SEDAR 21-AW-02: Computer code for the SEDAR 21 age-structured catch-free model for dusky sharks

Sustainable Fisheries Branch

National Marine Fisheries Service

Southeast Fisheries Science Center

Beaufort laboratory

101 Pivers Island Road, Beaufort, NC 28516

1 Overview

The model used in this assessment was an age-structured catch-free model, which was originally developed by Porch et al. (2006) for use in a goliath grouper assessment, and which was subsequently applied by Cortes et al. (2006) in a previous assessment of dusky sharks. The model is written in AD Model Builder software (Otter Research 2004), and includes three files. The first file is a .tpl file, which is the "nuts and bolts" of the code, and is provided here in Appendix A. The other two files are essentially input files; the first includes information on data feeding into the dusky shark model (an .inp file; see Appendix B), and the second includes information on the parameters feeding into the model, allowing one to specify bounds and prior distributions on various parameters (a .prm file, see Appendix C). In the spirit of reproducibility, all three files are presented here for completeness.

References

Cortés, E., E. Brooks, P. Apostolaki, and C. Brown, 2006. Stock Assessment of Dusky Shark in the U.S. Atlantic and Gulf of Mexico. Technical report, NMFS/SEFSC Sustainable Fisheries Division Contribution SFD-2006-014.

Ltd, O. R., 2004. An Introduction to AD Model Builder version 7.1.1. Box 2040, Sidney, British Columbia.

Porch, C. E., A. Eklund, and G. P. Scott. 2006. A catch-free stock assessment model with application to goliath grouper (Epinephelus itajara) off southern Florida. Fishery Bulletin **104**:89–101.

Appendix A: AD Model Builder code for estimation

```
 /////////////////////////////////////////////////////////////////////////////////////
DATA_SECTION
 /////////////////////////////////////////////////////////////////////////////////////

 // -------------- read data file ---------------------------------------------------------

 // general information
 //!! ad_comm::change_datafile_name("enric2.dat");

 !! cout << "general information " << endl;
  init_ivector year(1,5)
  init_int year_change
  init_ivector age(1,2)
  init_int nfs                                       // (n)umber of (s)ets of (f) fishing mortality-related parameters (fleets)
  init_int variance_scale                // controls how variance terms are represented (1=log scale, 2=arithmetic scale)
  init_int variance_modify           // + value = add annual modifiers to variance terms, - value = multiply annual modifiers by variance terms
  int year_prehistoric // last year of historical period (hist. period is the time span from virgin levels to when data becomes available)
  int year_modern // last year of early modern period (when data is available)
  int year_modern2                // last year of second modern period
  int nyears_modern                                  // number of years in the first modern period (when F can vary from trend indicated by effort data)
  int nyears_modern2                                  // number of years in the second modern period (when F can vary from trend indicated by effort data)
  int nyears_prehistoric                             // number of years in the prehistoric period (when F varies only as function of effort since little data)
  int nyears_past        // number of years in the prehistoric and two modern periods combined
  int nyears_proj                              // number of years to project into future
  int nyears                                         // number of years in simulation, past and future
  int n_eras                                         // number of time periods when F or q can vary from overall expectations(nyears_modern+nyears_modern2+1)
  int nyears_b4_change                               // number of years between prehistoric period and the time during the modern period when F is suspected to change
  int nages                                          // number of age classes
  int nqs                                            // (n)umber of (s)ets of (q) catchability-related parameters
  int nss                                            // (n)umber of (s)ets of (s) selectivity-related parameters
  int nids                                           // (n)umber of (s)ets of (i) index data-related parameters
 LOCAL_CALCS
  year_prehistoric  = year(2); year_modern=year(3); year_modern2=year(4);
  nyears_prehistoric= year_prehistoric - year(1)+1;
  nyears_modern     = year_modern - year_prehistoric;
  nyears_modern2     = year_modern2 - year_modern;
  nyears_proj       = year(5) - year_modern2;
  nyears_past       = nyears_prehistoric + nyears_modern + nyears_modern2;
  nyears         = nyears_past + nyears_proj;
  if(year_change<0 || year_change>year_modern) nyears_b4_change = nyears_past;
  else nyears_b4_change=year_change-year(1);
  n_eras=nyears_modern+nyears_modern2+1;
  nages=age(2)-age(1)+1;
 END_CALCS

 // spawning information
  init_number spawn_season
  init_vector maturity(1,nages)
  init_vector fecundity_input(1,nages)
 //motality information
  init_number mort_switch  //a 0 means use parametric function in .prm file, a one means use the following M@age vetor  PBC 10/2010
  init_vector M_age(1,nages)
  init_number M_age_indep
  init_number F1999   //switch for whether F in 1999 estimated as free parameter or not  PBC 10/2010

 // index (survey) information

 !! cout << "reading indices " << endl;
```

```
 init_int     n_index_series
!! cout << "n indices " << n_index_series << endl;
 init_ivector index_pdf(1,n_index_series)
 init_ivector index_units(1,n_index_series)
 init_vector  index_season(1,n_index_series)
 init_ivector index_scale(1,n_index_series)
 init_ivector ivs(1,n_index_series)                  // integer vector indexing the set of variance parameters used by each index of abundance
 init_ivector iqs(1,n_index_series)                  // integer vector indexing the set of q parameters used by each index of abundance
 init_ivector iss(1,n_index_series)                  // integer vector indexing the set of selectivity parameters used by each index of abundance
 init_vector sel_age0(1,n_index_series)            // gives selectivty of age zeroes PBC 11/3/2010
 init_matrix  index_obs(1,nyears_past,1,n_index_series+1)
 init_matrix  index_cv(1,nyears_past,1,n_index_series+1)
!! cout << "all indices " << index_obs << endl;
!! cout << " index CVs " << index_cv << endl;
 init_int     effort_pdf
!! cout << "effort pdf " << effort_pdf << endl;
 init_matrix  effort_inp(1,nyears_past,1,nfs+1)
 //proportion effort stuff for weighting removals selectivity PBC 10/2010
 init_int n_avg_f
 init_ivector ifs(1,n_avg_f)
 init_matrix prop_effort(1,nyears_past,1,n_avg_f)

!! cout << "reading projection specifications " << endl;
 init_int    reference_selectivity                    // specifies selectivity vector to use when calculating reference points (1 = fishery vector, 2 = maturity vector
 init_number Bref                                     // specifies biomass reference point
 init_number estimate_r_dev_proj                      // determines whether to estimate recruitment deviations in projections
 init_matrix in_prj(1,nyears_proj,1,3)        // projection specifications for F

// --------------- read parameter file ---------------------------------------------------
!! ad_comm::change_datafile_name("dusky32.prm");
!! cout << "reading parameter specifications " << endl;
 init_int n_par  // number of process parameters
 init_ivector n_sets(1,3)  // number of sets of each type of process parameter
!! nqs=n_sets(1); nss=n_sets(2); nids=n_sets(3);
!! cout << n_sets << endl;
 !! cout << "n_par " << n_par << endl;
 init_matrix par_specs(1,n_par,1,7)  // specifications for process parameters
 init_vector f_rho_specs(1,6)  // specifications for f process error correlation coefficient
 init_vector f_var_specs(1,6)  // specifications for f process error variance
 init_vector f_dev_specs(1,6)  // specifications for f process error deviations
 init_vector r_rho_specs(1,6)  // specifications for r process error correlation coefficient
 init_vector r_var_specs(1,6)  // specifications for r process error variance
 init_vector r_dev_specs(1,6)  // specifications for r process error deviations
 init_vector q_rho_specs(1,6)  // specifications for q process error correlation coefficient
 init_vector q_var_specs(1,6)  // specifications for q process error variance
 init_vector q_dev_specs(1,6)  // specifications for q process error deviations
// --------- derived variables pertaining to parameters that are constant (don't need to be differentiated)----------//
 int i
 int ie
 int k
 int n_series
 int n_par_phase
 ivector n_calls(1,1000)
 ivector npf(1,50)
 ivector nature(1,n_par);
 matrix  index_var(1,nyears_past,1,n_index_series);
 vector  best_guess(1,n_par);
 number  f_rho_best_guess;
 number  f_var_best_guess;
 number  f_dev_best_guess;
 number  r_rho_best_guess;
 number  r_var_best_guess;
 number  r_dev_best_guess;
 number  q_rho_best_guess;
 number  q_var_best_guess;
 number  q_dev_best_guess;
 number  F_best_guess;
 ivector iph(1,n_par);
 int     f_rho_iph;
 int     f_var_iph;
 int     f_dev_iph;
 int     r_rho_iph;
 int     r_var_iph;
 int     r_dev_iph;
 int     q_rho_iph;
 int     q_var_iph;
 int     q_dev_iph;
 int     r_dev_proj_iph;
 int     last_iph;
 ivector pdf(1,n_par);
 int     f_rho_pdf;
 int     f_var_pdf;
 int     f_dev_pdf;
 int     r_rho_pdf;
 int     r_var_pdf;
 int     r_dev_pdf;
 int     q_rho_pdf;
 int     q_var_pdf;
 int     q_dev_pdf;
 int     Trecover;
 vector  cv(1,n_par);
 number  f_rho_cv;
 number  f_var_cv;
 number  f_dev_cv;
 number  r_rho_cv;
 number  r_var_cv;
 number  r_dev_cv;
 number  q_rho_cv;
 number  q_var_cv;
 number  q_dev_cv;
 number  r_dev_proj_cv;
 number spawn_time;
 vector index_time(1,n_index_series);
 matrix effort_obs(1,nfs,1,nyears_past);
```

```
  vector F_proj_cv(1,nyears_proj);
 LOCAL_CALCS
  //cout << "reformat parameter control matrices" << endl;
  if(effort_pdf != 0) for ( k=1; k<=nfs; k++ ) for ( y=1; y<=nyears_past; y++ ) effort_obs(k,y)=effort_inp(y,k);
  else effort_obs=1.0;
  //cout << "effort obs " << effort_obs << endl;
  //cout << "after effort obs " << endl;
  if(effort_pdf != -1) effort_obs/=max(effort_obs);
  if(nyears_proj > 0) F_proj_cv=column(in_prj,2);
  best_guess=column(par_specs,2);  iph=ivector(column(par_specs,5)); pdf=ivector(column(par_specs,6)); cv=column(par_specs,7);  nature=ivector(column(par_specs,1));
  f_rho_best_guess=f_rho_specs(1); f_rho_iph=int(f_rho_specs(4));    f_rho_pdf=int(f_rho_specs(5));    f_rho_cv=f_rho_specs(6);
  f_var_best_guess=f_var_specs(1); f_var_iph=int(f_var_specs(4));    f_var_pdf=int(f_var_specs(5));    f_var_cv=f_var_specs(6);
  f_dev_best_guess=f_dev_specs(1); f_dev_iph=int(f_dev_specs(4));    f_dev_pdf=int(f_dev_specs(5));    f_dev_cv=f_dev_specs(6);
  r_rho_best_guess=r_rho_specs(1); r_rho_iph=int(r_rho_specs(4));    r_rho_pdf=int(r_rho_specs(5));    r_rho_cv=r_rho_specs(6);
  r_var_best_guess=r_var_specs(1); r_var_iph=int(r_var_specs(4));    r_var_pdf=int(r_var_specs(5));    r_var_cv=r_var_specs(6);
  r_dev_best_guess=r_dev_specs(1); r_dev_iph=int(r_dev_specs(4));    r_dev_pdf=int(r_dev_specs(5));    r_dev_cv=r_dev_specs(6);
  q_rho_best_guess=q_rho_specs(1); q_rho_iph=int(q_rho_specs(4));    q_rho_pdf=int(q_rho_specs(5));    q_rho_cv=q_rho_specs(6);
  q_var_best_guess=q_var_specs(1); q_var_iph=int(q_var_specs(4));    q_var_pdf=int(q_var_specs(5));    q_var_cv=q_var_specs(6);
  q_dev_best_guess=q_dev_specs(1); q_dev_iph=int(q_dev_specs(4));    q_dev_pdf=int(q_dev_specs(5));    q_dev_cv=q_dev_specs(6);
  F_best_guess=0.2;
  spawn_time=spawn_season/12.0; index_time=index_season/12.0;
  npf=1; for (int j=1; j<=4;j++) npf(j)=j; // constants and polynomials
  npf(5)=1; npf(6)=2; npf(7)=2; // knife-edge, logistic and gamma selectivity curves
  npf(8)=6; npf(9)=3; // Chapman-Richards and Gompertz growth curves
  npf(12)=2; // power
  npf(15)=5; // double logistic (LIZ added 5/23/2004)
  npf(16)=2; // exponential (LIZ added 4/25/2005)
  for (ie=1; ie<=n_par; ie++) { lower(ie)=par_specs(ie,2); upper(ie)=par_specs(ie,3);}
  last_iph=max(iph);
  if(last_iph<f_rho_iph) last_iph=f_rho_iph; if(last_iph<f_var_iph) last_iph=f_var_iph; if(last_iph<f_dev_iph) last_iph=f_dev_iph;
  if(last_iph<r_rho_iph) last_iph=r_rho_iph; if(last_iph<r_var_iph) last_iph=r_var_iph; if(last_iph<r_dev_iph) last_iph=r_dev_iph;
  if(last_iph<q_rho_iph) last_iph=q_rho_iph; if(last_iph<q_var_iph) last_iph=q_var_iph; if(last_iph<q_dev_iph) last_iph=q_dev_iph;
  last_iph+=1;
  if((estimate_r_dev_proj<=0.000001 && estimate_r_dev_proj>=-0.000001) || nyears_proj<=0) r_dev_proj_iph=-1; else { r_dev_proj_iph=last_iph; r_dev_proj_cv=estimate_r_dev_proj; }
  //cout << r_dev_proj_cv << " " << estimate_r_dev_proj << endl;
  if(nyears_b4_change<=nyears_prehistoric) f_dev_iph=-1;
 END_CALCS

 // --------- derived variables pertaining to the data that are constant (don't need to be differentiated)----------//

  vector index_avg(1,n_index_series+1)
  vector index_min(1,n_index_series+1)
  vector n_index_points(1,n_index_series+1)
  vector one_vector_age(1,nages)
  number aic
  number temp_dble
  number n_data
  number n_iter_pr
 LOCAL_CALCS
  cout << "Averaging and scaling index data" << endl;
  n_index_points=0.0; index_avg=0.0; index_min=1000.0;
  for (series=1; series<=n_index_series;series++) {
    for (y=1; y<=nyears_past;y++) {
      if(index_obs(y,series)>=0) {
        if(index_obs(y,series)>0.0 && index_obs(y,series)<index_min(series)) index_min(series)=index_obs(y,series);
        n_index_points(series) += 1.0 ;
      }
    }
    for (y=1; y<=nyears_past;y++) {
      if(index_pdf(series)==1 && index_obs(y,series)>=0 && index_obs(y,series)<index_min(series)) index_obs(y,series)=index_min(series)/1000.0; // no zero indices for lognormal
      if(index_obs(y,series)>=0) index_avg(series) += index_obs(y,series)/n_index_points(series);
    }
    for (y=1; y<=nyears_past;y++) if(index_units(series)<9 && index_scale(series)>0)  index_obs(y,series) /=  index_avg(series);
  }
  n_data=sum(n_index_points); n_series=n_index_series;
  zero=0.0; one=1.0; n_calls=0; i_zero=0; i_one=1; i_two=2; one_vector_age=one;
  n_iter_pr=100;
 END_CALCS

 ///////////////////////////////////////////////////////////////////////////////////
PARAMETER_SECTION
 // Warning: all variables in this section must be floating point, not integers
 //          integers may be declared locally by use of !! int i   etc..., but these will
 //          not apply outside the parameter section (whereas the ADMB types number, vector
 //          and matrix are global)
 ///////////////////////////////////////////////////////////////////////////////////

 // --------- specify estimated parameters --------------------------------------------------//

 // get parameter bounds
 LOCAL_CALCS
   cout << "specifying parameter bounds " << endl;
   dvector lb(1,n_par); lb=column(par_specs,3); dvector ub(1,n_par); ub=column(par_specs,4);
   double lb_f_rho; lb_f_rho=f_rho_specs(2); double ub_f_rho; ub_f_rho=f_rho_specs(3);
   double lb_f_var; lb_f_var=f_var_specs(2); double ub_f_var; ub_f_var=f_var_specs(3);
   double lb_f;     lb_f=f_dev_specs(2);     double ub_f;     ub_f=f_dev_specs(3);
   double lb_r_rho; lb_r_rho=r_rho_specs(2); double ub_r_rho; ub_r_rho=r_rho_specs(3);
   double lb_r_var; lb_r_var=r_var_specs(2); double ub_r_var; ub_r_var=r_var_specs(3);
   double lb_r;     lb_r=r_dev_specs(2);     double ub_r;     ub_r=r_dev_specs(3);
   double lb_q_rho; lb_q_rho=q_rho_specs(2); double ub_q_rho; ub_q_rho=q_rho_specs(3);
   double lb_q_var; lb_q_var=q_var_specs(2); double ub_q_var; ub_q_var=q_var_specs(3);
   double lb_q;     lb_q=q_dev_specs(2);     double ub_q;     ub_q=q_dev_specs(3);
   double lb_0;     lb_0=0.0001;             double ub_2;     ub_2=2.0;
 END_CALCS

 // set parameter vector to be estimated
 !! cout << "specifying parameters " << nfs << endl;

  init_bounded_number_vector par_est(1,n_par,lb,ub,iph)
  init_bounded_number f_rho(lb_f_rho,ub_f_rho,f_rho_iph)
  init_bounded_number f_var(lb_f_var,ub_f_var,f_var_iph)
  init_bounded_matrix f_devs(1,nfs,nyears_prehistoric+1,nyears_past-1,lb_f,ub_f,f_dev_iph) //changed 12/2010 to not estimate dev in final year
  init_bounded_number r_rho(lb_r_rho,ub_r_rho,r_rho_iph)
  init_bounded_number r_var(lb_r_var,ub_r_var,r_var_iph)
```

```
 init_bounded_vector r_devs(2,n_eras,lb_r,ub_r,r_dev_iph)
 init_bounded_number q_rho(lb_q_rho,ub_q_rho,q_rho_iph)
 init_bounded_number q_var(lb_q_var,ub_q_var,q_var_iph)
 init_bounded_matrix q_devs(1,nqs,2,n_eras,lb_q,ub_q,q_dev_iph)
 init_bounded_number Fspr20(lb_0,ub_2,last_iph)
 init_bounded_number Fspr30(lb_0,ub_2,last_iph)
 init_bounded_number Fspr40(lb_0,ub_2,last_iph)
 init_bounded_number Fspr50(lb_0,ub_2,last_iph)
 init_bounded_number Fspr60(lb_0,ub_2,last_iph)
 init_bounded_vector r_devs_proj(1,nyears_proj,lb_r,ub_r,r_dev_proj_iph)

// --------- derived variables that are functions of the parameters and therefore need derivatives ----------//

!! cout << "declaring state variables " << endl;
 matrix f_apical(1,nfs,1,nyears_past)
 vector r(1,nyears+1)
 matrix q(1,nqs,1,n_eras)

!! cout << "state (process) expectations (deterministic part)" << endl;
 vector r_process(1,nyears_past+1)
 matrix q_process(1,nqs,1,n_eras)
 vector m(1,nages)
 vector w(1,nages)
 vector fecundity(1,nages)
 matrix s(1,nss,1,nages)
 matrix sel_f_apical(1,nyears_past,1,nages);

!! cout << "declare index error parameters" << endl;
 vector i_d_var(1,nids)
 number overall_var
 number var1

!! cout << "declare likelihoods and priors" << endl;
 vector index_lklhd(1,n_index_series+1)
 number f_lklhd
 number r_lklhd
 vector q_lklhd(1,nqs)
 number f_prior
 number f_hist_prior
 number m_prior
 number r_prior
 number w_prior
 number v_prior
 vector q_prior(1,nqs)
 vector s_prior(1,nss)
 vector i_d_prior(1,nids)
 number q_process_prior
 number r_process_prior
 number penalty
 number equilibrium_penalty
 number projection_penalty

!! cout << "declare misc. temporary variables" << endl;
 number pred
 number slope0
 number sprtemp
 number yprtemp
 number yprold
 number ytemp
 number yold
 number var
 number spr0
 number survive
 number plus_age
 number spr20
 number spr30
 number spr40
 number spr50
 number spr60
 number spr01
 number sprmax
 number sprmat
 number ypr20
 number ypr30
 number ypr40
 number ypr50
 number ypr60
 number ypr01
 number yprmax
 number yprmsy
 number yprmat
 number Rspr20
 number Rspr30
 number Rspr40
 number Rspr50
 number Rspr60
 number R01
 number Rmax
 number Rmsy
 number Rmat
 number Bmat
 number Bmax
 number B01
 number Bspr20
 number Bspr30
 number Bspr40
 number Bspr50
 number Bspr60
 vector function_parameter(1,10)
 vector recruitment_parameter(1,10)
 vector f_hist_parameter(1,10)
 vector growth_parameter(1,10)
 vector s_latest(1,nages)
 vector s_equilibrium(1,nages)
```

```
 vector virgin_pred(1,n_index_series)
 matrix index_pred(1,nyears_past,1,n_index_series)
 matrix wbyage(1,nages,1,nyears)
 matrix f(1,nages,1,nyears)
 matrix n(1,nages+1,1,nyears+1)
 vector F_proj(1,nyears_proj)
 vector F_spr(1,n_iter_pr);        //values of full F to be used in per-recruit and equilibrium calculations
 vector spr_vec(1,n_iter_pr);      //spr as a function of F
 vector ypr_vec(1,n_iter_pr);      //ypr as a function of F

 objective_function_value obj_func;
!! cout << "declare standard deviation report variables (for mcmc)" << endl;
!! cout << "     and likeprof variables (for likelihood profile)" << endl;
 //sdreport_matrix mc_N(1,nages+1,1,nyears+1);
 //likeprof_number alpha
 //likeprof_number nat_mort
 sdreport_number  mc_B1975
 //likeprof_number  B1975
 number  B1975
 //sdreport_number  mc_B2009
 sdreport_number mc_B2009
 number nat_mort
 vector pup_survival_annual(1,nyears)
 sdreport_vector ssb(1,nyears)
 //likeprof_number pup_survival
 sdreport_number pup_survival
 sdreport_number mc_pup_survival
 //likeprof_number  I2003
 number I2009
 sdreport_number B2009
 //likeprof_number  B2009
 sdreport_number Bcurrent
 sdreport_number Bmsy
 sdreport_number Bmsst
 sdreport_number SSBmsy
 sdreport_number BBmsy
 sdreport_number BBmsst
 //likeprof_number BBmsy
 //likeprof_number BBmsst
 //likeprof_number  I1975
 number I1975
 sdreport_number sprmsy
 sdreport_number Fcurrent
 sdreport_number F2009
 likeprof_number Fmsy
 sdreport_number FFmsy
 sdreport_number alpha
 sdreport_number steepness

 sdreport_number Fmat
 sdreport_number Fmax
 sdreport_number F01
 number BoverBspr20
 number BoverBspr30
 number BoverBspr40
 number BoverBspr50
 number BoverBspr60
 number BoverBmat
 number BoverBmax
 number BoverB01
 number FoverFspr20
 number FoverFspr30
 number FoverFspr40
 number FoverFspr50
 number FoverFspr60
 number FoverFmat
 number FoverFmax
 number FoverF01
 vector B(1,nyears)
 vector BoverBref(1,nyears)
 vector log_F_apex(1,nyears)
   number  Bpro_5
   number  Bpro_4
   number  Bpro_3
   number  Bpro_2
   number  Bpro_1
   number  Bpro0
   number  Bpro1
   number  Bpro2
   number  Bpro3
   number  Bpro4
   number  Bpro5
   number  Bpro6
   number  Bpro7
   number  Bpro8
   number  Bpro9
   number  Bpro10
   number  Bpro11
   number  Bpro12
   number  Bpro13
   number  Bpro14
   number  Bpro15
   number  Bpro16
   number  Bpro17
   number  Bpro18
   number  Bpro19
   number  Bpro20
   number  Bpro21
   number  Bpro22
   number  Bpro23
   number  Bpro24
   number  Bpro25
   number  Bpro26
   number  Bpro27
```

```
    number  Bpro28
    number  Bpro29
    number  Bpro30
//  number  Bpro_5
//  number  Bpro_4
//  number  Bpro_3
//  number  Bpro_2
//  number  Bpro_1
//  number  Bpro0
//  number  Bpro1
//  number  Bpro2
//  number  Bpro3
//  number  Bpro4
//  number  Bpro5
//  number  Bpro6
//  number  Bpro7
//  number  Bpro8
//  number  Bpro9
//  number  Bpro10
//  number  Bpro11
//  number  Bpro12
//  number  Bpro13
//  number  Bpro14
//  number  Bpro15
  number Bvirgin

 !! cout << "Initialize parameters" << endl;

 ////////////////////////////////////////////////////////////////////////////////////
INITIALIZATION_SECTION
 ////////////////////////////////////////////////////////////////////////////////////
   par_est best_guess
   f_rho   f_rho_best_guess
   f_var   f_var_best_guess
   f_devs  f_dev_best_guess
   r_rho   r_rho_best_guess
   r_var   r_var_best_guess
   r_devs  r_dev_best_guess
   q_rho   q_rho_best_guess
   q_var   q_var_best_guess
   q_devs  q_dev_best_guess
   Fspr20  F_best_guess
   Fspr30  F_best_guess
   Fspr40  F_best_guess
   Fspr50  F_best_guess
   Fspr60  F_best_guess
   r_devs_proj  r_dev_best_guess


 ////////////////////////////////////////////////////////////////////////////////////
PROCEDURE_SECTION
 ////////////////////////////////////////////////////////////////////////////////////
  //cout<<"define parameters"<<endl;
  define_parameters();
  //cout<<"calc biomass"<<endl;
  calculate_biomass();
  //cout<<"calc obj fun"<<endl;
  calculate_the_objective_function();
  if(mceval_phase()) outputMCMC();
  //cout<<"procedure section done"<<endl;

 ////////////////////////////////////////////////////////////////////////////////////
 // FUNCTION SECTION
 ////////////////////////////////////////////////////////////////////////////////////

 //----------------------------------------------------------------------------------
FUNCTION define_parameters
 // defines process parameters and computes priors
 // this is quite complicated as you don't want to include priors in phases they aren't being estimated in!
 //----------------------------------------------------------------------------------
   int j, y, inow, i_in, ihist;
   //cout<<"define_parameters"<<endl;
   if(n_calls(1)==1) cout << "Define parameters" << endl;
   current_ph=current_phase(); n_calls(current_ph) += 1;
   i=1;                      // counters for keeping track of fixed (i) and estimated (ie) parameters, respectively
  //-------------compute expectations of state variables----------------//


  // apical fishing mortality rates for each fleet
    for ( k=1; k<=nfs; k++) {
      // apical fishing mortality rate during prehistoric period
      inow=i; f_hist_prior=0.; ihist=i;
      for ( j=1; j<=npf(nature(inow)); j++) {
        function_parameter(j)=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
        if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) f_hist_prior+=neg_log_lklhd(f_hist_parameter(j),best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
      }
      for ( y=1; y<=nyears_prehistoric; y++)f_apical(k,y)=function_value(nature(ihist),function_parameter,effort_obs(k,y));

     // add process errors to apical fishing mortality rates
      f_lklhd=0.;
      if(F1999==0){
        if(active(f_devs)) {  //changed 10/2010 PBC to allow random walk for 1st, 2nd modern eras (with a break in between)
          for (y=nyears_prehistoric+1; y<=(nyears_prehistoric+nyears_modern);y++) {
            if(f_dev_pdf==1) f_apical(k,y)=f_apical(k,y-1)*mfexp(f_devs(k,y)); else f_apical(k,y)=f_apical(k,y)+f_devs(k,y);
          }
          if(f_dev_pdf==1)f_apical(k,y)=f_apical(k,y-1)*exp(f_devs(k,y));
          //cout<<"y "<<y<<" f apical y-1 "<<f_apical(k,y-1)<<" f devs y "<<f_devs(k,y)<<" f apical y "<<f_apical(k,y)<<endl;
          for(y=(nyears_prehistoric+nyears_modern+2);y<=(nyears_past-1);y++){
            if(f_dev_pdf==1) f_apical(k,y)=f_apical(k,y-1)*mfexp(f_devs(k,y)); else f_apical(k,y)=f_apical(k,y)+f_devs(k,y);
          }
        }
      }
      else{
        if(active(f_devs)) {  //changed 10/2010 PBC to allow random walk for 1st, 2nd modern eras (with a break in between)
```

```
        for (y=nyears_prehistoric+1; y<=(nyears_prehistoric+nyears_modern-1);y++) {
          if(f_dev_pdf==1) f_apical(k,y)=f_apical(k,y-1)*mfexp(f_devs(k,y)); else f_apical(k,y)=f_apical(k,y)+f_devs(k,y);
        }
        f_apical(k,y)=f_apical(k,y-1)*exp(f_devs(k,y));
        f_apical(k,y+1)=f_apical(k,y)*exp(f_devs(k,y+1));
        for(y=(nyears_prehistoric+nyears_modern+2);y<=(nyears_past-1);y++){
          if(f_dev_pdf==1) f_apical(k,y)=f_apical(k,y-1)*mfexp(f_devs(k,y)); else f_apical(k,y)=f_apical(k,y)+f_devs(k,y);
        }
      }
    }
    f_apical(k,nyears_past)=(f_apical(k,nyears_past-1)+f_apical(k,nyears_past-2)+f_apical(k,nyears_past-3))*0.3333;

  } // end apical F loop
  //cout<<"f_apical "<<f_apical<<endl;
  //cout<<"f_devs "<<f_devs<<endl;

// expected natural mortality rate by age
  inow=i; m_prior=0.;
  for ( j=1; j<=npf(nature(inow)); j++) {
    function_parameter(j)=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
    if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) m_prior+=neg_log_lklhd(function_parameter(j),best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
  }
  if(mort_switch==0)for ( a=1; a<=nages; a++) m(a)=function_value(nature(inow),function_parameter,double(age(1)+a)-1);
  else m=M_age;

// expected relative recruitment
  inow=i; r_prior=0.; irn=i;
  for ( j=1; j<=npf(nature(inow)); j++) {
    recruitment_parameter(j)=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
    if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) r_prior+=neg_log_lklhd(recruitment_parameter(j),best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
    //cout<<"inow="<<inow<<" irn="<<irn<<" i="<<i<<" j="<<j<<" rec_par="<<recruitment_parameter(j)<<" par_est="<<par_est(i)<<" nature="<<nature(inow);
  }

// expected growth
  inow=i; w_prior=0.; iwn=i;
  for ( j=1; j<=npf(nature(inow)); j++) {
    growth_parameter(j)=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
    if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) w_prior+=neg_log_lklhd(growth_parameter(j),best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
  }
  for ( a=1; a<=nages-1; a++) {
    w(a)=function_value(nature(i-1),growth_parameter,double(age(1)+a)-1+0.5);
    if(fecundity_input(a)>=0) fecundity(a)=fecundity_input(a); else fecundity(a)=function_value(nature(i-1),growth_parameter,double(age(1)+a)-1+spawn_time);
  }
  if(m(nages)>0) plus_age=age(2)+mfexp(-m(nages))/(1-mfexp(-m(nages))); else plus_age=2*age(2);
  w(nages)=function_value(nature(iwn),growth_parameter,plus_age+0.5);
  if(fecundity_input(nages)>=0) fecundity(nages)=fecundity_input(nages); else fecundity(nages)=function_value(nature(i-1),growth_parameter,plus_age+spawn_time);
//cout << "weight at age " << w << endl;
//cout << "growth params " << growth_parameter << endl;


// virgin spawner-per recruit
  spr0=spr(maturity,fecundity,m,one_vector_age,zero,spawn_time,nages);

//LIZ modified 2 june 2004
// pup_survival=recruitment_parameter(1);
// alpha = pup_survival*spr0;
// recruitment_parameter(1)=alpha-1;

// expected q
  q_prior=0.;
  for (set=1; set<=nqs; set++) {
    inow=i;
    for ( j=1; j<=npf(nature(inow)); j++) {
      function_parameter(j)=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
      if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) q_prior(set)+=neg_log_lklhd(function_parameter(j),best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
    }
    for ( y=1; y<=n_eras; y++) {
      q_process(set,y)=function_value(nature(i-1),function_parameter,one);
    }
  }
// expected selectivity/vulnerability
  s_prior=0.;     //cout<<"nss "<<nss<<endl;
  for (set=1; set<=nss; set++) {
    inow=i;
    for ( j=1; j<=npf(nature(inow)); j++) {
      function_parameter(j)=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
      if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) s_prior(set)+=neg_log_lklhd(function_parameter(j),best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
    }
    for ( a=1; a<=nages; a++) s(set,a)=function_value(nature(i-1),function_parameter,double(age(1)+a-1));

  }
  //selectivity for apical f           Added by PBC 10/2010
  sel_f_apical=0.;
  for(j=1;j<=nyears_past;j++){
    for(k=1;k<=n_avg_f;k++){
      sel_f_apical(j)+=prop_effort(j,k)*s(ifs(k));
    }
    sel_f_apical(j)/=max(sel_f_apical(j)); //readjust to have maximum of 1
  }

// index observation variance
  i_d_prior=0.;
  for (set=1; set<=nids; set++) {
    i_d_var(set)=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
    if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) i_d_prior(set)+=neg_log_lklhd(i_d_var(set),best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
  }

// overall variance
  overall_var=get_function_parameters(i,i_in,iph(i),current_ph,par_est(i),pdf(i));
  if(best_guess(i-1)<0) i_in = -i_in; // special case for negative cv's
  if(pdf(i-1)>0 && iph(i-1)>0 && iph(i-1)<=current_ph) v_prior=neg_log_lklhd(overall_var,best_guess(i-1),one,one,zero,cv(i-1),zero,pdf(i-1),variance_scale,i_zero,i_in);
  //cout<<"i_d_var "<<i_d_var<<endl;

//-------------incorporate process errors----------------//
```

```
  // priors for apical fishing mortality rate process parameters
    if(active(f_rho)) f_prior+=neg_log_lklhd(f_rho,f_rho_best_guess,one,one,zero,f_rho_cv,zero,f_rho_pdf,variance_scale,i_zero,i_in);
    if(active(f_var)) f_prior+=neg_log_lklhd(f_var,f_var_best_guess,one,one,zero,f_var_cv,zero,f_var_pdf,variance_scale,i_zero,i_in);

  // priors for recruitment process parameters
    r_process_prior=0.;
    if(active(r_rho)) r_process_prior+=neg_log_lklhd(r_rho,r_rho_best_guess,one,one,zero,r_rho_cv,zero,r_rho_pdf,variance_scale,i_zero,i_in);
    if(active(r_var)) r_process_prior+=neg_log_lklhd(r_var,r_var_best_guess,one,one,zero,r_var_cv,zero,r_var_pdf,variance_scale,i_zero,i_in);

  // priors for q process parameters
    q_process_prior=0.;
    if(active(q_rho)) q_process_prior+=neg_log_lklhd(q_rho,q_rho_best_guess,one,one,zero,q_rho_cv,zero,q_rho_pdf,variance_scale,i_zero,i_in);
    if(active(q_var)) q_process_prior+=neg_log_lklhd(q_var,q_var_best_guess,one,one,zero,q_var_cv,zero,q_var_pdf,variance_scale,i_zero,i_in);

  // historical (1) and subsequent modern-era catchability coefficients
    q=q_process; q_lklhd=0.;
    if(active(q_devs)) {
      for (set=1; set<=nqs; set++) {
        for (y=2; y<=n_eras; y++) {
          if(q_dev_pdf==1) q(set,y)=q_process(set,y)*mfexp(q_devs(set,y)); else q(set,y)=q_process(set,y)+q_devs(set,y);
        }
      }
    }

 //----------------------------------------------------------------------------------
FUNCTION calculate_biomass
 //----------------------------------------------------------------------------------
   //cout<<"calc biomass"<<endl;

  if(n_calls(1)==1) cout << "Calculate biomass" << endl;
  index_pred=zero ; ssb=zero; r_process=one; r_lklhd=zero;

 //LIZ modified 2 june 2004
    pup_survival=recruitment_parameter(1); mc_pup_survival=recruitment_parameter(1);
    alpha = pup_survival*spr0;    steepness = alpha/(alpha+4.0) ;
    recruitment_parameter(1)=alpha-1;

  // calculate_fishing_mortality on all age classes (first two selectivity sets designated for historical and modern era fisheries)
  f=0.0;
  for (y=1; y<=nyears_past; y++) {
    for(k=1; k<=nfs; k++) {
    //  if(y<=nyears_prehistoric) set=1+2*(k-1); else set=2+2*(k-1); // first 2*nfs selection sets are for fisheries
    //  for (a=1; a<=nages; a++) f(a,y)+=f_apical(k,y)*s(set,a);
      for(a=1;a<=nages;a++)f(a,y)=f_apical(k,y)*sel_f_apical(y,a);
    }
  }

  // initial population structure assuming population at virgin levels (process errors assumed to average OUT)
  if(n_calls(1)==1) cout << "Calculating virgin abundance" << endl;
  n(1,1)=one;
  for (a=2; a<=nages; a++) {
    n(a,1)=n(a-1,1)*mfexp(-m(a-1));
    if(a==nages) n(a,1)=n(a,1)/(one-mfexp(-m(a)));
  }
   // time trajectory of population structure
  if(n_calls(1)==1) cout << "Calculating subsequent abundance" << endl;
  for (y=1; y<=nyears_past; y++)  {
    // distinguish historical period (no process errors) from modern epoch (has process errors)
    if(y<=nyears_prehistoric) t=1;
    else t=y-nyears_prehistoric+1;

    // update recruitment
    if(y>age(1)) r_process(y)=function_value(nature(irn),recruitment_parameter,ssb(y-age(1))); // x-year-olds in year x+1 were produced in year 1 (for which one can compute the ssb),
    if(active(r_devs) && t>1) {
      if(r_dev_pdf==1) r(y)=r_process(y)*mfexp(r_devs(t)); else r(y)=r_process(y)+r_devs(t);
    }
    else r(y)=r_process(y);
    n(1,y)=r(y);

    if(y>age(1))pup_survival_annual(y-age(1))=r(y)/(spr0*ssb(y-age(1))); //multiply by spr0 since ssb is really ssb/ssb0=ssb/spr0

    virgin_pred=0.0;
    //cout<<"plus age "<<plus_age<<endl;
    for (a=1; a<=nages; a++) {
      //cout<<"age "<<a<<endl;
      // average fecundity of plus-group during spawning season
      if(a==nages) {
        w(a)=function_value(nature(iwn),growth_parameter,plus_age+0.5);
        if(fecundity_input(a)>=0) fecundity(a)=fecundity_input(a); else fecundity(a)=function_value(nature(iwn),growth_parameter,plus_age+spawn_time);
      }
      wbyage(a,y)=w(a);
      //if(a==nages)cout<<"wt age "<<w<<endl;


      // relative spawning biomass
      ssb(y)+=maturity(a)*fecundity(a)*n(a,y)*mfexp(-(m(a)+f(a,y))*spawn_time)/spr0;

      // abundance at beginning of next year
      n(a+1,y+1)=n(a,y)*mfexp(-m(a)-f(a,y)); // t=1 in historical period, t=year in modern period
    } //age

      //define B1975, the biomass in 1975    *Paul changed Oct 2010
      if(y==16) {
       //cout << "y " << y << endl;
       B1975=0.0;Bvirgin=0;
       for (a=1; a<=nages; a++) {
         Bvirgin+=n(a,1)*w(a);
         B1975+=n(a,y)*wbyage(a,y);
       }
       B1975=B1975/Bvirgin; mc_B1975=B1975;
      }
      if(y==nyears_past) {
       //cout << "y " << y << endl;
```

```
      Bvirgin=0;B2009=0.0;
      for (a=1; a<=nages; a++) {
        Bvirgin+=n(a,1)*w(a);
        B2009+=n(a,y)*wbyage(a,nyears);
      }
      B2009=B2009/Bvirgin;
      mc_B2009=B2009;
     }

  // plus group age and abundance
  if((n(nages,y+1)+n(nages+1,y+1))>0)plus_age=(age(2)*n(nages,y+1)+(plus_age+1)*n(nages+1,y+1))/(n(nages,y+1)+n(nages+1,y+1));
  else plus_age=nages; //changed by PBC 10/2010 to prevent division by zero
  n(nages,y+1) += n(nages+1,y+1);
  //cout<<y<<" "<<plus_age<<endl;


} //year
//mc_N=n;

F2009=f_apical(1,nyears_past);

//compute recruitment, pup survival in next year (for back calculating numbers of age zeros in index in final year)
r_process(y)=function_value(nature(irn),recruitment_parameter,ssb(y-age(1))); // x-year-olds in year x+1 were produced in year 1 (for which one can compute the ssb),
if(active(r_devs) && t>1) {
  if(r_dev_pdf==1) r(y)=r_process(y)*mfexp(r_devs(t)); else r(y)=r_process(y)+r_devs(t);
}
else r(y)=r_process(y);

pup_survival_annual(y-1)=r(y)/(spr0*ssb(y-1)); //multiply by spr0 since ssb is really ssb/ssb0=ssb/spr0
n(1,y)=r(y);

for(y=1;y<=nyears_past;y++){
  //age zeroes   ADDED PBC 11/3/2010; note, we're assuming age zeroes sampled half way through the year, no age zeroes included in depletion calculations
  for(series=1;series<=n_index_series;series++){
        if(index_units(series)==1)         index_pred(y,series) +=      q(iqs(series),t)*sel_age0(series)*n(1,y+1)/sqrt(pup_survival_annual(y));
        else if(index_units(series)==2)    index_pred(y,series) += w(1)*q(iqs(series),t)*sel_age0(series)*n(1,y+1)/sqrt(pup_survival_annual(y));
  }
  for(a=1;a<=nages;a++){
    // predicted indices
    for (series=1; series<=n_index_series; series++) {
      if(index_pdf(series)>0) {
        //cout<<"iss "<<iss(series)<<endl;
        //cout<<"q "<<q(iqs(series),t)<<endl;
        //cout<<"s "<<s(iss(series),a)<<endl;
        //cout<<"time "<<index_time(series);
        if(index_units(series)==1)         index_pred(y,series) +=      q(iqs(series),t)*s(iss(series),a)*n(a,y)*mfexp(-(m(a)+f(a,y))*index_time(series));
        else if(index_units(series)==2)    index_pred(y,series) += w(a)*q(iqs(series),t)*s(iss(series),a)*n(a,y)*mfexp(-(m(a)+f(a,y))*index_time(series));
        else if(index_units(series)==10) { index_pred(y,series) +=      q(iqs(series),t)*s(iss(series),a)*n(a,y)*mfexp(-(m(a)+f(a,y))*index_time(series));
                                           virgin_pred(series)  +=                       s(iss(series),a)*n(a,1)*mfexp(-(m(a))*index_time(series)); }
        else if(index_units(series)==20) { index_pred(y,series) += w(a)*q(iqs(series),t)*s(iss(series),a)*n(a,y)*mfexp(-(m(a)+f(a,y))*index_time(series));
                                           virgin_pred(series)  += w(a)*                 s(iss(series),a)*n(a,1)*mfexp(-(m(a))*index_time(series)); }
      }
    }
  }
  // scale indices
  for (series=1; series<=n_index_series; series++)
    if(index_pdf(series)>0 && index_units(series)>9) index_pred(y,series) /= virgin_pred(series);
}

I1975=index_pred(16,6);
I2009=index_pred(nyears_past,6);

// Projections and equilibrium statistics based on overall selectivity during last year
if (sd_phase) {
  if(n_calls(1)==1) cout << "starting projections" << endl;
  for (y=1; y<=nyears_past; y++) {
     for (a=1; a<=nages; a++) s_latest(a)=f(a,y);
     log_F_apex(y)=max(s_latest);
     if(log_F_apex(y)>0) log_F_apex(y)=log(log_F_apex(y)); else log_F_apex(y)=-999;
  }
  Fcurrent=max(s_latest); Bcurrent=ssb(nyears_past); if(Fcurrent>0) s_latest=s_latest/Fcurrent;
  F2009=Fcurrent;
  alpha=recruitment_parameter(1)+1;  Trecover=-1;  //nat_mort=m(1);
  if(reference_selectivity==1) s_equilibrium=s_latest;
  else s_equilibrium=maturity;

  if (last_phase()) {
    //cout<<"made it to last phase"<<endl;
   // Compute equilibrium statistics
    if(n_calls(1)==1) cout << "Calculating equilibrium statistics" << endl;

    F_spr.fill_seqadd(0,.01);       //fill in Fs for per-recruit stuff added PBC 11/2010
    F01=goldensection(3, Fspr30, w, m, s_equilibrium, nages, maturity, fecundity, spawn_time, spr0, nature(irn),recruitment_parameter );
    //cout<<"F01 "<<F01<<endl;
    Fmax=goldensection(i_one, Fspr20, w, m, s_equilibrium, nages, maturity, fecundity, spawn_time, spr0, nature(irn),recruitment_parameter );
    //cout<<"Fmax "<<Fmax<<endl;
    Fmsy=goldensection(i_two, Fspr40, w, m, s_equilibrium, nages, maturity, fecundity, spawn_time, spr0, nature(irn),recruitment_parameter );
    //cout<<"Fmsy "<<Fmsy<<endl;
    Fmat =goldensection(i_two, Fspr40, w, m, maturity, nages, maturity, fecundity, spawn_time, spr0, nature(irn),recruitment_parameter );
    sprmat=spr(maturity,fecundity,m,maturity,Fmat,spawn_time,nages)/spr0;
    spr01=spr(maturity,fecundity,m,s_equilibrium,F01,spawn_time,nages)/spr0;
    sprmax=spr(maturity,fecundity,m,s_equilibrium,Fmax,spawn_time,nages)/spr0;
    sprmsy=spr(maturity,fecundity,m,s_equilibrium,Fmsy,spawn_time,nages)/spr0;
    spr20=spr(maturity,fecundity,m,s_equilibrium,Fspr20,spawn_time,nages)/spr0;
    spr30=spr(maturity,fecundity,m,s_equilibrium,Fspr30,spawn_time,nages)/spr0;
    spr40=spr(maturity,fecundity,m,s_equilibrium,Fspr40,spawn_time,nages)/spr0;
    spr50=spr(maturity,fecundity,m,s_equilibrium,Fspr50,spawn_time,nages)/spr0;
    spr60=spr(maturity,fecundity,m,s_equilibrium,Fspr60,spawn_time,nages)/spr0;
    for(y=1;y<=n_iter_pr;y++){
      spr_vec(y)=spr(maturity,fecundity,m,s_equilibrium,F_spr[y],spawn_time,nages)/spr0;
      ypr_vec(y)=ypr(w,m,s_equilibrium,F_spr[y],nages);
    }
    yprmat=ypr(w,m,maturity,Fmat,nages);
    ypr01=ypr(w,m,s_equilibrium,F01,nages);
```

```
 yprmax=ypr(w,m,s_equilibrium,Fmax,nages);
 yprmsy=ypr(w,m,s_equilibrium,Fmsy,nages);
 ypr20=ypr(w,m,s_equilibrium,Fspr20,nages);
 ypr30=ypr(w,m,s_equilibrium,Fspr30,nages);
 ypr40=ypr(w,m,s_equilibrium,Fspr40,nages);
 ypr50=ypr(w,m,s_equilibrium,Fspr50,nages);
 ypr60=ypr(w,m,s_equilibrium,Fspr60,nages);
 Bmat  =equilibrium_ssb(nature(irn),recruitment_parameter,sprmat);    Rmat=Bmat/sprmat;
 Bspr20=equilibrium_ssb(nature(irn),recruitment_parameter,spr20*spr0);    Rspr20=Bspr20/spr20;
 Bspr30=equilibrium_ssb(nature(irn),recruitment_parameter,spr30);    Rspr30=Bspr30/spr30;
 Bspr40=equilibrium_ssb(nature(irn),recruitment_parameter,spr40);    Rspr40=Bspr40/spr40;
 Bspr50=equilibrium_ssb(nature(irn),recruitment_parameter,spr50);    Rspr50=Bspr50/spr50;
 Bspr60=equilibrium_ssb(nature(irn),recruitment_parameter,spr60);    Rspr60=Bspr60/spr60;
 B01   =equilibrium_ssb(nature(irn),recruitment_parameter,spr01);    R01   =B01   /spr01;
 Bmax  =equilibrium_ssb(nature(irn),recruitment_parameter,sprmax);   Rmax  =Bmax  /sprmax;
 Bmsy  =equilibrium_ssb(nature(irn),recruitment_parameter,sprmsy);   Rmsy  =Bmsy  /sprmsy;
 Bmsst=Bmsy*(1-M_age_indep);
 if(Bspr20 >0) BoverBspr20 =Bcurrent/Bspr20 ; else {Bspr20=-9.0; Rspr20=-9.0; ypr20=-9.0; BoverBspr20 =-9.0;}
 if(Bspr30 >0) BoverBspr30 =Bcurrent/Bspr30 ; else {Bspr30=-9.0; Rspr30=-9.0; ypr30=-9.0; BoverBspr30 =-9.0;}
 if(Bspr40 >0) BoverBspr40 =Bcurrent/Bspr40 ; else {Bspr40=-9.0; Rspr40=-9.0; ypr40=-9.0; BoverBspr40 =-9.0;}
 if(Bspr50 >0) BoverBspr50 =Bcurrent/Bspr50 ; else {Bspr50=-9.0; Rspr50=-9.0; ypr50=-9.0; BoverBspr50 =-9.0;}
 if(Bspr60 >0) BoverBspr60 =Bcurrent/Bspr60 ; else {Bspr60=-9.0; Rspr60=-9.0; ypr60=-9.0; BoverBspr60 =-9.0;}
 if(B01    >0) BoverB01    =Bcurrent/B01    ; else {B01 = -9.0; R01 = -9.0; ypr01=-9.0; BoverB01    =-9.0;}
 if(Bmax   >0) BoverBmax   =Bcurrent/Bmax   ; else {Bmax=-9.0; Rmax=-9.0; yprmax=-9.0; BoverBmax   =-9.0;}
 if(Bmsy   >0) BBmsy   =Bcurrent/Bmsy   ; else {Bmsy=-9.0; Rmsy=-9.0; yprmsy=-9.0; BBmsy   =-9.0;}
 if(Bmat   >0) BoverBmat   =Bcurrent/Bmat   ; else {Bmat=-9.0; Rmat=-9.0; yprmat=-9.0; BoverBmat   =-9.0;}
 if(Bmsy   >0) BBmsst  =Bcurrent/(Bmsy*(1-M_age_indep));
 if(Fspr20 >0) FoverFspr20 =Fcurrent/Fspr20 ; else FoverFspr20 =-9.0;
 if(Fspr30 >0) FoverFspr30 =Fcurrent/Fspr30 ; else FoverFspr30 =-9.0;
 if(Fspr40 >0) FoverFspr40 =Fcurrent/Fspr40 ; else FoverFspr40 =-9.0;
 if(Fspr50 >0) FoverFspr50 =Fcurrent/Fspr50 ; else FoverFspr50 =-9.0;
 if(Fspr60 >0) FoverFspr60 =Fcurrent/Fspr60 ; else FoverFspr60 =-9.0;
 if(F01    >0) FoverF01    =Fcurrent/F01    ; else FoverF01    =-9.0;
 if(Fmax   >0) FoverFmax   =Fcurrent/Fmax   ; else FoverFmax   =-9.0;
 if(Fmsy   >0) FFmsy   =Fcurrent/Fmsy   ; else FFmsy   =-9.0;
 if(Fmat   >0) FoverFmat   =Fcurrent/Fmat   ; else FoverFmat   =-9.0;

// Compute projections
 if(n_calls(1)==1 && nyears_proj>0) cout << "Making projections" << endl;
 for (y=nyears_past+1; y<=nyears; y++)  {
   t=y-nyears_past;
   r(y)=function_value(nature(irn),recruitment_parameter,ssb(y-age(1))); // x-year-olds in year x+1 were produced in year 1 (for which one can compute the ssb),
   if(active(r_devs_proj)) { if(r_dev_pdf==1) r(y)=r(y)*mfexp(r_devs_proj(t)); else r(y)=r(y)+r_devs_proj(t); }
   n(1,y)=r(y);
   for (a=1; a<=nages; a++) {
     // average fecundity of plus-group during spawning season
     if(a==nages) {
       w(a)=function_value(nature(iwn),growth_parameter,plus_age+0.5);
       if(fecundity_input(a)>=0) fecundity(a)=fecundity_input(a); else fecundity(a)=function_value(nature(iwn),growth_parameter,plus_age+spawn_time);
     }
     wbyage(a,y)=w(a);
     if(in_prj(t,1) >= 0)        F_proj(t)=in_prj(t,1);   // note: this approach assumes there is no implementation uncertainty
     else if(in_prj(t,1) > -0.2) F_proj(t)=F01;           //       I had a hard time getting runs with long projections to converge
     else if(in_prj(t,1) > -1)   F_proj(t)=Fmat;           //       when I treated F_proj as a random variable, even with low implementation uncertainty
     else if(in_prj(t,1) > -2)   F_proj(t)=Fmsy;
     else if(in_prj(t,1) > -3)   F_proj(t)=Fmax;
     else if(in_prj(t,1) > -21)  F_proj(t)=Fspr20;
     else if(in_prj(t,1) > -31)  F_proj(t)=Fspr30;
     else if(in_prj(t,1) > -41)  F_proj(t)=Fspr40;
     else if(in_prj(t,1) > -51)  F_proj(t)=Fspr50;
     else if(in_prj(t,1) > -61)  F_proj(t)=Fspr60;
     else                        F_proj(t)=Fcurrent;
     if(F_proj(t)>0) log_F_apex(y)=log(F_proj(t)); else log_F_apex(y)=-999;
     f(a,y)=F_proj(t)*s_latest(a);
     ssb(y)+=maturity(a)*fecundity(a)*n(a,y)*mfexp(-(m(a)+f(a,y))*spawn_time)/spr0;
     n(a+1,y+1)=n(a,y)*mfexp(-m(a)-f(a,y));
   } //age
   plus_age=(age(2)*n(nages,y+1)+(plus_age+1)*n(nages+1,y+1))/(n(nages,y+1)+n(nages+1,y+1));
   n(nages,y+1) += n(nages+1,y+1);
 } //year
 B=ssb; BoverBref=-9.0;
 if(Bref > 0)                          BoverBref = B/Bref ;
 else if(Bref > -0.2 && B01    > 0)  BoverBref = B/B01    ;
 else if(Bref > -1   && Bmat   > 0)  BoverBref = B/Bmat   ;
 else if(Bref > -2   && Bmsy   > 0)  BoverBref = B/Bmsy   ;
 else if(Bref > -3   && Bmax   > 0)  BoverBref = B/Bmax   ;
 else if(Bref > -21  && Bspr20 > 0)  BoverBref = B/Bspr20 ;
 else if(Bref > -31  && Bspr30 > 0)  BoverBref = B/Bspr30 ;
 else if(Bref > -41  && Bspr40 > 0)  BoverBref = B/Bspr40 ;
 else if(Bref > -51  && Bspr50 > 0)  BoverBref = B/Bspr50 ;
 else if(Bref > -61  && Bspr60 > 0)  BoverBref = B/Bspr60 ;
 else                                BoverBref = B/Bcurrent ;
 if(Bspr30 >0) BoverBspr30 =Bcurrent/Bspr30 ; else BoverBspr30 =-9.0;
 if(Bspr40 >0) BoverBspr40 =Bcurrent/Bspr40 ; else BoverBspr40 =-9.0;
 if(Bspr50 >0) BoverBspr50 =Bcurrent/Bspr50 ; else BoverBspr50 =-9.0;
 if(Bspr60 >0) BoverBspr60 =Bcurrent/Bspr60 ; else BoverBspr60 =-9.0;
 if(B01    >0) BoverB01    =Bcurrent/B01    ; else BoverB01    =-9.0;
 if(Bmax   >0) BoverBmax   =Bcurrent/Bmax   ; else BoverBmax   =-9.0;
 if(Bmsy   >0) BBmsy   =Bcurrent/Bmsy   ; else BBmsy   =-9.0;
 for(y=nyears_past; y<=nyears; y++) if(BoverBref(y)>=1.0) {Trecover=y+year(1)-1; break;}
 Bpro_5=BoverBref(nyears_past-5);
 Bpro_4=BoverBref(nyears_past-4);
 Bpro_3=BoverBref(nyears_past-3);
 Bpro_2=BoverBref(nyears_past-2);
 Bpro_1=BoverBref(nyears_past-1);
 Bpro0=BoverBref(nyears_past);
 if(nyears_proj<1) Bpro1=-1; else Bpro1=BoverBref(nyears_past+1);
 if(nyears_proj<2) Bpro2=-1; else Bpro2=BoverBref(nyears_past+2);
 if(nyears_proj<3) Bpro3=-1; else Bpro3=BoverBref(nyears_past+3);
 if(nyears_proj<4) Bpro4=-1; else Bpro4=BoverBref(nyears_past+4);
 if(nyears_proj<5) Bpro5=-1; else Bpro5=BoverBref(nyears_past+5);
 if(nyears_proj<6) Bpro6=-1; else Bpro6=BoverBref(nyears_past+6);
 if(nyears_proj<7) Bpro7=-1; else Bpro7=BoverBref(nyears_past+7);
 if(nyears_proj<8) Bpro8=-1; else Bpro8=BoverBref(nyears_past+8);
```

```
      if(nyears_proj<9) Bpro9=-1; else Bpro9=BoverBref(nyears_past+9);
      if(nyears_proj<10) Bpro10=-1; else Bpro10=BoverBref(nyears_past+10);
      if(nyears_proj<11) Bpro11=-1; else Bpro11=BoverBref(nyears_past+11);
      if(nyears_proj<12) Bpro12=-1; else Bpro12=BoverBref(nyears_past+12);
      if(nyears_proj<13) Bpro13=-1; else Bpro13=BoverBref(nyears_past+13);
      if(nyears_proj<14) Bpro14=-1; else Bpro14=BoverBref(nyears_past+14);
      if(nyears_proj<15) Bpro15=-1; else Bpro15=BoverBref(nyears_past+15);
      if(nyears_proj<16) Bpro16=-1; else Bpro16=BoverBref(nyears_past+16);
      if(nyears_proj<17) Bpro17=-1; else Bpro17=BoverBref(nyears_past+17);
      if(nyears_proj<18) Bpro18=-1; else Bpro18=BoverBref(nyears_past+18);
      if(nyears_proj<19) Bpro19=-1; else Bpro19=BoverBref(nyears_past+19);
      if(nyears_proj<20) Bpro20=-1; else Bpro20=BoverBref(nyears_past+20);
      if(nyears_proj<21) Bpro21=-1; else Bpro21=BoverBref(nyears_past+21);
      if(nyears_proj<22) Bpro22=-1; else Bpro22=BoverBref(nyears_past+22);
      if(nyears_proj<23) Bpro23=-1; else Bpro23=BoverBref(nyears_past+23);
      if(nyears_proj<24) Bpro24=-1; else Bpro24=BoverBref(nyears_past+24);
      if(nyears_proj<25) Bpro25=-1; else Bpro25=BoverBref(nyears_past+25);
      if(nyears_proj<26) Bpro26=-1; else Bpro26=BoverBref(nyears_past+26);
      if(nyears_proj<27) Bpro27=-1; else Bpro27=BoverBref(nyears_past+27);
      if(nyears_proj<28) Bpro28=-1; else Bpro28=BoverBref(nyears_past+28);
      if(nyears_proj<29) Bpro29=-1; else Bpro29=BoverBref(nyears_past+29);
      if(nyears_proj<30) Bpro30=-1; else Bpro30=BoverBref(nyears_past+30);

    }// last_phase loop
  }// sd_phase loop

 //-----------------------------------------------------------------------------------
FUNCTION calculate_the_objective_function
 //-----------------------------------------------------------------------------------
  double penalty_wt;

  //cout<<"calc obj func"<<endl;

  if(n_calls(1)==1) cout << "Calculating objective function" << endl;
  index_lklhd=0.; obj_func=0.; penalty=0; equilibrium_penalty=0; projection_penalty=0; penalty_wt=0.001;

 // ---------------observation errors------------------
  if(overall_var<0)var1=log(1-overall_var);
  else var1=overall_var;
  for(y=1; y<=nyears_past; y++) {
    for(series=1; series<=n_index_series; series++) {
       if(index_pdf(series)>0  && index_obs(y,series)>=0)
         index_lklhd(series)+=neg_log_lklhd(index_obs(y,series),index_pred(y,series),one,one,zero,i_d_var(ivs(series))+var1+index_var(y,series),one,index_pdf(series),variance_scale,variance_modify,y);
    }
  }
  if(n_index_series>0) obj_func+=sum(index_lklhd);

 // ---------------process errors------------------
  if(active(r_devs)) {
    if(variance_scale==1 && r_dev_pdf==1 && r_var<zero) var=log(1.0+square(r_var));
    else if(variance_scale==1 && r_dev_pdf==1 && r_var>zero) var=r_var;
    else if(variance_scale==2 && r_dev_pdf==2 && r_var>zero) var=r_var;
    else var=get_variance(one,r_var,zero,r_dev_pdf,variance_scale,i_zero);
    r_lklhd=square(r_devs(2));
    for(t=3; t<=n_eras; t++) r_lklhd += square(r_devs(t)-r_rho*r_devs(t-1));
    r_lklhd=0.5*(r_lklhd/var+double(n_eras-1)*log(var));
    obj_func += r_lklhd;
  }

  if(active(f_devs)) {
    for(k=1; k<=nfs; k++) {
      if(variance_scale==1 && f_dev_pdf==1 && f_var<zero) var=log(1.0+square(f_var));
      else if(variance_scale==1 && f_dev_pdf==1 && f_var>zero) var=f_var;
      else if(variance_scale==2 && f_dev_pdf==2 && f_var>zero) var=f_var;
      else var=get_variance(f_apical(k,nyears_prehistoric+1),f_var,zero,f_dev_pdf,variance_scale,i_zero);
      f_lklhd=square(f_devs(k,nyears_prehistoric+1));
      //for(t=nyears_prehistoric+2; t<=nyears_b4_change; t++) f_lklhd += square(f_devs(k,t)-f_rho*f_devs(k,t-1));
      //f_lklhd=0.5*(f_lklhd/var+double(nyears_b4_change-nyears_prehistoric)*log(var));
      if(F1999==0){
        for(t=nyears_prehistoric+2; t<=(nyears_prehistoric+nyears_modern); t++) f_lklhd += square(f_devs(k,t)-f_rho*f_devs(k,t-1));
      }
      else{
        for(t=nyears_prehistoric+2; t<=(nyears_prehistoric+nyears_modern-1); t++) f_lklhd += square(f_devs(k,t)-f_rho*f_devs(k,t-1));
      }
      for(t=(nyears_prehistoric+nyears_modern+2);t<nyears_past;t++) f_lklhd += square(f_devs(k,t)-f_rho*f_devs(k,t-1));
      if(F1999==0)f_lklhd=0.5*(f_lklhd/var+double(nyears_past-nyears_prehistoric-1)*log(var));
      else f_lklhd=0.5*(f_lklhd/var+double(nyears_past-nyears_prehistoric-2)*log(var));
      obj_func += f_lklhd;
    }
  }

  if(active(q_devs)) {
    for (set=1; set<=nqs; set++) {
      if(variance_scale==1 && q_dev_pdf==1 && overall_var<zero) var=log(1.0+square(q_var*overall_var));
      else if(variance_scale==1 && q_dev_pdf==1 && overall_var>zero) var=q_var*overall_var;
      else if(variance_scale==2 && q_dev_pdf==2 && overall_var>zero) var=q_var*overall_var;
      else var=get_variance(q(nyears_prehistoric+1,set),q_var*overall_var,zero,q_dev_pdf,variance_scale,i_zero);
      q_lklhd(set)=square(q_devs(2,set));
      for(t=3; t<=n_eras; t++) q_lklhd(set) += square(q_devs(t,set)-q_rho*q_devs(t-1,set));
      q_lklhd(set)=0.5*(q_lklhd(set)/var+(n_eras-1)*log(var));
    }
    obj_func += sum(q_lklhd);
  }

 // ---------------Bayesian priors---------------------
  obj_func += m_prior+r_prior+f_prior+f_hist_prior+w_prior+v_prior+q_process_prior+r_process_prior+sum(q_prior)+sum(s_prior)+sum(i_d_prior);

 // --------------other penalties----------------------
  //if(steepness<0.21)penalty+=square(steepness-0.21)*10000.0;  //PBC added 10/19/2010
  if(F1999==1 && f_devs(1,nyears_modern)>0)penalty+=square(f_devs(1,nyears_prehistoric+nyears_modern))*1000.0;  //penalty for 1999 F deviation > 0 PBC 11/3/2010
  if(f_devs(1,nyears_modern+1)>0)penalty+=square(f_devs(1,nyears_prehistoric+nyears_modern+1))*1000.0;          //penalty for 2000 F deviation > 0 PBC 11/3/2010
  for (y=1; y<=nyears_past; y++) if(f_apical(1,y)>1.0)penalty+=square(f_apical(1,y)-1.0);
  for (y=1; y<=nyears_past; y++) if(r(y)<0) penalty += square(r(y))*1000.0;
  for (y=1; y<=n_eras; y++) for (set=1; set<=nqs; set++) if(q(set,y)<0) penalty += square(q(set,y))*1000.0;
```

```
  for (a=1; a<=nages; a++) {
    if(m(a)<0) penalty += square(m(a))*1000.0;
    if(w(a)<0) penalty += square(w(a))*1000.0;
    for (set=1; set<=nss; set++) if(s(set,a)<0) penalty += square(s(set,a))*1000.0;
  }
  if(current_ph<(last_iph-1)) {
    pred= Fcurrent ;
    if(pred<0.1) penalty+=neg_log_lklhd(0.1,pred,one,one,zero,overall_var,zero,variance_scale,variance_scale,i_zero,y);
    if(pred>one) penalty+=neg_log_lklhd(one,pred,one,one,zero,overall_var,zero,variance_scale,variance_scale,i_zero,y);
  }
  else if(last_phase()) {
    //equilibrium_penalty+=neg_log_lklhd(0.2,spr20,one,one,zero,10*overall_var,zero,variance_scale,variance_scale,i_zero,y);
    equilibrium_penalty+=square(0.2-spr20)/penalty_wt;
    equilibrium_penalty+=square(0.3-spr30)/penalty_wt;
    equilibrium_penalty+=square(0.4-spr40)/penalty_wt;
    equilibrium_penalty+=square(0.5-spr50)/penalty_wt;
    equilibrium_penalty+=square(0.6-spr60)/penalty_wt;
    if(active(r_devs_proj)) {
      if(variance_scale==1 && r_dev_pdf==1 && r_dev_proj_cv<zero) var=log(1.0+square(r_dev_proj_cv));
      else if(variance_scale==1 && r_dev_pdf==1 && r_dev_proj_cv>zero) var=r_dev_proj_cv;
      else if(variance_scale==2 && r_dev_pdf==2 && r_dev_proj_cv>zero) var=r_dev_proj_cv;
      else var=get_variance(one,r_dev_proj_cv,zero,r_dev_pdf,variance_scale,i_zero);
      projection_penalty=square(r_devs_proj(1));
      for(t=2; t<=nyears_proj; t++) projection_penalty += square(r_devs_proj(t)-r_rho*r_devs_proj(t-1));
      projection_penalty=0.5*(projection_penalty/var+double(nyears_proj)*log(var));
    }
  }
  obj_func+=(penalty+equilibrium_penalty+projection_penalty);
   // cout<<"calc obj func done"<<endl;

 //---------------------------------------------------------------------------------
FUNCTION outputMCMC
 //---------------------------------------------------------------------------------
  ofstream MCMCout("MCMC.out",ios::app);
  MCMCout << mc_pup_survival << " " << mc_B1975 << " " << mc_B2009 << " " <<steepness<<" ";
  MCMCout <<BBmsy<<" "<<FFmsy<<" "<<Bmsy<<" "<<Fmsy<<" "<<BBmsst<<endl;
  //MCMCout<<mc_pup_survival;
  //MCMCout<<mc_B1975;

  //<< Bpro_5 << " " << Bpro_4 << " "<< Bpro_3 << " "<< Bpro_2 << " " << Bpro_1 << " " ;
  //MCMCout << Bpro0 << " " << Bpro1 << " "<< Bpro2 << " "<< Bpro3 << " " << Bpro4 << " " << Bpro5 << " ";
  //MCMCout << Bpro6 << " "<< Bpro7 << " "<< Bpro8 << " " << Bpro9 << " " << Bpro10 << " ";
  //MCMCout << Bpro11 << " "<< Bpro12 << " "<< Bpro13 << " " << Bpro14 << " " << Bpro15 << " ";
  //MCMCout << Bpro16 << " "<< Bpro17 << " "<< Bpro18 << " " << Bpro19 << " " << Bpro20 << " ";
  //MCMCout << Bpro21 << " "<< Bpro22 << " "<< Bpro23 << " " << Bpro24 << " " << Bpro25 << " ";
  //MCMCout << Bpro26 << " "<< Bpro27 << " "<< Bpro28 << " " << Bpro29 << " " << Bpro30 << endl;
  MCMCout.close();

//need:
//Steepness
//F/effort parameters
//Abundance by age
//Apical F by year
//Apical B by year
//Predicted indices by year
//Fmsy
//B by age


 /////////////////////////////////////////////////////////////////////////////////
REPORT_SECTION   // uses regular C++ code
 /////////////////////////////////////////////////////////////////////////////////
  n_par_phase=initial_params::nvarcalc(); // number of active parameters
  double aic=2.0*(value(obj_func-equilibrium_penalty-projection_penalty)+double(n_par_phase));
  cout << "Writing report" << endl;

  adstring label;
  if(Bref > 0)                          label = "input value   ";
  else if(Bref > -0.2 && B01    > 0)  label = "B at F0.1     ";
  else if(Bref > -1   && Bmat   > 0)  label = "B at MSYadult ";
  else if(Bref > -2   && Bmsy   > 0)  label = "B at MSYfleet ";
  else if(Bref > -3   && Bmax   > 0)  label = "B at Fmax     ";
  else if(Bref > -21  && Bspr20 > 0)  label = "B at 20% spr  ";
  else if(Bref > -31  && Bspr30 > 0)  label = "B at 30% spr  ";
  else if(Bref > -41  && Bspr40 > 0)  label = "B at 40% spr  ";
  else if(Bref > -51  && Bspr50 > 0)  label = "B at 50% spr  ";
  else if(Bref > -61  && Bspr60 > 0)  label = "B at 60% spr  ";
  else                                label = "current level ";

  report.setf(ios::right, ios::adjustfield);
  report.setf(ios::scientific, ios::floatfield);
  report << "--------------------------------------------------------------------" << endl;
  report << "LIKELIHOOD RESULTS" << endl;
  report << "--------------------------------------------------------------------" << endl;
  report << "AIC                 : " << setw(12) << setprecision(5) << aic << endl;

  if(n_data<(n_par_phase+2)) {
    report << "AICc (small sample) : " << " undefined (too few data)" << "  " << n_data << "  " << (n_par_phase+2)<< endl;
  }
  else {
    double aicc=aic+2.0*double(n_par_phase*(n_par_phase+1)/(n_data-n_par_phase-1));
    report << "AICc (small sample) : " << setw(12) << setprecision(5) << aicc << endl;
  }
  report << "               " << endl;
  report << "OBJECTIVE FUNCTION  : " << setw(12) << setprecision(5) << obj_func << endl;
  report << " Observation errors : " << endl;
  report << "   Abundance indices: " ;
    for(series=1; series<=n_index_series-1; series++) report << setw(12) << setprecision(5) << index_lklhd(series) << " ";
    report << setw(12) << setprecision(5) << index_lklhd(n_index_series) << endl ;
  report << " Process errors     : " << endl;
  report << "   f fishing mort.  : " << setw(12) << setprecision(5) << f_lklhd  << endl;
  report << "   r recruitment    : " << setw(12) << setprecision(5) << r_lklhd  << endl;
  report << "   q catchability   : " ;
    for(set=1; set<=nqs-1; set++) report << setw(12) << setprecision(5) << q_lklhd(set) << " ";
```

```
  report << setw(12) << setprecision(5) << q_lklhd(nqs) << endl ;
report << " Priors             : " << endl;
report << "   F historical     : " << setw(12) << setprecision(5) << f_hist_prior  << endl;
report << "   F modern period  : " << setw(12) << setprecision(5) << f_prior  << endl;
report << "   m natural mort.  : " << setw(12) << setprecision(5) << m_prior  << endl;
report << "   r recruitment    : " << setw(12) << setprecision(5) << r_prior  << endl;
report << "   r process error  : " << setw(12) << setprecision(5) << r_process_prior  << endl;
report << "   k growth         : " << setw(12) << setprecision(5) << w_prior  << endl;
report << "   q catchability   : " ;
  for(set=1; set<=nqs-1; set++) report << setw(12) << setprecision(5) << q_prior(set) << " ";
  report << setw(12) << setprecision(5) << q_prior(nqs) << endl ;
report << "   q process error  : " << setw(12) << setprecision(5) << q_process_prior << endl;
report << "   s selectivity    : " ;
  for(set=1; set<=nss-1; set++) report << setw(12) << setprecision(5) << s_prior(set) << " ";
  report << setw(12) << setprecision(5) << s_prior(nss) << endl ;
report << "   index variances  : ";
  for(set=1; set<=nids-1; set++) report << setw(12) << setprecision(5) << i_d_prior(set) << " ";
  report << setw(12) << setprecision(5) << i_d_prior(nids) << endl ;
report << "   over-all var.    : " << setw(12) << setprecision(5) << v_prior << endl;
report << "  Penalties         : " << endl;
report << "   out-of-bounds    : " << setw(12) << setprecision(5) << penalty << endl;
report << "   equilibrium stats: " << setw(12) << setprecision(5) << equilibrium_penalty << endl;
report << "   projections      : " << setw(12) << setprecision(5) << projection_penalty << endl;
report << "               " << endl;
if(overall_var<zero) report << "OVERALL %CV         : " << setw(12) << setprecision(5) << -100.0*overall_var << endl;
else                 report << "OVERALL VARIANCE    : " << setw(12) << setprecision(5) << overall_var << endl;
report << "               " << endl; report << "               " << endl;
  steepness = alpha/(alpha+4.0) ;
report << "LIFE-TIME REPRODUCTIVE RATE (alpha): " << setw(12) << setprecision(5) << alpha << "   " << "steepness  " << steepness << endl;
report << "PUP SURVIVAL: " << setw(12) << setprecision(5) << pup_survival << "   " << "B1975  " << B1975 << endl;
report << "VIRGIN SPAWNERS PER RECRUIT (spr0): " << setw(12) << setprecision(5) << spr0 << endl;
report << "NATURAL MORTALITY RATE: " << setw(12) << setprecision(5) << nat_mort << endl;
report << "YEAR OF RECOVERY: " << setw(5) << setprecision(0) << Trecover << endl;
report << "               " << endl; report << "               " << endl;
report << "NUMBER OF FUNCTION EVALUATIONS (THIS PHASE): " << setw(12) << setprecision(5) << n_calls(current_ph) << endl;
report << "NUMBER OF FUNCTION EVALUATIONS (CUMULATIVE): " << setw(12) << setprecision(5) << sum(n_calls) << endl;
report << "               " << endl; report << "               " << endl;

report << "---------------------------------------------------------------------" << endl;
report << "MANAGEMENT BENCHMARKS" << endl;
report << "Type          F           Y/R        SSB        SPR        R" << endl;
report << "---------------------------------------------------------------------" << endl;
report.setf(ios::scientific, ios::floatfield);
report << "VIRGIN   " << setw(13) << setprecision(4) << zero   << " " << zero   << " " << one    << " " << one    << " " << one    << endl;
report << "MSY adult" << setw(13) << setprecision(4) << Fmat   << " " << yprmat << " " << Bmat   << " " << sprmat << " " << Rmat   << endl;
report << "MSY fleet" << setw(13) << setprecision(4) << Fmsy   << " " << yprmsy << " " << Bmsy   << " " << sprmsy << " " << Rmsy   << endl;
report << "MAX Y/R  " << setw(13) << setprecision(4) << Fmax   << " " << yprmax << " " << Bmax   << " " << sprmax << " " << Rmax   << endl;
report << "F0.1     " << setw(13) << setprecision(4) << F01    << " " << ypr01  << " " << B01    << " " << spr01  << " " << R01    << endl;
report << "20% SPR  " << setw(13) << setprecision(4) << Fspr20 << " " << ypr20  << " " << Bspr20  << " " << spr20  << " " << Rspr20 << endl;
report << "30% SPR  " << setw(13) << setprecision(4) << Fspr30 << " " << ypr30  << " " << Bspr30  << " " << spr30  << " " << Rspr30 << endl;
report << "40% SPR  " << setw(13) << setprecision(4) << Fspr40 << " " << ypr40  << " " << Bspr40  << " " << spr40  << " " << Rspr40 << endl;
report << "50% SPR  " << setw(13) << setprecision(4) << Fspr50 << " " << ypr50  << " " << Bspr50  << " " << spr50  << " " << Rspr50 << endl;
report << "60% SPR  " << setw(13) << setprecision(4) << Fspr60 << " " << ypr60  << " " << Bspr60  << " " << spr60  << " " << Rspr60 << endl;
report << "               " << endl; report << "               " << endl;

report << "---------------------------------------------------------------------" << endl;
report << "PRESENT CONDITION OF STOCK" << endl;
report << "Type          F          SSB" << endl;
report << "---------------------------------------------------------------------" << endl;
report.setf(ios::scientific, ios::floatfield);
report << "CURRENT    " << setw(13) << setprecision(4) << Fcurrent       << " " << Bcurrent      << endl;
report << " /MSY adult" << setw(13) << setprecision(4) << FoverFmat      << " " << BoverBmat     << endl;
report << " /MSY fleet" << setw(13) << setprecision(4) << FFmsy      << " " << BBmsy     << endl;
report << " /MAX Y/R  " << setw(13) << setprecision(4) << FoverFmax      << " " << BoverBmax     << endl;
report << " /F0.1     " << setw(13) << setprecision(4) << FoverF01       << " " << BoverB01      << endl;
report << " /20% SPR  " << setw(13) << setprecision(4) << FoverFspr20    << " " << BoverBspr20   << endl;
report << " /30% SPR  " << setw(13) << setprecision(4) << FoverFspr30    << " " << BoverBspr30   << endl;
report << " /40% SPR  " << setw(13) << setprecision(4) << FoverFspr40    << " " << BoverBspr40   << endl;
report << " /50% SPR  " << setw(13) << setprecision(4) << FoverFspr50    << " " << BoverBspr50   << endl;
report << " /60% SPR  " << setw(13) << setprecision(4) << FoverFspr60    << " " << BoverBspr60   << endl;
report << "               " << endl; report << "               " << endl;

report << "---------------------------------------------------------------------" << endl;
report << "RELATIVE ABUNDANCE ESTIMATES by age" << endl;
report << "Year" << " ";
report.setf(ios::fixed, ios::floatfield);
for (a=1; a<=nages-1; a++) report << setw(8) << setprecision(0) << a+age(1)-1 << "     ";
report << setw(8) << setprecision(0) << nages+age(1)-1 << endl;
report << "---------------------------------------------------------------------" << endl;
for (y=1; y<=nyears; y++) {
  report.setf(ios::fixed, ios::floatfield);
  report << setw(4) << setprecision(0) << y+year(1)-1 << "  ";
  report.setf(ios::scientific, ios::floatfield);
  for (a=1; a<=nages-1; a++) report << setw(12) << setprecision(4) << n(a,y) << " ";
  report << setw(12) << setprecision(4) << n(nages,y) << endl;
}
report << "               " << endl; report << "               " << endl;

report << "---------------------------------------------------------------------" << endl;
report << "FISHING MORTALITY RATE ESTIMATES by age" << endl;
report << "Year" << " ";
report.setf(ios::fixed, ios::floatfield);
for (a=1; a<=nages-1; a++) report << setw(8) << setprecision(0) << a+age(1)-1 << "     ";
report << setw(8) << setprecision(0) << nages+age(1)-1 << endl;
report << "---------------------------------------------------------------------" << endl;
for (y=1; y<=nyears; y++) {
  report.setf(ios::fixed, ios::floatfield);
  report << setw(4) << setprecision(0) << y+year(1)-1 << "  ";
  report.setf(ios::scientific, ios::floatfield);
  for (a=1; a<=nages-1; a++) report << setw(12) << setprecision(4) << f(a,y) << " ";
  report << setw(12) << setprecision(4) << f(nages,y) << endl;
}
report << "               " << endl; report << "               " << endl;
```

```
  report << "--------------------------------------------------------------------" << endl;
  report << "RELATIVE SPAWNING BIOMASS ESTIMATES" << endl;
  report << "Year" << "      " << "Spawning biomass (B) relative to" <<endl;
  report << "Year" << "       " << "virgin level" << "       " << label << endl;
  report.setf(ios::fixed, ios::floatfield);
  report << "--------------------------------------------------------------------" << endl;
  for (y=1; y<=nyears; y++) {
    report.setf(ios::fixed, ios::floatfield);
    report << setw(4) << setprecision(0) << y+year(1)-1 << "      ";
    report.setf(ios::scientific, ios::floatfield);
    report << setw(12) << setprecision(4) << ssb(y) << "      ";
    report << setw(12) << setprecision(4) << BoverBref(y) << endl;
  }
  report << "               " << endl; report << "               " << endl;


  report << "--------------------------------------------------------------------" << endl;
  report << "INDEX (CPUE) ESTIMATES" << endl;
  report << "Series" << "  Year" << "    Observed" << "    Predicted" << "   Variance" << "    Catchability" << endl;
  report << "--------------------------------------------------------------------" << endl;
  if(n_index_series<=0) report << "  None used" << endl;
  for(series=1; series<=n_index_series; series++) {
    report.setf(ios::fixed, ios::floatfield);
    if(index_pdf(series)==0)
      report << setw(4) << setprecision(0) << series << "    " << "Not used" << endl;
    else {
      for (y=1; y<=nyears_past; y++) {
        if(y<=nyears_prehistoric) t=1; else t=y-nyears_prehistoric+1;
        report.setf(ios::fixed, ios::floatfield);
        report << setw(4) << setprecision(0) << series << "    ";
        report << setw(4) << setprecision(0) << y+year(1)-1 << "  ";
        report.setf(ios::scientific, ios::floatfield);
        if(index_obs(y,series)>=0) report << setw(12) << setprecision(4) << index_obs(y,series); else report << setw(12) << setprecision(0) << -i_one;
        report << setw(12) << setprecision(4) << index_pred(y,series);
        if(index_obs(y,series)>=0) report  << "  " << get_variance(index_pred(y,series),i_d_var(ivs(series))*overall_var,index_cv(y,series),index_pdf(series),variance_scale,variance_modify);
         else report << "            ";
        report << setw(12) << setprecision(4) << q(iqs(series),t) << endl;
      }
    }
  }
  report << "               " << endl; report << "               " << endl;

  report << "--------------------------------------------------------------------" << endl;
  report << "WEIGHT ESTIMATES by age" << endl;
  report << "Year" << " ";
  report.setf(ios::fixed, ios::floatfield);
  for (a=1; a<=nages-1; a++) report << setw(8) << setprecision(0) << a+age(1)-1 << "     ";
  report << setw(8) << setprecision(0) << nages+age(1)-1 << endl;
  report << "--------------------------------------------------------------------" << endl;
  for (y=1; y<=nyears; y++) {
    report.setf(ios::fixed, ios::floatfield);
    report << setw(4) << setprecision(0) << y+year(1)-1 << "  ";
    report.setf(ios::scientific, ios::floatfield);
    for (a=1; a<=nages-1; a++) report << setw(12) << setprecision(4) << wbyage(a,y) << " ";
    report << setw(12) << setprecision(4) << wbyage(nages,y) << " " << endl;
  }
  report << "--------------------------------------------------------------------" << endl;
  report << "SELECTIVITY AT AGE" << endl;
  for (y=1; y<=nss; y++) {
    report.setf(ios::fixed, ios::floatfield);
    report << setw(4) << setprecision(0) << y << "  ";
    report.setf(ios::scientific, ios::floatfield);
    for (a=1; a<=nages; a++) report << setw(12) << setprecision(4) << s(y,a) << " " ;
    report << "               " << endl;
  }
  report << "               " << endl; report << "               " << endl;

  #include "dusky_make_Robject32.cxx"   // write the S-compatible report  [added PBC 8/20/2010]


 ////////////////////////////////////////////////////////////////////////////////
RUNTIME_SECTION
 ////////////////////////////////////////////////////////////////////////////////
  //convergence_criteria 1.e-2, 1.e-3, 1.e-4
  convergence_criteria 1.e-3, 1.e-4, 1.e-5
  maximum_function_evaluations  1000, 4000, 4000
  //maximum_function_evaluations  50, 100, 200, 200, 200
  //maximum_function_evaluations  1, 1, 1, 1, 1

////////////////////////////////////////
PRELIMINARY_CALCS_SECTION
////////////////////////////////////////
 for(y=1;y<=nyears_past;y++){
   for(i=1;i<=n_index_series;i++){
     index_var(y,i)=log(1+index_cv(y,i));
   }
 }
 ////////////////////////////////////////////////////////////////////////////////
TOP_OF_MAIN_SECTION
 ////////////////////////////////////////////////////////////////////////////////
 // set buffer sizes
 arrmblsize=500000;
 gradient_structure::set_MAX_NVAR_OFFSET(500);
 gradient_structure::set_NUM_DEPENDENT_VARIABLES(50000);

 ////////////////////////////////////////////////////////////////////////////////
GLOBALS_SECTION
 ////////////////////////////////////////////////////////////////////////////////
 #include <admodel.h>
 #include "c:\admb\borland\bcc551\bin\admb2r.cpp"    // Include S-compatible output functions (needs preceding) [added PBC 8/20/10]
 //#include <admb2r.cpp>
  double zero, one;
  dvector lower(1,1000);
```

```
 dvector upper(1,1000);
 int ifv,imv,imd,iwv,iwd,iwn,irv,ird,irn,i_zero,i_one,i_two,current_ph,series,set,y,a,t;

 // some C++ compilers dont supply this!
 dvariable mymax(dvariable x,dvariable y)
 {
 if (x>y)
   return x;
 else
   return y;
 }
//------------------------------------------------------------------------------------
 dvariable neg_log_lklhd(dvariable obs,dvariable pred,dvariable obs_1,dvariable pred_1,
                        dvariable rho,dvariable var,dvariable modifier,int pdf,int scale, int modify, int count)
//------------------------------------------------------------------------------------
 {
   int oldcount;
   dvariable answer, alph, beta, tmp2;
   dvariable tmp;
   tmp=0.0000001; //added tmp in various places to prevent div by zero PBC 10/2010

   // compute generic negative log-likelihood formulae
   if(obs<0.0 && count>=0)
     answer=0.0; // no data or process
   else {
      oldcount=count;
     if(count<0) count = -1*count;
     switch(pdf) {
       case 1: // autocorrelated lognormal
         //cout << obs << " " << pred << " " << obs_1 << " " << pred_1 << " " << var << " " <<modifier << " " << modify << " " <<scale<<endl;
          if(pred<=0 && oldcount>=0) pred=1.0E-10; // negative oldcount means this variable is supposed to be negative;
         if(var<0)                      var=log(1.0+square(var)) ;      // convert cv to variance on log scale
           else if(scale==2){
             var=log(1.0+var/square(mymax(tmp,pred))); // convert observation variance to log scale
           }
           else if(scale==0) var=1.0;                     // automatic equal weighting
         if(modify>0) var+=modifier; else if(modify<0) var*=modifier;
         if(var<=0) cout << "Non-positive log-scale variance: " << var << " " << modifier << endl;
         if(count==1) answer= 0.5*( square(log(obs/mymax(tmp,pred)+1.0E-10))/var + log(var) );
           else answer= 0.5*( square( log(obs/mymax(tmp,pred)+1.0E-10)-rho*log(obs_1/pred_1+1.0E-10) )/var + log(var) );
     break;
       case 2: // autocorrelated normal
         if(var<0)                      var=square(var*pred);      // convert cv to variance on observation scale
           else if(scale==1) var=square(pred)*(mfexp(var)-1); // convert log-scale variance to observation scale
           else if(scale==0) var=1.0;                     // automatic equal weighting
         if(modify>0) var+=modifier; else if(modify<0) var*=modifier;
         if(var<=0) cout << "Non-positive variance: " << var << " " << modifier << endl;
         if(count==1) answer= 0.5*( square(obs-pred)/var + log(var) );
           else       answer= 0.5*( square( (obs-pred)-rho*(obs_1-pred_1) )/var + log(var) );
         break;
       case 3: // uniform
         if(pred>=lower(count) && pred<=upper(count)) answer= log(upper(count)-lower(count));
         else answer=1.0e+32;
         break;
       case 4: // uniform on log-scale
         if(pred>=lower(count) && pred<=upper(count)) answer= log(log(upper(count)/lower(count)));
         else answer=1.0e+32;
         break;
       case 5: // gamma
         if(var<0)                      var=square(var*pred);      // convert cv to variance on observation scale
           else if(scale==1) var=square(pred)*(mfexp(var)-1); // convert log-scale variance to observation scale
           else if(scale==0) var=1.0;                     // automatic equal weighting
         if(modify>0) var+=modifier; else if(modify<0) var*=modifier;
         if(var<=0) cout << "Non-positive variance: " << var << " " << modifier << endl;
         alph=pred*pred/var; beta=var/pred;
         if(pred>0) answer= alph*log(beta)-(alph-1)*log(obs)+obs/beta+gammln(alph);
         else answer=1.0e+32;
         break;
       case 6: // beta
         if(var<0)                      var=square(var*pred);      // convert cv to variance on observation scale
           else if(scale==1) var=square(pred)*(mfexp(var)-1); // convert log-scale variance to observation scale
           else if(scale==0) var=1.0;                     // automatic equal weighting
         var=var/square(upper(count)-lower(count));             // rescale variance to beta (0,1) scale
         if(var<=0) cout << "Non-positive variance: " << var << endl;
         pred=(pred-lower(count))/(upper(count)-lower(count));     // rescale prediction to beta (0,1) scale
         obs=(obs-lower(count))/(upper(count)-lower(count));       // rescale observation to beta (0,1) scale
         alph=(pred*pred-pred*pred*pred-pred*var)/var; beta=alph*(1/obs-1);
         if(pred>=0 && pred<=1) answer= (1-alph)*log(obs)+(1-beta)*log(1-obs)-gammln(alph+beta)+gammln(alph)+gammln(beta);
         else answer=1.0e+32;
         break;
       default: // no such pdf accomodated
         cout << "The pdf must be either 1 (lognormal) or 2 (normal)" << endl;
         cout << "Presently it is " << pdf << endl;
         exit(0);
     }
   }
   return answer;
 }

//------------------------------------------------------------------------------------
 dvariable get_function_parameters(int &i, int &i_in, int iph, int current_phase, dvariable best, int pdf)
//------------------------------------------------------------------------------------
 {
   if(pdf==3 || pdf==4 || pdf==6) i_in=i; else i_in=i_one;
   i=i+1;
   return best;
 }

//------------------------------------------------------------------------------------
 dvariable function_value(int nature, dvar_vector par_func, dvariable obs)
//------------------------------------------------------------------------------------
 {
   dvariable answer;
```

```
   // constants
   if(nature==1 || nature==13 || nature==14 || nature==50)
     return par_func(1);

   // polynomial of degree nature-1
   else if( nature<5) {
     if(obs == zero) return par_func(1);
     else {
       answer=par_func(1);
       for(int j=2; j<nature; j++) answer=answer+par_func(j)*pow(obs,j-1);
       return answer+par_func(nature)*pow(obs,nature-1);  // trick to avoid calculating the derivative of the final sum twice
     }
   }

   // knife edge selectivity function
   else if( nature==5) {
     if(obs < par_func(1) ) return 0; else return 1;
   }

   // logisitic selectivity function
   else if( nature==6) {
     return 1/(1+mfexp(-(obs-par_func(1))/par_func(2)));
   }

   // gamma selectivity function in terms of mode and CV (assuming sel. of oldest age is constant)
   else if( nature==7) {
   // original code:
     //return pow((mfexp(1-obs/par_func(1))*obs/par_func(1)),1.0/square(par_func(2))-1.0);
   // new entry for dusky (LIZ 2 may 2005)
     answer=pow(obs/(par_func(1)*par_func(2)),par_func(1));
     return answer*mfexp(par_func(1)-obs/par_func(2));
   }

   // Chapman-Richards growth function (reduces to vonB with par_func(4)=1
   else if( nature==8) {
     //if(par_func(5)<=0 || par_func(1) <=0 || (1-par_func(4)*mfexp(-par_func(2)*(obs-par_func(3))))<=0) cout << "Error in growth parameters" << endl; //LIZ commented out 5/23/2004;
     // original line of code:
     //return mfexp(log(par_func(5))+par_func(6)*(log(par_func(1))+log(1-par_func(4)*mfexp(-par_func(2)*(obs-par_func(3))))/par_func(4))) ;
     // hard wire to convert TL to FL: FL=0.8396TL-3.1902;  LIZ 25 april 2005
     answer=par_func(1)*(1-par_func(4)*exp(-par_func(2)/par_func(4)*(obs-par_func(3)))) ; // answer is TL
     answer=0.8396*answer-3.1902;  // answer is FL
     answer=par_func(5)*pow(answer,par_func(6)); // answer is kg
     return answer;
   }

   // Gompertz growth function
   else if( nature==9) {
     return par_func(1)*mfexp(-mfexp(-par_func(2)*(obs-par_func(3))));
   }

   // Beverton and Holt asymptotic function (par_func(1)=alpha-1)
   else if( nature==10) {
     return (par_func(1)+1)*obs/(1+obs*par_func(1));
   }

   // Ricker function (par_func(1)=alpha-1)
   else if( nature==11) {
     return obs*pow(par_func(1)+1,1-obs);
   }

   // power function y=a*x**b
   else if( nature==12) {
     if(obs == zero)return zero;
     else return par_func(1)*pow(obs,par_func(2));
   }

   // double logistic function
    else if ( nature==15)  {
       return (1/(1+mfexp(-(obs-par_func(1))/par_func(2))))*(1-(1/(1+mfexp(-(obs-par_func(3))/par_func(4)))))/par_func(5) ;
   }

   // exponential function of form:  par_func(1)*exp(par_func(2)*obs)
    else if ( nature==16)  {
       return  par_func(1)*exp(par_func(2)*obs) ;
   }

   // invalid function type
   else {
     cout << "No such function type accomodated" << endl; exit(0);
     return answer;
   }
 }

//-----------------------------------------------------------------------------------
 double get_variance(dvariable pred,dvariable var,dvariable modifier, int pdf,int scale, int modify)
//------------------------------------------------------------
 {
   switch(pdf) {
     case 1: // autocorrelated lognormal
       if(pred<0) pred=1.0E-10;
       if(var<0)                      var=log(1.0+var*var) ;      // convert cv to variance on log scale
         else if(scale==2) var=log(1.0+var/pred/pred); // convert observation variance to log scale
         else if(scale==0) var=1.0;                    // automatic equal weighting
         if(modify>0) var+=modifier; else if(modify<0) var*=modifier;
       break;
     case 2: // autocorrelated normal
       if(var<0)                     var=var*var*pred*pred;      // convert cv to variance on observation scale
         else if(scale==1) var=pred*pred*(mfexp(var)-1); // convert log-scale variance to observation scale
         else if(scale==0) var=1.0;                    // automatic equal weighting
         if(modify>0) var+=modifier; else if(modify<0) var*=modifier;
       break;
     default: // no such pdf accomodated
       exit(0);
   }
```

```
   return value(var);
 }


//----------------------------------------------------------------------------------
 dvariable spr(dvar_vector pp, dvar_vector ww, dvar_vector mm, dvar_vector ss, dvariable ff, dvariable tau ,int na)
//  Computes equilibrium spawn per recruit
//----------------------------------------------------------------------------------
 {
   dvariable answer;
   dvariable survive;
   dvariable zz;
   survive=1;
   answer=0;
   for (a=1; a<na; a++) {
     zz=mm(a)+ff*ss(a);
     answer+=pp(a)*ww(a)*mfexp(-zz*tau)*survive;
     survive=survive*mfexp(-zz);
   }
   zz=mm(na)+ff*ss(na);
   return answer+pp(na)*ww(na)*mfexp(-zz*tau)*survive/(1-mfexp(-zz));
 }

//----------------------------------------------------------------------------------
 dvariable ypr(dvar_vector ww, dvar_vector mm, dvar_vector ss, dvariable ff,int na)
//  Computes equilibrium yield per recruit
//----------------------------------------------------------------------------------
 {
   dvariable answer;
   dvariable survive;
   dvariable zz;
   survive=1;
   answer=0;
   for (a=1; a<na; a++) {
     zz=mm(a)+ff*ss(a);
     answer+=ww(a)*ss(a)*(1-mfexp(-zz))*survive/zz;
     survive=survive*mfexp(-zz);
   }
   zz=mm(na)+ff*ss(na);
   return ff*(answer+ww(na)*ss(na)*survive/zz);
 }

//----------------------------------------------------------------------------------
 dvariable equilibrium_ssb(int nature, dvar_vector par_func, dvariable spratio)
//  Computes equilibrium spawning biomass
//----------------------------------------------------------------------------------
 {
   // Beverton and Holt asymptotic function
   if( nature==10)      return ((par_func(1)+1)*spratio-1.0)/(par_func(1));      // Beverton and Holt asymptotic function
   else if( nature==11) return 1.0 + log(spratio)/log(par_func(1));              // Ricker dome
 }

//----------------------------------------------------------------------------------
 dvariable goldensection(int typ, dvariable bf, dvar_vector ww, dvar_vector mm, dvar_vector ss, int na, dvar_vector mat, dvar_vector fec, dvariable tau, dvariable spr00, int sr_nature, dvar_vector par_func)
//  Computes F's at maximum equilibrium yield per recruit and MSY
//----------------------------------------------------------------------------------
 {
   dvariable y1, y2, f0, f1, f2, f3, af, cf, sprtemp, slope0;
   double g1, g2;
   int iter;
   af=0.0001; cf=3.0; g1=0.618034; g2=0.381966;
   if(typ==i_two) {
     for (iter=1; iter<29; iter++) {
       cf=cf-0.1;
       sprtemp=spr(mat, fec, mm, ss, cf, tau, na)/spr00; y1=equilibrium_ssb(sr_nature,par_func,sprtemp)/sprtemp;
       if(y1>0) break;
     }
   }
   if(bf>(cf-0.1)) bf=bf-(bf-cf+0.1);
   f0=af; f3=cf;

   if(fabs(cf-bf)>fabs(bf-af)) { f1=bf; f2=bf+g2*(cf-bf); }
   else { f2=bf; f1=bf-g2*(bf-af); }
   y1= -ypr(ww, mm, ss, f1, na); y2= -ypr(ww, mm, ss, f2, na); // yield per recruit
   if(typ==3) { slope0=0.1*ypr(ww, mm, ss, 0.001, na); y1=fabs(slope0+y1+ypr(ww, mm, ss, f1-0.001, na)); y2=fabs(slope0+y2+ypr(ww, mm, ss, f2-0.001, na)); }
   if(typ==i_two) {
      sprtemp=spr(mat, fec, mm, ss, f1, tau, na)/spr00; y1=y1*equilibrium_ssb(sr_nature,par_func,sprtemp)/sprtemp;
      sprtemp=spr(mat, fec, mm, ss, f2, tau, na)/spr00; y2=y2*equilibrium_ssb(sr_nature,par_func,sprtemp)/sprtemp;
   }
   for (iter=1; iter<21; iter++) {
     if(y2<y1) {
       f0=f1; f1=f2; f2=g1*f1+g2*f3; y1=y2; y2= -ypr(ww, mm, ss, f2, na);
       if(typ==3) y2=fabs(slope0+y2+ypr(ww, mm, ss, f2-0.001, na));
       if(typ==i_two) {sprtemp=spr(mat, fec, mm, ss, f2, tau, na)/spr00; y2=y2*equilibrium_ssb(sr_nature,par_func,sprtemp)/sprtemp; }
     }
     else      {
       f3=f2; f2=f1; f1=g1*f2+g2*f0; y2=y1; y1= -ypr(ww, mm, ss, f1, na);
       if(typ==3) y1=fabs(slope0+y1+ypr(ww, mm, ss, f1-0.001, na));
       if(typ==i_two) {sprtemp=spr(mat, fec, mm, ss, f1, tau, na)/spr00; y1=y1*equilibrium_ssb(sr_nature,par_func,sprtemp)/sprtemp; }
     }
   }
   if(y1<y2) return f1;
   else return f2;
 }
```

Appendix B: AD Model Builder data input file for dusky sharks

```
 #/////////////////////////////////////////////////////////////////////////////////////////
# Dusky Shark Model
#/////////////////////////////////////////////////////////////////////////////////////////
```

```
#// INPUT DATA FILE FOR PROGRAM AP-MODEL
#//
#//   Important notes:
#//    (1) Comments may be placed BEFORE or AFTER any line of data, however they MUST begin
#//        with a # symbol in the first column.
#//    (2) No comments of any kind may appear on the same line as the data (the #
#//        symbol will not save you here)
#//    (3) Blank lines without a # symbol are not allowed.
#//
#// Manufactured data
#/////////////////////////////////////////////////////////////////////////////////////////
#/////////////////////////////////////////////////////////////////////////////////////////
#
####################################################
# GENERAL INFORMATION
####################################################
# first year in simulation (beginning of historical period)
# |     last year of F/effort relationship
# |     |     last year of 1st modern period
# |     |     |     last year when data are available
# |     |     |     |    end of simulation (year to project to)
# |     |     |     |     |
  1960  1979  1999  2009  2009
# year when fishing mortality rate in modern period becomes relatively constant so that no f_devs are estimated from that point on
# (enter negative value if no such period exists)  THIS NO LONGER WORKS AFTER ADOPTION OF 2ND 'MODERN' PERIOD (keep at -1)   PBC 10/2010
-1
# first and last age in the simulation
  1 40
# number of fisheries in the simulation (FLEETS ARE: RECREATIONAL, BOTTOM LONGLINE, PELAGIC LONGLINE
1
# scale of variance parameters (1 = log scale variance, 2 = observation scale variance, 0=force equal weighting)
1
# method of modifying variance parameters (0= do not modify, 1 = add annual values to variance, -1 = multiply annual values by variance)
#note - adding now assumed for indices; this switch doesn't work for them PBC 10/2010
-1
# spawning season (integer representing number of months elapsed when spawning occurs)
6
# maturity schedule (shifted by 1 year to account for gestation )
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.01 0.02 0.05 0.13 0.28  0.51 0.74  0.88 0.95 0.98 0.99 1 1 1 1 1 1 1 1 1 1 1 1 1  1 1
#0 2.38124E-05 4.19479E-05 7.38942E-05 0.000130167 0.000229283 0.00040384 0.000711197  0.001252186  0.002203784 0.003875742 0.006807519 0.011930458 0.020827773 0.036117846 0.061922807  0.104171967
0.170022594  0.265177633 0.388649957 0.528281694 0.66362395 0.776558819 0.859598887  0.915149823 0.95 0.970990047  0.983323137 0.990464513 0.994564709 0.996907352 0.998242087  0.99900135 0.999432865
0.999677983 0.999817179  0.999896212 0.999941082 0.999966554 0.999981014
# fecundity schedule : (7.13 pups_per_female)*(0.5 for sex ratio)*(0.33 for triennial reproductive cycle)
1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187  1.187 1.187  1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187 1.187  1.187 1.187 1.187 1.187  1.187 1.187 1.187 1.187
1.187 1.187 1.187 1.187  1.187 1.187 1.187 1.187
#1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775  1.775 1.775  1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775 1.775  1.775 1.775 1.775 1.775  1.775 1.775 1.775 1.775
1.775 1.775 1.775 1.775  1.775 1.775 1.775 1.775
#3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55  3.55 3.55  3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55  3.55 3.55 3.55 3.55  3.55 3.55 3.55 3.55 3.55 3.55 3.55 3.55  3.55 3.55
3.55 3.55
#natural mortality switch (0 = use parametric distribution in .prm file; 1 = use mortality at age vector provided below)  PBC added 10/2010
1
#natural mortality @ age vector  PBC 10/2010
0.104 0.104 0.104 0.104 0.104 0.098 0.092 0.088 0.084 0.080 0.077 0.074  0.072 0.070  0.068 0.066 0.064 0.063 0.061 0.060 0.059 0.058 0.057 0.056  0.055 0.054 0.053 0.052  0.052 0.051 0.048 0.048
0.048 0.048 0.048 0.048  0.048 0.048 0.048 0.048
#age independent natural mortality (used to calculate MSST) PBC 11/2010
0.0666
#F_1999 switch (if 1, estimate as a free parameter) PBC 10/2010
0
####################################################
# INDICES OF ABUNDANCE (e.g., CPUE) If there are no series, there should be no entries between the comment lines.
####################################################
# number of index data series
6
# pdf of observation error for each series (1) lognormal, (2) normal
  1  1  1  1  1  1
# units (1=numbers, 2=weight, 10=number relative to virgin levels, 20=weight relative to virgin levels (in case of 10 or 20, you should fix the corresponding q to 1)
  1  1  1  1  1   20
# months elapsed at time index observed
  6  6  6  6  6   1
# option to (1) scale or (0) not to scale index observations
  0  0  0  0  0  0
# set of index variance parameters each series is linked to
  1  2  3  4  5  6
#  1  1  1  1  1  1  1
# set of q parameters each series is linked to
  1   2   3   4   5   6
# set of selection parameters each series is linked to
  1 2 3 4 5 6
# selectivity of age zeroes  NOTE: 1/2 year of pup survival will be applied to match indices wiht sel_age0>0   PBC 11/3/2010
  1.0 0.0 0.0 1.0 0.0 0.0
# observed indices by series
# VIMS-LL  LPS BLLOP NELL PLLOP rel index year
-1 -1 -1 -1 -1 1 1960
-1 -1 -1 -1 -1 -1 1961
-1 -1 -1 -1 -1 -1 1962
-1 -1 -1 -1 -1 -1 1963
-1 -1 -1 -1 -1 -1 1964
-1 -1 -1 -1 -1 -1 1965
-1 -1 -1 -1 -1 -1 1966
-1 -1 -1 -1 -1 -1 1967
-1 -1 -1 -1 -1 -1 1968
-1 -1 -1 -1 -1 -1 1969
-1 -1 -1 -1 -1 -1 1970
-1 -1 -1 -1 -1 -1 1971
-1 -1 -1 -1 -1 -1 1972
-1 -1 -1 -1 -1 -1 1973
-1 -1 -1 -1 -1 -1 1974
4.152081718 -1 -1 -1 -1 0.83 1975
-1 -1 -1 -1 -1 -1 1976
0.194114188 -1 -1 -1 -1 -1 1977
-1 -1 -1 -1 -1 -1 1978
-1 -1 -1 -1 -1 -1 1979
2.207717062 -1 -1 -1 -1 -1 1980
1.759660005 -1 -1 -1 -1 -1 1981
```

```
-1 -1 -1 -1 -1 -1 1982
-1 -1 -1 -1 -1 -1 1983
-1 -1 -1 -1 -1 -1 1984
-1 -1 -1 -1 -1 -1 1985
-1 2.166388685 -1 -1 -1 -1 1986
-1 2.169591033 -1 -1 -1 -1 1987
-1 1.838147975 -1 -1 -1 -1 1988
-1 1.887784375 -1 -1 -1 -1 1989
0.061208279 1.425045033 -1 -1 -1 -1 1990
0.082101274 1.423443859 -1 -1 -1 -1 1991
0.021248103 0.454733471 -1 -1 4.099330469 -1 1992
0.339353425 1.256921743 -1 -1 1.906665334 -1 1993
-1 0.541196878 0.6820312 -1 3.10102928 -1 1994
0.164055244 0.602041497 1.443153367 -1 1.239332467 -1 1995
0.49994799 0.986323304 1.233613661 8.19E-02 1.215948836 -1 1996
-1 0.943091601 2.245361751 -1 0.555810932 -1 1997
0.16859683 0.513976916 1.346600758 0.346920437 1.44798641 -1 1998
0.816692158 0.539595704 2.204275534 -1 0.390326771 -1 1999
1.234801795 0.505971045 0.735443282 -1 0.81662836 -1 2000
0.29274181 0.307425445 0.92649419 0.374575127 0.352553213 -1 2001
0.939995144 0.6452732 0.28041343 -1 0.172679125 -1 2002
0.170990722 0.417906465 0.371830263 -1 0.104326971 -1 2003
0.971195603 0.614850891 0.408807858 1.083311922 0.564804637 -1 2004
2.087135 0.734938955 0.454002696 -1 0.456880184 -1 2005
2.68798168 0.339448929 0.569044103 -1 0.81662836 -1 2006
0.275718067 1.22169591 0.679976889 1.006405736 0.32737084 -1 2007
0.124219064 1.48108613 0.954227387 -1 0.226641351 -1 2008
2.74844484 0.983120955 1.464723631 3.106921797 0.20505646 -1 2009
# annual scaling factors for variance (use this option to account for annual differences in the variance, e.g., to down-weight observations based on very little data)
#these are CVs for each index
#VIMS-LL LPS BLLOP NELL PLLOP rel I year
1 1 1 1 1 1 1960
1 1 1 1 1 1 1961
1 1 1 1 1 1 1962
1 1 1 1 1 1 1963
1 1 1 1 1 1 1964
1 1 1 1 1 1 1965
1 1 1 1 1 1 1966
1 1 1 1 1 1 1967
1 1 1 1 1 1 1968
1 1 1 1 1 1 1969
1 1 1 1 1 1 1970
1 1 1 1 1 1 1971
1 1 1 1 1 1 1972
1 1 1 1 1 1 1973
1 1 1 1 1 1 1974
0.517967964 1 1 1 1 0.202 1975
1 1 1 1 1 1 1976
1.921390289 1 1 1 1 1 1977
1 1 1 1 1 1 1978
1 1 1 1 1 1 1979
0.542346839 1 1 1 1 1 1980
0.519144033 1 1 1 1 1 1981
1 1 1 1 1 1 1982
1 1 1 1 1 1 1983
1 1 1 1 1 1 1984
1 1 1 1 1 1 1985
1 0.123 1 1 1 1 1986
1 0.121 1 1 1 1 1987
1 0.298 1 1 1 1 1988
1 0.168 1 1 1 1 1989
2.539903017 0.154 1 1 1 1 1990
2.292280987 0.16 1 1 1 1 1991
5.18132773 0.292 1 1 0.274 1 1992
1.242009261 0.242 1 1 0.218 1 1993
1 0.377 0.39 1 0.217 1 1994
1.835483785 0.322 0.34 1 0.258 1 1995
0.861412327 0.412 0.34 0.749211298 0.29 1 1996
1 0.378 0.36 1 0.353 1 1997
1.52575651 0.491 0.38 0.528330768 0.296 1 1998
0.945595917 0.677 0.39 1 0.392 1 1999
0.682447462 0.526 0.66 1 0.307 1 2000
1.277351042 0.658 0.44 0.484182628 0.373 1 2001
0.949115836 0.611 0.51 1 0.889 1 2002
2.162337588 0.38 0.37 1 0.632 1 2003
0.712542783 0.337 0.38 0.306838177 0.311 1 2004
0.689898558 0.335 0.5 1 0.297 1 2005
0.498442566 0.458 0.55 1 0.284 1 2006
1.118394279 0.242 0.66 0.516586471 0.32 1 2007
2.036706755 0.208 0.62 1 0.425 1 2008
0.747135782 0.257 0.32 0.340328548 0.294 1 2009
##################################################
# INDEX OF RELATIVE EFFORT (you must enter values for each year, even if they are only dummy values)
##################################################
# how to treat effort data (0) do not use values below, instead replace with a default of 1.0 for all years
# |                         (-1) use values below
# |                         (1) use values below, then rescale relative to maximum value
-1
#PLL only
# for PLLOP, all values are relative to 2006
0.136078863  1960
0.152192852 1961
0.313712274 1962
0.344763898 1963
0.518518    1964
0.532013406 1965
0.370241883 1966
0.306795922 1967
0.350879073 1968
0.474688886 1969
0.531462997 1970
0.708082968 1971
0.748572649 1972
0.744727613 1973
```

```
0.746467158 1974
1.050098531 1975
0.982828729 1976
0.967009329 1977
0.821613703 1978
0.64839896  1979
0.684916931294 1980
0.860950275587 1981
0.923392892881 1982
0.853895079174 1983
0.905531078468 1984
1.03618806762  1985
1.115859891055 1986
0.748962763349 1987
 0.85925219342 1988
0.988216909936 1989
0.99376136923  1990
 1.09471398935 1991
1.092475992956 1992
1.063453029393 1993
1.134114656742 1994
1.225265375072 1995
1.413870523441 1996
1.420528592152 1997
1.265261702383 1998
1.268601523701 1999
1.212533863577 2000
1.132079571603 2001
0.938286141625 2002
1.0035313743   2003
1.132829779698 2004
1.036663967425 2005
1.0            2006
1.048678166956 2007
1.048678166956 2008
1.048678166956 2009
####################################################
#  Selectivity weightings for each fishery by year    Added 10/2010 by PBC
####################################################
#number of selectivities to average over for total apical F
3
#which selectivities contribute to total apical F
2 3 5
#weighting of each selectivity by year
0.005612407 0.005612407 0.988775186
0.004652321 0.004652321 0.990695358
0.005174252 0.005174252 0.989651496
0.005446681 0.005446681 0.989106638
0.006145301 0.006145301 0.987709398
0.006069437 0.006069437 0.987861127
0.003948137 0.003948137 0.992103726
0.003614537 0.003614537 0.992770927
0.00430233 0.00430233   0.991395339
0.003583029 0.003583029 0.992833943
0.003202719 0.003202719 0.993594561
0.002407699 0.002407699 0.995184602
0.002278062 0.002278062 0.995443876
0.00228977 0.00228977   0.99542046
0.002284458 0.002284458 0.995431084
0.001626065 0.001626065 0.996747869
0.001736975 0.001736975 0.99652605
0.00176529 0.00176529   0.99646942
0.002076384 0.002076384 0.995847231
0.002628159 0.002628159 0.994743682
0.099648613 0.051503778 0.848847609
0.145544857 0.075225431 0.779229712
0.18709156 0.096699008  0.716209432
0.239701388 0.123890604 0.636408008
0.265301424 0.137122084 0.597576492
0.272871884 0.141034906 0.58609321
0.285759953 0.147696155 0.566543892
0.372964074 0.192767948 0.434267978
0.369787136 0.191125936 0.439086928
0.364177562 0.188226605 0.447595832
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
0.378723404 0.195744681 0.425531915
####################################################
# Projection specifications
####################################################
# selectivity for reference points (1=fishery, 2=use maturity vector)
1
# non-negative=input reference (should have value between 0 and 1)
# otherwise, -0.1=B at F0.1, -1=B at msy, -2=B at Fmax, -20=Bspr20, -30=Bspr30, -40=Bspr40, -50=Bspr50, -60=Bspr60, -999=Bcurrent)
-1
# control for recruitment deviations (0=none, + = variance, - = -cv)
0
```

```
# projected F values (non-negative=input F, -0.1=F0.1, 1=Fmsy, -2=Fmax, -20=Fspr20, -30=Fspr30, -40=Fspr40, -50=Fspr50, -60=Fspr60, -999=Fcurrent)
# |       Std. error (or negative CV) of implementation uncertainty (no being used at present)
# |       |       year
# |       |       |
  -999    -0.01    2010
  -999    -0.01    2011
  -999    -0.01    2012
  -999    -0.01    2013
  -999    -0.01    2014
  -999    -0.01    2015
  -999    -0.01    2016
  -999    -0.01    2017
  -999    -0.01    2018
  -999    -0.01    2019
  -999    -0.01    2020
  -999    -0.01    2021
  -999    -0.01    2022
  -999    -0.01    2023
  -999    -0.01    2024
  -999    -0.01    2025
  -999    -0.01    2026
```

Appendix C: AD Model Builder parameter input file for dusky sharks

```
#///////////////////////////////////////////////////////////////////////////////////////////
#///////// ---- DUSKY SHARK ASSESSMENT - Oct 2010 ---    ////////////////////////////////////
#///////////////////////////////////////////////////////////////////////////////////////////
#// PARAMETER FILE FOR PROGRAM DATAPOOR
#//
#//   Important notes:
#//    (1) Comments may be placed BEFORE or AFTER any line of data, however they MUST begin
#//        with a # symbol in the first column.
#//    (2) No comments of any kind may appear on the same line as the data (the #
#//        symbol will not save you here)
#//    (3) Blank lines without a # symbol are not allowed.
#//
#///////////////////////////////////////////////////////////////////////////////////////////
#///////////////////////////////////////////////////////////////////////////////////////////
#
####################################################
# DIMENSION ARRAYS
####################################################
# total number of process parameters (basically, everything before process deviation parameters)
43
# number of sets of each parameter type
#  no. of survey catchabilities q
#  | no. of selectivity vectors (must be 2 sets of selection parameters for each fishery representing the prehistoric and modern periods, plus parameters for the indices)
#  | |  index variances
#  | | |
6 6 6
####################################################
# SPECIFICATIONS FOR PROCESS PARAMETERS
####################################################
# nature of function (1=constant, 2-3=polynomials, 13=process correlation, 14=process variance scaling parameter
# | best guess of parameter value (median of prior)
# | |    lower bound for parameter
# | |  |  upper bound for parameter
# | |  |  |  phase of estimation (enter -1 to fix at best guess and not estimate)
# | |  |  |  | probability density function of prior (0=none, 1=lognormal, 2=normal)
# | |  |  |  | | negative value is read as CV, positive value is read as standard error (must be on logscale if overall_pdf=1, aritmetic scale otherwise) of prior
# | |  |  |  | |     |
#---------------------------
#Apical fishing mortality parameters
#...................................................................
#LPL fishery
#F/effort regression parameters for expected F during prehistoric era
  2 0 -0.01E-09     0.7 -1 2 -1
  2 0.10     0.01E-11     0.7 1 0 -0.5
#---------------------------
#Population parameters (looks like the code expects best estimate=median)
#m  natural mortality rate function of form:  par_func(1)*exp(par_func(2)*obs)
    16       0.206E+00       0.5000E-01       0.5000E+00     -1     1      -0.3000E+00
    16      -0.050E+00      -0.5000E+00       0.5000E+00     -1     1      -0.3000E+00
#r   pup-survival LN(mode=0.72; median=0.78; mean=0.82), CV=0.3
    10       0.814E+00       0.5000E+00       0.9800E+00     3     1      -0.300E+00
#von bert and L-W  (use TL for vonBert and convert to FL in program; W params are for FL to kg)
8 4.21E+02 1.00E-04 1.00E+12 -1 0 1.00E+00
8 3.90E-02 0.00E+00 1.00E+12 -1 0 1.00E+00
8 -7.04E+00 -9.00E+00 1.00E+12 -1 0 1.00E+00
8 1.00E+00 0.00E+00 1.00E+12 -1 0 1.00E+00
8 3.24E-05 0.00E+00 1.00E+12 -1 0 1.00E+00
8 2.79E+00 0.00E+00 1.00E+12 -1 0 1.00E+00
#q (VIMS  LPS  BLLOP  NELL  PLLOP  relative_B)
    1       0.7410E-01       0.1000E-03       0.1000E+02     1     0       0.1000E+01
    1       0.2200E-01       0.1000E-03       0.1000E+02     1     0       0.1000E+01
    1       0.3200E-01       0.1000E-03       0.1000E+02     1     0       0.1000E+01
    1       0.1200E-01       0.1000E-03       0.1000E+02     1     0       0.1000E+01
    1       0.1700E+01       0.1000E-03       0.2000E+02     1     0       0.1000E+01
    1       0.1000E+01       0.1000E-03       0.1000E+03    -1     0       0.1000E+01
#---------------------------
#Fishery selection parameters  Note: Now using index selection parameters only with a pointer to the ones used to model F (see data file); PBC 10/2010
#.........................................................................................
#s_prehistoric fishery 3 -- PLLOP
#     15       0.2000E+01       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#     15       0.6000E+00       0.1000E-01       0.2800E+02    -2     0      -0.3000E+00
#     15       0.2800E+02       0.0000E+00       0.5000E+02    -2     0       0.0000E+00
#     15       0.5000E+01       0.1000E-01       0.5800E+02    -2     0      -0.3000E+00
#     15       0.9870E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#s_modern fishery 3 -- PLLOP
#     15       0.2000E+01       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#     15       0.6000E+00       0.1000E-01       0.2800E+02    -2     0      -0.3000E+00
```

```
#     15       0.2800E+02       0.0000E+00       0.5000E+02    -2     0       0.0000E+00
#     15       0.5000E+01       0.1000E-01       0.5800E+02    -2     0      -0.3000E+00
#     15       0.9870E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#.........................................................................................
#----------------------------
#Index selection parameters
#----------------------------
#_s_survey 1 (VIMS LL - fishery independent)
     15       0.001E+00       0.000E+00       0.2000E+02    -2     0       0.0000E+00
     15       0.25E+00       0.1000E-01       0.2800E+02    -2     0      -0.3000E+00
     15       2.00E+00       0.0000E+00       0.5000E+02    -2     0       0.0000E+00
     15       4.5000E+00       0.1000E-01       0.5800E+02    -2     0      -0.3000E+00
     15       0.5453E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#_s_survey 2 (LPS)
     15       3.0000E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
     15       0.0599E+00       0.1000E-01       0.2800E+02    -2     0      -0.3000E+00
     15       1.4054E+01       0.0000E+00       0.5000E+02    -2     0       0.0000E+00
     15       4.3270E+00       0.1000E-01       0.5800E+02    -2     0      -0.3000E+00
     15       0.9108E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#_s_survey 3 (BLLOP)
     5        1.0000E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
     #15       0.1000E+01       0.1000E-01       0.2800E+02    -2     0      -0.3000E+00
     #15       0.3200E+02       0.0000E+00       0.5000E+02    -2     0       0.0000E+00
     #15       0.4000E+01       0.1000E-01       0.5800E+02    -2     0      -0.3000E+00
     #15       0.9940E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#_s_survey 4 (NELL)
     6       3.0961E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
     6       0.28305E+00       0.1000E-01       0.2800E+02    -2     0      -0.3000E+00
#_s_survey 5 (PLLOP)
     15       2.2000E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
     15       0.8205E+00       0.1000E-01       0.2800E+02    -2     0      -0.3000E+00
     15       1.3562E+01       0.0000E+00       0.5000E+02    -2     0       0.0000E+00
     15       7.7744E+00       0.1000E-01       0.5800E+02    -2     0      -0.3000E+00
     15       0.7268E+00       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#_s_survey 6 (relative total B)
     5       0.1000E+01       0.0000E+00       0.2000E+02    -2     0       0.0000E+00
#----------------------------
#Variance parameters
#----------------------------
#idv - 'additional variance' parameters (PBC 10/2010)
     14      0.4    0       2       3      0       0.1
     14      0.4    0       2       3      0       0.1
     14      0.4    0       2       3      0       0.1
     14      0.4    0       2       3      0       0.1
     14      0.4    0       2       3      0       0.1
     14      1.000E-08    0       2       -1      0       0.1
#overall var
     1       -1.000E-04   -3     -0.00001       -3     2       -0.5
####################################################
# SPECIFICATIONS FOR PROCESS DEVIATION PARAMETERS
####################################################
# best guess of parameter value (central tendency of prior)
# | lower bound for parameter
# |   | upper bound for parameter
# |   |    | phase of estimation (enter -1 to fix at best guess and not estimate)
# |   |    |    | probability density function of prior
# |   |    |    |   | standard error or negative CV of prior (superfluous in case of deviations)
#_f___|       |   |    |   | |
# correlation coef.
      0.50  -0.001 1.0   -1    0  0.1
# logscale variance of deviations
      0.1     0    1000. -1    0  0.1
# deviations themselves
      -0.01  -1     0.7     2    1   0.1
#_r___|       |   |    |   | |
      0.5      -0.001 1.0   -1    0  0.1
      0.15    0     100.0  -1    0  0.1
      0.0000 -4     4      -3    1   0.1
#_q___|       |   |    |   | |
      0.0    -0.001 1.0   -1    0  0.1
      0.10    0     100.0 -1    0  0.1
      0.0000 -5     5      -1    1   0.1
# End of file #
```