.00000135
0.00000000 0.00000000 0.00000000 0.00000000 0.00000137 0.00037312 0.02048956 0.22664309 0.50498417 0.22655949 0.02034485 0.00036729
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000137 0.00037312 0.02048956 0.22664240 0.50479945 0.22504241 0.02016915
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000137 0.00037312 0.02048950 0.22655949 0.50141923 0.22309888
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000137 0.00037755 0.02122521 0.25282166 0.75636334
#1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
#0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
#0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
#0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
#0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
#0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
#0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
#0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
#0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
#0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
#0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
#0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0

0.0 #p_lenc_cHAL proportion of length comp mass below size limit considered when matching length comp
0.0 #p_lenc.cCMB proportion of length comp mass below size limit considered when matching length comp
0.0 #p_lenc_HB2 proportion of length comp mass below size limit considered when matching length comp
0.0 #p_lenc_HB3 proportion of length comp mass below size limit considered when matching length comp
0.0 #p_lenc_HB4 proportion of length comp mass below size limit considered when matching length comp
0.0 #p_lenc_MRFSS2 proportion of length comp mass below size limit considered when matching length comp
0.0 #p_lenc_MRFSS3 proportion of length comp mass below size limit considered when matching length comp
0.0 #p_lenc_MRFSS4 proportion of length comp mass below size limit considered when matching length comp

0.0 #p_lenc_cHAL_D proportion of length comp mass above size limit considered when matching length comp of discards
0.0 #p_lenc_HB_D2 proportion of length comp mass above size limit considered when matching length comp of discards
0.0 #p_lenc_HB_D3 proportion of length comp mass above size limit considered when matching length comp of discards
0.0 #p_lenc_HB_D4 proportion of length comp mass above size limit considered when matching length comp of discards

999 #end of data file flag

III. File to create R object (vs_make_Robject4.cxx).

```
// Create a file with an R object from AD Model Builder

// open the file using the default AD Model Builder file name, and
// 6 digits of precision
open_r_file(adprogram_name + ".rdat", 6);

// Example of an INFO object
open_r_info_list("info", true);
                wrt_r_item("title", "SEDAR 17 Benchmark Assessment");
                wrt_r_item("species", "Vermilion Snapper");
                wrt_r_item("model", "Statistical Catch at Age");
                wrt_r_item("rec.model", "BH-steep");
                wrt_r_item("base.run", "vs00X.tpl");
                wrt_r_item("units.lengthTL", "mm");
                wrt_r_item("units.lengthFL", "mm");
                wrt_r_item("units.weight", "lb");
                wrt_r_item("units.biomass", "metric tons");
                wrt_r_item("units.ssb", "10^12 eggs");
                wrt_r_item("units.ypr", "lb");
//              wrt_r_item("units.landings", "1000 lb");
                wrt_r_item("units.discards", "1000 dead fish");
   wrt_r_item("units.numbers", "number fish");
   wrt_r_item("units.naa", "number fish");
                wrt_r_item("units.rec", "number fish");
close_r_info_list();


// VECTOR object of parameters and estimated quantities
open_r_info_list("parms", false);
                wrt_r_item("styr", styr);
                wrt_r_item("endyr", endyr);
                wrt_r_item("styrR", styr_rec_dev);
                wrt_r_item("M.msst", M_constant);
                wrt_r_item("Linf", Linf);
                wrt_r_item("K", K);
                wrt_r_item("t0", t0);
   wrt_r_item("TL2FL_a",lenpar_TL2FL_a);
   wrt_r_item("TL2FL_b",lenpar_TL2FL_b);
                wrt_r_item("wgt.a", wgtpar_a);
                wrt_r_item("wgt.b", wgtpar_b);
                wrt_r_item("FL2fec.a", fecpar_a);
                wrt_r_item("FL2fec.b", fecpar_b);
   wrt_r_item("batches", fecpar_batches);
   wrt_r_item("spawn.time", spawn_time_frac);
                wrt_r_item("D.mort.c.hal", Dmort_cHAL);
                wrt_r_item("D.mort.hb", Dmort_HB);
                wrt_r_item("D.mort.rec", Dmort_MRFSS);
                wrt_r_item("q.fst", mfexp(log_q_FST));
   wrt_r_item("q.cvt", mfexp(log_q_CVT));
                wrt_r_item("q.hal", mfexp(log_q_HAL));
                wrt_r_item("q.hb", mfexp(log_q_HB));
                wrt_r_item("q.rec", mfexp(log_q_MRFSS));
                wrt_r_item("q.rate",q_rate);
                wrt_r_item("F.prop.c.hal", F_cHAL_prop);
                wrt_r_item("F.prop.c.cmb", F_cCMB_prop);
                wrt_r_item("F.prop.hb", F_HB_prop);
                wrt_r_item("F.prop.rec", F_MRFSS_prop);
                wrt_r_item("F.prop.c.hal.D", F_cHAL_D_prop);
                wrt_r_item("F.prop.hb.D", F_HB_D_prop);
                wrt_r_item("F.prop.rec.D", F_MRFSS_D_prop);
                wrt_r_item("L.bias", L_early_bias);
                wrt_r_item("B0", B0);
                wrt_r_item("Bstyr.B0", totB(styr)/B0);
                wrt_r_item("SSB0", S0);
                wrt_r_item("SSBstyr.SSB0", SSB(styr)/S0);
                wrt_r_item("Rstyr.R0", rec(styr)/R0);
                wrt_r_item("BH.biascorr",BiasCor);
                wrt_r_item("BH.Phi0", spr_F0);
                wrt_r_item("BH.R0", R0);
                wrt_r_item("BH.steep", steep);
                wrt_r_item("R.autocorr",R_autocorr);
                wrt_r_item("R0", R0); //same as BH.R0, but used in BSR.time.plots
                wrt_r_item("R.virgin.bc", R_virgin); //bias-corrected virgin recruitment
                wrt_r_item("rec.lag", 1.0);
                wrt_r_item("msy", msy_out);
                wrt_r_item("Fmsy", F_msy_out);
                wrt_r_item("SSBmsy", SSB_msy_out);
```

```
                wrt_r_item("msst", (1.0-M_constant)*SSB_msy_out);
                wrt_r_item("Bmsy", B_msy_out);
                wrt_r_item("Rmsy", R_msy_out);
                wrt_r_item("sprmsy",spr_msy_out);
                wrt_r_item("Dmsy", D_msy_out);
                wrt_r_item("Fend.Fmsy", fullF(endyr)/F_msy_out);
                wrt_r_item("SSBend.SSBmsy", SSB(endyr)/SSB_msy_out);
close_r_info_list();

// VECTOR object of likelihood contributions
open_r_vector("like");
   wrt_r_item("lk.unwgt.data", fval_unwgt);
                wrt_r_item("lk.total", fval);
   wrt_r_item("lk.U.fst", f_FST_cpue*w_I_FST);
   wrt_r_item("lk.U.cvt", f_CVT_cpue*w_I_CVT);
   wrt_r_item("lk.U.hal", f_HAL_cpue*w_I_HAL);
   wrt_r_item("lk.U.hb", f_HB_cpue*w_I_HB);
   wrt_r_item("lk.U.rec", f_MRFSS_cpue*w_I_MRFSS);
   wrt_r_item("lk.L.c.htr", f_cHTR_L*w_L);
   wrt_r_item("lk.L.c.hal", f_cHAL_L*w_L);
   wrt_r_item("lk.L.c.cmb", f_cCMB_L*w_L);
   wrt_r_item("lk.L.hb", f_HB_L*w_L);
   wrt_r_item("lk.L.rec", f_MRFSS_L*w_L);
   wrt_r_item("lk.D.c.hal", f_cHAL_D*w_D);
   wrt_r_item("lk.D.hb", f_HB_D*w_D);
   wrt_r_item("lk.D.rec", f_MRFSS_D*w_D);
   wrt_r_item("lk.lenc.fst", f_FST_lenc*w_lc);
//    wrt_r_item("lk.lenc.c.cvt", f_CVT_lenc*w_lc);
   wrt_r_item("lk.lenc.c.hal", f_cHAL_lenc*w_lc);
   wrt_r_item("lk.lenc.c.cmb", f_cCMB_lenc*w_lc);
   wrt_r_item("lk.lenc.hb", f_HB_lenc*w_lc);
   wrt_r_item("lk.lenc.rec", f_MRFSS_lenc*w_lc);
   wrt_r_item("lk.lenc.c.hal.D", f_cHAL_D_lenc*w_lc);
   wrt_r_item("lk.lenc.hb.D", f_HB_D_lenc*w_lc);
   wrt_r_item("lk.agec.cvt", f_CVT_agec*w_ac);
   wrt_r_item("lk.agec.c.hal", f_cHAL_agec*w_ac);
   wrt_r_item("lk.agec.hb", f_HB_agec*w_ac);
   wrt_r_item("lk.agec.rec", f_MRFSS_agec*w_ac);

   wrt_r_item("lk.SRfit", f_N_dev*w_R);
   wrt_r_item("lk.fullF", f_fullF_constraint*w_fullF);
   wrt_r_item("lk.SRinit", f_N_dev_early*w_R_init);
   wrt_r_item("lk.SRend", f_N_dev_end*w_R_end);
   wrt_r_item("lk.Fend", f_Fend_constraint*w_F);
   wrt_r_item("lk.B1dB0", f_B1dB0_constraint*w_B1dB0);
   wrt_r_item("lk.cvlen.dev", f_cvlen_dev_constraint*w_cvlen_dev);
   wrt_r_item("lk.cvlen.diff", f_cvlen_diff_constraint*w_cvlen_diff);

                wrt_r_item("w.L", w_L);
                wrt_r_item("w.D", w_D);
                wrt_r_item("w.lenc", w_lc);
                wrt_r_item("w.agec", w_ac);
                wrt_r_item("w.U.fst", w_I_FST);
                wrt_r_item("w.U.cvt", w_I_CVT);
                wrt_r_item("w.U.hal", w_I_HAL);
                wrt_r_item("w.U.hb", w_I_HB);
                wrt_r_item("w.U.rec", w_I_MRFSS);
                wrt_r_item("w.R", w_R);
                wrt_r_item("w.R.init", w_R_init);
                wrt_r_item("w.R.end", w_R_end);
                wrt_r_item("w.F", w_F);
                wrt_r_item("w.B1dB0", w_B1dB0);
                wrt_r_item("w.fullF_extra", w_fullF);
                wrt_r_item("w.cvlen.dev", w_cvlen_dev);
                wrt_r_item("w.cvlen.diff", w_cvlen_diff);
close_r_vector();

// VECTOR object of parameters and estimated quantities
open_r_vector("sel.parms");

                wrt_r_item("selpar.L50.fst", selpar_L50_FST);
                wrt_r_item("selpar.slope.fst", selpar_slope_FST);
                wrt_r_item("selpar.L502.fst", selpar_L502_FST);
                wrt_r_item("selpar.slope2.fst", selpar_slope2_FST);

                wrt_r_item("selpar.L50.cvt", selpar_L50_CVT);
                wrt_r_item("selpar.slope.cvt", selpar_slope_CVT);
                wrt_r_item("selpar.L502.cvt", selpar_L502_CVT);
                wrt_r_item("selpar.slope2.cvt", selpar_slope2_CVT);
```

```
                wrt_r_item("selpar.L50.c.hal1", selpar_L50_cHAL1);
                wrt_r_item("selpar.slope.c.hal1", selpar_slope_cHAL1);
                wrt_r_item("selpar.L502.c.hal1", selpar_L502_cHAL1);
                wrt_r_item("selpar.slope2.c.hal1", selpar_slope2_cHAL1);
                wrt_r_item("selpar.L50.c.hal2", selpar_L50_cHAL2);
                wrt_r_item("selpar.slope.c.hal2", selpar_slope_cHAL2);
                wrt_r_item("selpar.L502.c.hal2", selpar_L502_cHAL2);
                wrt_r_item("selpar.slope2.c.hal2", selpar_slope2_cHAL2);

                wrt_r_item("selpar.L50.c.cmb1", selpar_L50_cCMB1);
                wrt_r_item("selpar.slope.c.cmb1", selpar_slope_cCMB1);
                wrt_r_item("selpar.L502.c.cmb1", selpar_L502_cCMB1);
                wrt_r_item("selpar.slope2.c.cmb1", selpar_slope2_cCMB1);
                wrt_r_item("selpar.L50.c.cmb2", selpar_L50_cCMB2);

                wrt_r_item("selpar.L50.hb1", selpar_L50_HB1);
                wrt_r_item("selpar.slope.hb1", selpar_slope_HB1);
                wrt_r_item("selpar.L502.hb1", selpar_L502_HB1);
                wrt_r_item("selpar.slope2.hb1", selpar_slope2_HB1);
                wrt_r_item("selpar.L50.hb2", selpar_L50_HB2);
                wrt_r_item("selpar.slope.hb2", selpar_slope_HB2);
                wrt_r_item("selpar.L502.hb2", selpar_L502_HB2);
                wrt_r_item("selpar.slope2.hb2", selpar_slope2_HB2);
                wrt_r_item("selpar.L50.hb3", selpar_L50_HB3);
                wrt_r_item("selpar.slope.hb3", selpar_slope_HB3);
                wrt_r_item("selpar.L502.hb3", selpar_L502_HB3);
                wrt_r_item("selpar.slope2.hb3", selpar_slope2_HB3);
                wrt_r_item("selpar.L50.hb4", selpar_L50_HB4);
                wrt_r_item("selpar.slope.hb4", selpar_slope_HB4);
                wrt_r_item("selpar.L502.hb4", selpar_L502_HB4);
                wrt_r_item("selpar.slope2.hb4", selpar_slope2_HB4);

                wrt_r_item("selpar.L50.rec3", selpar_L50_MRFSS3);
                wrt_r_item("selpar.slope.rec3", selpar_slope_MRFSS3);
                wrt_r_item("selpar.L502.rec3", selpar_L502_MRFSS3);
                wrt_r_item("selpar.slope2.rec3", selpar_slope2_MRFSS3);
                wrt_r_item("selpar.L50.rec4", selpar_L50_MRFSS4);
                wrt_r_item("selpar.slope.rec4", selpar_slope_MRFSS4);
                wrt_r_item("selpar.L502.rec4", selpar_L502_MRFSS4);
                wrt_r_item("selpar.slope2.rec4", selpar_slope2_MRFSS4);

    wrt_r_item("selpar.Age1.c.hal.D2", selpar_Age1_cHAL_D2);
    wrt_r_item("selpar.Age2.c.hal.D2", selpar_Age2_cHAL_D2);
                //wrt_r_item("selpar.L50.c.hal.D2", selpar_L50_cHAL_D2);
                //wrt_r_item("selpar.slope.c.hal.D2", selpar_slope_cHAL_D2);
                //wrt_r_item("selpar.L502.c.hal.D2", selpar_L502_cHAL_D2);
                //wrt_r_item("selpar.slope2.c.hal.D2", selpar_slope2_cHAL_D2);

    wrt_r_item("selpar.Age1.hb.D3", selpar_Age1_HB_D3);
    wrt_r_item("selpar.Age2.hb.D3", selpar_Age2_HB_D3);
    wrt_r_item("selpar.Age1.hb.D4", selpar_Age1_HB_D4);
    wrt_r_item("selpar.Age2.hb.D4", selpar_Age2_HB_D4);

    wrt_r_item("p.lenc.cHAL", p_lenc_cHAL);
    wrt_r_item("p.lenc.cCMB", p_lenc_cCMB);
    wrt_r_item("p.lenc.HB2", p_lenc_HB2);
    wrt_r_item("p.lenc.HB3", p_lenc_HB3);
    wrt_r_item("p.lenc.HB4", p_lenc_HB4);
    wrt_r_item("p.lenc.MRFSS2", p_lenc_MRFSS2);
    wrt_r_item("p.lenc.MRFSS3", p_lenc_MRFSS3);
    wrt_r_item("p.lenc.MRFSS4", p_lenc_MRFSS4);

    wrt_r_item("p.lenc.cHAL.D", p_lenc_cHAL_D);
    wrt_r_item("p.lenc.HB.D3", p_lenc_HB_D3);
    wrt_r_item("p.lenc.HB.D4", p_lenc_HB_D4);


close_r_vector();


    open_r_matrix("N.age");
    wrt_r_matrix(N, 2, 2);
    wrt_r_namevector(styr, (endyr+1));
    wrt_r_namevector(1, nages);
    close_r_matrix();

    open_r_matrix("N.age.mdyr");
    wrt_r_matrix(N_mdyr, 2, 2);
    wrt_r_namevector(styr, endyr);
    wrt_r_namevector(1, nages);
```

```
  close_r_matrix();

  open_r_matrix("B.age");
  wrt_r_matrix(B, 2, 2);
  wrt_r_namevector(styr, (endyr+1));
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("Z.age");
  wrt_r_matrix(Z, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("L.age.pred.num");
  wrt_r_matrix(L_total_num, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("L.age.pred.klb");
  wrt_r_matrix(L_total_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

//  open_r_matrix("L.age.pred.cHTR.num");
//  wrt_r_matrix(L_cHTR_num, 2, 2);
//  wrt_r_namevector(styr,endyr);
//  wrt_r_namevector(1, nages);
//  close_r_matrix();
//
//  open_r_matrix("L.age.pred.cHAL.num");
//  wrt_r_matrix(L_cHAL_num, 2, 2);
//  wrt_r_namevector(styr,endyr);
//  wrt_r_namevector(1, nages);
//  close_r_matrix();
//
//  open_r_matrix("L.age.pred.cCMB.num");
//  wrt_r_matrix(L_cCMB_num, 2, 2);
//  wrt_r_namevector(styr,endyr);
//  wrt_r_namevector(1, nages);
//  close_r_matrix();
//
//  open_r_matrix("L.age.pred.HB.num");
//  wrt_r_matrix(L_HB_num, 2, 2);
//  wrt_r_namevector(styr,endyr);
//  wrt_r_namevector(1, nages);
//  close_r_matrix();
//
//  open_r_matrix("L.age.pred.rec.num");
//  wrt_r_matrix(L_MRFSS_num, 2, 2);
//  wrt_r_namevector(styr,endyr);
//  wrt_r_namevector(1, nages);
//  close_r_matrix();

// LIST object with annual selectivity at age by fishery

open_r_list("size.age.fishery");

  open_r_matrix("len.cHTR.mm");
  wrt_r_matrix(len_cHTR_mm, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("len.cHAL.mm");
  wrt_r_matrix(len_cHAL_mm, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("len.cCMB.mm");
  wrt_r_matrix(len_cCMB_mm, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("len.HB.mm");
  wrt_r_matrix(len_HB_mm, 2, 2);
```

```
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("len.rec.mm");
  wrt_r_matrix(len_MRFSS_mm, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("len.cHAL.D.mm");
  wrt_r_matrix(len_cHAL_D_mm, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("len.HB.D.mm");
  wrt_r_matrix(len_HB_D_mm, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("len.rec.D.mm");
  wrt_r_matrix(len_MRFSS_D_mm, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();



  open_r_matrix("wgt.cHTR.klb");
  wrt_r_matrix(wgt_cHTR_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("wgt.cHAL.klb");
  wrt_r_matrix(wgt_cHAL_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("wgt.cCMB.klb");
  wrt_r_matrix(wgt_cCMB_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("wgt.HB.klb");
  wrt_r_matrix(wgt_HB_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("wgt.rec.klb");
  wrt_r_matrix(wgt_MRFSS_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("wgt.cHAL.D.klb");
  wrt_r_matrix(wgt_cHAL_D_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("wgt.HB.D.klb");
  wrt_r_matrix(wgt_HB_D_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

  open_r_matrix("wgt.rec.D.klb");
  wrt_r_matrix(wgt_MRFSS_D_klb, 2, 2);
  wrt_r_namevector(styr,endyr);
  wrt_r_namevector(1, nages);
  close_r_matrix();

close_r_list();
```

```
open_r_list("sel.age");

   open_r_vector("sel.v.wgted.L");
   for (iage=1; iage<=nages; iage++)
   {
      wrt_r_item(iage, sel_wgted_L(iage));
   }
   close_r_vector();

   open_r_vector("sel.v.wgted.D");
   for (iage=1; iage<=nages; iage++)
   {
      wrt_r_item(iage, sel_wgted_D(iage));
   }
   close_r_vector();

   open_r_vector("sel.v.wgted.tot");
   for (iage=1; iage<=nages; iage++)
   {
      wrt_r_item(iage, sel_wgted_tot(iage));
   }
   close_r_vector();

   open_r_vector("sel.v.fst");
   for (iage=1; iage<=nages; iage++)
   {
      wrt_r_item(iage, sel_FST_vec(iage));
   }
   close_r_vector();

   open_r_vector("sel.v.cvt");
   for (iage=1; iage<=nages; iage++)
   {
      wrt_r_item(iage, sel_CVT_vec(iage));
   }
   close_r_vector();


   open_r_matrix("sel.m.c.htr");
   wrt_r_matrix(sel_cHTR, 2, 2);
   wrt_r_namevector(styr,endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();

   open_r_matrix("sel.m.c.hal");
   wrt_r_matrix(sel_cHAL, 2, 2);
   wrt_r_namevector(styr,endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();

   open_r_matrix("sel.m.c.cmb");
   wrt_r_matrix(sel_cCMB, 2, 2);
   wrt_r_namevector(styr,endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();

   open_r_matrix("sel.m.hb");
   wrt_r_matrix(sel_HB, 2, 2);
   wrt_r_namevector(styr,endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();

   open_r_matrix("sel.m.rec");
   wrt_r_matrix(sel_MRFSS, 2, 2);
   wrt_r_namevector(styr,endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();

   open_r_matrix("sel.m.c.hal.D");
   wrt_r_matrix(sel_cHAL_D, 2, 2);
   wrt_r_namevector(styr,endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();

   open_r_matrix("sel.m.hb.D");
   wrt_r_matrix(sel_HB_D, 2, 2);
   wrt_r_namevector(styr,endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();
```

```
    open_r_matrix("sel.m.rec.D");
    wrt_r_matrix(sel_MRFSS_D, 2, 2);
    wrt_r_namevector(styr,endyr);
    wrt_r_namevector(1, nages);
    close_r_matrix();

close_r_list();


//LIST object with predicted and observed composition data
open_r_list("comp.mats");

    open_r_matrix("lcomp.fst.ob");
    wrt_r_matrix(obs_FST_lenc, 2, 2);
    wrt_r_namevector(styr_FST_lenc,endyr_FST_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.fst.pr");
    wrt_r_matrix(pred_FST_lenc, 2, 2);
    wrt_r_namevector(styr_FST_lenc,endyr_FST_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

//    open_r_matrix("lcomp.cvt.ob");
//    wrt_r_matrix(obs_CVT_lenc, 2, 2);
//    wrt_r_namevector(styr_CVT_lenc,endyr_CVT_lenc);
//    wrt_r_namevector(lenbins);
//    close_r_matrix();
//
//    open_r_matrix("lcomp.cvt.pr");
//    wrt_r_matrix(pred_CVT_lenc, 2, 2);
//    wrt_r_namevector(styr_CVT_lenc,endyr_CVT_lenc);
//    wrt_r_namevector(lenbins);
//    close_r_matrix();

    open_r_matrix("lcomp.c.hal.ob");
    wrt_r_matrix(obs_cHAL_lenc, 2, 2);
    wrt_r_namevector(styr_cHAL_lenc,endyr_cHAL_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.c.hal.pr");
    wrt_r_matrix(pred_cHAL_lenc, 2, 2);
    wrt_r_namevector(styr_cHAL_lenc,endyr_cHAL_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.c.cmb.ob");
    wrt_r_matrix(obs_cCMB_lenc, 2, 2);
    wrt_r_namevector(yrs_cCMB_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.c.cmb.pr");
    wrt_r_matrix(pred_cCMB_lenc, 2, 2);
    wrt_r_namevector(yrs_cCMB_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.hb.ob");
    wrt_r_matrix(obs_HB_lenc, 2, 2);
    wrt_r_namevector(styr_HB_lenc,endyr_HB_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.hb.pr");
    wrt_r_matrix(pred_HB_lenc, 2, 2);
    wrt_r_namevector(styr_HB_lenc,endyr_HB_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.rec.ob");
    wrt_r_matrix(obs_MRFSS_lenc, 2, 2);
    wrt_r_namevector(styr_MRFSS_lenc,endyr_MRFSS_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();
```

```
open_r_matrix("lcomp.rec.pr");
wrt_r_matrix(pred_MRFSS_lenc, 2, 2);
wrt_r_namevector(styr_MRFSS_lenc,endyr_MRFSS_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.hb.D.ob");
wrt_r_matrix(obs_HB_D_lenc, 2, 2);
wrt_r_namevector(styr_HB_D_lenc,endyr_HB_D_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.hb.D.pr");
wrt_r_matrix(pred_HB_D_lenc, 2, 2);
wrt_r_namevector(styr_HB_D_lenc,endyr_HB_D_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.c.hal.D.ob");
wrt_r_matrix(obs_cHAL_D_lenc, 2, 2);
wrt_r_namevector(yrs_cHAL_D_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.c.hal.D.pr");
wrt_r_matrix(pred_cHAL_D_lenc, 2, 2);
wrt_r_namevector(yrs_cHAL_D_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();



open_r_matrix("acomp.cvt.ob");
wrt_r_matrix(obs_CVT_agec, 2, 2);
wrt_r_namevector(styr_CVT_agec,endyr_CVT_agec);
wrt_r_namevector(1,nages);
close_r_matrix();

open_r_matrix("acomp.cvt.pr");
wrt_r_matrix(pred_CVT_agec, 2, 2);
wrt_r_namevector(styr_CVT_agec,endyr_CVT_agec);
wrt_r_namevector(1,nages);
close_r_matrix();

open_r_matrix("acomp.c.hal.ob");
wrt_r_matrix(obs_cHAL_agec, 2, 2);
wrt_r_namevector(yrs_cHAL_agec);
wrt_r_namevector(1, nages);
close_r_matrix();

open_r_matrix("acomp.c.hal.pr");
wrt_r_matrix(pred_cHAL_agec, 2, 2);
wrt_r_namevector(yrs_cHAL_agec);
wrt_r_namevector(1, nages);
close_r_matrix();

open_r_matrix("acomp.hb.ob");
wrt_r_matrix(obs_HB_agec, 2, 2);
wrt_r_namevector(yrs_HB_agec);
wrt_r_namevector(1, nages);
close_r_matrix();

open_r_matrix("acomp.hb.pr");
wrt_r_matrix(pred_HB_agec, 2, 2);
wrt_r_namevector(yrs_HB_agec);
wrt_r_namevector(1, nages);
close_r_matrix();

open_r_matrix("acomp.rec.ob");
wrt_r_matrix(obs_MRFSS_agec, 2, 2);
wrt_r_namevector(styr_MRFSS_agec,endyr_MRFSS_agec);
wrt_r_namevector(1,nages);
close_r_matrix();

open_r_matrix("acomp.rec.pr");
wrt_r_matrix(pred_MRFSS_agec, 2, 2);
wrt_r_namevector(styr_MRFSS_agec,endyr_MRFSS_agec);
wrt_r_namevector(1,nages);
close_r_matrix();
```

```
close_r_list();

// DATA FRAME of time series
open_r_df("t.series", styr, (endyr+1), 2);
                wrt_r_namevector(styr,(endyr+1));
                wrt_r_df_col("year", styr,(endyr+1));
                wrt_r_df_col("F.full", fullF);
                wrt_r_df_col("F.Fmsy", fullF/F_msy_out);
                wrt_r_df_col("F.c.htr", F_cHTR_out);
                wrt_r_df_col("F.c.hal", F_cHAL_out);
                wrt_r_df_col("F.c.cmb", F_cCMB_out);
                wrt_r_df_col("F.hb", F_HB_out);
                wrt_r_df_col("F.rec", F_MRFSS_out);
                wrt_r_df_col("F.c.hal.D", F_cHAL_D_out);
                wrt_r_df_col("F.hb.D", F_HB_D_out);
                wrt_r_df_col("F.rec.D", F_MRFSS_D_out);

   wrt_r_df_col("recruits", rec);
   //wrt_r_df_col("logR.dev", log_dev_N_rec); //excludes yrs deviations not estimated
   wrt_r_df_col("logR.dev", log_dev_R); //places zeros in yrs deviations not estimated
   wrt_r_df_col("SSB", SSB);
   wrt_r_df_col("SSB.SSBmsy", SSB/SSB_msy_out);
   wrt_r_df_col("SSB.msst", SSB/SSB_msy_out/(1.0-M_constant));
   wrt_r_df_col("B", totB);
   wrt_r_df_col("B.B0", totB/B0);
   wrt_r_df_col("SPR.static", spr_static);

   wrt_r_df_col("total.L.klb", L_total_klb_yr);
   wrt_r_df_col("total.L.knum", L_total_knum_yr);
   wrt_r_df_col("total.D.klb",D_total_klb_yr);
   wrt_r_df_col("total.D.knum",D_total_knum_yr);

   wrt_r_df_col("U.fst.ob", obs_FST_cpue);
   wrt_r_df_col("U.fst.pr", pred_FST_cpue);
   wrt_r_df_col("U.cvt.ob", obs_CVT_cpue);
   wrt_r_df_col("U.cvt.pr", pred_CVT_cpue);
   wrt_r_df_col("U.hal.ob", obs_HAL_cpue);
   wrt_r_df_col("U.hal.pr", pred_HAL_cpue);
   wrt_r_df_col("U.hb.ob", obs_HB_cpue);
   wrt_r_df_col("U.hb.pr", pred_HB_cpue);
   wrt_r_df_col("U.rec.ob", obs_MRFSS_cpue);
   wrt_r_df_col("U.rec.pr", pred_MRFSS_cpue);

   wrt_r_df_col("L.c.htr.ob", obs_cHTR_L);
   wrt_r_df_col("L.c.htr.pr", pred_cHTR_L_klb);
   wrt_r_df_col("L.c.hal.ob", obs_cHAL_L);
   wrt_r_df_col("L.c.hal.pr", pred_cHAL_L_klb);
   wrt_r_df_col("L.c.cmb.ob", obs_cCMB_L);
   wrt_r_df_col("L.c.cmb.pr", pred_cCMB_L_klb);
   wrt_r_df_col("L.hb.ob", obs_HB_L);
   wrt_r_df_col("L.hb.pr", pred_HB_L_knum);
   wrt_r_df_col("L.rec.ob", obs_MRFSS_L);
   wrt_r_df_col("L.rec.pr", pred_MRFSS_L_knum);

   wrt_r_df_col("D.c.hal.ob", obs_cHAL_D);
   wrt_r_df_col("D.c.hal.pr", pred_cHAL_D_knum);
   wrt_r_df_col("D.hb.ob", obs_HB_D);
   wrt_r_df_col("D.hb.pr", pred_HB_D_knum);
   wrt_r_df_col("D.rec.ob", obs_MRFSS_D);
   wrt_r_df_col("D.rec.pr", pred_MRFSS_D_knum);

   //comp sample sizes
   wrt_r_df_col("lcomp.fst.n", nsamp_FST_lenc_allyr);
//    wrt_r_df_col("lcomp.cvt.n", nsamp_CVT_lenc_allyr);
   wrt_r_df_col("lcomp.c.hal.n", nsamp_cHAL_lenc_allyr);
   wrt_r_df_col("lcomp.c.cmb.n", nsamp_cCMB_lenc_allyr);
   wrt_r_df_col("lcomp.hb.n", nsamp_HB_lenc_allyr);
   wrt_r_df_col("lcomp.rec.n", nsamp_MRFSS_lenc_allyr);
   wrt_r_df_col("lcomp.c.hal.D.n", nsamp_cHAL_D_lenc_allyr);
   wrt_r_df_col("lcomp.hb.D.n", nsamp_HB_D_lenc_allyr);

   wrt_r_df_col("acomp.cvt.n", nsamp_CVT_agec_allyr);
   wrt_r_df_col("acomp.c.hal.n", nsamp_cHAL_agec_allyr);
   wrt_r_df_col("acomp.hb.n", nsamp_HB_agec_allyr);
   wrt_r_df_col("acomp.rec.n", nsamp_MRFSS_agec_allyr);

close_r_df();


// DATA FRAME of L and D time series by fishery
```

```
open_r_df("LD.pr.tseries", styr, (endyr+1), 2);
                wrt_r_namevector(styr,(endyr+1));
                wrt_r_df_col("year", styr,(endyr+1));

   wrt_r_df_col("L.c.htr.klb", pred_cHTR_L_klb);
   wrt_r_df_col("L.c.htr.knum", pred_cHTR_L_knum);
   wrt_r_df_col("L.c.hal.klb", pred_cHAL_L_klb);
   wrt_r_df_col("L.c.hal.knum", pred_cHAL_L_knum);
   wrt_r_df_col("L.c.cmb.klb", pred_cCMB_L_klb);
   wrt_r_df_col("L.c.cmb.knum", pred_cCMB_L_knum);
   wrt_r_df_col("L.hb.klb", pred_HB_L_klb);
   wrt_r_df_col("L.hb.knum", pred_HB_L_knum);
   wrt_r_df_col("L.rec.klb", pred_MRFSS_L_klb);
   wrt_r_df_col("L.rec.knum", pred_MRFSS_L_knum);

   wrt_r_df_col("D.c.hal.klb", pred_cHAL_D_klb);
   wrt_r_df_col("D.c.hal.knum", pred_cHAL_D_knum);
   wrt_r_df_col("D.hb.klb", pred_HB_D_klb);
   wrt_r_df_col("D.hb.knum", pred_HB_D_knum);
   wrt_r_df_col("D.rec.klb", pred_MRFSS_D_klb);
   wrt_r_df_col("D.rec.knum", pred_MRFSS_D_knum);
close_r_df();


open_r_df("a.series", 1, nages, 2);
                wrt_r_namevector(1,nages);
                wrt_r_df_col("age", 1,nages);
                wrt_r_df_col("lengthTL", meanlen_TL);
                wrt_r_df_col("lengthFL", meanlen_FL);
                wrt_r_df_col("length.cv", len_cv);
                wrt_r_df_col("weight", wgt_lb);     //for FishGraph
                wrt_r_df_col("wgt.klb", wgt_klb);
                wrt_r_df_col("wgt.mt", wgt_mt);
                wrt_r_df_col("wgt.wgted.L.klb", wgt_wgted_L_klb);
                wrt_r_df_col("wgt.wgted.D.klb", wgt_wgted_D_klb);
                wrt_r_df_col("prop.female", prop_f);
                wrt_r_df_col("mat.female", maturity_f);
                wrt_r_df_col("fecundity", fecundity);
                wrt_r_df_col("reprod", reprod);
                wrt_r_df_col("M", M);
close_r_df();

open_r_df("eq.series", 1, n_iter_msy, 2);
                wrt_r_namevector(1,n_iter_msy);
                wrt_r_df_col("F.eq", F_msy);
                wrt_r_df_col("spr.eq", spr_msy);
                wrt_r_df_col("R.eq", R_eq);
                wrt_r_df_col("SSB.eq", SSB_eq);
                wrt_r_df_col("B.eq", B_eq);
                wrt_r_df_col("L.eq.klb", L_eq_klb);
                wrt_r_df_col("D.eq.knum", D_eq);
close_r_df();

open_r_df("pr.series", 1, n_iter_spr, 2);
                wrt_r_namevector(1,n_iter_spr);
                wrt_r_df_col("F.spr", F_spr);
                wrt_r_df_col("spr", spr_spr);
                wrt_r_df_col("SPR", spr_spr/spr_F0);
                wrt_r_df_col("ypr.lb", L_spr);
close_r_df();


open_r_list("CLD.est.mats");

   open_r_matrix("Lw.c.htr");
      wrt_r_matrix(L_cHTR_klb, 1,1);
   close_r_matrix();

   open_r_matrix("Lw.c.hal");
      wrt_r_matrix(L_cHAL_klb, 1,1);
   close_r_matrix();

   open_r_matrix("Lw.c.cmb");
      wrt_r_matrix(L_cCMB_klb, 1,1);
   close_r_matrix();

   open_r_matrix("Lw.hb");
      wrt_r_matrix(L_HB_klb, 1,1);
   close_r_matrix();
```

```
    open_r_matrix("Lw.rec");
       wrt_r_matrix(L_MRFSS_klb, 1,1);
    close_r_matrix();

    open_r_matrix("Lw.total");
       wrt_r_matrix(L_total_klb, 1,1);
    close_r_matrix();

    open_r_matrix("Ln.c.htr");
       wrt_r_matrix(L_cHTR_num, 1,1);
    close_r_matrix();

    open_r_matrix("Ln.c.hal");
       wrt_r_matrix(L_cHAL_num, 1,1);
    close_r_matrix();

    open_r_matrix("Ln.c.cmb");
       wrt_r_matrix(L_cCMB_num, 1,1);
    close_r_matrix();

    open_r_matrix("Ln.hb");
       wrt_r_matrix(L_HB_num, 1,1);
    close_r_matrix();

    open_r_matrix("Ln.rec");
       wrt_r_matrix(L_MRFSS_num, 1,1);
    close_r_matrix();

    open_r_matrix("Ln.total");
       wrt_r_matrix(L_total_num, 1,1);
    close_r_matrix();

    open_r_matrix("Dn.c.hal");
       wrt_r_matrix(D_cHAL_num,1,1);
    close_r_matrix();

    open_r_matrix("Dn.hb");
       wrt_r_matrix(D_HB_num, 1,1);
    close_r_matrix();

    open_r_matrix("Dn.rec");
       wrt_r_matrix(D_MRFSS_num, 1,1);
    close_r_matrix();
close_r_list();

close_r_file();
```

IV. Routines for creating R object (admb2r.cpp), detailed in NOAA Tech. Memo. NMFS-SEFSC-546.

```
/**********************************************************************************
*
* ADMB2R
*
* Routines for writing data from AD Model Builder to a file read by R with "dget"
*
* Jennifer Martin, Mike Prager and Andi Stephens
* NOAA, National Marine Fisheries Service
*
* jennifer.martin@noaa.gov
* mike.prager@noaa.gov
* andi.stephens@noaa.gov
*
* Original version:  February 2006
*
* Many of these routines are implemented with templates and overloaded functions.
*
* In order to make them compiler-independent, overloaded functions are defined for
* all types anticipated for use.  These functions are gathered together under a
* single comment block describing them as a group.
*
* Overloaded functions are often paired with a similarly named "do_" function or
* template. The overloaded function passes various combinations of options to the
* "do_" function for processing.
*
**********************************************************************************/
/* CHANGELOG
*  Version 0.50   Andi   May 2006       Revised for compatibility
*  Version 0.85   MHP   8 Aug 2006     Changed names of ...info_vector to ...info_list
              to agree with actual R structure.  Changed default of argument
              "writestamp" to true (as documented). Moved documentation
              from Word to LaTeX, as Word file has deteriorated.
*  Version 0.99   Andi  11 Aug 2006    Commented out non-integer functs for
                              writing row/col names.
                              Added test_missing function.
*  Version 1.00   MHP   16 Aug 2006    Added factor of 2 to missing test, changed
                              declaration of dum_matrix to four parameters for
                              compatibility with older versions of ADMB.
*  Version 1.00   MHP   30 Aug 2006    Corrected missing-value test.  Inserted some
                              vector routines that had been removed in error.
*  Version 1.00   JLM   01 Sep 2006    Switch the default behavior of writing out missing
                              values to false. Edited do_wrt_r_namevector to
                              independently assign default vector min/max.
                              Add additional details to some comment sections
                              for clarity.
*  Version 1.01   JLM   09 Mar 2007    Fixed bug that didn't write out default row.names
                              for a dataframe (close_r_df).
                                                                    Added function to write out a
vector using one
                              statement (wrt_r_complete_vector).
                              Fixed a bug that didn't write error messages
                              in many cases.
                                                                    Removed periods at end of
first two comment
                              lines for clarity (do_open_r_file).
**********************************************************************************/
#include <time.h>      // needed for timestamp
#include <string.h>    // string manipulation routines
#include <vector>      // vectors -- no need to manage array allocation by hand
#include <iomanip>     // needed to set precision
#include <iostream>    // needed for converting between data types
#include <sstream>     // needed for converting between data types

// GLOBAL VARIABLES

const char* version = "1.01";              // Version number
int i;                                  // For indices


// ** File I/O Variables

string outfile;                         // output file name
ofstream rfile;                         // ofstream for output file
ofstream errfile;                       // output stream for error message
string err_msg = "ADMB2R error messages:  ";       // error message
```

```
// ** General Housekeeping Variables

int level;                          // Current nesting level for object names
bool OKflag = true;                 // error flag
string mflag;                       // is open object a matrix or data frame
string vecflag;                     // is open vector a list or simple vector
vector<bool> ObjDoneFlag;           // flag to track object completion
vector<string> prevObj;             // ( names of previous object, used in keeping
                                    // ( track of whether the object is complete

// ** Data Writing Variables

double missing = -99999.;           // No data/missing data indicator
double epsilon = 1e-6;              // A small number
bool writeNA = false;               // Flag to turn on/off writing NA for missing data
bool naflag = false;                // Flag to signal use of NA matrix or vector
imatrix dum_matrix;                 // Dummy variable in use of NA matrix
ivector dum_vector;                 // Dummy variable in use of NA vector
int dim1 = -1;                      // matrix dimensions or data frame min/max
int dim2 = -1;                      // matrix dimensions or data frame min/max
int digits = -1;                    // ( Digits of data precision; -1 will write data
                                    // ( as it appears in ADMB; 0 writes integers

// ** R Names Variables

vector<string> Rnames;              // vector of names to write when closing the R object
string colnames;                    // list of names to be used for R columns
string rownames;                    // list of names to be used for R rows
int rowflag = 0;                    // Flag to write matrix or data frame row names
int colflag = 0;                    // Flag to write matrix or data frame column names
const char* quote = "\"";           // Double-quote character (")
const char* cquote = ",\"";         // comma plus double quote (,")

//=============================================================================
// convert
//
// Utility routine to convert one type to another, e.g., double to string
//
// Use:
//     out_type - type desired
//     in_value - type to convert from
//=============================================================================
template <class out_type, class in_value>
out_type convert(const in_value & t) {
   stringstream stream;
   stream << t; // insert value to stream
   out_type result; // store conversion's result here
   stream >> result; // write value to result
   return result;
} // end convert


//=============================================================================
// test_missing
//
// Utility routine to test for the missing value.
// Returns true if the input value is the missing value.
//
//=============================================================================

bool test_missing(double num) {
   return (fabs(num - missing) < epsilon);
}

//=============================================================================
// write_errmsg
//
// Utility routine to write error message string (global) to screen and to logfile.
//
// No arguments.
//=============================================================================
void write_errmsg() {
   // write error message to screen and file
   if ( err_msg != "ADMB2R error messages:  " ) {
      cout << "**** ADMB2R Error: Please check file admb2r.log for error messages." << endl;
   }
   errfile.open ("admb2r.log");
   errfile << err_msg << endl;
```

```
   errfile.close();
} // end write_errmsg

//================================================================================
// do_open_r_file
//
// Open the output file and initialize the R data object.
//
// Arguments are passed to do_open_r_file by the overloaded open_r_file function.
//
// ARGUMENTS
//   fname - name of output file
//   numdigits - data precision
//         digits = -1 (default) or digits = 0 writes data out with default precision
//         of 6; digits > 0 will write ALL data out in scientific notation with the
//         specified number of digits after the decimal place.
//================================================================================
void do_open_r_file(char* fname, int numdigits) {
   // initialize nesting level
   level = 0;

   // initialize dummy matrix
   dum_matrix.allocate(1,1,1,1);
               // initialize dummy vector
               dum_vector.allocate(1,1);

   // intialize object completion tracking variables
   ObjDoneFlag.clear();
   ObjDoneFlag.push_back(true);
   prevObj.clear();
   prevObj.push_back("");

   // initialize list of all R objects
   Rnames.clear();
   Rnames.push_back("c(");

   // initialize row/column names and vector of list objects
   colnames.clear();
   rownames.clear();

   // Open the R output file
   outfile = fname;
   rfile.open (fname);
   // check to make sure it opened OK
   if ( ! rfile.is_open() ) {
      err_msg = err_msg + "\n**** ADMB2R Error: Unable to open " + outfile;
      OKflag = false;
                        write_errmsg();
      return;
   }

   // write a brief file header
   rfile <<"### This file written with ADMB2R version " << version << endl;
   rfile <<"### Read into R or S with x=dget('";
   rfile <<  fname << "')";
   rfile << endl;
   rfile << endl;

   // begin overall R structure (list)
   rfile << "structure(list(";
   rfile << endl;
   rfile << endl;

   // set precision based on the digits value specified.
   digits = numdigits;
   if ( digits > 0 ) {
      rfile << scientific;
      rfile.precision(digits);
   }

} // End do_open_r_file

//================================================================================
// open_r_file
//
// Overloaded function to get the R file name, precision, and set the missing value
// indicator.
//
// Allows user to specify one of three combinations of open_r_file:
// open_r_file(fname)
```

```
// open_r_file(fname, numdigits)
// open_r_file(fname, numdigits, ismissing)
//
// ARGUMENTS:
//   fname - name of output file.
//         This argument is passed to do_open_rfile for further handling.
//   numdigits - (optional) data precision
//         This argument is passed to do_open_rfile for further handling.
//         digits = -1 (default) or digits = 0 writes data out with default precision
//         of 6; digits > 0 will write ALL data out in scientific notation with the
//         specified number of digits after the decimal place.
//   ismissing - (optional) indicates the value that is used to represent a missing datum.
//         If ismissing is specified, data matching this value will be replaced by NA in
//         the R file.
//=============================================================================================
void open_r_file(char* fname, int numdigits = -1) {

  // no missing value supplied, so turn off flag to write NAs to file
                writeNA = false;

  // pass info to do_open_r_file for processing
                do_open_r_file(fname, numdigits);

} // End open_r_file (numdigits)

//=============================================================================================
void open_r_file(char* fname, int numdigits, double ismissing) {

  // missing is value supplied, so turn on flag to write NAs to file
                writeNA = true;
  // asign missing value to global variable
                missing = ismissing;

  // pass info to do_open_r_file for processing
  do_open_r_file(fname, numdigits);

} // End open_r_file (numdigits, ismissing)


//=============================================================================================
// close_r_file
//
// Close R object and do housekeeping
//
// No arguments.
//=============================================================================================
void close_r_file() {

   if ( OKflag == false ) {
      write_errmsg();
      if ( rfile.is_open() ) rfile.close();
      return; // exit if there was an earlier error
   }

   // check that there is at least one item in file
   if ( Rnames[0] == "c(" ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: No data written to " + outfile;
      OKflag = false;
      write_errmsg();
      return;
   }

   // check level; if level not equal to zero, one of the list objects didn't close properly
   if ( level != 0 ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Close list object with close_r_list().";
      write_errmsg();
      OKflag = false;
      return;
   }

   // check that final object is complete
   if ( ObjDoneFlag[0] = false ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: " + prevObj[level] + " is not complete!";
      OKflag = false;
      write_errmsg();
      return;
   }
```

```
   rfile << endl;
   rfile << "### Calling close_r_file -- last call in program." << endl;
   rfile << ")," << endl;
   rfile << endl;
   rfile << ".Names = " << Rnames[0] << "))";
   rfile.close();

   // clear global variables: Rnames, colnames, rownames, ListNames
   Rnames.clear();
   colnames.clear();
   rownames.clear();
   prevObj.clear();
   prevObj.push_back("");
   ObjDoneFlag.clear();
   ObjDoneFlag.push_back(true);

   // re-set nesting level
   level = 0;

   write_errmsg();

} // end close_r_file


//=============================================================================================
// wrt_r_comment
//
// Write a comment to the output file.  Cannot be used before open_r_file.
//
// ARGUMENTS:
//     text - text to write as comment
//=============================================================================================
void wrt_r_comment(char* text) {
   if ( ! rfile.is_open() ) { // exit if file hasn't been opened yet
      OKflag = false;
      err_msg = err_msg + "**** ADMB2R Error:  No open file\n";
                          write_errmsg();
      return;
   }

   // if something goes wrong
   if ( rfile.bad() ) {
      if ( rfile.is_open() ) rfile.close();
      OKflag = false;
      err_msg = err_msg + "**** ADMB2R Error: Unable to write to " + outfile + "\n";
                          write_errmsg();
      return;
   }

   rfile << "### " << text << endl;
} // end wrt_r_comment


//=============================================================================================
//  reg_Rnames
//
//  Adds object names to Names list, checks that previous object is complete,
//  and does other housekeeping chores
//
//  Called by open_r_matrix, open_r_list, open_r_info_list, open_r_vector, and open_r_df.
//
//  ARGUMENTS:
//    name - name of object to write
//    description - type of R object (used in error reporting if object is incomplete).
//=============================================================================================
int reg_Rnames(char* name, string description) {

   // check for previous object completion
   if ( OKflag == false ) return 0; // exit if there was an earlier error
   if ( prevObj[level] != "" ) {      // make sure there are previous objects to check
      //previous object is incomplete
      if ( ObjDoneFlag[level] == false ) {
         if ( rfile.is_open() ) rfile.close();
         err_msg = err_msg + "\n**** ADMB2R Error: " + prevObj[level] + " is still open";
         OKflag = false;
                                    write_errmsg();
         return 0;
      }
   }
```

```
   // add object name to list
   // if item not first in list add comma separator
   if ( Rnames[level] != "c(" ) {
      Rnames[level] = Rnames[level] + ", ";
      rfile << ", ";
   }
   Rnames[level] = Rnames[level] + quote + name + quote;

   // intialize object completion variables
   prevObj[level] = description + name;
   ObjDoneFlag[level] = false;

   if ( OKflag == true ) return 1;
   else return 0;

}  // end reg_Rnames


//=========================================================================================
// add_colname
//
// Utility to add column name to list, with comma as necessary
//
// ARGUMENTS:
//      name - string to output.
//=========================================================================================
void add_colname(char* name) {
   if ( colnames != "c(" ) {
      colnames = colnames + ", ";  // Object is not first item; preceed with a comma
   }

   colnames = colnames + quote + name + quote;  // Add name to list

} // end add_colname


//=========================================================================================
// add_rowname
//
// Utility to add row name to list, with comma as necessary
//
// ARGUMENTS:
//      name - string to output.
//=========================================================================================
void add_rowname(char* name) {
   if ( rownames != "c(" ) {
      rownames = rownames + ", ";  // Object is not first item; preceed with a comma
   }

   rownames = rownames + quote + name + quote;  // Add name to list

} // end add_rowname


//=========================================================================================
// check_rownames
//
// In writing row or column names, determines whether the item to write is a row or a column.
// Performs bounds-checking.
//
// Called from do_wrt_r_namevector and do_wrt_r_numvector
//
// ARGUMENTS
//     start - first value in the series, or index of the first element in the vector
//     stop - last value in the series, or index of the last element in the vector
//     inc - how much to increment the values in the series
//=========================================================================================
template <class T>
string check_rownames (T start, T stop, T inc) {

   string return_val = "error";           // return string indicating error, row or col

   T nitems = ((stop - start)/inc) + 1;    // number of items to write
   T diff = dim2 - dim1 + 1;               // number of values in matrix or data frame

   int M_dim;                          // dimension of row or column
   string cr_names;                      // variable to specify row or column names

   // determine if object is matrix or data frame and whether item is row or column names
```

```
  if ( mflag == "matrix" && rowflag == 2 && rownames == "" ) { // process rows first
     cr_names = rownames;              // item to write is row names for matrix
     return_val = "row";
  } else {
     if ( mflag == "matrix" && colflag == 2 && colnames == "" ) {
        cr_names = colnames;           // item to write is column names for matrix
        return_val = "col";
     }

     else {
        if ( mflag == "data frame" ) {
           cr_names = rownames;       // item is row names for data frame
           return_val = "row";
        }
     }
  }

  // validate number of items
  if ( mflag == "matrix" ) {           // this is a matrix object
     if ( cr_names == rownames ) {     // process rows first
        M_dim = dim1;
     }                                 // get matrix row dimension
     else {
        if ( cr_names == colnames ) M_dim = dim2;
     }                                 // get matrix col dimension

     // check to see if # elements OK
     if ( nitems != M_dim ) {
        if ( rfile.is_open() ) rfile.close();
        err_msg = err_msg + "\n**** ADMB2R Error: Number of matrix indices in wrt_r_namevector for ";
        err_msg = err_msg + prevObj[level] + " shoud be " + convert<string>(M_dim);
        OKflag = false;
                                       write_errmsg();
        return "error";
     }
  }

  // end of branch for matrix, begin branch for data frame

  else {
     // check that this is being called from a matrix or data frame
     if ( mflag == "" ) {
        if ( rfile.is_open() ) rfile.close();
        err_msg = err_msg + "\n**** ADMB2R Error: Invalid use of wrt_r_namevector for " + prevObj[level];
        OKflag = false;
                                       write_errmsg();
        return "error";
     }
     // check that number of items to print is the same as the data frame
     if ( nitems != diff ) {
        if ( rfile.is_open() ) rfile.close();
        err_msg = err_msg + "\n**** ADMB2R Error: Number of items to write in wrt_r_namevector for " ;
        err_msg = err_msg + prevObj[level] + " shoud be " + convert<string>(diff);
        OKflag = false;
                                       write_errmsg();
        return "error";
     }
  }
  return return_val;
}; // end check_rownames

//=============================================================================
// open_r_matrix
//
// Opens the matrix object and does housekeeping tasks
//
// ARGUMENTS:
//      name - name of matrix to write to file.
//=============================================================================

void open_r_matrix (char* name) {

  // add info object name to list and check for object completion
  int flag = reg_Rnames(name, "Matrix Object ");
  if ( flag == 0 ) return;

  // set matrix/data frame flag for wrt_r_namevector
  mflag = "matrix";

  rfile << name << " = structure(c(" << endl;
```

```
};

//================================================================================
// close_r_matrix
//
// Closes the matrix object and does housekeeping tasks
//
// No arguments.
//================================================================================
void close_r_matrix() {
   if ( OKflag == false ) return;    // exit if there was an earlier error

   // check that row and column names are not empty
   if (rownames.empty() ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Please add row names to ";
      err_msg = err_msg + prevObj[level] + " using wrt_r_namevector";
      OKflag = false;
                              write_errmsg();
      return;
   }
   if (colnames.empty() ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Please add column names to ";
      err_msg = err_msg + prevObj[level] + " using wrt_r_namevector";
      OKflag = false;
                              write_errmsg();
      return;
   }

   rfile << ".Dimnames = list(" << endl;

   // Write row info
   rfile << rownames;

   // write out row/column names
   rfile <<  ", " << endl;

   rfile << colnames;

   // write out rest of object
   rfile << "))" << endl;
   rfile << endl;

   // re-set matrix/data frame flag for wrt_r_namevector
   mflag.clear();

   // set object complete flag
   ObjDoneFlag[level] = true;

   // clear row/col names and dimensions and write flags for next use
   rownames.clear();
   colnames.clear();
   rowflag = 0;
   colflag = 0;
   dim1 = -1;
   dim2 = -1;

   // if something goes wrong
   if ( rfile.bad() ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Unable to write to " + outfile;
      OKflag = false;
                              write_errmsg();
      return;
   }
} // end close_r_matrix

//================================================================================
// do_wrt_r_matrix
//
// Write a matrix subobject to the R data object.
// Matrices differ from data frames in that columns need not have names,
// and they are of uniform type in all columns, e.g. double.
//
// After this function is used, the column names must be written separately
// to complete the matrix.  That is done because the column names may or may
// not be the same as the column indices.  (Row names are assumed to be the
// same as the row indices, generally years.)
//
```

```
// Arguments are passed to do_wrt_r_matrix by the overloaded wrt_r_matrix function.
//
// ARGUMENTS:
//   xx - the matrix
//   na_matrix - a boolean matrix indicating which positions in the xx matrix
//      should be replaced with the NA missing value indicator. A value of 1 (true)
//      indicates the spot to replace with NA.
//=============================================================================================
template <class T>
void do_wrt_r_matrix (const T& xx, imatrix& na_matrix) {

   int ir, ic;                    // row/column iterators in for statement
   int ra, rz, ca, cz;            // for matrix bounds

   ra = xx.rowmin();              // Get starting row index
   rz = xx.rowmax();              // Get ending row index
   ca = xx.colmin();              // Get starting column index
   cz = xx.colmax();              // Get ending column index

   // Write the matrix data
   for ( ic=ca; ic<=cz; ic++ ) {
      for ( ir=ra; ir<=rz; ir++ ) {

         // if value is the missing value indicator, and we're
         // using a missing value "value",  write "NA" instead
         if ( test_missing(convert<double>(xx(ir,ic))) && writeNA == true ) {
            rfile << "NA";
         }
         // if instead we're using a matrix of booleans to
         // indicate the position of missing values, check
         // to see if this position is a missing value
         else if (naflag && na_matrix[ir][ic]){
            rfile << "NA";
         }
         // otherwise use the value we're given.
         else {
            rfile << xx(ir,ic);
         }
         // write proper punctuation
         if ( ic==cz && ir==rz ) {
            rfile << ");";
         } else {
            rfile << ", ";
         }
      }
      rfile << endl;
   }

   // Write dimensions of the matrix and save to dim1 and dim2
   i = rz-ra+1;     // # of row elements
   dim1 = i;
   rfile << ".Dim = c(" << i;
   i = cz-ca+1;    // # of column elements
   dim2 = i;
   rfile << "," << i << ")," << endl;

   //set matrix row and column names
   string temp;
   if ( rowflag == 0 ) rownames = "NULL";
   if ( rowflag == 1 ) {                  // write matrix row indices
      rownames = "c(";
      for ( ir=ra; ir<=rz; ir++ ) {
         temp = convert <string> (ir);   //convert index (integer) to string
         rownames = rownames + quote + temp + quote; //add index to list
         if ( ir==rz )
            rownames = rownames + ")";  // add appropriate punctuation
         else
            rownames = rownames + ", ";
      }
   }
   if ( rowflag == 2 ) rownames.clear();    // write row names with wrt_r_namevector

   if ( colflag == 0 ) colnames = "NULL";
   if ( colflag == 1 ) {                  // write matrix col indices
      colnames = "c(";
      for ( ic=ca; ic<=cz; ic++ ) {
         temp = convert <string> (ic);   //convert index (integer) to string
         colnames = colnames + quote + temp + quote; //add index to list
         if ( ic==cz )
            colnames = colnames + ")";  // add appropriate punctuation
```

```
        else
          colnames = colnames + ", ";
      }
    }
    if ( colflag == 2 ) colnames.clear();    // write column names with wrt_r_namevector

};  // end do_wrt_r_matrix

//=============================================================================================
// wrt_r_matrix
//
// Overloaded function to write a matrix object.
// Defined here for types dvar_matrix, dmatrix, and imatrix.
//
// ARGUMENTS
//   xx - the matrix
//      This argument is passed to do_wrt_r_matrix for further handling.
//   rowoption, coloption - flags for whether to write row names and column names. Optional.
//      0 = write NULL for row or column (Default).
//      1 = write names with same index as matrix.
//      2 = write names with a vector or sequence of numbers.
//   isna - is an NA_matrix supplied. Optional.
//      false = no matrix will be supplied (Default). true = a NA matrix will follow.
//   na_matrix - (optional) a boolean matrix indicating which positions in the xx matrix
//      should be replaced with the NA missing value indicator. A value of 1 (true)
//      indicates the spot to replace with NA. This argument is passed to do_wrt_r_matrix
//      for further handling.
//=============================================================================================
void wrt_r_matrix(const dvar_matrix& xx, int rowoption = 0, int coloption = 0,
                  bool isna = false, imatrix& na_matrix = dum_matrix) {

    // Set global flags
    rowflag = rowoption;
    colflag = coloption;
    naflag = isna;

    do_wrt_r_matrix<dvar_matrix> (xx, na_matrix);

} // wrt_r_matrix_wrt (dvar_matrix)
//=============================================================================================
void wrt_r_matrix(const dmatrix& xx, int rowoption = 0, int coloption = 0,
                  bool isna = false, imatrix& na_matrix = dum_matrix) {

    // Set global flags
    rowflag = rowoption;
    colflag = coloption;
    naflag = isna;

    do_wrt_r_matrix<dmatrix> (xx, na_matrix);

} // wrt_r_matrix_wrt (dmatrix)
//=============================================================================================
void wrt_r_matrix(const imatrix& xx, int rowoption = 0, int coloption = 0,
                  bool isna = false, imatrix& na_matrix = dum_matrix) {

    // Set global flags
    rowflag = rowoption;
    colflag = coloption;
    naflag = isna;

    do_wrt_r_matrix<imatrix> (xx, na_matrix);

} // end wrt_r_matrix (imatrix)


//=============================================================================================
// do_wrt_r_namevector
//
// Function template for writing ADMB vector type row or column items
// Called from overloaded wrt_r_namevector functions.
//
// ARGUMENTS:
//    rowvec = the vector to use
//    start = position in vector at which to start writing
//    stop = position in vector at which to end writing
//=============================================================================================
template <class T>
void do_wrt_r_namevector (const T& rowvec, int start, int stop) {

    string temp;
```

```
   string cr_names;                        // temp name for row or column names list

   // if using defaults (start=0, stop=0) then get vector bounds
   if ( start == 0 && stop == 0 ) {
      start = (rowvec).indexmin();
      stop = (rowvec).indexmax();
   } else if ( stop == 0 ) {
      stop = (rowvec).indexmax();
   }

   // do error checking and get which item (row or column names) to write
   string test = check_rownames<int> (start, stop, 1);
   if ( test == "error" ) return;

   if ( test == "row" ) cr_names = rownames;
   else cr_names = colnames;

   // now assign values to row or column names
   cr_names = "c(";
   for ( i=start; i<=stop; i++ ) {
      temp = convert <string> (rowvec[i]);       //convert vector to string
      cr_names = cr_names + quote + temp + quote; //add index to list
      if ( i==stop ) {
         cr_names = cr_names + ")";
      } else {
         cr_names = cr_names + ", ";
      }
   }

   if ( test == "col" )  colnames = cr_names;      // re-assign back to row- or colnames
   else rownames = cr_names;

}; // end do_wrt_r_namevector
//=============================================================================================
// do_wrt_r_numvector
//
// Function template for writing row or column items using a series of numbers
// Called from wrt_r_namevector
//
// ARGUMENTS:
//     start = value to start the series
//     stop = value to end the series
//     inc = the increment between series values
//=============================================================================================
template <class T>
void do_wrt_r_numvector (const T& start, const T& stop, T inc) {

   string temp;
   string cr_names; // temp name for row or column names list
   T iter;          // iterator

   // do error checking and get which item (row or column names) to write
   string test = check_rownames<T> (start, stop, inc);
   if ( test == "error" ) return;

   if ( test == "row" ) cr_names = rownames;
   else cr_names = colnames;

   // now assign values to row or column names
   cr_names = "c(";
   temp = convert <string> (start);
   cr_names = cr_names + quote + temp + quote;
   iter = start + inc;

   while ( iter<=stop ) {
      temp = convert <string> (iter);
      cr_names = cr_names + ", " + quote + temp + quote;
      iter = iter + inc;
   }

   cr_names = cr_names + ")";
   if ( test == "col" )  colnames = cr_names;  // re-assign back to rownames or colnames
   else rownames = cr_names;

}; // End do_wrt_r_numvector

//=============================================================================================
// wrt_r_namevector
//
// Overloaded function to write matrix or data frame row/column names.
```

```
// Defined here for int and ivector.
// (The functions for ADMB dvector and dvar_vector types have been commented out to
//  prevent possible rounding errors.)
//
// Arguments
//    start - value to start row/column names with
//    stop - value to end row/column names with
//    inc - value to increment row/column names. Optional.
//=============================================================================
void wrt_r_namevector(const int& start, const int& stop, int inc = 1) {
   if ( OKflag == false ) return; // exit if there was an earlier error

   do_wrt_r_numvector<int> (start, stop, inc);

} // end wrt_r_namevector (int)

////===========================================================================
void wrt_r_namevector(const ivector& rowvec, int start = 0, int stop = 0) {
   if ( OKflag == false ) return; // exit if there was an earlier error

   do_wrt_r_namevector<ivector> (rowvec, start, stop);

} // end wrt_r_namevector (ivector)

////===========================================================================
//void wrt_r_namevector(const dvector& rowvec, int start = 0, int stop = 0) {
//    if ( OKflag == false ) return; // exit if there was an earlier error
//
//    do_wrt_r_namevector<dvector> (rowvec, start, stop);
//
//}  // end wrt_r_namevector (dvector)
//
//=============================================================================
//void wrt_r_namevector(const dvar_vector& rowvec, int start = 0, int stop = 0) {
//    if ( OKflag == false ) return; // exit if there was an earlier error
//
//    do_wrt_r_namevector<dvar_vector> (rowvec, start, stop);
//
//} // end wrt_r_namevector (dvar_vector)


//=============================================================================
// open_r_list
//
// Initalize a LIST object: add object to list of R objects, increment level and
// initialize the level checking variables
//
// ARGUMENTS:
//      name - name of object
//=============================================================================
void open_r_list(char* name) {
   // add info object name to list and check for object completion
   int flag = reg_Rnames(name, "List Object ");
   if ( flag == 0 ) return;

   // write beginning of structure to file
   rfile << name << " = structure(list(" << endl;
   rfile << endl;

   // increase level and initialize variables
   level = level + 1;
   Rnames.push_back("c(");

   // initialize object completion checking variables
   prevObj.push_back("");
   ObjDoneFlag.push_back(true);

} // end open_r_list


//=============================================================================
// close_r_list
//
// Close LIST object, decrement level and do housekeeping
//
// No arguments.
//=============================================================================
void close_r_list() {
   if ( OKflag == false ) return; // exit if there was an earlier error
```

```
   // check that at least one item is included in list
   if ( Rnames[level] == "c(" ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: No data written to " + prevObj[level];
      OKflag = false;
                              write_errmsg();
      return;
   }

   // add closing punctuation and list of subobjects
   rfile << "), .Names = " << Rnames[level] << "))" << endl;
   rfile << endl;

   // clear/remove global variables for this level
   Rnames.erase(Rnames.end(),Rnames.end());
   Rnames[level] = "c(";

   // decrease level
   level = level - 1;
   // set object complete flag
   ObjDoneFlag[level] = true;


   // if something goes wrong
   if ( rfile.bad() ) {
      if ( rfile.is_open() ) rfile.close();
      OKflag = false;
      err_msg = err_msg + "\n**** ADMB2R Error: Unable to write to " + outfile;
                              write_errmsg();
      return;
   }
} // end r_list_close


//=========================================================================================
//   open_r_info_list
//
//   Initialize an information vector object and optionally write its DATE subobject.
//   The info-vector contains descriptive information about the data.
//   All major R objects should begin with an info-vector object.
//
//   ARGUMENTS
//      name - name of INFO object (e.g., "metadata")
//      writestamp - whether or not to write the date subobject. Optional.
//=========================================================================================
void open_r_info_list(char* name, bool writestamp=true) {
   // add info object name to list and check for object completion
   int flag = reg_Rnames(name, "Info Object ");
   if ( flag == 0 ) return;

   rfile << name << " = structure(list(" << endl;

   // inform the world there's an info_list open
   vecflag = "info";

   // initialize names of info data
   colnames = "c(";
   if ( writestamp ) {

      // get date and time
      time_t ltime;
      struct tm *today;
      char tmpbuf[50];

      time( &ltime );  // get time as a long integer.
      today = localtime( &ltime );  // convert to local time.
      strftime( tmpbuf, 50,
            "%A, %d %b %Y at %H:%M:%S", today );  // apply formatting.

      // write date and time stamp
      rfile << "date = " << quote << tmpbuf << quote;

      // add to names of info data
      colnames = colnames + quote + "date" + quote;
   }
} //end open_r_info_list


//=========================================================================================
// close_r_info_list
```

```
//
// Close info-vector object
//
// No arguments.
//=========================================================================================
void close_r_info_list() {
   if ( OKflag == false ) return;          // exit if there was an earlier error

   if ( colnames == "c(" ) {               // check that there is at least one item
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: No items written to ";
      err_msg = err_msg + prevObj[level] + " using wrt_r_item.";
      OKflag = false;
                              write_errmsg();
      return;
   }

   rfile << ")," << endl;
   rfile << ".Names = " << colnames << "))" << endl;
   rfile << endl;

   colnames.clear();                   // clear row and column names
   rownames.clear();

   vecflag.clear();

   ObjDoneFlag[level] = true;          // set object complete flag

} // end close_r_info_list


//=========================================================================================
//   open_r_vector
//
//   Initialize a simple vector object.
//   The vector an named, unordered list of numbers, characters, or logical values.
//   ARGUMENTS:
//      name - name of vector object (e.g., "agevector")
//=========================================================================================
void open_r_vector(char* name) {
   // add vector name to list and check for object completion
   int flag = reg_Rnames(name, "Info Object ");
   if ( flag == 0 ) return;

   // inform the world a vector item is open
   vecflag = "vector";

   rfile << name << " = structure(c(" << endl;

   // initialize names of info data
   colnames = "c(";
} //end open_r_vector


//=========================================================================================
// close_r_vector
//
// Close simple-vector object
//
// No arguments.
//=========================================================================================
void close_r_vector() {
   if ( OKflag == false ) return;          // exit if there was an earlier error

   if ( colnames == "c(" ) {               // check that there is at least one item
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: No items written to ";
      err_msg = err_msg + prevObj[level] + " using wrt_r_item.";
      OKflag = false;
                              write_errmsg();
      return;
   }

   rfile << ")," << endl;
   rfile << ".Names = " << colnames << "))" << endl;
   rfile << endl;

   colnames.clear();                   // clear row and column names
   rownames.clear();
   vecflag.clear();
```

```
   ObjDoneFlag[level] = true;              // set object complete flag

} // end close_r_vector


//=============================================================================================
// wrt_r_item
//
// Overloaded function to write a name - value pair to the INFO object.
// Defined here for char *, int, double, bool, dvarible values
//
// ARGUMENTS
//   name - name of data subobject (int or char*)
//   value - corresponding datum
//=============================================================================================
void wrt_r_item(char* name, char* value) {
   if ( OKflag == false ) return;                  // exit if there is an error
   if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

   add_colname(name);
   if (vecflag == "vector") {
      rfile << value;
   } else {
   rfile << name << " = " << quote << value << quote;
   }

} //end wrt_r_item(char*)
//=============================================================================================
void wrt_r_item(char* name, bool value) {
   if ( OKflag == false ) return;                  // exit if there is an error
   if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

   add_colname(name);

   if (vecflag == "vector") {
      if ( value ) {
         rfile << "TRUE";
      } else {
         rfile << "FALSE";
      }
   } else {
      if ( value ) {
         rfile << name << " = " << "TRUE";
      } else {
         rfile << name << " = " << "FALSE";
      }
   }

} // end wrt_r_item(boolean)
//=============================================================================================
void wrt_r_item(int name, bool value) {
   if ( OKflag == false ) return;                  // exit if there is an error
   if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

   add_colname(convert<char*>(name));

   if (vecflag == "vector") {
      if ( value ) {
         rfile << "TRUE";
      } else {
         rfile << "FALSE";
      }
   } else {
      if ( value ) {
         rfile << convert<char*>(name) << " = " << "TRUE";
      } else {
         rfile << convert<char*>(name) << " = " << "FALSE";
      }
   }

} // end wrt_r_item(boolean)
//=============================================================================================
void wrt_r_item(char* name, int value) {
   if ( OKflag == false ) return;                  // exit if there is an error
   if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

   add_colname(name);
   if (vecflag == "vector") {
      rfile << value;
```

```
    } else {
    rfile << name << " = " <<  value;
    }

} //end wrt_r_item(integer)
//=============================================================================================
void wrt_r_item(int name, int value) {
    if ( OKflag == false ) return;                  // exit if there is an error
    if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

    add_colname(convert<char*>(name));
    if (vecflag == "vector") {
        rfile << value;
    } else {
    rfile << convert<char*>(name) << " = " <<  value;
    }

} //end wrt_r_item(integer)
//=============================================================================================
void wrt_r_item(char* name, double value) {
    if ( OKflag == false ) return;                  // exit if there is an error
    if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

    add_colname(name);
    if (vecflag == "vector") {
        rfile << value;
    } else {
    rfile << name << " = " <<  value;
    }

} //end wrt_r_item(double)
//=============================================================================================
void wrt_r_item(int name, double value) {
    if ( OKflag == false ) return;                  // exit if there is an error
    if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

    add_colname(convert<char *>(name));
    if (vecflag == "vector") {
        rfile << value;
    } else {
    rfile << convert<char*>(name) << " = " <<  value;
    }

} //end wrt_r_item(double)
//=============================================================================================
void wrt_r_item(char* name, dvariable value) {
    if ( OKflag == false ) return;                  // exit if there is an error
    if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

    add_colname(name);
    if (vecflag == "vector") {
        rfile << value;
    } else {
    rfile << name << " = " <<  value;
    }

} //end wrt_r_item(dvariable)
//=============================================================================================
void wrt_r_item(int name, dvariable value) {
    if ( OKflag == false ) return;                  // exit if there is an error
    if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

    add_colname(convert<char*>(name));
    if (vecflag == "vector") {
        rfile << value;
    } else {
    rfile << convert<char*>(name) << " = " <<  value;
    }

} //end wrt_r_item(dvariable)

// overloaded functions to write NA's when no value argument is given
//=============================================================================================
void wrt_r_item(char* name) {
    if ( OKflag == false ) return;                  // exit if there is an error
    if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

    add_colname(name);
    if (vecflag == "vector") {
        rfile << "NA";
```

```
   } else {
   rfile << name << " = NA";
   }

} //end wrt_r_item
//=============================================================================
void wrt_r_item(int name) {
   if ( OKflag == false ) return;                  // exit if there is an error
   if ( colnames != "c(" ) rfile << "," << endl;    // comma needed if not first item.

   add_colname(convert<char*>(name));
   if (vecflag == "vector") {
      rfile << "NA";
   } else {
   rfile << convert<char*>(name) << " = NA";
   }

} //end wrt_r_item

//=============================================================================
// open_r_df
//
// Initializes a data frame object.
// Data frames differ from matrices in that they require named columns,
// which are individually written and may be of different types, e.g.,
// columns of integers interspersed among columns of doubles.
//
// ARGUMENTS
//    name - name of object
//    start, stop - define the width of the data frame
//                  and the data frame's coordinate system
//                  i.e., bounds for the data frame vectors.
//    writerow - flag to write row.names (optional).
//       0 = do not write row.names (default).
//       1 = write row.names using the index values supplied
//       2 = write row.names with vector or other values
//=============================================================================
void open_r_df(char* name, int start = -1, int stop = -1, int writerow = 0) {
   string temp;

   // add info object name to list and check for object completion
   int flag = reg_Rnames(name, "Data Frame Object ");
   if ( flag == 0 ) return;

   // intialize matrix/data frame flag
   mflag = "data frame";

   // set rowflag
   rowflag = writerow;

   // check validity of start/stop values
   if ( start == stop && start != -1 ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Invalid index min and max values in open_r_df for the data frame " + name;
      OKflag = false;
                              write_errmsg();
      return;
   }

   // intialize min and max values for data bounds checking
   dim1 = start;
   dim2 = stop;


   //initialize vector names list
   colnames = "c(";

   // if user has chosen to write the row.names with the index values (writerow = 1), get row values now
                  if ( rowflag == 1 ) wrt_r_namevector(dim1, dim2);

   // write beginning of object
   rfile << name << " = structure(list(" << endl;

} // end open_r_df


//=============================================================================
// close_r_df
//
// Writes data frame row and column names, adds punctuation, and does housekeeping.
```

```
//
// No arguments.
//================================================================================
void close_r_df() {
   if ( OKflag == false ) return; // exit if there was an earlier error

   // check that column names not empty
   if ( colnames == "" || colnames == "c(" ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: No column names supplied for ";
      err_msg = err_msg + prevObj[level] ;
      OKflag = false;
                              write_errmsg();
      return;
   }
   // check that if user wanted to write the row.names they called wrt_r_namevector
   if ( rowflag == 2 && rownames == "" ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: No row names supplied for ";
      err_msg = err_msg + prevObj[level];
      OKflag = false;
                              write_errmsg();
      return;
   }

   // write closing punctuation
   rfile << ")," << endl;

   // Write names of vectors
   rfile << ".Names = " << colnames << ")," << endl;

   // If row names are being used write out row names
               if ( rowflag == 0 ) {
                              rfile << "row.names = c(NA, " << ( dim2 - dim1 + 1 ) << ")," << endl;
               }
   else {
      rfile << "row.names = " << rownames << "," << endl;
   }

   // write out rest of object
   rfile << "class = \"data.frame\")" << endl;
   rfile << endl;


   // clear row/col names and dimensions and write flags for next use
   rownames.clear();
   colnames.clear();
   rowflag = 0;
   colflag = 0;
   dim1 = -1;
   dim2 = -1;

   // clear matrix/data frame flag
   mflag.clear();

   // set object complete flag
   ObjDoneFlag[level] = true;

   // if something goes wrong
   if ( rfile.bad() ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Unable to write to " + outfile;
      OKflag = false;
                              write_errmsg();
      return;
   }
} // end close_r_df


//================================================================================
// do_df_col_wrt_vec
//
// Function template for writing a VECTOR as part of an R data frame.
// Called by wrt_r_df_col when ADMB vector types are used.
//
// ARGUMENTS
//    name - the name of the vector to be written as the R column name
//    xx - the ADMB dvector to be written
//    shift - if the vector doesn't have the same index range, shift is the value
//      in the data frame's index coordinates that corresponds to the first
```

```
//      element in vector xx.
//    na_vector - boolean vector indicating which positions in the xx vector
//      should be replaced with the NA missing value indicator. A value of 1 (true)
//      indicates the spot to replace with NA.
//=============================================================================================
template <class T>
void do_df_col_wrt_vec (char* name, const T& xx, int shift, int* na_vector = NULL) {

   // if this is not the first item, print the comma separating the previous item
   if ( colnames != "c(" ) rfile << "," << endl;

   // add item to column names
   add_colname(name);

   // write column to file
   rfile << name << " = c(";

   // get vector's index min and max
   int ja = (xx).indexmin();
   int jz = (xx).indexmax();

   // set difference between vector's coordinate system and data frame's coordinate system
   int yshift = 0;

   // index values not initialized; set to first vector's index specifications
   if ( dim1 == -1 && dim2 == -1 ) {
      dim1 = ja;
      dim2 = jz;
   }

   // re-set vector index coordinates into data frame coordinates
   if ( shift != -999999 ) {
      yshift = ja - shift;
      jz = jz - ja + shift;
      ja = shift;
   }

   // Write the data:
   for ( int y=dim1; y<=dim2; y++ ) {
      if ( y<ja || y>jz ) {        // ( If out of range
         rfile << "NA";
      }     // ( write NA
      else { // if value is the missing value indicator, write "NA" instead
         if ( test_missing(convert<double>(xx[y + yshift])) && writeNA == true ) {
            rfile << "NA";
         } // if the value in the boolean vector is true, write "NA"
         else if (naflag && na_vector[y] == true) {
            rfile << "NA";
         }
         else {
            rfile << xx[y + yshift];
         } // If in range and not a missing datum, write data
      }
      if ( y==dim2 ) {
         rfile << ")";
      }      // ( Write appropriate punctuation
      else {                  // ( to separate or terminate the
         rfile << ", ";
      }     // ( values in the vector.
   }

}; // end do_df_col_wrt_vec


//=============================================================================================
// wrt_r_df_col
//
// Overloaded function to write an ADMB vector types as part of an R data frame.
// Defined here for dvector, ivector, and dvar_vector.
//
// ARGUMENTS
//    name - the name of the vector to be written as the R column name
//    xx - the ADMB dvector to be written
//    shift - if the vector doesn't have the same index range, shift is the value
//      in the data frame's index coordinates that corresponds to the first
//      element in vector xx. (optional)
//    isna - is an NA vector supplied. Optional.
//      false = no vector will be supplied (Default). true = a NA vector will follow.
//    na_vector - (optional) a boolean vector indicating which positions in the xx vector
//      should be replaced with the NA missing value indicator. A value of 1 (true)
```

```
//      indicates the spot to replace with NA. This argument is passed to do_df_col_wrt_vec
//      for further handling.
//=============================================================================================
void wrt_r_df_col(char* name, const dvector& xx, int shift = -999999,
                bool isna = false, int* na_vector = NULL) {
   if ( OKflag == false ) return; // exit if there was an earlier error

   naflag = isna;

   do_df_col_wrt_vec<dvector> (name, xx, shift, na_vector);

} // end wrt_r_df_col (dvector)
//=============================================================================================
void wrt_r_df_col(char* name, const ivector& xx, int shift = -999999,
                bool isna = false, int* na_vector = NULL) {
   if ( OKflag == false ) return; // exit if there was an earlier error

   naflag = isna;

   do_df_col_wrt_vec<ivector> (name, xx, shift, na_vector);

} // end wrt_r_df_col (ivector)
//=============================================================================================
void wrt_r_df_col(char* name, const dvar_vector& xx, int shift = -999999,
                bool isna = false, int* na_vector = NULL) {
   if ( OKflag == false ) return; // exit if there was an earlier error

   naflag = isna;

   do_df_col_wrt_vec<dvar_vector> (name, xx, shift, na_vector);

} // end wrt_r_df_col (dvar_vector)


//=============================================================================================
// do_df_col_wrt_num
//
// Function template for writing a SERIES of NUMBERS as part of an R data frame.
// Called by wrt_r_df_col when a series of numbers are used.
//
// ARGUMENTS:
//    name - the name of the vector to be written as the R column name
//    start - value to start the numeric series
//    stop - value to end the numeric series
//    inc - the increment between values in the series
//    na_vector - a boolean vector indicating which positions in the series
//      should be replaced with the NA missing value indicator. A value of 1 (true)
//      indicates the spot to replace with NA.
//=============================================================================================
template <class T>
void do_df_col_wrt_num (char* name, const T& start, const T& stop, T inc,
                    int* na_vector = NULL) {

   // index values not initialized
   if ( dim1 == -1 && dim2 == -1 ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Index min and max values unspecified";
      err_msg = err_msg + " in open_r_df for " + prevObj[level];
      OKflag = false;
                              write_errmsg();
      return;
   }

   if ( colnames != "c(" ) rfile << "," << endl;  // Comma needed if vector is not first

   add_colname(name);

   rfile << name << " = c(";    // Column start


   // Write the data:
   T iter;
   iter = start;
   for ( int y=dim1; y<=dim2; y++ ) {
      if ( iter > stop ) {               // If out of range
         rfile << "NA";
      }           // write NA
      else if (naflag && na_vector[y]) {
         rfile << "NA";
      }
```

```
      else if (writeNA && test_missing(iter)) {
         rfile << "NA";
      }
      else {
         rfile << iter;
      }           // write out value to file
      iter = iter + inc;
      if ( y==dim2 ) {
         rfile << ")";
      }            // Write appropriate punctuation
      else {                        // to separate or terminate the
         rfile << ", ";
      }           // values in the vector.
   }
}; // end do_df_col_wrt_num


//=============================================================================
// wrt_r_df_col
//
// Overloaded function to write a series of integers as part of an R data frame.
// Defined here for int.
// (Functions for double and dvariable types are commented out to prevent possible errors.)
//
// ARGUMENTS:
//    name - the name of the vector to be written as the R column name
//    xx - the ADMB dvector to be written
//    start - value to start the numeric series
//    stop - value to end the numeric series
//    inc - the increment between values in the series
//    isna - is an NA vector supplied. Optional.
//      false = no vector will be supplied (Default). true = a NA vector will follow.
//    na_vector - (optional) a boolean vector indicating which positions in the xx vector
//      should be replaced with the NA missing value indicator. A value of 1 (true)
//      indicates the spot to replace with NA. This argument is passed to do_df_col_wrt_vec
//      for further handling.
//=============================================================================
void wrt_r_df_col(char* name, const int& start, const int& stop, int inc = 1,
                  bool isna = false, int* na_vector = NULL) {

   if ( OKflag == false ) return; // exit if there was an earlier error

   naflag = isna;

   do_df_col_wrt_num<int> (name, start, stop, inc, na_vector);

} // end wrt_r_df_col (int)
//=============================================================================
//void wrt_r_df_col(char* name, const double& start, const double& stop, double inc = 1.00,
//                  bool isna = false, int* na_vector = NULL) {
//
//   if ( OKflag == false ) return; // exit if there was an earlier error
//
//   naflag = isna;
//
//   do_df_col_wrt_num<double> (name, start, stop, inc, na_vector);
//
//} // end wrt_r_df_col (double)
//=============================================================================
//void wrt_r_df_col(char* name, const dvariable& start, const dvariable& stop, double inc = 1.00,
//                  bool isna = false, int* na_vector = NULL) {
//
//   if ( OKflag == false ) return; // exit if there was an earlier error
//
//   const double newstart = value(start);
//   const double newstop = value(stop);
//
//   naflag = isna;
//
//   do_df_col_wrt_num<double> (name, newstart, newstop, inc, na_vector);
//
//} // end wrt_r_df_col (dvariable)



//=============================================================================
// open_r_complete_vector
//
// Opens the vector object and does housekeeping tasks
//
```

```
// ARGUMENTS:
//      name - name of vector to write to file.
//=============================================================================================

void open_r_complete_vector (char* name) {
   if ( OKflag == false ) return;    // exit if there was an earlier error

   // add vector object name to list and check for object completion
   int flag = reg_Rnames(name, "Vector Object ");
   if ( flag == 0 ) return;

   rfile << name << " = structure(c(" << endl;
};

//=============================================================================================
// close_r_complete_vector
//
// Closes the vector object and does housekeeping tasks
//
// No arguments.
//=============================================================================================
void close_r_complete_vector() {
   if ( OKflag == false ) return;    // exit if there was an earlier error

   colnames.clear();                    // clear row and column names
   rownames.clear();

   ObjDoneFlag[level] = true;              // set object complete flag

   // if something goes wrong
   if ( rfile.bad() ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Unable to write to " + outfile;
      OKflag = false;
                              write_errmsg();
      return;
   }
} // end close_r_complete_vector

//=============================================================================================
// do_wrt_r_complete_vector
//
// Write a vector subobject to the R data object all in one shot.

//
// Arguments are passed to do_wrt_r_complete_vector by the overloaded wrt_r_complete_vector function.
//
// ARGUMENTS:
//   xvec - the vector
//   name_flag - integer that indicates whether there is a vector of names. A value of 0
//      indicates no vector of names. A value of 1 indicates there is a vector of names.
//   name_vector - an integer vector of names to describe each element in the xvec vector.
//   na_vector - a boolean vector indicating which positions in the xvec vector
//      should be replaced with the NA missing value indicator. A value of 1 (true)
//      indicates the spot to replace with NA.
//=============================================================================================
template <class T>
void do_wrt_r_complete_vector (const T& xvec, int name_flag, const ivector& name_vector,
                                                                                    ivector& na_vector) {

   int ir;                    // row iterator in for statement
   int ra, rz, na, nz;          // for vector bounds
               int i;                     // counter in for loop
               int nelem1, nelem2;          // number of elements in each vector, for bounds checking

   if ( OKflag == false ) return;    // exit if there was an earlier error

               ra = (xvec).indexmin();              // Get starting index value
   rz = (xvec).indexmax();              // Get ending index value
               nelem1 = rz - ra + 1;              // number of elements in xvec

               // get bounds of NA vector, if used
               if (naflag){
                              na = (na_vector).indexmin();         // Get starting index value
      nz = (na_vector).indexmax();         // Get ending index value
                  nelem2 = nz - na + 1;              // number of elements
                              // write error message if number of elements is not the same
                              if ( nelem2 != nelem1 ) {
         if ( rfile.is_open() ) rfile.close();
         err_msg = err_msg + "\n**** ADMB2R Error: Number of vector elements in ";
```

```
        err_msg = err_msg + prevObj[level] + " is different than the NA vector used.";
        OKflag = false;
                                        write_errmsg();
        return;
    }
            }

  // Write the vector data
            i==0;
  for ( ir=ra; ir<=rz; ir++ ) {

    i = i + 1;

                        // if value is the missing value indicator, and we're
    // using a missing value "value",  write "NA" instead
    if ( test_missing(convert<double>(xvec(ir))) && writeNA == true ) {
      rfile << "NA";
    }
    // if instead we're using a vector of booleans to
    // indicate the position of missing values, check
    // to see if this position is a missing value
    else if (naflag && na_vector(na + i - 1) ){
      rfile << "NA";
    }
    // otherwise use the value we're given.
    else {
      rfile << xvec(ir);
    }
    // write proper punctuation
    if ( ir==rz ) {
      rfile << "),";
    } else {
      rfile << ", ";
    }
  }
  rfile << endl;

  //write default names if no name vector is present
            if ( name_flag == 0 ) {
                        rfile << ".Names = NULL)" << endl;
            } else {
                        na = (name_vector).indexmin();          // Get starting index value
                        nz = (name_vector).indexmax();          // Get ending index value
                        nelem2 = nz - na + 1;                   // number of elements

                        // first check to make sure that both vectors have the same dimensions
                        if ( nelem2 != nelem1 ) {
      if ( rfile.is_open() ) rfile.close();
      err_msg = err_msg + "\n**** ADMB2R Error: Number of vector elements in ";
      err_msg = err_msg + prevObj[level] + " is different than the names vector used.";
      OKflag = false;
                                        write_errmsg();
      return;
    }

                        rfile << ".Names = c(";
                        // write out the vector
                        for ( ir=na; ir<=nz; ir++ ) {
                                    rfile << name_vector(ir);
                                    // write proper punctuation
                                    if ( ir==nz ) {
                                                rfile << "))" << endl;
                                    } else {
                                                rfile << ", ";
                                    }
                        }
            }
            rfile << endl;

} // end do_wrt_r_complete_vector

//=========================================================================================
// wrt_r_complete_vector
//
// Overloaded function to write a vector object all in one function call.
// Defined here for types dvar_vector, dvector, and ivector.
//
// ARGUMENTS
//   name - name of vector object to write (e.g., "agevector")
//   xvec - the vector
```

```
//      This argument is passed to do_wrt_r_complete_vector for further handling.
//   namevec - Vector to use to write names of vector items. Optional.
//   isna - is an NA_vector supplied. Optional.
//      false = no vector will be supplied (Default). true = a NA vector will follow.
//   na_vector - (optional) a boolean vector indicating which positions in the xvec vector
//      should be replaced with the NA missing value indicator. A value of 1 (true)
//      indicates the spot to replace with NA. This argument is passed to do_wrt_r_complete_vector
//      for further handling.
//=============================================================================================
void wrt_r_complete_vector(char* name, const dvar_vector& xvec,
                bool isna = false, ivector& na_vector = dum_vector) {

   if ( OKflag == false ) return;    // exit if there was an earlier error

                open_r_complete_vector(name);

                // Set global flags
   naflag = isna;

   do_wrt_r_complete_vector<dvar_vector> (xvec, 0, dum_vector, na_vector);

                close_r_complete_vector();

} // wrt_r_complete_vector (dvar_vector)
//=============================================================================================
void wrt_r_complete_vector(char* name, const dvector& xvec,
                bool isna = false, ivector& na_vector = dum_vector) {

   if ( OKflag == false ) return;    // exit if there was an earlier error

                open_r_complete_vector(name);

                // Set global flags
   naflag = isna;

   do_wrt_r_complete_vector<dvector> (xvec, 0, dum_vector, na_vector);

                close_r_complete_vector();

} // wrt_r_complete_vector (dvector)
//=============================================================================================
void wrt_r_complete_vector(char* name, const ivector& xvec,
                bool isna = false, ivector& na_vector = dum_vector) {

   if ( OKflag == false ) return;    // exit if there was an earlier error

                open_r_complete_vector(name);

                // Set global flags
   naflag = isna;

   do_wrt_r_complete_vector<ivector> (xvec, 0, dum_vector, na_vector);

                close_r_complete_vector();

} // end wrt_r_complete_vector (ivector)
//=============================================================================================
void wrt_r_complete_vector(char* name, const dvar_vector& xvec, const ivector& namevec,
                bool isna = false, ivector& na_vector = dum_vector) {

   if ( OKflag == false ) return;    // exit if there was an earlier error

                open_r_complete_vector(name);

                // Set global flags
   naflag = isna;

   do_wrt_r_complete_vector<dvar_vector> (xvec, 1, namevec, na_vector);

                close_r_complete_vector();

} // wrt_r_complete_vector (dvar_vector, ivector)
//=============================================================================================
void wrt_r_complete_vector(char* name, const dvector& xvec, const ivector& namevec,
                bool isna = false, ivector& na_vector = dum_vector) {

   if ( OKflag == false ) return;    // exit if there was an earlier error

                open_r_complete_vector(name);
```

```
                // Set global flags
   naflag = isna;

   do_wrt_r_complete_vector<dvector> (xvec, 1, namevec, na_vector);

                close_r_complete_vector();

} // wrt_r_complete_vector (dvector, ivector)
//=============================================================================================
void wrt_r_complete_vector(char* name, const ivector& xvec, const ivector& namevec,
             bool isna = false, ivector& na_vector = dum_vector) {

   if ( OKflag == false ) return;    // exit if there was an earlier error

                open_r_complete_vector(name);

                // Set global flags
   naflag = isna;

   do_wrt_r_complete_vector<ivector> (xvec, 1, namevec, na_vector);

                close_r_complete_vector();

} // end wrt_r_complete_vector (ivector, ivector)
//=============================================================================================
// End File admb2r.cpp
//=============================================================================================
```