S E D A R

Southeast Data, Assessment, and Review

==========

SEDAR-17-AW-12

AD Model Builder code to implement catch-age assessment model of Spanish mackerel

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
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 smb1.tpl (see Globals Section and the very last line of code).

I. AD Model Builder code (smb1.tpl)

```
//SMB1.tpl
//Base run for SEDAR 17, Spanish mackerel
DATA_SECTION
//Create ascii file for output
//!!CLASS ofstream report1("rsresults.rep",ios::out);  //create file for output

!!cout << "Starting Spanish Mackerel 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;

//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
init_int nlenbins;

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

//discard mortality constants
init_number set_Dmort_HL;
init_number set_Dmort_GN;
init_number set_Dmort_MRFSS;

//initial proportion female
init_number set_prop_f_a0;

//Total number of iterations for spr calcs
init_int n_iter_spr;
//Total number of iterations for msy calcs
init_int n_iter_msy;
//starting index of ages for exploitation rate: if model has age-0s, ages of E are (value-1) to oldest
init_int set_E_age_st;
//bias correction (set to 1.0 for no bias correction or 0.0 to compute from rec variance)
init_number set_BiasCor;
// Von Bert parameters for males and females, respectively
```

```
init_number set_Linf_m;
init_number set_K_m;
init_number set_t0_m;
init_number set_Linf_f;
init_number set_K_f;
init_number set_t0_f;
//CV of length at age
init_number set_len_cv;

//length(mm)-weight(whole weigt in g) relationship: W=aL^b
init_number wgtpar_a;
init_number wgtpar_b;
//weight-weight relationship:whole weight to gutted weight -- gutted=a*whole
//init_number wgtpar_whole2gutted  NOT CURRENTLY USED

//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
//#########################################################################
//############################Commercial Hand Lines fishery ################################
//CPUE
//FL trip ticket
init_int styr_FL_HL_cpue;
init_int endyr_FL_HL_cpue;
init_vector obs_FL_HL_cpue(styr_FL_HL_cpue,endyr_FL_HL_cpue);//Observed CPUE
init_vector FL_HL_cpue_cv(styr_FL_HL_cpue,endyr_FL_HL_cpue); //CV of cpue
//Logbook HL (north of FL)
init_int styr_LB_HL_cpue;
init_int endyr_LB_HL_cpue;
init_vector obs_LB_HL_cpue(styr_LB_HL_cpue,endyr_LB_HL_cpue);//Observed CPUE
init_vector LB_HL_cpue_cv(styr_LB_HL_cpue,endyr_LB_HL_cpue); //CV of cpue
// Landings (1000 lb whole weight)
init_int styr_HL_L;
init_int endyr_HL_L;
init_vector obs_HL_L(styr_HL_L,endyr_HL_L);
init_vector HL_L_cv(styr_HL_L,endyr_HL_L);    //vector of CV of landings by year
// Discards (1000s)
init_int styr_HL_D;
init_int endyr_HL_D;
init_vector obs_HL_released(styr_HL_D,endyr_HL_D); //vector of observed releases by year, multiplied by discard mortality for fitting
init_vector HL_D_cv(styr_HL_D,endyr_HL_D);    //vector of CV of discards by year
// Length Compositions (1cm bins)
init_int nyr_HL_lenc;
init_ivector yrs_HL_lenc(1,nyr_HL_lenc);
init_vector nsamp_HL_lenc(1,nyr_HL_lenc);
init_matrix obs_HL_lenc(1,nyr_HL_lenc,1,nlenbins);
// Age Compositions
init_int nyr_HL_agec;
init_ivector yrs_HL_agec(1,nyr_HL_agec);
init_vector nsamp_HL_agec(1,nyr_HL_agec);
init_matrix obs_HL_agec(1,nyr_HL_agec,1,nages);
//#########################################################################
//################################Poundnet fishery ######################################
// Landings (1000 lb whole weight)
init_int styr_PN_L;
init_int endyr_PN_L;
init_vector obs_PN_L(styr_PN_L,endyr_PN_L);
init_vector PN_L_cv(styr_PN_L,endyr_PN_L);
// Length Compositions (1cm bins)
init_int styr_PN_lenc;
init_int endyr_PN_lenc;
init_vector nsamp_PN_lenc(styr_PN_lenc,endyr_PN_lenc);
init_matrix obs_PN_lenc(styr_PN_lenc,endyr_PN_lenc,1,nlenbins);
// Age compositions
init_int nyr_PN_agec;
init_ivector yrs_PN_agec(1,nyr_PN_agec);
init_vector nsamp_PN_agec(1,nyr_PN_agec);
init_matrix obs_PN_agec(1,nyr_PN_agec,1,nages);
//#########################################################################
//####################Commercial Gillnet fishery ##########################
//CPUE
//FL trip ticket before gillnet ban
init_int styr_FL_gill1_cpue;
init_int endyr_FL_gill1_cpue;
init_vector obs_FL_gill1_cpue(styr_FL_gill1_cpue,endyr_FL_gill1_cpue);//Observed CPUE
init_vector FL_gill1_cpue_cv(styr_FL_gill1_cpue,endyr_FL_gill1_cpue); //CV of cpue

//FL trip ticket after gillnet ban
init_int styr_FL_gill2_cpue;
init_int endyr_FL_gill2_cpue;
```

```
init_vector obs_FL_gill2_cpue(styr_FL_gill2_cpue,endyr_FL_gill2_cpue);//Observed CPUE
init_vector FL_gill2_cpue_cv(styr_FL_gill2_cpue,endyr_FL_gill2_cpue); //CV of cpue

//Comm logbook for all points north of FL
init_int styr_LB_gill_cpue;
init_int endyr_LB_gill_cpue;
init_vector obs_LB_gill_cpue(styr_LB_gill_cpue,endyr_LB_gill_cpue);//Observed CPUE
init_vector LB_gill_cpue_cv(styr_LB_gill_cpue,endyr_LB_gill_cpue); //CV of cpue

// Landings - (1000 lb whole weight)
init_int styr_GN_L;
init_int endyr_GN_L;
init_vector obs_GN_L(styr_GN_L,endyr_GN_L); //vector of observed landings by year
init_vector GN_L_cv(styr_GN_L,endyr_GN_L);    //vector of CV of landings by year

// Discards (1000s)
init_int styr_GN_D;
init_int endyr_GN_D;
init_vector obs_GN_released(styr_GN_D,endyr_GN_D); //vector of observed releases by year, multiplied by discard mortality for fitting
init_vector GN_D_cv(styr_GN_D,endyr_GN_D);    //vector of CV of discards by year

// Length Compositions (1cm bins)
init_int styr_GN_lenc;
init_int endyr_GN_lenc;
init_vector nsamp_GN_lenc(styr_GN_lenc,endyr_GN_lenc);
init_matrix obs_GN_lenc(styr_GN_lenc,endyr_GN_lenc,1,nlenbins);

// Age Compositions
init_int styr_GN_agec;
init_int endyr_GN_agec;
init_vector nsamp_GN_agec(styr_GN_agec,endyr_GN_agec);
init_matrix obs_GN_agec(styr_GN_agec,endyr_GN_agec,1,nages);
//#########################################################################
//###############################Castnet fishery ##########################################
//CPUE
init_int styr_CN_cpue;
init_int endyr_CN_cpue;
init_vector obs_CN_cpue(styr_CN_cpue,endyr_CN_cpue);//Observed CPUE
init_vector CN_cpue_cv(styr_CN_cpue,endyr_CN_cpue); //CV of cpue
// Landings (1000 lb whole weight)
init_int styr_CN_L;
init_int endyr_CN_L;
init_vector obs_CN_L(styr_CN_L,endyr_CN_L);
init_vector CN_L_cv(styr_CN_L,endyr_CN_L);
// Length Compositions (1cm bins)
init_int nyr_CN_lenc;
init_ivector yrs_CN_lenc(1,nyr_CN_lenc);
init_vector nsamp_CN_lenc(1,nyr_CN_lenc);
init_matrix obs_CN_lenc(1,nyr_CN_lenc,1,nlenbins);
// Age compositions
init_int nyr_CN_agec;
init_ivector yrs_CN_agec(1,nyr_CN_agec);
init_vector nsamp_CN_agec(1,nyr_CN_agec);
init_matrix obs_CN_agec(1,nyr_CN_agec,1,nages);
//#########################################################################
//############################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 of fish)
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 (1cm 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);
//#########################################################################
//############################Shrimp Bycatch ###################################
// Bycatch (1000s of fish)
init_int styr_shrimp_B;
init_int endyr_shrimp_B;
init_vector obs_shrimp_B(styr_shrimp_B,endyr_shrimp_B);
init_vector shrimp_B_cv(styr_shrimp_B,endyr_shrimp_B);
//#########################################################################
//############################SEAMAP###################################
//YOY CPUE
init_int styr_SMAP_YOY_cpue;
init_int endyr_SMAP_YOY_cpue;
init_vector obs_SMAP_YOY_cpue(styr_SMAP_YOY_cpue,endyr_SMAP_YOY_cpue);//Observed CPUE
init_vector SMAP_YOY_cpue_cv(styr_SMAP_YOY_cpue,endyr_SMAP_YOY_cpue); //CV of cpue
//1-yr-old CPUE
init_int styr_SMAP_1YR_cpue;
init_int endyr_SMAP_1YR_cpue;
init_vector obs_SMAP_1YR_cpue(styr_SMAP_1YR_cpue,endyr_SMAP_1YR_cpue);//Observed CPUE
init_vector SMAP_1YR_cpue_cv(styr_SMAP_1YR_cpue,endyr_SMAP_1YR_cpue); //CV of cpue
//#########################################################################
//#################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_LB_HL;
init_number set_w_I_FL_HL;
init_number set_w_I_FL_gill1;
init_number set_w_I_FL_gill2;
init_number set_w_I_LB_gill;
init_number set_w_I_CN;
init_number set_w_I_MRFSS;
init_number set_w_I_SMAP_YOY;
init_number set_w_I_SMAP_1YR;
init_number set_w_R;
init_number set_w_R_init;
init_number set_w_R_end;
init_number set_w_F;
init_number set_w_B1dB0;        // weight on B1/B0
init_number set_w_fullF;        //penalty for any fullF>5
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 guesses or fixed values
init_number set_steep;
//init_number set_M;
init_vector set_M(1,nages); //age-dependent: used in model
init_number set_M_constant; //age-independent: used only for MSST

//--index catchability-------------------------------------------------------------------------------------------------------------

init_number set_logq_FL_HL;         //catchability coefficient (log) for FL hand lines
init_number set_logq_LB_HL;
init_number set_logq_FL_gill1;      //catchability coefficient (log) for FL gillnet 1985-1994
init_number set_logq_FL_gill2;      //catchability coefficient (log) for FL gillnet 1996-2007
init_number set_logq_LB_gill;       //catchability coefficient (log) for GA-NY logbook gillnet
init_number set_logq_CN;
init_number set_logq_MRFSS;
init_number set_logq_SMAP_YOY;
init_number set_logq_SMAP_1YR;

//--F's---------------------------------
init_number set_F_hist;

init_number set_log_avg_F_HL;  //hand lines
init_vector set_log_F_dev_HL(styr_HL_L,endyr_HL_L);
init_number set_log_avg_F_HL_D;  //hand lines
init_vector set_log_F_dev_HL_D(styr_HL_D,endyr_HL_D);
init_number set_log_avg_F_PN;  //pound nets
init_vector set_log_F_dev_PN(styr_PN_L,endyr_PN_L);
init_number set_log_avg_F_GN;  //hand lines
init_vector set_log_F_dev_GN(styr_GN_L,endyr_GN_L);
init_number set_log_avg_F_GN_D;  //hand lines
init_vector set_log_F_dev_GN_D(styr_GN_D,endyr_GN_D);
init_number set_log_avg_F_CN;  //cast nets
init_vector set_log_F_dev_CN(styr_CN_L,endyr_CN_L);
init_number set_log_avg_F_MRFSS;  //mrfss
init_vector set_log_F_dev_MRFSS(styr_MRFSS_L,endyr_MRFSS_L);
init_number set_log_avg_F_MRFSS_D;  //mrfss
```

```
init_vector set_log_F_dev_MRFSS_D(styr_MRFSS_D,endyr_MRFSS_D);
init_number set_log_avg_F_shrimp;  //shrimp
init_vector set_log_F_dev_shrimp(styr_shrimp_B,endyr_shrimp_B);

//Set some more initial guesses of estimated parameters
init_number set_log_R0;
init_number set_R_autocorr;

//Initial guesses of estimated selectivity parameters
init_number set_selpar_L50_HL_keep;
init_number set_selpar_slope_HL;
init_number set_selpar_L50_PN;
init_number set_selpar_slope_PN;
init_number set_selpar_L50_GN_keep;
init_number set_selpar_slope_GN;
init_number set_selpar_L50_GN2;  //dome shaped for gillnet after net ban
init_number set_selpar_slope_GN2;
init_number set_selpar_L502_GN2;
init_number set_selpar_slope2_GN2;
init_number set_selpar_L50_CN;
init_number set_selpar_slope_CN;
init_number set_selpar_L50_MRFSS_keep;
init_number set_selpar_slope_MRFSS;
init_number set_selpar_age0_MRFSS;
init_number set_selpar_age1_MRFSS;
init_number set_selpar_age2_MRFSS;
init_number set_selpar_age3_MRFSS;

init_vector set_sel_historical_F(1,nages);
init_vector set_sel_HL_D_F(1,nages);  //selectivity vectors for female, male discards input directly
init_vector set_sel_HL_D_M(1,nages);
init_vector set_sel_GN_D_F(1,nages);  //selectivity vectors for female, male discards input directly
init_vector set_sel_GN_D_M(1,nages);
init_vector set_sel_MRFSS_D_F(1,nages);  //selectivity vectors for female, male discards input directly
init_vector set_sel_MRFSS_D_M(1,nages);
init_vector set_sel_shrimp_B(1,nages); //shrimp bycatch selectivity input directly

init_number set_L50_diff; //difference between males and females

//aging error matrix
init_matrix set_age_error_matrix(1,nages,1,nages);

//historic recreational landings multiplier
init_number L_rec_multiply;

// #######Indices for year(iyear), age(iage),length(ilen) ###############
int iyear;
int iyear2;
int iage;
int ilen;
int E_age_st;   //starting age for exploitation rate: (value-1) to oldest

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
  number Linf_m;
  number K_m;
  number t0_m;
  number Linf_f;
  number K_f;
  number t0_f;
  vector wgt_g_f(1,nages);       //whole wgt in g - females
  vector wgt_kg_f(1,nages);      //whole wgt in kg
  vector wgt_f(1,nages);         //whole wgt in mt
  vector wgt_klb_f(1,nages);     //whole wgt in 1000 lb
```

```
  vector wgt_g_m(1,nages);       //whole wgt in g - males
  vector wgt_kg_m(1,nages);      //whole wgt in kg
  vector wgt_m(1,nages);         //whole wgt in mt
  vector wgt_klb_m(1,nages);     //whole wgt in 1000 lb
  vector meanlen_m(1,nages);     //mean length at age -males
  vector meanlen_f(1,nages);     //mean length at age -females
  number sqrt2pi;
  number g2mt;                   //conversion of grams to metric tons
  number g2kg;                   //conversion of grams to kg
  number mt2klb;                 //conversion of metric tons to 1000 lb

  number nlenbins2;
  matrix lenprob_m(1,nages,1,nlenbins);          //distn of size at age (age-length key, 1cm bins) - males
  matrix lenprob_f(1,nages,1,nlenbins);          //distn of size at age (age-length key, 1cm bins) - females
  matrix lenprob_m2(1,nages,1,nlenbins+20);          //distn of size at age (age-length key, 1cm bins) - males
  matrix lenprob_f2(1,nages,1,nlenbins+20);          //distn of size at age (age-length key, 1cm bins) - females
  vector lenbins2(1,nlenbins+20);

  init_bounded_number log_len_cv(-5,-0.3,3);
  //init_bounded_number log_len_cv(-4.6,-0.7,2) //cv expressed in log-space, bounds correspond to 0.01, 0.5
  //init_bounded_dev_vector log_len_cv_dev(1,nages,-2,2,3)
  vector len_cv(1,nages);

//----Predicted length and age compositions
  number prop_f_a0; //proportion female at age 0

  matrix pred_HL_lenc(1,nyr_HL_lenc,1,nlenbins);
  matrix pred_PN_lenc(styr_PN_lenc,endyr_PN_lenc,1,nlenbins);
  matrix pred_GN_lenc(styr_GN_lenc,endyr_GN_lenc,1,nlenbins);
  matrix pred_CN_lenc(1,nyr_CN_lenc,1,nlenbins);
  matrix pred_MRFSS_lenc(styr_MRFSS_lenc,endyr_MRFSS_lenc,1,nlenbins);

  matrix pred_HL_agec(1,nyr_HL_agec,1,nages);
  matrix pred_PN_agec(1,nyr_PN_agec,1,nages);
  matrix pred_GN_agec(styr_GN_agec,endyr_GN_agec,1,nages);
  matrix pred_CN_agec(1,nyr_CN_agec,1,nages);
  matrix pred_MRFSS_agec(styr_MRFSS_agec,endyr_MRFSS_agec,1,nages);

  //nsamp_X_allyr vectors used only for R output of comps with nonconsecutive yrs
  vector nsamp_HL_lenc_allyr(styr,endyr);
  vector nsamp_HL_agec_allyr(styr,endyr);
  vector nsamp_PN_agec_allyr(styr,endyr);
  vector nsamp_CN_lenc_allyr(styr,endyr);
  vector nsamp_CN_agec_allyr(styr,endyr);

//----Aging error
  matrix age_error_matrix(1,nages,1,nages);

//-----Population----------------------------------------------------------------------------------
  matrix N_F(styr,endyr+1,1,nages);          //Population numbers for females by year and age at start of yr
  matrix N_M(styr,endyr+1,1,nages);          //Population numbers for males by year and age at start of yr
  matrix N_F_mdyr(styr,endyr+1,1,nages);       //Population numbers by year and age at mdpt of yr: used for comps and SSB
  matrix N_M_mdyr(styr,endyr+1,1,nages);
  matrix B(styr,endyr+1,1,nages);            //Biomass by year and age - sexes combined
  vector totB(styr,endyr+1);                 //Total biomass by year
  sdreport_vector SSB(styr,endyr);       //Spawning biomass by year
  number SSB_extra;
  sdreport_vector rec(styr,endyr+1);       //Recruits by year
  matrix prop_f(styr,endyr+1,1,nages);       //Proportion female by year and age
  vector prop_f_F0(1,nages);               //proportion of females in unexploited pop
  vector maturity_f(1,nages);              //Proportion of female mature at age
  vector reprod(1,nages);                  //recruitment calcs

//---Stock-Recruit Function (Beverton-Holt, steepness parameterization)----------
  init_bounded_number log_R0(5,20,1);       //log(virgin Recruitment)
  //number log_R0;
  sdreport_number R0;
  init_bounded_number steep(0.25,0.9,3);     //steepness
  //number steep;  //uncomment to fix steepness, comment line directly above
  init_bounded_dev_vector log_dev_N_rec(styr_rec_dev,endyr,-3,3,3); //log recruitment deviations
  //vector log_dev_N_rec(styr_rec_dev,endyr);
  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(0,1.0,3);  //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 - for males = 0
```

```
  number B0;                        //equal to bpr_F0*R0 = virgin Biomass (combined sex)
  number R1;                         //Recruits in styr

  sdreport_number S1S0;                //SSB(styr) / virgin SSB
  sdreport_number popstatus;            //SSB(endyr) / virgin SSB

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

  number L50_diff; //shift in selectivity b/w males and females (same for all gears)
  vector sel_historical_F(1,nages);

   //Commercial handline
  vector sel_HL_keep_M(1,nages);  //time invariant - fish that are kept - males
  vector sel_HL_keep_F(1,nages);  //time invariant - fish that are kept - females
  vector sel_HL_D_M(1,nages);  //selectivity for discards - males
  vector sel_HL_D_F(1,nages);  //selectivity for discards - females
  init_bounded_number selpar_slope_HL(0.5,20.0,3);
  init_bounded_number selpar_L50_HL_keep(0.1,10.,2);
  //number selpar_slope_HL;
  //number selpar_L50_HL_keep;

  //Poundnets
  vector sel_PN_M(1,nages);  //time invariant - males
  vector sel_PN_F(1,nages);  //females
  init_bounded_number selpar_slope_PN(0.5,20.0,3);
  init_bounded_number selpar_L50_PN(-1.0,10.,2);

  //Commercial gillnet
  vector sel_GN_keep_M(1,nages);  //pre-1995 - fish that are kept - males
  vector sel_GN_keep_F(1,nages);  //pre-1995 - fish that are kept - females
  vector sel_GN_keep_M2(1,nages);  //pre-1995 - fish that are kept - males
  vector sel_GN_keep_F2(1,nages);  //pre-1995 - fish that are kept - females
  vector sel_GN_D_M(1,nages);  //selectivity for discards - males
  vector sel_GN_D_F(1,nages);  //selectivity for discards - females
  init_bounded_number selpar_slope_GN(0.5,20.0,3);
  init_bounded_number selpar_L50_GN_keep(0.1,10.,2);
  init_bounded_number selpar_slope_GN2(0.1,9.0,1);
  init_bounded_number selpar_L50_GN2(0.1,10.,2);
  init_bounded_number selpar_slope2_GN2(0.1,9.0,1);
  init_bounded_number selpar_L502_GN2(0.1,10.,2);

  //number selpar_slope_GN;
  //number selpar_L50_GN_keep;

  //Castnets
  vector sel_CN_M(1,nages);  //time invariant - males
  vector sel_CN_F(1,nages);  //females
  init_bounded_number selpar_slope_CN(0.5,20.0,3);
  init_bounded_number selpar_L50_CN(0.1,10.,2);

  //MRFSS
  vector sel_MRFSS_keep_M(1,nages);  //time invariant - fish that are kept - males
  vector sel_MRFSS_keep_F(1,nages);  //time invariant - fish that are kept - females
  vector sel_MRFSS_D_M(1,nages);  //selectivity for discards - males
  vector sel_MRFSS_D_F(1,nages);  //selectivity for discards - females
  //init_bounded_number selpar_slope_MRFSS(0.5,20.0,3);
  //init_bounded_number selpar_L50_MRFSS_keep(0.1,10.,2);
  number selpar_slope_MRFSS;
  number selpar_L50_MRFSS_keep;
  init_bounded_number selpar_age0_MRFSS(0.001,1.0,2);
  number selpar_age1_MRFSS;
  init_bounded_number selpar_age2_MRFSS(0.001,1.0,3);
  init_bounded_number selpar_age3_MRFSS(0.001,1.0,3);

  //shrimp
  vector sel_shrimp(1,nages);

  //effort-weighted, recent selectivities
  vector sel_wgted_L_F(1,nages);  //toward landings,females
  vector sel_wgted_D_F(1,nages);  //toward discards
  vector sel_wgted_tot_F(1,nages);//toward Z, landings plus dead discards
  number max_sel_wgted_tot_F;
  vector sel_wgted_L_M(1,nages);  //toward landings,males
  vector sel_wgted_D_M(1,nages);  //toward discards
  vector sel_wgted_tot_M(1,nages);//toward Z, landings plus dead discards
  number max_sel_wgted_tot_M;

//-------CPUE Predictions---------------------------------
  vector pred_LB_HL_cpue(styr_LB_HL_cpue,endyr_LB_HL_cpue);          //predicted logbook handline index (pounds/trip)
  matrix N_LB_HL(styr_LB_HL_cpue,endyr_LB_HL_cpue,1,nages);          //used to compute above index
```

```
  vector pred_FL_HL_cpue(styr_FL_HL_cpue,endyr_FL_HL_cpue);            //predicted FL handline index (pounds/trip)
  matrix N_FL_HL(styr_FL_HL_cpue,endyr_FL_HL_cpue,1,nages);            //used to compute above index
  vector pred_FL_gill1_cpue(styr_FL_gill1_cpue,endyr_FL_gill1_cpue);   //predicted FL trip ticket gillnet index prior to net ban (pounds/trip)
  matrix N_FL_gill1(styr_FL_gill1_cpue,endyr_FL_gill1_cpue,1,nages);   //used to compute above index
  vector pred_FL_gill2_cpue(styr_FL_gill2_cpue,endyr_FL_gill2_cpue);   //predicted FL trip ticket gillnet index after net ban (pounds/trip)
  matrix N_FL_gill2(styr_FL_gill2_cpue,endyr_FL_gill2_cpue,1,nages);   //used to compute above index
  vector pred_LB_gill_cpue(styr_LB_gill_cpue,endyr_LB_gill_cpue);      //predicted logbook gillnet index north of FL (pounds/(net*area*hour))
  matrix N_LB_gill(styr_LB_gill_cpue,endyr_LB_gill_cpue,1,nages);      //used to compute above index
  vector pred_CN_cpue(styr_CN_cpue,endyr_CN_cpue);                     //predicted FL trip ticket castnet index (pounds/trip)
  matrix N_CN(styr_CN_cpue,endyr_CN_cpue,1,nages);                     //used to compute above index
  vector pred_MRFSS_cpue(styr_MRFSS_cpue,endyr_MRFSS_cpue);            //predicted MRFSS index (number per angler-trip)
  matrix N_MRFSS(styr_MRFSS_cpue,endyr_MRFSS_cpue,1,nages);            //used to compute above index
  vector pred_SMAP_YOY_cpue(styr_SMAP_YOY_cpue,endyr_SMAP_YOY_cpue);   //predicted SMAP_YOY index (number per tow)
  vector pred_SMAP_1YR_cpue(styr_SMAP_1YR_cpue,endyr_SMAP_1YR_cpue);   //predicted SMAP_1YR index (number per tow)

 //---Catchability (CPUE q's)-------------------------------------------------------------
  init_bounded_number log_q_LB_HL(-20,-5,-1);
  init_bounded_number log_q_FL_gill1(-20,-5,-1);
  init_bounded_number log_q_FL_gill2(-20,-5,-1);
  init_bounded_number log_q_LB_gill(-20,-5,-1);
  init_bounded_number log_q_FL_HL(-20,-5,1);
  init_bounded_number log_q_CN(-20,-5,-1);
  init_bounded_number log_q_MRFSS(-20,-5,-1);
  init_bounded_number log_q_SMAP_YOY(-20,-5,1);
  init_bounded_number log_q_SMAP_1YR(-20,-5,-1);


  //init_bounded_number q_rate(-0.1,0.1,-3);
  number q_rate;

 //---Landings Bias-----------------------------------------------------------------
  //init_bounded_number L_early_bias(0.1,10.0,3);
  //number L_early_bias;
  //number L_commHAL_bias;

 //---C is landings in (numbers), L is landings in wgt (mt)-------------------------------------------------

  matrix C_HL_M(styr,endyr,1,nages);              //landings (numbers) at age
  matrix C_HL_F(styr,endyr,1,nages);              //landings (numbers) at age
  matrix L_HL(styr,endyr,1,nages);               //landings (mt) at age
  matrix L_HL_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_HL_L(styr_HL_L,endyr_HL_L);  //yearly landings (klb) summed over ages

  matrix C_PN_M(styr,endyr,1,nages);              //landings (numbers) at age
  matrix C_PN_F(styr,endyr,1,nages);              //landings (numbers) at age
  matrix L_PN(styr,endyr,1,nages);               //landings (mt) at age
  matrix L_PN_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_PN_L(styr_PN_L,endyr_PN_L);             //yearly landings (klb) summed over ages

  matrix C_GN_M(styr,endyr,1,nages);              //landings (numbers) at age
  matrix C_GN_F(styr,endyr,1,nages);              //landings (numbers) at age
  matrix L_GN(styr,endyr,1,nages);               //landings (mt) at age
  matrix L_GN_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_GN_L(styr_GN_L,endyr_GN_L);  //yearly landings (klb) summed over ages

  matrix C_total(styr,endyr,1,nages);             //catch in number
  matrix L_total(styr,endyr,1,nages);             //landings in klb
  vector L_total_yr(styr,endyr);                 //total landings (klb) by yr summed over ages
  matrix D_total(styr,endyr,1,nages);             //discards in klb
  vector D_total_yr(styr,endyr);                 //total discards (klb) by yr summed over ages
  matrix B_total(styr,endyr,1,nages);             //bycatch in klb
  vector B_total_yr(styr,endyr);                 //total bycatch (klb) by yr summed over ages

  matrix C_CN_M(styr,endyr,1,nages);              //landings (numbers) at age
  matrix C_CN_F(styr,endyr,1,nages);              //landings (numbers) at age
  matrix L_CN(styr,endyr,1,nages);               //landings (mt) at age
  matrix L_CN_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_CN_L(styr_CN_L,endyr_CN_L);  //yearly landings (klb) summed over ages

  matrix C_MRFSS_M(styr,endyr,1,nages);              //landings (numbers) at age
  matrix C_MRFSS_F(styr,endyr,1,nages);              //landings (numbers) at age
  matrix L_MRFSS(styr,endyr,1,nages);               //landings (mt) at age
  matrix L_MRFSS_klb(styr,endyr,1,nages);           //landings (1000 lb whole weight) at age
  vector pred_MRFSS_L(styr_MRFSS_L,endyr_MRFSS_L);     //yearly landings (numbers in 1000's) summed over ages

 //---Discards (number dead fish) ---------------------------------------------------

  matrix C_HL_D_M(styr_HL_D,endyr_HL_D,1,nages);   //discards (numbers) at age
  matrix C_HL_D_F(styr_HL_D,endyr_HL_D,1,nages);   //discards (numbers) at age
  matrix L_HL_D(styr,endyr,1,nages);          //discards in mt whole weight
```

```
  vector pred_HL_D(styr_HL_D,endyr_HL_D);      //yearly discards summed over ages (lb)
  vector obs_HL_D(styr_HL_D,endyr_HL_D);      //observed releases multiplied by discard mortality (lb)

  matrix C_GN_D_M(styr_GN_D,endyr_GN_D,1,nages);   //discards (numbers) at age
  matrix C_GN_D_F(styr_GN_D,endyr_GN_D,1,nages);   //discards (numbers) at age
  matrix L_GN_D(styr,endyr,1,nages);        //discards in mt whole weight
  vector pred_GN_D(styr_GN_D,endyr_GN_D);      //yearly discards summed over ages (lb)
  vector obs_GN_D(styr_GN_D,endyr_GN_D);      //observed releases multiplied by discard mortality

  matrix C_MRFSS_D_M(styr_MRFSS_D,endyr_MRFSS_D,1,nages);     //discards (numbers) at age
  matrix C_MRFSS_D_F(styr_MRFSS_D,endyr_MRFSS_D,1,nages);     //discards (numbers) at age
  matrix L_MRFSS_D(styr,endyr,1,nages);        //discards in mt whole weight
  vector pred_MRFSS_D(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

//---Bycatch  ------------------------------------------------------------------
  matrix C_shrimp_M(styr_shrimp_B,endyr_shrimp_B,1,nages);
  matrix C_shrimp_F(styr_shrimp_B,endyr_shrimp_B,1,nages);
  matrix L_shrimp(styr,endyr,1,nages);        //bycatch in mt whole weight
  vector pred_shrimp_B(styr_shrimp_B,endyr_shrimp_B);

//---MSY calcs-----------------------------------------------------------------------------

  number F_HL_prop;       //proportion of F_full attributable to hand lines, last three yrs
  number F_PN_prop;
  number F_GN_prop;
  number F_CN_prop;
  number F_MRFSS_prop;
  number F_HL_D_prop;
  number F_GN_D_prop;
  number F_MRFSS_D_prop;
  number F_shrimp_prop;
  number F_temp_sum;      //sum of geom mean full Fs in last yrs, used to compute F_fishery_prop

  number SSB_msy_out;           //SSB at msy
  number F_msy_out;            //F at msy
  number msy_out;              //max sustainable yield
  number B_msy_out;            //total biomass at MSY
  number E_msy_out;            //exploitation rate at MSY (ages E_age_st plus)
  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_F(1,nages);         //numbers at age for MSY calculations: beginning of yr - Females
  vector N_age_msy_mdyr_F(1,nages);    //numbers at age for MSY calculations: mdpt of yr
  vector C_age_msy_F(1,nages);         //catch at age for MSY calculations
  vector Z_age_msy_F(1,nages);         //total mortality at age for MSY calculations
  vector D_age_msy_F(1,nages);         //discard mortality (dead discards) at age for MSY calculations
  vector F_L_age_msy_F(1,nages);       //fishing mortality (landings, not discards) at age for MSY calculations
  vector F_D_age_msy_F(1,nages);
  vector N_age_msy_M(1,nages);         //numbers at age for MSY calculations: beginning of yr - Males
  vector N_age_msy_mdyr_M(1,nages);    //numbers at age for MSY calculations: mdpt of yr
  vector C_age_msy_M(1,nages);         //catch at age for MSY calculations
  vector Z_age_msy_M(1,nages);         //total mortality at age for MSY calculations
  vector D_age_msy_M(1,nages);         //discard mortality (dead discards) at age for MSY calculations
  vector F_L_age_msy_M(1,nages);       //fishing mortality (landings, not discards) at age for MSY calculations
  vector F_D_age_msy_M(1,nages);
  vector F_msy(1,n_iter_msy);        //values of full F to be used in per-recruit and 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(1,n_iter_msy);     //equilibrium landings(mt) 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 E_eq(1,n_iter_msy);     //equilibrium exploitation rates 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 EdE_msy(styr,endyr);
  vector SdSSB_msy(styr,endyr);
  number SdSSB_msy_end;
  number FdF_msy_end;
  number EdE_msy_end;

//--------Mortality------------------------------------------------------------------
  vector M(1,nages);                    //age-dependent natural mortality
  number M_constant;                    //age-indpendent: used only for MSST
  matrix F_M(styr,endyr,1,nages);          //males
  matrix F_F(styr,endyr,1,nages);          //females

  number F_hist;                    //historical F input into assessment model
```

```
  vector F_F_hist(1,nages);                    //historical F used to get stable age dist in 1st year
  vector Z_hist(1,nages);
  vector fullF(styr,endyr);                  //Fishing mortality rate by year
  vector E(styr,endyr);                     //Exploitation rate by year
  sdreport_vector fullF_sd(styr,endyr);
  sdreport_vector E_sd(styr,endyr);
  matrix Z_M(styr,endyr,1,nages);            //males
  matrix Z_F(styr,endyr,1,nages);           //females

  init_bounded_number log_avg_F_HL(-10,1,1);
  init_bounded_dev_vector log_F_dev_HL(styr_HL_L,endyr_HL_L,-20,5,2);
  matrix F_HL_M(styr,endyr,1,nages);
  matrix F_HL_F(styr,endyr,1,nages);
  vector F_HL_out(styr,endyr); //used for intermediate calculations in fcn get_mortality
  //number log_F_init_commDV;

  init_bounded_number log_avg_F_PN(-10,0,1);
  init_bounded_dev_vector log_F_dev_PN(styr_PN_L,endyr_PN_L,-20,5,2);
  matrix F_PN_M(styr,endyr,1,nages);
  matrix F_PN_F(styr,endyr,1,nages);
  vector F_PN_out(styr,endyr);       //used for intermediate calculations in fcn get_mortality

  init_bounded_number log_avg_F_GN(-10,0,1);
  init_bounded_dev_vector log_F_dev_GN(styr_GN_L,endyr_GN_L,-20,5,2);
  matrix F_GN_M(styr,endyr,1,nages);
  matrix F_GN_F(styr,endyr,1,nages);
  vector F_GN_out(styr,endyr); //used for intermediate calculations in fcn get_mortality

  init_bounded_number log_avg_F_CN(-10,0,1);
  init_bounded_dev_vector log_F_dev_CN(styr_CN_L,endyr_CN_L,-20,5,2);
  matrix F_CN_M(styr,endyr,1,nages);
  matrix F_CN_F(styr,endyr,1,nages);
  vector F_CN_out(styr,endyr);       //used for intermediate calculations in fcn get_mortality

  init_bounded_number log_avg_F_MRFSS(-10,0,1);
  init_bounded_dev_vector log_F_dev_MRFSS(styr_MRFSS_L,endyr_MRFSS_L,-20,5,2);
  matrix F_MRFSS_M(styr,endyr,1,nages);
  matrix F_MRFSS_F(styr,endyr,1,nages);
  vector F_MRFSS_out(styr,endyr); //used for intermediate calculations in fcn get_mortality
  number MRFSS_ratio;  //ratio of F_MRFSS/F_GN for saltwater report years to apply to missing years;
                //straight interpolation used from last angler report to first 'real' year of landings

//--Discard mortality stuff---------------------------------------------------------------------------
  init_bounded_number log_avg_F_HL_D(-10,1,1);
  init_bounded_dev_vector log_F_dev_HL_D(styr_HL_D,endyr_HL_D,-20,5,2);
  matrix F_HL_D_M(styr,endyr,1,nages);
  matrix F_HL_D_F(styr,endyr,1,nages);
  vector F_HL_D_out(styr,endyr);
  number F_HL_D_ratio;  //ratio of average discard F to fishery F, for projection discards back before data

  init_bounded_number log_avg_F_GN_D(-10,0,1);
  init_bounded_dev_vector log_F_dev_GN_D(styr_GN_D,endyr_GN_D,-20,5,2);
  matrix F_GN_D_F(styr,endyr,1,nages);  //females
  matrix F_GN_D_M(styr,endyr,1,nages);  //males
  vector F_GN_D_out(styr,endyr); //used for intermediate calculations in fcn get_mortality
  number F_GN_D_ratio;  //ratio of average discard F to fishery F, for projection discards back before data

  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,-20,5,2);
  matrix F_MRFSS_D_M(styr,endyr,1,nages);
  matrix F_MRFSS_D_F(styr,endyr,1,nages);
  vector F_MRFSS_D_out(styr,endyr); //used for intermediate calculations in fMRFSS get_mortality
  number F_MRFSS_D_ratio;  //ratio of average discard F to fishery F, for projection discards back before data

  number Dmort_HL;
  number Dmort_GN;
  number Dmort_MRFSS;

//--------Bycatch ----------------------------;
  init_bounded_number log_avg_F_shrimp(-10,0,1);
  init_bounded_dev_vector log_F_dev_shrimp(styr_shrimp_B,endyr_shrimp_B,-20,6,2);
  matrix F_shrimp(styr,endyr,1,nages);
  vector F_shrimp_out(styr,endyr);       //used for intermediate calculations in fcn get_mortality

////---Per-recruit stuff---------------------------------------------------------------------------
  vector N_age_spr_F(1,nages);        //numbers at age for SPR calculations: beginning of year, females only
  vector N_age_spr_mdyr_F(1,nages);   //numbers at age for SPR calculations: midyear, females only
  vector C_age_spr_F(1,nages);        //catch at age for SPR calculations, females only
  vector Z_age_spr_F(1,nages);        //total mortality at age for SPR calculations, females only
  vector F_L_age_spr_F(1,nages);      //fishing mortality (landings, not discards) at age for SPR calculations, females
```

```
  vector N_age_spr_M(1,nages);         //numbers at age for SPR calculations: beginning of year, males only
  vector N_age_spr_mdyr_M(1,nages);    //numbers at age for SPR calculations: midyear, males only
  vector C_age_spr_M(1,nages);         //catch at age for SPR calculations, males only
  vector Z_age_spr_M(1,nages);         //total mortality at age for SPR calculations, males only
  vector F_L_age_spr_M(1,nages);       //fishing mortality (landings, not discards) at age for SPR calculations, males
  vector spr_static(styr,endyr);     //vector of static SPR values by year
  vector F_spr(1,n_iter_spr);        //values of full F to be used in per-recruit and equilibrium calculations
  vector spr_spr(1,n_iter_spr);      //reporductive capacity-per-recruit values corresponding to F values in F_spr
  vector L_spr(1,n_iter_spr);        //landings(mt)-per-recruit values corresponding to F values in F_spr
  vector E_spr(1,n_iter_spr);        //exploitation rate 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
  vector N_spr_F_init(1,nages);          //Used to compute spr at F=0: start of year
  vector N_spr_F_init_mdyr(1,nages);          //Used to compute spr at F=0: midpt of year
  number spr_F_init;                 //Spawning biomass per recruit at F=F_F_hist

  number bpr_F0;                   //Biomass per recruit at F=0

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

  number w_I_LB_HL;
  number w_I_FL_gill1;
  number w_I_FL_gill2;
  number w_I_LB_gill;
  number w_I_FL_HL;
  number w_I_CN;
  number w_I_MRFSS;
  number w_I_SMAP_1YR;
  number w_I_SMAP_YOY;
  number w_R;
  number w_R_init;
  number w_R_end;
  number w_F;
  number w_B1dB0;
  number w_fullF;
  number w_cvlen_dev;
  number w_cvlen_diff;

  number f_LB_HL_cpue;
  number f_FL_HL_cpue;
  number f_FL_gill1_cpue;
  number f_FL_gill2_cpue;
  number f_LB_gill_cpue;
  number f_CN_cpue;
  number f_MRFSS_cpue;
  number f_SMAP_YOY_cpue;
  number f_SMAP_1YR_cpue;

  number f_HL_L;
  number f_PN_L;
  number f_GN_L;
  number f_CN_L;
  number f_MRFSS_L;

  number f_HL_D;
  number f_GN_D;
  number f_MRFSS_D;

  number f_shrimp_B;

  number f_HL_lenc;
  number f_PN_lenc;
  number f_GN_lenc;
  number f_CN_lenc;
  number f_MRFSS_lenc;

  number f_HL_agec;
  number f_PN_agec;
  number f_GN_agec;
  number f_CN_agec;
  number f_MRFSS_agec;

  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; //used only during model development. can be removed.

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


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

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

 //##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
 //##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
 PRELIMINARY_CALCS_SECTION
 // Set values of fixed parameters or set initial guess of estimated parameters
  Dmort_HL=set_Dmort_HL;
  Dmort_GN=set_Dmort_GN;
  Dmort_MRFSS=set_Dmort_MRFSS;

  obs_HL_D=Dmort_HL*obs_HL_released;
  obs_GN_D=Dmort_GN*obs_GN_released;
  obs_MRFSS_D=Dmort_MRFSS*obs_MRFSS_released;

  obs_MRFSS_L(styr_MRFSS_L,1979)=L_rec_multiply*obs_MRFSS_L(styr_MRFSS_L,1979);
  obs_MRFSS_D(styr_MRFSS_L,1979)=L_rec_multiply*obs_MRFSS_D(styr_MRFSS_L,1979);

  E_age_st=set_E_age_st;   //E computed over ages E_age_st +   [E_age_st-1+ if model starts with age 0]

  Linf_f=set_Linf_f;
  Linf_m=set_Linf_m;
  K_f=set_K_f;
  K_m=set_K_m;
  t0_f=set_t0_f;
  t0_m=set_t0_m;

  nlenbins2=nlenbins+20;  //lenbins,nlenbins for plus group; lenbins2,nlenbins2 assume truncation at 90cm
  lenbins2(1)=lenbins(1);
  for(ilen=2;ilen<=nlenbins2;ilen++){
    lenbins2(ilen)=lenbins2(ilen-1)+1;
  }
  prop_f_a0=set_prop_f_a0;
  //selpar_L50_D_2=t0-log(1.0-304.8/Linf)/K; //age at size limit: 304.8 = limit in mm

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

  log_q_FL_HL=set_logq_FL_HL;
  log_q_FL_gill1=set_logq_FL_gill1;
  log_q_FL_gill2=set_logq_FL_gill2;
  log_q_LB_gill=set_logq_LB_gill;
```

```
log_q_MRFSS=set_logq_MRFSS;

//q_rate=set_q_rate;

//L_early_bias=set_L_early_bias;
//L_commHAL_bias=1.0;

w_L=set_w_L;
w_D=set_w_D;
w_lc=set_w_lc;
w_ac=set_w_ac;

w_I_LB_HL=set_w_I_LB_HL;
w_I_FL_HL=set_w_I_FL_HL;
w_I_FL_gill1=set_w_I_FL_gill1;
w_I_FL_gill2=set_w_I_FL_gill2;
w_I_LB_gill=set_w_I_LB_gill;
w_I_CN=set_w_I_CN;
w_I_MRFSS=set_w_I_MRFSS;
w_I_SMAP_YOY=set_w_I_SMAP_YOY;
w_I_SMAP_1YR=set_w_I_SMAP_1YR;

w_R=set_w_R;
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_fullF=set_w_fullF;
w_cvlen_dev=set_w_cvlen_dev;
w_cvlen_diff=set_w_cvlen_diff;

F_hist=set_F_hist;
log_avg_F_HL=set_log_avg_F_HL;
log_F_dev_HL=set_log_F_dev_HL;
log_avg_F_PN=set_log_avg_F_PN;
log_F_dev_PN=set_log_F_dev_PN;
log_avg_F_GN=set_log_avg_F_GN;
log_F_dev_GN=set_log_F_dev_GN;
log_avg_F_CN=set_log_avg_F_CN;
log_F_dev_CN=set_log_F_dev_CN;
log_avg_F_MRFSS=set_log_avg_F_MRFSS;
log_F_dev_MRFSS=set_log_F_dev_MRFSS;

log_avg_F_HL_D=set_log_avg_F_HL_D;
log_F_dev_HL_D=set_log_F_dev_HL_D;
log_avg_F_GN_D=set_log_avg_F_GN_D;
log_F_dev_GN_D=set_log_F_dev_GN_D;
log_avg_F_MRFSS_D=set_log_avg_F_MRFSS_D;
log_F_dev_MRFSS_D=set_log_F_dev_MRFSS_D;

log_avg_F_shrimp=set_log_avg_F_shrimp;
log_F_dev_shrimp=set_log_F_dev_shrimp;

log_len_cv=log(set_len_cv);
log_R0=set_log_R0;

L50_diff=set_L50_diff; //shift in selectivity b/w males and females (same for all gears)
sel_historical_F=set_sel_historical_F;
selpar_slope_HL=set_selpar_slope_HL;
selpar_L50_HL_keep=set_selpar_L50_HL_keep;
sel_HL_D_M=set_sel_HL_D_M;  //selectivity for discards - males
sel_HL_D_F=set_sel_HL_D_F;  //selectivity for discards - females
selpar_slope_PN=set_selpar_slope_PN;
selpar_L50_PN=set_selpar_L50_PN;
selpar_slope_GN=set_selpar_slope_GN;
selpar_L50_GN_keep=set_selpar_L50_GN_keep;
selpar_slope_GN2=set_selpar_slope2_GN2;
selpar_L50_GN2=set_selpar_L50_GN2;
selpar_slope2_GN2=set_selpar_slope2_GN2;
selpar_L502_GN2=set_selpar_L502_GN2;
sel_GN_D_M=set_sel_GN_D_M;  //selectivity for discards - males
sel_GN_D_F=set_sel_GN_D_F;  //selectivity for discards - females
selpar_slope_CN=set_selpar_slope_CN;
selpar_L50_CN=set_selpar_L50_CN;
selpar_slope_MRFSS=set_selpar_slope_MRFSS;
selpar_L50_MRFSS_keep=set_selpar_L50_MRFSS_keep;
sel_MRFSS_D_M=set_sel_MRFSS_D_M;  //selectivity for discards - males
sel_MRFSS_D_F=set_sel_MRFSS_D_F;  //selectivity for discards - females
sel_shrimp=set_sel_shrimp_B;  //selectivity for bycatch (same for both sexes)
```

```
 selpar_age0_MRFSS=set_selpar_age0_MRFSS;
 selpar_age1_MRFSS=set_selpar_age1_MRFSS;
 selpar_age2_MRFSS=set_selpar_age2_MRFSS;
 selpar_age3_MRFSS=set_selpar_age3_MRFSS;

 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;         //converstion of metric tons to 1000 lb
 //df=0.001; //difference for msy derivative approximations
 zero_dum=0.0;

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

 SSB_msy_out=0.0;

 maturity_f=maturity_f_obs;

////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_HL_lenc_allyr=missing;
      nsamp_HL_agec_allyr=missing;
      nsamp_PN_agec_allyr=missing;
      nsamp_CN_lenc_allyr=missing;
      nsamp_CN_agec_allyr=missing;
      for (iyear=1; iyear<=nyr_HL_lenc; iyear++)
        {
          nsamp_HL_lenc_allyr(yrs_HL_lenc(iyear))=nsamp_HL_lenc(iyear);
        }
      for (iyear=1; iyear<=nyr_HL_agec; iyear++)
        {
          nsamp_HL_agec_allyr(yrs_HL_agec(iyear))=nsamp_HL_agec(iyear);
        }
      for (iyear=1; iyear<=nyr_PN_agec; iyear++)
        {
          nsamp_PN_agec_allyr(yrs_PN_agec(iyear))=nsamp_PN_agec(iyear);
        }
      for (iyear=1; iyear<=nyr_CN_lenc; iyear++)
        {
          nsamp_CN_lenc_allyr(yrs_CN_lenc(iyear))=nsamp_CN_lenc(iyear);
        }
      for (iyear=1; iyear<=nyr_CN_agec; iyear++)
        {
          nsamp_CN_agec_allyr(yrs_CN_agec(iyear))=nsamp_CN_agec(iyear);
        }


  age_error_matrix=set_age_error_matrix;

//fill in reproductive caculations with zero's
  reprod.initialize();
  prop_f.initialize();
  prop_f_F0.initialize();

//fill in F's, Catch matrices, and log rec dev with zero's
   F_HL_M.initialize();
   F_HL_F.initialize();
   C_HL_M.initialize();
   C_HL_F.initialize();
   F_PN_M.initialize();
   F_PN_F.initialize();
   C_PN_F.initialize();
   C_PN_M.initialize();
```

```
  F_GN_M.initialize();
  F_GN_F.initialize();
  C_GN_M.initialize();
  C_GN_F.initialize();
  F_CN_M.initialize();
  F_CN_F.initialize();
  C_CN_M.initialize();
  C_CN_F.initialize();
  F_MRFSS_M.initialize();
  F_MRFSS_F.initialize();
  C_MRFSS_M.initialize();
  C_MRFSS_F.initialize();
  F_shrimp.initialize();
  C_shrimp_M.initialize();
  C_shrimp_F.initialize();

  C_HL_D_M.initialize();
  C_HL_D_F.initialize();
  C_GN_D_M.initialize();
  C_GN_D_F.initialize();
  C_MRFSS_D_M.initialize();
  C_MRFSS_D_F.initialize();

  L_GN_D.initialize();       //discards in 1000lb whole weight
  L_HL_D.initialize();       //discards in 1000lb whole weight
  L_MRFSS_D.initialize();       //discards in 1000lb whole weight
  L_shrimp.initialize();       //discards in 1000lb whole weight

  F_HL_D_M.initialize();
  F_HL_D_F.initialize();
  F_GN_D_M.initialize();
  F_GN_D_F.initialize();
  F_MRFSS_D_M.initialize();
  F_MRFSS_D_F.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_and_weight_at_age();
  //cout << "got length and weight transitions" <<endl;
  get_selectivity();
  //cout << "got selectivity" << endl;
  get_length_at_age_dist();
  //cout<< "got predicted length at age distribution"<<endl;
  get_spr_F0();
  //cout << "got F0 spr" << 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_length_comps();
  //cout<< "got length comps"<< endl;
  get_age_comps();
  //cout<< "got age comps"<< endl;
  get_sel_weighted_current();
  //cout<<"got sel weighted"<<endl;
  get_indices();
  //cout << "got indices" << endl;
  evaluate_objective_function();
```

```
 //cout << "objective function calculations complete" << endl;


FUNCTION get_length_and_weight_at_age
  //compute mean length (mm) and weight (whole and gutted) at age
    meanlen_m=Linf_m*(1.0-mfexp(-K_m*(agebins-t0_m)));   //length in mm - males
    meanlen_f=Linf_f*(1.0-mfexp(-K_f*(agebins-t0_f)));   //length in mm - females
    wgt_g_m=wgtpar_a*pow(meanlen_m,wgtpar_b);        //wgt in grams
    wgt_kg_m=g2kg*wgt_g_m;                           //wgt in grams
    wgt_m=g2mt*wgt_g_m;     //mt of whole wgt: g2mt converts g to mt
    wgt_klb_m=mt2klb*wgt_m;                          //1000 lb of whole wgt
    wgt_g_f=wgtpar_a*pow(meanlen_f,wgtpar_b);        //wgt in grams
    wgt_kg_f=g2kg*wgt_g_f;                           //wgt in grams
    wgt_f=g2mt*wgt_g_f;     //mt of whole wgt: g2mt converts g to mt
    wgt_klb_f=mt2klb*wgt_f;

FUNCTION get_selectivity
// ------- compute landings and discard selectivities

  for (iage=1; iage<=nages; iage++)
  {
    sel_HL_keep_M(iage)=1./(1.+mfexp(-1.*selpar_slope_HL*(double(agebins(iage))-selpar_L50_HL_keep-L50_diff)));
    sel_HL_keep_F(iage)=1./(1.+mfexp(-1.*selpar_slope_HL*(double(agebins(iage))-selpar_L50_HL_keep)));
    sel_PN_M(iage)=1./(1.+mfexp(-1.*selpar_slope_PN*(double(agebins(iage))-selpar_L50_PN-L50_diff)));
    sel_PN_F(iage)=1./(1.+mfexp(-1.*selpar_slope_PN*(double(agebins(iage))-selpar_L50_PN)));
    sel_GN_keep_M(iage)=1./(1.+mfexp(-1.*selpar_slope_GN*(double(agebins(iage))-selpar_L50_GN_keep-L50_diff)));
    sel_GN_keep_F(iage)=1./(1.+mfexp(-1.*selpar_slope_GN*(double(agebins(iage))-selpar_L50_GN_keep)));
    sel_GN_keep_M2(iage)=(1./(1.+mfexp(-1.*selpar_slope_GN2*(double(agebins(iage))-
                  selpar_L50_GN2))))*(1-(1./(1.+mfexp(-1.*selpar_slope2_GN2*
                  (double(agebins(iage))-(selpar_L50_GN2+selpar_L502_GN2))))); //double logistic
    sel_CN_M(iage)=1./(1.+mfexp(-1.*selpar_slope_CN*(double(agebins(iage))-selpar_L50_CN-L50_diff)));
    sel_CN_F(iage)=1./(1.+mfexp(-1.*selpar_slope_CN*(double(agebins(iage))-selpar_L50_CN)));
    //sel_MRFSS_keep_M(iage)=1./(1.+mfexp(-1.*selpar_slope_MRFSS*(double(agebins(iage))-selpar_L50_MRFSS_keep-L50_diff)));
    //sel_MRFSS_keep_F(iage)=1./(1.+mfexp(-1.*selpar_slope_MRFSS*(double(agebins(iage))-selpar_L50_MRFSS_keep)));
    if(iage==1)
    {
      sel_MRFSS_keep_F(iage)=selpar_age0_MRFSS;
      sel_MRFSS_keep_M(iage)=selpar_age0_MRFSS;
    }
    if(iage==2)
    {
      sel_MRFSS_keep_F(iage)=selpar_age1_MRFSS;
      sel_MRFSS_keep_M(iage)=selpar_age1_MRFSS;
    }
    if(iage==3)
    {
      sel_MRFSS_keep_F(iage)=selpar_age2_MRFSS;
      sel_MRFSS_keep_M(iage)=selpar_age2_MRFSS;
    }
    if(iage>3)
    {
      sel_MRFSS_keep_F(iage)=selpar_age3_MRFSS;
      sel_MRFSS_keep_M(iage)=selpar_age3_MRFSS;
    }
    //discards, bycatch are preset
  }
  //sel_MRFSS_keep_M=sel_MRFSS_keep_M/max(sel_MRFSS_keep_M);
  sel_GN_keep_M2=sel_GN_keep_M2/max(sel_GN_keep_M2); //renormalize
  sel_GN_keep_F2=sel_GN_keep_M2;

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<=nlenbins2;ilen++)
    {
      //convert to centimeters for matching
      lenprob_m2(iage,ilen)=(mfexp(-(square(lenbins2(ilen)-0.1*meanlen_m(iage))/
      (2.*square(len_cv(iage)*0.1*meanlen_m(iage)))))/(sqrt2pi*0.1*len_cv(iage)*meanlen_m(iage)));

      lenprob_f2(iage,ilen)=(mfexp(-(square(lenbins2(ilen)-0.1*meanlen_f(iage))/
      (2.*square(len_cv(iage)*0.1*meanlen_f(iage)))))/(sqrt2pi*0.1*len_cv(iage)*meanlen_f(iage)));
    }
    lenprob_m2(iage)/=sum(lenprob_m2(iage)); //standardize to account for truncated normal (i.e., no sizes<0)
    lenprob_f2(iage)/=sum(lenprob_f2(iage)); //standardize to account for truncated normal (i.e., no sizes<0)
    for(ilen=1;ilen<=nlenbins;ilen++){
      lenprob_m(iage,ilen)=lenprob_m2(iage,ilen);
      lenprob_f(iage,ilen)=lenprob_f2(iage,ilen);
```

```
  }
  for(ilen=nlenbins+1;ilen<=nlenbins2;ilen++){
   lenprob_m(iage,nlenbins)+=lenprob_m2(iage,ilen);
   lenprob_f(iage,nlenbins)+=lenprob_f2(iage,ilen);
  }
 }

FUNCTION get_spr_F0
 reprod=elem_prod(maturity_f,wgt_f);
 //at mdyr, apply half this yr's mortality, half next yr's
 N_spr_F0(1)=prop_f_a0*mfexp(-1.0*M(1)/2.0);  //at midpt of year
 N_bpr_F0(1)=1.;
 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)/2.0 + M(iage)/2.0));
  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=prop_f_a0*sum(elem_prod(N_bpr_F0,wgt_f))+(1.-prop_f_a0)*sum(elem_prod(N_bpr_F0,wgt_m));
 B0=R0*bpr_F0;

FUNCTION get_mortality
 fullF=0.0;

 for (iyear=styr; iyear<=endyr; iyear++) {
  if(iyear<styr_HL_L){
   F_HL_out(iyear)=0.0;
  }
  else{
   F_HL_out(iyear)=mfexp(log_avg_F_HL+log_F_dev_HL(iyear));
  }
  F_HL_F(iyear)=sel_HL_keep_F*F_HL_out(iyear);
  F_HL_M(iyear)=sel_HL_keep_M*F_HL_out(iyear);
  fullF(iyear)+=F_HL_out(iyear);

  if(iyear<styr_PN_L){
   F_PN_out(iyear)=0.0;
  }
  else{
   F_PN_out(iyear)=mfexp(log_avg_F_PN+log_F_dev_PN(iyear));
  }
  F_PN_F(iyear)=sel_PN_F*F_PN_out(iyear);
  F_PN_M(iyear)=sel_PN_M*F_PN_out(iyear);
  fullF(iyear)+=F_PN_out(iyear);

  if(iyear<styr_GN_L){
   F_GN_out(iyear)=0.0;
  }
  else {
   F_GN_out(iyear)=mfexp(log_avg_F_GN+log_F_dev_GN(iyear));
  }
  if(iyear<1995){
   F_GN_F(iyear)=sel_GN_keep_F*F_GN_out(iyear);
   F_GN_M(iyear)=sel_GN_keep_M*F_GN_out(iyear);
  }
  else{
   F_GN_F(iyear)=sel_GN_keep_F2*F_GN_out(iyear);
   F_GN_M(iyear)=sel_GN_keep_M2*F_GN_out(iyear);
  }
  fullF(iyear)+=F_GN_out(iyear);

  if(iyear<styr_CN_L){
   F_CN_out(iyear)=0.0;
  }
  else{
   F_CN_out(iyear)=mfexp(log_avg_F_CN+log_F_dev_CN(iyear));
  }
  F_CN_F(iyear)=sel_CN_F*F_CN_out(iyear);
  F_CN_M(iyear)=sel_CN_M*F_CN_out(iyear);
  fullF(iyear)+=F_CN_out(iyear);

  if(iyear<styr_MRFSS_L){
   F_MRFSS_out(iyear)=0.0;
  }
  else{
   F_MRFSS_out(iyear)=mfexp(log_avg_F_MRFSS+log_F_dev_MRFSS(iyear));
```

```
  }
  F_MRFSS_F(iyear)=sel_MRFSS_keep_F*F_MRFSS_out(iyear);
  F_MRFSS_M(iyear)=sel_MRFSS_keep_M*F_MRFSS_out(iyear);
  fullF(iyear)+=F_MRFSS_out(iyear);

  //discards
  if(iyear<styr_HL_D){
    F_HL_D_out(iyear)=0.0;
  }
  else{
    F_HL_D_out(iyear)=mfexp(log_avg_F_HL_D+log_F_dev_HL_D(iyear));
  }
  fullF(iyear)+=F_HL_D_out(iyear);
  F_HL_D_M(iyear)=sel_HL_D_M*F_HL_D_out(iyear);
  F_HL_D_F(iyear)=sel_HL_D_F*F_HL_D_out(iyear);

  if(iyear<styr_GN_D){
    F_GN_D_out(iyear)=0.0;
  }
  else{
    F_GN_D_out(iyear)=mfexp(log_avg_F_GN_D+log_F_dev_GN_D(iyear));
  }
  fullF(iyear)+=F_GN_D_out(iyear);
  F_GN_D_M(iyear)=sel_GN_D_M*F_GN_D_out(iyear);
  F_GN_D_F(iyear)=sel_GN_D_F*F_GN_D_out(iyear);

  if(iyear<styr_MRFSS_D){
    F_MRFSS_D_out(iyear)=0.0;
  }
  else{
    F_MRFSS_D_out(iyear)=mfexp(log_avg_F_MRFSS_D+log_F_dev_MRFSS_D(iyear));
  }
  fullF(iyear)+=F_MRFSS_D_out(iyear);
  F_MRFSS_D_M(iyear)=sel_MRFSS_D_M*F_MRFSS_D_out(iyear);
  F_MRFSS_D_F(iyear)=sel_MRFSS_D_F*F_MRFSS_D_out(iyear);

  if(iyear<styr_shrimp_B){
    F_shrimp_out(iyear)=0.0;
  }
  else{
    F_shrimp_out(iyear)=mfexp(log_avg_F_shrimp+log_F_dev_shrimp(iyear));
  }
  fullF(iyear)+=F_shrimp_out(iyear);
  F_shrimp(iyear)=sel_shrimp*F_shrimp_out(iyear);
 }


 for (iyear=styr; iyear<=endyr; iyear++) {
  F_F(iyear)=F_HL_F(iyear);
  F_F(iyear)+=F_PN_F(iyear);
  F_F(iyear)+=F_GN_F(iyear);
  F_F(iyear)+=F_CN_F(iyear);
  F_F(iyear)+=F_MRFSS_F(iyear);
  F_F(iyear)+=F_HL_D_F(iyear);
  F_F(iyear)+=F_GN_D_F(iyear);
  F_F(iyear)+=F_MRFSS_D_F(iyear);
  F_F(iyear)+=F_shrimp(iyear);
  Z_F(iyear)=M+F_F(iyear);

  F_M(iyear)=F_HL_M(iyear);
  F_M(iyear)+=F_PN_M(iyear);
  F_M(iyear)+=F_GN_M(iyear);
  F_M(iyear)+=F_CN_M(iyear);
  F_M(iyear)+=F_MRFSS_M(iyear);
  F_M(iyear)+=F_HL_D_M(iyear);
  F_M(iyear)+=F_GN_D_M(iyear);
  F_M(iyear)+=F_MRFSS_D_M(iyear);
  F_M(iyear)+=F_shrimp(iyear);
  Z_M(iyear)=M+F_M(iyear);
 }
 F_F_hist=sel_historical_F*F_hist;
 Z_hist=F_F_hist+M;

FUNCTION get_bias_corr
 var_rec_dev=norm2(log_dev_N_rec(styr_rec_dev,(endyr-2))-sum(log_dev_N_rec(styr_rec_dev,(endyr-2)))
          /(nyrs_rec-2.0))/(nyrs_rec-3.0); //sample variance from yrs styr_rec_dev-2005
 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;

 //Assume equilibrium age structure for first year dependent on historical F
 N_spr_F_init(1)=prop_f_a0;
 N_spr_F_init_mdyr(1)=N_spr_F_init(1)*mfexp((-1.*(Z_hist(1)))/2.0);
 for (iage=2; iage<=nages; iage++)
 {
  N_spr_F_init(iage)=N_spr_F_init(iage-1)*mfexp(-1.*(Z_hist(iage-1)));
  N_spr_F_init_mdyr(iage)=N_spr_F_init(iage)*mfexp((-1.*(Z_hist(iage)))/2.0);
 }
 N_spr_F_init(nages)=N_spr_F_init(nages)/(1.0-mfexp(-1.*(Z_hist(nages))));
 N_spr_F_init_mdyr(nages)=N_spr_F_init(nages)*mfexp((-1.*(Z_hist(nages)))/2.0);
 spr_F_init=sum(elem_prod(N_spr_F_init_mdyr,reprod));

 //R1=(R0/((5.0*steep-1.0)*spr_F_init))*(BiasCor*4.0*steep*spr_F_init-spr_F0*(1.0-steep));
 R1=(R0/((5.0*steep-1.0)*spr_F_init))*(4.0*steep*spr_F_init-spr_F0*(1.0-steep));  //take out bias correction in first year because recruitment
deterministic at the beginning
 if(R1<0.0)
   {
    R1=1.0;
   }

 N_M(styr,1)=(1.-prop_f_a0)*R1;
 N_F(styr,1)=prop_f_a0*R1;
 N_M_mdyr(styr,1)=N_M(styr,1)*mfexp(-1.*Z_M(styr,1)/2.0);
 N_F_mdyr(styr,1)=N_F(styr,1)*mfexp(-1.*Z_F(styr,1)/2.0);
 for (iage=2; iage<=nages; iage++)
 {
  N_M(styr,iage)=N_M(styr,iage-1)*mfexp(-1.*Z_hist(iage-1));
  N_M_mdyr(styr,iage)=N_M(styr,iage)*mfexp(-1.*Z_M(styr,iage)/2.0);
  N_F(styr,iage)=N_F(styr,iage-1)*mfexp(-1.*Z_hist(iage-1));
  N_F_mdyr(styr,iage)=N_F(styr,iage)*mfexp(-1.*Z_F(styr,iage)/2.0);
 }
 //plus group calculation
 N_M(styr,nages)=N_M(styr,nages)/(1.-mfexp(-1.*Z_hist(nages)));
 N_M_mdyr(styr,nages)=N_M(styr,nages)*mfexp(-1.*Z_M(styr,nages)/2.0);
 N_F(styr,nages)=N_F(styr,nages)/(1.-mfexp(-1.*Z_hist(nages)));
 N_F_mdyr(styr,nages)=N_F(styr,nages)*mfexp(-1.*Z_F(styr,nages)/2.0);

 SSB(styr)=sum(elem_prod(N_F_mdyr(styr),reprod));
 B(styr)=elem_prod(N_M(styr),wgt_m)+elem_prod(N_F(styr),wgt_f);
 totB(styr)=sum(B(styr));

//Rest of years
 for (iyear=styr; iyear<endyr; iyear++)
 {

  N_F(iyear+1)(2,nages)=++elem_prod(N_F(iyear)(1,nages-1),(mfexp(-1.*Z_F(iyear)(1,nages-1))));
  N_F(iyear+1,nages)+=N_F(iyear,nages)*mfexp(-1.*Z_F(iyear,nages));//plus group
  N_F_mdyr(iyear+1)(2,nages)=elem_prod(N_F(iyear+1)(2,nages),(mfexp(-1.*(Z_F(iyear+1)(2,nages))/2.0))); //mdyr
  N_F_mdyr(iyear+1,1)=0.0;
  N_M(iyear+1)(2,nages)=++elem_prod(N_M(iyear)(1,nages-1),(mfexp(-1.*Z_M(iyear)(1,nages-1))));
  N_M(iyear+1,nages)+=N_M(iyear,nages)*mfexp(-1.*Z_M(iyear,nages));//plus group
  N_M_mdyr(iyear+1)(2,nages)=elem_prod(N_M(iyear+1)(2,nages),(mfexp(-1.*(Z_M(iyear+1)(2,nages))/2.0))); //mdyr
  N_M_mdyr(iyear+1,1)=0.0;
  SSB(iyear+1)=sum(elem_prod(N_F_mdyr(iyear+1),reprod));

  for(iage=1;iage<=nages;iage++){
   prop_f(iyear,iage)=N_F(iyear,iage)/(N_F(iyear,iage)+N_M(iyear,iage)+dzero_dum);
  }

  if(iyear<(styr_rec_dev-1)) //recruitment follows S-R curve exactly
  {
     //add 0.00001 to avoid log(zero)
     N_F(iyear+1,1)=prop_f_a0*BiasCor*mfexp(log(((0.8*R0*steep*SSB(iyear+1))/(0.2*R0*spr_F0*
        (1.0-steep)+(steep-0.2)*SSB(iyear+1)))+dzero_dum));
  }
  else{
     N_F(iyear+1,1)=prop_f_a0*mfexp(log(((0.8*R0*steep*SSB(iyear+1))/(0.2*R0*spr_F0*
        (1.0-steep)+(steep-0.2)*SSB(iyear+1)))+dzero_dum)+log_dev_N_rec(iyear+1));
  }


  N_M(iyear+1,1)=N_F(iyear+1,1)*(1./prop_f_a0-1.);  //this is how it works out algebraically
  N_M_mdyr(iyear+1,1)=N_M(iyear+1,1)*mfexp(-1.*(Z_M(iyear+1,1)/2));
  N_F_mdyr(iyear+1,1)=N_F(iyear+1,1)*mfexp(-1.*(Z_F(iyear+1,1)/2));
  B(iyear+1)=elem_prod(N_F(iyear+1),wgt_f)+elem_prod(N_M(iyear+1),wgt_m);
  totB(iyear+1)=sum(B(iyear+1));
 }
```

```
   //last year (projection) has no recruitment variability
   N_M(endyr+1)(2,nages)=++elem_prod(N_M(endyr)(1,nages-1),(mfexp(-1.*Z_M(endyr)(1,nages-1))));
   N_M(endyr+1,nages)+=N_M(endyr,nages)*mfexp(-1.*Z_M(endyr,nages));//plus group
   N_M_mdyr(endyr+1)(1,nages)=elem_prod(N_M(endyr+1)(1,nages),(mfexp(-1.*(Z_M(endyr)(1,nages))/2.0))); //mdyr
   N_F(endyr+1)(2,nages)=++elem_prod(N_F(endyr)(1,nages-1),(mfexp(-1.*Z_F(endyr)(1,nages-1))));
   N_F(endyr+1,nages)+=N_F(endyr,nages)*mfexp(-1.*Z_F(endyr,nages));//plus group
   N_F_mdyr(endyr+1)(1,nages)=elem_prod(N_F(endyr+1)(1,nages),(mfexp(-1.*(Z_F(endyr)(1,nages))/2.0))); //mdyr
   SSB_extra=sum(elem_prod(N_F_mdyr(endyr+1),reprod));
   N_F(endyr+1,1)=prop_f_a0*mfexp(log(((0.8*R0*steep*SSB_extra)/(0.2*R0*spr_F0*
            (1.0-steep)+(steep-0.2)*SSB_extra))+dzero_dum));
   N_M(endyr+1,1)=(1.-prop_f_a0)*mfexp(log(((0.8*R0*steep*SSB_extra)/(0.2*R0*spr_F0*
            (1.0-steep)+(steep-0.2)*SSB_extra))+dzero_dum));
   B(endyr+1)=elem_prod(N_M(endyr+1),wgt_m)+elem_prod(N_F(endyr+1),wgt_f);
   totB(endyr+1)=sum(B(endyr+1));

 //Recruitment time series
   rec=column(N_F,1)+column(N_M,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++)
     {
       C_HL_M(iyear,iage)=N_M(iyear,iage)*F_HL_M(iyear,iage)*
         (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
       C_HL_F(iyear,iage)=N_F(iyear,iage)*F_HL_F(iyear,iage)*
         (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
       C_PN_M(iyear,iage)=N_M(iyear,iage)*F_PN_M(iyear,iage)*
         (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
       C_PN_F(iyear,iage)=N_F(iyear,iage)*F_PN_F(iyear,iage)*
         (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
       C_GN_M(iyear,iage)=N_M(iyear,iage)*F_GN_M(iyear,iage)*
         (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
       C_GN_F(iyear,iage)=N_F(iyear,iage)*F_GN_F(iyear,iage)*
         (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
       C_CN_M(iyear,iage)=N_M(iyear,iage)*F_CN_M(iyear,iage)*
         (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
       C_CN_F(iyear,iage)=N_F(iyear,iage)*F_CN_F(iyear,iage)*
         (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
       C_MRFSS_M(iyear,iage)=N_M(iyear,iage)*F_MRFSS_M(iyear,iage)*
         (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
       C_MRFSS_F(iyear,iage)=N_F(iyear,iage)*F_MRFSS_F(iyear,iage)*
         (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
       C_shrimp_M(iyear,iage)=N_M(iyear,iage)*F_shrimp(iyear,iage)*
         (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
       C_shrimp_F(iyear,iage)=N_F(iyear,iage)*F_shrimp(iyear,iage)*
         (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
     }
   }

 FUNCTION get_landings_wgt
 //---Predicted landings-------------------------
   for (iyear=styr; iyear<=endyr; iyear++)
   {
     L_HL(iyear)=elem_prod(C_HL_M(iyear),wgt_m)+elem_prod(C_HL_F(iyear),wgt_f);          //in mt
     L_HL_klb(iyear)=elem_prod(C_HL_F(iyear),wgt_klb_f)+elem_prod(C_HL_M(iyear),wgt_klb_m); //in 1000 lb
     L_PN(iyear)=elem_prod(C_PN_M(iyear),wgt_m)+elem_prod(C_PN_F(iyear),wgt_f);   //in mt
     L_PN_klb(iyear)=elem_prod(C_PN_F(iyear),wgt_klb_f)+elem_prod(C_PN_M(iyear),wgt_klb_m); //in 1000 lb
     L_GN(iyear)=elem_prod(C_GN_M(iyear),wgt_m)+elem_prod(C_GN_F(iyear),wgt_f); //in mt
     L_GN_klb(iyear)=elem_prod(C_GN_F(iyear),wgt_klb_f)+elem_prod(C_GN_M(iyear),wgt_klb_m); //in 1000 lb
     L_CN(iyear)=elem_prod(C_CN_M(iyear),wgt_m)+elem_prod(C_CN_F(iyear),wgt_f); //in mt
     L_CN_klb(iyear)=elem_prod(C_CN_F(iyear),wgt_klb_f)+elem_prod(C_CN_M(iyear),wgt_klb_m); //in 1000 lb
     L_MRFSS(iyear)=elem_prod(C_MRFSS_M(iyear),wgt_m)+elem_prod(C_MRFSS_F(iyear),wgt_f); //in mt
     L_MRFSS_klb(iyear)=elem_prod(C_MRFSS_F(iyear),wgt_klb_f)+elem_prod(C_MRFSS_M(iyear),wgt_klb_m); //in 1000 lb
     L_shrimp(iyear)=elem_prod(C_shrimp_F(iyear),wgt_f)+elem_prod(C_shrimp_M(iyear),wgt_m); //in mt
   }
   for (iyear=styr_HL_L; iyear<=endyr_HL_L; iyear++){
     pred_HL_L(iyear)=sum(L_HL_klb(iyear));
   }
   for (iyear=styr_PN_L; iyear<=endyr_PN_L; iyear++){
     pred_PN_L(iyear)=sum(L_PN_klb(iyear));
   }
   for (iyear=styr_GN_L; iyear<=endyr_GN_L; iyear++){
     pred_GN_L(iyear)=sum(L_GN_klb(iyear));
```

```
  }
  for (iyear=styr_CN_L; iyear<=endyr_CN_L; iyear++){
    pred_CN_L(iyear)=sum(L_CN_klb(iyear));
  }
  for(iyear=styr_MRFSS_L;iyear<=endyr_MRFSS_L;iyear++){
    pred_MRFSS_L(iyear)=sum(C_MRFSS_F(iyear)+C_MRFSS_M(iyear))/1000.0; //in 1000's (numbers)
  }
  for(iyear=styr_shrimp_B;iyear<=endyr_shrimp_B;iyear++){
    pred_shrimp_B(iyear)=sum(C_shrimp_F(iyear)+C_shrimp_M(iyear))/1000.0; //in 1000's (numbers)
  }

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

  for (iyear=styr_HL_D; iyear<=endyr_HL_D; iyear++){
    for (iage=1; iage<=nages; iage++){
      C_HL_D_F(iyear,iage)=N_F(iyear,iage)*F_HL_D_F(iyear,iage)*
        (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
      C_HL_D_M(iyear,iage)=N_M(iyear,iage)*F_HL_D_M(iyear,iage)*
        (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
    }
    L_HL_D(iyear)=elem_prod(C_HL_D_F(iyear),wgt_f)+elem_prod(C_HL_D_M(iyear),wgt_m);        //discards in 1000lb whole weight
    pred_HL_D(iyear)=(sum(C_HL_D_M(iyear))+sum(C_HL_D_F(iyear)))/1000.0; //pred annual dead discards in 1000s
  }
  for (iyear=styr_GN_D; iyear<=endyr_GN_D; iyear++){
    for (iage=1; iage<=nages; iage++){
      C_GN_D_F(iyear,iage)=N_F(iyear,iage)*F_GN_D_F(iyear,iage)*
        (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
      C_GN_D_M(iyear,iage)=N_M(iyear,iage)*F_GN_D_M(iyear,iage)*
        (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
    }
    L_GN_D(iyear)=elem_prod(C_GN_D_F(iyear),wgt_f)+elem_prod(C_GN_D_M(iyear),wgt_m);        //discards in 1000lb whole weight
    pred_GN_D(iyear)=(sum(C_GN_D_M(iyear))+sum(C_GN_D_F(iyear)))/1000.0; //pred annual dead discards in 1000s
  }
  for (iyear=styr_MRFSS_D; iyear<=endyr_MRFSS_D; iyear++){
    for (iage=1; iage<=nages; iage++){
      C_MRFSS_D_F(iyear,iage)=N_F(iyear,iage)*F_MRFSS_D_F(iyear,iage)*
        (1.-mfexp(-1.*Z_F(iyear,iage)))/Z_F(iyear,iage);
      C_MRFSS_D_M(iyear,iage)=N_M(iyear,iage)*F_MRFSS_D_M(iyear,iage)*
        (1.-mfexp(-1.*Z_M(iyear,iage)))/Z_M(iyear,iage);
    }
    L_MRFSS_D(iyear)=elem_prod(C_MRFSS_D_F(iyear),wgt_f)+elem_prod(C_MRFSS_D_M(iyear),wgt_m);        //discards in 1000lb whole weight
    pred_MRFSS_D(iyear)=(sum(C_MRFSS_D_M(iyear))+sum(C_MRFSS_D_F(iyear)))/1000.0; //pred annual dead discards in 1000s
  }

FUNCTION get_indices
 //FL hand lines cpue
  for (iyear=styr_FL_HL_cpue; iyear<=endyr_FL_HL_cpue; iyear++)
  {   //index in whole wgt (lb) units, wgt_klb in 1000 lb, but the multiplier (1000) is absorbed by q
      N_FL_HL(iyear)=elem_prod(elem_prod(N_F_mdyr(iyear),sel_wgted_tot_F),wgt_klb_f)+
       elem_prod(elem_prod(N_M_mdyr(iyear),sel_wgted_tot_F),wgt_klb_m);
      pred_FL_HL_cpue(iyear)=mfexp(log_q_FL_HL)*sum(N_FL_HL(iyear));
  }
   //SEAMAP YOY cpue
  for (iyear=styr_SMAP_YOY_cpue; iyear<=endyr_SMAP_YOY_cpue; iyear++)
  {   //index in 1000's (numbers)
      pred_SMAP_YOY_cpue(iyear)=mfexp(log_q_SMAP_YOY)*(N_F(iyear,1)+N_M(iyear,1));
  }
  //SEAMAP 1YR cpue
  for (iyear=styr_SMAP_1YR_cpue; iyear<=endyr_SMAP_1YR_cpue; iyear++)
  {   //index in 1000's (numbers)
      pred_SMAP_1YR_cpue(iyear)=mfexp(log_q_SMAP_1YR)*(N_F(iyear,2)+N_M(iyear,2));
  }

FUNCTION get_length_comps
   //Hand lines
  for (iyear=1;iyear<=nyr_HL_lenc;iyear++)
  {

pred_HL_lenc(iyear)=(C_HL_F(yrs_HL_lenc(iyear))*lenprob_f+C_HL_M(yrs_HL_lenc(iyear))*lenprob_m)/(sum(C_HL_F(yrs_HL_lenc(iyear))+C_H
L_M(yrs_HL_lenc(iyear)))+dzero_dum);
  }
  //Pound Nets
  for (iyear=styr_PN_lenc;iyear<=endyr_PN_lenc;iyear++)
  {
    pred_PN_lenc(iyear)=(C_PN_F(iyear)*lenprob_f+C_PN_M(iyear)*lenprob_m)/sum(C_PN_F(iyear)+C_PN_M(iyear));
  }
  //Gillnet
  for (iyear=styr_GN_lenc;iyear<=endyr_GN_lenc;iyear++)
  {
```

```
  pred_GN_lenc(iyear)=(C_GN_F(iyear)*lenprob_f+C_GN_M(iyear)*lenprob_m)/sum(C_GN_F(iyear)+C_GN_M(iyear));
 }
 //Castnets
 for (iyear=1;iyear<=nyr_CN_lenc;iyear++)
 {
  iyear2=yrs_CN_lenc(iyear);
  pred_CN_lenc(iyear)=(C_CN_F(iyear2)*lenprob_f+C_CN_M(iyear2)*lenprob_m)/sum(C_CN_F(iyear2)+C_CN_M(iyear2));
 }
 //MRFSS
 for (iyear=styr_MRFSS_lenc;iyear<=endyr_MRFSS_lenc;iyear++)
 {
  pred_MRFSS_lenc(iyear)=(C_MRFSS_F(iyear)*lenprob_f+C_MRFSS_M(iyear)*lenprob_m)/sum(C_MRFSS_F(iyear)+C_MRFSS_M(iyear));
 }

FUNCTION get_age_comps
  //Hand lines
 for (iyear=1;iyear<=nyr_HL_agec;iyear++)
 {
  pred_HL_agec(iyear)=(C_HL_F(yrs_HL_agec(iyear))+C_HL_M(yrs_HL_agec(iyear)))/
                (sum(C_HL_F(yrs_HL_agec(iyear))+C_HL_M(yrs_HL_agec(iyear)))+dzero_dum);
 }
 pred_HL_agec=pred_HL_agec*age_error_matrix;
 //Pound nets
 for (iyear=1;iyear<=nyr_PN_agec;iyear++)
 {
  pred_PN_agec(iyear)=(C_PN_F(yrs_PN_agec(iyear))+C_PN_M(yrs_PN_agec(iyear)))/
                sum(C_PN_F(yrs_PN_agec(iyear))+C_PN_M(yrs_PN_agec(iyear)));
 }
 pred_PN_agec=pred_PN_agec*age_error_matrix;
//Gill nets
 for (iyear=styr_GN_agec;iyear<=endyr_GN_agec;iyear++)
 {
  pred_GN_agec(iyear)=(C_GN_F(iyear)+C_GN_M(iyear))/
                sum(C_GN_F(iyear)+C_GN_M(iyear));
 }
 //Cast nets
 for (iyear=1;iyear<=nyr_CN_agec;iyear++)
 {
  pred_CN_agec(iyear)=(C_CN_F(yrs_CN_agec(iyear))+C_CN_M(yrs_CN_agec(iyear)))/
                sum(C_CN_F(yrs_CN_agec(iyear))+C_CN_M(yrs_CN_agec(iyear)));
 }
 //MRFSS
 for (iyear=styr_MRFSS_agec;iyear<=endyr_MRFSS_agec;iyear++)
 {
  pred_MRFSS_agec(iyear)=(C_MRFSS_F(iyear)+C_MRFSS_M(iyear))/
                sum(C_MRFSS_F(iyear)+C_MRFSS_M(iyear));
 }
//-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----
FUNCTION get_sel_weighted_current //get average of most recent 3 years selectivity for MSY calcs
 F_temp_sum=0.0;
 F_temp_sum+=mfexp((3.0*log_avg_F_HL+sum(log_F_dev_HL(endyr-2,endyr)))/3.0);
 F_temp_sum+=mfexp((3.0*log_avg_F_PN+sum(log_F_dev_PN(endyr-2,endyr)))/3.0);
 F_temp_sum+=mfexp((3.0*log_avg_F_GN+sum(log_F_dev_GN(endyr-2,endyr)))/3.0);
 F_temp_sum+=mfexp((3.0*log_avg_F_CN+sum(log_F_dev_CN(endyr-2,endyr)))/3.0);
 F_temp_sum+=mfexp((3.0*log_avg_F_MRFSS+sum(log_F_dev_MRFSS(endyr-2,endyr)))/3.0);

 F_temp_sum+=mfexp((3.0*log_avg_F_HL_D+sum(log_F_dev_HL_D(endyr-2,endyr)))/3.0);
 F_temp_sum+=mfexp((3.0*log_avg_F_GN_D+sum(log_F_dev_GN_D(endyr-2,endyr)))/3.0);
 F_temp_sum+=mfexp((3.0*log_avg_F_MRFSS_D+sum(log_F_dev_MRFSS_D(endyr-2,endyr)))/3.0);

 F_temp_sum+=mfexp((3.0*log_avg_F_shrimp+sum(log_F_dev_shrimp(endyr-2,endyr)))/3.0);

 F_HL_prop=mfexp((3.0*log_avg_F_HL+sum(log_F_dev_HL(endyr-2,endyr)))/3.0)/F_temp_sum;
 F_PN_prop=mfexp((3.0*log_avg_F_PN+sum(log_F_dev_PN(endyr-2,endyr)))/3.0)/F_temp_sum;
 F_GN_prop=mfexp((3.0*log_avg_F_GN+sum(log_F_dev_GN(endyr-2,endyr)))/3.0)/F_temp_sum;
 F_CN_prop=mfexp((3.0*log_avg_F_CN+sum(log_F_dev_CN(endyr-2,endyr)))/3.0)/F_temp_sum;
 F_MRFSS_prop=mfexp((3.0*log_avg_F_MRFSS+sum(log_F_dev_MRFSS(endyr-2,endyr)))/3.0)/F_temp_sum;

 F_HL_D_prop=mfexp((3.0*log_avg_F_HL_D+sum(log_F_dev_HL_D(endyr-2,endyr)))/3.0)/F_temp_sum;
 F_GN_D_prop=mfexp((3.0*log_avg_F_GN_D+sum(log_F_dev_GN_D(endyr-2,endyr)))/3.0)/F_temp_sum;
 F_MRFSS_D_prop=mfexp((3.0*log_avg_F_MRFSS_D+sum(log_F_dev_MRFSS_D(endyr-2,endyr)))/3.0)/F_temp_sum;

 F_shrimp_prop=mfexp((3.0*log_avg_F_shrimp+sum(log_F_dev_shrimp(endyr-2,endyr)))/3.0)/F_temp_sum;


sel_wgted_L_F=F_HL_prop*sel_HL_keep_F+F_PN_prop*sel_PN_F+F_GN_prop*sel_GN_keep_F2+F_CN_prop*sel_CN_F+F_MRFSS_prop*sel_MR
FSS_keep_F;
 sel_wgted_D_F=F_HL_D_prop*sel_HL_D_F+F_GN_D_prop*sel_GN_D_F+F_MRFSS_D_prop*sel_MRFSS_D_F+F_shrimp_prop*sel_shrimp;
 sel_wgted_tot_F=sel_wgted_L_F+sel_wgted_D_F;
```

```
sel_wgted_L_M=F_HL_prop*sel_HL_keep_M+F_PN_prop*sel_PN_M+F_GN_prop*sel_GN_keep_M2+F_CN_prop*sel_CN_M+F_MRFSS_prop*sel
_MRFSS_keep_M;
 sel_wgted_D_M=F_HL_D_prop*sel_HL_D_M+F_GN_D_prop*sel_GN_D_M+F_MRFSS_D_prop*sel_MRFSS_D_M+F_shrimp_prop*sel_shrimp;
 sel_wgted_tot_M=sel_wgted_L_M+sel_wgted_D_M;

 //max_sel_wgted_tot_F=max(sel_wgted_tot_F);
 //sel_wgted_tot_F/=max_sel_wgted_tot_F;
//sel_wgted_L_F/=max_sel_wgted_tot_F; //landings sel bumped up by same amount as total sel
 //sel_wgted_D_F/=max_sel_wgted_tot_F;

 //max_sel_wgted_tot_M=max(sel_wgted_tot_M);
 //sel_wgted_tot_M/=max_sel_wgted_tot_M;
 //sel_wgted_L_M/=max_sel_wgted_tot_M; //landings sel bumped up by same amount as total sel
 //sel_wgted_D_M/=max_sel_wgted_tot_M;

FUNCTION get_msy

 //fill in Fs for per-recruit stuff
 F_msy.fill_seqadd(0,.001);  //step size should be of inverse dimension to n_iter_msy

 //compute values as functions of F
 for(int ff=1; ff<=n_iter_msy; ff++){
  //int ff=1001;
  //uses fishery-weighted F's
  Z_age_msy_F=0.0;
  F_L_age_msy_F=0.0;
  F_D_age_msy_F=0.0;
  Z_age_msy_M=0.0;
  F_L_age_msy_M=0.0;
  F_D_age_msy_M=0.0;

  F_L_age_msy_F=F_msy(ff)*sel_wgted_L_F;
  F_D_age_msy_F=F_msy(ff)*sel_wgted_D_F;
  Z_age_msy_F=M+F_L_age_msy_F+F_D_age_msy_F;
  F_L_age_msy_M=F_msy(ff)*sel_wgted_L_M;
  F_D_age_msy_M=F_msy(ff)*sel_wgted_D_M;
  Z_age_msy_M=M+F_L_age_msy_M+F_D_age_msy_M;

  N_age_msy_F(1)=prop_f_a0;
  for (iage=2; iage<=nages; iage++)
  {
   N_age_msy_F(iage)=N_age_msy_F(iage-1)*mfexp(-1.*Z_age_msy_F(iage-1));
  }
  N_age_msy_F(nages)=N_age_msy_F(nages)/(1.0-mfexp(-1.*Z_age_msy_F(nages)));
  N_age_msy_mdyr_F(1,(nages-1))=elem_prod(N_age_msy_F(1,(nages-1)),
                        mfexp((-1.*Z_age_msy_F(1,(nages-1)))/2.0));
  N_age_msy_mdyr_F(nages)=(N_age_msy_mdyr_F(nages-1)*
                  (mfexp(-1.*(Z_age_msy_F(nages-1)/2 + Z_age_msy_F(nages)/2) )))
                  /(1.0-mfexp(-1.*Z_age_msy_F(nages)));

  N_age_msy_M(1)=1.-prop_f_a0;
  for (iage=2; iage<=nages; iage++)
  {
   N_age_msy_M(iage)=N_age_msy_M(iage-1)*mfexp(-1.*Z_age_msy_M(iage-1));
  }
  N_age_msy_M(nages)=N_age_msy_M(nages)/(1.0-mfexp(-1.*Z_age_msy_M(nages)));
  N_age_msy_mdyr_M(1,(nages-1))=elem_prod(N_age_msy_M(1,(nages-1)),
                        mfexp((-1.*Z_age_msy_M(1,(nages-1)))/2.0));
  N_age_msy_mdyr_M(nages)=(N_age_msy_mdyr_M(nages-1)*
                  (mfexp(-1.*(Z_age_msy_M(nages-1)/2 + Z_age_msy_M(nages)/2) )))
                  /(1.0-mfexp(-1.*Z_age_msy_M(nages)));

  for(iage=1;iage<=nages;iage++){
   prop_f_F0(iage)=N_age_msy_F(iage)/(N_age_msy_F(iage)+N_age_msy_M(iage));  //not really at F=0 here; just using vector
  }

  spr_msy(ff)=sum(elem_prod(N_age_msy_mdyr_F,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_F*=R_eq(ff); //proportion female/male already accounted for
  N_age_msy_mdyr_F*=R_eq(ff);
  N_age_msy_M*=R_eq(ff);
  N_age_msy_mdyr_M*=R_eq(ff);

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

```
    C_age_msy_F(iage)=N_age_msy_F(iage)*(F_L_age_msy_F(iage)/Z_age_msy_F(iage))*
                (1.-mfexp(-1.0*Z_age_msy_F(iage)));
    C_age_msy_M(iage)=N_age_msy_M(iage)*(F_L_age_msy_M(iage)/Z_age_msy_M(iage))*
                (1.-mfexp(-1.0*Z_age_msy_M(iage)));
    D_age_msy_F(iage)=N_age_msy_F(iage)*(F_D_age_msy_F(iage)/Z_age_msy_F(iage))*
                (1.-mfexp(-1.0*Z_age_msy_F(iage)));
    D_age_msy_M(iage)=N_age_msy_M(iage)*(F_D_age_msy_M(iage)/Z_age_msy_M(iage))*
                (1.-mfexp(-1.0*Z_age_msy_M(iage)));
   }

   SSB_eq(ff)=sum(elem_prod(N_age_msy_mdyr_F,reprod));
   B_eq(ff)=sum(elem_prod(N_age_msy_F,wgt_f))+sum(elem_prod(N_age_msy_M,wgt_m));
   L_eq(ff)=sum(elem_prod(C_age_msy_F,wgt_f))+sum(elem_prod(C_age_msy_M,wgt_m));
   E_eq(ff)=(sum(C_age_msy_F(E_age_st,nages))+sum(C_age_msy_M(E_age_st,nages)));
   E_eq(ff)/=(sum(N_age_msy_F(E_age_st,nages))+sum(N_age_msy_M(E_age_st,nages)));
   D_eq(ff)=(sum(D_age_msy_F)+sum(D_age_msy_M))/1000.0;

  }

  msy_out=max(L_eq);

  for(ff=1; ff<=n_iter_msy; ff++)
  {
   if(L_eq(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);
       E_msy_out=E_eq(ff);
       F_msy_out=F_msy(ff);
       spr_msy_out=spr_msy(ff);
     }
  }

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

  //compute total catch-at-age and landings
  C_total=C_HL_F+C_HL_M+C_PN_F+C_PN_M+C_GN_M+C_GN_F+C_CN_M+C_CN_F+C_MRFSS_M+C_MRFSS_F; //catch in number fish
  L_total=L_HL+L_PN+L_GN+L_CN+L_MRFSS;   //landings in mt whole weight
  D_total=L_HL_D+L_MRFSS_D+L_GN_D; //total discards in mt whole weight
  B_total=L_shrimp;

  //compute exploitation rate of age E_age_st +
  for(iyear=styr; iyear<=endyr; iyear++)
  {
   E(iyear)=(sum(C_total(iyear)(E_age_st,nages)))/(sum(N_F(iyear)(E_age_st,nages))+sum(N_M(iyear)(E_age_st,nages)));
   L_total_yr(iyear)=sum(L_total(iyear));
   B_total_yr(iyear)=sum(B_total(iyear));
   D_total_yr(iyear)=sum(D_total(iyear));
  }

  steep_sd=steep;
  fullF_sd=fullF;
  E_sd=E;

  if(E_msy_out>0)
    {
     EdE_msy=E/E_msy_out;
     EdE_msy_end=EdE_msy(endyr);
    }
  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_F(1)=prop_f_a0;
  for(iage=2; iage<=nages; iage++)
  {
   N_age_spr_F(iage)=N_age_spr_F(iage-1)*mfexp(-1.*Z_F(iyear,iage-1));
  }
  N_age_spr_F(nages)=N_age_spr_F(nages)/(1.0-mfexp(-1.*Z_F(iyear,nages)));
  N_age_spr_mdyr_F(1,(nages-1))=elem_prod(N_age_spr_F(1,(nages-1)),
                          mfexp(-1.*Z_F(iyear)(1,(nages-1))/2.0));
  N_age_spr_mdyr_F(nages)=(N_age_spr_mdyr_F(nages-1)*
                    (mfexp(-1.*(Z_F(iyear)(nages-1)/2.0 + Z_F(iyear)(nages)/2.0) )))
                    /(1.0-mfexp(-1.*Z_age_msy_F(nages)));
  spr_static(iyear)=sum(elem_prod(N_age_spr_mdyr_F,reprod))/spr_F0;
 }

 //fill in Fs for per-recruit stuff
 F_spr.fill_seqadd(0,.01);

 //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_F=0.0;
  F_L_age_spr_F=0.0;
  Z_age_spr_M=0.0;
  F_L_age_spr_M=0.0;

  F_L_age_spr_F=F_spr(ff)*sel_wgted_L_F;
  F_L_age_spr_M=F_spr(ff)*sel_wgted_L_M;

  Z_age_spr_F=M+F_L_age_spr_F+F_spr(ff)*sel_wgted_D_F;
  Z_age_spr_M=M+F_L_age_spr_M+F_spr(ff)*sel_wgted_D_M;

  N_age_spr_F(1)=prop_f_a0;
  N_age_spr_M(1)=1.-prop_f_a0;
  for (iage=2; iage<=nages; iage++){
   N_age_spr_F(iage)=N_age_spr_F(iage-1)*mfexp(-1.*Z_age_spr_F(iage-1));
   N_age_spr_M(iage)=N_age_spr_M(iage-1)*mfexp(-1.*Z_age_spr_M(iage-1));
  }
  N_age_spr_F(nages)=N_age_spr_F(nages)/(1-mfexp(-1.*Z_age_spr_F(nages)));
  N_age_spr_mdyr_F(1,(nages-1))=elem_prod(N_age_spr_F(1,(nages-1)),
                          mfexp((-1.*Z_age_spr_F(1,(nages-1)))/2.0));
  N_age_spr_mdyr_F(nages)=(N_age_spr_mdyr_F(nages-1)*
                    (mfexp(-1.*(Z_age_spr_F(nages-1)/2 + Z_age_spr_F(nages)/2) )))
                    /(1.0-mfexp(-1.*Z_age_spr_F(nages)));

  for(iage=1;iage<=nages;iage++){
   prop_f_F0(iage)=N_age_spr_F(iage)/(N_age_spr_F(iage)+N_age_spr_M(iage));  //not really at F=0 here; just using vector
  }
  spr_spr(ff)=sum(elem_prod(N_age_spr_mdyr_F,reprod));
  L_spr(ff)=0.0;
  for (iage=1; iage<=nages; iage++)
  {
   C_age_spr_F(iage)=N_age_spr_F(iage)*(F_L_age_spr_F(iage)/Z_age_spr_F(iage))*
                (1.-mfexp(-1.*Z_age_spr_F(iage)));
   C_age_spr_M(iage)=N_age_spr_M(iage)*(F_L_age_spr_M(iage)/Z_age_spr_M(iage))*
                (1.-mfexp(-1.*Z_age_spr_M(iage)));
   L_spr(ff)+=(C_age_spr_M(iage)*wgt_m(iage)+C_age_spr_F(iage)*wgt_f(iage));
  }
  E_spr(ff)=(sum(C_age_spr_M(E_age_st,nages))+sum(C_age_spr_F(E_age_st,nages)))/
          (sum(N_age_spr_M(E_age_st,nages))+sum(N_age_spr_F(E_age_st,nages)));

 }

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

FUNCTION evaluate_objective_function
 fval=0.0;
 fval_unwgt=0.0;

//---likelihoods----------------------------
 //fval=square(x_dum-3.0);
```

```
//---Indices-------------------------------
 f_LB_HL_cpue=0.0;
 for (iyear=styr_LB_HL_cpue; iyear<=endyr_LB_HL_cpue; iyear++)
 {
  f_LB_HL_cpue+=square(log((pred_LB_HL_cpue(iyear)+dzero_dum)/
     (obs_LB_HL_cpue(iyear)+dzero_dum)))/(2.0*square(LB_HL_cpue_cv(iyear)));
 }
 fval+=w_I_LB_HL*f_LB_HL_cpue;
 fval_unwgt+=f_LB_HL_cpue;

 f_FL_HL_cpue=0.0;
 for (iyear=styr_FL_HL_cpue; iyear<=endyr_FL_HL_cpue; iyear++)
 {
  f_FL_HL_cpue+=square(log((pred_FL_HL_cpue(iyear)+dzero_dum)/
     (obs_FL_HL_cpue(iyear)+dzero_dum)))/(2.0*square(FL_HL_cpue_cv(iyear)));
 }
 fval+=w_I_FL_HL*f_FL_HL_cpue;
 fval_unwgt+=f_FL_HL_cpue;

  f_FL_gill1_cpue=0.0;
 for (iyear=styr_FL_gill1_cpue; iyear<=endyr_FL_gill1_cpue; iyear++){
  f_FL_gill1_cpue+=square(log((pred_FL_gill1_cpue(iyear)+dzero_dum)/
     (obs_FL_gill1_cpue(iyear)+dzero_dum)))/(2.0*square(FL_gill1_cpue_cv(iyear)));
 }
 fval+=w_I_FL_gill1*f_FL_gill1_cpue;
 fval_unwgt+=f_FL_gill1_cpue;

 f_FL_gill2_cpue=0.0;
 for (iyear=styr_FL_gill2_cpue; iyear<=endyr_FL_gill2_cpue; iyear++){
  f_FL_gill2_cpue+=square(log((pred_FL_gill2_cpue(iyear)+dzero_dum)/
     (obs_FL_gill2_cpue(iyear)+dzero_dum)))/(2.0*square(FL_gill2_cpue_cv(iyear)));
 }
 fval+=w_I_FL_gill2*f_FL_gill2_cpue;
 fval_unwgt+=f_FL_gill2_cpue;

 f_LB_gill_cpue=0.0;
 for (iyear=styr_LB_gill_cpue; iyear<=endyr_LB_gill_cpue; iyear++)
 {
  f_LB_gill_cpue+=square(log((pred_LB_gill_cpue(iyear)+dzero_dum)/
     (obs_LB_gill_cpue(iyear)+dzero_dum)))/(2.0*square(LB_gill_cpue_cv(iyear)));
 }
 fval+=w_I_LB_gill*f_LB_gill_cpue;
 fval_unwgt+=f_LB_gill_cpue;

 f_CN_cpue=0.0;
 for (iyear=styr_CN_cpue; iyear<=endyr_CN_cpue; iyear++)
 {
  f_CN_cpue+=square(log((pred_CN_cpue(iyear)+dzero_dum)/
     (obs_CN_cpue(iyear)+dzero_dum)))/(2.0*square(CN_cpue_cv(iyear)));
 }
 fval+=w_I_CN*f_CN_cpue;
 fval_unwgt+=f_CN_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)));
 }
 //cout<<w_I_MRFSS<<endl<<f_MRFSS_cpue<<endl<<endl;
 fval+=w_I_MRFSS*f_MRFSS_cpue;
 fval_unwgt+=f_MRFSS_cpue;

 f_SMAP_YOY_cpue=0.0;
 for (iyear=styr_SMAP_YOY_cpue; iyear<=endyr_SMAP_YOY_cpue; iyear++)
 {
  f_SMAP_YOY_cpue+=square(log((pred_SMAP_YOY_cpue(iyear)+dzero_dum)/
     (obs_SMAP_YOY_cpue(iyear)+dzero_dum)))/(2.0*square(SMAP_YOY_cpue_cv(iyear)));
 }
 fval+=w_I_SMAP_YOY*f_SMAP_YOY_cpue;
 fval_unwgt+=f_SMAP_YOY_cpue;

 f_SMAP_1YR_cpue=0.0;
 for (iyear=styr_SMAP_1YR_cpue; iyear<=endyr_SMAP_1YR_cpue; iyear++)
 {
  f_SMAP_1YR_cpue+=square(log((pred_SMAP_1YR_cpue(iyear)+dzero_dum)/
     (obs_SMAP_1YR_cpue(iyear)+dzero_dum)))/(2.0*square(SMAP_1YR_cpue_cv(iyear)));
 }
 fval+=w_I_SMAP_1YR*f_SMAP_1YR_cpue;
```

```
  fval_unwgt+=f_SMAP_1YR_cpue;

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

  f_HL_L=0.0; //in 1000s total pounds
  for (iyear=styr_HL_L; iyear<=endyr_HL_L; iyear++)
  {
     f_HL_L+=square(log((pred_HL_L(iyear)+dzero_dum)/
       (obs_HL_L(iyear)+dzero_dum)))/(2.0*square(HL_L_cv(iyear)));
  }
  fval+=w_L*f_HL_L;
  fval_unwgt+=f_HL_L;

  f_PN_L=0.0; //in 1000s total pounds
  for (iyear=styr_PN_L; iyear<=endyr_PN_L; iyear++)
  {
      f_PN_L+=square(log((pred_PN_L(iyear)+dzero_dum)/
       (obs_PN_L(iyear)+dzero_dum)))/(2.0*square(PN_L_cv(iyear)));
  }
  fval+=w_L*f_PN_L;
  fval_unwgt+=f_PN_L;

  f_GN_L=0.0; //in 1000s total pounds
  for (iyear=styr_GN_L; iyear<=endyr_GN_L; iyear++)
  {
     f_GN_L+=square(log((pred_GN_L(iyear)+dzero_dum)/
       (obs_GN_L(iyear)+dzero_dum)))/(2.0*square(GN_L_cv(iyear)));
  }
  fval+=w_L*f_GN_L;
  fval_unwgt+=f_GN_L;

  f_CN_L=0.0; //in 1000s total pounds
  for (iyear=styr_CN_L; iyear<=endyr_CN_L; iyear++)
  {
     f_CN_L+=square(log((pred_CN_L(iyear)+dzero_dum)/
       (obs_CN_L(iyear)+dzero_dum)))/(2.0*square(CN_L_cv(iyear)));
  }
  fval+=w_L*f_CN_L;
  fval_unwgt+=f_CN_L;

  f_MRFSS_L=0.0; //in 1000s - numbers
  for (iyear=styr_MRFSS_L; iyear<=endyr_MRFSS_L; iyear++)
  {
     f_MRFSS_L+=square(log((pred_MRFSS_L(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 & Bycatch--------------------------------

  f_HL_D=0.0; //in 1000s
  for (iyear=styr_HL_D; iyear<=endyr_HL_D; iyear++)
  {
     f_HL_D+=square(log((pred_HL_D(iyear)+dzero_dum)/
       (obs_HL_D(iyear)+dzero_dum)))/(2.0*square(HL_D_cv(iyear)));
  }
  fval+=w_D*f_HL_D;
  fval_unwgt+=f_HL_D;

  f_GN_D=0.0; //in 1000s (numbers)
  for (iyear=styr_GN_D; iyear<=endyr_GN_D; iyear++)
  {
     f_GN_D+=square(log((pred_GN_D(iyear)+dzero_dum)/
       (obs_GN_D(iyear)+dzero_dum)))/(2.0*square(GN_D_cv(iyear)));
  }
  fval+=w_D*f_GN_D;
  fval_unwgt+=f_GN_D;

  f_MRFSS_D=0.0; //in 1000s (numbers)
  for (iyear=styr_MRFSS_D; iyear<=endyr_MRFSS_D; iyear++)
  {
     f_MRFSS_D+=square(log((pred_MRFSS_D(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;

  f_shrimp_B=0.0;
  for (iyear=styr_shrimp_B; iyear<=endyr_shrimp_B; iyear++)
```

```
  {
     f_shrimp_B+=square(log((pred_shrimp_B(iyear)+dzero_dum)/
       (obs_shrimp_B(iyear)+dzero_dum)))/(2.0*square(shrimp_B_cv(iyear)));
  }
  fval+=w_D*f_shrimp_B;
  fval_unwgt+=f_shrimp_B;

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

  f_HL_lenc=0.0;
  for (iyear=1; iyear<=nyr_HL_lenc; iyear++)
  {
   f_HL_lenc-=nsamp_HL_lenc(iyear)*
      sum( elem_prod((obs_HL_lenc(iyear)+dzero_dum),
         log(elem_div((pred_HL_lenc(iyear)+dzero_dum),
           (obs_HL_lenc(iyear)+dzero_dum)))));
  }
  fval+=w_lc*f_HL_lenc;
  fval_unwgt+=f_HL_lenc;

  f_PN_lenc=0.0;
  for (iyear=styr_PN_lenc; iyear<=endyr_PN_lenc; iyear++)
  {
   f_PN_lenc-=nsamp_PN_lenc(iyear)*
      sum( elem_prod((obs_PN_lenc(iyear)+dzero_dum),
         log(elem_div((pred_PN_lenc(iyear)+dzero_dum),
           (obs_PN_lenc(iyear)+dzero_dum)))));
  }
  fval+=w_lc*f_PN_lenc;
  fval_unwgt+=f_PN_lenc;

  f_GN_lenc=0.0;
  for (iyear=styr_GN_lenc; iyear<=endyr_GN_lenc; iyear++)
  {
   f_GN_lenc-=nsamp_GN_lenc(iyear)*
      sum( elem_prod((obs_GN_lenc(iyear)+dzero_dum),
         log(elem_div((pred_GN_lenc(iyear)+dzero_dum),
           (obs_GN_lenc(iyear)+dzero_dum)))));
  }
  fval+=w_lc*f_GN_lenc;
  fval_unwgt+=f_GN_lenc;

  f_CN_lenc=0.;
  for (iyear=1; iyear<=nyr_CN_lenc; iyear++)
  {
   f_CN_lenc-=nsamp_CN_lenc(iyear)*
       sum(elem_prod((obs_CN_lenc(iyear)+dzero_dum),
          log(elem_div((pred_CN_lenc(iyear)+dzero_dum),
            (obs_CN_lenc(iyear)+dzero_dum)))));
  }
  fval+=w_lc*f_CN_lenc;
  fval_unwgt+=f_CN_lenc;

  f_MRFSS_lenc=0.;
  for (iyear=styr_MRFSS_lenc; iyear<=endyr_MRFSS_lenc; iyear++)
  {
   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_HL_agec=0.0;
  for (iyear=1; iyear<=nyr_HL_agec; iyear++)
  {
   f_HL_agec-=nsamp_HL_agec(iyear)*
      sum( elem_prod((obs_HL_agec(iyear)+dzero_dum),
         log(elem_div((pred_HL_agec(iyear)+dzero_dum),
           (obs_HL_agec(iyear)+dzero_dum)))));
  }
  fval+=w_ac*f_HL_agec;
  fval_unwgt+=f_HL_agec;

  f_PN_agec=0.0;
  for (iyear=1; iyear<=nyr_PN_agec; iyear++)
  {
   f_PN_agec-=nsamp_PN_agec(iyear)*
```

```
      sum( elem_prod((obs_PN_agec(iyear)+dzero_dum),
         log(elem_div((pred_PN_agec(iyear)+dzero_dum),
           (obs_PN_agec(iyear)+dzero_dum)))));
  }
  fval+=w_ac*f_PN_agec;
  fval_unwgt+=f_PN_agec;

  f_GN_agec=0.0;
  for (iyear=styr_GN_agec; iyear<=endyr_GN_agec; iyear++)
  {
   f_GN_agec-=nsamp_GN_agec(iyear)*
      sum( elem_prod((obs_GN_agec(iyear)+dzero_dum),
         log(elem_div((pred_GN_agec(iyear)+dzero_dum),
           (obs_GN_agec(iyear)+dzero_dum)))));
  }
  fval+=w_ac*f_GN_agec;
  fval_unwgt+=f_GN_agec;

  f_CN_agec=0.0;
  for (iyear=1; iyear<=nyr_CN_agec; iyear++)
  {
   f_CN_agec-=nsamp_CN_agec(iyear)*
      sum( elem_prod((obs_CN_agec(iyear)+dzero_dum),
         log(elem_div((pred_CN_agec(iyear)+dzero_dum),
           (obs_CN_agec(iyear)+dzero_dum)))));
  }
  fval+=w_ac*f_CN_agec;
  fval_unwgt+=f_CN_agec;

  f_MRFSS_agec=0.0;
  for (iyear=styr_MRFSS_agec; iyear<=endyr_MRFSS_agec; iyear++)
  {
   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 3 yrs
   f_N_dev_end=norm2(log_dev_N_rec(endyr-2,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;
  //cout << "avg MRFSS " << log_avg_F_MRFSS <<endl;;

  //cout<<"pred ac"<<endl<<pred_HL_agec<<endl<<endl<<"obs ac"<<endl<<endl<<obs_HL_agec;

REPORT_SECTION
  cout<<"start report"<<endl;
  get_sel_weighted_current();
  cout<<"got sel 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 << "BC Fmsy=" << F_msy_out<< "   BC SSBmsy=" << SSB_msy_out <<endl;
  cout<<"Pop status="<<SSB(endyr)/SSB_msy_out<<endl;
  cout<<"SSB last year "<<SSB(endyr);
  cout<<"SSB msy "<<SSB_msy_out<<endl;
  cout << "var_rec_resid="<<var_rec_dev<<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 Emsy " << E_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 E/Emsy " << E/E_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_yr/msy_out <<endl;
  report << "BC F(2006)/Fmsy " << fullF(endyr)/F_msy_out << endl;
  report << "BC E(2006)/Emsy " << E(endyr)/E_msy_out << endl;
  report << "BC SSB(2006)/SSBmsy " << SSB(endyr)/SSB_msy_out << endl;
  report << "BC Predicted Landings(2006)/MSY " << L_total_yr(endyr)/msy_out <<endl;
  report  << " "<<endl;

  report << "Mortality and growth" << endl;
  report << "M "<<M<<endl;
  report << "Linf,males="<<Linf_m << "   K, males=" <<K_m<<"   t0_males="<< t0_m<<endl;
  report << "mean length, females " << meanlen_f << endl;
  report << "Linf,males="<<Linf_f << "   K, males=" <<K_f<<"   t0_fales="<< t0_f<<endl;
  report << "mean length, males " << meanlen_m << endl;
  report << "cv length " << len_cv << endl;
  report << "wgt, males" << wgt_m << endl;
  report << "wgt, females" << wgt_f << endl;
  report<<" "<<endl;

  report << "Stock-Recruit " << endl;
  report << "R0= " << R0 << endl;
  report << "Steepness= " << steep << endl;
  report << "spr_F0= " << spr_F0 << endl;
  report << "Recruits(R) " << rec << endl;
  report << "VirginSSB " << S0 << endl;
  report << "SSB(styr)/VirginSSB " << S1S0 << endl;
  report << "SSB(2006)/VirginSSB " << popstatus << endl;
  report << "SSB " << SSB << endl;
  report << "Biomass " << totB << endl;
  report << "log recruit deviations (styr_rec_dev-2007)   " << log_dev_N_rec(styr_rec_dev,2007) <<endl;
  report << "variance of log rec dev (select yrs)  "<<var_rec_dev<<endl;
  report << "autocorrelation " << R_autocorr <<endl;
  report<<" "<<endl;

  report << "Exploitation rate (1958-2007)" << endl;
 report << E << endl;
 report << "Fully-selected F (1958-2007)" << endl;
 report << fullF << endl;
 report << "Comm hand lines F" << endl;
 report << F_HL_out << endl;
 report << "Poundnet F" << endl;
```

```
report << F_PN_out << endl;
report << "Gillnet F" << endl;
report << F_GN_out << endl;
report << "Castnet F" << endl;
report << F_CN_out << endl;
report << "MRFSS F" << endl;
report << F_MRFSS_out << endl;
report << "Shrimp Bycatch F"<<endl;
report << F_shrimp_out << endl;

report << "selpar_L50_HL_keep" <<endl;
report << selpar_L50_HL_keep <<endl;
report << "selpar_slope_HL" <<endl;
report << selpar_slope_HL <<endl;
report << "selpar_L50_PN" <<endl;
report << selpar_L50_PN <<endl;
report << "selpar_slope_PN" <<endl;
report << selpar_slope_PN <<endl;
report << "selpar_L50_GN_keep" <<endl;
report << selpar_L50_GN_keep <<endl;
report << "selpar_slope_GN" <<endl;
report << selpar_slope_GN <<endl;
report << "selpar_L50_CN" <<endl;
report << selpar_L50_CN <<endl;
report << "selpar_slope_CN" <<endl;
report << selpar_slope_CN <<endl;
report << "selpar_L50_MRFSS_keep" <<endl;
report << selpar_L50_MRFSS_keep <<endl;
report << "selpar_slope_MRFSS" <<endl;
report << selpar_slope_MRFSS <<endl;
report << "Hand lines selectivity - females" << endl;
report << sel_HL_keep_F << endl;
report << "Hand lines selectivity - males" << endl;
report << sel_HL_keep_M << endl;
report << "Handline DISCARD selectivity - females" << endl;
report << sel_HL_D_F << endl;
report << "Handline DISCARD selectivity - males" << endl;
report << sel_HL_D_M << endl;
report << "Poundnet selectivity - females" << endl;
report << sel_PN_F << endl;
report << "Poundnet selectivity - males" << endl;
report << sel_PN_M << endl;
report << "Gillnet selectivity - pre-1995- females" << endl;
report << sel_GN_keep_F << endl;
report << "Gillnet selectivity - pre-1995- males" << endl;
report << sel_GN_keep_M << endl;
report << "Gillnet selectivity - post-1995- females" << endl;
report << sel_GN_keep_F2 << endl;
report << "Gillnet selectivity - post-1995- males" << endl;
report << sel_GN_keep_M2 << endl;
report << "Castnet selectivity - females" << endl;
report << sel_CN_F << endl;
report << "Castnet selectivity - males" << endl;
report << sel_CN_M << endl;
report << "MRFSS selectivity - females" << endl;
report << sel_MRFSS_keep_F << endl;
report << "MRFSS selectivity - males" << endl;
report << sel_MRFSS_keep_M << endl;

report << "mean log F - HL" << endl;
report << log_avg_F_HL << endl;
report << "log F deviations - HL" << endl;
report << log_F_dev_HL << endl;
report << "mean log F - PN" << endl;
report << log_avg_F_PN << endl;
report << "log F deviations - PN" << endl;
report << log_F_dev_PN << endl;
report << "mean log F - GN" << endl;
report << log_avg_F_GN << endl;
report << "log F deviations - GN" << endl;
report << log_F_dev_GN << endl;
report << "mean log F - CN" << endl;
report << log_avg_F_CN << endl;
report << "log F deviations - CN" << endl;
report << log_F_dev_CN << endl;
report << "mean log F - MRFSS" << endl;
report << log_avg_F_MRFSS << endl;
report << "log F deviations - MRFSS" << endl;
report << log_F_dev_MRFSS << endl;
report << "mean log F - shrimp" << endl;
```

```
report << log_avg_F_shrimp << endl;
report << "log F deviations - shrimp" << endl;
report << log_F_dev_shrimp << endl;

report << "mean log F - HL - Discards" << endl;
report << log_avg_F_HL_D << endl;
report << "log F deviations - HL - Discards" << endl;
report << log_F_dev_HL_D << endl;
report << "mean log F - GN - Discards" << endl;
report << log_avg_F_GN_D << endl;
report << "log F deviations - GN - Discards" << endl;
report << log_F_dev_GN_D << endl;
report << "mean log F - MRFSS - Discards" << endl;
report << log_avg_F_MRFSS_D << endl;
report << "log F deviations - MRFSS - Discards" << endl;
report << log_F_dev_MRFSS_D << endl;

report << "Obs  LB_HL U "<<obs_LB_HL_cpue << endl;
report << "pred LB_HL U "<<pred_LB_HL_cpue << endl;
report << "Obs  FL_HL U "<<obs_FL_HL_cpue << endl;
report << "pred FL_HL U "<<pred_FL_HL_cpue << endl;
report << "Obs  FL gill1 U "<<obs_FL_gill1_cpue << endl;
report << "pred FL gill1 U "<<pred_FL_gill1_cpue << endl;
report << "Obs  FL gill2 U "<<obs_FL_gill2_cpue << endl;
report << "pred FL gill2 U "<<pred_FL_gill2_cpue << endl;
report << "Obs  LB_gill U "<<obs_LB_gill_cpue << endl;
report << "pred LB_gill U "<<pred_LB_gill_cpue << endl;
report << "Obs  CN U "<<obs_CN_cpue << endl;
report << "pred CN U "<<pred_CN_cpue << endl;
report << "Obs  MRFSS U "<<obs_MRFSS_cpue << endl;
report << "pred MRFSS U "<<pred_MRFSS_cpue << endl;
report << "Obs  SMAP_YOY U "<<obs_SMAP_YOY_cpue << endl;
report << "pred SMAP_YOY U "<<pred_SMAP_YOY_cpue << endl;
report << "Obs  SMAP_1YR U "<<obs_SMAP_1YR_cpue << endl;
report << "pred SMAP_1YR U "<<pred_SMAP_1YR_cpue << endl;

report << "Obs  HL landings (1000 lb) "<<obs_HL_L << endl;
report << "pred HL landings (1000 lb) "<<pred_HL_L << endl;
report << "Obs  PN landings (1000 lb) "<<obs_PN_L << endl;
report << "pred PN landings (1000 lb) "<<pred_PN_L << endl;
report << "Obs  GN landings (1000 lb) "<<obs_GN_L << endl;
report << "pred GN landings (1000 lb) "<<pred_GN_L << endl;
report << "Obs  CN landings (1000 lb) "<<obs_CN_L << endl;
report << "pred CN landings (1000 lb) "<<pred_CN_L << endl;
report << "Obs  MRFSS landings (1000's) "<<obs_MRFSS_L << endl;
report << "pred MRFSS landings (1000's) "<<pred_MRFSS_L << endl;
report << "Obs shrimp bycatch (1000's) "<<obs_shrimp_B<<endl;
report << "pred shrimp bycatch (1000's) "<<pred_shrimp_B<<endl;

report << "Obs HL discards (1000's) "<<obs_HL_D<<endl;
report << "pred HL discards (1000's) "<<pred_HL_D<<endl;
report << "Obs GN discards (1000's) "<<obs_GN_D<<endl;
report << "pred GN discards (1000's) "<<pred_GN_D<<endl;
report << "Obs MRFSS discards (1000's) "<<obs_MRFSS_D<<endl;
report << "pred MRFSS discards (1000's) "<<pred_MRFSS_D<<endl;

#include "sm_make_Robject1.cxx"   // write the S-compatible report
```

II. Data input file (SMB1.dat). Be aware that the following contains wrap-around text that ADMB may require as single-line input.

```
##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>
##  Data Input File
##  SEDAR17 Assessment: Spanish Mackerel
##
##--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>--><>

#starting and ending year of model
1950
2007
#Starting year to estimate recruitment deviation from S-R curve
1982
##Number of ages (16 classes is 0,1,...9,10+)
11
##vector of agebins, last is a plus group
0 1 2 3 4 5 6 7 8 9 10
#number length bins 11cm to 70+
60
#Vector of length bins (cm)(midpoint of bin) - last value = plus group
11.5 12.5 13.5 14.5 15.5 16.5 17.5 18.5 19.5 20.5 21.5 22.5 23.5 24.5 25.5 26.5
27.5 28.5 29.5 30.5 31.5 32.5 33.5 34.5 35.5 36.5 37.5 38.5 39.5 40.5 41.5 42.5
43.5 44.5 45.5 46.5 47.5 48.5 49.5 50.5 51.5 52.5 53.5 54.5 55.5 56.5 57.5 58.5
59.5 60.5 61.5 62.5 63.5 64.5 65.5 66.5 67.5 68.5 69.5 70.5
#discard mortality constant
0.88 #HL
1.0 #GN
0.88 #MRFSS
#initial proportion female (age 0)
0.5
#number of iterations in spr calculations (max F examined is (value-1)/100
201
#number of iterations in msy calculations (max F examined is (value-1)/1000
2001
#starting age for exploitation rate
1
#multiplicative bias correction (may set to 1.0 for none or negative to compute from rec variance)
-1.0
#starting values for VonBert params for males (Linf, K, t0), units in mm FL
520.5
0.4727
-1.2308
#starting values for VonBert params for females (Linf, K, t0), units in mm FL ***** NEED TO BE UPDATED
628.7
0.3599
-1.355
#starting value of constant cv of length at age ******was 0.1 for the RS assessment
0.2
#length-weight (whole wgt) coefficients a and b, W=aL^b, (W=Kg, L=mm)
1.52E-05
2.91
#weight-weight conversion (gutted wgt=a*whole weight) NOT CURRENTLY USED
#time-variant vector of % maturity-at-age for females (ages 0-10)
0.0     0.939   1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
#time-variant vector of proportion female (ages 0-10+) NOT CURRENTLY USED
#0.5     0.5     0.5     0.5     0.5     0.5     0.5     0.5     0.5     0.5     0.5
################################Commercial Hand lines fishery#########################################################
#starting/ending years for FL Trip ticket index
1985
2007
#CPUE and CV
0.56 0.75 0.90 1.08 1.06 0.97 0.80 0.82 0.83 0.98 0.87 0.98 0.93 1.02 1.06 1.08 1.22 1.25 1.32 1.21 1.18 1.09 1.04
0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15 0.15
#starting/ending years for logbook index (North of FL)
1998
2007
#CPUE and CV
0.87 1.12 0.88 0.97 1.19 0.93 1.00 0.86 1.16 0.80
0.14 0.15 0.16 0.15 0.18 0.19 0.21 0.14 0.19 0.15
#Starting and ending years of landings time series, respectively
1958
2007
#Observed landings (1000 lb whole weight)  and CV's
10 9 25 20 76 54 103 153 173 142 123 103 127 119 134 162 283 623 582 125 44 50 50 37 91 30 50 59 56
116 104 142 250 285 73 61 69 200 83 93 176 202 278 419 362 416 761 698 839 753
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 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/ending years for HL discards
1986
2007
#Observed discards (1000's) and CV's
0.4 0.8 1.0 1.4 2.2 2.9 0.5 0.7 0.5 3.3 0.2 0.8 4.0 3.5 3.7 3.9 4.5 4.0 2.7 2.6 3.0 3.1
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 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 handlines)
19
1986 1987 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
#sample sizes of length comp data by year
24 45 33 161 202 191 73 31 102 98 775 2879 2507 4314 3230 762 225 469 92
#Handline length comp samples (year,age) (1cm length bins)

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	0.0107	0.0000	0.0000	0.0000	0.0000	0.0000	0.0107	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0107	0.0000	0.0107	0.2074	0.1967	0.0000	0.0000	0.0000
	0.2074	0.0213	0.0000	0.0000	0.0107	0.0000	0.0000	0.0107	0.0000	0.0000	0.0639
	0.1967	0.0000	0.0320	0.0107							
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	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0222	0.0000	0.0000	0.0000	0.0222	0.0000	0.0000	0.0222
	0.0444	0.0222	0.0889	0.0444	0.0889	0.0667	0.0444	0.0222	0.0000	0.0667	0.0889
	0.0889	0.0222	0.0889	0.1556							
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	0.0000	0.0000	0.0965	0.0000	0.0000	0.0965	0.0000	0.0000
	0.0055	0.1020	0.0000	0.0000	0.0000	0.0110	0.0110	0.1020	0.0110	0.0000	0.0000
	0.1020	0.0055	0.0055	0.0165	0.0110	0.0110	0.2039	0.0000	0.0055	0.0000	0.1020
	0.0055	0.0000	0.0965	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	0.0060	0.0181	0.0362
	0.0302	0.0483	0.0906	0.0906	0.1027	0.0604	0.0302	0.0256	0.0242	0.0181	0.0302
	0.0121	0.0060	0.0000	0.0362	0.0135	0.0181	0.0000	0.0467	0.0210	0.0392	0.0256
	0.0452	0.0135	0.0121	0.0135	0.0271	0.0075	0.0121	0.0135	0.0060	0.0075	0.0000
	0.0060	0.0000	0.0060	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	0.0058	0.0000	0.0000
	0.0015	0.0131	0.0785	0.0698	0.0726	0.1425	0.0886	0.0741	0.0567	0.0597	0.0174
	0.0277	0.0290	0.0189	0.0305	0.0015	0.0102	0.0000	0.0131	0.0015	0.0174	0.0116
	0.0058	0.0015	0.0174	0.0116	0.0174	0.0058	0.0116	0.0232	0.0000	0.0000	0.0058
	0.0116	0.0000	0.0058	0.0407							
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	0.0150	0.0299	0.0399
	0.0150	0.0299	0.0100	0.0349	0.0249	0.0200	0.0249	0.0599	0.0399	0.0499	0.0499
	0.0599	0.0599	0.0649	0.0499	0.0200	0.0299	0.0200	0.0150	0.0299	0.0299	0.0200
	0.0100	0.0150	0.0150	0.0050	0.0050	0.0000	0.0000	0.0100	0.0100	0.0050	0.0000
	0.0000	0.0000	0.0117	0.0702							
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	0.0137	0.0000	0.0000	0.0000	0.0000	0.0137	0.0000	0.0137	0.0274	0.0000
	0.0000	0.0548	0.0274	0.0685	0.0274	0.0274	0.0548	0.0000	0.0137	0.0274	0.0137
	0.0137	0.0000	0.0274	0.0411	0.0411	0.0274	0.0137	0.0411	0.0137	0.0000	0.0411
	0.0000	0.0000	0.0137	0.0000	0.0000	0.0137	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0137	0.0137	0.3014							
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	0.0323	0.0000	0.0968	0.2258
	0.0323	0.0000	0.0968	0.1290	0.1290	0.1290	0.0968	0.0000	0.0000	0.0323	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	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
	0.0098	0.0000	0.0000	0.0000	0.0294	0.0000	0.0000	0.0000	0.0098	0.0000	0.0098
	0.0000	0.0000	0.0000	0.0098	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0196	0.0000	0.0000	0.0000	0.0098	0.0000	0.0000	0.0098
	0.0098	0.0000	0.0098	0.8725							
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	0.0102
	0.0102	0.0000	0.0204	0.0102	0.0204	0.0510	0.0306	0.0612	0.0612	0.0408	0.0306
	0.0306	0.0204	0.0918	0.0306	0.0306	0.0816	0.0204	0.0306	0.0306	0.0918	0.0306
	0.0612	0.0204	0.0102	0.0102	0.0102	0.0102	0.0102	0.0102	0.0000	0.0102	0.0000
	0.0102	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	0.0013	0.0000	0.0000	0.0000	0.0000	0.0026	0.0038	0.0051	0.0038
	0.0077	0.0077	0.0038	0.0115	0.0167	0.0500	0.0564	0.0756	0.0949	0.0872	0.0679
	0.0513	0.0385	0.0256	0.0256	0.0128	0.0192	0.0179	0.0205	0.0269	0.0258	0.0154
	0.0154	0.0154	0.0154	0.0103	0.0141	0.0320	0.0115	0.0308	0.0256	0.0103	0.0128
	0.0064	0.0090	0.0103	0.0051							

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 0.0014 0.0058 0.0096 0.0082 0.0175
0.0244 0.0326 0.0319 0.0395 0.0374 0.0463 0.0395 0.0450 0.0460 0.0439 0.0422
0.0429 0.0347 0.0374 0.0268 0.0278 0.0240 0.0271 0.0299 0.0299 0.0347 0.0330
0.0347 0.0285 0.0209 0.0185 0.0175 0.0127 0.0124 0.0079 0.0034 0.0031 0.0027
0.0151 0.0010 0.0010 0.0010
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 0.0004 0.0012 0.0032 0.0036 0.0071
0.0191 0.0365 0.0532 0.0647 0.0771 0.0874 0.0671 0.0536 0.0556 0.0425 0.0433
0.0365 0.0318 0.0344 0.0195 0.0171 0.0195 0.0330 0.0278 0.0278 0.0278 0.0203
0.0179 0.0127 0.0147 0.0163 0.0115 0.0060 0.0024 0.0028 0.0012 0.0024 0.0004
0.0008 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 0.0009 0.0023 0.0028
0.0100 0.0158 0.0281 0.0380 0.0487 0.0631 0.0580 0.0591 0.0577 0.0584 0.0531
0.0441 0.0450 0.0387 0.0410 0.0332 0.0299 0.0313 0.0269 0.0223 0.0274 0.0262
0.0274 0.0262 0.0195 0.0185 0.0123 0.0088 0.0067 0.0042 0.0056 0.0016 0.0014
0.0007 0.0005 0.0007 0.0042
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 0.0012 0.0015 0.0034 0.0090 0.0059 0.0105 0.0130 0.0124
0.0170 0.0272 0.0346 0.0510 0.0470 0.0547 0.0569 0.0541 0.0463 0.0420 0.0399
0.0448 0.0448 0.0470 0.0565 0.0535 0.0465 0.0365 0.0281 0.0229 0.0158 0.0151
0.0142 0.0117 0.0096 0.0083 0.0071 0.0022 0.0012 0.0025 0.0022 0.0006 0.0006
0.0000 0.0006 0.0000 0.0003
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 0.0013 0.0013 0.0013 0.0013 0.0000 0.0052 0.0066 0.0052 0.0039 0.0315
0.0564 0.0958 0.0958 0.1050 0.0787 0.0722 0.0577 0.0315 0.0249 0.0197 0.0328
0.0131 0.0092 0.0144 0.0105 0.0144 0.0276 0.0328 0.0341 0.0315 0.0157 0.0039
0.0092 0.0131 0.0066 0.0052 0.0052 0.0052 0.0052 0.0052 0.0013 0.0013 0.0013
0.0000 0.0013 0.0000 0.0039
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
0.0044 0.0000 0.0089 0.0178 0.0267 0.0356 0.0178 0.0400 0.0267 0.0356 0.0267
0.0622 0.0356 0.0711 0.0578 0.0356 0.0489 0.0267 0.0400 0.0356 0.0444 0.0356
0.0356 0.0311 0.0356 0.0222 0.0444 0.0311 0.0133 0.0133 0.0044 0.0222 0.0000
0.0044 0.0000 0.0044 0.0044
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 0.0043 0.0021 0.0000 0.0000 0.0000 0.0000 0.0043 0.0021 0.0085
0.0107 0.0320 0.0917 0.1195 0.1109 0.1109 0.0704 0.0853 0.0469 0.0661 0.0469
0.0320 0.0213 0.0277 0.0171 0.0107 0.0064 0.0171 0.0149 0.0064 0.0043 0.0021
0.0064 0.0000 0.0043 0.0043 0.0043 0.0021 0.0000 0.0000 0.0021 0.0016 0.0021
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 0.0000 0.0000
0.0000 0.0000 0.0000 0.0119 0.0000 0.0238 0.0119 0.0357 0.0238 0.0475 0.0357
0.0594 0.0832 0.0357 0.0594 0.0594 0.0358 0.0238 0.0475 0.0357 0.0594 0.0358
0.0000 0.0713 0.0713 0.0121 0.0238 0.0360 0.0475 0.0000 0.0121 0.0000 0.0002
0.0000 0.0002 0.0000 0.0000

#Number and vector of years of age compositions (comm hand line)
13
1989 1990 1992 1995 1996 1997 1998 1999 2000 2001 2002 2006 2007
#sample sizes of age comp data by year
62 38 79 25 41 35 84 130 93 246 26 153 25
#commercial handline age comp samples (year,age)
0.0000 0.4192 0.3892 0.1437 0.0240 0.0240 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0869 0.1915 0.2092 0.5124 0.0000 0.0000 0.0000 0.0000 0.0000
0.0027 0.3602 0.5196 0.0978 0.0109 0.0060 0.0027 0.0000 0.0000 0.0000 0.0000
0.6321 0.2925 0.0755 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0400 0.5600 0.2000 0.0800 0.0000 0.0800 0.0400 0.0000 0.0000 0.0000 0.0000
0.0604 0.1275 0.1879 0.4295 0.0940 0.0805 0.0000 0.0201 0.0000 0.0000 0.0000
0.1288 0.0522 0.2449 0.1010 0.2810 0.1307 0.0278 0.0337 0.0000 0.0000 0.0000
0.0000 0.0286 0.4705 0.2366 0.1548 0.0551 0.0258 0.0256 0.0000 0.0002 0.0028
0.0000 0.2730 0.3549 0.1969 0.0973 0.0514 0.0195 0.0068 0.0000 0.0000 0.0000
0.0236 0.2840 0.3540 0.1914 0.0828 0.0229 0.0266 0.0048 0.0100 0.0000 0.0000
0.1482 0.3924 0.1310 0.2931 0.0000 0.0353 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0222 0.1702 0.0988 0.2148 0.1852 0.1975 0.0840 0.0075 0.0124 0.0075
0.0000 0.0000 0.0000 0.2000 0.2000 0.6000 0.0000 0.0000 0.0000 0.0000 0.0000

####################################Pound nets#################################
#Starting and ending years for landings time series
1950
2007
#Poundnet landings vector (1000 lb whole weight) and CV's
13 6 3 1 4 6 16 15 6 17 21 122 14 65 32 90 111 23 73 84 104 26 23 51 25 62 77 29 2 1 4 2
11 13 14 33 39 235 183 505 509 468 397 328 345 207 302 208 118 302 206 222 136 111
72 50 10 14
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 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 for poundnet length compositions
1982

2007
#sample sizes for poundnet length compositions
259 42 56 296 181 557 666 1194 1189 1583 2206 549 510 1203 531 944 827 1152 133 283 438 64 56 243 143 213
#poundnet length comps
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0077 0.0000 0.0000 0.0039 0.0077
0.0154 0.0232 0.0425 0.0772 0.0695 0.0656 0.1004 0.0502 0.0579 0.0347 0.0386
0.0618 0.0579 0.0695 0.0347 0.0386 0.0618 0.0309 0.0232 0.0154 0.0000 0.0039
0.0000 0.0039 0.0039 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
0.0238 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0476 0.0476 0.0238
0.0714 0.1190 0.0952 0.0952 0.0714 0.0952 0.0476 0.0238 0.0000 0.0000 0.0238
0.0000 0.0000 0.0000 0.0238 0.0476 0.0000 0.0000 0.0238 0.0476 0.0476 0.0000
0.0238 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 0.0000 0.0000 0.0000 0.0000 0.0179 0.0000 0.0000 0.0000 0.0000 0.0357 0.0179
0.0714 0.0000 0.1250 0.0893 0.0714 0.0536 0.1250 0.1071 0.1250 0.0000 0.0179
0.0000 0.0179 0.0179 0.0000 0.0179 0.0000 0.0000 0.0179 0.0179 0.0000 0.0000
0.0179 0.0179 0.0000 0.0000 0.0179 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 0.0034 0.0000 0.0000 0.0203 0.0338 0.0405 0.1182
0.1554 0.1520 0.1081 0.0676 0.0912 0.1081 0.0507 0.0101 0.0034 0.0000 0.0034
0.0034 0.0034 0.0101 0.0101 0.0000 0.0034 0.0000 0.0000 0.0000 0.0000 0.0000
0.0034 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 0.0000 0.0000 0.0000 0.0000 0.0000 0.0110 0.0055 0.0000 0.0221 0.0608 0.0994
0.0829 0.0387 0.0442 0.0166 0.0497 0.0497 0.0442 0.0331 0.0442 0.0497 0.0276
0.0552 0.0221 0.0387 0.0166 0.0497 0.0552 0.0497 0.0000 0.0166 0.0000 0.0055
0.0000 0.0055 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0055 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 0.0018 0.0000 0.0000 0.0000 0.0000 0.0000 0.0018 0.0000 0.0108 0.0018 0.0251
0.0162 0.0431 0.0628 0.1041 0.0736 0.0844 0.0431 0.0503 0.0395 0.0628 0.0628
0.0862 0.0628 0.0557 0.0359 0.0162 0.0180 0.0036 0.0090 0.0090 0.0036 0.0000
0.0036 0.0036 0.0018 0.0018 0.0000 0.0036 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0018 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 0.0015 0.0000 0.0000 0.0000 0.0000 0.0015 0.0030 0.0120 0.0255 0.0450 0.0631
0.0931 0.0916 0.0691 0.0390 0.0300 0.0240 0.0225 0.0195 0.0225 0.0405 0.0240
0.0495 0.0465 0.0706 0.0420 0.0300 0.0465 0.0195 0.0105 0.0060 0.0060 0.0090
0.0015 0.0090 0.0075 0.0015 0.0045 0.0015 0.0030 0.0030 0.0015 0.0000 0.0000
0.0015 0.0000 0.0000 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 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0059 0.0235 0.0335 0.0888
0.1248 0.1742 0.1717 0.0653 0.0260 0.0084 0.0084 0.0092 0.0168 0.0193 0.0377
0.0360 0.0410 0.0318 0.0302 0.0226 0.0092 0.0050 0.0017 0.0008 0.0017 0.0000
0.0000 0.0000 0.0000 0.0025 0.0008 0.0000 0.0000 0.0000 0.0000 0.0000 0.0008
0.0000 0.0000 0.0017 0.0000 0.0008 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
0.0008 0.0000 0.0000 0.0008 0.0017 0.0042 0.0067 0.0059 0.0177 0.0227 0.0429 0.0563
0.0782 0.1001 0.1564 0.1505 0.1060 0.0521 0.0126 0.0143 0.0151 0.0151 0.0210
0.0294 0.0185 0.0143 0.0050 0.0050 0.0050 0.0034 0.0042 0.0034 0.0017 0.0017
0.0017 0.0008 0.0042 0.0025 0.0017 0.0034 0.0008 0.0017 0.0025 0.0008 0.0017
0.0008 0.0008 0.0000 0.0000 0.0008 0.0017 0.0008 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0032 0.0120 0.0063 0.0139 0.0215 0.0303 0.0436 0.0670
0.0777 0.0853 0.0783 0.0708 0.0474 0.0328 0.0234 0.0234 0.0373 0.0423 0.0366
0.0303 0.0347 0.0392 0.0354 0.0246 0.0208 0.0126 0.0063 0.0063 0.0082 0.0019
0.0051 0.0038 0.0025 0.0013 0.0032 0.0038 0.0006 0.0044 0.0000 0.0006 0.0000
0.0000 0.0006 0.0006 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 0.0005 0.0000 0.0023 0.0050 0.0159 0.0286
0.0363 0.0440 0.0485 0.0394 0.0295 0.0213 0.0413 0.0485 0.0707 0.1097 0.1088
0.1170 0.0766 0.0539 0.0277 0.0186 0.0136 0.0082 0.0068 0.0050 0.0050 0.0050
0.0014 0.0023 0.0018 0.0005 0.0009 0.0005 0.0014 0.0005 0.0000 0.0009 0.0009
0.0018 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 0.0018 0.0000 0.0000 0.0036 0.0018 0.0091
0.0055 0.0182 0.0109 0.0164 0.0182 0.0401 0.0729 0.0692 0.0820 0.0455 0.0401
0.0510 0.0874 0.0984 0.1020 0.0638 0.0474 0.0383 0.0237 0.0128 0.0073 0.0073
0.0036 0.0036 0.0000 0.0055 0.0036 0.0018 0.0000 0.0018 0.0018 0.0000 0.0000
0.0036 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 0.0020 0.0000 0.0020 0.0039 0.0196 0.0550 0.0432
0.0491 0.0688 0.0943 0.1238 0.0707 0.0432 0.0609 0.0373 0.0393 0.0275 0.0452
0.0452 0.0275 0.0196 0.0255 0.0118 0.0157 0.0138 0.0098 0.0079 0.0039 0.0118
0.0059 0.0020 0.0020 0.0059 0.0020 0.0020 0.0000 0.0000 0.0020 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	0.0000	0.0000	0.0000
	0.0000	0.0008	0.0008	0.0033	0.0058	0.0058	0.0233	0.0366	0.0623	0.0798	0.0865
	0.1056	0.1255	0.1205	0.0973	0.0657	0.0490	0.0316	0.0224	0.0141	0.0125	0.0083
	0.0067	0.0116	0.0025	0.0033	0.0025	0.0008	0.0017	0.0008	0.0008	0.0000	0.0000
	0.0008	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0017	0.0000	0.0008	0.0000
	0.0000	0.0008	0.0008	0.0067							
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	0.0019	0.0000	0.0000	0.0094	0.0075	0.0226	0.0508	0.0471	0.0810
	0.1092	0.1224	0.0998	0.0979	0.0810	0.0791	0.0508	0.0490	0.0188	0.0169	0.0151
	0.0094	0.0132	0.0019	0.0038	0.0056	0.0000	0.0000	0.0000	0.0000	0.0019	0.0000
	0.0038	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	0.0011	0.0021
	0.0021	0.0000	0.0021	0.0000	0.0000	0.0000	0.0042	0.0095	0.0318	0.0975	0.1377
	0.1780	0.1314	0.1176	0.0763	0.0667	0.0297	0.0191	0.0138	0.0201	0.0169	0.0106
	0.0053	0.0064	0.0021	0.0148	0.0000	0.0000	0.0011	0.0000	0.0011	0.0000	0.0000
	0.0000	0.0000	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	0.0012	0.0000
	0.0012	0.0060	0.0024	0.0012	0.0012	0.0012	0.0000	0.0133	0.0278	0.0556	0.0713
	0.1258	0.1729	0.1125	0.0967	0.0641	0.0544	0.0363	0.0387	0.0206	0.0073	0.0145
	0.0109	0.0169	0.0048	0.0048	0.0073	0.0048	0.0036	0.0036	0.0073	0.0024	0.0024
	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0012	0.0000	0.0000	0.0012	0.0024
	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	0.0017	0.0000	0.0026
	0.0000	0.0139	0.0269	0.0642	0.0477	0.0260	0.0243	0.0260	0.0512	0.0868	0.1024
	0.1076	0.0911	0.0929	0.0668	0.0365	0.0260	0.0278	0.0148	0.0122	0.0104	0.0095
	0.0043	0.0043	0.0043	0.0035	0.0009	0.0026	0.0017	0.0009	0.0009	0.0026	0.0009
	0.0017	0.0009	0.0009	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
	0.0075	0.0000	0.0000	0.0000	0.0075	0.0000	0.0226	0.0226	0.0301	0.0301	0.1353
	0.0977	0.1128	0.1654	0.0376	0.0226	0.0526	0.0602	0.0226	0.0226	0.0150	0.0075
	0.0000	0.0000	0.0150	0.0150	0.0150	0.0075	0.0226	0.0075	0.0150	0.0000	0.0000
	0.0000	0.0000	0.0075	0.0075	0.0000	0.0075	0.0000	0.0000	0.0075	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	0.0035	0.0035	0.0000	0.0106	0.0247	0.0212	0.0247	0.0530	0.0707	0.0707
	0.1060	0.0848	0.1025	0.1166	0.0530	0.0424	0.0459	0.0459	0.0389	0.0106	0.0141
	0.0106	0.0071	0.0071	0.0071	0.0000	0.0071	0.0000	0.0106	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0035	0.0000	0.0035	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	0.0023	0.0068	0.0205	0.0411	0.0457	0.1301
	0.1393	0.1256	0.1073	0.0913	0.0731	0.0228	0.0479	0.0137	0.0297	0.0228	0.0114
	0.0160	0.0160	0.0046	0.0114	0.0023	0.0068	0.0046	0.0046	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0023	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	0.0000	0.0000	0.0000	0.0156	0.0313	0.0313	0.0156	0.1563	0.0938
	0.1250	0.1406	0.0625	0.0938	0.0469	0.0000	0.0313	0.0000	0.0156	0.0000	0.0156
	0.0000	0.0313	0.0000	0.0000	0.0156	0.0313	0.0000	0.0156	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0000	0.0156	0.0000	0.0156	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	0.0000	0.0000	0.0179	0.0357	0.0714	0.0893
	0.0714	0.2321	0.1607	0.1250	0.0179	0.0357	0.0000	0.0179	0.0000	0.0000	0.0000
	0.0000	0.0179	0.0000	0.0357	0.0000	0.0000	0.0179	0.0179	0.0000	0.0000	0.0000
	0.0179	0.0000	0.0000	0.0000	0.0179	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	0.0576	0.0370	0.0864	0.0412	0.0206	0.0206	0.0123	0.0329	0.0658	0.0988
	0.1111	0.0741	0.1358	0.0453	0.0617	0.0412	0.0329	0.0123	0.0082	0.0041	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	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
	0.0070	0.0070	0.0070	0.0140	0.0070	0.0140	0.0210	0.0350	0.0629	0.0280	0.0420
	0.0769	0.1469	0.0979	0.1049	0.1259	0.0350	0.0280	0.0070	0.0280	0.0000	0.0140
	0.0000	0.0070	0.0140	0.0210	0.0070	0.0000	0.0140	0.0000	0.0070	0.0000	0.0000
	0.0000	0.0000	0.0140	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	0.0000	0.0000	0.0070	0.0000							
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0047	0.0469
	0.0235	0.0516	0.0469	0.0235	0.0000	0.0047	0.0094	0.0376	0.0282	0.0939	0.1831
	0.1174	0.1268	0.0610	0.0610	0.0141	0.0282	0.0000	0.0000	0.0000	0.0047	0.0000
	0.0047	0.0047	0.0094	0.0047	0.0000	0.0000	0.0047	0.0000	0.0000	0.0047	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							

#number of years, year vector for poundnet age comps
5
1992 1995 1998 1999 2001
#sample sizes for pound net age comps
28 20 50 23 60
#poundnet age comps
0.6210 0.3723 0.0045 0.0022 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.9094 0.0906 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0167 0.8551 0.1040 0.0202 0.0040 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0026 0.9974 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0714 0.7766 0.1096 0.0196 0.0134 0.0000 0.0093 0.0000 0.0000 0.0000 0.0000
####################Commercial Gillnet CPUE and Landings##########################
#Commercial Gillnet CPUE Index from FL Trip ticket before net ban
#Starting and ending years of CPUE index
1985
1994
#Observed CPUE and assumed CVs
0.46 0.59 0.83 0.64 0.93 0.79 0.65 0.63 2.10 2.40
0.11 0.11 0.11 0.11 0.11 0.09 0.09 0.11 0.30 0.19
#Commercial Gillnet CPUE from FL after net ban
#Starting and ending years of CPUE index
1996
2007
#Observed CPUE and assumed CVs
1.25 0.77 1.05 1.05 1.09 0.88 0.85 0.94 0.62 1.11 1.17 1.21
0.15 0.30 0.15 0.15 0.12 0.12 0.13 0.13 0.13 0.12 0.12 0.11
#Commercial Gillnet CPUE from logbook N of FL
#Starting and ending years of CPUE index
1998
2007
#Observed CPUE and assumed CVs
#0.59 0.79 1.26 1.86 1.05 0.54 0.62 0.91 1.15 1.11 ##WRONG
0.89 0.73 0.83 1.22 1.32 1.11 1.20 0.73 0.86 1.11
0.22 0.18 0.20 0.29 0.30 0.25 0.28 0.17 0.20 0.26
#Commercial Gillnet Landings
#Starting and ending years of post-WW2 landings time series, respectively
1950
2007
#Observed landings (in 1000 lb whole weight - rounded) and assumed CVs (gear="Other" added in as well)
3008 2837 3674 3115 2940 4004 4765 5861 5297 2471 2774 3017 2349 2160 2478 2467 1910 3181 3211 3056 3059
3019 3250 2641 3686 7045 10926 6753 6250 6268 6373 2868 6981 3430 3674 3349 2357 2529 3328 3246
2845 3853 3131 4657 5107 1449 2470 2710 2900 1557 1576 1515 1318 951 788 1209 1417 1705
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 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
1986
2007
#Observed discards (1000s) and assumed CVs - 1985-1997 were based on extrapolations from 1998-2007; CV's doubled
12 12 14 7 12 14 14 23 26 8 15 18 9 14 10 11 12 9 7 8 7 6
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 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 gillnet length composition sample data
1984
2007
#sample size of commercial gillnet length comp data by year
968 389 1517 180 1510 456 3485 6268 9933 7945 7536 1111 2951 1532 6318 7414 3723 1371 941 855 1089 2101 2626 2052
#commercial gillnet length composition samples (year,lengthbin 1cm)
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00109 0.00192
0.00657 0.00823 0.01477 0.02891 0.03845 0.05261 0.10004 0.11584 0.10575 0.11584 0.10030
0.07196 0.05124 0.03707 0.04143 0.02944 0.01635 0.00981 0.01199 0.00763 0.00763 0.00654
0.00327 0.00218 0.00327 0.00109 0.00218 0.00327 0.00000 0.00000 0.00028 0.00110 0.00055
0.00028 0.00000 0.00000 0.00110
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00047 0.00140 0.00000 0.00272
0.01905 0.02721 0.01633 0.03040 0.02994 0.04626 0.06259 0.09525 0.05987 0.08981 0.05171
0.04899 0.04899 0.04899 0.02449 0.01361 0.02177 0.00816 0.00816 0.00000 0.00544 0.01089
0.00863 0.02768 0.02721 0.02449 0.01633 0.01454 0.03538 0.01135 0.01633 0.01407 0.01003
0.00910 0.00365 0.00272 0.00599
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00630 0.00630 0.01292 0.02174
0.01386 0.02077 0.04311 0.04531 0.05349 0.08495 0.11012 0.09879 0.12396 0.09376 0.07992
0.06230 0.03083 0.01982 0.00692 0.00629 0.00504 0.00220 0.00378 0.00063 0.00063 0.00158
0.00000 0.00126 0.00063 0.00189 0.00692 0.00566 0.00378 0.00252 0.00378 0.00566 0.00315
0.00252 0.00252 0.00252 0.00189
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00336 0.00841 0.02187 0.02187 0.03869 0.05214 0.03364
0.02018 0.00673 0.00168 0.00168 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.01436 0.00000 0.01436 0.01436 0.04308 0.02872

	0.02872	0.02872	0.08615	0.02872	0.07180	0.02872	0.04308	0.05744	0.05744	0.04308	0.02872
	0.01436	0.02872	0.04308	0.08615							
0.00000	0.00000	0.00058	0.00058	0.00000	0.00116	0.00175	0.00000	0.00058	0.00000	0.00000	0.00058
	0.00291	0.00408	0.00175	0.00349	0.00524	0.00931	0.01223	0.01339	0.02038	0.01708	0.00873
	0.00951	0.00311	0.01098	0.02312	0.03682	0.04363	0.06647	0.07123	0.08261	0.06560	0.06803
	0.05520	0.04801	0.03868	0.03732	0.03936	0.02653	0.02041	0.02517	0.01633	0.01759	0.00476
	0.01633	0.00680	0.00884	0.00476	0.00544	0.00952	0.00408	0.00408	0.00544	0.00272	0.00408
	0.00340	0.00204	0.00272	0.00544							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00219	0.00658	0.01535	0.02412	0.02632	0.03509	0.03509	0.02412
	0.02412	0.02632	0.01974	0.01316	0.01316	0.01316	0.01974	0.01316	0.02412	0.03728	0.03947
	0.05044	0.02851	0.04386	0.06360	0.03509	0.03947	0.05921	0.05263	0.03289	0.05702	0.03728
	0.02193	0.02632	0.01316	0.02632							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00017	0.00017	0.00000	0.00000	0.00536	0.02077	0.03116	0.06248
	0.05746	0.01624	0.03282	0.02912	0.01116	0.01265	0.03785	0.03663	0.03840	0.04746	0.04111
	0.03354	0.02890	0.02603	0.05746	0.02790	0.03061	0.02321	0.02006	0.02575	0.01989	0.03078
	0.01823	0.02823	0.03210	0.02625	0.01707	0.01995	0.01343	0.00945	0.00796	0.00613	0.00448
	0.00597	0.00149	0.00099	0.00315							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00841	0.01682	0.02524	0.02562	0.00932	0.00155
	0.00207	0.00426	0.00646	0.01137	0.01756	0.02363	0.03112	0.05169	0.05206	0.06033	0.04688
	0.04339	0.06073	0.03526	0.03384	0.03112	0.02673	0.04317	0.04291	0.03360	0.02286	0.02972
	0.03140	0.02467	0.02183	0.02441	0.01756	0.01485	0.01601	0.01369	0.01291	0.00736	0.00555
	0.00336	0.00271	0.00220	0.00375							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00049	0.00008	0.00113	0.00348	0.00226	0.00485	0.00597	0.00417
	0.00948	0.00728	0.02068	0.03014	0.03710	0.05245	0.05308	0.06146	0.06641	0.07120	0.05976
	0.04868	0.04886	0.04091	0.04184	0.03841	0.03482	0.03426	0.02733	0.02677	0.02575	0.02134
	0.02158	0.01632	0.01529	0.01440	0.00987	0.00875	0.00915	0.00581	0.00485	0.00390	0.00303
	0.00215	0.00247	0.00095	0.00103							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00153	0.00000	0.00000	0.00000	0.00000	0.00459	0.00043	0.00229	0.00228
	0.00293	0.01222	0.01275	0.03446	0.04794	0.05888	0.06518	0.06341	0.06381	0.06887	0.05985
	0.05198	0.04602	0.03707	0.03795	0.03728	0.03675	0.03827	0.03153	0.03054	0.02804	0.02413
	0.02217	0.01750	0.01511	0.01239	0.01000	0.00598	0.00522	0.00533	0.00217	0.00098	0.00065
	0.00098	0.00011	0.00022	0.00022							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00013	0.00013	0.00013	0.00000	0.00000	0.00013	0.00066
	0.00199	0.00491	0.01008	0.01990	0.03649	0.05932	0.06913	0.07829	0.08386	0.07617	0.07126
	0.06542	0.06091	0.05268	0.03755	0.04061	0.02720	0.02787	0.02176	0.02243	0.02282	0.02004
	0.01579	0.01367	0.01274	0.01221	0.00836	0.00809	0.00584	0.00464	0.00252	0.00133	0.00080
	0.00066	0.00080	0.00040	0.00027							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00053	0.00000	0.00318
	0.08112	0.12538	0.01802	0.10285	0.06865	0.07607	0.07024	0.04240	0.07819	0.03074	0.02650
	0.03392	0.02438	0.02438	0.02544	0.02067	0.01590	0.01537	0.01272	0.01325	0.01643	0.01749
	0.01537	0.01219	0.01219	0.00477	0.00265	0.00318	0.00159	0.00159	0.00106	0.00053	0.00053
	0.00053	0.00000	0.00000	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00068	0.00102	0.00949	0.01186	0.01525
	0.01864	0.03050	0.04812	0.05829	0.08201	0.06879	0.07455	0.06743	0.06235	0.06676	0.04846
	0.05219	0.04575	0.03863	0.04236	0.02745	0.02745	0.02101	0.01355	0.01118	0.01254	0.01050
	0.00949	0.00644	0.00644	0.00339	0.00339	0.00136	0.00102	0.00102	0.00034	0.00000	0.00000
	0.00034	0.00000	0.00000	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00049	0.00049	0.00781	0.02966	0.02043
	0.02687	0.04461	0.06819	0.08720	0.07496	0.07546	0.08994	0.06322	0.05821	0.04696	0.05325
	0.04249	0.03281	0.03576	0.02938	0.02687	0.01621	0.00983	0.00835	0.01179	0.01272	0.00246
	0.00688	0.00393	0.00491	0.00246	0.00197	0.00147	0.00098	0.00000	0.00000	0.00000	0.00000
	0.00049	0.00000	0.00049	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00058	0.00158	0.00302	0.00648	0.01252	0.02145
	0.04074	0.06235	0.07159	0.09408	0.08897	0.08609	0.07866	0.05802	0.05327	0.05258	0.04175
	0.03210	0.03646	0.04039	0.01915	0.03000	0.01670	0.01355	0.00705	0.00880	0.01400	0.00202
	0.00144	0.00173	0.00130	0.00058	0.00058	0.00000	0.00014	0.00014	0.00014	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00114	0.00000	0.00040	0.00153	0.00520	0.00584	0.02001	0.02811
	0.04648	0.04560	0.05765	0.05799	0.07194	0.06076	0.06202	0.05558	0.04883	0.04309	0.03783
	0.04935	0.03357	0.02714	0.02774	0.02576	0.02067	0.02433	0.02914	0.02226	0.01824	0.01656
	0.01666	0.01170	0.00868	0.00714	0.00337	0.00278	0.00228	0.00163	0.00040	0.00030	0.00020
	0.00010	0.00000	0.00000	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00107	0.00000	0.00053	0.00131	0.00353	0.00759	0.01393	0.01611
	0.02984	0.04366	0.06946	0.07353	0.09123	0.08848	0.08459	0.06979	0.05919	0.05135	0.03906
	0.03212	0.02854	0.02447	0.03013	0.02190	0.02112	0.01816	0.01884	0.00998	0.00891	0.00673
	0.00610	0.00599	0.00450	0.00406	0.00435	0.00319	0.00203	0.00087	0.00116	0.00087	0.00058
	0.00000	0.00087	0.00029	0.00000							

0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00072 0.00072 0.00288 0.01663 0.04413 0.04497
0.08284 0.08722 0.10551 0.10051 0.09695 0.08606 0.08807 0.04670 0.03076 0.03727 0.02348
0.02062 0.01108 0.01552 0.00589 0.01180 0.00735 0.00442 0.00514 0.00219 0.00370 0.00442
0.00291 0.00147 0.00365 0.00074 0.00147 0.00074 0.00000 0.00072 0.00074 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00387 0.00858 0.02489 0.03495 0.05196
0.05919 0.06390 0.09043 0.07917 0.07714 0.07774 0.08507 0.06405 0.06331 0.03981 0.03589
0.03217 0.02518 0.01899 0.01725 0.01254 0.00704 0.00625 0.00555 0.00397 0.00079 0.00159
0.00159 0.00000 0.00000 0.00000 0.00238 0.00000 0.00317 0.00079 0.00079 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00298 0.01039 0.02142 0.03316
0.05925 0.05447 0.05938 0.07479 0.09920 0.10019 0.07264 0.07275 0.04893 0.04542 0.03696
0.02803 0.03404 0.02032 0.01530 0.02534 0.01629 0.01401 0.00999 0.00496 0.00864 0.00397
0.00099 0.00298 0.00496 0.00397 0.00298 0.00333 0.00099 0.00298 0.00000 0.00298 0.00000
0.00099 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00184 0.00275 0.00826 0.01928
0.01561 0.03030 0.05051 0.03949 0.05234 0.05510 0.04683 0.05234 0.09458 0.07346 0.07346
0.07254 0.05326 0.03673 0.02663 0.02663 0.02296 0.02755 0.01928 0.01653 0.01745 0.01102
0.00918 0.00459 0.01377 0.00551 0.00367 0.00184 0.00459 0.00367 0.00184 0.00092 0.00275
0.00000 0.00000 0.00092 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.01389 0.00000 0.00045 0.01493 0.03061
0.06301 0.06942 0.13258 0.06744 0.11135 0.13868 0.06624 0.03429 0.07671 0.02085 0.04387
0.01683 0.00998 0.03702 0.00655 0.00447 0.00402 0.00328 0.00328 0.01598 0.00179 0.00164
0.00119 0.00134 0.00089 0.00164 0.00149 0.00149 0.00060 0.00089 0.00045 0.00060 0.00015
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
0.00000 0.00000 0.00000 0.00000 0.00000 0.00011 0.00011 0.00045 0.00112 0.00090 0.01357
0.01492 0.05460 0.06011 0.06170 0.10005 0.10599 0.12382 0.12684 0.07874 0.07750 0.04466
0.03121 0.03220 0.01617 0.00922 0.00810 0.00810 0.00439 0.00517 0.00416 0.00484 0.00281
0.00157 0.00101 0.00124 0.00034 0.00079 0.00067 0.00056 0.00056 0.00045 0.00056 0.00011
0.00034 0.00011 0.00000 0.00011
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00049 0.00244 0.00292 0.02144 0.04191 0.05556 0.06287
0.06238 0.06384 0.05263 0.03509 0.03655 0.03411 0.05068 0.03996 0.04630 0.04435 0.05263
0.04191 0.04727 0.03558 0.03655 0.02437 0.01657 0.01559 0.01511 0.00731 0.00926 0.00877
0.00634 0.00146 0.00731 0.00439 0.00390 0.00097 0.00244 0.00244 0.00244 0.00000 0.00049
0.00195 0.00049 0.00000 0.00097
#Starting, ending years of age compositions for commercial gillnet fishery
1988
2007
#sample sizes of age comps by year (minimum sample size of 45)
72 135 216 175 250 90 23 154 417 246 363 528 539 452 376 323 336 249 315 182
#age composition samples (year,age)
0.0434 0.2491 0.4436 0.1852 0.0775 0.0012 0.0000 0.0000 0.0000 0.0000 0.0000
0.0008 0.2382 0.2236 0.1829 0.1902 0.1260 0.0333 0.0049 0.0000 0.0000 0.0000
0.1309 0.1426 0.1383 0.2333 0.1670 0.1229 0.0599 0.0051 0.0000 0.0000 0.0000
0.0099 0.4689 0.3294 0.1120 0.0622 0.0139 0.0000 0.0037 0.0000 0.0000 0.0000
0.0030 0.2277 0.4971 0.1891 0.0411 0.0236 0.0116 0.0068 0.0000 0.0000 0.0000
0.0173 0.4350 0.3619 0.0595 0.0493 0.0302 0.0070 0.0172 0.0226 0.0000 0.0000
0.0000 0.0015 0.1983 0.3652 0.4350 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.1740 0.3536 0.2853 0.1080 0.0440 0.0112 0.0113 0.0087 0.0038 0.0000 0.0000
0.0368 0.2926 0.3406 0.2617 0.0520 0.0139 0.0013 0.0007 0.0003 0.0000 0.0000
0.0277 0.2774 0.4358 0.1944 0.0471 0.0038 0.0061 0.0044 0.0000 0.0034 0.0000
0.2052 0.2025 0.3912 0.1343 0.0415 0.0166 0.0064 0.0024 0.0000 0.0000 0.0000
0.0743 0.4165 0.2150 0.1637 0.0765 0.0335 0.0162 0.0032 0.0010 0.0000 0.0000
0.0200 0.4020 0.3626 0.0981 0.0795 0.0215 0.0133 0.0028 0.0002 0.0000 0.0000
0.1309 0.3358 0.2806 0.1964 0.0446 0.0085 0.0015 0.0009 0.0006 0.0001 0.0000
0.0951 0.1218 0.2953 0.1897 0.1929 0.0853 0.0164 0.0009 0.0003 0.0017 0.0005
0.0906 0.3250 0.2890 0.1780 0.0725 0.0347 0.0066 0.0034 0.0002 0.0000 0.0000
0.0728 0.3701 0.2414 0.1582 0.0706 0.0536 0.0202 0.0123 0.0003 0.0000 0.0003
0.1065 0.6798 0.1561 0.0492 0.0069 0.0003 0.0012 0.0000 0.0000 0.0000 0.0000
0.0319 0.4266 0.3547 0.1276 0.0460 0.0110 0.0009 0.0004 0.0007 0.0001 0.0000
0.2768 0.4505 0.1700 0.0474 0.0325 0.0088 0.0039 0.0101 0.0000 0.0000 0.0000
####################################Cast net#######################################
#Starting and ending years for CPUE index
1999
2007
#Observed CPUE values (numbers) and CVs,
0.77 0.77 0.83 0.95 1.39 1.48 1.17 0.86 0.78
0.30 0.26 0.26 0.24 0.24 0.26 0.24 0.26 0.26
#Starting and ending years for landings time series
1995
2007
#Castnet landings vector (1000 lb whole weight) and CV's
34 197 76 33 345 622 934 1420 2270 1745 1716 1380 549

0.05 0.05 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 (castnet)
10
1996 1999 2000 2001 2002 2003 2004 2005 2006 2007
#sample sizes of length comp data by year
50 50 3360 3683 1967 1686 893 1381 577 55
#CN recreational length comp samples (year,lengthbin)
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 0.0400 0.0800 0.0200 0.3200 0.1000 0.2000 0.0800 0.1000 0.0600 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 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 0.0200 0.0000 0.0000
0.0000 0.0200 0.0600 0.0000 0.0400 0.0600 0.1000 0.1600 0.1800 0.1400 0.0000
0.0200 0.0600 0.0600 0.0200 0.0400 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0200 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 0.0003 0.0000 0.0024 0.0018 0.0030
0.0057 0.0170 0.0345 0.0536 0.0753 0.1107 0.1110 0.1065 0.0896 0.0810 0.0568
0.0393 0.0307 0.0182 0.0128 0.0104 0.0083 0.0065 0.0101 0.0202 0.0161 0.0167
0.0161 0.0116 0.0125 0.0101 0.0086 0.0018 0.0003 0.0000 0.0000 0.0000 0.0006
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 0.0003 0.0000 0.0000 0.0000 0.0000 0.0005 0.0019
0.0087 0.0160 0.0350 0.0538 0.0741 0.0969 0.0923 0.0923 0.0834 0.0646 0.0595
0.0448 0.0459 0.0391 0.0342 0.0388 0.0255 0.0234 0.0166 0.0125 0.0122 0.0054
0.0054 0.0035 0.0024 0.0033 0.0030 0.0008 0.0014 0.0005 0.0011 0.0003 0.0003
0.0000 0.0003 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 0.0015 0.0025 0.0076 0.0066 0.0198
0.0275 0.0437 0.0524 0.0788 0.0935 0.0727 0.0895 0.0793 0.0615 0.0651 0.0452
0.0290 0.0305 0.0330 0.0224 0.0219 0.0163 0.0127 0.0122 0.0107 0.0122 0.0102
0.0097 0.0061 0.0046 0.0051 0.0046 0.0036 0.0025 0.0031 0.0010 0.0000 0.0010
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 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0018 0.0036 0.0059 0.0071 0.0119
0.0249 0.0421 0.0445 0.0635 0.0647 0.0647 0.0641 0.0552 0.0540 0.0540 0.0480
0.0486 0.0386 0.0261 0.0225 0.0202 0.0178 0.0237 0.0297 0.0338 0.0380 0.0196
0.0184 0.0142 0.0071 0.0059 0.0047 0.0095 0.0065 0.0042 0.0006 0.0000 0.0006
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 0.0011
0.0034 0.0078 0.0168 0.0526 0.1086 0.1389 0.1445 0.1165 0.0694 0.0482 0.0358
0.0358 0.0258 0.0336 0.0213 0.0202 0.0134 0.0179 0.0134 0.0112 0.0112 0.0123
0.0123 0.0078 0.0090 0.0045 0.0022 0.0011 0.0011 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 0.0007 0.0000 0.0000 0.0007 0.0014
0.0101 0.0239 0.0500 0.0833 0.1086 0.0941 0.0789 0.0637 0.0601 0.0463 0.0413
0.0398 0.0427 0.0406 0.0268 0.0311 0.0203 0.0174 0.0268 0.0123 0.0116 0.0181
0.0167 0.0138 0.0058 0.0043 0.0014 0.0022 0.0007 0.0022 0.0007 0.0000 0.0007
0.0007 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 0.0069
0.0139 0.0225 0.0416 0.0329 0.0711 0.0919 0.1127 0.1057 0.0936 0.0745 0.0815
0.0641 0.0312 0.0364 0.0347 0.0277 0.0052 0.0087 0.0087 0.0069 0.0035 0.0104
0.0052 0.0052 0.0017 0.0017 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 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0182 0.0000 0.0000 0.0545 0.0182 0.0727 0.1273 0.0545
0.0727 0.0727 0.0545 0.0727 0.0545 0.0364 0.0182 0.0000 0.0545 0.0545 0.0909
0.0364 0.0364 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
#Number and vector of years of age compositions (castnet)
5
1996 2000 2004 2005 2006
#sample sizes of age comp data by year
34 110 147 212 50
#CN age comp samples (year,age)
0.0000 0.0441 0.6471 0.2096 0.0662 0.0331 0.0000 0.0000 0.0000 0.0000
0.0000 0.0027 0.2680 0.5451 0.1347 0.0416 0.0030 0.0000 0.0049 0.0000 0.0000
0.0000 0.0026 0.3607 0.2423 0.2801 0.0589 0.0230 0.0111 0.0193 0.0000 0.0018
0.0000 0.0309 0.4798 0.2657 0.1408 0.0553 0.0161 0.0021 0.0046 0.0048 0.0000
0.0000 0.0000 0.2502 0.1698 0.2486 0.2172 0.0764 0.0378 0.0000 0.0000 0.0000
################################## Recreational ##################################
#Recreational MRFSS CPUE Index

#Starting and ending years of CPUE index
1987
2007
#Observed CPUE and assumed CVs
0.84 1.17 1.00 0.90 0.77 0.83 0.66 1.01 0.81 1.06 1.09 0.93 1.16 1.07 1.13 1.36 1.19 0.99 1.29 0.82 0.88
0.28 0.25 0.29 0.24 0.21 0.17 0.22 0.23 0.30 0.26 0.22 0.27 0.22 0.23 0.25 0.28 0.30 0.28 0.27 0.27 0.24
#Recreational Charter+Private boat landings
#Starting and ending years for landings time series
1950
2007
#MRFSS + headboat landings vector (1000's)

5126	5167	5208	5249	5290	5331	5372	5412	5453	5494	5535	5602
5669	5736	5803	5870	5493	5117	4740	4364	3988	3657	3326	2995
2664	2333	2002	1671	1341	1010	679	888	904	127	971	487
	889	1185	1744								
1227	1359	1548	1382	955	1220	876	841	1113	688	1087	1737
	1243	1280									
1532	883	1088	907	1051							

0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 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
1950
2007
#Observed discards (1000s) and assumed CVs
203 205 206 208 210 211 213 214 216 218 219 222 224 227 230 232 217 203 188 173 158 145 131
118 105 92 79 66 53 40 26 62 7 5 26 55 318 62 64 240 161 365 350 245 752 391 357 420 267 641
827 676 614 812 420 748 283 565
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 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 length compositions (mrfss)
1981
2007
#sample sizes of length comp data by year
226 155 62 189 106 394 1210 455 2068 2430 2440 1807 1278 2568 988 1272 2027 1101 1694 1659 1324 1168 886 630 635 461 586
#mrfss length comp samples (year,age) (1cm length bins)

0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00362	0.00000	0.00000	0.00000	0.00000	0.00362	0.00000	0.00362	0.01449	0.02404	0.01749
	0.01087	0.02632	0.04628	0.05842	0.04561	0.11656	0.06450	0.04577	0.04851	0.09855	0.04912
	0.02162	0.00906	0.02923	0.00979	0.01673	0.00664	0.00906	0.00492	0.01756	0.00000	0.00906
	0.00000	0.01804	0.00609	0.01797	0.01707	0.00000	0.02103	0.00662	0.00752	0.03364	0.02040
	0.03391	0.00000	0.00662	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00892	0.00000	0.00000	0.00000	0.00067	0.00718	0.00000	0.03753
	0.04423	0.10541	0.08570	0.10372	0.10879	0.07692	0.08864	0.07915	0.07921	0.01672	0.02211
	0.02195	0.00147	0.01672	0.01586	0.00561	0.00520	0.00132	0.00247	0.00922	0.00270	0.00270
	0.00040	0.00270	0.00928	0.00519	0.00000	0.00132	0.00337	0.00000	0.00000	0.00000	0.00205
	0.00095	0.00272	0.00000	0.02189							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00098	0.00098	0.00000
	0.00098	0.11638	0.04518	0.01697	0.05758	0.03703	0.00098	0.08778	0.07516	0.06031	0.03240
	0.00385	0.00575	0.01345	0.00000	0.01460	0.17091	0.02284	0.01653	0.15323	0.00000	0.00000
	0.00116	0.00556	0.00000	0.00000	0.00000	0.02799	0.00115	0.00000	0.00000	0.02914	0.00115
	0.00000	0.00000	0.00000	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.01196	0.00202	0.00000	0.00031	0.00086
	0.10787	0.01116	0.17359	0.03853	0.10349	0.07771	0.07536	0.07303	0.06918	0.08838	0.09658
	0.01636	0.00326	0.00000	0.00308	0.00152	0.00173	0.00000	0.00094	0.00232	0.00435	0.01599
	0.00010	0.00085	0.00043	0.00000	0.00189	0.00000	0.00094	0.01483	0.00094	0.00043	0.00000
	0.00000	0.00000	0.00000	0.00000							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.04052	0.00000	0.05840	0.01947	0.05759	0.04070	0.02106	0.03446
	0.00999	0.00082	0.00041	0.00041	0.01947	0.03044	0.13030	0.04346	0.02153	0.00194	0.01040
	0.03025	0.02858	0.00980	0.01753	0.02210	0.03660	0.00106	0.02851	0.00000	0.00766	0.01947
	0.03893	0.05840	0.05840	0.02433	0.00000	0.02212	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00918	0.01947	0.02625							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00076	0.00000	0.00055	0.00000	0.00000	0.00763	0.00152	0.01176	0.00333	0.01588	0.05447
	0.04818	0.00852	0.05360	0.05747	0.04563	0.04955	0.04968	0.11074	0.05457	0.09613	0.02522
	0.04546	0.06968	0.02623	0.02279	0.01473	0.01791	0.00583	0.00696	0.00187	0.01067	0.00180
	0.01275	0.00612	0.00035	0.00057	0.01164	0.00413	0.00680	0.00063	0.00018	0.00161	0.00326
	0.01064	0.00000	0.01543	0.00675							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00011	0.00018	0.00018	0.00342	0.00526	0.02442	0.02349	0.03092	0.03772	0.03057
	0.04012	0.05955	0.05407	0.04801	0.05073	0.05531	0.06261	0.04193	0.05381	0.03389	0.02638
	0.02116	0.03058	0.01589	0.01944	0.01274	0.03621	0.02331	0.01964	0.00961	0.01293	0.00511
	0.01349	0.00825	0.01401	0.00737	0.00355	0.00346	0.00623	0.00350	0.00026	0.00812	0.00261
	0.00300	0.00467	0.00646	0.02571							

0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00031
	0.00031	0.00000	0.00000	0.00031	0.00000	0.00031	0.00031	0.00467	0.00457	0.00110	0.03750
	0.06679	0.01383	0.01404	0.00875	0.03213	0.03022	0.03056	0.08809	0.03582	0.07200	0.06307
	0.04615	0.02764	0.03970	0.06305	0.04933	0.02328	0.03734	0.03148	0.03057	0.01010	0.00465
	0.02658	0.02570	0.00902	0.02050	0.00447	0.00447	0.02050	0.00000	0.00180	0.00000	0.00460
	0.00180	0.00807	0.00000	0.00454							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00003
	0.00017	0.00148	0.00298	0.00967	0.00643	0.01136	0.01745	0.03507	0.05058	0.05542	0.04447
	0.04159	0.03349	0.05793	0.08149	0.04882	0.05694	0.04771	0.04563	0.04791	0.04432	0.02880
	0.02523	0.03184	0.02308	0.01422	0.01082	0.02525	0.01252	0.01091	0.00778	0.00852	0.00587
	0.00729	0.00513	0.00564	0.00463	0.00349	0.00512	0.00296	0.00225	0.00407	0.00080	0.00139
	0.00218	0.00210	0.00265	0.00452							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00114
	0.00162	0.00121	0.00502	0.02019	0.00967	0.00895	0.02047	0.03401	0.02165	0.04796	0.07846
	0.04025	0.05674	0.05384	0.04021	0.05312	0.04536	0.04307	0.05562	0.04712	0.03963	0.02307
	0.03733	0.02098	0.02666	0.02872	0.01246	0.00688	0.00429	0.00854	0.01950	0.00443	0.00983
	0.00768	0.00489	0.00298	0.00270	0.00365	0.00546	0.00635	0.00394	0.00836	0.00351	0.00204
	0.00516	0.00137	0.00270	0.01117							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00033	0.00000	0.00000	0.00000	0.00006	0.00014
	0.00094	0.00271	0.00159	0.00306	0.00847	0.03183	0.01767	0.02077	0.02271	0.03777	0.04063
	0.03224	0.03041	0.05685	0.04382	0.05337	0.04902	0.05743	0.06246	0.04097	0.04426	0.05088
	0.01746	0.02199	0.03182	0.02014	0.01938	0.01347	0.04056	0.02003	0.00588	0.01053	0.00804
	0.00898	0.00576	0.00375	0.00846	0.00527	0.00627	0.00532	0.00871	0.00347	0.00104	0.00120
	0.00131	0.00192	0.00093	0.01793							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00211	0.00295	0.00000
	0.00000	0.00000	0.00042	0.00077	0.00211	0.00388	0.00921	0.01852	0.03473	0.05260	0.04822
	0.04587	0.06668	0.05408	0.04367	0.04067	0.04950	0.04627	0.06189	0.06524	0.04856	0.05433
	0.02273	0.04021	0.01927	0.01230	0.01668	0.01515	0.01233	0.01337	0.00414	0.00785	0.00631
	0.01147	0.00701	0.00367	0.00518	0.00158	0.00383	0.01155	0.00275	0.00498	0.00511	0.00381
	0.00374	0.00081	0.00101	0.01086							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00066	0.00000	0.00200	0.00188	0.00665	0.00392	0.00803	0.01219	0.01985	0.02442	0.03190
	0.03615	0.04897	0.04072	0.04886	0.07822	0.08186	0.06788	0.03912	0.07079	0.04298	0.04159
	0.02921	0.02619	0.02654	0.02338	0.02780	0.01923	0.02310	0.00903	0.01193	0.01464	0.00359
	0.01591	0.00305	0.00651	0.01602	0.00753	0.00634	0.00133	0.00375	0.00195	0.00294	0.00113
	0.00355	0.00271	0.00216	0.00182							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00032	0.00000	0.00007	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00075	0.00032	0.00085	0.00406	0.01052	0.01497	0.08537	0.05400	0.03626
	0.05740	0.04276	0.06262	0.04604	0.07834	0.05100	0.04534	0.06977	0.04340	0.03304	0.05637
	0.02682	0.02466	0.02804	0.02154	0.01794	0.01610	0.00816	0.00637	0.00300	0.00731	0.00172
	0.00236	0.00044	0.00490	0.00123	0.00120	0.00541	0.00383	0.00240	0.00243	0.00000	0.00443
	0.00571	0.00399	0.00082	0.00560							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00014	0.00000
	0.00000	0.00000	0.00023	0.00171	0.00000	0.00074	0.00901	0.01789	0.07003	0.04180	0.05327
	0.05919	0.06987	0.06366	0.05246	0.07131	0.08946	0.05373	0.05142	0.04132	0.04588	0.02138
	0.03326	0.01522	0.02186	0.00607	0.01300	0.01192	0.01785	0.00938	0.00123	0.00209	0.00552
	0.00747	0.00037	0.00123	0.01128	0.00227	0.00265	0.00265	0.00036	0.00500	0.00490	0.00000
	0.00037	0.00000	0.00000	0.00956							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00415	0.00070	0.00376	0.00399	0.00978	0.03025	0.04964	0.04775
	0.03521	0.04735	0.05328	0.05970	0.04529	0.06514	0.06948	0.07912	0.05276	0.03308	0.02811
	0.06084	0.02118	0.01629	0.00280	0.01012	0.00718	0.01048	0.01225	0.01558	0.01150	0.03645
	0.00520	0.01617	0.01135	0.00305	0.00520	0.00285	0.00428	0.00596	0.00299	0.00745	0.00000
	0.00515	0.00520	0.00033	0.00159							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00162	0.00000	0.00000	0.00000	0.00000
	0.00530	0.00032	0.00000	0.00000	0.00000	0.00000	0.00033	0.00480	0.01706	0.04296	0.04700
	0.04972	0.04381	0.05360	0.06909	0.04553	0.07970	0.05474	0.06729	0.04615	0.04775	0.03582
	0.02147	0.03269	0.03076	0.02251	0.03371	0.02632	0.01519	0.02164	0.02666	0.00826	0.00554
	0.00869	0.00593	0.00472	0.00461	0.00188	0.00107	0.00242	0.00597	0.00017	0.00501	0.00000
	0.00018	0.00038	0.00000	0.00161							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00182	0.00016	0.00000	0.00000	0.00000	0.00640	0.00543	0.03636	0.01853	0.03883
	0.03746	0.07167	0.05940	0.05665	0.07540	0.06684	0.04077	0.03797	0.04157	0.04402	0.04001
	0.03322	0.02069	0.03726	0.01062	0.03432	0.01590	0.02272	0.00784	0.01499	0.01931	0.00765
	0.01896	0.00163	0.01446	0.00149	0.00147	0.00404	0.00941	0.00748	0.00228	0.00218	0.00000
	0.00298	0.00064	0.00000	0.02916							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00322
	0.00534	0.00374	0.00054	0.00046	0.00000	0.00221	0.00049	0.00240	0.02536	0.05333	0.08085
	0.08074	0.05542	0.04998	0.07829	0.04539	0.07263	0.03860	0.05265	0.04971	0.04203	0.03037
	0.01999	0.02540	0.02659	0.01547	0.00964	0.02176	0.00677	0.00868	0.00554	0.00404	0.00537
	0.01344	0.00625	0.01082	0.00394	0.01165	0.00115	0.00583	0.00144	0.00484	0.00547	0.00387
	0.00018	0.00307	0.00363	0.00143							
0.00000	0.00000	0.00000	0.00000	0.00048	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00123	0.00308	0.00074	0.00000	0.00010	0.00000	0.00144	0.00275	0.00971	0.02567	0.04282
	0.06192	0.04030	0.04061	0.07217	0.06010	0.06546	0.07764	0.08287	0.06096	0.06891	0.04651
	0.04245	0.02445	0.02763	0.02772	0.01940	0.01875	0.00541	0.00600	0.01018	0.00547	0.01031
	0.01033	0.01107	0.00062	0.00003	0.00039	0.00209	0.00676	0.00062	0.00089	0.00265	0.00047
	0.00001	0.00011	0.00009	0.00063							
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	0.00000	0.00000	0.00000	0.00000	0.00035	0.00000	0.00000	0.00302	0.00570	0.02687	0.02475

0.03863 0.03702 0.03266 0.04751 0.05193 0.04186 0.05169 0.13013 0.06141 0.06881 0.05134
0.05359 0.03779 0.04074 0.02812 0.02041 0.02346 0.01733 0.01395 0.01846 0.01992 0.00278
0.00382 0.00747 0.00502 0.00566 0.00204 0.00043 0.00166 0.00106 0.00208 0.00170 0.00664
0.00047 0.00113 0.00000 0.01060
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00103 0.00000 0.00000 0.00079 0.00158 0.00133 0.01567 0.01560 0.03844
0.04316 0.06550 0.04940 0.06701 0.04070 0.07488 0.05001 0.04138 0.08343 0.06439 0.03631
0.02398 0.03855 0.04043 0.02761 0.02969 0.03126 0.02057 0.01337 0.00524 0.01096 0.00530
0.02370 0.00270 0.00915 0.00015 0.01375 0.00566 0.00210 0.00013 0.00223 0.00045 0.00186
0.00000 0.00023 0.00000 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 0.00096 0.00711 0.00695 0.02757 0.03140 0.04730
0.05472 0.08114 0.06290 0.09150 0.06551 0.06711 0.03576 0.04592 0.05671 0.06726 0.02084
0.01748 0.04095 0.01561 0.00974 0.00588 0.00390 0.02954 0.00913 0.01887 0.00529 0.00817
0.00313 0.00052 0.02192 0.00274 0.00370 0.00400 0.01795 0.00198 0.00189 0.00054 0.00063
0.00122 0.00157 0.00121 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 0.00018 0.00120 0.00054 0.01188 0.03671 0.04658
0.05473 0.05092 0.03581 0.04441 0.01769 0.03531 0.04803 0.05045 0.03372 0.08380 0.04364
0.03401 0.05246 0.04146 0.02966 0.01804 0.04339 0.01553 0.00318 0.02085 0.02692 0.00335
0.01061 0.01710 0.00051 0.00349 0.00961 0.01554 0.00196 0.00348 0.00855 0.00031 0.00216
0.00168 0.01032 0.00135 0.02888
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.01089 0.00596 0.03503 0.03401
0.06351 0.03662 0.05821 0.04943 0.05624 0.06274 0.06357 0.11963 0.03531 0.03934 0.04983
0.04513 0.04570 0.01559 0.04791 0.03726 0.00903 0.00949 0.00889 0.00099 0.01493 0.01587
0.00023 0.00129 0.00075 0.00562 0.00127 0.00006 0.00954 0.00000 0.00092 0.00595 0.00000
0.00000 0.00000 0.00000 0.00325
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00099 0.00069 0.00432 0.01325 0.04260
0.05277 0.03607 0.01767 0.04151 0.02576 0.05011 0.03822 0.06859 0.11604 0.05742 0.05677
0.04746 0.03991 0.04610 0.04324 0.03105 0.02549 0.02032 0.02133 0.01262 0.00502 0.00326
0.00320 0.00573 0.01153 0.01417 0.00095 0.00825 0.00554 0.01063 0.00413 0.00000 0.00000
0.00020 0.00053 0.00000 0.01654
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00176 0.00024 0.00543 0.04050 0.04722
0.05039 0.07041 0.07846 0.04631 0.03953 0.03323 0.06318 0.01430 0.03488 0.05173 0.05877
0.02955 0.05761 0.03887 0.02850 0.02106 0.04339 0.02396 0.01928 0.01096 0.00739 0.01159
0.00923 0.01099 0.00841 0.00813 0.00648 0.00310 0.01022 0.00380 0.00457 0.00381 0.00049
0.00148 0.00000 0.00000 0.00077

#Starting and ending year of mrfss age composition data
1988
2007
#sample sizes of mrfss age comp data by year
108 34 271 192 194 102 171 69 77 307 214 88 129 46 203 234 230 204 251 181
#mrfss age comps (year,lengthbin)
0.00082 0.17895 0.48583 0.27450 0.03487 0.02124 0.00000 0.00137 0.00241 0.00000 0.00000
0.00000 0.33417 0.18757 0.21873 0.14522 0.08203 0.02516 0.00678 0.00033 0.00000 0.00000
0.03201 0.67563 0.20889 0.07059 0.00420 0.00668 0.00149 0.00050 0.00001 0.00000 0.00000
0.04746 0.66547 0.18441 0.08063 0.00938 0.00745 0.00181 0.00339 0.00000 0.00000 0.00000
0.00820 0.60382 0.25076 0.06385 0.04269 0.00955 0.01599 0.00363 0.00150 0.00000 0.00000
0.06474 0.57364 0.21403 0.04417 0.06521 0.01775 0.00886 0.01020 0.00085 0.00055 0.00000
0.02423 0.72100 0.15734 0.07717 0.02026 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.21005 0.61921 0.14782 0.00000 0.02292 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.57346 0.31496 0.01550 0.05285 0.04323 0.00000 0.00000 0.00000 0.00000 0.00000
0.04840 0.66870 0.23526 0.02918 0.00515 0.00892 0.00439 0.00000 0.00000 0.00000 0.00000
0.06628 0.46870 0.32545 0.08796 0.03585 0.00229 0.00118 0.01116 0.00111 0.00000 0.00002
0.00000 0.87557 0.03425 0.05508 0.01253 0.00674 0.01118 0.00465 0.00000 0.00000 0.00000
0.00000 0.53609 0.28063 0.04620 0.05270 0.01928 0.01402 0.03953 0.00432 0.00286 0.00438
0.10309 0.37022 0.26540 0.10489 0.03117 0.03475 0.06742 0.00133 0.02172 0.00000 0.00000
0.07538 0.41333 0.22398 0.17284 0.08558 0.02181 0.00061 0.00173 0.00062 0.00384 0.00029
0.03765 0.87137 0.07961 0.00740 0.00245 0.00148 0.00000 0.00004 0.00000 0.00000 0.00000
0.04448 0.72650 0.10940 0.08447 0.02717 0.00282 0.00515 0.00000 0.00000 0.00000 0.00000
0.01894 0.91732 0.02493 0.01718 0.00656 0.01005 0.00499 0.00000 0.00002 0.00000 0.00000
0.11140 0.76966 0.04942 0.02995 0.02341 0.00067 0.01484 0.00064 0.00000 0.00000 0.00000
0.16387 0.75420 0.05543 0.01136 0.00707 0.00404 0.00000 0.00404 0.00000 0.00000 0.00000

################## Shrimp Bycatch ###
#starting, ending years for shrimp bycatch
1950
2007
#bycatch estimates (in thousands)
11122 8316 6343 11122 9231 9267 6448 9223 2969 6818 11122 752 7003 752 752 6879 1241 752
4850 7951 872 11122 6184 5360 7924 5749 6895 752 752 1515 5614 752 6863 7430 752 8149
6102 4606 6205 11122 11097 11122 7388 2377 631 7983 511 3382 417 7005 6341 1416 266
363 130 451 116 451
#CV's for bycatch
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
################################## SEAMAP ################################

```
#SEAMAP YOY Index
#Starting and ending years of CPUE index
1989
2007
#Observed CPUE and assumed CVs
1.04 1.45 1.94 1.14 0.69 0.68 1.21 0.73 0.26 0.59 0.79 1.26 1.86 1.05 0.54 0.62 0.91 1.15 1.11
0.38 0.23 0.21 0.34 0.16 0.22 0.17 0.22 0.31 0.19 0.24 0.24 0.30 0.20 0.24 0.16 0.20 0.18 0.16
#1 Yr Old index
#Starting and ending years of CPUE index
1990
2007
#Observed CPUE and assumed CVs
0.93 0.69 1.78 0.55 1.16 0.55 1.02 0.74 2.39 1.69 1.83 0.82 0.60 0.62 0.86 0.61 1.14 0.71
0.31 0.26 0.15 0.44 0.17 0.29 0.29 0.45 0.64 0.32 0.23 0.32 0.25 0.29 0.29 0.48 0.31 0.21
################Likelihood Component
Weighting####################################################################################
#Weights in objective fcn
1000.0		#landings
1000.0		#discards
0.05				#length comps
1.0				#age comps
0.0	#NC logbook hand lines index
100.0		#FL hand lines cpue index
0.0	#FL gillnet cpue index - pre netban
0.0	#FL gillnet cpue index - post netban
0.0	#NC gillnet logbook index
0.0	#castnet index
0.0	#MRFSS index
100.0	#SEAMAP YOY Index
0.0	#SEAMAP 1-Yr-Old Index
1.0		#S-R residuals
0.0				#constrain first several years of recruitment variability
1.0		#additional constraint on variability of recruitment in last three yrs
0.0				#constrain variability of F in last three years
0.0				#constraint B1/B0
1.0		#penalty if F exceeds 5.0
0.0				#penalty on deviation in CV at age
0.0		#penalty on first difference in CV at age
################Parameter values and initial guesses##################################################################################
#steepness (fixed or initial guess)
0.75
#natural mortality at age
0.50 0.41 0.36 0.33 0.31 0.30 0.29 0.29 0.29 0.28 0.28
#age-independent natural mortality (used only to compute MSST=(1-M)SSBmsy)
0.35
#log catchabilities (initial guesses)
-11.0
-11.0
-11.0
-11.0
-11.0
-11.0
-11.0
-11.0
-11.0
#historical full F
0.20
#FL hand lines
#log mean F - landings
-4
#F deviations (HL-L)
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 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
#log mean F - discards
-4.0  #hand lines - landings
#F deviations (HL-D)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#log mean F - PN
-4
#F deviations (PN)
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 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 0 0 0 0 0 0 0 0
#log mean F - GN
-2
#F deviations (GN)
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 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 0 0 0 0 0 0 0 0
#log mean F - GN - discards
-5
#F deviations (GN - discards)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#log mean F - CN
-4
```

#F deviations (CN)
0 0 0 0 0 0 0 0 0 0 0 0 0
#log mean F - MRFSS
-2.0
#F deviations (MRFSS)
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 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 0 0 0 0 0 0 0 0
#log mean F - MRFSS - discards
-4.0
#F deviations (MRFSS discards)
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 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 0 0 0 0 0 0 0 0
#log mean F - shrimp
-1.0
#F deviations (shrimp)
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 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 0 0 0 0 0 0 0 0
#log_R0 - log virgin recruitment
18
R autocorrelation
0.0
#Selectivity parameters.
2.0 #selpar_L50_HL_keep b1
2.0 #selpar_slope_HL   b3
2.0 #selpar_L50_PN_keep b1
2.0 #selpar_slope_PN   b3
2.0 #selpar_L50_GN pre netban b1
2.0 #selpar_slope_GN pre netban   b3
2.0 #selpar_L50_GN post netban ascending b1
2.0 #selpar_slope_GN post netban ascending   b3
2.0 #selpar_L50_GN post netban descending b1
2.0 #selpar_slope_GN post netban descending   b3
2.0 #selpar_L50_CN b1
2.0 #selpar_slope_CN   b3
2.0 #selpar_L50_MRFSS_keep b1
2.0 #selpar_slope_MRFSS   b3
0.0 #selpar_age0_MRFSS
1.0 #selpar_age1_MRFSS
0.7 #selpar_age2_MRFSS
0.5 #selpar_age3_MRFSS
0.05 0.5 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 #historical selectivity - females
1.00 0.48 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 #selectivity vector for discards - HL - females b1
1.00 0.64 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 #selectivity vector for discards - HL - males
1.00 0.33 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 #selectivity vector for discards - GN - females b1
1.00 0.40 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 #selectivity vector for discards - GN - males
1.00 0.34 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 #selectivity vector for discards - MRFSS - females b1
1.00 0.41 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 #selectivity vector for discards - MRFSS - males
1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 #selectivity vector for shrimp bycatch
0.2 #init_number set_L50_diff: difference between males and females
#Aging Error Matrix
1 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 1
historic recreational landings multiplier
1.0
999 #end of data file flag

III. File to create R object (sm_make_Robject1.cxx).

```
// Example file to 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", "Spanish Mackerel");
                wrt_r_item("model", "Statistical Catch at Age");
                wrt_r_item("rec.model", "BH-steep");
                wrt_r_item("base.run", "sm00X.tpl");
                wrt_r_item("units.length", "cm");
                wrt_r_item("units.weight0", "kg");
                wrt_r_item("units.biomass", "metric tons");
                wrt_r_item("units.ssb", "metric tons");
                wrt_r_item("units.ypr", "metric tons");
                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("prop.F.age0",prop_f_a0);
//              //wrt_r_item("M", M);
                wrt_r_item("Linf1", set_Linf_m);
                wrt_r_item("K1", set_K_m);
                wrt_r_item("t01", set_t0_m);
                wrt_r_item("Linf2", set_Linf_f);
                wrt_r_item("K2", set_K_f);
                wrt_r_item("t02", set_t0_f);
                wrt_r_item("len.cv",mfexp(log_len_cv));
                wrt_r_item("wgt.a", wgtpar_a);
                wrt_r_item("wgt.b", wgtpar_b);
   //wrt_r_item("whole2gutted", wgtpar_whole2gutted);
                wrt_r_item("q.LB.HL", mfexp(log_q_LB_HL));
                wrt_r_item("q.FL.gill1", mfexp(log_q_FL_gill1));
                wrt_r_item("q.FL.gill2", mfexp(log_q_FL_gill2));
                wrt_r_item("q.LB.gill", mfexp(log_q_LB_gill));
                wrt_r_item("q.FL.HL", mfexp(log_q_FL_HL));
                wrt_r_item("q.CN", mfexp(log_q_CN));
                wrt_r_item("q.MRFSS", mfexp(log_q_MRFSS));
                wrt_r_item("q.SMAP.YOY", mfexp(log_q_SMAP_YOY));
                wrt_r_item("q.SMAP.1YR", mfexp(log_q_SMAP_1YR));
                wrt_r_item("q.rate",q_rate);
                wrt_r_item("F.prop.HL", F_HL_prop);
                wrt_r_item("F.prop.PN", F_PN_prop);
                wrt_r_item("F.prop.GN", F_GN_prop);
                wrt_r_item("F.prop.CN", F_CN_prop);
                wrt_r_item("F.prop.MRFSS", F_MRFSS_prop);
                wrt_r_item("F.prop.HL.D", F_HL_D_prop);
                wrt_r_item("F.prop.GN.D", F_GN_D_prop);
                wrt_r_item("F.prop.MRFSS.D", F_MRFSS_D_prop);
                wrt_r_item("F.prop.shrimp", F_shrimp_prop);
                wrt_r_item("F.hist",F_hist);
                wrt_r_item("Dmort.HL",Dmort_HL);
                wrt_r_item("Dmort.GN",Dmort_GN);
                wrt_r_item("Dmort.MRFSS",Dmort_MRFSS);
                wrt_r_item("BH.Phi0", spr_F0);
                wrt_r_item("BPR.F0",bpr_F0);
                wrt_r_item("SPR.Fhist",spr_F_init);
   wrt_r_item("B0",B0);
                wrt_r_item("SSB0", S0);
                wrt_r_item("SSBstyr.SSB0", SSB(styr)/S0);
//              wrt_r_item("S1.S0", S1/S0);
                wrt_r_item("Rstyr.R0", rec(styr)/R0);
//              wrt_r_item("R1.R0", R1/R0);
                wrt_r_item("BH.biascorr",BiasCor);
```

```
                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("R1",R1);
                wrt_r_item("rec.lag", 0.0);
                wrt_r_item("msy", msy_out);
                wrt_r_item("Fmsy", F_msy_out);
                wrt_r_item("Emsy", E_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("prop.f.a0",prop_f_a0);
                //wrt_r_item("L.bias", L_early_bias);
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.LB.HL", f_LB_HL_cpue*w_I_LB_HL);
                wrt_r_item("lk.U.FL.gill1", f_FL_gill1_cpue*w_I_FL_gill1);
                wrt_r_item("lk.U.FL.gill2", f_FL_gill2_cpue*w_I_FL_gill2);
                wrt_r_item("lk.U.LB.gill", f_LB_gill_cpue*w_I_LB_gill);
                wrt_r_item("lk.U.FL.HL", f_FL_HL_cpue*w_I_FL_HL);
                wrt_r_item("lk.U.CN", f_CN_cpue*w_I_CN);
                wrt_r_item("lk.U.MRFSS", f_MRFSS_cpue*w_I_MRFSS);
                wrt_r_item("lk.U.SMAP.YOY", f_SMAP_YOY_cpue*w_I_SMAP_YOY);
                wrt_r_item("lk.U.SMAP.1YR", f_SMAP_1YR_cpue*w_I_SMAP_1YR);
    wrt_r_item("lk.L.HL", f_HL_L*w_L);
    wrt_r_item("lk.L.PN", f_PN_L*w_L);
    wrt_r_item("lk.L.GN", f_GN_L*w_L);
    wrt_r_item("lk.L.CN", f_CN_L*w_L);
    wrt_r_item("lk.L.MRFSS", f_MRFSS_L*w_L);
    wrt_r_item("lk.D.HL", f_HL_D*w_D);
    wrt_r_item("lk.D.GN", f_GN_D*w_D);
    wrt_r_item("lk.D.MRFSS", f_MRFSS_D*w_D);
    wrt_r_item("lk.B.shrimp",f_shrimp_B*w_D);
    wrt_r_item("lk.lenc.HL", f_HL_lenc*w_lc);
    wrt_r_item("lk.lenc.PN", f_PN_lenc*w_lc);
    wrt_r_item("lk.lenc.GN", f_GN_lenc*w_lc);
    wrt_r_item("lk.lenc.CN", f_CN_lenc*w_lc);
    wrt_r_item("lk.lenc.MRFSS", f_MRFSS_lenc*w_lc);
    wrt_r_item("lk.agec.HL", f_HL_agec*w_ac);
    wrt_r_item("lk.agec.PN", f_PN_agec*w_ac);
    wrt_r_item("lk.agec.GN", f_GN_agec*w_ac);
    wrt_r_item("lk.agec.CN", f_CN_agec*w_ac);
    wrt_r_item("lk.agec.MRFSS", f_MRFSS_agec*w_ac);
    wrt_r_item("lk.SRfit", f_N_dev*w_R);
    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.fullF", f_fullF_constraint*w_fullF);
    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.I.LB.HL", w_I_LB_HL);
                wrt_r_item("w.I.FL.gill1", w_I_FL_gill1);
                wrt_r_item("w.I.FL.gill2", w_I_FL_gill2);
                wrt_r_item("w.I.LB.gill", w_I_LB_gill);
                wrt_r_item("w.I.FL.HL", w_I_FL_HL);
                wrt_r_item("w.I.CN", w_I_CN);
                wrt_r_item("w.I.MRFSS", w_I_MRFSS);
                wrt_r_item("w.I.SMAP.YOY", w_I_SMAP_YOY);
                wrt_r_item("w.I.SMAP.1YR", w_I_SMAP_1YR);
                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.HL", selpar_L50_HL_keep);
                wrt_r_item("selpar.slope.HL", selpar_slope_HL);
                wrt_r_item("selpar.L50.PN", selpar_L50_PN);
                wrt_r_item("selpar.slope.PN", selpar_slope_PN);
                wrt_r_item("selpar.L50.GN", selpar_L50_GN_keep);
                wrt_r_item("selpar.slope.GN", selpar_slope_GN);
                wrt_r_item("selpar.L50.CN", selpar_L50_CN);
                wrt_r_item("selpar.slope.CN", selpar_slope_CN);
                wrt_r_item("selpar.L50.MRFSS", selpar_L50_MRFSS_keep);
                wrt_r_item("selpar.slope.MRFSS", selpar_slope_MRFSS);

close_r_vector();

//// Example of a MATRIX object with no row or column names
//wrt_r_matrix("F.at.age", F);

//// Example of a MATRIX object with row names, but no column names;
//// rows specified by constructing a series since a vector of years
//// is not a variable
//wrt_r_matrix("N.at.age.1", N, 2);
//wrt_r_namevector(1968, 1979);
//

// Matrix with with row and column names;
// rows specified by constructing a series since the a vector of years
// is not a variable; columns specified by constructing a series since
// a vector of ages isn't supplied
   open_r_matrix("prop.F");
   wrt_r_matrix(prop_f, 2, 2);
   wrt_r_namevector(styr, (endyr+1));
   wrt_r_namevector(1, nages);
   close_r_matrix();

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

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

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

   open_r_matrix("N.M.age.mdyr");
   wrt_r_matrix(N_M_mdyr, 2, 2);
   wrt_r_namevector(styr, (endyr+1));
   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.F.age");
   wrt_r_matrix(Z_F, 2, 2);
   wrt_r_namevector(styr, endyr);
   wrt_r_namevector(1, nages);
   close_r_matrix();

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

open_r_matrix("L.age.pred.wgt.mt");
wrt_r_matrix(L_total, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(1, nages);
close_r_matrix();

//open_r_matrix("L.age.pred.wgt.klb");
//wrt_r_matrix(L_total*mt2klb, 2, 2);
//wrt_r_namevector(styr,endyr);
//wrt_r_namevector(1, nages);
//close_r_matrix();

open_r_matrix("C.age.pred.num.HL.M");
wrt_r_matrix(C_HL_M, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.HL.F");
wrt_r_matrix(C_HL_F, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.PN.M");
wrt_r_matrix(C_PN_M, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.PN.F");
wrt_r_matrix(C_PN_F, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.GN.M");
wrt_r_matrix(C_GN_M, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.GN.F");
wrt_r_matrix(C_GN_F, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.CN.M");
wrt_r_matrix(C_CN_M, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.CN.F");
wrt_r_matrix(C_CN_F, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.MRFSS.M");
wrt_r_matrix(C_MRFSS_M, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.MRFSS.F");
wrt_r_matrix(C_MRFSS_F, 2, 2);
wrt_r_namevector(styr, endyr);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("C.age.pred.num.HL.D.M");
wrt_r_matrix(C_HL_D_M, 2, 2);
```

```
    wrt_r_namevector(styr_HL_D, endyr_HL_D);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

    open_r_matrix("C.age.pred.num.HL.D.F");
    wrt_r_matrix(C_HL_D_F, 2, 2);
    wrt_r_namevector(styr_HL_D, endyr_HL_D);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

    open_r_matrix("C.age.pred.num.GN.D.M");
    wrt_r_matrix(C_GN_D_M, 2, 2);
    wrt_r_namevector(styr_GN_D, endyr_GN_D);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

    open_r_matrix("C.age.pred.num.GN.D.F");
    wrt_r_matrix(C_GN_D_F, 2, 2);
    wrt_r_namevector(styr_GN_D, endyr_GN_D);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

    open_r_matrix("C.age.pred.num.MRFSS.D.M");
    wrt_r_matrix(C_MRFSS_D_M, 2, 2);
    wrt_r_namevector(styr_MRFSS_D, endyr_MRFSS_D);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

    open_r_matrix("C.age.pred.num.MRFSS.D.F");
    wrt_r_matrix(C_MRFSS_D_F, 2, 2);
    wrt_r_namevector(styr_MRFSS_D, endyr_MRFSS_D);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

    open_r_matrix("C.age.pred.num.shrimp.D.M");
    wrt_r_matrix(C_shrimp_M, 2, 2);
    wrt_r_namevector(styr_shrimp_B, endyr_shrimp_B);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

    open_r_matrix("C.age.pred.num.shrimp.D.F");
    wrt_r_matrix(C_shrimp_F, 2, 2);
    wrt_r_namevector(styr_shrimp_B, endyr_shrimp_B);
    wrt_r_namevector(0, nages-1);
    close_r_matrix();

// LIST object with annual selectivity at age by fishery

open_r_list("sel.age");

    open_r_vector("sel.v.HL.M");
    for (iage=1; iage<=nages; iage++)
    {
      wrt_r_item(iage-1, sel_HL_keep_M(iage));
    }
    close_r_vector();

    open_r_vector("sel.v.HL.F");
    for (iage=1; iage<=nages; iage++)
    {
      wrt_r_item(iage-1, sel_HL_keep_F(iage));
    }
    close_r_vector();

    open_r_vector("sel.v.HL.D.M");
    for (iage=1; iage<=nages; iage++)
    {
      wrt_r_item(iage-1, sel_HL_D_M(iage));
    }
    close_r_vector();

    open_r_vector("sel.v.HL.D.F");
    for (iage=1; iage<=nages; iage++)
    {
      wrt_r_item(iage-1, sel_HL_D_F(iage));
    }
    close_r_vector();

    open_r_vector("sel.v.PN.M");
    for (iage=1; iage<=nages; iage++)
```

```
{
   wrt_r_item(iage-1, sel_PN_M(iage));
}
close_r_vector();

open_r_vector("sel.v.PN.F");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_PN_F(iage));
}
close_r_vector();

open_r_vector("sel.v.GN.M.1950");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_GN_keep_M(iage));
}
close_r_vector();

open_r_vector("sel.v.GN.F.1950");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_GN_keep_F(iage));
}
close_r_vector();

open_r_vector("sel.v.GN.M.1995");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_GN_keep_M2(iage));
}
close_r_vector();

open_r_vector("sel.v.GN.F.1995");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_GN_keep_F2(iage));
}
close_r_vector();

open_r_vector("sel.v.GN.D.M");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_GN_D_M(iage));
}
close_r_vector();

open_r_vector("sel.v.GN.D.F");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_GN_D_F(iage));
}
close_r_vector();

open_r_vector("sel.v.CN.M");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_CN_M(iage));
}
close_r_vector();

open_r_vector("sel.v.CN.F");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_CN_F(iage));
}
close_r_vector();

open_r_vector("sel.v.MRFSS.M");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_MRFSS_keep_M(iage));
}
close_r_vector();

open_r_vector("sel.v.MRFSS.F");
for (iage=1; iage<=nages; iage++)
{
   wrt_r_item(iage-1, sel_MRFSS_keep_F(iage));
}
```

```
    close_r_vector();

    open_r_vector("sel.v.MRFSS.D.M");
    for (iage=1; iage<=nages; iage++)
    {
       wrt_r_item(iage-1, sel_MRFSS_D_M(iage));
    }
    close_r_vector();

    open_r_vector("sel.v.MRFSS.D.F");
    for (iage=1; iage<=nages; iage++)
    {
       wrt_r_item(iage-1, sel_MRFSS_D_F(iage));
    }
    close_r_vector();

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

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

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

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

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

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

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

close_r_list();


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

    open_r_matrix("lcomp.HL.ob");
    wrt_r_matrix(obs_HL_lenc, 2, 2);
    wrt_r_namevector(yrs_HL_lenc);
    wrt_r_namevector(lenbins);
    close_r_matrix();

    open_r_matrix("lcomp.HL.pr");
    wrt_r_matrix(pred_HL_lenc, 2, 2);
    wrt_r_namevector(1,nyr_HL_lenc);
```

```
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.PN.ob");
wrt_r_matrix(obs_PN_lenc, 2, 2);
wrt_r_namevector(styr_PN_lenc,endyr_PN_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.PN.pr");
wrt_r_matrix(pred_PN_lenc, 2, 2);
wrt_r_namevector(styr_PN_lenc,endyr_PN_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.GN.ob");
wrt_r_matrix(obs_GN_lenc, 2, 2);
wrt_r_namevector(styr_GN_lenc,endyr_GN_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.GN.pr");
wrt_r_matrix(pred_GN_lenc, 2, 2);
wrt_r_namevector(styr_GN_lenc,endyr_GN_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.CN.ob");
wrt_r_matrix(obs_CN_lenc, 2, 2);
wrt_r_namevector(yrs_CN_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.CN.pr");
wrt_r_matrix(pred_CN_lenc, 2, 2);
wrt_r_namevector(1,nyr_CN_lenc);
wrt_r_namevector(lenbins);
close_r_matrix();

open_r_matrix("lcomp.MRFSS.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.MRFSS.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("acomp.HL.ob");
wrt_r_matrix(obs_HL_agec, 2, 2);
wrt_r_namevector(yrs_HL_agec);
wrt_r_namevector(1, nages);
close_r_matrix();

open_r_matrix("acomp.HL.pr");
wrt_r_matrix(pred_HL_agec, 2, 2);
wrt_r_namevector(1,nyr_HL_agec);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("acomp.PN.ob");
wrt_r_matrix(obs_PN_agec, 2, 2);
wrt_r_namevector(yrs_PN_agec);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("acomp.PN.pr");
wrt_r_matrix(pred_PN_agec, 2, 2);
wrt_r_namevector(1,nyr_PN_agec);
wrt_r_namevector(0, nages-1);
close_r_matrix();

open_r_matrix("acomp.GN.ob");
wrt_r_matrix(obs_GN_agec, 2, 2);
wrt_r_namevector(styr_GN_agec,endyr_GN_agec);
wrt_r_namevector(0, nages-1);
close_r_matrix();
```

```
  open_r_matrix("acomp.GN.pr");
  wrt_r_matrix(pred_GN_agec, 2, 2);
  wrt_r_namevector(styr_GN_agec,endyr_GN_agec);
  wrt_r_namevector(0, nages-1);
  close_r_matrix();

  open_r_matrix("acomp.CN.ob");
  wrt_r_matrix(obs_CN_agec, 2, 2);
  wrt_r_namevector(yrs_CN_agec);
  wrt_r_namevector(0, nages-1);
  close_r_matrix();

  open_r_matrix("acomp.CN.pr");
  wrt_r_matrix(pred_CN_agec, 2, 2);
  wrt_r_namevector(1,nyr_CN_agec);
  wrt_r_namevector(0, nages-1);
  close_r_matrix();

  open_r_matrix("acomp.MRFSS.ob");
  wrt_r_matrix(obs_MRFSS_agec, 2, 2);
  wrt_r_namevector(styr_MRFSS_agec,endyr_MRFSS_agec);
  wrt_r_namevector(0, nages-1);
  close_r_matrix();

  open_r_matrix("acomp.MRFSS.pr");
  wrt_r_matrix(pred_MRFSS_agec, 2, 2);
  wrt_r_namevector(styr_MRFSS_agec,endyr_MRFSS_agec);
  wrt_r_namevector(0, nages-1);
  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.HL", F_HL_out);
        wrt_r_df_col("F.PN", F_PN_out);
        wrt_r_df_col("F.GN", F_GN_out);
        wrt_r_df_col("F.CN", F_CN_out);
        wrt_r_df_col("F.MRFSS", F_MRFSS_out);
        wrt_r_df_col("F.HL.D", F_HL_D_out);
        wrt_r_df_col("F.GN.D", F_GN_D_out);
        wrt_r_df_col("F.MRFSS.D", F_MRFSS_D_out);
        wrt_r_df_col("F.shrimp.B",F_shrimp_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("B.B0", totB/B0);
  wrt_r_df_col("SPR.static", spr_static);
  wrt_r_df_col("U.LB.HL.ob", obs_LB_HL_cpue);
  wrt_r_df_col("U.LB.HL.pr", pred_LB_HL_cpue);
  wrt_r_df_col("U.FL.HL.ob", obs_FL_HL_cpue);
  wrt_r_df_col("U.FL.HL.pr", pred_FL_HL_cpue);
  wrt_r_df_col("U.FL.gill1.ob", obs_FL_gill1_cpue);
  wrt_r_df_col("U.FL.gill1.pr", pred_FL_gill1_cpue);
  wrt_r_df_col("U.FL.gill2.ob", obs_FL_gill2_cpue);
  wrt_r_df_col("U.FL.gill2.pr", pred_FL_gill2_cpue);
  wrt_r_df_col("U.LB.gill.ob", obs_LB_gill_cpue);
  wrt_r_df_col("U.LB.gill.pr", pred_LB_gill_cpue);
  wrt_r_df_col("U.CN.ob", obs_CN_cpue);
  wrt_r_df_col("U.CN.pr", pred_CN_cpue);
  wrt_r_df_col("U.MRFSS.ob", obs_MRFSS_cpue);
  wrt_r_df_col("U.MRFSS.pr", pred_MRFSS_cpue);
  wrt_r_df_col("U.SMAP.YOY.ob", obs_SMAP_YOY_cpue);
  wrt_r_df_col("U.SMAP.YOY.pr", pred_SMAP_YOY_cpue);
  wrt_r_df_col("U.SMAP.1YR.ob", obs_SMAP_1YR_cpue);
  wrt_r_df_col("U.SMAP.1YR.pr", pred_SMAP_1YR_cpue);
  wrt_r_df_col("total.L.wgt.mt", L_total_yr);
  wrt_r_df_col("total.L.wgt.klb", L_total_yr*mt2klb);
  wrt_r_df_col("total.D.wgt.mt", D_total_yr);
  wrt_r_df_col("total.D.wgt.klb", D_total_yr*mt2klb);
  wrt_r_df_col("total.B.wgt.mt", B_total_yr);
  wrt_r_df_col("total.B.wgt.klb", B_total_yr*mt2klb);
```

```
   wrt_r_df_col("L.HL.ob", obs_HL_L);
   wrt_r_df_col("L.HL.pr", pred_HL_L);
   wrt_r_df_col("L.PN.ob", obs_PN_L);
   wrt_r_df_col("L.PN.pr", pred_PN_L);
   wrt_r_df_col("L.GN.ob", obs_GN_L);
   wrt_r_df_col("L.GN.pr", pred_GN_L);
   wrt_r_df_col("L.CN.ob", obs_CN_L);
   wrt_r_df_col("L.CN.pr", pred_CN_L);
   wrt_r_df_col("L.MRFSS.ob", obs_MRFSS_L);
   wrt_r_df_col("L.MRFSS.pr", pred_MRFSS_L);
   wrt_r_df_col("D.shrimp.B.ob", obs_shrimp_B);
   wrt_r_df_col("D.shrimp.B.pr", pred_shrimp_B);

   wrt_r_df_col("D.HL.ob", obs_HL_D);
   wrt_r_df_col("D.HL.pr", pred_HL_D);
   wrt_r_df_col("D.GN.ob", obs_GN_D);
   wrt_r_df_col("D.GN.pr", pred_GN_D);
   wrt_r_df_col("D.MRFSS.ob", obs_MRFSS_D);
   wrt_r_df_col("D.MRFSS.pr", pred_MRFSS_D);

   //comp sample sizes
   wrt_r_df_col("lcomp.HL.n", nsamp_HL_lenc_allyr);
   wrt_r_df_col("lcomp.PN.n", nsamp_PN_lenc);
   wrt_r_df_col("lcomp.GN.n", nsamp_GN_lenc);
   wrt_r_df_col("lcomp.CN.n", nsamp_CN_lenc_allyr);
   wrt_r_df_col("lcomp.MRFSS.n", nsamp_MRFSS_lenc);
   wrt_r_df_col("acomp.HL.n", nsamp_HL_agec_allyr);
   wrt_r_df_col("acomp.PN.n", nsamp_PN_agec_allyr);
   wrt_r_df_col("acomp.GN.n", nsamp_GN_agec);
   wrt_r_df_col("acomp.CN.n", nsamp_CN_agec_allyr);
   wrt_r_df_col("acomp.MRFSS.n", nsamp_MRFSS_agec);

close_r_df();

open_r_df("a.series", 1, nages, 2);
                wrt_r_namevector(1,nages);
                wrt_r_df_col("age", agebins(1),agebins(nages));
                wrt_r_df_col("length1", meanlen_m);
                wrt_r_df_col("length2", meanlen_f);
                wrt_r_df_col("length.cv", len_cv);
                wrt_r_df_col("weight0.M", wgt_kg_m);     //for FishGraph
                wrt_r_df_col("weight0.F", wgt_kg_f);     //for FishGraph
                wrt_r_df_col("wgt.g.M", wgt_g_m);
                wrt_r_df_col("wgt.g.F", wgt_g_f);
                wrt_r_df_col("wgt.mt.M", wgt_m);
                wrt_r_df_col("wgt.mt.F", wgt_f);
                wrt_r_df_col("wgt.klb.M", wgt_klb_m);
                wrt_r_df_col("wgt.klb.F", wgt_klb_f);

                wrt_r_df_col("prop.female.F0", prop_f_F0);
                wrt_r_df_col("mat.female", maturity_f);
                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("E.eq", E_eq);
                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", L_eq);
                wrt_r_df_col("D.eq", 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("E.spr", E_spr);
                wrt_r_df_col("spr", spr_spr);
                wrt_r_df_col("SPR", spr_spr/spr_F0);
                wrt_r_df_col("ypr", L_spr);
close_r_df();


open_r_list("CLD.est.mats");
   open_r_matrix("Lw.HL");
```

```
      wrt_r_matrix(L_HL, 1,1);
   close_r_matrix();
   open_r_matrix("Lw.PN");
      wrt_r_matrix(L_PN, 1,1);
   close_r_matrix();
   open_r_matrix("Lw.GN");
      wrt_r_matrix(L_GN, 1,1);
   close_r_matrix();
   open_r_matrix("Lw.CN");
      wrt_r_matrix(L_CN, 1,1);
   close_r_matrix();
   open_r_matrix("Lw.MRFSS");
      wrt_r_matrix(L_MRFSS, 1,1);
   close_r_matrix();
   open_r_matrix("Dw.HL");
      wrt_r_matrix(L_HL_D, 1,1);
   close_r_matrix();
   open_r_matrix("Dw.GN");
      wrt_r_matrix(L_GN_D, 1,1);
   close_r_matrix();
   open_r_matrix("Dw.MRFSS");
      wrt_r_matrix(L_MRFSS_D, 1,1);
   close_r_matrix();
   open_r_matrix("Dw.shrimp");
      wrt_r_matrix(L_shrimp, 1,1);
   close_r_matrix();

   open_r_matrix("Ln.HL");
      wrt_r_matrix(C_HL_M+C_HL_F, 1,1);
   close_r_matrix();
   open_r_matrix("Ln.PN");
      wrt_r_matrix(C_PN_M+C_PN_F, 1,1);
   close_r_matrix();
   open_r_matrix("Ln.GN");
      wrt_r_matrix(C_GN_M+C_PN_F, 1,1);
   close_r_matrix();
   open_r_matrix("Ln.CN");
      wrt_r_matrix(C_CN_M+C_CN_F, 1,1);
   close_r_matrix();
   open_r_matrix("Ln.MRFSS");
      wrt_r_matrix(C_MRFSS_M+C_MRFSS_F, 1,1);
   close_r_matrix();
   open_r_matrix("Dn.HL");
      wrt_r_matrix(C_HL_D_M+C_HL_D_F, 1,1);
   close_r_matrix();
   open_r_matrix("Dn.GN");
      wrt_r_matrix(C_GN_D_M+C_GN_D_F, 1,1);
   close_r_matrix();
   open_r_matrix("Dn.MRFSS");
      wrt_r_matrix(C_MRFSS_D_M+C_MRFSS_D_F, 1,1);
   close_r_matrix();
   open_r_matrix("Dn.shrimp");
      wrt_r_matrix(C_shrimp_M+C_shrimp_F, 1,1);
   close_r_matrix();

//    // need to add discards in weight in both TPL and this code
close_r_list();

   open_r_matrix("age.error");
   wrt_r_matrix(set_age_error_matrix, 2, 2);
   wrt_r_namevector(1, nages);
   wrt_r_namevector(1, nages);
   close_r_matrix();

open_r_list("len.prob");

   open_r_matrix("len.prob.m");
   wrt_r_matrix(lenprob_m, 2, 2);
   wrt_r_namevector(1, nages);
   wrt_r_namevector(1, nlenbins);
   close_r_matrix();

   open_r_matrix("len.prob.f");
   wrt_r_matrix(lenprob_f, 2, 2);
   wrt_r_namevector(1, nages);
   wrt_r_namevector(1, nlenbins);
   close_r_matrix();

close_r_list();
```

```
////// Example of a DATA FRAME object composed of two items.
////// The data frame will span 1 to the number of ages. The vectors
////// span 2 to the number of ages, so NAs will be written for the first
////// data point in the two vectors. The row names will be included,
////// similar to the matrix object just before this one.
////open_r_df("aseries", 1, nages, 2);
////              wrt_r_df_col("N.pred", predicted_N);
////              wrt_r_df_col("N.ratio", ratio_N);
////              wrt_r_namevector(3, 9);
////close_r_df();
////
////// Example of a LIST object
////open_r_list("C.at.age.mats");
////
////              // matrix with row and column names
////    wrt_r_matrix("Est", C, 1, 1);
////
////              // another matrix with row and column names
////              wrt_r_matrix("Obs", obs_catch_at_age, 1, 1);
////
////close_r_list();
////
//

// close file
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
//=============================================================================================
```