The RUNTIME_SECTION allows the user to modify the default behavior of the function minimizer during the phases of the estimation process:

RUNTIME_SECTION

```
convergence_criteria .1, .1, .001
maximum_function_evaluations 20, 20, 1000
```

The convergence_criteria affects the criterion used by the function minimizer to decide when the optimization process has occurred. The function minimizer compares the maximum value of the vector of derivatives of the objective function, with respect to the independent variables, to the numbers after the convergence_criteria keyword. The first number is used in the first phase of the optimization, the second number in the second phase, and so on. If there are more phases to the optimization than there are numbers, the last number is used for the rest of the phases of the optimization. The numbers must be separated by commas. The spaces are optional. The maximum_function_evaluations keyword controls the maximum number of evaluations of the objective function that will be performed by the function minimizer in each stage of the minimization procedure.

1.17 Natural resource management: the Schaeffer-Pella-Tomlinson Model

It is typical of many models in natural resource management that the model tends to be rather unstable numerically. In addition, some of the model parameters are often poorly determined. Notwithstanding these difficulties, it is often necessary to make decisions about resource management based on the analysis provided by these models. This example provides a good opportunity for presenting some more advanced features of AD Model Builder that are designed to overcome these difficulties.

The Schaeffer-Pella-Tomlinson model is employed in fisheries management. The model assumes that the total biomass of an exploited fish stock satisfies an ordinary differential equation of the form

$$\frac{dB}{dt} = rB\left(1 - \left(\frac{B}{k}\right)^{m-1}\right) - FB \qquad \text{where} \quad m > 1$$
 (1.8)

([7], page 303), where B is the biomass, F is the instantaneous fishing mortality rate, r is a parameter often referred to an the "intrinsic rate of increase," k is the unfished equilibrium stock size,

$$C = FB \tag{1.9}$$

is the catch rate, and m is a parameter that determines where the maximum productivity of the stock occurs. If the value of m is fixed at 2, the model is referred to as the "Schaeffer model." The explicit form of the difference equation corresponding to this differential equation is

$$B_{t+\delta} = B_t + rB_t \delta - rB_t \left(\frac{B_t}{k}\right)^{m-1} \delta - F_t B_t \delta \tag{1.10}$$

To get a semi-implicit form of this difference equation that has better numerical stability than the explicit version, we replace some of the terms B_t on the right hand side of 1.10 by $B_{t+\delta}$, to get

$$B_{t+\delta} = B_t + rB_t \delta - rB_{t+\delta} \left(\frac{B_t}{k}\right)^{m-1} \delta - F_t B_{t+\delta} \delta \tag{1.11}$$

We solve for $B_{t+\delta}$ to give

$$B_{t+\delta} = \frac{B_t(1+r\delta)}{1+(r(B_t/k)^{m-1}+F_t)\delta}$$
(1.12)

The catch $C_{t+\delta}$ over the period $(t, t + \delta)$ is given by

$$C_{t+\delta} = F_t B_{t+\delta} \delta \tag{1.13}$$

1.18 Bayesian considerations in the Pella-Tomlinson Model

The parameter k is referred to as the "carrying capacity" or the "unfished equilibrium biomass level," because it is the value that the biomass of the population will eventually assume if there is no fishing. For a given value of k, the parameter m determines the level of maximum productivity—that is, the level of biomass B_{MAX} for which the removals from fishing can be the greatest:

$$B_{\text{MAX}} = \frac{k}{\sqrt[m-1]{m}}$$

For m=2, maximum productivity is obtained by that level of fishing pressure that reduces the stock to 50% of the carrying capacity. For the data available in many real fisheries problems, the parameter m is very poorly determined. It is common practice, therefore, to simply assume that m=2. Similarly, it is commonly assumed that the carrying capacity k does not change over time, even though changes such as habitat degradation may well lead to changes in k.

We want to construct a statistical model where the carrying capacity can be varying slowly over time if there appears to be any information in the fisheries data supporting this hypothesis. What is meant by "slowly"? The answer to this question will depend on the particular situation. For our purposes, "slowly" means "slowly enough" so that the model has some chance of supplying a useful analysis of the situation at hand. We refer to this as "the assumption of manageability." The point is that since we are going to use this model anyways to try and mange a resource, we may as well assume that the model's assumptions are satisfied—at least well enough that we have some hope of success. This may seem extremely arbitrary, and it is. However, it is not as arbitrary as assuming that the carrying capacity is constant.

We assume that $k_{i+1} = k_i \exp(\kappa_i)$, where the κ_i are independent normally distributed random variables with mean 0, and that $\log(m-1)$ is normally distributed with mean 0.

The parameters log(k) are assumed to have the structure of a random walk, which is the simplest type of time series. This Bayesian approach is a very simple method for including a time series structure into the parameters of a nonlinear model.

We don't know the true catches C_i in each year. What we have are estimates C_i^{obs} of the catch. We assume that the quantities $\log(C_i^{obs}/C_i)$ are normally distributed with mean 0.

Finally, we must deal with the fishing mortality F. Estimates of F are not obtained directly. Instead, what is observed is an index of fishing mortality—in this case, fishing effort. We assume that for each year, we have an estimate E_i of fishing effort and that the fishing mortality rate F_i in year i satisfies the relationship $F_i = qE_i \exp(\eta_i)$, where q is a parameter referred to as the "catchability" and the η_i are normally distributed random variables with mean 0.

We assume that the variance of the η_i is 10 times the variance in the observed catch errors, and that the variance of the κ_i is 0.1 times the variance in the observed catch errors. We assume that the variance in $\log(m-1)$ is 0.25. Then, given the data, the Bayesian posterior distribution for the model parameters is proportional to

$$-(3n-1)/2\log\left(\sum_{i=1}^{n} \left(\log(C_i^{obs}) - \log(C_i)\right)^2 + .1\sum_{i=1}^{n} \eta_i^2 + 10\sum_{i=2}^{n} \kappa_i^2\right) - 2.\log(m-1)^2$$

$$(1.14)$$

The number of initial parameters in the model (that is, the number of independent variables in the function to be minimized) is 2n+4. For the halibut data, there are 56 years of data, which gives 116 parameters. As estimates of the model parameters, we use the mode of the posterior distribution, which can by found by minimizing -1 times expression (1.8). The covariance matrix of the model parameters are estimated by computing the inverse of the Hessian of expression (1.8) at the minimum. The template for the model follows. To improve the readability, the entire template has been included. The various sections are discussed below.

```
DATA_SECTION

init_int nobs;
init_matrix data(1,nobs,1,3)

vector obs_catch(1,nobs);
vector cpue(1,nobs);
vector effort(1,nobs);
number avg_effort

INITIALIZATION_SECTION

m 2.
beta 1.
r 1.

PARAMETER_SECTION

init_bounded_number q(0.,1.)
init_bounded_number beta(0.,5.)
```

```
init_bounded_number r(0.,5,2)
 init_number log_binit(2)
 init_bounded_dev_vector effort_devs(1,nobs,-5.,5.,3)
 init_bounded_number m(1,10.,4)
 init_bounded_vector k_devs(2,nobs,-5.,5.,4)
 number binit
 vector pred_catch(1,nobs)
 vector biomass(1,nobs)
 vector f(1,nobs)
 vector k(1,nobs)
 vector k_trend(1,nobs)
 sdreport_number k_1
 sdreport_number k_last
 sdreport_number k_change
 sdreport_number k_ratio
 sdreport_number B_projected
 number tmp_mort;
 number bio_tmp;
 number c_tmp;
 objective_function_value ff;
PRELIMINARY_CALCS_SECTION
 // get the data out of the data matrix into
 obs_catch=column(data,2);
 cpue=column(data,3);
 // divide the catch by the cpue to get the effort
 effort=elem_div(obs_catch,cpue);
 // normalize the effort and save the average
 double avg_effort=mean(effort);
 effort/=avg_effort;
 cout << " beta" << beta << endl;</pre>
PROCEDURE_SECTION
 // calculate the fishing mortality
 calculate_fishing_mortality();
 // calculate the biomass and predicted catch
 calculate_biomass_and_predicted_catch();
 // calculate the objective function
 calculate_the_objective_function();
FUNCTION calculate_fishing_mortality
 // calculate the fishing mortality
 f=q*effort;
 if (active(effort_devs)) f=elem_prod(f,exp(effort_devs));
```

```
FUNCTION calculate_biomass_and_predicted_catch
  // calculate the biomass and predicted catch
  if (!active(log_binit))
   log_binit=log(obs_catch(1)/(q*effort(1)));
  binit=exp(log_binit);
  biomass[1]=binit;
  if (active(k_devs))
   k(1)=binit/beta;
   for (int i=2;i<=nobs;i++)</pre>
     k(i)=k(i-1)*exp(k_devs(i));
   }
  }
  else
   // set the whole vector equal to the constant k value
   k=binit/beta;
  // only calculate these for the standard deviation report
  if (sd_phase)
   k_1=k(1);
   k_last=k(nobs);
   k_{ratio}=k(nobs)/k(1);
   k_change=k(nobs)-k(1);
  }
  // two time steps per year desired
  int nsteps=2;
  double delta=1./nsteps;
  // Integrate the logistic dynamics over n time steps per year
  for (int i=1; i<=nobs; i++)
   bio_tmp=1.e-20+biomass[i];
   c_{tmp=0.;}
   for (int j=1; j<=nsteps; j++)
   {
     //This is the new biomass after time step delta
     bio_tmp=bio_tmp*(1.+r*delta)/
```

```
(1.+ (r*pow(bio_tmp/k(i),m-1.)+f(i))*delta);
     // This is the catch over time step delta
     c_tmp+=f(i)*delta*bio_tmp;
   pred_catch[i]=c_tmp; // This is the catch for this year
   if (i<nobs)
     biomass[i+1]=bio_tmp;// This is the biomass at the begining of the next
   }
                        // year
   else
   {
     B_projected=bio_tmp; // get the projected biomass for std dev report
   }
 }
FUNCTION calculate_the_objective_function
 if (!active(effort_devs))
   ff=nobs/2.*log(norm2(log(obs_catch)-log(1.e-10+pred_catch)));
 }
 else if(!active(k_devs))
   ff= .5*(size_count(obs_catch)+size_count(effort_devs))*
     log(norm2(log(obs_catch)-log(1.e-10+pred_catch))
     +0.1*norm2(effort_devs));
 }
 else
   ff= .5*( size_count(obs_catch)+size_count(effort_devs)
     +size_count(k_devs) )*
     log(norm2(log(obs_catch)-log(pred_catch))
     + 0.1*norm2(effort_devs)+10.*norm2(k_devs));
 }
 // Bayesian contribution for Pella Tomlinson m
 ff+=2.*square(log(m-1.));
 if (current_phase()<3)</pre>
   ff+=1000.*square(log(mean(f)/.4));
 }
```

The data are contained in three columns, with the catch and catch per unit effort data contained in the second and third columns. The matrix data is defined in order to read the data. The second and third columns of data, which is what we are interested in, will then

be put into the vectors obs_catch and cpue. (Later, we get the fishing effort by dividing the obs_catch by the cpue.)

```
DATA_SECTION
  init_int nobs
  init_matrix data(1,nobs,1,3)
  vector obs_catch(1,nobs)
  vector cpue(1,nobs)
  vector effort(1,nobs)
  number avg_effort
```

The INITIALIZATION_SECTION is used to define default values for some model parameters if the standard default provided by AD Model Builder is not acceptable. If the model finds the parameter file (whose default name is admodel.par), it will read in the initial values for the parameters from there. Otherwise, the default values will be used—unless the parameters appear in the INITIALIZATION_SECTION, in which case, those values will be used.

```
INITIALIZATION_SECTION
  m 2.
  beta 1.
  r 1.
```

The PARAMETER_SECTION for this model introduces several new features of AD Model Builder. The statement

```
init_bounded_number r(0.,5.,2)
```

declares an initial parameter whose value will be constrained to lie between 0.0 and 5.0. It is often necessary to put bounds on the initial parameters in nonlinear models to get stable model performance. This is accomplished in AD Model Builder simply by declaring the initial parameter to be bounded and providing the desired bounds. The default initial value for a bounded object is the average of the lower and upper bounds.

The third number '2' in the declaration determines that this initial parameter will not be made active until the second phase of the minimization. This introduces the concept of "phases" in the minimization process.

As soon as nonlinear statistical models become a bit complicated, one often finds that simply attempting to estimate all the parameters simultaneously does not work very well. In short, "You can't get there from here." A better strategy is to keep some of the parameters fixed and to first minimize the function with respect to the other parameters. More parameters are added in a stepwise relaxation process. In AD Model Builder, each step of this relaxation process is termed a "phase." The parameter r is not allowed to vary until the second phase. Initial parameters that are allowed to vary will be termed "active." In the first phase, the active parameter are beta and q. The default phase for an initial parameter is phase 1 if no phase number is included in its declaration. The phase number for an initial parameter is the last number in the declaration for that parameter. The general order for

the arguments in the definition of any initial parameter is the size data for a vector or matrix object (if needed), the bounds for a bounded object (if needed), followed by the phase number (if desired).

It is often a difficult problem to decide what the order of relaxation for the initial parameters should be. This must sometimes be done by trial and error. However, AD Model Builder makes the process a lot simpler. One only needs to change the phase numbers of the initial parameters in the PARAMETER_SECTION and recompile the program.

Often in statistical modeling, it is useful to regard a vector of quantities x_i as consisting of an overall mean, μ , and a set of deviations from that mean, δ_i , so that

$$x_i = \mu + \delta_i$$
 where $\sum_i \delta_i = 0$

AD Model Builder provides support for this modeling construction with the

```
init_bounded_dev_vector
```

declaration. The components of an object created by this declaration will automatically sum to zero without any user intervention. The line

```
init_bounded_dev_vector effort_devs(1,nobs,-5.,5.,3)
```

declares effort_devs to be this kind of object. The bounds -5.,5. control the range of the deviations. Putting reasonable bounds on such deviations often improves the stability of the estimation procedure.

AD Model Builder has sdreport_number, sdreport_vector, and sdreport_matrix declarations in the PARAMETER_SECTION. These objects behave the same as number, vector, and matrix objects, with the additional property that they are included in the report of the estimated standard deviations and correlation matrix.

For example, merely by including the statement sdreport_number B_projected, one can obtain the estimated standard deviation of the biomass projection for the next year. (Of course, you must also set B_projected equal to the projected biomass. This is done in the PROCEDURE_SECTION.)

PARAMETER_SECTION

```
init_bounded_number q(0.,1.)
init_bounded_number beta(0.,5.)
init_bounded_number r(0.,5,2)
init_number log_binit(2)
init_bounded_dev_vector effort_devs(1,nobs,-5.,5.,3)
init_bounded_number m(1,10.,4)
init_bounded_vector k_devs(2,nobs,-5.,5.,4)
number binit
vector pred_catch(1,nobs)
vector biomass(1,nobs)
```

```
vector k(1,nobs)
vector k_trend(1,nobs)
sdreport_number k_1
sdreport_number k_last
sdreport_number k_change
sdreport_number k_ratio
sdreport_number B_projected
number tmp_mort;
number bio_tmp;
number c_tmp;
objective_function_value ff;
```

The PRELIMINARY_CALCS_SECTION carries out a few simple operations on the data. The model expects to have catch and effort data, but the input file contained catch and "cpue" ("catch/effort") data. We divide the catch data by the cpue data to get the effort data. The AUTODIF operation elem_div, which performs element-wise divisions of vector objects, is used. As usual, the same thing could have been accomplished by employing a loop and writing element-wise code. The effort data are then normalized—that is, they are divided by their average so that their average becomes 1. This is done so that we have a good idea what the catchability parameter q should be to give reasonable values for the fishing mortality rate (since F = qE).

extract a column from a matrix Notice that the PRELIMINARY_CALCS_SECTION section is C^{++} code, so that statements must be ended with a ;.

```
PRELIMINARY_CALCS_SECTION
```

```
// get the data out of the data matrix into
obs_catch=column(data,2);
cpue=column(data,3);
// divide the catch by the cpue to get the effort
effort=elem_div(obs_catch,cpue);
// normalize the effort and save the average
double avg_effort=mean(effort);
effort/=avg_effort;
```

The PROCEDURE_SECTION contains several new AD Model Builder features. Some have to do with the notion of carrying out the minimization in a number of steps or phases. The line

```
if (active(effort_devs)) f=elem_prod(f,exp(effort_devs));
```

introduces the active function. This function can be used on any initial parameter and will return a value "true" if that parameter is active in the current phase. The idea here is that if the initial parameters effort_devs are not active, then since their value is zero, carrying out the calculations will have no effect, and we can save time by avoiding the calculations. The active function is also used in the statement

```
if (!active(log_binit))
```

```
{
  log_binit=log(obs_catch(1)/(q*effort(1)));
}
```

The idea is that if the log_binit initial parameter (this is the logarithm of the biomass at the beginning of the first year) is not active, then we set it equal to the value that produces the observed catch (using the relationship C = qEB, such that B = C/(qE). The active function is also used in the calculations of the objective function, so that unnecessary calculations are avoided.

The following code helps to deal with convergence problems in this type of nonlinear model. The problem is that the starting parameter values are often so bad that the optimization procedure will try to make the population very large and the exploitation rate very small, because this is the best local solution near the starting parameter values. To circumvent this problem, we include a penalty function to keep the average value of the fishing mortality rate f close to 0.2 during the first two phases of the minimization. In the final phase, the size of the penalty term is reduced to a very small value. The function current_phase() returns the value of the current phase of the minimization:

```
if (current_phase()<3)
{
   ff+=1000.*square(log(mean(f)/.4));
}</pre>
```

1.19 Using functions to improve code organization

Subroutines or functions are used to improve the organization of the code. The code for the main part of the PROCEDURE_SECTION that invokes the functions should be placed at the top of the PROCEDURE_SECTION.

```
PROCEDURE_SECTION
  // calculate the fishing mortality
  calculate_fishing_mortality();
  // calculate the biomass and predicted catch
  calculate_biomass_and_predicted_catch();
  // calculate the objective function
  calculate_the_objective_function();
```

There are three user-defined functions called at the beginning of the PROCEDURE_SECTION. The code to define the functions comes next. To define a function whose name is, say, name, the template directive FUNCTION name is used. Notice that no parentheses () are used in the definition of the function, but to call the function, the statement takes the form name();