Data input/output

TMB and stock assessment

Arni Magnusson
Anders Nielsen

ICES, 2–6 Nov 2015

Data flow

file → **R** → **TMB** → **R** → **file**

read *list, declare* *report* *write*

Read into R

file → **R** → TMB → R → file

Read into R

file → **R** → TMB → R → file

Text file

- `read.table`
- `read.csv`
- `scan`

Read into R

file → **R** → TMB → R → file

Text file

`read.table`

`read.csv`

`scan`

Sometimes no data file

simulated data, URL, database, example dataset, etc.

Passing data to TMB

file → **R** → **TMB** → R → file

Passing data to TMB

file → **R** → **TMB** → R → file

R

```
list(x=10, y=c(1.1, 2.2))
```

Passing data to TMB

file → **R** → **TMB** → R → file

R

```
list(x=10, y=c(1.1, 2.2))
```

TMB

```
DATA_INTEGER(x)
DATA_VECTOR(y)
```

Passing data to TMB

file → **R** → **TMB** → R → file

R

```
list(x=10, y=c(1.1, 2.2))
```

TMB

```
DATA_INTEGER(x)
DATA_VECTOR(y)
```

```
data <- list()
data$x <- 10
data$y <- c(1.1, 2.2)
data$z <- mymatrix
```

Passing data to TMB

file → **R** → **TMB** → R → file

R

```
list(x=10, y=c(1.1, 2.2))
```

```
data <- list()
data$x <- 10
data$y <- c(1.1, 2.2)
data$z <- mymatrix
```

TMB

```
DATA_INTEGER(x)
DATA_VECTOR(y)
```

```
DATA_INTEGER(x)
DATA_VECTOR(y)
DATA_ARRAY(z)
```

TMB data types

Integer	`DATA_INTEGER`	`DATA_IVECTOR`	`DATA_IARRAY`
Double	`DATA_SCALAR`	`DATA_VECTOR`	`DATA_ARRAY`

TMB data types

Integer	`DATA_INTEGER`	`DATA_IVECTOR`	`DATA_IARRAY`
Double	`DATA_SCALAR`	`DATA_VECTOR`	`DATA_ARRAY`

Matrix			`DATA_MATRIX`
Sparse			`DATA_SPARSE_MATRIX`
Factor		`DATA_FACTOR`	

Exercise

Multiple linear regression in R

```
lm(1/mpg ~ wt + hp, data=mtcars)
```

Now implement this model in TMB

Size

Automatic size allocation

```
data <- list()
data$x <- 10                    DATA_INTEGER(x)
data$y <- c(1.1, 2.2)           DATA_VECTOR(y)
data$z <- mymatrix              DATA_ARRAY(z)
```

Size

Automatic size allocation

```
data <- list()
data$x <- 10                    DATA_INTEGER(x)
data$y <- c(1.1, 2.2)           DATA_VECTOR(y)
data$z <- mymatrix              DATA_ARRAY(z)
```

Size can be accessed within C++

Length	`.size()`
Rows	`.dim[0]`
Columns	`.dim[1]`

Indexing from 0

In C++ the first element is **number 0**

R

```
data <- list()
data$y <- c(1.1, 2.2)
data$z <- mymatrix

y[1] ... y[n]

z[1,1] ... z[m,n]
```

TMB

```
DATA_VECTOR(y)
DATA_ARRAY(z)
```

Indexing from 0

In C++ the first element is **number 0**

R

```
data <- list()
data$y <- c(1.1, 2.2)
data$z <- mymatrix

y[1] ... y[n]

z[1,1] ... z[m,n]
```

TMB

```
DATA_VECTOR(y)
DATA_ARRAY(z)

y(0) ... y(n-1)
```

Indexing from 0

In C++ the first element is **number 0**

R

```
data <- list()
data$y <- c(1.1, 2.2)
data$z <- mymatrix

y[1] ... y[n]

z[1,1] ... z[m,n]
```

TMB

```
DATA_VECTOR(y)
DATA_ARRAY(z)

y(0) ... y(n-1)

z(0,0) ... z(m-1,n-1)
```

Report to R

file → R → **TMB** → **R** → file

Report to R

file → R → **TMB** → **R** → file

Estimated parameters, derived quantities, random effects, SE

Report to R

file → R → **TMB** → **R** → file

Estimated parameters, derived quantities, random effects, SE

TMB	**R**
`ADREPORT(y)`	`sdreport(model)` `summary(sdreport(model))`
`REPORT(x)`	`model$report()`

Parameter list

Convert estimates and SE from tabular to **list format**

```
pl <- model$env$parList()
jointrep <- sdreport(model, getJointPrecision=TRUE)
allsd <- sqrt(diag(solve(jointrep$jointPrecision)))
plsd <- model$env$parList(par=allsd)
```

Becomes useful when we work with larger models

Write to file

file → R → TMB → **R** → **file**

Write to file

file → R → TMB → **R** → **file**

Text file

- `write.table`
- `write.csv`
- `write`