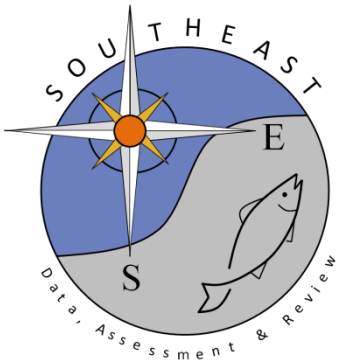
The tpl file, data file, and control file for a Stochastic Stock Reduction Analysis (SSRA) program

Joseph Munyandorero

SEDAR47-RW-01

17 May 2016
Updated 24 May 2016



Please cite this document as:

Munyandorero, J. 2016. The tpl file, data file, and control file for a Stochastic Stock Reduction Analysis (SSRA) program. SEDAR47-RW-01. SEDAR, North Charleston, SC. 35 pp.

This working paper contains the tpl file, data file, and control file for a Stochastic Stock Reduction Analysis (SSRA) program. The code implements the SSRA described by Martell, S.J.D, Pine III, W.E., and C. J. Walters. 2008. Parameterizing age-structured models from a fisheries management perspective. Can. J. Fish. Aquat. Sci. 65: 1586-1600. Some modifications have been made to the original code.

SSRA.tpl

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// [Stochastic] Stock Reduction Analysis
//  using Martell et al. (2008) parameterization
//  for Fmsy and MSY, adapted largely from code
//  from S. Martell for the pub (meanage.tpl)
//
// Note: run in MCMC mode to make 'stochastic'
//
// Wade Cooper FWCC-FWRI 2014
// JM FWCC - FWRI 2015, calculate annual egg production as
// dvariable eggs =sum(elem_prod(nt(i),fa)); instead of
// dvariable E0=sum(elem_prod(nt(i),fa)); as did Wade C
// J. Munyandorero and Joe O'
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

TOP_OF_MAIN_SECTION
 gradient_structure::set_NUM_DEPENDENT_VARIABLES(6000);
 gradient_structure::set_MAX_NVAR_OFFSET(20000);
 arrmblsize = 40000000;
//gradient_structure::set_GRADSTACK_BUFFER_SIZE(35000000); // This is about the highest I can get it on my laptop, but Taylor and Hicks say 2e9 is the windows limit
//gradient_structure::set_CMPDIF_BUFFER_SIZE(35000000);   // This is about the highest I can get it on my laptop, but Taylor and Hicks say 2e9 is the windows limit

GLOBALS_SECTION                                 // added 25Mar2016 JRO
 #include <admodel.h>                           // added 25Mar2016 JRO
 ofstream basicMCMC("mcmc_results.dat");        // added 25Mar2016 JRO

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
DATA_SECTION

 init_adstring DataFile;
 !! ad_comm::change_datafile_name(DataFile);

 init_int testing;

 //###################################
```

```
 //Mode configuration years
 init_int syr;                                                    //start year of
model and catch
 init_int eyr;                                                    //end year of
model and catch

 //###################################
 //Get life history vectors
 init_int Amax;                                                   //max age
 init_number spawn_offset                                 //spawning offset (fraction of year at month of
peak spawning)
 init_vector la(0,Amax);                                          //lenght at age
 init_vector wa(0,Amax);                                          //weight at age
 init_vector wa_spawn(0,Amax);                            // wgt@age at peak spawning
 init_vector ma(0,Amax);                                          //maturity at age
 init_vector fa(0,Amax);                                  //fecundity at age
 init_vector morta(0,Amax);                               //mortality at age

 !!if (testing==1) cout<<"la: "<<la<<"\nwa: "<<wa<<"\nm: "<<ma<<"\nfa: "<<fa<<"\nmorta:
"<<morta<<endl;

 //###################################
 //Get catch time series
 init_vector Ct(syr,eyr);                                 //catch
 int nyr;
        //number of years
 !!nyr=(eyr-syr)+1;

 //###################################
 //Get Indices Estimates and CVs
 init_int inum;                                                   //number of indices
 init_vector isyr(1,inum);                                        //start year of indices
 init_vector ieyr(1,inum);                                        //end year of indices
 init_vector itype(1,inum);                               //1=biomass, 2=number
 init_matrix iIt(1,inum,isyr,ieyr);                       //indices of abundance
 matrix It(1,inum,isyr,ieyr);
 init_matrix ItCV(1,inum,isyr,ieyr);                      //indices of abundance
 matrix ItVar(1,inum,isyr,ieyr)

 LOCAL_CALCS
 for (int i=1; i<=inum; i++){
   for (int y=isyr(i); y<=ieyr(i); y++){
     ItVar(i,y)=log(ItCV(i,y)*ItCV(i,y)+1);
   }
```

```
 }
 END_CALCS

 !!if (testing==1) cout<<"Indices of abundance\n"<<iIt<<endl;

 //#################################
 //Fisheries vulnerability section
 init_int numVaBlocks                                                    //vulnerability blocks
 init_matrix vulMat(1,numVaBlocks,0,Amax+1)          //1st index is 1st year of block
 matrix va(syr,eyr,0,Amax)                                               //year and age specific
vulnerability

 LOCAL_CALCS
 int curBlock=1;
 if (vulMat(1,0)!=syr){cout<<"Need to set first vul block year to start year!";exit(1);}
 for (int y=syr; y<=eyr; y++){
  for (int a=0; a<=Amax; a++) {
   if (curBlock<numVaBlocks && y==vulMat(curBlock+1,0)) curBlock++;
   va(y,a)=vulMat(curBlock,a+1);
  }
 }
 END_CALCS

 !!if (testing==1) cout<<"Vulnerability at age matrix\n"<<va<<endl;

 //#################################
 //Index selectivity section
 init_vector numItSelBlocks(1,inum)                                      //index
selecitivty blocks
 int numItSelObs
 !!numItSelObs=sum(numItSelBlocks);
 init_matrix ItSelMat(1,numItSelObs,0,2+Amax)          //1st index is 1st year of block
 3darray ItSel(1,inum,syr,eyr,0,Amax)                                    //year and age
specific selectivities

 LOCAL_CALCS
 ItSel.initialize(); //set all == 0
 if (testing==1) cout<<"Number of It Selectivity observations: "<<numItSelObs<<endl;

 //set the observations to the appropriate place in the array
 for (int i=1; i<=numItSelObs; i++){  //loop through ItSelMat input matrix
  int index=ItSelMat(i,0);
  int year=ItSelMat(i,1);
  if (year==isyr(index)) year=syr;  //set the first observation to the start year of the model
```

```
   for (int a=0; a<=Amax; a++){
     ItSel(index,year,a)=ItSelMat(i,a+2);
   }
  }

  //loop through array and fill in
  for (int i=1; i<=inum; i++){
   for (int y=syr+1; y<=eyr; y++){
    if (sum(ItSel(i,y))==0) ItSel(i,y)=ItSel(i,y-1);
   }
  }

  if (testing==1) cout<<"ItSel\n"<<ItSel<<endl;

 END_CALCS

  //###################################
  //Get Composition Data -- SS format where fishery=fleet 1, indices=fleet 2+
  init_int nobs_comp                                                    //number of comp
observations
  init_matrix compdata(1,nobs_comp,0,3+Amax)          //first column is fleet num, 2nd is year, 3rd+ is
age comp data
  3darray compobs(1,1+inum,syr,eyr,0,Amax)                  //array to store the data
  matrix sampN(1,1+inum,syr,eyr)                                    //input sample size
  !!compobs.initialize();                                           //set's to zero
  !!sampN.initialize();

 // !!cout << "compdata =" << compdata << endl;
 // !!exit(100);

 LOCAL_CALCS
  for (int i=1; i<=inum; i++) { //loop through and scale input indices to mean of 1 for obj fnx
   It(i)=iIt(i)/mean(iIt(i));
  }

  for (int i=1; i<=nobs_comp; i++) { //loop through composition data rows (fleets x years)
   int fleet=compdata(i,0);
   int year=compdata(i,1);
   sampN(fleet,year)=compdata(i,2);
   for (int j=0; j<=Amax; j++) { //loop through columns (fleet #, year, ages)
    compobs(fleet,year,j) = compdata(i,j+3);
   }
  }
```

```
 if (testing==1) cout << "Composition data\n" << compobs << endl;
END_CALCS

 //#################################
 //Initial parameter guesses, priors, & phaze
 //Fmsy
 init_number iFmsyLow;
 init_number iFmsyHi;
 init_number iFmsy;
 init_number iFmsyPrior;
 init_number iFmsyPriorCV;
 number iFmsyPriorVar;
 init_int phz_Fmsy;
 !!iFmsyPriorVar=log(iFmsyPriorCV*iFmsyPriorCV+1);
 number logFmsyLow;
 number logFmsyHi;

 //MSY
 init_number iMSYLow;
 init_number iMSYHi;
 init_number iMSY;
 init_number iMSYPrior;
 init_number iMSYPriorCV;
 number iMSYPriorVar;
 init_int phz_MSY;
 !!iMSYPriorVar=log(iMSYPriorCV*iMSYPriorCV+1);
 number logMSYLow;
 number logMSYHi;

// !!cout << "iFmsyLow =" << iMSYLow << endl;
// !!cout << "iFmsyHi =" << iMSYHi << endl;
// !!cout << "iFmsy =" << iMSY << endl;
// !!cout << "iFmsyPrior =" << iMSYPrior << endl;
// !!cout << "iFmsyPriorCV =" << iMSYPriorCV << endl;
// !!cout << "phz_Fmsy =" << phz_MSY << endl;
// !!cout << "iFmsyPriorVar =" << iMSYPriorVar << endl;
// !!exit(100);

//Recruit deviations
init_number iRecdevLow;
init_number iRecdevHi;
init_int    phz_Recdevs;
 //  !!cout << "Low dev =" << iRecdevLow << endl;
 //  !!cout << "High dev =" << iRecdevHi << endl;
```

```
 //  !!cout << "Phase dev =" << phz_Recdevs << endl;
 //  !!exit(100);
 init_number sd_wt;

 LOCAL_CALCS
  logFmsyLow=log(iFmsyLow);
  logFmsyHi=log(iFmsyHi);
  logMSYLow=log(iMSYLow);
  logMSYHi=log(iMSYHi);

  //get block number
 END_CALCS

  //#################################
  //Priors
  //recruit compensation prior
  init_number ireck;                                          //recruit compensation
prior
  init_number ireckCV;
  number ireckVar                                                  //not lognormal
  !!ireckVar=square(ireck*ireckCV);

  //F observations prior
  init_number numFObs                                         //number of F
observations
  init_vector FObsYear(1,numFObs)                          //year with F obs estimate
  init_vector FObsPrior(1,numFObs)                         //F obs prior estimate
  init_vector FObsCV(1,numFObs)                                //CV of F obs estimate
  vector FObsVar(1,numFObs)                                    //lognormal variance of
F obs estimate

 LOCAL_CALCS
  for (int i=1; i<=numFObs; i++){
    FObsVar(i)=log(FObsCV(i)*FObsCV(i)+1);
  }
 END_CALCS

 // !!cout << "numFObs =" << numFObs << endl;
 // !!cout << "FObsYear =" << FObsYear << endl;
 // !!cout << "FObsPrior =" << FObsPrior << endl;
 // !!cout << "FObsCV =" << FObsCV << endl;
 // !!exit(100);
```

```
 //#################################
 //Lambda
 init_vector lamIndices(1,inum)
 init_vector lamComps(1,1+inum)
 init_number lamFmsyPr
 init_number lamMSYPr
 init_number lamRecKPr
 init_vector lamFObs(1,numFObs)

//  !!cout << "lamIndices =" << lamIndices << endl;
//  !!cout << "lamComps =" << lamComps << endl;
//  !!cout << "lamFmsyPr =" << lamFmsyPr<< endl;
//  !!cout << "lamMSYPr =" <<lamMSYPr << endl;
//  !!cout << "lamRecKPr =" << lamRecKPr<< endl;
//  !!cout << "lamFObs =" <<lamFObs << endl;
//  !!exit(100);

 init_int  st_recent_yrs                    // added 31Mar2016 JOMU
 int       n_recent_yrs                     // added 31Mar2016 JOMU
 !!n_recent_yrs = (eyr - st_recent_yrs) + 1;     // added 31Mar2016 JOMU

 // !!cout << "start recent yrs =" << st_recent_yrs<< endl;
 // !!cout << "number recent =" <<n_recent_yrs << endl;
 // !!exit(100);

 //#################################
 //Set up the survivorship vectors
 // and virgin eggs/biomass per recruit
 vector lxo(0,Amax)
 vector lxo_spawn(0,Amax)
 number phiEo
 number phiBo

 LOCAL_CALCS
//get virgin survival - these are permanant
 lxo(0)=1.0;
 lxo_spawn(0)=1.0*exp(-morta(0)*spawn_offset);
 for (int i=1; i<=Amax; i++) {
  lxo(i)=lxo(i-1)*exp(-morta(i-1));
  lxo_spawn(i)=lxo_spawn(i-1)*exp(-morta(i-1)*spawn_offset);
 }
 phiEo = sum(elem_prod(lxo_spawn,fa));
 phiBo= sum(elem_prod(elem_prod(lxo_spawn,wa_spawn),ma));                    //note: use end
year/last block vulnerability
```

```
                                        // use proportion mature ma, instead of va(eyr)
                                        // also use weight@ age at the time of spawning
 cout<<"lxo: " << lxo <<endl;
  cout<<"lxo@spawn: " << lxo_spawn <<endl;
 cout <<"phiEo: " << phiEo <<endl;
 cout <<"phiBo: " << phiBo <<endl;

 END_CALCS


 int iterMCMC;
 int curr_phase                         // added 25Mar2016 JRO


 LOCAL_CALCS
    basicMCMC << "negLL" ;
    for (int y=syr ; y<=eyr; y++) basicMCMC << "\tF"  <<y ;
    for (int y=syr ; y<=eyr; y++) basicMCMC << "\tN"  <<y ;
    for (int y=syr ; y<=eyr; y++) basicMCMC << "\tVB" <<y ;
    for (int y=syr ; y<=eyr; y++) basicMCMC << "\tBt" <<y ;
     for (int y=syr ; y<=eyr; y++) basicMCMC<< "\tSSB" <<y ;
    basicMCMC
<<"\tFmsy\tMSY\tBmsy\tSo\tbeta\tRo\tReck\th\tBo\tEo\tcur_F\tcur_VB\tFratio\tBratio"<<endl ;
 END_CALCS

  //EOF test value
  init_int TestVal;
  !! if (TestVal != 123456) { cout << "Test Number is not 123456" << endl; exit(1); }
  !! if(TestVal == 123456) { cout << "Data loaded successfully" << endl; }

  //diagnostic files in testing
  //!! ofstream calclog("PopDynCalcs.dat");
  //!! calclog <<"Population dynamics calculation log"<<endl;

  //!!   ofstream negLLLog("NegLLLog.dat");
  //!! negLLLog <<"Negative LL Log\n"<<endl;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
PARAMETER_SECTION

// Leading parameters
  init_bounded_number logFmsy(logFmsyLow,logFmsyHi,phz_Fmsy);
  init_bounded_number logMSY(logMSYLow,logMSYHi,phz_MSY);
```

```
 init_bounded_vector wt(syr,eyr-1,iRecdevLow,iRecdevHi,phz_Recdevs);     // Recruitment deviations //
added 31Mar2016 JOMU
 number Fmsy;
 number MSY;
 //number sd_wt;

 //establish calculated numbers
 likeprof_number So;                                                    //BH stock recruitment param
 number beta;                                                           //BH stock recruitment param
 number Ro;                                                             //virgin recruitment
 number Eo;                                                             //virgin eggs
 number juvSurv;                                                        //juveniles survival
 number Rmsy;                                                           //recruitment at msy
 number Bmsy;
 number reck;                                                           //compensation ratio
 likeprof_number Bo;                                                    //virgin biomass
 number phiE;
 number phiB;
 number reckPen;
 number FPen;

//  number cF;                                                          //Current F
//  number cB;                                                          //current Biomass

 vector histFmsy(1,numVaBlocks-1)
 vector histMSY(1,numVaBlocks-1)

 //Set up vector arrays to hold all the life history estimates
 vector age(0,Amax);

 //Set up vectors and matrices to hold the population dynamics
 matrix nt(syr,eyr+1,0,Amax);
 matrix nt_spawn(syr,eyr+1,0,Amax);
 matrix Fat(syr,eyr,0,Amax);
 matrix tbt(syr,eyr+1,0,Amax)  // total biomass at age
 sdreport_vector Ft(syr,eyr);
 matrix CAA(syr,eyr,0,Amax);
 //vector eggs(syr,eyr)
 matrix ssb(syr,eyr+1,0,Amax)  // ssb biomass at age
 vector SSB(syr,eyr)
 sdreport_vector Nt(syr,eyr);  // originally Nt(syr,eyr+1);
 sdreport_vector tBt(syr,eyr);  // Annnual Total biomass
 sdreport_vector Bt(syr,eyr);  // exploited biomass (originally Bt(syr,eyr+1));
 vector BtScaled(syr,eyr+1);                                    //scaled to mean of 1
```

```
 vector NtScaled(syr,eyr+1);                          //scaled to mean of 1
 3darray comppred(1,1+inum,syr,eyr,0,Amax)   //array to store the predicted prop at age
 matrix effN(1,1+inum,syr,eyr);                //effective sample size

 number tempsum;                                          //accumulator for like calcs
 number currYear;
 vector LI(1,inum);                                       //likelihood for indices
 vector LComp(1,inum+1);                                  //likelihood for comps
 vector LFObsPrior(1,numFObs);
 number LRecKPrior;
 number LFmsyPrior;
 number LMSYPrior;
 number LPen;

 number Lwt                                  // likelihood for recruitment deviations

 // stuffs for status  // added 31Mar2016 JOMU
 vector geoM_F(st_recent_yrs,eyr);
 vector geoM_VB(st_recent_yrs,eyr);
 number cur_F;
 number cur_VB;
 number Fratio;
 number Bratio

 objective_function_value negLL;

 //Set up leading parameters
LOCAL_CALCS
 logFmsy=log(iFmsy);
 logMSY=log(iMSY);
 //sd_wt = sqrt(sum(norm2(wt)));
END_CALCS

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
PRELIMINARY_CALCS_SECTION
 iterMCMC = 0;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
PROCEDURE_SECTION
//*********MAIN FUNCTION CALLS*************
 initial_calculations();
 if (testing==1) cout<<"Initial calculations completed"<<endl;

 get_leading_parms();
```

```
  if (testing==1) cout<<"Got leading parameters"<<endl;

  dynamic_model();
  if (testing==1) cout<<"Age structured population simulated"<<endl;

  calc_objective_function();
  if (testing==1) cout<<"Objective function minimized"<<endl;

  //code to output MCMC results
  if(mceval_phase()) MCMC_report();

  if (testing==1) {
    initial_report();
    cout<<"Completed procedure successfully"<<endl;
    exit(1);

  }

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
FUNCTION initial_calculations
  nt.initialize();
  LI.fill_seqadd(0.,0.); //set to zero
  LComp.fill_seqadd(0.,0.); //set to zero
  LFObsPrior.fill_seqadd(0.,0.);
  histFmsy.initialize();
  histMSY.initialize();
  LRecKPrior=0.;
  LFmsyPrior=0.;
  LMSYPrior=0.;
  negLL=0.0;
  reckPen=0.;
  FPen=0.;
  LPen=0.;
  iterMCMC=0;
  tempsum=0.0;
  Fmsy=mfexp(logFmsy);
  MSY=mfexp(logMSY);
  Bmsy = MSY/Fmsy;           // added 31Mar2016 JOMU
  Fratio = cur_F/Fmsy;       // added 31Mar2016 JOMU
  Bratio = (cur_VB*Fmsy)/MSY; // added 31Mar2016 JOMU

  curr_phase=current_phase() ;
```

```
  if (testing==1) cout<<"Fmsy: "<<Fmsy<<"\tMSY: "<<MSY<<endl;


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
FUNCTION get_leading_parms
  dvariable histFmsyTemp=0.;
  dvariable histMSYTemp=0.;

  //use end year/block vulnerability to get stock recruit params
  get_ro_reck(MSY,Fmsy,va(eyr),Ro,reck,reckPen);          //using Martell et al 2008 calcs

  So=reck/phiEo;                                          //B-H alpha (Max egg to age-0 survival rate)
  beta=(reck-1.)/(Ro*phiEo);                              //B-H beta
  Bo=Ro*phiBo;
  Eo=Ro*phiEo;
  juvSurv=Ro/Eo;

  if (testing==1) cout<<"Reck: "<<reck<<"\tSo: "<<So<<"\tbeta: "<<beta<<endl;

  //Get historic leading params -- if change in vulnerability over time, will change
  //  the FMSY/MSY, but assume the stock recruit parameters remain the same
  //  so need to back-calculate Fmsy/MSY from the stock recruit parameters
  for (int i=1; i<=numVaBlocks-1; i++) {
    get_CF(histFmsyTemp,histMSYTemp,va(vulMat(i,0)));
    histFmsy(i)=histFmsyTemp;
    histMSY(i)=histMSYTemp;
  }

  if (testing==1) cout<<"Historic FMSY: "<<histFmsy<<"\nHistoric MSY: "<<histMSY<<endl;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
FUNCTION dynamic_model
  Ft.initialize();
  // preliminary calcs
  nt(syr)=lxo*Ro;                    //virgin so lxo*Ro (Initial state)
  //nt_spawn(syr)=lxo_spawn*Ro;
  //  if (testing==1) ofstream finstout("FInstCalcs.dat");
  //  if (testing==1) !!finstout<<"F instantanous output"<<endl;

  //ofstream calclog("PopDynCalcs.dat",ios::app);
  //calclog<<"\n\n\nFmsy: "<<Fmsy<<"\tMSY: "<<MSY<<endl;
  //calclog<<"Reck: "<<reck<<"\tRo: "<<Ro<<endl;
  //calclog<<"alpha: "<<So<<"\tbeta: "<<beta<<endl;
  //calclog<<"lxo: "<<lxo<<endl;
```

```
  //calclog<<"Ro: "<<Ro<<endl;

  for(int i=syr; i<=eyr; i++)
  {
    currYear=i;  // may be commented out
    Ft(i)=get_F_instantaneous(Ct(i),va(i),morta,nt(i), wa,CAA(i));
    //calclog << "\nYear "<<i<<endl;
    //calclog << "Ft(y) "<<Ft(i)<<endl;
   //  dvariable eggs = sum(elem_prod(nt_spawn(i),fa));

    for (int j=0; j<Amax; j++){
      nt(i+1,j+1) = nt(i,j)*exp(-(morta(j)+Ft(i)*va(i,j)));
      nt_spawn(i,j) = nt(i,j)*exp(-spawn_offset*(morta(j)+Ft(i)*va(i,j)));
      tbt(i,j)= nt(i,j)*wa(j);
      ssb(i,j) = nt_spawn(i,j)*wa_spawn(j)*ma(j);
    }
    //calclog << "nt(y) "<<nt(i+1)<<endl;

    //dvariable E0=sum(elem_prod(nt(i),fa));                    // Wade C initially had this popn egg
production
   dvariable eggs = sum(elem_prod(nt_spawn(i),fa));                                    //popn egg
production NOTE: if fa = wgt @spawning * ma,
                                          // eggs is SSB
    //calclog << "E0 "<<E0<<endl;
    nt(i+1,0)=So*eggs/(1.+beta*eggs);
    //calclog << "nt(y,0) "<<nt(i+1,0)<<endl;
    if(i<eyr) {
     nt(i+1,0)*=exp(wt(i)-0.5*sd_wt);                 //Adding in recruit deviations
    }
    if(nt(i+1,0)<0) {
      nt(i+1,0)=0;
      tbt(i+1,0)=0;
     }
 //Explicit calculation of F @ age
  for(int y=syr; y<=eyr; y++)
  {
    for (int a=0; a<=Amax; a++)
        {
         Fat(y,a) = Ft(y)*va(y,a);
         }
  }

    Nt(i)=sum(nt(i));
    //calclog << "Nt(y) "<<Nt<<endl;
```

```
   tBt(i)= sum(tbt(i));  //Total biomass;
   Bt(i)=elem_prod(va(i),wa)*nt(i);  //vulnerable biomass; equ to
sum(elem_prod(elem_prod(va,wa),nt(i)))
   SSB(i) = sum(ssb(i));
   //calclog << "Bt(y) "<<Bt<<endl;
  }
   //cout <<"nt =" << SSB << "\n";
   //exit(100);


  //calculate predicted proportion at age (in numbers as per observed comp data)
  for (int y=syr; y<=eyr; y++){
   for (int a=0; a<=Amax; a++) {
    comppred(1,y,a)=CAA(y,a)/sum(CAA(y)); //for fishery
    for (int i=2; i<=inum+1; i++) {
     comppred(i,y,a)=(nt(y,a)* ItSel(i-1,y,a))/(sum(elem_prod(nt(y),ItSel(i-1,y))));  //for indices
    }
   }
  }

// Stuffs for stock status  init_int // added 31Mar2016 JOMU

  for(int i=st_recent_yrs; i<=eyr; i++)
    {
    if (i==st_recent_yrs)
       {geoM_VB(i) =  pow(Bt(i),1.0/n_recent_yrs);
        geoM_F(i)  =  pow(Ft(i),1.0/n_recent_yrs);
       }
    else if (i>st_recent_yrs)
       {
       geoM_VB(i)=  pow(Bt(i),1.0/n_recent_yrs)*geoM_VB(i-1);
       geoM_F(i) =  pow(Ft(i),1.0/n_recent_yrs)*geoM_F(i-1);
       }
    }
  cur_F  = geoM_F(eyr);      //This is current F = geometric mean of F over the most recent years
  cur_VB = geoM_VB(eyr);      //This is current vulnerable biomass = geometric mean of VB over the most
recent years

   // cout <<"F =" << Ft << "\n";
   // cout <<"geoM_F =" << geoM_F << "\n";
   // cout <<"B =" << Bt << "\n";
   // cout <<"geoM_VB =" << geoM_VB << "\n";
   // cout <<"Current VB =" << cur_VB << "\n";
   // cout <<"Current F =" << cur_F << "\n";
```

```
  // exit(100);

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
FUNCTION calc_objective_function

 double pi=3.141593;
 //ofstream negLLLog("NegLLLog.dat",ios::app);
 int count=0;

 //fit indices
 //scale the indices by index specific selectivity and to a mean of 1
 // NOTE THAT THE PREDICTED POPULATION INDICES (i.e. pertaining to index-specific selectivity).
 // BY SCALING THEM TO THEIR MEANS, THEIR MAGNITUDSES BECOME COMPARABLE TO THE
OBSERVED RELATIVE
 // in other words, BtScaled and NtScaled are the predicted indices.
 for (int i=1; i<=inum; i++) {
  for (int y=isyr(i); y<=ieyr(i); y++){
   if (itype(i)==1) {  //biomass index
    BtScaled=(elem_prod(wa,ItSel(i,y))*trans(nt))/mean(elem_prod(wa,ItSel(i,y))*trans(nt));
    // LI(i)+=lamIndices(i)*(0.5*log(2.*pi)+0.5*log(ItVar(i,y))+log(It(i,y))+square(log(It(i,y)+.000001)-
log(BtScaled(y)+.000001))/(2*ItVar(i,y)));
     LI(i)+=lamIndices(i)*(0.5*log(2.*pi)+0.5*log(ItVar(i,y))+square(log(It(i,y)+.000001)-
log(BtScaled(y)+.000001))/(2*ItVar(i,y)));

   }
   if (itype(i)==2) {  //numbers index
    NtScaled=(ItSel(i,y)*trans(nt))/mean(ItSel(i,y)*trans(nt));
    // LI(i)+=lamIndices(i)*(0.5*log(2.*pi)+0.5*log(ItVar(i,y))+log(It(i,y))+square(log(It(i,y)+.000001)-
log(NtScaled(y)+.000001))/(2*ItVar(i,y)));
     LI(i)+=lamIndices(i)*(0.5*log(2.*pi)+0.5*log(ItVar(i,y))+square(log(It(i,y)+.000001)-log(
NtScaled(y)+.000001))/(2*ItVar(i,y)));
   }
   if(isnan(value(LI(i))) | isinf(value(LI(i))) | isinf(-1*value(LI(i)))){
        cout<<LI(i)<<" in indices"<<endl;
        cout << "trans(nt):\n"<< trans(nt) << endl;
    cout << "ItSel(i):\n"<< ItSel(i) << endl;
    cout << "ItSel(i,y):\n"<< ItSel(i,y) << endl;
    cout << "NtScaled:\n"<< NtScaled << endl;
    cout <<"\n\n\nFmsy: "<<Fmsy<<"\tMSY: "<<MSY<<endl;
                cout <<"Reck: "<<reck<<"\tRo: "<<Ro<<endl;
                cout <<"alpha: "<<So<<"\tbeta: "<<beta<<endl;
                cout <<"lxo: "<<lxo<<endl;
                cout <<"Ro: "<<Ro<<endl;
```

```
            exit(1);
     }

   }
   //cout << "LI(" << i<< ") "<<LI(i)<<endl;
   //exit(100);
  }

  //fit composition data
  for (int y=syr; y<=eyr; y++){
   for (int i=1; i<=inum+1; i++) {  //here, this indexes "fleet" where i=1 is catch fleet, i=2+ are indices
    tempsum=0.0;
    for (int a=0; a<=Amax; a++) {
     if (compobs(i,y,a)>0 && comppred(i,y,a)>0) {
      tempsum+=compobs(i,y,a)*log(comppred(i,y,a));
     }
    }
    LComp(i)+=lamComps(i)*(-1*sampN(i,y)*tempsum);
    //calculate effective sample size in same loop
    effN(i,y)=(comppred(i,y)*(1.0-comppred(i,y)))/norm2(compobs(i,y)-comppred(i,y));
    if(isnan(value(LComp(i))) | isinf(value(LComp(i))) | isinf(-1*value(LComp(i)))){ cout<<LComp(i)<<" in
comp"<<endl; exit(1); }
   }
  }

   //for (int i=1; i<=inum+1; i++) {  //here, this indexes "fleet" where i=1 is catch fleet, i=2+ are indices
    // negLLLog << "LComp(" << i<< ") "<<LComp(i)<<endl;
           //}

  //add Fmsy prior
//  if(iFmsyPriorCV>=0)
LFmsyPrior+=lamFmsyPr*(0.5*log(2.*pi)+0.5*log(iFmsyPriorVar)+log(iFmsyPrior)+square(log(iFmsyPrior
+0.000001)-log(Fmsy+0.000001))/(2*iFmsyPriorVar));
 if(iFmsyPriorCV>=0)
LFmsyPrior+=lamFmsyPr*(0.5*log(2.*pi)+0.5*log(iFmsyPriorVar)+square(log(iFmsyPrior+0.000001)-
log(Fmsy+0.000001))/(2*iFmsyPriorVar));

  if(isnan(value(LFmsyPrior)) | isinf(value(LFmsyPrior)) | isinf(-1*value(LFmsyPrior))){
cout<<LFmsyPrior<<" in LFmsyPrior"<<endl; exit(1); }
  //if(iFmsyPriorCV>=0) negLLLog << "LFmsyPrior "<<LFmsyPrior<<endl;

  //add MSY prior
```

```
 // if(iMSYPriorCV>=0)
LMSYPrior+=lamMSYPr*(0.5*log(2.*pi)+0.5*log(iMSYPriorVar)+log(iMSYPrior)+square(log(iMSYPrior+0.0
00001)-log(MSY+0.000001))/(2*iMSYPriorVar));
   if(iMSYPriorCV>=0)
LMSYPrior+=lamMSYPr*(0.5*log(2.*pi)+0.5*log(iMSYPriorVar)+square(log(iMSYPrior+0.000001)-
log(MSY+0.000001))/(2*iMSYPriorVar));
  if(isnan(value(LMSYPrior)) | isinf(value(LMSYPrior)) | isinf(-1*value(LMSYPrior))){ cout<<LMSYPrior<<"
in LMSYPrior"<<endl; exit(1); }
  //if(iMSYPriorCV>=0) negLLLog << "LMSYPrior "<<LMSYPrior<<endl;

  //add reck prior - do as normal distribution since it goes negative (LPen will remove negatives)
  if(ireckCV>=0) LRecKPrior+=lamRecKPr*(0.5*log(2.*pi)+0.5*log(ireckVar)+square(ireck-
reck)/(2*ireckVar));
  if(isnan(value(LRecKPrior)) | isinf(value(LRecKPrior)) | isinf(-1*value(LRecKPrior))){
cout<<LRecKPrior<<" in reck"<<endl; exit(1); }
  //if(ireckCV>=0) negLLLog << "LRecKPrior "<<LRecKPrior<<endl;

  //add in F-observations prior
  for (int i=1; i<=numFObs; i++) {
   if(FObsCV(i)>=0)
LFObsPrior(i)+=lamFObs(i)*(0.5*log(2.*pi)+0.5*log(FObsVar(i))+log(FObsPrior(i))+square(log(FObsPrior(i)
+.000001)-log(Ft(FObsYear(i))+.000001))/(2*FObsVar(i)));
   if(isnan(value(LFObsPrior(i))) | isinf(value(LFObsPrior(i))) | isinf(-1*value(LFObsPrior(i)))){
cout<<LFObsPrior(i)<<" in LFObsPrior"<<endl; exit(1); }
   //if(FObsCV(i)>=0) negLLLog << "FObsCV(" << i << ") "<<LFObsPrior(i)<<endl;
  }

  //add penalty for reck being negative
  LPen+=(nyr-1)*(reckPen+FPen)*1000;
  if(isnan(value(LPen)) | isinf(value(LPen)) | isinf(-1*value(LPen))){ cout<<LPen<<" in upen"<<endl;
exit(1); }
  //negLLLog << "LPen "<<LPen<<endl;

  // Add likelihood for recruitment deviations
  Lwt=nyr*norm2(wt/(2.0*sd_wt));

  negLL+=sum(LI)+sum(LComp)+LFmsyPrior+LMSYPrior+LRecKPrior+sum(LFObsPrior)+LPen+Lwt;
  //negLLLog << "negLL "<<negLL<<"\n"<<endl;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
//Martell code (meanage.tpl) from Martell et al. (2008)
FUNCTION dvariable get_F_instantaneous(const double& Ct,const dvar_vector& vul,const dvar_vector&
m,const dvar_vector& Nj,const dvar_vector& Wj,dvar_vector& Cij)
```

```
 //  if (testing==1) ofstream finstout("FInstCalcs.dat",ios::app);
 //  if (testing==1) finstout<<"\n\nEntering F-inst fnx for year "<<currYear<<endl;
 //  if (testing==1) finstout<<"Ct: "<<Ct <<endl;
 //  if (testing==1) finstout<<"vul: "<<vul <<endl;
 //  if (testing==1) finstout<<"m: "<<m <<endl;
 //  if (testing==1) finstout<<"Nj: "<<Nj <<endl;
 //  if (testing==1) finstout<<"Wj: "<<Wj <<endl;

 int a,A;
 double minsurv=0.01;
 double step=1.;
 a=Nj.indexmin(); A=Nj.indexmax();
 dvariable fest,pCt,diffCt;  // pCt = predicted catch; fest = estimated F
 dvar_vector fage(a,A);
 dvar_vector zage(a,A);
 dvar_vector ominus(a,A);
 dvar_vector bage(a,A);

 //  bage=elem_prod(elem_prod(Nj,Wj),vul);  //this is how Steve had it - vulnerable biomass per recruit
used in baranov, don't think appropriate
 bage=elem_prod(Nj,Wj); //wtc - change to just biomass at age for baranov
 //Use below if catch is in numbers
 //bage=elem_prod(Nj,vul);
 fest=Ct/(0.98*sum(bage));                    //initial guess for F
 dvariable ctmp=Ct;
 //  if (testing==1) finstout<<"Initial fest: "<<fest <<endl;
 //  if (testing==1) finstout<<"sum(bage): "<<sum(bage)<<endl;
 //  if (testing==1) finstout<<"bage: "<<bage<<endl;

 if((1.-fest)<minsurv)
 {
  fest=1.-posfun((1.-fest),minsurv,0.001*FPen);
  ctmp=fest*0.98*sum(bage);
 //  if (testing==1) finstout<<"Kludge for get F \t"<<fest<<"\t"<<Ct<<"\t"<<sum(bage)<<endl;
 }

 for(int iter=1; iter<=10; iter++)
 {
  fage=fest*vul;
  zage=fage+m;
  ominus=(1.-mfexp(-zage));
  //pCt=fage/zage*ominus*Nt*Wj;
  pCt=sum(elem_prod(elem_prod(elem_div(fage,zage),ominus),bage));
```

```
   //derivative of the catch equation
   diffCt=sum(elem_div(elem_prod(bage,ominus),zage)
      -elem_div(elem_prod(elem_prod(bage,fage),ominus),square(zage))
      +elem_div(elem_prod(elem_prod(bage,fage),mfexp(-zage)),zage));

   //fest+=(ctmp-pCt)/diffCt;
   fest-=step*(pCt-ctmp)/diffCt;                    //Newton-Raphson update
   //    if (testing==1) finstout<<"fest iteration "<<iter<<": "<<fest <<endl;

   if(ctmp-pCt<=1.e-4) break;
  }
  //  if (testing==1) finstout<<"F est: "<<fest<<endl;

  //cout<<ctmp-pCt<<endl;
  Cij=elem_prod(elem_prod(elem_div(fage,zage),ominus),Nj);
  //Cij/=sum(Cij);
  //  if (testing==1) finstout<<"Cij: "<<Cij <<endl;
  //  if (testing==1) finstout<<"Exiting F-inst fnx\n\n"<<endl;

  return(fest);

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
//Martell code (meanage.tpl) from Martell et al. (2008)
FUNCTION void get_ro_reck(const dvariable& ce,const dvariable& fe,const dvar_vector& vul,dvariable&
ro,dvariable& reck,dvariable& fpen)
  //Use this function to calculate the partial derivatives (Table 2 in Martell et al. 2007)
  //and return estimates of bo and reck conditional on theta.
  //Arguments: fe=fmsy, ce=MSY

  int i;
  dvariable phie,phif,phiq,dphif_df=0.,dphiq_df=0.,dlz_df=0.,fpen1=0.,fpen2=0.;
     //dvar_vector lx=pow(exp(-m),age-1.);
     dvar_vector lz(0,Amax);
     dvar_vector za=(morta+fe*vul);
     dvar_vector sa=1.-exp(-za);
     dvar_vector qa=elem_prod(elem_div(vul,za),sa);

     if (testing==1) cout << "sa "<<sa<<endl;
     if (testing==1) cout << "qa "<<qa<<endl;
     if (testing==1) cout << "vul "<<vul<<endl;

     //dvar_vector dlz_df(1,nage);
     lz(0)=1.0; dlz_df=0.;
```

```
    phie=sum(elem_prod(lxo,fa));
    //phib=sum(elem_prod(elem_prod(lxo,wa),vul));
    for(i=0; i<=Amax; i++)
    {
     if(i>0) lz(i)=lz(i-1)*exp(-za(i-1));
     if(i>0) dlz_df=dlz_df*exp(-za(i-1)) - lz(i-1)*vul(i-1)*exp(-za(i-1));  //From Steve's code
     //       if(i>0) dlz_df=dlz_df*exp(-za(i-1)) - lz(i-1)*vul(i-1)*exp(-morta(i-1));  //From T Miller's code, R.
Forrest code
     dphif_df=dphif_df+fa(i)*dlz_df;
     dphiq_df=dphiq_df+wa(i)*qa(i)*dlz_df+(lz(i)*wa(i)*vul(i))/za(i)*(exp(-za(i))-sa(i)/za(i));
    }

    phif=sum(elem_prod(lz,fa));
    phiq=sum(elem_prod(elem_prod(lz,wa),qa));


    //dvar_vector t2=elem_div(elem_prod(elem_prod(lz,vul),wa),za);
    //dvar_vector t3=exp(-za)-elem_div(sa,za);
    //dphiq_df=sum(elem_prod(elem_prod(wa,qa),dlz_df)+elem_prod(t2,t3));
    reck=phie/phif-(fe*phiq*phie/square(phif)*dphif_df)/(phiq+fe*dphiq_df);
    if (testing==1) cout << "reck (pre-pen) "<<reck<<endl;
    reck=posfun(reck,1.01,fpen1);  //Minimum
    if (testing==1) cout << "reck (post-pen) "<<reck<<endl;

    //reck=h-(fmsy*phiq*phie/square(phif)*df_du)/(phiq+fmsy*dq_du);
    dvariable re=ce/(fe*phiq);
    ro=re*(reck-1.)/(reck-phie/phif);
    ro=posfun(ro,.001,fpen2);  //Minimum
          fpen=fpen1+fpen2;

    if (testing==1) cout << "re "<<re<<endl;
    if (testing==1) cout << "ro "<<ro<<endl;

    if (testing==1) cout << "phie "<<phie<<endl;
    if (testing==1) cout << "phif "<<phif<<endl;
    if (testing==1) cout << "phiq "<<phiq<<endl;
    if (testing==1) cout << "Rmsy "<<re<<endl;

//bo=ro*phib;
//cout<<"reck & Bo\t"<<reck<<"\t"<<bo<<endl;
//SSBe=value(re*sum(elem_prod(elem_prod(lz,wa),ma))/2.);

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
//Martell code (meanage.tpl) from Martell et al. (2008)
```

```
FUNCTION void calc_partials(const dvariable& fe,const dvar_vector& vul,dvariable& phie,dvariable&
phif,dvariable& phiq,dvariable& dphif_df,dvariable& dphiq_df,dvariable& dRe_df)
 //Use this function to calculate the partial derivatives (Table 2 in Martell et al. 2007)
 //Arguments: fe=fishing rate
 //if (testing==1) cout<<"Beginning calc_partials"<<endl;
 int i;
 //dvector lx=value(pow(exp(-m),age-1.));
 dvector lz(0,Amax);
 dvector za=value(morta+fe*vul);
 dvector sa=1.-exp(-za);
 dvector qa=elem_prod(elem_div(value(vul),za),sa);
 dvariable dlz_df=0;

 lz(0)=1.0;
 dphiq_df=0; dphif_df=0;
 phie=sum(elem_prod(lxo,fa));
 for(i=0; i<=Amax; i++)
 {
  if(i>0) lz(i)=lz(i-1)*exp(-za(i-1));
  if(i>0) dlz_df=dlz_df*exp(-za(i-1)) - lz(i-1)*vul(i-1)*exp(-za(i-1));  //From Steve's code
  dphif_df=dphif_df+fa(i)*dlz_df;
  dphiq_df=dphiq_df+wa(i)*qa(i)*dlz_df+(lz(i)*wa(i)*vul(i))/za(i)*(exp(-za(i))-sa(i)/za(i));
 }
 phif=sum(elem_prod(lz,fa));
 phiq=sum(elem_prod(elem_prod(lz,wa),qa));
 dRe_df=value(Ro/(reck-1.))*phie/square(phif)*dphif_df;
 //if (testing==1) cout<<"End calc_partials"<<endl;

 if (testing==1) cout << "phie "<<phie<<endl;
 if (testing==1) cout << "phif "<<phif<<endl;
 if (testing==1) cout << "phiq "<<phiq<<endl;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
//Martell code (meanage.tpl) from Martell et al. (2008)
FUNCTION void get_CF(dvariable& fe,dvariable& msy,const dvar_vector& vul)
 //This function uses Newton-Raphson method to iteratively solve for F*
 //Then calculates C* given F* (See eq 1.3 in Martell 2007)
 int iter;
 dvariable dy,ddy,re;
 dvariable phie,phif,phiq,dphif_df,dphiq_df,dRe_df;
 fe=mean(morta);

 for(iter= 1; iter<=50; iter++)
 {
```

```
   calc_partials(fe,vul,phie,phif,phiq,dphif_df,dphiq_df,dRe_df);
   re=Ro*(reck-phie/phif)/(reck-1.);
   if (testing==1) cout<<"Rmsy (iteration" <<iter<<") "<<re<<endl;
   dy=re*phiq+fe*phiq*dRe_df+fe*re*dphiq_df;
   ddy=phiq*dRe_df+re*dphiq_df;
   //Newton update
   fe=fe-dy/ddy;
   if(dy<1.e-5) {
    //cout<<"Breaking after "<<iter<<" iterations"<<endl;
    break;

   }
  //cout<<"Fe dy\t"<<fe<<" "<<dy<<" "<<fe-dy/ddy<<endl;
  }
  msy=fe*re*phiq;

//cout<<"Fe "<<fe<<endl;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
FUNCTION MCMC_report

  //For some ADMB-reason, this will not let me do a header line
  // on first iteration and then append -- only outputs one line
  //As such, am doing things ackwardly here, printing out headers first as separate file
  //if(iterMCMC==0) {

    //Note: something goofy with my admb install, I can append to the document if I first
    // create it new as per this normal way of adding a header line to MCMC output.  It will only
    //  append one time at a time, so need to output the headers to a seperate file and then
    //  splice back in.
    //    ofstream ofest("mcmc_results.mcmc");
    //    ofest << "negLL";
    //    for (int y=syr; y<=eyr; y++) ofest << "\tF"<<y;
    //    for (int y=syr; y<=eyr; y++) ofest << "\tN"<<y;
    //    ofest << endl;
    //
    //    ofstream ofpar("mcmc_par.mcmc");
    //    ofpar<<"Fmsy\tMSY\tSo\tbeta\tRo\tBo\tEo"<<endl;

    //ofstream mcmcResultsHead("mcmc_results_head.mcmc");
    //mcmcResultsHead << "negLL";
    //for (int y=syr; y<=eyr; y++) mcmcResultsHead << "\tF"<<y;
    //for (int y=syr; y<=eyr; y++) mcmcResultsHead << "\tN"<<y;
    //for (int y=syr; y<=eyr; y++) mcmcResultsHead << "\tB"<<y;
```

```
    //for (int y=syr; y<=eyr; y++) mcmcResultsHead << "\tBmsy"<<y;
  //for (int y=syr; y<=eyr; y++) mcmcResultsHead << "\tBt"<<y;
  //mcmcResultsHead << endl;

  //ofstream mcmcParHead("mcmc_par_head.mcmc");
  //mcmcParHead<<"Fmsy\tMSY\tBmsy\tSo\tbeta\tRo\tReck\th\tBo\tEo"<<endl;

  //iterMCMC++;
 //}
  iterMCMC++ ;                          // modified, added 25Mar2016 JRO
  basicMCMC << negLL        << "\t"
          << Ft          << "\t"
          << Nt          << "\t"
          << tBt         << "\t"
          << Bt          << "\t"
          << SSB         << "\t"
          << Fmsy        << "\t"
          << MSY         << "\t"
          << Bmsy        << "\t"
          << So          << "\t"
          << beta        << "\t"
          << Ro          << "\t"
          << reck        << "\t"
          << reck/(reck+4) << "\t"
          << Bo          << "\t"
          << Eo          << "\t"
          << cur_F       << "\t"          // added 31Mar2016 JOMU
          << cur_VB      << "\t"          // added 31Mar2016 JOMU
          << Fratio      << "\t"          // added 31Mar2016 JOMU
          << Bratio      << "\t"          // added 31Mar2016 JOMU
          << endl ;
          ;

 //ofstream ofest("mcmc_results.mcmc",ios::app);
 //ofest << negLL;
 //for (int y=syr; y<=eyr; y++) ofest << "\t"<<Ft(y);
 //for (int y=syr; y<=eyr; y++) ofest << "\t"<<Nt(y);
 //for (int y=syr; y<=eyr; y++) ofest << "\t"<<tBt(y);
 //for (int y=syr; y<=eyr; y++) ofest << "\t"<<Bt(y);
 //ofest << endl;

 //ofstream ofpar("mcmc_par.mcmc",ios::app);
```

```
//ofpar<<Fmsy<<"\t"<<MSY<<"\t"<<Bmsy<<"\t"<<So<<"\t"<<beta<<"\t"<<Ro<<"\t"<<reck<<"\t"<<reck/
(reck+4)<<"\t"<<Bo<<"\t"<<Eo<<endl;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
REPORT_SECTION

 report<<"negLL\n"<<negLL<<endl;
 report<<"\nLI\n"<<LI<<endl;
 report<<"\nLComp\n"<<LComp<<endl;
 report<<"\nLFmsyPrior\n"<<LFmsyPrior<<endl;
 report<<"\nLMSYPrior\n"<<LMSYPrior<<endl;
 report<<"\nLRecKPrior\n"<<LRecKPrior<<endl;
 report<<"\nLFObsPrior\n"<<LFObsPrior<<endl;
 report<<"\nLPen\n"<<LPen<<endl;
 report<<"\nrecAnomalies\n"<<Lwt<<endl;
 report<<"\nFMSY\n"<<Fmsy<<endl;
 report<<"\nHistoric_FMSY\n"<<histFmsy<<endl;
 report<<"\nMSY\n"<<MSY<<endl;
 report<<"\nBmsy\n"<<Bmsy<<endl;
 report<<"\ncur_F\n"<<cur_F<<endl;
 report<<"\ncur_VB\n"<<cur_VB<<endl;
 report<<"\nFratio\n"<<Fratio<<endl;
 report<<"\nBratio\n"<<Bratio<<endl;
 report<<"\nHistoric_MSY\n"<<histMSY<<endl;
 report<<"\nRo\n"<<Ro<<endl;
 report<<"\nCompensation_ratio\n"<<reck<<endl;
 report<<"\nSteepness\n"<<reck/(reck+4)<<endl;
 report<<"\nEo\n"<<Eo<<endl;
 report<<"\nBo\n"<<Bo<<endl;
 report<<"\nB-H_alpha\n"<<So<<endl;
 report<<"\nB-H_beta\n"<<beta<<endl;
 report<<"\nJuvenive_Survival\n"<<juvSurv<<endl;
// report<<"\Egg production\n"<<eggs<<endl;

 report<<"\nTime_Series"<<endl;
 report<<"Year\tNum\tBiomass\tVulBiomass\tSSB\tCatch\tFRate";
 for (int i=1; i<=inum; i++) report<<"\tIndexObs"<<i<<"\tIndexPred"<<i;
 report<<endl;
 for (int y=syr; y<=eyr; y++) {
   report<<y<<"\t"<<Nt(y)<<"\t"<<tBt(y)<<"\t"<<Bt(y)<<"\t"<<SSB(y)<<"\t"<<Ct(y)<<"\t"<<Ft(y);
   for (int i=1; i<=inum; i++) {
     if (y>=isyr(i) & y<=ieyr(i)) {
       if (itype(i)==1) {  //biomass index
```

```
      BtScaled=(elem_prod(wa,ItSel(i,y))*trans(nt))/mean(elem_prod(wa,ItSel(i,y))*trans(nt));
      report<<"\t"<<It(i,y)<<"\t"<<BtScaled(y);
     }
     else if (itype(i)==2){
      NtScaled=(ItSel(i,y)*trans(nt))/mean(ItSel(i,y)*trans(nt));
      report<<"\t"<<It(i,y)<<"\t"<<NtScaled(y);
     }
    }
    else report<<"\tNA"<<"\tNA";
  }
  report<<endl;

 }

 report<<"\nEffective_Sample_Sizes"<<endl;
 report<<"Series\tYear\tEffN"<<endl;
 for (int i=1; i<=inum+1; i++) {
  for (int y=syr; y<=eyr; y++) {
   if (sum(compobs(i,y))>0 && sum(comppred(i,y))>0) {
    if (i==1) report<<"Fleet"<<i<<"\t"<<y<<"\t"<< effN(i,y)<<endl;
    else if (i>1) report<<"Index"<<i-1<<"\t"<<y<<"\t"<< effN(i,y)<<endl;

   }

  }
 }

 dvar_vector age(0,Amax);
 age.fill_seqadd(1,1);

 report<<"\nPredicted_Numbers_at_Age"<<endl;
 report<<"Year\t"<<age<<endl;
 for (int y=syr; y<=eyr; y++) {
  report<<y;
  for (int a=0; a<=Amax; a++) {
   report<<"\t"<<nt(y,a);
  }
  report<<endl;
 }

 report<<"\nPredicted_Catch_at_Age"<<endl;
 report<<"Year\t"<<age<<endl;
 for (int y=syr; y<=eyr; y++) {
  report<<y;
```

```
   for (int a=0; a<=Amax; a++) {
     report<<"\t"<<CAA(y,a);
   }
   report<<endl;
  }

  report<<"\nPredicted_Fishing mortality_at_Age"<<endl;
  report<<"Year\t"<<age<<endl;
  for (int y=syr; y<=eyr; y++) {
   report<<y;
   for (int a=0; a<=Amax; a++) {
     report<<"\t"<<Fat(y,a);
   }
   report<<endl;
  }
FUNCTION initial_report

  ofstream ofs_gen("InitialReport.dat");
  ofs_gen<<"negLL\n"<<negLL<<endl;
  ofs_gen<<"\nLI\n"<<LI<<endl;
  ofs_gen<<"\nLComp\n"<<LComp<<endl;
  ofs_gen<<"\nLFmsyPrior\n"<<LFmsyPrior<<endl;
  ofs_gen<<"\nLMSYPrior\n"<<LMSYPrior<<endl;
  ofs_gen<<"\nLRecKPrior\n"<<LRecKPrior<<endl;
  ofs_gen<<"\nLFObsPrior\n"<<LFObsPrior<<endl;
  ofs_gen<<"\nLPen\n"<<LPen<<endl;
  ofs_gen<<"\nFMSY\n"<<Fmsy<<endl;
  ofs_gen<<"\nHistoric_FMSY\n"<<histFmsy<<endl;
  ofs_gen<<"\nMSY\n"<<MSY<<endl;
  ofs_gen<<"\nHistoric_MSY\n"<<histMSY<<endl;
  ofs_gen<<"\nRo\n"<<Ro<<endl;
  ofs_gen<<"\nCompensation_ratio\n"<<reck<<endl;
  ofs_gen<<"\nSteepness\n"<<reck/(reck+4)<<endl;
  ofs_gen<<"\nEo\n"<<Eo<<endl;
  ofs_gen<<"\nBo\n"<<Bo<<endl;
  ofs_gen<<"\nB-H_alpha\n"<<So<<endl;
  ofs_gen<<"\nB-H_beta\n"<<beta<<endl;
  ofs_gen<<"\nJuvenive_Survival\n"<<juvSurv<<endl;

  ofs_gen<<"\nTime_Series"<<endl;
  ofs_gen<<"Year\tNum\tBiomass\tVulBiomass\tCatch\tFRate";
  for (int i=1; i<=inum; i++) ofs_gen<<"\tIndexObs"<<i<<"\tIndexPred"<<i;
  ofs_gen<<endl;
  for (int y=syr; y<=eyr; y++) {
```

```
  ofs_gen<<y<<"\t"<<Nt(y)<<"\t"<<Bt(y)<<"\t"<<Ct(y)<<"\t"<<Ft(y);
  for (int i=1; i<=inum; i++) {
   if (y>=isyr(i) & y<=ieyr(i)) {
    if (itype(i)==1) {  //biomass index
     BtScaled=(elem_prod(wa,ItSel(i,y))*trans(nt))/mean(elem_prod(wa,ItSel(i,y))*trans(nt));
     ofs_gen<<"\t"<<It(i,y)<<"\t"<<BtScaled(y);
    }
    else if (itype(i)==2){
     NtScaled=(ItSel(i,y)*trans(nt))/mean(ItSel(i,y)*trans(nt));
     ofs_gen<<"\t"<<It(i,y)<<"\t"<<NtScaled(y);
    }
   }
   else ofs_gen<<"\tNA"<<"\tNA";
  }
  ofs_gen<<endl;

 }

 ofs_gen<<"\nEffective_Sample_Sizes"<<endl;
 ofs_gen<<"Series\tYear\tEffN"<<endl;
 for (int i=1; i<=inum+1; i++) {
  for (int y=syr; y<=eyr; y++) {
   if (sum(compobs(i,y))>0 && sum(comppred(i,y))>0) {
    if (i==1) ofs_gen<<"Fleet"<<i<<"\t"<<y<<"\t"<< effN(i,y)<<endl;
    else if (i>1) ofs_gen<<"Index"<<i-1<<"\t"<<y<<"\t"<< effN(i,y)<<endl;

   }

  }
 }

 dvar_vector age(0,Amax);
 age.fill_seqadd(1,1);

 ofs_gen<<"\nPredicted_Numbers_at_Age"<<endl;
 ofs_gen<<"Year\t"<<age<<endl;
 for (int y=syr; y<=eyr; y++) {
  ofs_gen<<y;
  for (int a=0; a<=Amax; a++) {
   ofs_gen<<"\t"<<nt(y,a);
  }
  ofs_gen<<endl;
 }
```

```
 ofs_gen<<"\nPredicted_Catch_at_Age"<<endl;
 ofs_gen<<"Year\t"<<age<<endl;
 for (int y=syr; y<=eyr; y++) {
  ofs_gen<<y;
  for (int a=0; a<=Amax; a++) {
    ofs_gen<<"\t"<<CAA(y,a);
  }
  ofs_gen<<endl;
 }

 ofs_gen<<"\nPredicted_Fishing mortality_at_Age"<<endl;
 ofs_gen<<"Year\t"<<age<<endl;
 for (int y=syr; y<=eyr; y++) {
  ofs_gen<<y;
  for (int a=0; a<=Amax; a++) {
    ofs_gen<<"\t"<<Fat(y,a);
  }
  ofs_gen<<endl;
 }
```

Control file: SSRA.dat

four_ndx.dat
#ENP_SEsurvey_1975_2014.dat
#ENP_MRFSS_SEsurvey_1975_2014.dat
#ENP_MRFSS_SEsurvey_1975_2014_rev.dat

Data file: four_ndx.dat

#Testing: 0=off, 1=on (if on, will run at initial values and exit after one iteration)
0

#####################################
Model configuration
#####################################

Model start/stop and catch data
1950 2014

#####################################
Life History
#####################################

#Maximum age (includes age 0 so total entries are MaxAge+1)
37

#Spawning offset (i.e., fraction of the year at the beginning of the month of peak spawning: 0: jan1, 1 Dec.1, some fraction in between)
0.67

#Read in vectors directly so can be unique growth functions
Length at age (mm)
137.92 324.24 493.89 648.37 789.03 917.12 1033.74 1139.93 1236.63
1324.67 1404.85 1477.85 1544.32 1604.84 1659.95
1710.14 1755.83 1797.44 1835.32 1869.82 1901.23
1929.83 1955.87 1979.59 2001.18 2020.84 2038.75
2055.05 2069.89 2083.41 2095.71 2106.92 2117.12
2126.41 2134.87 2142.58 2149.59 2155.98

Weight at age (kg) - Jan 1
0.04 0.58 2.13 4.94 9.06 14.41 20.86 28.23 36.30 44.89 53.83 62.96 72.12
81.22 90.15 98.84 107.23 115.28 122.96 130.24 137.12 143.59 149.67 155.34 160.64
165.57 170.14 174.38 178.30 181.92 185.27 188.34 191.18 193.78 196.17 198.37 200.38
202.23

```
# Weight at age (kg) - peak of spawning (here, August)
0.30977768    1.486258885    3.864523961    7.553662684    12.51627875    18.62579569
        25.70772213    33.5680738    42.01212263    50.85610434    59.93393387    69.10047317
        78.23248938    87.22812672    96.00547727    104.5006584    112.6656743    120.4662457
        127.8797224    134.8931489    141.5015175    147.7062241    153.5137248    158.934384
        163.9815003    168.6704913    173.0182194    177.0424415    180.7613646    184.1932926
        187.3563486    190.2682628    192.9462125    195.4067067    197.665506    199.7375717
        201.6370379    203.3772018

# Maturity at age
0       0       0       0       0       0       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1       1       1       1       1
        1       1       1       1       1       1       1       1       1       1       1       1
        1

# Fecundity at age (Eggs)  - Here these are weight@age at peak spawning
0       0       0       0       0       0       25.70772213    33.5680738    42.01212263
        50.85610434    59.93393387    69.10047317    78.23248938    87.22812672    96.00547727
        104.5006584    112.6656743    120.4662457    127.8797224    134.8931489    141.5015175
        147.7062241    153.5137248    158.934384    163.9815003    168.6704913    173.0182194
        177.0424415    180.7613646    184.1932926    187.3563486    190.2682628    192.9462125
        195.4067067    197.665506    199.7375717    201.6370379    203.3772018

# Mortality at age
1.641685257    0.767212948    0.527554098    0.414081373    0.34769687    0.304133385
        0.27340059    0.250614497    0.233098161    0.219258752    0.208087852    0.198915225
        0.191277613    0.18484432    0.179372804    0.174681043    0.1706297    0.167110262
        0.164036933    0.161340971    0.158966637    0.156868259    0.155008051    0.153354487
        0.151881056    0.150565311    0.149388121    0.148333091    0.147386091    0.146534891
        0.145768853    0.145078691    0.144456266    0.143894426    0.143386859    0.142927986
        0.14251286    0.142137086

# Catch (Conditioned on so known w/o error)
####################################

# catches  (Total catch in units of weight-at-age function)
92320   123527 112368 164214 87648   72861   61657   63199   95449   95041   81660   81524   76036
        98670   144710 137715 112750 116600 131322 93482   97304   74945   91878   92490   93183
        90229   83840   102790 73368   62132   66879   186185 98349   43756   75742   135989 74619
        92012   73225   64307   11183   17354   7021    10008   11959   10909   2982    10728   13704
        8144    31698   30849   26615   54639   46530   66558   74205   92284   65777   34072   18051
        16704   8962    28034   16491
```

```
############################
# Index/CPUE
############################

# number of indices
4

# Index start years (one for each index): ENP (juv),MRFSS (WFL,shore bt), MRFSS (EFL,offshore bt),
# Dive reef index

1975 1997 1997  1994

# Index end years (one for each index)
2014 2014 2014 2014

#Type of index for each index (1=biomass, 2=numbers)
2 2 2 2

# Indices

0.506   1.330   0.882   1.065   0.746   0.781   0.435   0.299   0.351   0.261   0.163   0.128   0.109
        0.155   0.347   0.178   0.173   0.201   0.277   0.649   1.080   1.177   0.774   0.566   0.566
        0.782   0.785   0.802   2.067   2.104   2.491   3.894   5.466   3.847   2.465   0.231   0.280
        0.447   0.522   0.566
0.289159534     0.28512006      0.268661363     0.607562304     0.867746399     0.912574427
        0.924122745     1.510899577     1.841169473     3.001326488     3.36804316      1.111989389
        1.049952529     0.232084813     0.445220482     0.106408056     0.882470786     0.295488417
0.549224753     0.556   0.261   0.300   0.246   0.360   0.590   1.144   1.712   1.622   2.027   2.126
        1.669   0.676   1.262   0.568   1.091   1.241
0.179   0.243   0.667   0.333   0.531   0.581   0.700   0.603   0.729   1.064   1.031   1.278   1.351
        1.251   1.275   1.342   1.913   1.561   1.739   1.654   0.975

# Indices CV

0.153   0.116   0.115   0.130   0.192   0.147   0.163   0.218   0.203   0.203   0.270   0.254   0.298
        0.293   0.185   0.204   0.234   0.348   0.162   0.102   0.109   0.090   0.102   0.132   0.139
        0.133   0.113   0.113   0.082   0.081   0.085   0.070   0.058   0.075   0.093   0.298   0.300
        0.220   0.183   0.193
0.5609  0.4512  0.2944  0.2398  0.2703  0.2227  0.2065  0.1679  0.1501  0.1424  0.1358  0.2181  0.1796
        0.3427  0.2893  0.4983  0.2801  0.3055
0.8383  0.4606  0.507   0.541   0.486   0.419   0.380   0.299   0.276   0.297   0.257   0.260   0.337
        0.454   0.462   0.445   0.440   0.308
0.326   0.395   0.314   0.301   0.261   0.140   0.129   0.111   0.095   0.089   0.099   0.109   0.102
        0.089   0.110   0.097   0.084   0.091   0.088   0.097   0.148
```

```
####################################
# Selectivity
####################################

#######################
# Fisheries Selectivity
#   aka Vulnerability
#######################
#Number of fisheries selectivity/vulnerability blocks (eg regulatory changes)
2

#Vulnerability at age per each block where year=1st year of block
#Year vul-at-age_vector
1950    0.00    0.01    0.05    0.28    0.72    0.95    1.00    1.00    1.00    1.00    1.00    1.00
        1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00
        1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.0000  1.0000  1.0000  1.0000  1.0000
        1.0000  1
#1990   0.00905471      0.174247889     0.262865242     0.163976084     0.072594978     0.029554841
        0.017161099     0.020176205     0.031690635     0.04581276      0.053108714     0.04723581
        0.033030365     0.019402403     0.010250257     0.005112922     0.002477315     0.0011834
        0.000561471     0.000265534     0.000125385     5.91643E-05     2.79078E-05     1.31619E-05
        6.207E-06       2.92704E-06     1.38028E-06     6.50884E-07     3.06929E-07     1.44734E-07
        6.82503E-08     3.21838E-08     1.51764E-08     7.15653E-09     3.3747E-09      1.59136E-09
        7.50414E-10     3.53862E-10
1990 0.00905471         0.183302599     0.446167841     0.610143925     0.682738902     0.712293744
        0.729454843     0.749631048     0.781321684     0.827134444     0.880243157     0.927478967
        0.960509332     0.979911736     0.990161993     0.995274915     0.99775223      0.99893563
        0.999497101     0.999762635     0.99988802      0.999947185     0.999975092     0.999988254
        0.999994461     0.999997388     0.999998769     0.999999419     0.999999726     0.999999871
        0.999999939     0.999999972     0.999999987     0.999999994     0.999999997     0.999999999
        1       1

#######################
#Number of selectivity blocks per index
1 1 1 1

# Indices selectivity at first year of block === SEE OTHOLITH Folder
# Needs to be in order -- index 1 1st block, index 1 2nd block, index 2 1st block, etc
#Index  Year    Sel-at-age_vector (ENP index and MRFSS inshore: same selectivity; MRFSS offshore and
dive: same logistic selectivity
1       1975    0.03300582      0.649743034     0.979637284     0.608273081     0.262655184
        0.092947436     0.02900122      0.008295949     0.0022269       0.000569434     0.000140136
        3.34345E-05     7.77544E-06     1.76979E-06     3.95523E-07     8.70102E-08     1.88801E-08
```

```
        4.04757E-09     8.58506E-10     1.80365E-10     3.75704E-11     7.76581E-12     1.594E-12
        3.25101E-13     6.59198E-14     1.32949E-14     2.66815E-15     5.33031E-16     1.06037E-16
        0       0       0       0       0       0       0       0       0
2       1997    0.03300582      0.649743034     0.979637284     0.608273081     0.262655184
        0.092947436     0.02900122      0.008295949     0.0022269       0.000569434     0.000140136
        3.34345E-05     7.77544E-06     1.76979E-06     3.95523E-07     8.70102E-08     1.88801E-08
        4.04757E-09     8.58506E-10     1.80365E-10     3.75704E-11     7.76581E-12     1.594E-12
        3.25101E-13     6.59198E-14     1.32949E-14     2.66815E-15     5.33031E-16     1.06037E-16
        0       0       0       0       0       0       0       0       0
#3      1997    0.000753696     0.00159697      0.003380552     0.007141883     0.015025122
        0.031335285     0.064196607     0.127001348     0.235769001     0.395488532     0.581131462
        0.746330939     0.861863829     0.929731752     0.965586767     0.98347168      0.992137306
        0.996276835     0.99824086      0.999169696     0.999608294     0.999815251     0.999912872
        0.999958912     0.999980625     0.999990863     0.999995692     0.999997968     0.999999042
        0.999999548     0.999999787     0.9999999       0.999999953     0.999999978     0.999999989
        0.999999995     0.999999998     0.999999999
3       1997    0.00905471      0.183302599     0.446167841     0.610143925     0.682738902
        0.712293744     0.729454843     0.749631048     0.781321684     0.827134444     0.880243157
        0.927478967     0.960509332     0.979911736     0.990161993     0.995274915     0.99775223
        0.99893563      0.999497101     0.999762635     0.99988802      0.999947185     0.999975092
        0.999988254     0.999994461     0.999997388     0.999998769     0.999999419     0.999999726
        0.999999871     0.999999939     0.999999972     0.999999987     0.999999994     0.999999997
        0.999999999     1       1
4       1994    0.000753696     0.00159697      0.003380552     0.007141883     0.015025122
        0.031335285     0.064196607     0.127001348     0.235769001     0.395488532     0.581131462
        0.746330939     0.861863829     0.929731752     0.965586767     0.98347168      0.992137306
        0.996276835     0.99824086      0.999169696     0.999608294     0.999815251     0.999912872
        0.999958912     0.999980625     0.999990863     0.999995692     0.999997968     0.999999042
        0.999999548     0.999999787     0.9999999       0.999999953     0.999999978     0.999999989
        0.999999995     0.999999998     0.999999999

####################################
# Age Composition
####################################

#Number of age comp records
0

#Compositional (proportion) data (here, fleet 1=catch, fleet 2+ are indices)
#Note: all zero values are effectively ignored in multinomial
[logLike=EffN*sum(ObsP@A*log(PredP@A))]
#Note: EffN can be sample observations, the model will spit out effective size estimate as
EffN=[predP@A*(1-predP@A)][(obsP@A-predP@A)^2]
#Fleet   Year    NsampRaw        Prop-at-age_vector
```

#fleet	year	N	0	1	2	3	4	5	6	7	8	9
	10	11	12	13	14	15	16	17	18	19	20	21
	22	23	24	25	26	27	28	29	30	31	32	33
	34	35	36	37								

```
####################################
# Estimated parameters with priors
#Low/high bounds, initial guess, FMSY prior (lognormal), prior CV and phz (use -1 for phz if not estimating)
#Note: turn prior CV negative to turn off prior
####################################
#Fmsy (current Fmsy w/ respect to changes in vulnerability)
#Fmsy/M~.9 (Zhou et al. 2012), M~0.18 -->> Fmsy=0.162

0.01 0.5 0.1 0.5 .5 1

#MSY (current MSY - Kg)

1000 200000 70000 10 0.5 2  # based on KG

####################################
# Derived parameter priors
# Note: turn prior CV negative to turn off prior
####################################

#Recruit deviations: low, high, phase, and SD
-5.0 5.0 3 0.6  # SD of 0.6 based on Rose et al. 2001 , Fish and Fisheries 2: 293-327

#### Compensation ratio prior and its CV ####
# Normally distributed since can go negative if Fmsy upper bound too high
#CR=-(4*h)/(h-1) for reference to steepness
#0.80h = 16CR   #Shertzer and Conn (2012) for normally distributed h, sd=0.19,
#0.81h = 17.1CR #Joseph steepness prior based on Mangel
16 0.3

 #### F-rate observations ####
# Number of prior estimates of year-specific F (-log(1-ut*va))
1

# Years of prior estimates
2014   #1989
```

```
# Estimates
0.5  # originally 0.1

# Estimates CV
0.1  # originally 0.1


#####################################
# Lambdas
#####################################

## Indices (1 lambda for each index so 1+)
 1  1  1  1
# 0.183700629 0.35134402 0.62 0.264783759  ## Francis reweighting of residuals
# 5.75 3.13 5.3 7.44    ### 5.3 based on reciprocal of SD of residuals in loess fits
#5.75 1  5.3 7.44
 #104.37 45.24 284.72 16.72 # based on Cvs of residuals in loess fit

## Age Comps (1 lambda for  catch + each index age comps so 2+)
# 10  1   1
1  1 1 1 1


## Fmsy Prior
1

## MSY Prior
1

## Rec K Prior
1

## F-obs Prior(s) (1 for each F observation)
1

## For stock status
 2012  # start of recent years

#####################################
# EOF (don't change)
#####################################
123456
```

<end of working paper>