C: Commands and Functions

Fish 559; Day2: 09h00-10h30

The “If” statement

- All the normal C++ statements can be used in the Procedure Section:
    - The if statement:

```
if (condition)
   Statements-1;
else
   Statements-2;
```

    - Combining statements:

```
{ statement; statement }
```

Constructing conditions

- RELATIONAL OPERATORS:
  - ==   Equal to (NOT =)
  - <=   Less than or equal to
  - >=   Greater than or equal to
  - <   Less than
  - >   Greater than
  - !=   Not equal to
- LOGICAL OPERATORS:
  - &&   Condition is true if both sub-conditions are true
  - || Condition is true if one of the sub-conditions is true
  - !   Condition is true if sub-condition is false

- Note: TMB will not work if a parameter or derived variable appears in an condition. "You should not use if(x) statements where x is a PARAMETER, or is derived from a variable of type PARAMETER. TMB will remove the if(x) statement, so the code will produce unexpected results."

The For statement

```
for (initial state; terminal condition; increment)
  { statements }
```

- Initial state: Usually involves setting a “loop control variable” to some initial value.
- Terminal condition: The loop will run until this condition is TRUE (note it need not involve the “loop control variable”).
- Increment: This usually involves updating the “loop control variable” (e.g. X++; X--).
- Note: Parameters and derived variables should not appear in the terminal condition.

The For statement-II

```
Biomass(1) = 1;
for (Year=1; Year<=Nyear; Year++)
   Biomass(Year+1)=Biomass(Year)*r;
```

What does “for(;;;)” do???

Arrays, Matrices, and 3d-arrays

- Arrays and matrices behave the same way as a mathematical vector.
- You can reference whole arrays or elements of arrays (the former is generally faster).
- We will often use arrays in conjunction with “for” loops.

Built-in Functions in TMB-I
(Mathematical functions)

- Default functions:

  sin, cos, asin, atan, acos, sinh, cosh, tanh, fabs*, exp, log, sqrt, pow

These functions operate on numbers and vectors, i.e:

number = exp(number); vector = exp(vector)

- One can also find the minimum and maximum of a vector: min(x)*; max(x)*

* Are these differentiable?

Built-in Functions in TMB-II

(Vector & matrix functions)

http://kaskr.github.io/adcomp/matrix_arrays_8cpp-example.html

- c = a*b: $c_i = a_i * b_i$
- c = a/b: $c_{i,j} = a_{i,j} / b_{i,j}$
- matrix_b = atomic::logdet (matrix_a) – determinant
- matrix_b = atomic::matinv(matrix_a) – inverse
- matrix_b = matrix_a.transpose() - transpose

Built-in Functions in ADMB-III
(Extraction functions)

- Extracting from arrays and matrices:

```
vector = matrix.col(index)
vector = matrix.row(index)
vector = matrix.diagonal();
```

- Subvectors: extract a subset of a vector:

```
vector2 = vector1(index1,index2)
```

* note arrays start at 0 so index1=2 means third element

FUNCTIONs-I

- Why functions?
  - We often wish to break the evaluation of the objective function into sub-components (which we may wish to call several times).
  - We can insert (commonly-used and tested) functions “straight” into a new program.

FUNCTIONs-II

ADMB has a square function and I wanted one too..

```
template <class Type> Type square(Type x){return x*x;}
```

This appears before the objective function and I can use it there

```
if (y < nyears-1) N(y+1,0) = exp(logNBar+Nrec(y)-square(SigmaR)/2.0);
```

Typing is super important!

FUNCTIONs-III

Sometimes we want to write functions that we will re-use:

```
template <class Type> vector<Type> DLogistic( vector<Type> XX,
vector<Type>  sp)
 {
  int n1 = 0;
  int n2 = XX.size();
  Type binwidth2 = XX(1) - XX(0);

  vector<Type> Select(n2);

  Type peak = sp(0);
  Type upselex = exp(sp(2));
  Type downselex = exp(sp(3));
return(Select);
}
```

This function returns a vector and takes as input two vectors. Note also the careful use of indexes (they start at zero).

FUNCTIONs-IV

Will this work?

```
template<class Type>
Type posfun(Type x, Type eps, Type &pen)
{
  if ( x >= eps ){
    return x;
  } else {
    pen += Type(0.01) * pow(x-eps,2);
    return eps/(Type(2.0)-x/eps);
  }
}
```

& not passing pen, passing the location of pen in memory. It maps the value back to the original pen instead of creating a new variable pen that is local to this function

Making sure that 0.01 is a real number

Hints: Debugging

- TMB code is NOT easy to debug:
  - The error messages will appear on the screen. One error can lead to many message so compile early and frequently.
  - Use “_std::cout << "SSBComp " << SSBComp << "\n";_” to output variables to the screen so you can check that the function is being calculated correctly. Sadly this only shows the first function calls.
  - Start with a previously working CPP file and modify it. You can use the “template” function to get a default cpp file.
  - I often test code by running it with no variables being estimated (just an objective function equal to _dummy*dummy_ where _dummy_ is a new variable that does not appear in the model). This allows me to check the calculations “by hand”.

Programming Style

- Comment, comment, comment:
  - Always include a comment (indicated by "//") at the start of the program that states what it does.
  - Split ideas / blocks of code with blank lines and comments.
  - Use "inline" comments to refer to equations and meanings of variables.
  - Indent blocks of code to increase clarity.

Programming Hints

- Start simple: develop a routine, test it, develop the next routine, …
- Use std::cout: test code by writing out intermediate results (you should then check that the values output are correct given the values for the parameters – the first call of the function will involve the parameters being set to their initial values).
- Cross compare: Write critical bits of code in EXCEL or R and check you get the correct function value.
- Never, ever, ever, believe your code is correct: always work on the assumption there is an error there somewhere!

Debugging (from Cole Monnahan)

- Build changes slowly, recompiling often
- When compilation fails, look at first error only
- For runtime errors, comment out sections to test and replace with simpler code (usually a wrong index)
- Use REPORT macros to print intermediate steps in R (more later)

```
TMB has received an error from Eigen. The following condition was not met:
index >= 0 && index < size()
Please check your matrix-vector bounds etc., or run your program through a debugger.

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.

MEANS ARRAY IS OUT OF BOUNDS
```

The Example Problem-I

- Fit the dynamic Schaefer model to the catch and effort data for Cape hake off the west coast of South Africa. E=exploitation rate

$$B_{t+1} = B_t + rB_t(1 - B_t / K) - E_t B_t ; \;\; B_{1964} = K$$

$$-\ell nL = n\ell\mathrm{n}\sigma + \tfrac{1}{2\sigma^2}\sum_y \left(\ell\mathrm{n}I_y - \ell\mathrm{n}(q\,B_t)\right)^2 + n\ell\mathrm{n}\,\tau + \tfrac{1}{2\tau^2}\sum_y \left(\ell\mathrm{n}C_y - \ell\mathrm{n}(E_t\,B_t)\right)^2$$

- The data are located in the file LECTC2.DAT

The Example Problem-II

```
template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(C);
  DATA_VECTOR(I);
  int n = C.size();

  PARAMETER(logR);
  PARAMETER(logK);
  PARAMETER(logQ);
  PARAMETER(logSigma);
  PARAMETER_VECTOR(FF); // differnt exploit rate every year
  Type r = exp(logR);
  Type k = exp(logK);
  Type q = exp(logQ);
  Type sigma = exp(logSigma);
```

The data basic data

Extract the length of C

Estimated parameters (names MUST match those in the R code)

The Example Problem-III

```
int n1 = 0;
n1 = n + 1;
vector<Type> B(n1);
vector<Type> Ihat(n);
vector<Type> Chat(n);
vector<Type> ExpOut(n);
Type f;
```

Temporary variables (functions of the parameters)

The Example Problem-IV

```
Type f;
 B(0) = k;
 for(int t=0; t<n; t++)
 {
   Type Expl = 1.0/(1.0+exp(-FF(t)));
   B(t+1) = B(t) + r*B(t)*(1-B(t)/k) - Expl*B(t);
   Chat(t) = Expl*B(t);
   ExpOut(t) = Expl;
   Ihat(t) = q*B(t);
 }
 f = -sum(dnorm(log(C), log(Chat), Type(0.05), true));
 f -= sum(dnorm(log(I), log(Ihat), sigma, true));
```

Selected to avoid “ifs”