Introduction to ADMB

FISH 599; Lecture 13

What Does ADMB Do?

- ADMB can find the value of the parameter vector that minimizes a (complex) non-linear function of many variables.
- It can compute measures of uncertainty by:
  - estimating asymptotic variance-covariance matrices;
  - computing likelihood profiles for parameters and model outputs - relatively how much better is one hypothesis over another; and
  - sampling parameter vectors from Bayesian posteriors using the Markov Chain Monte Carlo (MCMC) algorithm.

Specifying the Problem

- ADMB programs are written using a template (a file with a .TPL extension).
- The TPL file specifies:
  - the function to be minimized ;
  - the parameters that are to be varied to minimize the function;
  - any variables that depend on the parameters but are not parameters themselves; and
  - the data (constants) that are part of the function.

The ADMB "Approach"

- Create a .TPL file. This file provides the specifications to:
  - read in the data;
  - identify the parameters and any derived variables;
  - define the model; and
  - define the objective function to be minimized.
- Convert the TPL file to a C++ file (a CPP file) using tpl2cpp.exe.
- Compile and link the resultant C++ program as you would any other C++ program.
- The ADMB documentation is very terse but can be helpful – consult it!

The First Example - I

- We start with a least-squares problem:
  - Find *a* and *b* by minimizing:

$$SS = \sum_i \left(y_i - (a + bx_i) \right)^2$$

- Program: LECT13A.TPL
- Data File: LECT13A.DAT
- To compile: ADMB -s LECT13A
- To run: LECT13A

- Have a look at the various output files

The First Example - II (Linear Regression)

Sections:

DATA – the input data (read from LECTA.DAT)

PARAMETER – the model parameters, any functions of the parameters, and the objective function.

PROCEDURE – where the calculations are done

There are other SECTIONS which will come later.

```
DATA_SECTION
  init_int NData;
  init_vector x(1,NData);
  init_vector y(1,NData);

PARAMETER_SECTION
  init_number a;
  init_number b;

  vector ypred(1,NData);
  objective_function_value obj_fun;

PROCEDURE_SECTION
  ypred = a + b*x;
  obj_fun = norm2(y-ypred);
```

The First Example - III (Linear Regression)

Reserved words:
- To specify storage types (in blue)
- To specify built-in functions (in green)

catch is a reserved word!!

Some notes:
- You can add comments (lines starting with //) – use them!
- ADMB works with vectors – see how the predicted values are computed.
- The "objective_function_value" is the variable being minimized.

```
DATA_SECTION
  init_int NData;
  init_vector x(1,NData);
  init_vector y(1,NData);

PARAMETER_SECTION
  init_number a;
  init_number b;

  vector ypred(1,NData);
  objective_function_value obj_fun;

PROCEDURE_SECTION
  ypred = a + b*x;
  obj_fun = norm2(y-ypred);
 // This is a comment
```

Our First Example - IV (Adding output)


```
DATA_SECTION
  init_int NData;
  init_vector x(1,NData);
  init_vector y(1,NData);

PARAMETER_SECTION
  init_number a;
  init_number b;

  vector ypred(1,NData);
  objective_function_value obj_fun;

PROCEDURE_SECTION
  ypred = a + b*x;
  cout << ypred << endl;
  obj_fun = norm2(y-ypred);
 // This is a comment
```

```
REPORT_SECTION
report << ypred << endl;
report << y << endl;
```

Use:

- "cout" to output the values for variables to the screen; and
- "report" to output results to the .REP file (only after the minimization is complete)

The First Example - V
(Output Files of Interest)

- LECT13A.PAR – lists the parameter estimates, the final objective function, and the gradient – the latter should be close to zero!
- LECT13A.STD – lists the parameter estimates and the (asymptotic) standard errors.
- LECT13A.COR – lists the parameter estimates, their (asymptotic) standard errors, and their (asymptotic) correlation matrix.
- LECT13A.REP – lists any variables specified in the REPORT_SECTION.

Dissecting The Program (arithmetic)

```
ypred = a + b*x;
obj_fun = norm2(y-ypred);
```

- The first line is a vector operation - it defines a vector, each element of which, ypred(i), is *a*+*b***x(i)*.
- The second line sums the squares of the differences between the observed (*y*) and predicted (*ypred*) values – this is also a vector operation.

Something About Arithmetic

- The following are the ADMB arithmetic expressions:
  - Z=a+b – addition (note Z+=b)
  - Z=a-b – subtraction
  - Z=a*b – multiplication
  - Z=a/b – division
- ADMB implements all the standard functions (and more):
  - Z=log(a)
  - Z=mfexp(a)
  - Z=pow(a,b) – exponentiation
  - Z=square(a)
  - Z=norm2(vec1,vec2)

Declaring Variables

- Before a variable is used, it has to be declared.
- Key distinction among variables:
  - Fixed throughout the program (data to be read in, values for fixed parameters, e.g. *Survival*) – declare in the DATA_SECTION.
  - The parameters to be estimated and any variables that are functions of these parameters – declare in the PARAMETER_SECTION.
- Declarations are of the form:

"Data type" "variable name";

Data Types

ADMB needs to know what dimensions and format your data/variables/parameters will be in!

- The most-basic types:
  - int (e.g. int Count) – integer.
  - number (e.g. number Biomass) – real number.
  - vector (e.g. vector me(1,10)) – set of real numbers indexed by 1,2,..,10.
  - ivector – as for vector, except that the contents are integers.
  - matrix – two dimensional array of real;
  - imatrix – two dimensional array of integer;
  - 3darray – three dimensional array of real; and
  - 4darray – four dimensional array of real
- All variables MUST be declared before they are used – no exemptions!

Variable Names-I

- Variable names:
  - Must start with an alphabetic character.
  - Don't use any reserved words (if, else, etc.)
  - Choose descriptive but not overly long variable names (e.g. biomass, MSY).
  - ADMB (and C) is case-sensitive, i.e. the variables biomass and Biomass are NOT the same variable.

Variable Names - II

- Other rules / hints:
  - Use underscores to split names within a variable name (e.g. my_biomass).
  - Avoid re-using the same variable for different purposes.
  - Don’t forget those semi-colons!

Programming the Procedure Section-I
(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 }
```

Programming the Procedure Section-II (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: ADMB may not work if a parameter or derived variable appears in an condition.

Programming the Procedure Section-III
(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.

Arrays, Matrices, and 3d-arrays

- Arrays, matrices, and 3d-arrays 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 ADMB-I
(Mathematical functions)

- Default functions:

  sin, cos, asin, atan, acos, sinh, cosh, tanh, fabs*, exp, log, log10, sqrt, pow, gammln, log_comb

- 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 ADMB-II
(Vector & matrix functions)

- c = elem_prod(a,b): $c_i = a_i * b_i$
- c = elem_div(a,b): $c_{i,j} = a_{i,j} / b_{i,j}$
- matrix_b = det(matrix_a) – determinant
- matrix_b = inv(matrix_a) – inverse
- number_a = norm(x): $a = \sqrt{\sum_i x_i^2}$
- number_a = norm2(x): $a = \sum_i \sum_j x_{i,j}^2$
- matrix_b = trans(matrix_a) - transpose

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

- Extracting from arrays and matrices:

```
vector = column(matrix,index)
vector = extract_row(matrix,index)
vector = extract_diagonal(matrix)
```

- Subvectors: extract a subset of a vector:

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

Functions and crashes

- Nothing causes ADMB more problems than:
  (a) IF statements; and
  (b) Functions that have very large derivatives.
  (c) Discontinuous functions or functions that are "bumpy"
- How to overcome keeping a variable (x) positive (and differentiable):
  (a) Use posfunc(x, eps, penalty)
  (b) x = sqrt(x*x);

FUNCTIONs-I

- FUNCTIONS are part of the PROCEDURE_SECTION.
- 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 (an example)

```
PROCEDURE_SECTION

  // Set the fixed values
  SetFixed();

  // Calculate the N matrix
  InitialN();

  // Find the Index Likelihood
  Index_Likelihood();
  // Find the age-composition likelihood
  if (CAAESS > 0) Age_Likelihood();
  // Find the length-composition likelihood
  if (NLLESS > 0) Length_Likelihood();

FUNCTION SetFixed
  Code

FUNCTION InitialN
  Code
```

These are function calls – note the need for "()"

This the BODY of the function SetFixed – note the need for the word FUNCTION

FUNCTIONs-III (passing parameters)

- Most computer languages (including C++) allow parameters to be "passed" between a calling program and a function.
- This is generally avoided when using ADMB.
- To make a variable common to all functions, declare it in the DATA_SECTION (input data or functions of the input data; any integer values) or the PARAMETER_SECTION (variables that depend on the model parameters).

Declaring Variables in the PROCEDURE_SECTION

- You will often need "temporary" variables in the PROCEDURE_SECTION or the REPORT_SECTION. These can be declared by adding:
  - "int variable_name" for an integer;
  - "dvariable variable_name" for a real;
  - "dvar_vector variable_name(a,b)" for a vector.

  at the start of a FUNCTION.

Commands in other Sections

- Standard C++ commands can be used in the *PROCEDURE*, *REPORT* and *PRELIMINARY_CALCS* sections.
- To use standard C++ commands in the *DATA* and *PARAMETER* sections use either:

```
!! Statement;

LOCAL_CALCS
    Statements;
END_CALCS
```

Hints: Debugging

- ADMB code is NOT easy to debug:
  - If you run the DOS version of the compiler, the error messages relate to the CPP file and NOT the TPL file.
  - Watch for errors during the TPL2CPP phase of the compilation.
  - Use "*cout*" to output variables to the screen so you can check that the function is being calculated correctly.
  - 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.
  - Include comments in your data files (indicated by "#").

The Second Example Problem-I

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

$$B_{t+1} = B_t + rB_t(1 - B_t / K) - C_t;\ B_{1917} = K$$

$$-\ell nL = \sum_y \left(\ell \mathrm{n} I_y - \ell \mathrm{n}(q\, B_t)\right)^2$$

- The data are located in the file LECT13B.DAT

The Second Example Problem-II

```
DATA_SECTION
 init_int Fyear;
 init_int Lyear;
 init_vector Catch(Fyear,Lyear);
 init_vector CPUE(Fyear,Lyear);
 INITIALIZATION_SECTION
 logr -0.6
 logq -9
 logK 8.5
PARAMETER_SECTION
 init_number logr(1);
 init_number logq(1);
 init_number logK(1);
 number r;
 number q;
 number K;
  vector Bio(Fyear,Lyear+1);
 objective_function_value objn;
```

First and last years, and the catch and CPUE data

Parameters in log-space to prevent negative values

Temporary variables

The Second Example Problem-III

```
PROCEDURE_SECTION
 int Year;
 dvariable SS;
 r = mfexp(logr);  q = mfexp(logq);
 K = mfexp(logK);
 Bio(Fyear) = K;
  for (Year=Fyear;Year<=Lyear;Year++)
  {
   Bio(Year+1) = Bio(Year) +
     r*Bio(Year)*(1.0-Bio(Year)/K) - Catch(Year);
   if (Bio(Year+1) < 1)  Bio(Year+1) = 1;
  }
 SS = 0;
for (Year=Fyear;Year<=Lyear;Year++)
  if (CPUE(Year) > 0)
   SS += square(log(q*Bio(Year)/CPUE(Year)));
 objn = SS;
 cout << SS << endl;
```

Change for negative values

559

ADMB and R

- R provides:
  - A way to implement run control for ADMB-based calculations.
  - A way to rapidly visualize output.

- http://admb-project.org/community/related-software/r
- http://www.sefsc.noaa.gov/mprager/rinter.html

ADMB Programs-I

- Input:

  mymodel.dat

- Output:

  mymodel.cor, mymodel.par, mymodel.std, mymodel.rep

- Arguments:

  -ind, -nohess, -nox, etc.

ADMB Programs-II

- To run an ADMB program (e.g. mymodel.exe), we typically:
  - Open a DOS window.
  - Type: "mymodel –ind run.txt"
- We would like to do this from R, perhaps with different input files and summarize the results for multiple runs automatically.

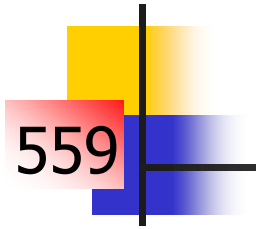
559

The Shell Command-I

- The Shell command in R invokes a system call, e.g.:
  - > shell("echo Hello world")
  - > shell("cd")
  - > shell("dir")

The Shell Command-II

- Some of the arguments of the shell command:
  - cmd: the command to be executed.
  - intern: whether to capture the output as an R string.
  - invisible: hide the DOS window.
  - wait: suspend R until the command finishes (default=T).



Reading Text Files Using R

Key Commands:

Scan() – reads a vector
ReadLines() – reads a line
Read.table() – read a table of values

Example:

```
infile <- "D:\\Test\\CatWaa.DAT"
value <- scan(infile, comment="#", quiet=TRUE)
```

Writing Text Files Using R

Key Commands:

write() – writes a vector
write.table() – writes a table of values

Example:

```
outfile <- "C:\\andre\\backs\\Test.All"
 write("# Original", outfile)
 write(value, outfile, append=T)
 write("# Twice", outfile, append=T)
 write(2*value, outfile, append=T)
```

File Management in R

getwd() – what is the current working directory
setwd(dir) – specify the current working directory
file.remove(file) – delete a file
list.files(dir) – list the contents of directory
file.copy(from, to) – copy a file
file.rename(from, to) – rename a file
file.append(file1,file2) – add file2 to file1.
file.exists(file) – does the file exist?

The file.copy function has an "overwrite=T" option, use it!