

Natural Mortality Modelling

Assessment for All initiative(a4a)

16 February, 2017

The document explains the approach being developed by a4a to integrate uncertainty in natural mortality into stock assessment and advice. It presents a mixture of text and code, where the first explains the concepts behind the methods, while the last shows how these can be run with the software provided.

Required packages

To follow this tutorial you should have installed the following packages:

- CRAN: copula, triangle, coda, XML, reshape2, latticeExtra
- FLR: FLCore, FLa4a

You can do so as follows,

```
install.packages(c("copula", "triangle", "coda", "XML", "reshape2", "latticeExtra"))  
# from FLR  
install.packages(c("FLCore", "FLa4a"), repos="http://flr-project.org/R")
```

```
# This chunk loads all necessary packages, trims pkg messages  
library(FLa4a)  
library(XML)  
library(reshape2)  
library(latticeExtra)  
# datasets  
data(ple4)  
data(ple4.indices)  
data(ple4.index)  
data(rfLen)
```

Background

In the a4a natural mortality is dealt with as an external parameter to the stock assessment model. The rationale to modelling natural mortality is similar to that of growth: one should be able to grab information from a range of sources and feed it into the assessment.

The mechanism used by a4a is to build an interface that makes it transparent, flexible and hopefully easy to explore different options. In relation to natural mortality it means that the analyst should be able to use distinct models like Gislason's, Charnov's, Pauly's, etc in a coherent framework making it possible to compare the outcomes of the assessment.

Within the a4a framework, the general method for inserting natural mortality in the stock assessment is to:

- Create an object of class **a4aM** which holds the natural mortality model and parameters.
- Add uncertainty to the parameters in the **a4aM** object.
- Apply the **m()** method to the **a4aM** object to create an age or length based **FLQuant** object of the required dimensions.

The resulting **FLQuant** object can then be directly inserted into an **FLStock** object to be used for the assessment.

In this section we go through each of the steps in detail using a variety of different models.

For more information on the a4a methodologies refer to Jardim, et.al, 2014, Millar, et.al, 2014 and Scott, et.al, 2016.

a4aM - The M class

Natural mortality is implemented in a class named **a4aM**. This class is made up of three objects of the class **FLModelSim**. Each object is a model that represents one effect: an age or length effect, a scaling (level) effect and a time trend, named **shape**, **level** and **trend**, respectively. The impact of the models is multiplicative, i.e. the overall natural mortality is given by **shape** x **level** x **trend**. Check the help files for more information.

```
showClass("a4aM")
```

```
## Class "a4aM" [package "FLa4a"]
##
## Slots:
##
## Name:      shape      level      trend      name      desc      range
## Class: FLModelSim FLModelSim FLModelSim character character numeric
##
## Extends: "FLComp"
```

The **a4aM** constructor requires that the models and parameters are provided. The default method will build each of these models as a constant value of 1.

As a simple example, the usual “0.2” guess estimate could be set up by setting the **level** model to have a single parameter with a fixed value, while the other two models, **shape** and **trend**, have a default value of 1 (meaning that they have no effect).

```
mod02 <- FLModelSim(model=~a, params=FLPar(a=0.2))
m1 <- a4aM(level=mod02)
m1
```

```
## a4aM object:
##   shape: ~1
##   level: ~a
##   trend: ~1
```

More interesting natural mortality shapes can be set up using biological knowledge. The following example uses an exponential decay over ages (implying that the resulting **FLQuant** generated by the **m()** method will be age based). We also use Jensen’s second estimator as a scaling **level** model, which is based on the von Bertalanffy K parameter, $M = 1.5K$.

```
shape2 <- FLModelSim(model=~exp(-age-0.5))
level2 <- FLModelSim(model=~1.5*k, params=FLPar(k=0.4))
m2 <- a4aM(shape=shape2, level=level2)
m2
```

```
## a4aM object:
##   shape: ~exp(-age - 0.5)
##   level: ~1.5 * k
##   trend: ~1
```

Note that the **shape** model has **age** as a parameter of the model but is not set using the **params** argument.

The **shape** model does not have to be age-based. For example, here we set up a **shape** model using Gislason’s second estimator (\cdot): $M_t = K(\frac{L_{inf}}{l})^{1.5}$. We use the default **level** and **trend** models. % Current **m()** method

is not ideal for length based methods as you cannot specify length range and half-widths to make compatible with FLStockLen

```
shape_len <- FLModelSim(model=~K*(linf/len)^1.5, params=FLPar(linf=60, K=0.4))
m_len <- a4aM(shape=shape_len)
```

Another option is to model how an external factor may impact the natural mortality. This can be added through the `trend` model. Suppose natural mortality can be modelled with a dependency on the NAO index, due to some mechanism that results in having lower mortality when NAO is negative and higher when it's positive. In this example, the impact is represented by the NAO value on the quarter before spawning, which occurs in the second quarter.

We use this to make a complicated natural mortality model with an age based shape model, a level model based on K and a trend model driven by NAO, where mortality increases by 50% if NAO is positive on the first quarter.

```
# Get NAO
nao.orig <- read.table("https://www.esrl.noaa.gov/psd/data/correlation/nao.data", skip=1, nrow=62, na.s
dnms <- list(quant="nao", year=1948:2009, unit="unique", season=1:12, area="unique")
# Build an FLQuant from the NAO data
nao.flq <- FLQuant(unlist(nao.orig[, -1]), dimnames=dnms, units="nao")
# Build covar by calculating mean over the first 3 months
nao <- seasonMeans(nao.flq[, , 1:3])
# Turn into Boolean
nao <- (nao>0)
# Constructor
trend3 <- FLModelSim(model=~1+b*nao, params=FLPar(b=0.5))
shape3 <- FLModelSim(model=~exp(-age-0.5))
level3 <- FLModelSim(model=~1.5*k, params=FLPar(k=0.4))
m3 <- a4aM(shape=shape3, level=level3, trend=trend3)
m3

## a4aM object:
##   shape: ~exp(-age - 0.5)
##   level: ~1.5 * k
##   trend: ~1 + b * nao
```

Adding uncertainty to natural mortality parameters with a multivariate normal distribution

Uncertainty on natural mortality is added through uncertainty on the parameters.

In this section we'll show how to add multivariate normal uncertainty. We make use of the class `FLModelSim` method `mvnorm()`, which is a wrapper for the method `mvnorm()` distributed by the package **MASS**.

We'll create an `a4aM` object with an exponential shape, a `level` model based on k and temperature (Jensen's third estimator), and a `trend` model driven by the NAO (as above). Afterwards a variance-covariance matrix for the `level` and `trend` models will be included. Finally, create an object with 100 iterations using the `mvnorm()` method.

Create the object:

```
shape4 <- FLModelSim(model=~exp(-age-0.5))
level4 <- FLModelSim(model=~k^0.66*t^0.57, params=FLPar(k=0.4, t=10), vcov=array(c(0.002, 0.01, 0.01, 1)
trend4 <- FLModelSim(model=~1+b*nao, params=FLPar(b=0.5), vcov=matrix(0.02))
m4 <- a4aM(shape=shape4, level=level4, trend=trend4)
```

```
# Call mvrnorm()
m4 <- mvrnorm(100, m4)
m4
```

```
## a4aM object:
##   shape: ~exp(-age - 0.5)
##   level: ~k^0.66 * t^0.57
##   trend: ~1 + b * nao
```

Inspect the level model (for example):

```
m4@level

## An object of class "FLModelSim"
## Slot "model":
## ~k^0.66 * t^0.57
##
## Slot "params":
## An object of class "FLPar"
## iters: 100
##
## params
##           k           t
## 0.39811(0.0457) 9.87074(0.8623)
## units: NA
##
## Slot "vcov":
##      [,1] [,2]
## [1,] 0.002 0.01
## [2,] 0.010 1.00
##
## Slot "distr":
## [1] "norm"
```

Note the variance in the parameters:

```
params(trend(m4))

## An object of class "FLPar"
## iters: 100
##
## params
##           b
## 0.50488(0.136)
## units: NA
```

Note the shape model has no parameters and no uncertainty:

```
params(shape(m4))

## An object of class "FLPar"
## [1] NA
## units: NA
```

In this particular case, the **shape** model will not be randomized because it doesn't have a variance-covariance matrix. Also note that because there is only one parameter in the **trend** model, the randomization will use a univariate normal distribution.

The same model could be achieved by using **mnrnorm()** on each model component:

```
m4 <- a4aM(shape=shape4, level=mvrnorm(100, level4), trend=mvrnorm(100, trend4))
```

%Note: How to include ageing error ???

Adding uncertainty to natural mortality parameters with statistical copulas

We can also use copulas to add parameter uncertainty to the natural mortality model, similar to the way we use them for the growth model. Using a triangle distribution. We use the package triangle, where this distribution is parametrized using the minimum, maximum and median values. This can be very attractive if the analyst needs to scrape information from the web or literature and perform some kind of meta-analysis. As stated above these processes make use of the methods implemented for the `FLModelSim` class.

% EXPAND...

In the following example we'll use again Gislason's second estimator, $M_l = K(\frac{L_{inf}}{l})^{1.5}$ and a triangle copula to model parameter uncertainty. The method `mvrtriangle()` is used to create 1000 iterations.

```
linf <- 60
k <- 0.4
# vcov matrix (make up some values)
mm <- matrix(NA, ncol=2, nrow=2)
# 10% cv
diag(mm) <- c((linf*0.1)^2, (k*0.1)^2)
# 0.2 correlation
mm[upper.tri(mm)] <- mm[lower.tri(mm)] <- c(0.05)
# a good way to check is using cov2cor
cov2cor(mm)

##           [,1]      [,2]
## [1,] 1.0000 0.2083
## [2,] 0.2083 1.0000

# create object
mgis2 <- FLModelSim(model=~k*(linf/len)^1.5, params=FLPar(linf=linf, k=k), vcov=mm)
# set the lower, upper and (optionally) centre of the parameters (without the centre, the triangle is s
pars <- list(list(a=55,b=65), list(a=0.3, b=0.6, c=0.35))
mgis2 <- mvrtriangle(1000, mgis2, paramMargins=pars)
mgis2

## An object of class "FLModelSim"
## Slot "model":
## ~k * (linf/len)^1.5
##
## Slot "params":
## An object of class "FLPar"
## iters: 1000
##
## params
##           linf           k
## 59.94225(2.1905) 0.41233(0.0751)
## units: NA
##
## Slot "vcov":
```

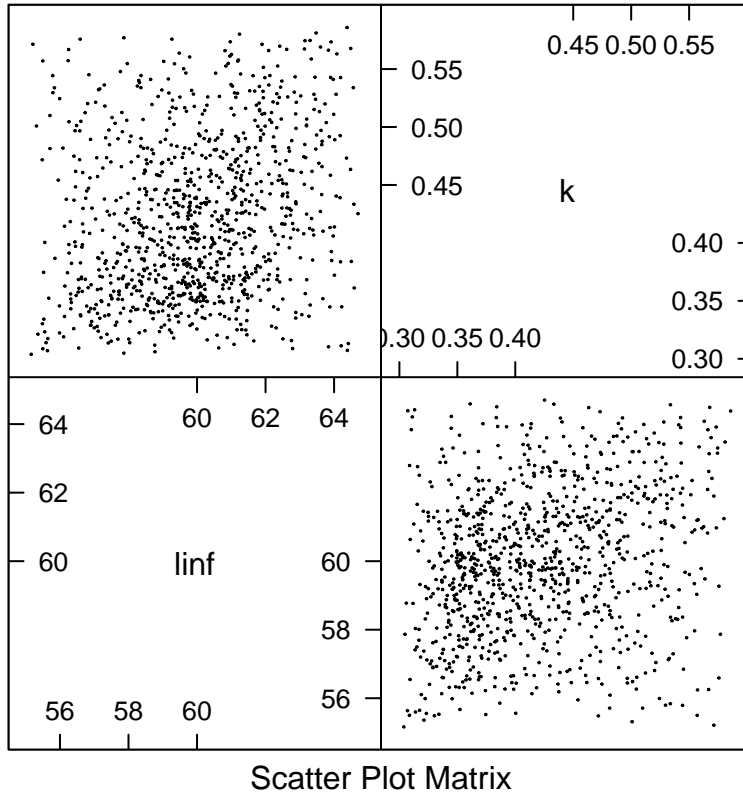


Figure 1: Parameter estimates for Gislason’s second natural mortality model from using a triangle distribution.

```
##      [,1] [,2]
## [1,] 36.00 0.0500
## [2,]  0.05 0.0016
##
## Slot "distr":
## [1] "un t copula family triangle"
```

The resulting parameter estimates and marginal distributions can be seen in Figure 1 and Figure 2.

We now have a new model that can be used for the `shape` model. You can use the constructor or the `set` method to add the new model. Note that we have a quite complex method now for `M`. A length based `shape` model from Gislason’s work, Jensen’s third model based on temperature `level` and a time `trend` depending on NAO. All of the component models have uncertainty in their parameters.

```
m5 <- a4aM(shape=mgis2, level=level4, trend=trend4)
# or
m5 <- m4
shape(m5) <- mgis2
```

Computing natural mortality time series - the “m” method

Now that we have set up the natural mortality `a4aM` model and added parameter uncertainty to each component, we are ready to generate the `FLQuant` of natural mortality. For that we need the `m()` method.

The `m()` method is the workhorse method for computing natural mortality. The method returns an `FLQuant`

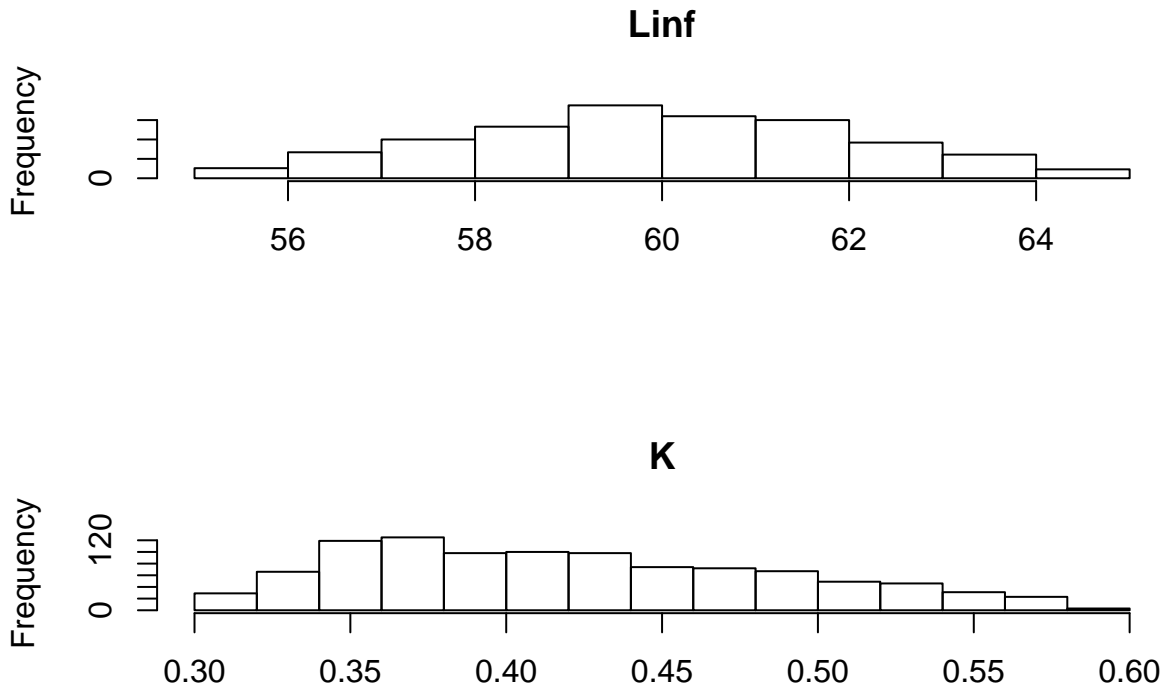


Figure 2: Marginal distributions of the parameters for Gislason’s second natural mortality model using a triangle distribution.

that can be inserted in an `FLStock` for usage by the assessment method.

%The method uses the `range` slot to work out the dimensions of the `FLQuant` object. % Future developments will also allow for easy insertion into `FLStockLen` objects.

The size of the `FLQuant` object is determined by the `min`, `max`, `minyear` and `maxyear` elements of the `range` slot of the `a4aM` object. By default the values of these elements are set to 0. Giving an `FLQuant` with length 1 in the `quant` and `year` dimension. The `range` slot can be set by hand, or by using the `rngquant()` and `rngyear()` methods.

The name of the first dimension of the output `FLQuant` (e.g. ‘age’ or ‘len’) is determined by the parameters of the `shape` model. If it is not clear what the name should be then the name is set to ‘quant’.

Here we demonstrate `m()` using the simple `a4aM` object we created above that has constant natural mortality.

Start with the simplest model:

```
m1
```

```
## a4aM object:
##   shape: ~1
##   level: ~a
##   trend: ~1
```

Check the range:

```
range(m1)
```

```
##      min      max plusgroup  minyear  maxyear  minmbar  maxmbar
##      0       0       0         0         0         0         0
```

Simple - no ages or years:

```
m(m1)
```

```
## An object of class "FLQuant"  
## , , unit = unique, season = all, area = unique  
##  
##      year  
## quant 0  
##      0 0.2  
##  
## units:  NA
```

Set the quant and year ranges:

```
rngquant(m1) <- c(0,7)      # set the quant range  
rngyear(m1) <- c(2000, 2010) # set the year range  
range(m1)
```

```
##      min      max plusgroup  minyear  maxyear  minmbar  maxmbar  
##      0       7       0      2000     2010       0       0
```

Create the object with the M estimates by age and year, note the name of the first dimension is 'quant'.

```
m(m1)
```

```
## An object of class "FLQuant"  
## , , unit = unique, season = all, area = unique  
##  
##      year  
## quant 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010  
##      0 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##      1 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##      2 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##      3 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##      4 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##      5 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##      6 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##      7 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  
##  
## units:  NA
```

The next example has an age-based shape. As the **shape** model has 'age' as a variable which is not included in the **FLPar** slot it is used as the name of the first dimension of the resulting **FLQuant**. Note that in this case the **mbar** values in the range become relevant, once that **mbar** is used to compute the mean level. This mean level will match the value given by the **level** model. The **mbar** range can be changed with the **rngmbar()** method. We illustrate this by making an **FLQuant** with age varying natural mortality.

Check the model and set the ranges:

```
m2
```

```
## a4aM object:  
##   shape: ~exp(-age - 0.5)  
##   level: ~1.5 * k  
##   trend: ~1
```

```
rngquant(m2) <- c(0,7)  
rngyear(m2) <- c(2000, 2003)  
range(m2)
```



```
##      min      max plusgroup  minyear  maxyear  minmbar  maxmbar
##      0        7        0      2000      2003        0        0
```

```
m(m2)
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## age 2000      2001      2002      2003
##  0 0.60000000 0.60000000 0.60000000 0.60000000
##  1 0.22072766 0.22072766 0.22072766 0.22072766
##  2 0.08120117 0.08120117 0.08120117 0.08120117
##  3 0.02987224 0.02987224 0.02987224 0.02987224
##  4 0.01098938 0.01098938 0.01098938 0.01098938
##  5 0.00404277 0.00404277 0.00404277 0.00404277
##  6 0.00148725 0.00148725 0.00148725 0.00148725
##  7 0.00054713 0.00054713 0.00054713 0.00054713
##
## units:  NA
```

Note that the level value is:

```
predict(level(m2))
```

```
##      iter
##      1
##  1 0.6
```

Which is the same as:

```
m(m2)["0"]
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## age 2000 2001 2002 2003
##  0 0.6  0.6  0.6  0.6
##
## units:  NA
```

This is because the mbar range is currently set to “0” and “0” (see above) and the mean natural mortality value over this range is given by the level model.

We can change the mbar range:

```
rngmbar(m2)<- c(0,5)
range(m2)
```

```
##      min      max plusgroup  minyear  maxyear  minmbar  maxmbar
##      0        7        0      2000      2003        0        5
```

Which rescales the the natural mortality at age:

```
m(m2)
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
```

```
##      year
## age 2000      2001      2002      2003
##   0 2.2812888 2.2812888 2.2812888 2.2812888
##   1 0.8392392 0.8392392 0.8392392 0.8392392
##   2 0.3087389 0.3087389 0.3087389 0.3087389
##   3 0.1135787 0.1135787 0.1135787 0.1135787
##   4 0.0417833 0.0417833 0.0417833 0.0417833
##   5 0.0153712 0.0153712 0.0153712 0.0153712
##   6 0.0056547 0.0056547 0.0056547 0.0056547
##   7 0.0020803 0.0020803 0.0020803 0.0020803
##
## units:  NA
```

Check that the mortality over the mean range is the same as the level model:

```
quantMeans(m(m2)[as.character(0:5)])
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## age 2000 2001 2002 2003
##   all 0.6  0.6  0.6  0.6
##
## units:  NA
```

The next example uses a time trend for the `trend` model. We use the `m3` model we made earlier. The `trend` model for this model has a covariate, 'nao'. This needs to be passed to the `m()` method. The year range of the 'nao' covariate should match that of the `range` slot.

Simple, pass in a single nao value (only one year):

```
m(m3, nao=1)
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## age 0
##   0 0.9
##
## units:  NA
```

Set ages:

```
rngquant(m3) <- c(0,7)
m(m3, nao=0)
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## age 0
##   0 0.60000000
##   1 0.22072766
##   2 0.08120117
##   3 0.02987224
##   4 0.01098938
```

```
## 5 0.00404277
## 6 0.00148725
## 7 0.00054713
##
## units: NA
```

With ages and years - passing in the NAO data as numeric (1,0,1,0)

```
rngyear(m3) <- c(2000, 2003)
m(m3, nao=as.numeric(nao[,as.character(2000:2003)]))
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##   year
## age 2000      2001      2002      2003
## 0 0.90000000 0.60000000 0.90000000 0.60000000
## 1 0.33109150 0.22072766 0.33109150 0.22072766
## 2 0.12180175 0.08120117 0.12180175 0.08120117
## 3 0.04480836 0.02987224 0.04480836 0.02987224
## 4 0.01648407 0.01098938 0.01648407 0.01098938
## 5 0.00606415 0.00404277 0.00606415 0.00404277
## 6 0.00223088 0.00148725 0.00223088 0.00148725
## 7 0.00082069 0.00054713 0.00082069 0.00054713
##
## units: NA
```

The final example show how `m()` can be used to make an `FLQuant` with uncertainty (see Figure 3). We use the `m4` object from earlier with uncertainty on the `level` and `trend` parameters.

```
rngquant(m4) <- c(0,7)
rngyear(m4) <- c(2000, 2003)
flq <- m(m4, nao=as.numeric(nao[,as.character(2000:2003)]))
flq
```

```
## An object of class "FLQuant"
## iters: 100
##
## , , unit = unique, season = all, area = unique
##
##   year
## age 2000      2001      2002
## 0 3.0079919(0.359714) 2.0389830(0.229792) 3.0079919(0.359714)
## 1 1.1065784(0.132331) 0.7500999(0.084536) 1.1065784(0.132331)
## 2 0.4070874(0.048682) 0.2759463(0.031099) 0.4070874(0.048682)
## 3 0.1497591(0.017909) 0.1015150(0.011441) 0.1497591(0.017909)
## 4 0.0550933(0.006588) 0.0373453(0.004209) 0.0550933(0.006588)
## 5 0.0202677(0.002424) 0.0137386(0.001548) 0.0202677(0.002424)
## 6 0.0074561(0.000892) 0.0050541(0.000570) 0.0074561(0.000892)
## 7 0.0027429(0.000328) 0.0018593(0.000210) 0.0027429(0.000328)
##   year
## age 2003
## 0 2.0389830(0.229792)
## 1 0.7500999(0.084536)
## 2 0.2759463(0.031099)
## 3 0.1015150(0.011441)
```

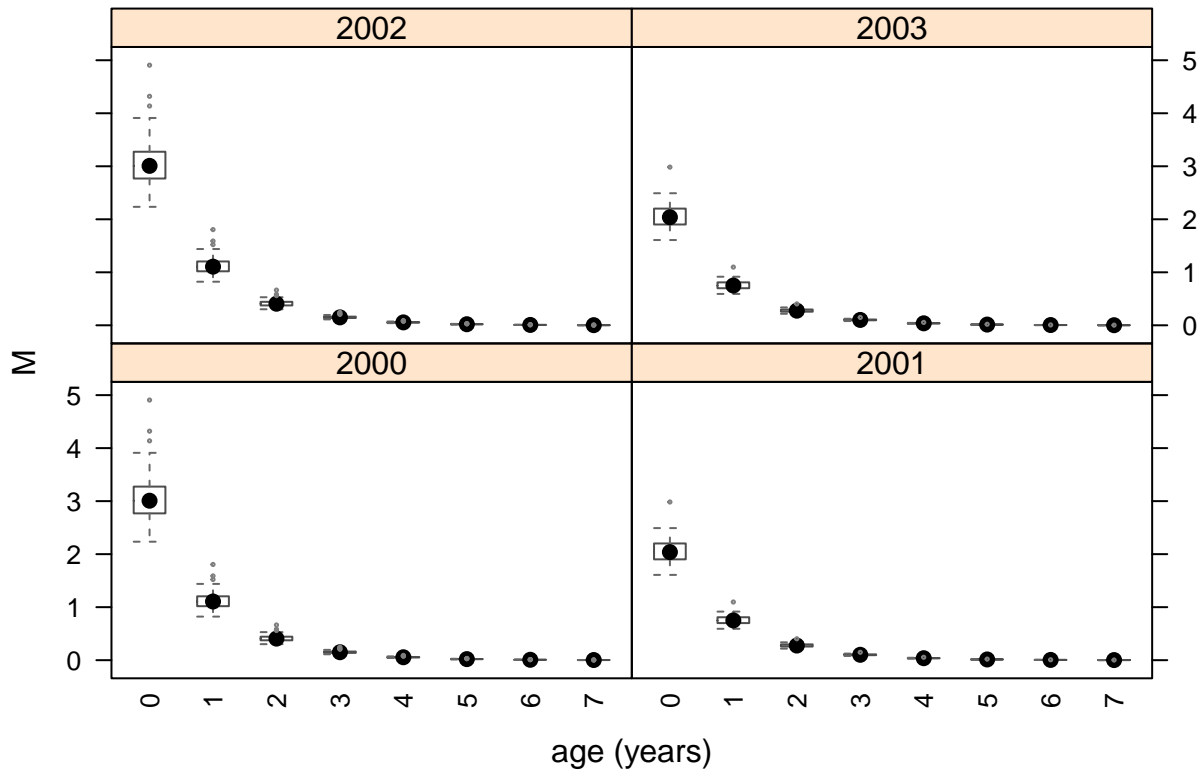


Figure 3: Natural mortality with age and year trend.

```
## 4 0.0373453(0.004209)
## 5 0.0137386(0.001548)
## 6 0.0050541(0.000570)
## 7 0.0018593(0.000210)
##
## units: NA
dim(flq)

## [1] 8 4 1 1 1 100
```

More information

Documentation can be found at (<http://flr-project.org/FLa4a>). You are welcome to:

- Submit suggestions and bug-reports at: (<https://github.com/flr/FLa4a/issues>)
- Send a pull request on: (<https://github.com/flr/FLa4a/>)
- Compose a friendly e-mail to the maintainer, see `packageDescription('FLa4a')`

Software Versions

- R version 3.3.2 (2016-10-31)
- FLCore: 2.6.0.20170214
- FLa4a: 1.0.0
- **Compiled:** Thu Feb 16 09:58:35 2017

License

This document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International license.

Author information

Ernesto Jardim. European Commission Joint Research Centre (JRC), Institute for the Protection and Security of the Citizen (IPSC), Maritime Affairs Unit, Via E. Fermi 2749, 21027 Ispra VA, Italy. <https://ec.europa.eu/jrc/99>