B: Introduction to TMB

Fish 559; Day 1: 15h30-17h30

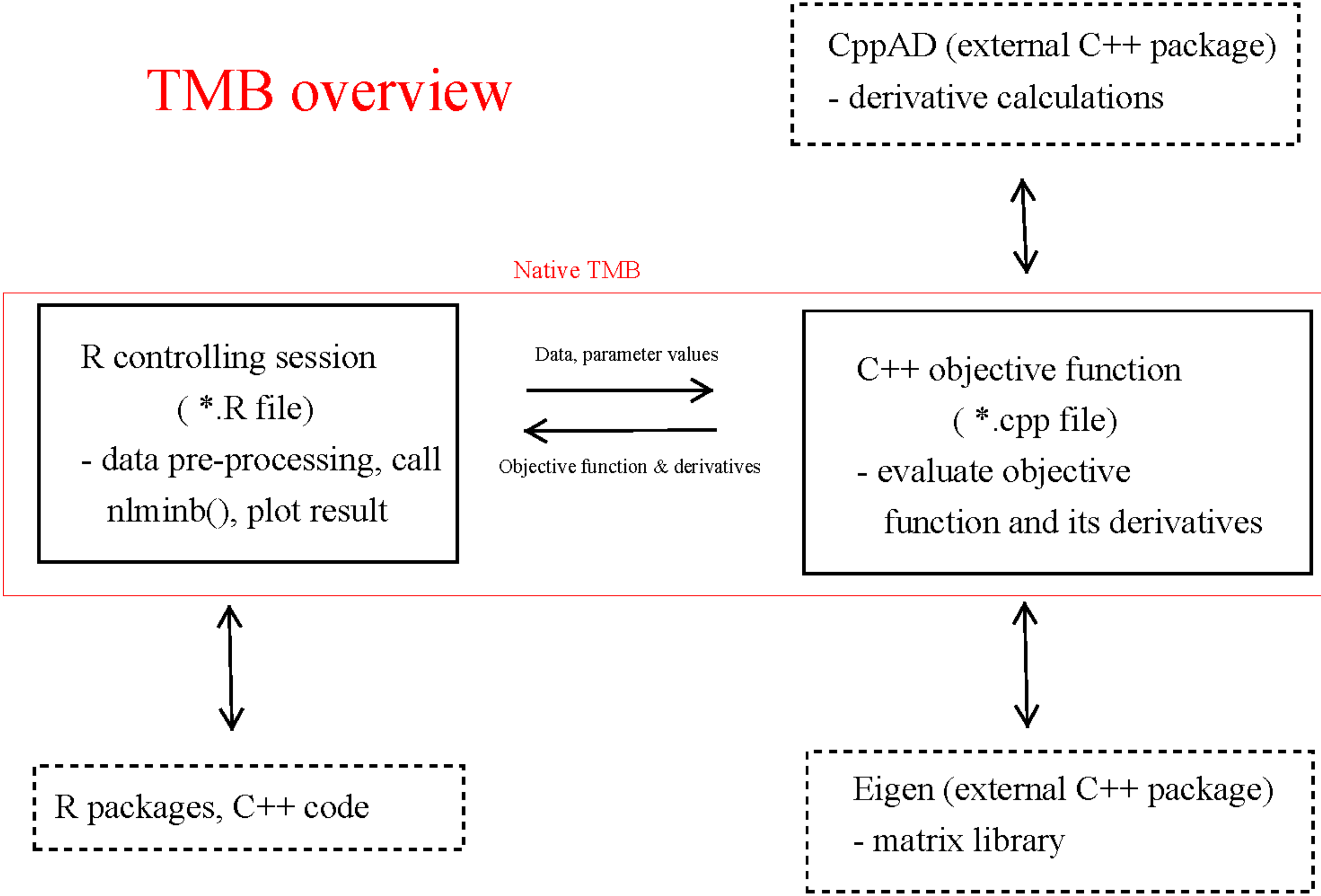
Specifying the Problem

- TMB programs are written using a template (stored in a CPP file).
- The CPP 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 function is minimized from R.

The TMB “Approach”

- Read in the data
- Create R lists that specify the data and the parameters.
- Create a .CPP file. This file provides the specifications to:
  - identify the parameters and any derived variables;
  - define the model; and
  - define the objective function to be minimized.
- Compile and link the resultant C++ program as you would any other C++ program (done through R).
- Link the compiled CPP code
- Run the analysis and analyze the results.

TMB overview



Taken from https://github.com/James-Thorson/2016_Spatio-temporal_models

559

Installing TMB-I

TMB is available is from githib:

https://github.com/kaskr/adcomp

- Download the ZIP file and UNZIP it to a folder call "adcomp". You can clone the web-site using:

  git clone https://github.com/kaskr/adcomp

- I downloaded TMB from CRAN

- Hint: You may need to install Rtools (if so – do this FIRST)

559

Installing TMB-II

- OR install from R:

```
install.packages("TMB")
library(TMB)

#test that TMB is working:
runExample(all=TRUE)
```

- Hint: You may need to install Rtools (if so – do this FIRST)

Installing Rtools

- Hint: You may need to install Rtools (if so – do this FIRST)
  https://cran.r-project.org/bin/windows/Rtools/
- When installing with the Install Wizard (Windows), check the box that allows the installer to modify your PATH
- Restart R after installing Rtools.

The R link into things

```
setwd("D:\\courses\\FISH 559_18\\Tmb\\")
data <- list(x=rivers)
parameters <- list(mu=0,logSigma=0)

require(TMB)
compile('LectB1.cpp')
dyn.load(dynlib('LectB1'))

model <- MakeADFun(data,parameters)
fit   <- nlminb(model$par, model$fn, model$gr)
rep   <- sdreport(model)
```

List of Data

List of Estimated Parameters

Create a DLL

Standard call to nlminb

This code can be found in "LectB1.R"

559

Overview of the Process

- Write the model and likelihood in XX.CPP
- Define the data inputs as a list and the parameters as a list
- Compile the cpp file.
- Construction a model object using "MakeADFun"
- Minimize the function (e.g. using nlminb)
- Look at the results.

The First Real Example - I

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

$$SS = \sum_i \left(y_i - (b_0 + b_1 x_i) \right)^2$$

- Program: LECTB2.CPP and LECTB2.R
- Data File: LECTB2.DAT

- Have a look at the various files

The First Real Example - II

R code

```
data <- read.table("LectB2.dat", header=TRUE)
parameters <- list(b0=0, b1=0, logSigma=0)
require(TMB)
compile("LectB2.cpp")
dyn.load(dynlib("LectB2"))

model <- MakeADFun(data, parameters,
DLL="LectB2",silent=T)

fit <- nlminb(model$par, model$fn, model$gr)
best <- model$env$last.par.best
rep <- sdreport(model)
print(best)
print(rep)
```

Specify parameters

Compile and load

Create a model object

Fit the model and extract output

Note the file names are case-sensitive

The First Real Example - III

```
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(x);
  DATA_VECTOR(y);
  int n = y.size();
PARAMETER(b0);
  PARAMETER(b1);
  PARAMETER(logSigma);
  vector<Type> yfit(n);
  Type neglogL = 0.0;
  yfit = b0 + b1*x;
  neglogL = -sum(dnorm(y, yfit, exp(logSigma), true));
  return neglogL;
}
```

Specify data

Specify parameters

Temp storage

Negative log-likelihood

Dissecting The Program (arithmetic)

```
yfit = b0 + b1*x;
neglogL = -sum(dnorm(y, yfit, exp(logSigma), true));
```

- The first line is a vector operation - it defines a vector, each element of which, yfit(i), is *b0+b1*x(i)*.
- The second line sums the likelihood (assumed to be normal) of the observed (*y*) data given the predicted (*yfit*) values – this is also a vector operation and follows the R formal.

Understanding MakeADFun

- **data**: the data to be passed to the function (as a list)
- **parameters**: a list of estimable parameters (including random and fixed effects)
- **map**: provides a way to fix parameters (not estimate, or make parameters equal)
- **random**: which variables are random
- **DLL**: name of the DLL

Two phases (and map)

NA here means fixed

```
## Phase 1
map <- list(u=factor(rep(NA,data$nG)),log_sigma=factor(NA))
obj <- MakeADFun(data,parameters,map=map,DLL="nmix")
opt <- nlminb(obj$par,obj$fn,obj$gr)
pl <- obj$env$parList(opt$par) ## Parameter estimate after phase 1

## Phase 2
obj <- MakeADFun(data,pl,random="u",DLL="nmix")
opt <- nlminb(obj$par,obj$fn,obj$gr)
```

559

More on map

Lets say you have an vector beta and you want the 2nd and 4th elements to be same. You can achieve this using:

```
map(beta=factor(c(1,2,3,2)))
```

559

Sdreport (see Schaefer.cpp)

By definition, TMB will return asymptotic variances for the parameters. These can be viewed using *sdreport(obj)*.

To compute standard errors for derived quantities, you need to define an ADREPORT variable.

```
Type gamma = alpha*alpha;
ADREPORT(gamma);
```

Getting variance information

- To extract the Hessian matrix
- model$he()

- To extract the variance-covariance matrix for "model":
  solve(model$he())

- To extract the correlation matrix for "model"
  cov2cor(solve(model$he()))

- To extract the standard errors for "model"
  sqrt(diag(solve(model$he())))

559

Reporting-I

To simply report results back to R (no variances), use REPORT.

```
Type delta = alpha*alpha*alpha;
REPORT(delta);
```

To access the results being reported back to R use:

```
obj$report();
```

559

Reporting-II

To access the parameter estimates (fixed and random) use:

```
rep <- sdreport(obj)

summary(rep,select="fixed")
summary(rep,select="random")

xx <- summary(rep)
Use <- row.names(xx) == "SSB"
SSB <- xx[Use,]
```

Reporting-III

To report results back to R (with variances), use ADREPORT.

```
REPORT(CW);
REPORT(N);
REPORT(S);
REPORT(SigmaR);
REPORT(Paa);

ADREPORT(FullF);
```

Something About Arithmetic

- The following are the TMB arithmetic expressions:
  - Z=a+b – addition (note Z+=b is Z=Z+b)
  - Z=a-b – subtraction
  - Z=a*b – multiplication
  - Z=a/b – division
- TMB implements all the standard functions (and more):
  - Z=log(a)
  - Z=exp(a)
  - Z=square(a)

Declaring Variables

- Before a variable is used, it has to be declared.
- Key distinction among variables:
  - Data that are passed from R (declared in the form DATA_XX(var_name)).
  - The parameters to be estimated (declared in the form PARAMETER(var_name)). Note NO integer parameters!
  - Temporary variables (declared in the form “Type xx”)

Data Types

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

- The most-basic types:
  - DATA_INTEGER (e.g. int Count) – integer.
  - DATA_IVECTOR – vector of integers.
  - DATA_IMATRIX – matrix of integers.
  - DATA_IARRAY – array of integers.
  - DATA_SCALAR (e.g. number) – real.
  - DATA_VECTOR – vector of reals.
  - DATA_MATRIX – matrix of reals.
  - DATA_ARRAY – array of reals.
  - DATA_STRUC – A structure.
  - DATA_STRING – A string.
- All variables MUST be declared before they are used – no exemptions!

559

Local variables

- Type p; [scalar]
- vector<Type> q(5); [vector]
- matrix<Type> z(5,5); [matrix]
- array<Type>k(5,5,5); [array]
- int I; [integer]
- vector<int>jj(5); [integer vector]

559

3d arrays

This is a little complicated – you first declare a vector of matrices:

```
vector<matrix<Type> > x(3);
```

Then declare the matrices themselves:

```
matrix<Type> Temp(3,4);
```

Then assign the matrices to each element:

```
X(1) = Temp; x(2) = Temp;
```

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).
  - TMB 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!
  - All arrays start at 0 (this is C), which can be confusing for R users.

559

Getting more information

- The Wiki on the web-site provides several useful hints, especially for people who are already familiar with R and ADMB
  - The snippets are really helpful!
  - Look at how to create packages