Stock assessment with the a4astatistical catch-at-age framework

Ernesto Jardim[1], Danai Mantopoulou Palouka[2], and Colin Millar[3]

[1]Independent Expert, Brussels, Belgium
[2]European Commission, Joint Research Centre, Sustainable resources directorate, Water and Marine Resources unit, 21027 Ispra (VA), Italy
[3]International Council for the Exploration of the Sea (ICES), H. C. Andersens Boulevard 44-46, 1553 Copenhagen V, Denmark
[*]Corresponding author ernesto.jardim@gmail.com

October 11, 2024

Abstract

This chapter presents the statistical catch-at-age stock assessment model developed as part of the Assessment For All (a4a) initiative of the European Commission Joint Research Centre. The stock assessment model framework is a non-linear catch-at-age model implemented in R, FLR and ADMB, that can be applied rapidly to a wide range of situations with low parametrization requirements. The model structure is defined by submodels, which are the different parts that require structural assumptions. There are 5 submodels in operation: F-at-age, abundance indices catchability-at-age, recruitment, observation variance of catch-at-age and abundance indices-at-age, and population's age structurein the first year. All submodels use the same type of specification process, the Rformula interface, wich gives lot's of flexibility to explore models and combination of submodels.

Contents

1 Before starting

1.1 License, documentation and development status

The software is released under the EUPL 1.1.

For more information on the a4amethodologies refer to Jardim, et.al, 2014, Millar, et.al, 2014 and Scott, et.al, 2016.

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')'

1.2 Installing and loading libraries

To run the FLa4a methods the reader will need to install the package and its dependencies and load them. Some datasets are distributed with the package and as such need to be loaded too.

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

```
# libraries
library(devtools)
library(FLa4a)
library(XML)
library(reshape2)
library(ggplotFL)
# datasets
data(ple4)
data(ple4.indices)
data(ple4.index)
data(rfLen)
```

```
packageVersion("FLCore")
```

```
## [1] '2.6.20.920'
```

```
packageVersion("FLa4a")
```

```
## [1] '1.9.0'
```

1.3 How to read this document

The target audience for this document are readers with some experience in R and some background on stock assessment.

Table 1: Mathematical notation

Symbol	Description
Variables	
C	catches
F	fishing mortality
M	natural mortality
R	recruitment
Q	vessel or fleet catchability
w	weights
l	likelihood
I	abundance index
S	spawning stock biomass
CV	coefficient of variation
D	residuals or deviances
N	normal distribution
β	parameter
a	stock-recruitment parameter
b	stock-recruitment parameter
σ^2	variance of catch
τ^2	variance of index
ϕ^2	variance of predicted recruitment
υ^2	variance of residuals
Subscripts	
a	age
y	year
C	catch
I	abundance index
N	normal distribution
s	survey
SR	stock recruitment relationship
Superscripts and accents	
$\hat{}$	observation
$\tilde{}$	prediction
c	catches
s	abundance index

The document explains the approach being developed by `a4a`for fish stock assessment and scientific 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. Moreover, having the code allows the reader to copy/paste and replicate the analysis presented here.

The sections and subsections are as independent as possible, so it can be used as a reference document for the `FLa4a`.

1.4 Notation

Along this chapter the notation presented in Table 1 will be used. Mathematical descriptions will be kept as simple as possible for readability.

2 Introduction

The a4astock assessment framework is based in a non-linear catch-at-age model implemented in R, FLRand ADMBthat can be applied rapidly to a wide range of situations with low setup requirements.

The framework is built of submodels which define the different parts of a statistical catch at age model that require structural assumptions. In the a4aframework these are fishing mortality-at-age, abundance indicies catchability-at-age, recruitment, observation variances of catch-at-age and abundance indices-at-age, and abundance-at-age in the first year of the data series (see section 3 for details).

Other important processes, like natural mortality, individual growth and reproduction, are treated as fixed(inputs??), as it's common in stock assessment methods. Nevertheless, the a4aframework provides methods to condition these processes prior to the model fit, and propagate their uncertainty into the assessment process. See chapters XX and section XX.

The submodels formulation uses linear models, which opens the possibility of using the linear modelling tools available in R. For example, mgcv gam formulas or factorial design formulas using lm().

The 'language' of linear models has been developing within the statistical community for many years, and constitutes an elegant way of defining models without going through the complexity of mathematical representations. This approach makes it also easier to communicate among scientists:

- 1965 J. A. Nelder, notation for randomized block design
- 1973 Wilkinson and Rodgers, symbolic description for factorial designs
- 1990 Hastie and Tibshirani, introduced notation for smoothers
- 1991 Chambers and Hastie, further developed for use in S

3 Stock assessment framework maths description

The stock assessment model behind the framework is based in two main equations to link observations to processes, the Baranov's catch equation and abundance indices equation.

Catches in numbers by age and year are defined in terms of the three quantities: natural mortality, fishing mortality and recruitment; using a modified form of the well known Baranov catch equation:

$$C_{ay} = \frac{F_{ay}}{F_{ay} + M_{ay}} \left(1 - e^{-(F_{ay}+M_{ay})}\right) R_y e^{-\sum(F_{ay}+M_{ay})}$$

Survey indices by age and year are defined in terms of the same three quantities with the addition of survey catchability:

$$I_{ays} = Q_{ays} R_y e^{-\sum(F_{ay}+M_{ay})}$$

Observed catches and observed survey indices are assumed to be log-normally distributed, or equivalently, normally distributed on the log-scale, with specific observation variance:

$$\log \hat{C}_{ay} \sim \text{Normal}\Big(\log C_{ay}, \sigma^2_{ay} \Big)$$
$$\log \hat{I}_{ays} \sim \text{Normal}\Big(\log I_{ays}, \tau^2_{ays} \Big)$$

The log-likelihood can now be defined as the sum of the log-likelihood of the observed catches:

$$\ell_C = \sum_{ay} w^{(c)}_{ay} \; \ell_N \Big(\log C_{ay}, \sigma^2_{ay};\; \log \hat{C}_{ay} \Big)$$

and the log-likelihood of the observed survey indices as:

$$\ell_I = \sum_{s} \sum_{ay} w^{(s)}_{ays} \; \ell_N \Big(\log I_{ays}, \tau^2_{ays};\; \log \hat{I}_{ays} \Big)$$

giving the total log-likelihood

$$\ell = \ell_C + \ell_I$$

which is defined in terms of the strictly positive quantites, M_{ay}, F_{ay}, Q_{ays} and R_y, and the observation variances σ_{ay} and τ_{ays}. As such, the log-likelihood is over-parameterised as there are many more parameters than observations. In order to reduce the number of parameters, M_{ay} is assumed known (as is common).

The remaining parameters are written in terms of a linear combination of covariates x_{ayk}, e.g.

$$\log F_{ay} = \sum_k \beta_k x_{ayk}$$

where k is the number of parameters to be estimated and is sufficiently small. Using this tecnique the quantities $\log F$, $\log Q$, $\log \sigma$ and $\log \tau$ (in bold in the equations above) can be described by a reduced number of parameters. The following section has more discussion on the use of linear models in `a4a`.

The `a4a`statistical catch-at-age model can addionally allow for a functional relationship that links predicted recruitment $\tilde{R}$ based on spawning stock biomass and modelled recruitment R, to be imposed as a fixed variance random effect. [NEEDS REVISION, sentence not clear]

Options for the relationship are the hard coded models Ricker, Beverton Holt, smooth hockeystick or geometric mean. This is implemented by including a third component in the log-

likelihood:

$$\ell_{SR} = \sum_y \ell_N\Big(\log \tilde{R}_y(a,b), \phi_y^2;\ \log R_y\Big)$$

giving the total log-likelihood

$$\ell = \ell_C + \ell_I + \ell_{SR}$$

Using the (time varying) Ricker model as an example, predicted recruitment is

$$\tilde{R}_y(a_y, b_y) = a_y S_{y-1} e^{-b_y S_{y-1}}$$

where S is spawning stock biomass derived from the model parameters F and R, and the fixed quantites M and mean weights by year and age. It is assumed that R is log-normally distributed, or equivalently, normally distributed on the log-scale about the (log) recruitment predicted by the SR model $\tilde{R}$, with known variance ϕ^2, i.e.

$$\log R_y \sim \text{Normal}\Big(\log \tilde{R}_y, \phi_y^2\Big)$$

which leads to the definition of ℓ_{SR} given above. In all cases a and b are strictly positive, and with the quantities F, R, etc. linear models are used to parameterise $\log a$ and/or $\log b$, where relevant.

By default, recruitment R as apposed to the reruitment predicted from a stock recruitment model $\tilde{R}$, is specified as a linear model with a parameter for each year, i.e.

$$\log R_y = \gamma_y$$

This is to allow modelled recruitment R_y to be shrunk towards the stock recruitment model. However, if it is considered appropriate that recruitment can be determined exactly by a relationship with covariates, it is possible, to instead define $\log R$ in terms of a linear model in the same way as $\log F$, $\log Q$, $\log \sigma$ and $\log \tau$.

4 Submodel structure

The `a4a`stock assessment framework allows the user to set up a large number of different models. The mechanics which provide this flexibility are designed around the concept of submodels. Each unknown variable that must be estimated is treated as a linear model, for which the user has to define the model structure using `R`formulas, including `mgcv` gam formulas. All submodels use the same specification process, the `R`formula interface, wich gives lot's of flexibility to explore models and combination of submodels.

There are 5 submodels in operation:

- a model for F-at-age (F_{ay}),
- a (list) of model(s) for abundance indices catchability-at-age (Q_{ays}),
- a model for recruitment (R_y),
- a list of models for the observation variance of catch-at-age and abundance indices ($\{\sigma^2_{ay}, \tau^2_{ays}\}$),
- a model for the initial age structure $N_{a,y=1}$,

When setting the structure of each submodel the user is in fact building the predictive model and its parameters. The optimization process, done through `ADMB`, estimates the parameters and their variance-covariance matrix, allowing further analysis to be carried out, like simulation, prediction, diagnostics, etc. All the statistical machinery will be at the user's reach.

4.1 Submodel building blocks and fundamental R formulas

The elements available to build submodels formulas are 'age' and 'year', which can be used to build models with different structures.

In `R`'s linear modelling language, a constant model is coded as `~1`, while a slope over time would simply be `~year`, a smoother over time `~s(year, k=10)`, a model with a coefficient for each year (also called dummy variables) would be `~factor(year)`. Transformations of the variables are as usual, e.g. `~sqrt(year)`, etc; while combinations of all the above can be done non-convergence will limit the possibilities.

Using the F submodel as example the following specifies the models described in the previous paragraph:

```
# models
m1 <- ~1
m2 <- ~year
m3 <- ~s(year, k = 10)
m4 <- ~factor(year)
m5 <- ~sqrt(year)

# fits
fit1 <- sca(ple4, ple4.indices, fmodel = m1, fit = "MP")
fit2 <- sca(ple4, ple4.indices, fmodel = m2, fit = "MP")
fit3 <- sca(ple4, ple4.indices, fmodel = m3, fit = "MP")
fit4 <- sca(ple4, ple4.indices, fmodel = m4, fit = "MP")
fit5 <- sca(ple4, ple4.indices, fmodel = m5, fit = "MP")

# plot
lst <- FLStocks(constant = ple4 + fit1, linear = ple4 + fit2, smooth = ple4 + fit3,
    factor = ple4 + fit4, sqrt = ple4 + fit5)
```

```
lst <- lapply(lst, fbar)
lgnd <- list(points = FALSE, lines = TRUE, space = "right")
xyplot(data ~ year, groups = qname, lst, auto.key = lgnd, type = "l", ylab = "fishing mortalit
```

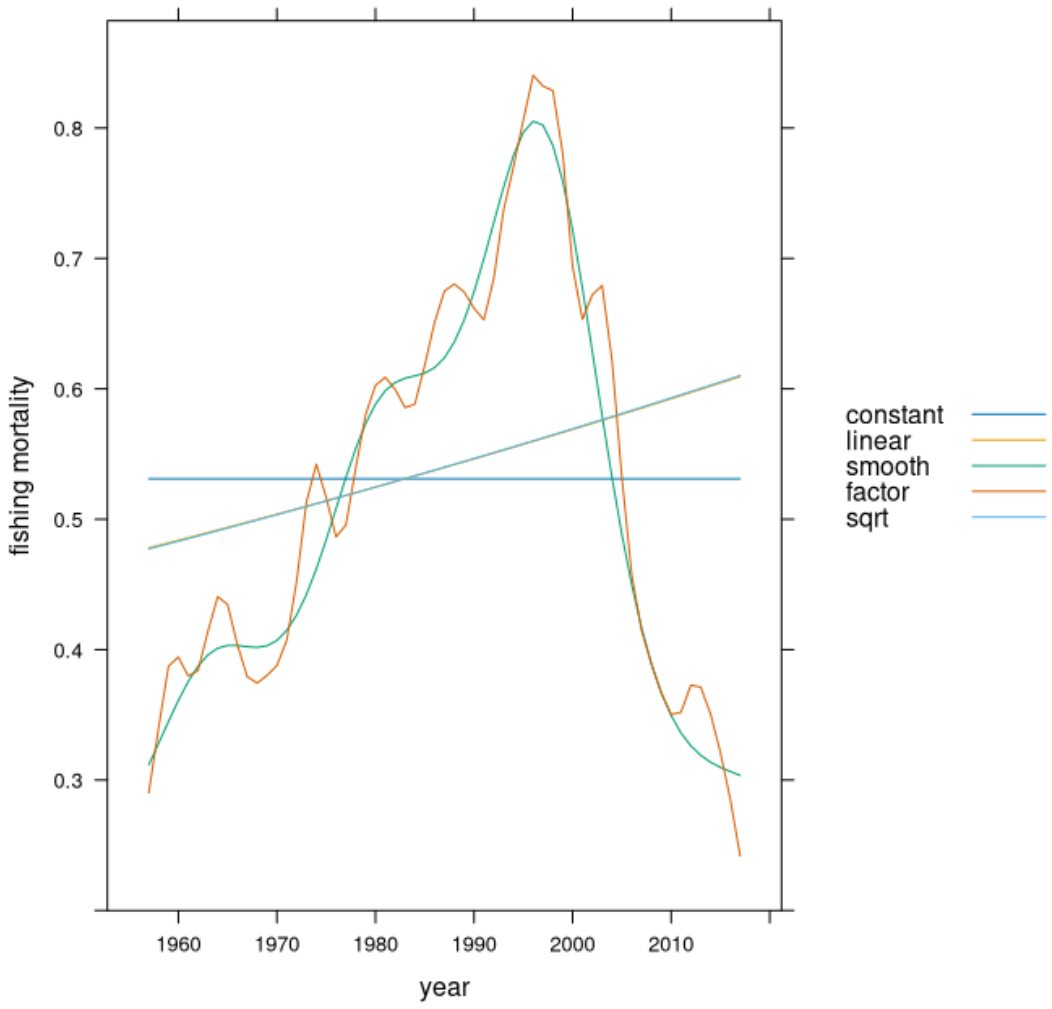


Figure 1: Example of fundamental R formulas

The models above and their combinations can be used to model both 'age' and 'year'. The corresponding fits for age are:

```
# models
m1 <- ~1
m2 <- ~age
m3 <- ~s(age, k = 3)
m4 <- ~factor(age)
m5 <- ~sqrt(age)

# fits
```

```r
fit1 <- sca(ple4, ple4.indices, fmodel = m1, fit = "MP")
fit2 <- sca(ple4, ple4.indices, fmodel = m2, fit = "MP")
fit3 <- sca(ple4, ple4.indices, fmodel = m3, fit = "MP")
fit4 <- sca(ple4, ple4.indices, fmodel = m4, fit = "MP")
fit5 <- sca(ple4, ple4.indices, fmodel = m5, fit = "MP")

# plot
lst <- FLStocks(constant = ple4 + fit1, linear = ple4 + fit2, smooth = ple4 + fit3,
    factor = ple4 + fit4, sqrt = ple4 + fit5)
lst <- lapply(lst, function(x) harvest(x)[, "2000"])
xyplot(data ~ age, groups = qname, lst, auto.key = lgnd, type = "l", ylab = "fishing mortality
```

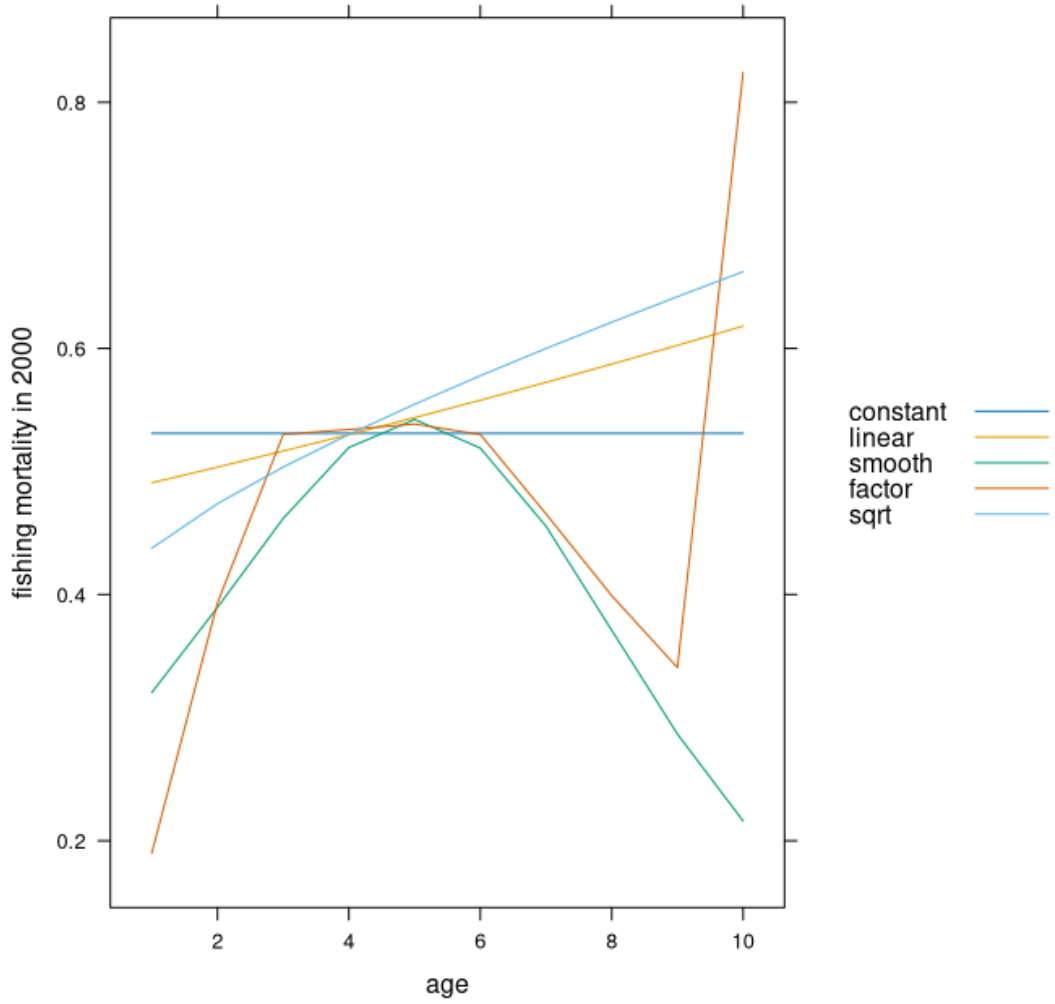


Figure 2: Example of fundamental R formulas

4.2 The major effects available for modelling

Although the building blocks for formulas are 'age' and 'year', in fact there are three effects that can be modelled for each submodel: 'age', 'year' and 'cohort'. As examples note the following models for fishing mortality.

```
# the age effect
ageeffect <- ~factor(age)

# the year effect
yeareffect <- ~factor(year)

# the cohort
cohorteffect <- ~factor(year - age)

# the fits
fit1 <- sca(ple4, ple4.indices, fmodel = yeareffect)
fit2 <- sca(ple4, ple4.indices, fmodel = ageeffect)
fit3 <- sca(ple4, ple4.indices, fmodel = cohorteffect)
```

and the graphical representation of the three models in Figures 3 to 5.

```
wireframe(harvest(fit1), main = "year effect")
```

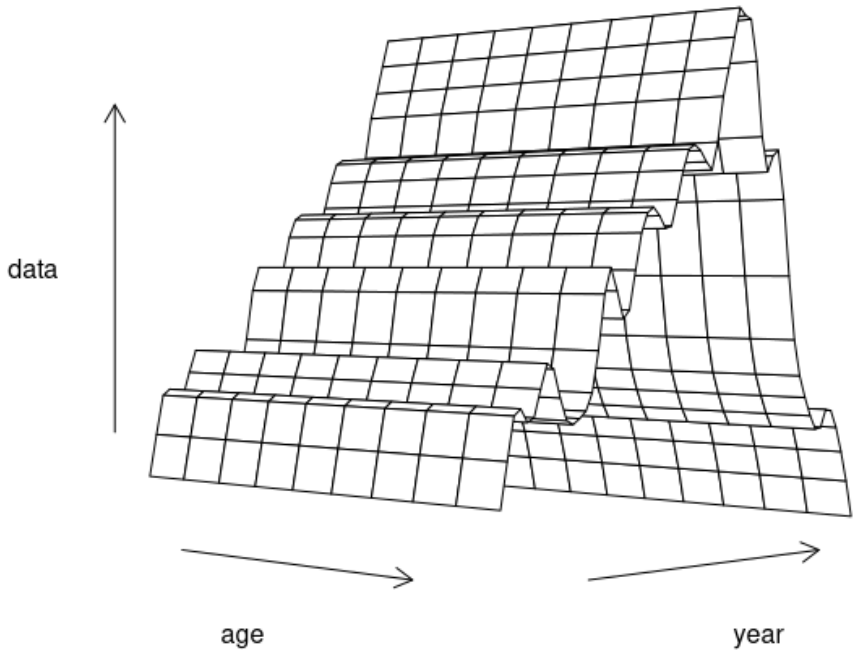


Figure 3: Major effects: the year effect (factor(year))

```
wireframe(harvest(fit2), main = "age effect")
```

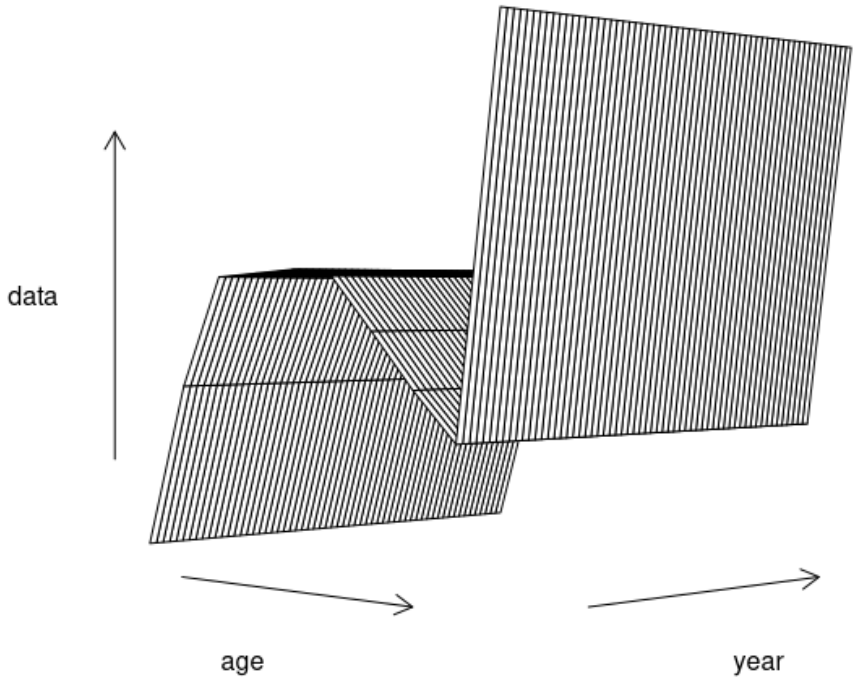


Figure 4: Major effects: the age effect (factor(age))

```
wireframe(harvest(fit3), main = "cohort effect")
```

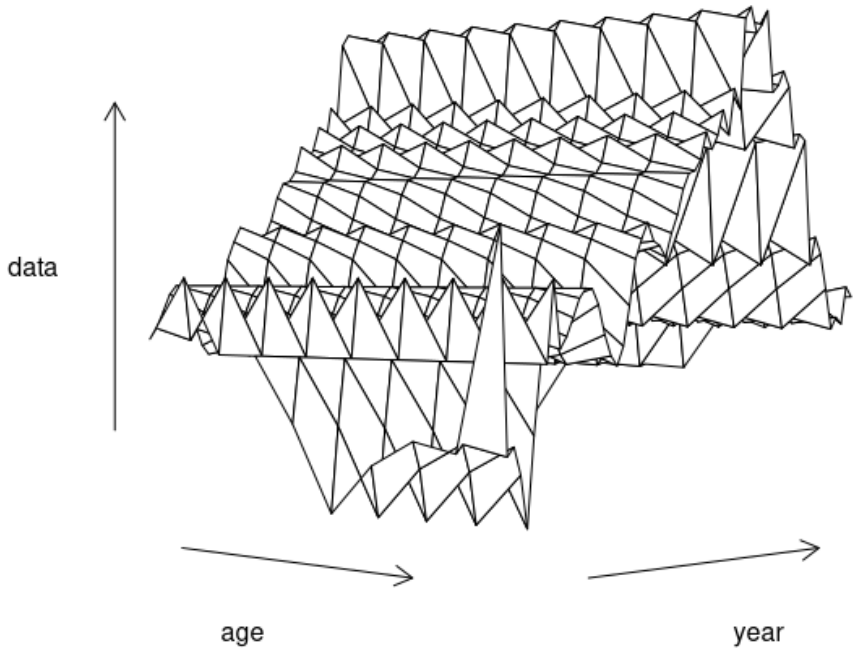


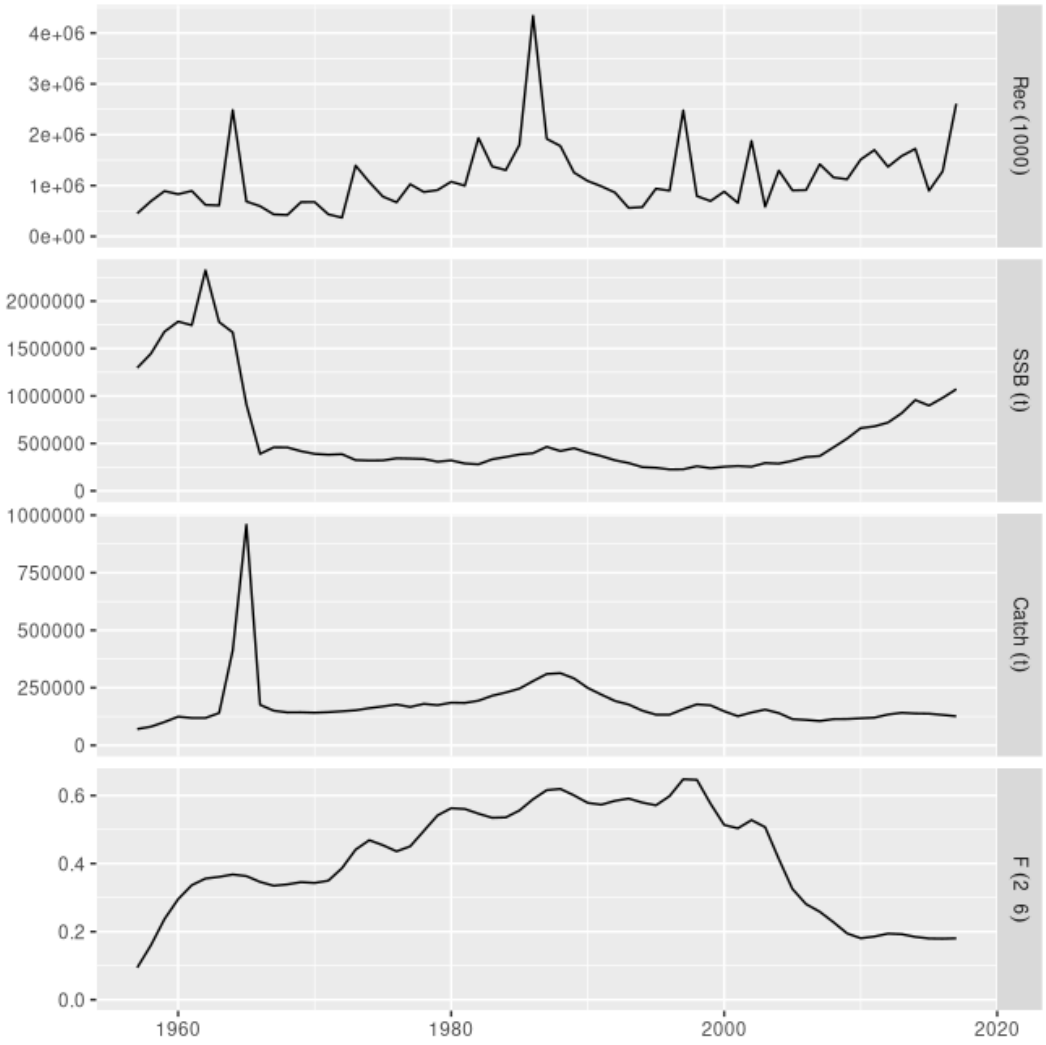
Figure 5: Major effects: the cohort effect (factor(year-age))

4.3 The submodel class and methods

5 The statistical catch-at-age stock assessment framework

The a4astock assessment framework is implemented in Rthrough the method sca(). The method call requires as a minimum a FLStock object and a FLIndices object, in which case the default submodels will be set by the method.

```
data(ple4)
data(ple4.indices)
fit <- sca(ple4, ple4.indices)
stk <- ple4 + fit
plot(stk)
```



By calling the fitted object the default submodel formulas are printed in the console:

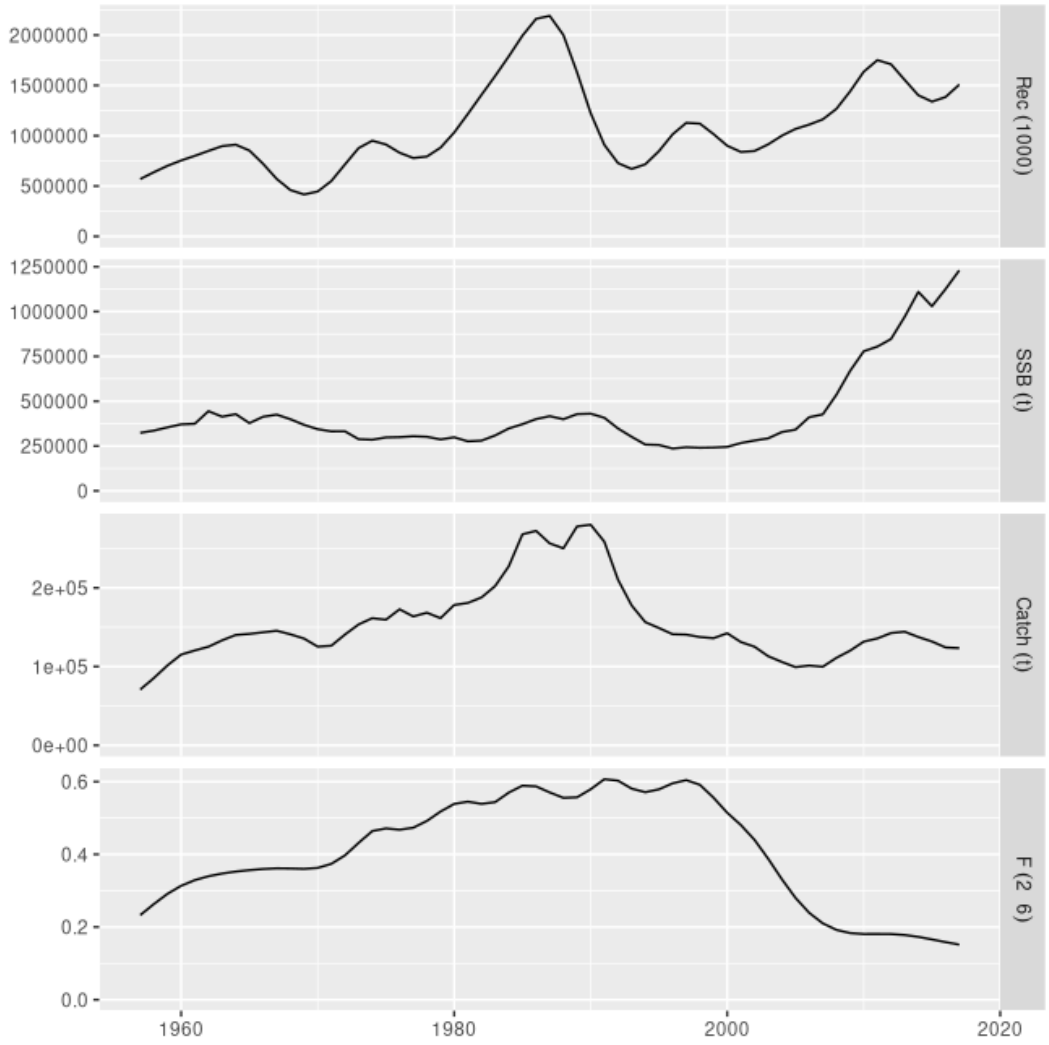
```
fit

## a4a model fit for: PLE
##
## Call:
## .local(stock = stock, indices = indices)
##
## Time used:
##  Pre-processing     Running a4a Post-processing           Total
##       0.8463514      13.8111489       0.1660097      14.8235099
##
## Submodels:
##   fmodel: ~te(age, year, k = c(6, 30), bs = "tp") + s(age, k = 6)
##  srmodel: ~factor(year)
##  n1model: ~s(age, k = 3)
##   qmodel:
```

```
##     BTS-Isis-early:                   ~s(age, k = 6)
##     BTS-Combined (ISIS and TRIDENS): ~s(age, k = 6)
##     SNS:                              ~s(age, k = 5)
##     BTS-Combined (all):               ~s(age, k = 6)
##     IBTS_Q3:                          ~s(age, k = 6)
##     IBTS_Q1:                          ~s(age, k = 5)
##   vmodel:
##     catch:                            ~s(age, k = 3)
##     BTS-Isis-early:                   ~1
##     BTS-Combined (ISIS and TRIDENS): ~1
##     SNS:                              ~1
##     BTS-Combined (all):               ~1
##     IBTS_Q3:                          ~1
##     IBTS_Q1:                          ~1
```

To set specific submodels the user has to write the relevant R formula and include it in the call. The arguments for each submodel are self-explanatory: fishing mortality is 'fmodel', indices' catchability is 'qmodel', stock-recruitment is 'srmodel', observation variance is 'vmodel' and for initial year's abundance is 'n1model'. The following model comes closer to the official stock assessment of North Sea plaice, as such we'll name it 0 and keep it for future comparisons:

For future referencing we'll start with a base fit to be used for future comparisons, named fit 0.

```
fmod0 <- ~s(age, k = 6) + s(year, k = 10) + te(age, year, k = c(3, 8))
qmod0 <- list(~s(age, k = 4), ~s(age, k = 3), ~s(age, k = 3) + year, ~s(age, k = 3),
    ~s(age, k = 4), ~s(age, k = 6))
srmod0 <- ~s(year, k = 20)
vmod0 <- list(~s(age, k = 4), ~1, ~1, ~1, ~1, ~1, ~1, ~1)
n1mod0 <- ~s(age, k = 3)
fit0 <- sca(ple4, ple4.indices, fmodel = fmod0, qmodel = qmod0, srmodel = srmod0,
    n1model = n1mod0, vmodel = vmod0)
stk0 <- ple4 + fit0
plot(stk0)
```



As before by calling the fitted object submodels' formulas are printed in the console:

```
fit

## a4a model fit for: PLE
##
## Call:
## .local(stock = stock, indices = indices)
##
## Time used:
##  Pre-processing     Running a4a Post-processing           Total
##       0.8463514      13.8111489       0.1660097      14.8235099
##
## Submodels:
##   fmodel: ~te(age, year, k = c(6, 30), bs = "tp") + s(age, k = 6)
##  srmodel: ~factor(year)
##  n1model: ~s(age, k = 3)
##   qmodel:
```

```
##     BTS-Isis-early:                  ~s(age, k = 6)
##     BTS-Combined (ISIS and TRIDENS): ~s(age, k = 6)
##     SNS:                             ~s(age, k = 5)
##     BTS-Combined (all):              ~s(age, k = 6)
##     IBTS_Q3:                         ~s(age, k = 6)
##     IBTS_Q1:                         ~s(age, k = 5)
##   vmodel:
##     catch:                           ~s(age, k = 3)
##     BTS-Isis-early:                  ~1
##     BTS-Combined (ISIS and TRIDENS): ~1
##     SNS:                             ~1
##     BTS-Combined (all):              ~1
##     IBTS_Q3:                         ~1
##     IBTS_Q1:                         ~1
```

The method `sca` has other arguments which may be set by the user:

covar: a `FLQuant` with covariates;

wkdir: a folder (character) where the `ADMB`files will be saved for posterior inspection by the user;

verbose: be more verbose (logical);

fit: type of fit (character),

- 'MP' runs the minimizer without trying to invert the hessian and as such doesn't return the covariance matrix of the parameters, normally used inside `MSE`loops where parameter variance may not be relevant;
- 'assessment' runs minimizer and inverts hessian, returns the covariance matrix of the estimated parameters and the convergence criteria set in `ADMB`;
- 'MCMC' runs `ADMB`'s MCMC fit

center: shall observations be centered before fitting (logical);

mcmc: `ADMB`'s MCMC arguments (character vector), must be paired with `fit="MCMC"`.

There are a set of methods for `a4a`fit objects which help manipulating `sca()` results, namely:

+: update the stock object with the fitted fishing mortalities, population abundance and catch in numbers at age;

5.1 Fishing mortality submodel (F_{ay})

We will now take a look at some examples for F models and the forms that we can get.

5.1.1 Separable model

One of the most useful models for fishing mortality is one in which 'age' and 'year' effects are independent, that is, where the shape of the selection pattern does not change over time, but the overall level of fishing mortality do. Commonly called a 'separable model'.

A full separable model in a4ais written using the `factor` function which converts age and year effects into categorical values, forcing a different coefficient to be estimated for each level of both effects. This model has `age x year` number of parameters.

```
fmod1 <- ~factor(age) + factor(year)
fit1 <- sca(ple4, ple4.indices, fmodel = fmod1, fit = "MP")
```

One can reduce the number of parameters and add dependency along both effects, although still keeping independence of each other, by using smoothers rather than `factor`. We'll use a (unpenalised) thin plate spline provided by package mgcv method `s()`. We're using the North Sea Plaice data, and since it has 10 ages we will use a simple rule of thumb that the spline should have fewer than $\frac{10}{2} = 5$ degrees of freedom, and so we opt for 4 degrees of freedom. We will also do the same for year and model the change in F through time as a smoother with 20 degrees of freedom.

```
fmod2 <- ~s(age, k = 4) + s(year, k = 20)
fit2 <- sca(ple4, ple4.indices, fmodel = fmod2, fit = "MP")
```

An interesting extension of the separable model is the 'double separable' where a third factor or smoother is added for the cohort effect.

```
fmod3 <- ~s(age, k = 4) + s(year, k = 20) + s(as.numeric(year - age), k = 10)
fit3 <- sca(ple4, ple4.indices, fmodel = fmod3, fit = "MP")
```

Figures 6 and 7 depicts the three models selectivities for each year. Each separable model has a single selectivity that changes it's overall scale in each year, while the double separable introduces some variability over time by modeling the cohort factor.

```
flqs <- FLQuants(factor = harvest(fit1), smooth = harvest(fit2), double = harvest(fit3))
pset <- list(strip.background = list(col = "gray90"))
xyplot(data ~ age | qname, groups = year, data = flqs, type = "l", col = 1, layout = c(3,
    1), ylab = "fishing mortality", par.settings = pset)
```

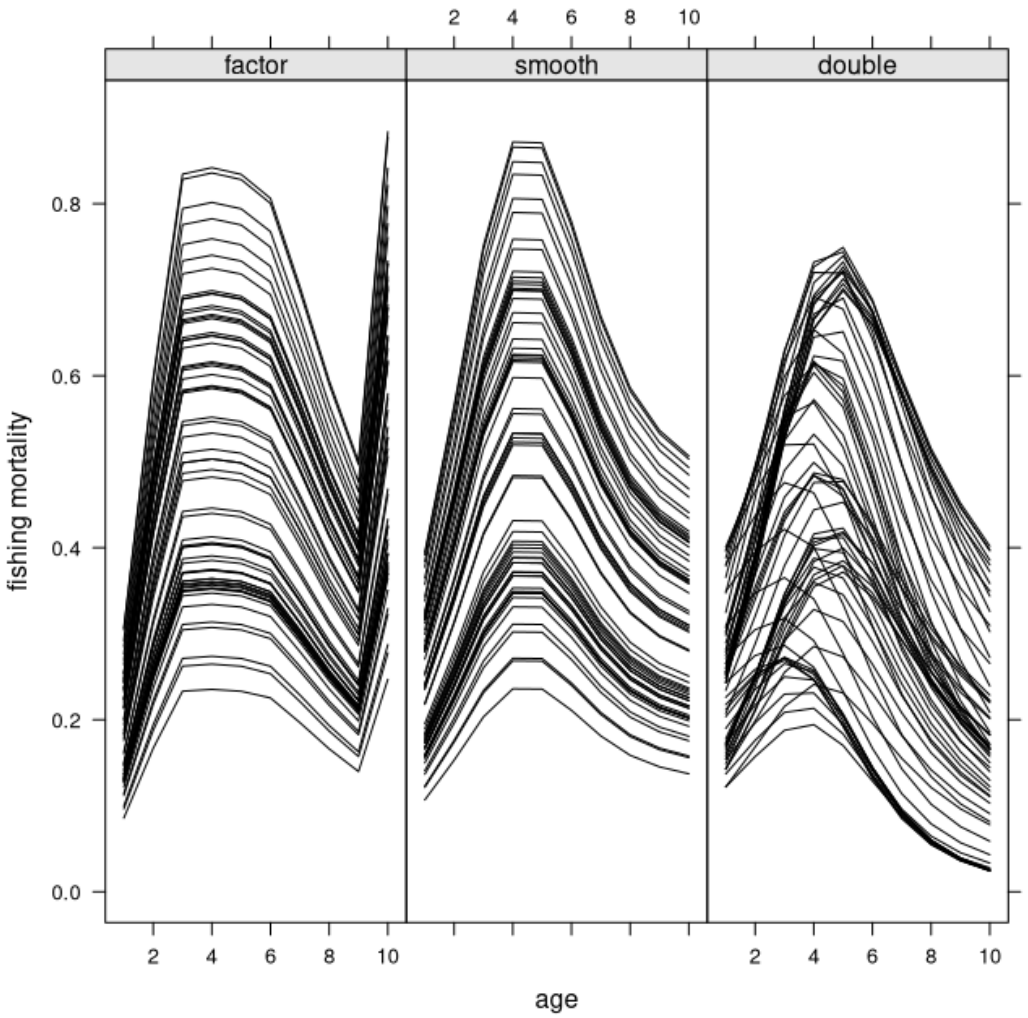


Figure 6: Selection pattern of separable models. Each line represents the selection pattern in a specific year. Independent age and year effects (factor), internally dependent age and year (smooth), double separable (double).

```
wireframe(data ~ age + year | qname, data = as.data.frame(flqs), layout = c(3, 1))
```

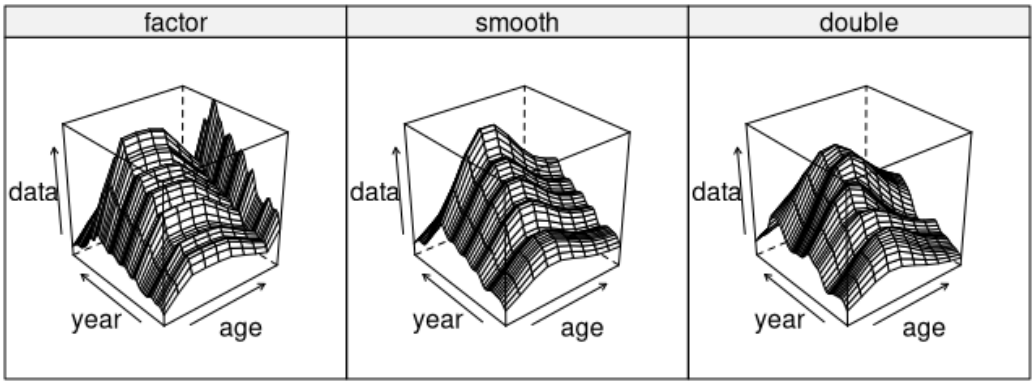


Figure 7: Fishing mortality of separable models. Independent age and year effects (factor), internally dependent age and year (smooth), double separable (double).

5.1.2 Constant selectivity for contiguous ages or years

To set these models we'll use the method `replace()` to define which ages or years will be modelled together with a single coefficient. The following example shows `replace()` in operation. The dependent variables used in the model will be changed and attributed the same age or year, as such during the fit observations of those age or year with will be seen as replicates. One can think of it as sharing the same mean value, which will be estimated by the model.

```
age <- 1:10
# last age same as previous
replace(age, age > 9, 9)

##  [1] 1 2 3 4 5 6 7 8 9 9

# all ages after age 6
replace(age, age > 6, 6)
```

```
##  [1] 1 2 3 4 5 6 6 6 6 6
```

```
year <- 1950:2010
replace(year, year > 2005, 2005)
```

```
##  [1] 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964
## [16] 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979
## [31] 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994
## [46] 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2005 2005 2005 2005
## [61] 2005
```

In the F submodel one can use this method to fix the estimation of F in the plus group to be the same as in the last non-aggregated age.

```
fmod <- ~s(replace(age, age > 9, 9), k = 4) + s(year, k = 20)
fit <- sca(ple4, ple4.indices, fmod)
```

```
wireframe(harvest(fit), zlab = "F")
```

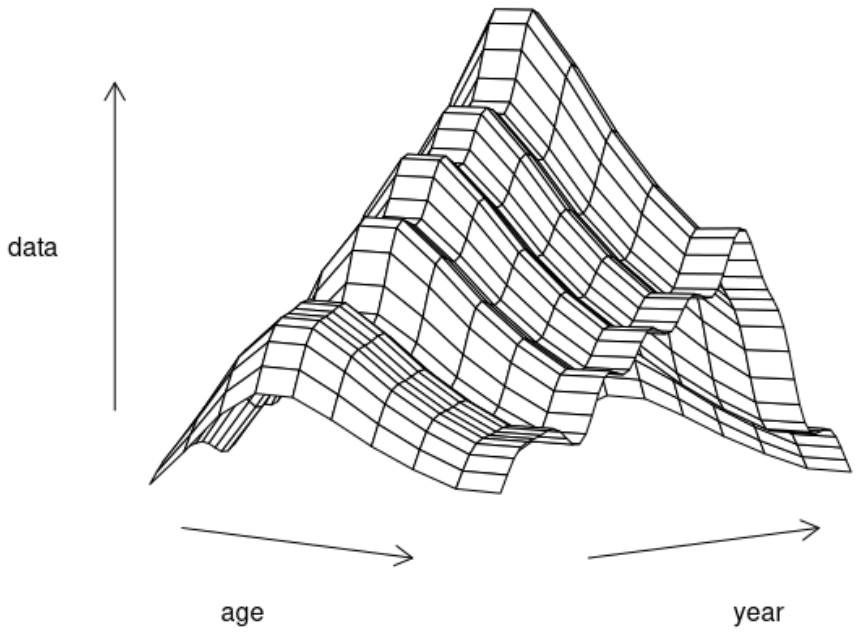


Figure 8: F-at-age fixed above age 9

Or estimate the average F in the most recent years, instead of averaging after the assessment to compute the *statu quo* selection pattern.

```
fmod <- ~s(age, k = 4) + s(replace(year, year > 2013, 2013), k = 20)
fit <- sca(ple4, ple4.indices, fmod)
```

```
wireframe(data ~ age + year, data = harvest(fit)[, ac(2010:2017)], screen = c(z = -130,
    y = 0, x = -60), zlab = "F")
```

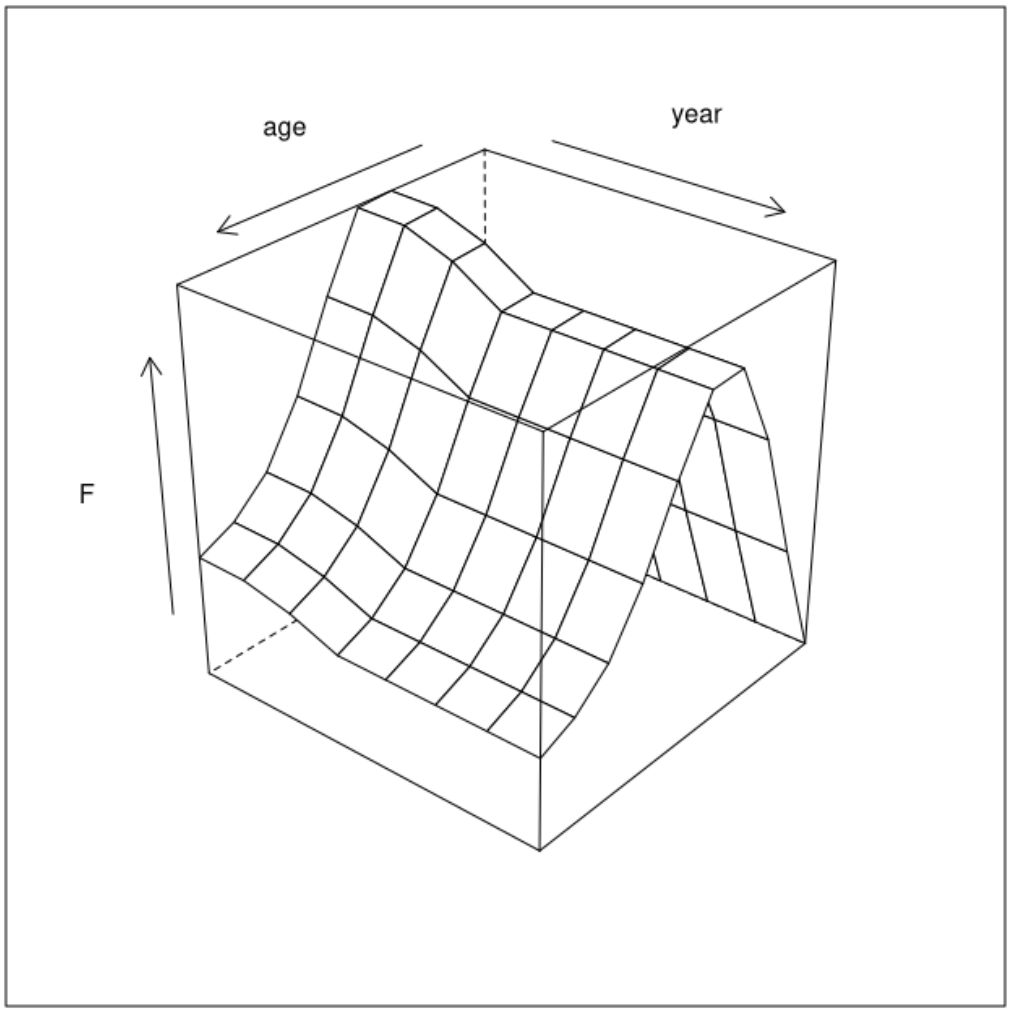


Figure 9: F-at-age fixed for the most recent 5 years

5.1.3 Time blocks selectivity

To define blocks of data (sca()) uses the method `breakpts()`, which creates a factor from a vector with levels defined by the second argument.

```
year <- 1950:2010
# two levels separated in 2000
breakpts(year, 2000)
```

```
##  [1] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
##  [7] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
## [13] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
## [19] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
## [25] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
## [31] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
## [37] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
## [43] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000] (1949,2000]
## [49] (1949,2000] (1949,2000] (1949,2000] (2000,2010] (2000,2010] (2000,2010]
## [55] (2000,2010] (2000,2010] (2000,2010] (2000,2010] (2000,2010] (2000,2010]
## [61] (2000,2010]
## Levels: (1949,2000] (2000,2010]
```

```
# five periods with equal interval
breakpts(year, seq(1949, 2010, length = 6))
```

```
##  [1] (1949,1961.2]   (1949,1961.2]   (1949,1961.2]   (1949,1961.2]
##  [5] (1949,1961.2]   (1949,1961.2]   (1949,1961.2]   (1949,1961.2]
##  [9] (1949,1961.2]   (1949,1961.2]   (1949,1961.2]   (1949,1961.2]
## [13] (1961.2,1973.4] (1961.2,1973.4] (1961.2,1973.4] (1961.2,1973.4]
## [17] (1961.2,1973.4] (1961.2,1973.4] (1961.2,1973.4] (1961.2,1973.4]
## [21] (1961.2,1973.4] (1961.2,1973.4] (1961.2,1973.4] (1961.2,1973.4]
## [25] (1973.4,1985.6] (1973.4,1985.6] (1973.4,1985.6] (1973.4,1985.6]
## [29] (1973.4,1985.6] (1973.4,1985.6] (1973.4,1985.6] (1973.4,1985.6]
## [33] (1973.4,1985.6] (1973.4,1985.6] (1973.4,1985.6] (1973.4,1985.6]
## [37] (1985.6,1997.8] (1985.6,1997.8] (1985.6,1997.8] (1985.6,1997.8]
## [41] (1985.6,1997.8] (1985.6,1997.8] (1985.6,1997.8] (1985.6,1997.8]
## [45] (1985.6,1997.8] (1985.6,1997.8] (1985.6,1997.8] (1985.6,1997.8]
## [49] (1997.8,2010]   (1997.8,2010]   (1997.8,2010]   (1997.8,2010]
## [53] (1997.8,2010]   (1997.8,2010]   (1997.8,2010]   (1997.8,2010]
## [57] (1997.8,2010]   (1997.8,2010]   (1997.8,2010]   (1997.8,2010]
## [61] (1997.8,2010]
## 5 Levels: (1949,1961.2] (1961.2,1973.4] (1973.4,1985.6] ... (1997.8,2010]
```

Note `seq()` computes 'left-open' intervals, which means that to include 1950 the sequence must start one year earlier.

These methods can be used to create discrete time series, for which a different selection pattern is allowed in each block. This is called an interaction in statistical modelling parlance, and typically a `*` denotes an interaction term; for smoothers an interaction is achieved using the `by` argument. When this argument is a `factor` a replicate of the smooth is produced for each factor level.

In the next case we'll use the `breakpts()` to split the time series at 1990, although keeping the same shape in both periods, a thin plate spline with 3 knots (Figure 10).

```
fmod <- ~s(age, k = 3, by = breakpts(year, 1990))
fit <- sca(ple4, ple4.indices, fmod)
```

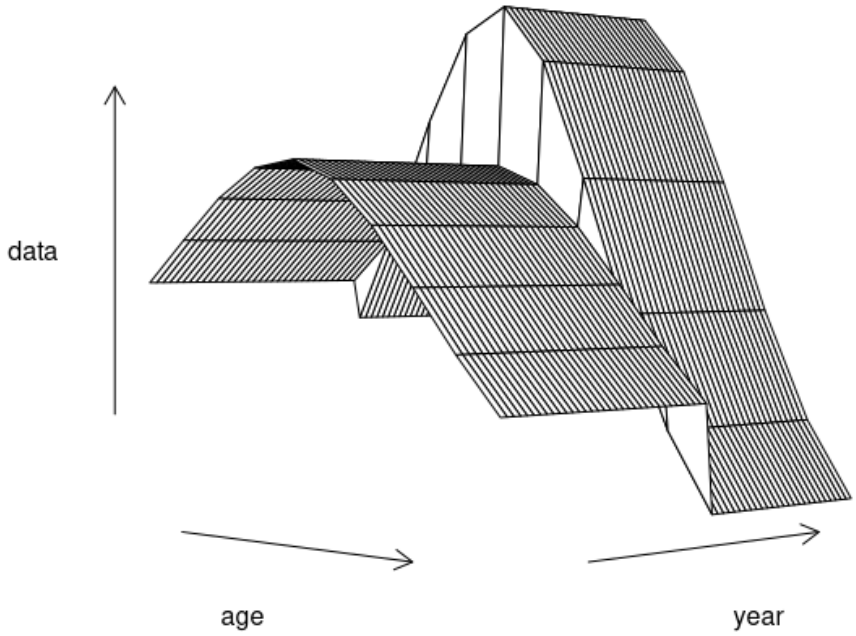


Figure 10: F-at-age in two periods using in both cases a thin plate spline with 3 knots

5.1.4 Time changing selectivity

In many cases, it may be desirable to allow the selection pattern to evolve over time, from year to year. Again there are several ways to do this, one way is to estimate a mean selection pattern, while also allowing F to vary over time for each age. This is like a seperate smoother over year, with 'age blocks' so, looking back at previous examples, we have:

```
fmodel <- ~s(year, k = 15, by = factor(age)) + s(age, k = 4)
```

This is a type of interaction between age and year, but the only connection (or correlation) across ages is via the smoother on age, however there are still 15 degrees of freedom for each age, so the model 5 x 15 + 4 = 69 degrees of freedom. To include correlation across ages and years together then the tensor product ('te()' function) is used, this has the effect of restricting

the flexibility of the model for F. In the following, there is a smoother in 2 dimensions (age and year) where there is 5 degrees of freedom in the age direction, and 15 in the year dimension, resulting in a total of 5 x 15 = 65 degrees of freedom

```
fmodel <- ~te(age, year, k = c(5, 15))
```

Often the above formulations provide too much flexibility, and a more complicated, but simpler model is preferable:

```
fmodel <- ~s(age, k = 4) + s(year, k = 15) + te(age, year, k = c(3, 5))
```

in the above model, the main effects for age and year still have similar flexibility to the full tensor model, however, the interaction (or the change in F at age over time) has been restricted, so that the full model now has 4 + 15 + 3 x 5 = 34 degrees of freedom.

5.1.5 Trawl fleets

5.1.6 Nets and Liners fleets

5.1.7 Multigear fleets

5.1.8 Trawl surveys

5.1.9 Closed form selection pattern

One can use a closed form for the selection pattern. The only requirement is to be able to write it as a Rformula, the example below uses a logistic form.

```
fmod <- ~I(1/(1 + exp(-age)))
fit <- sca(ple4, ple4.indices, fmod)
```

```
wireframe(harvest(fit), zlab = "F")
```

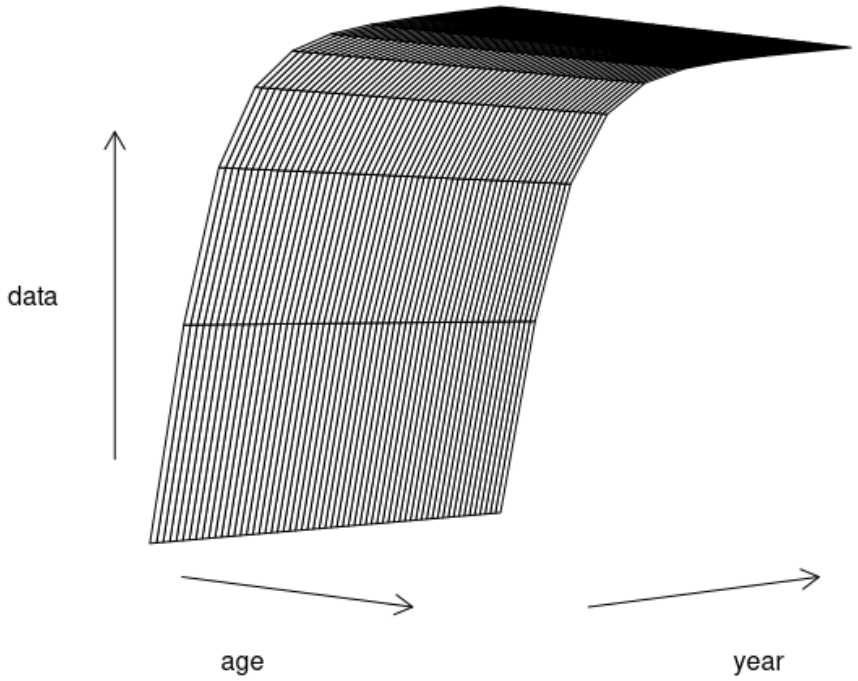


Figure 11: F-at-age logistic

5.2 Abundance indices catchability submodel (Q_{ays})

The catchability submodel is set up the same way as the F submodel and the tools available are the same. The only difference is that the submodel is set up as a list of formulas, where each formula relates with one abundance index. There's no limitation in the number of indices or type that can be used for a fit. It's the analyst that has to decide based on her/his expertise and knowledge of the stock and fleet dynamics.

5.2.1 Catchability submodel for age based indices

A first model is simply a dummy effect on age, which means that a coefficient will be estimated for each age. Note that this kind of model considers that levels of the factor are independent (Figure 12).

```
qmod <- list(~factor(age))
fit <- sca(ple4, ple4.indices[1], qmodel = qmod)
```

```
qhat <- predict(fit)$qmodel[[1]]
wireframe(qhat, zlab = "q")
```

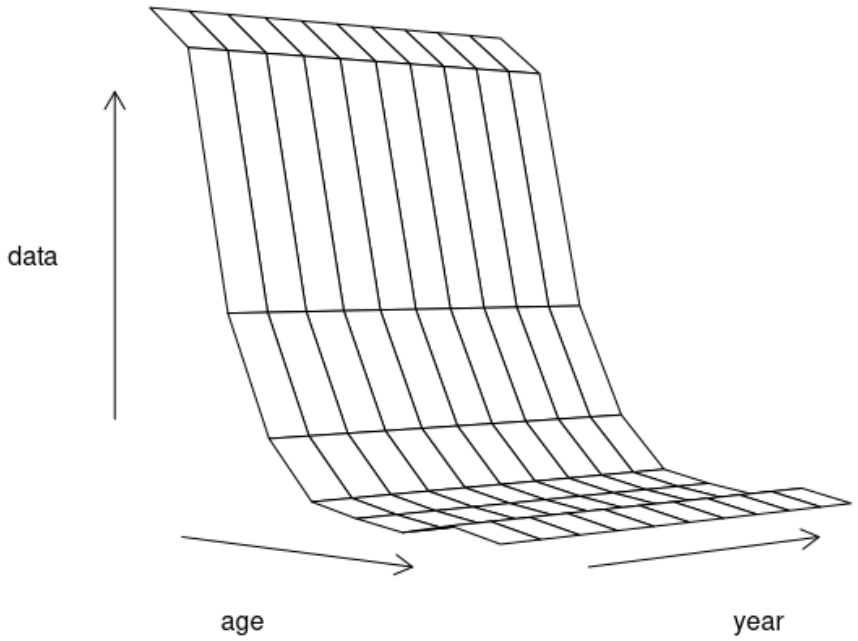


Figure 12: Catchability age independent model

If one considers catchability at a specific age to be dependent on catchability on the other ages, similar to a selectivity modelling approach, one option is to use a smoother at age, and let the data 'speak' regarding the shape (Figure 13).

```
qmod <- list(~s(age, k = 4))
fit <- sca(ple4, ple4.indices[1], qmodel = qmod)
qhat <- predict(fit)$qmodel[[1]]
wireframe(qhat, zlab = "q")
```

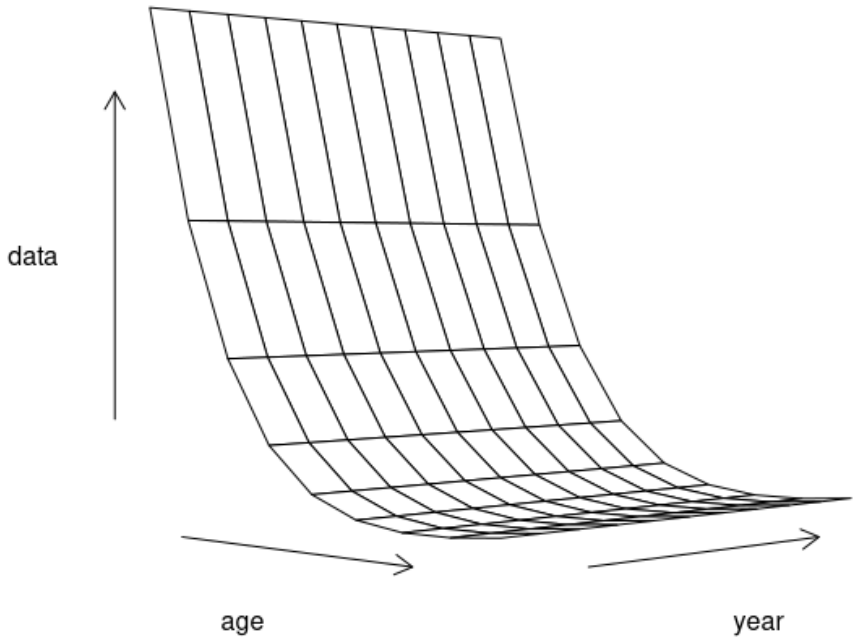


Figure 13: Catchability smoother age model

Finally, one may want to investigate a trend in catchability with time, very common in indices built from CPUE data. In the example given here we'll use a linear trend in time, set up by a simple linear model (Figure 14).

```
qmod <- list(~s(age, k = 4) + year)
fit <- sca(ple4, ple4.indices[1], qmodel = qmod)
qhat <- predict(fit)$qmodel[[1]]
wireframe(qhat, zlab = "q")
```

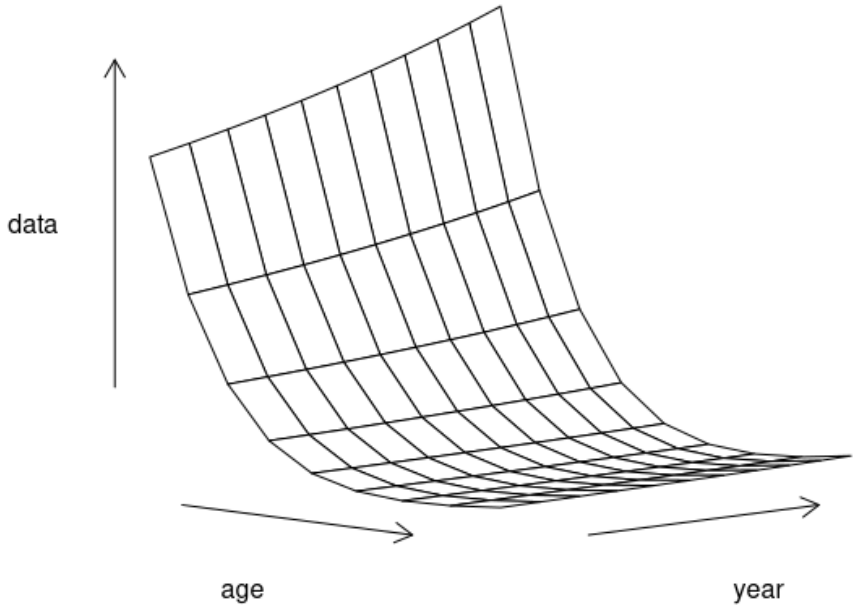


Figure 14: Catchability with a linear trend in year

5.2.2 Catchability submodel for age aggregated biomass indices

The previous section was focused on age disaggregated indices, but age aggregated indices (CPUE, biomass, DEPM, etc) may also be used to tune the total biomass of the population. In these cases a different class for the index must be used, the `FLIndexBiomass`, which uses a vector `index` with the age dimension called 'all'. Note that in this case the qmodel should be set without age factors, although it can have a 'year' component and covariates if needed. An interesting feature with biomass indices is the age range they refer to can be specified.

```
# simulating a biomass index (note the name of the first dimension element)
# using the ple4 biomass and an arbritary catchability of 0.001 plus a
# lognormal error.
dnms <- list(age = "all", year = range(ple4)["minyear"]:range(ple4)["maxyear"])
bioidx <- FLIndexBiomass(FLQuant(NA, dimnames = dnms))
index(bioidx) <- stock(ple4) * 0.001
```

```
index(bioidx) <- index(bioidx) * exp(rnorm(index(bioidx), sd = 0.1))
range(bioidx)[c("startf", "endf")] <- c(0, 0)
# note the name of the first dimension element
index(bioidx)
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## age   1957    1958    1959    1960    1961    1962    1963    1964    1965
##   all  422.52  435.80  464.96  474.43  533.84  575.93  569.60  594.08  520.53
##      year
## age   1966    1967    1968    1969    1970    1971    1972    1973    1974
##   all  504.21  550.70  568.61  423.72  502.22  417.84  505.72  320.34  464.49
##      year
## age   1975    1976    1977    1978    1979    1980    1981    1982    1983
##   all  442.60  568.69  469.33  552.32  445.29  401.95  461.90  405.27  562.49
##      year
## age   1984    1985    1986    1987    1988    1989    1990    1991    1992
##   all  489.61  685.85  675.91  872.77  749.98  630.32  506.46  435.57  405.09
##      year
## age   1993    1994    1995    1996    1997    1998    1999    2000    2001
##   all  426.49  363.47  387.05  305.12  406.58  326.51  401.38  332.17  314.63
##      year
## age   2002    2003    2004    2005    2006    2007    2008    2009    2010
##   all  417.98  417.54  400.95  383.45  416.66  482.58  631.49  587.45  744.50
##      year
## age   2011    2012    2013    2014    2015    2016    2017
##   all  547.98  873.80  914.57 1140.97  856.52  933.23  993.77
##
## units:  t
```

```
# fitting the model
fit <- sca(ple4, FLIndices(bioidx), qmodel = list(~1))
```

To estimate a constant selectivity over time one used the model `~1`. As a matter of fact the estimate value, 0.001, is not very far from the simulated one, 0.001.

An example where the biomass index refers only to age 2 to 4 (for example a CPUE that targets these particular ages).

```
# creating the index
dnms <- list(age = "all", year = range(ple4)["minyear"]:range(ple4)["maxyear"])
bioidx <- FLIndexBiomass(FLQuant(NA, dimnames = dnms))
# but now use only ages 2:4
```

```
index(bioidx) <- tsb(ple4[ac(2:4)]) * 0.001
index(bioidx) <- index(bioidx) * exp(rnorm(index(bioidx), sd = 0.1))
range(bioidx)[c("startf", "endf")] <- c(0, 0)
# to pass this information to the model one needs to specify an age range
range(bioidx)[c("min", "max")] <- c(2, 4)
# fitting the model
fit <- sca(ple4, FLIndices(bioidx), qmodel = list(~1))
```

Once more the estimate value, 9.9×10^{-4}, is not very far from the simulated one, 0.001.

5.2.3 Catchability submodel for single age indices

Similar to age aggregated indices one may have an index that relates only to one age, like a recruitment index. In this case the `FLIndex` object must have in the first dimension the age it referes to. The fit is then done relating the index with the proper age in numbers. Note that in this case the qmodel should be set without age factors, although it can have a 'year' component and covariates if needed.

```
idx <- ple4.indices[[1]][1]
fit <- sca(ple4, FLIndices(recidx = idx), qmodel = list(~1))
# the estimated catchability is
predict(fit)$qmodel[[1]]
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##    year
## age 1985       1986       1987       1988       1989       1990
##   1 0.00030194 0.00030194 0.00030194 0.00030194 0.00030194 0.00030194
##    year
## age 1991       1992       1993       1994       1995
##   1 0.00030194 0.00030194 0.00030194 0.00030194 0.00030194
##
## units:
```

5.3 Stock-recruitment submodel (R_y)

The S/R submodel is a special case, in the sense that it can be set up with the same linear tools as the F and Q models, but it can also use some hard coded models. The example shows how to set up a simple dummy model with `factor()`, a smooth model with `s()`, a Ricker model (`ricker()`), a Beverton and Holt model (`bevholt()`), a hockey stick model (`hockey()`), and a geometric mean model (`geomean()`). See Figure 15 for results. As mentioned before, the 'structural' models have a fixed variance, which must be set by defining the coefficient of variation.

```
srmod <- ~factor(year)
fit <- sca(ple4, ple4.indices, srmodel = srmod)
srmod <- ~s(year, k = 10)
fit1 <- sca(ple4, ple4.indices, srmodel = srmod)
srmod <- ~ricker(CV = 0.05)
fit2 <- sca(ple4, ple4.indices, srmodel = srmod)
srmod <- ~bevholt(CV = 0.05)
fit3 <- sca(ple4, ple4.indices, srmodel = srmod)
srmod <- ~hockey(CV = 0.05)
fit4 <- sca(ple4, ple4.indices, srmodel = srmod)
srmod <- ~geomean(CV = 0.05)
fit5 <- sca(ple4, ple4.indices, srmodel = srmod)
```

```
flqs <- FLQuants(factor = stock.n(fit)[1], smother = stock.n(fit1)[1], ricker = stock.n(fit2)[
    bevholt = stock.n(fit3)[1], hockey = stock.n(fit4)[1], geomean = stock.n(fit5)[1])
xyplot(data ~ year, groups = qname, data = flqs, type = "l", auto.key = list(points = FALSE,
    lines = TRUE, columns = 3), ylab = "No. recruits")
```

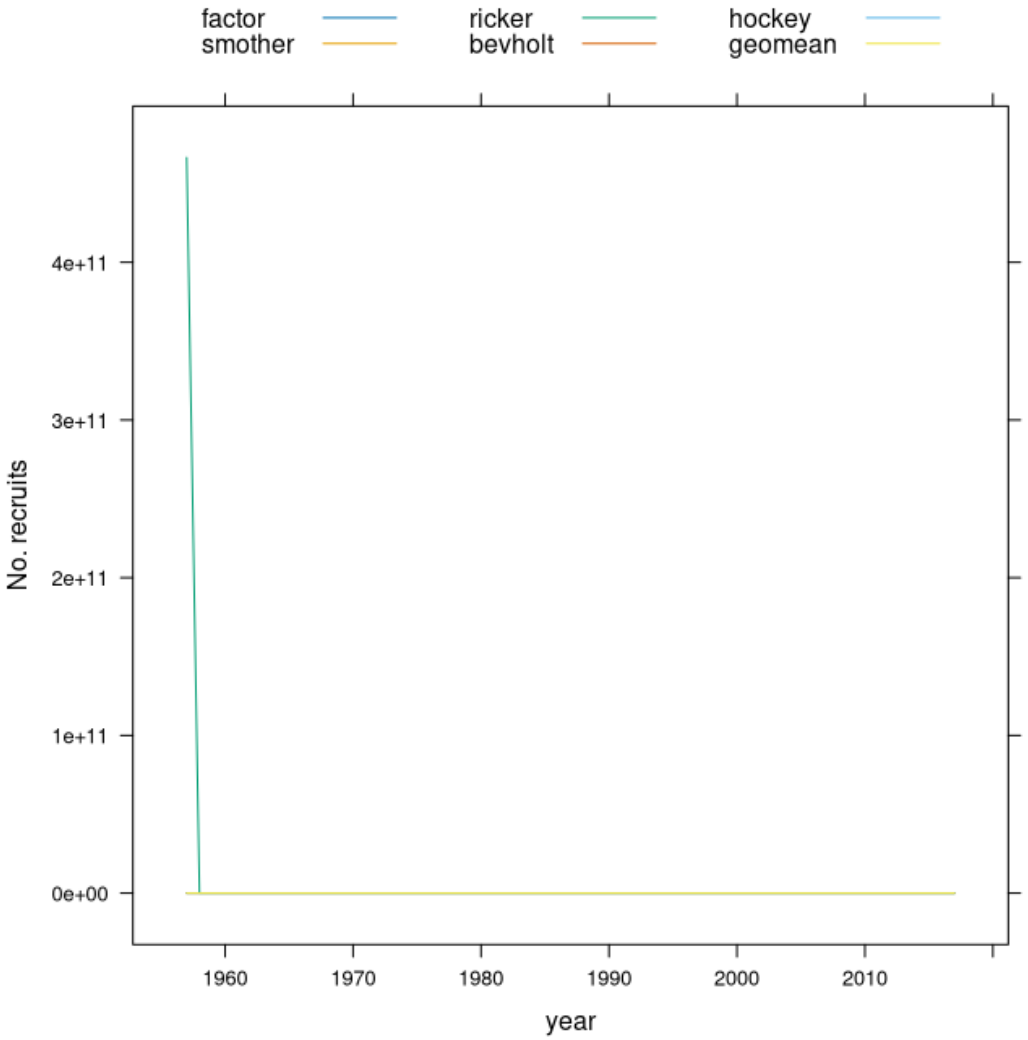


Figure 15: Stock-recruitment models fits

5.4 Observation variance submodel ($\{\sigma^2_{ay}, \tau^2_{ays}\}$)

The variance model allows the user to set up the shape of the observation variances σ^2_{ay} and τ^2_{ays}. This is an important subject related with fisheries data used for input to stock assessment models.

The defaults assume a U-shape model for catch-at-age and constant variance for abundance indices. The first relies on the fact that it's common to have more precision on the most represented ages and less precision on the less frequent ages which tend to be the younger and older individuals. These sizes are less caught by the fleets and as such do not appear as often at the auction markets samples. With regards to the abundance indices, one assumes a scientific survey to have a well designed sampling scheme and protocols which keep observation error at similar levels across ages.

```
vmod <- list(~s(age, k = 3), ~1)
fit1 <- sca(ple4, ple4.indices[1], vmodel = vmod)
```

```
vmod <- list(~s(age, k = 3), ~s(age, k = 3))
fit2 <- sca(ple4, ple4.indices[1], vmodel = vmod)
```

Variance estimated for the constant model is 0.476 while for the U-shape model, fitted with a smoother, changes with ages (Figure 16).

```
wireframe(predict(fit2)$vmodel[[2]], zlab = "variance")
```

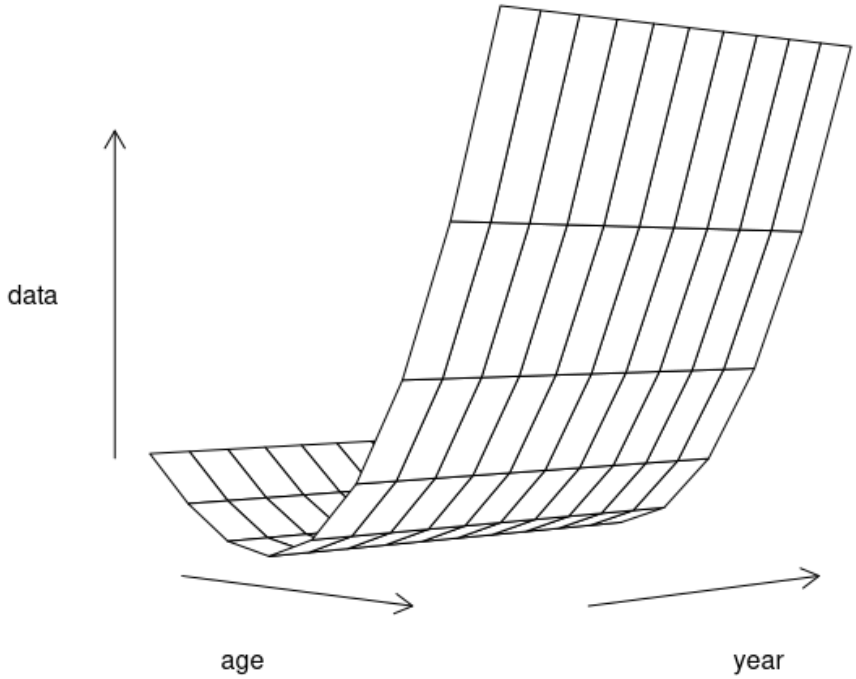


Figure 16: Abundance index observation variance estimate

Observation variance options have an impact in the final estimates of population abundance, which can be seen in Figure 17.

```
flqs <- FLQuants(smother = stock.n(fit1), factor = stock.n(fit2))
xyplot(data ~ year | age, groups = qname, data = flqs, type = "l", scales = list(y = list(rela
    draw = FALSE)), auto.key = list(points = FALSE, lines = TRUE, columns = 2), par.settings =
    "black")), strip.background = list(col = "gray90")), ylab = "")
```

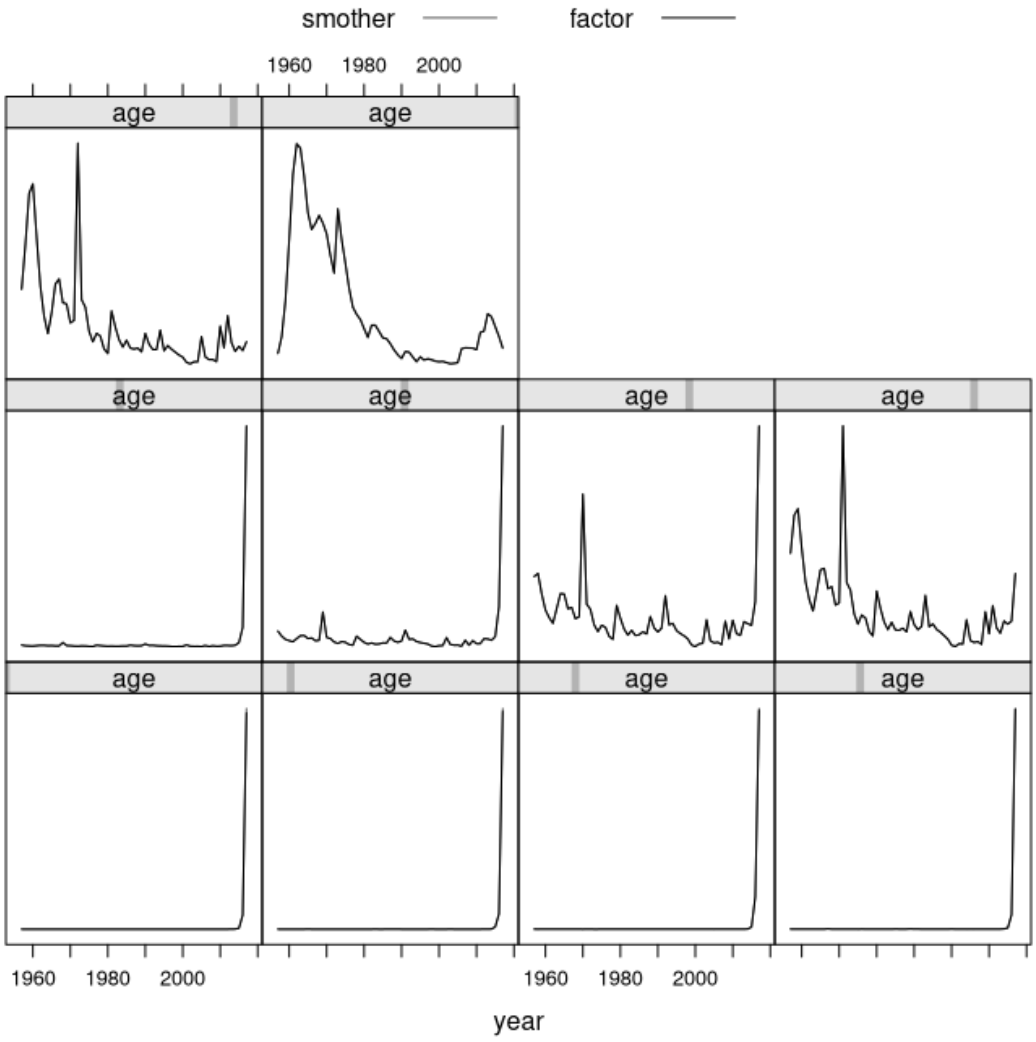


Figure 17: Population estimates using two different variance models

5.5 Initial year abundance submodel ($N_{a,y=1}$)

The submodel for the stock number at age in the first year of the time series is set up with the usual modelling tools (Figure 18). Beare in mind that the year effect does not make sense here since it refers to a single year, the first in the time series of data available. This model has its influence limited to the initial lower triangle of the population matrix, which in assessments with long time series doesn't make much difference. Nevertheless, when modelling stocks with short time series in relation to the number of ages present, it becomes more important and should be given proper attention.

```
n1mod <- ~s(age, k = 3)
fit1 <- sca(ple4, ple4.indices, fmodel = fmod0, qmodel = qmod0, srmodel = srmod0,
    vmodel = vmod0, n1model = n1mod)
n1mod <- ~factor(age)
fit2 <- sca(ple4, ple4.indices, fmodel = fmod0, qmodel = qmod0, srmodel = srmod0,
```

```
    vmodel = vmod0, n1model = n1mod)
flqs <- FLQuants(smother = stock.n(fit1)[, 1], factor = stock.n(fit2)[, 1])
```

```
pset <- list(superpose.line = list(col = c("gray50", "black"), lty = c(1, 2)))
xyplot(data ~ age, groups = qname, data = flqs, type = "l", auto.key = lgnd, par.settings = ps
    ylab = "")
```

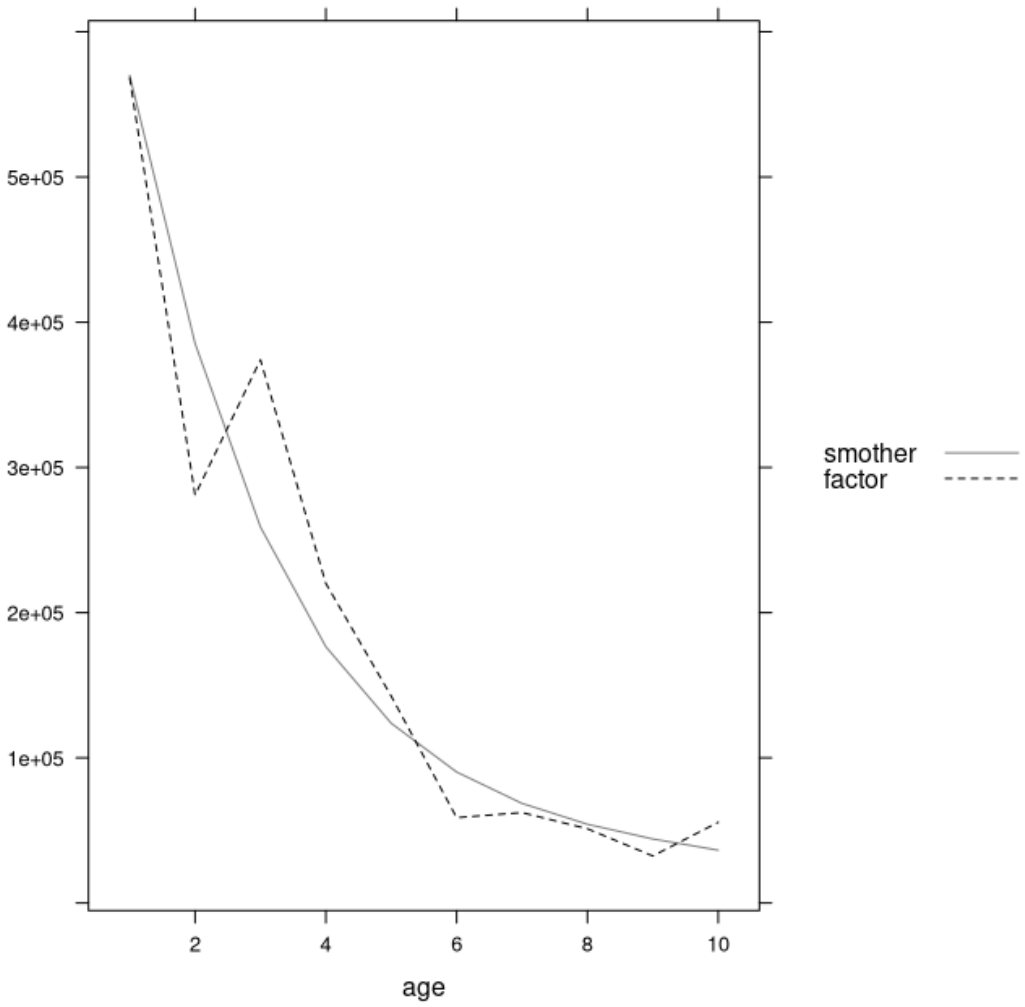


Figure 18: Nay=1 models

The impact in the overall perspective of the stock status is depicted in Figure 19. As time goes by the effect of this model vanishes and the fits become similar.

```
flqs <- FLQuants(smother = stock.n(fit1), factor = stock.n(fit2))
pset$strip.background <- list(col = "gray90")
scl <- list(y = list(relation = "free", draw = FALSE))
xyplot(data ~ year | factor(age), groups = qname, data = flqs, type = "l", scales = scl,
    auto.key = lgnd, par.settings = pset, ylab = "")
```

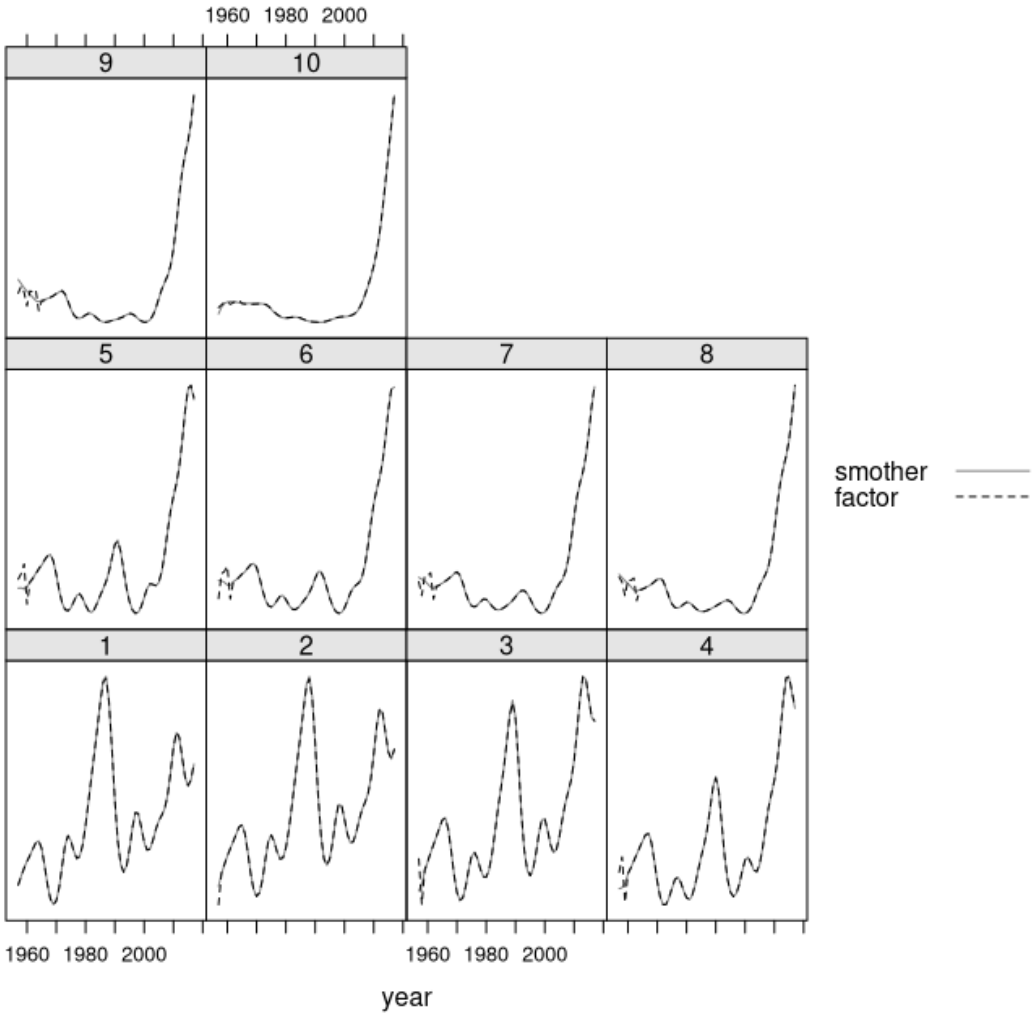


Figure 19: Population estimates using two different variance models

5.6 Data weigthing

By default the likelihood components are not weighted and the contribution of each to the maximum likelihood depends on their own likelihood score. However, the user may change these weights by penalizing data points, the w_{ays} in section **??**. The likelihood score of each data point will be multiplied by the normalized weights ($\sum w_{ays} = 1$). This is done by adding a variance matrix to the `catch.n` and `index.n` slots of the stock and index objects. The values should be given as coefficients of variation on the log scale, so that variance is $\log(CV^2 + 1)$. Figures Figure 20 and 21 show the results of the two fits in the population abundance and stock summary.

```
stk <- ple4
idx <- ple4.indices[1]
# cv of observed catches
varslt <- catch.n(stk)
```

```
varslt[] <- 0.4
catch.n(stk) <- FLQuantDistr(catch.n(stk), varslt)
# cv of observed indices
varslt <- index(idx[[1]])
varslt[] <- 0.1
index.var(idx[[1]]) <- varslt
# run
fit1 <- sca(stk, idx, fmodel = fmod0, qmodel = qmod0, srmodel = srmod0, vmodel = vmod0,
    n1model = n1mod0)
flqs <- FLQuants(nowgt = stock.n(fit0), extwgt = stock.n(fit1))
```

```
xyplot(data ~ year | factor(age), groups = qname, data = flqs, type = "l", scales = scl,
    auto.key = lgnd, par.settings = pset, ylab = "")
```

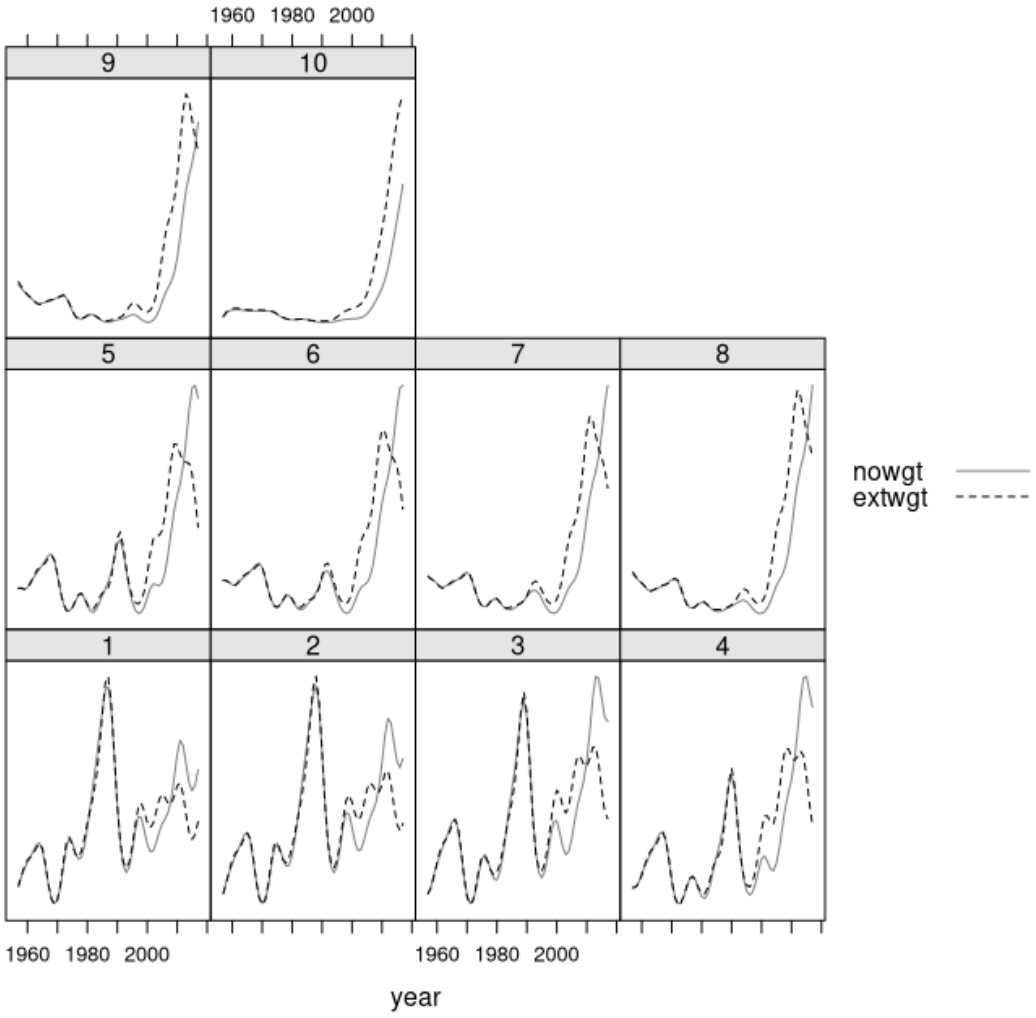


Figure 20: Stock summary of distinct likelihood weightings

```
flsts <- FLStocks(nowgt = ple4 + fit0, wgt = ple4 + fit1)
plot(flsts)
```

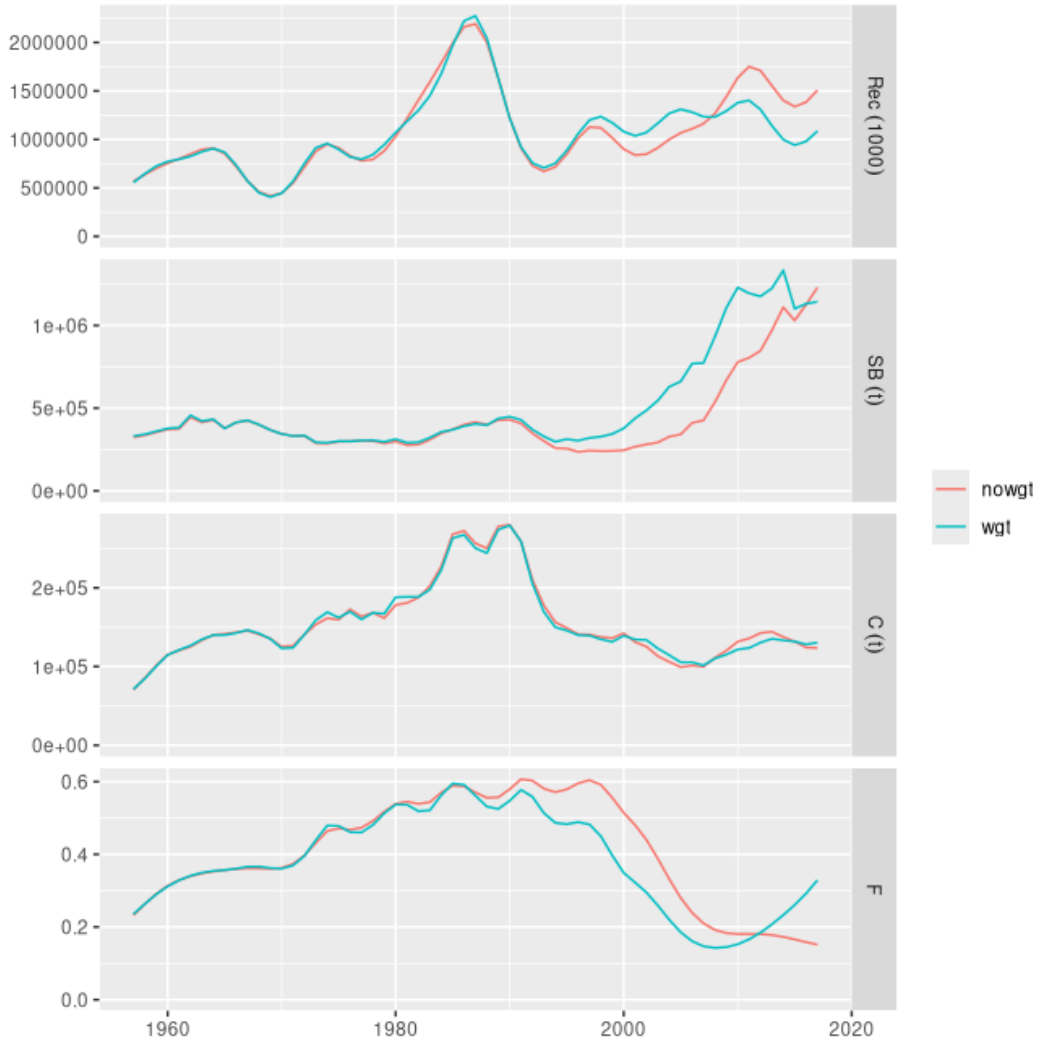


Figure 21: Population estimates using two different variance models

Note that by using a smaller CV for the index, one is increasing the contribution of the survey and penalizing catch at age, in relative terms. The ratio between likelihood scores of both fits show this effect with catch at age increasing by 2.3 while the index increases almost 8 fold.

```
fit0 <- sca(ple4, ple4.indices[1], fmodel = fmod0, qmodel = qmod0, srmodel = srmod0,
    vmodel = vmod0, n1model = n1mod0)
(fitSumm(fit1)/fitSumm(fit0))[c(2, 8, 9), ]

##       nlogl nlogl_comp1 nlogl_comp2
##    5.158773    2.577961    9.762901
```

5.7 Working with covariates

In linear model one can use covariates to explain part of the variance observed on the data that the 'core' model does not explain. The same can be done in the a4aframework. The example below uses the North Atlantic Oscillation (NAO) index to model recruitment.

```
nao <- read.table("https://www.cpc.ncep.noaa.gov/products/precip/CWlink/pna/norm.nao.monthly.b
    skip = 1, fill = TRUE, na.strings = "-99.90")
dnms <- list(quant = "nao", year = 1950:2024, unit = "unique", season = 1:12, area = "unique")
nao <- FLQuant(unlist(nao[, -1]), dimnames = dnms, units = "nao")
nao <- seasonMeans(trim(nao, year = dimnames(stock.n(ple4))$year))
```

First by simply assuming that the NAO index drives recruitment (‘r fign('naor')‘).

```
srmod <- ~nao
fit2 <- sca(ple4, ple4.indices[1], qmodel = list(~s(age, k = 4)), srmodel = srmod,
    covar = FLQuants(nao = nao))
flqs <- FLQuants(simple = stock.n(fit)[1], covar = stock.n(fit2)[1])
```

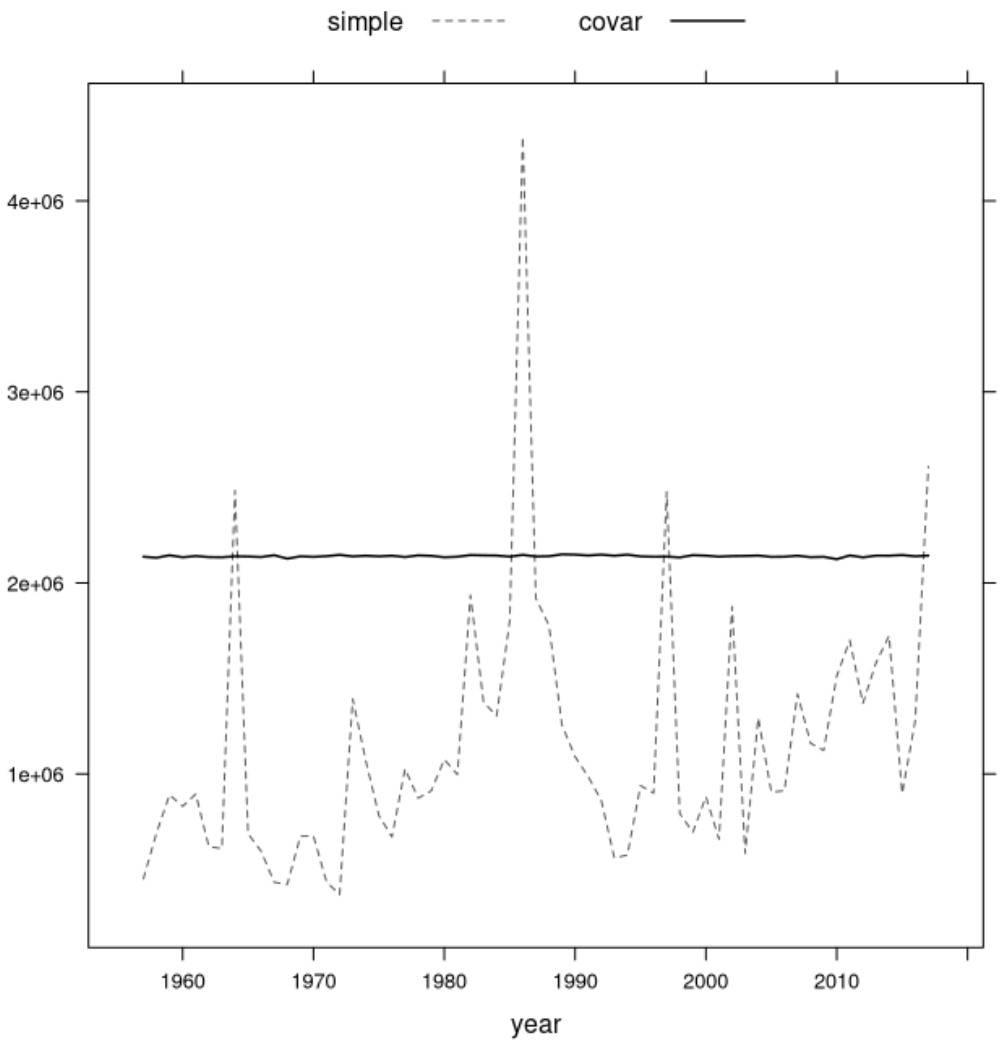


Figure 22: Recruitment model with covariates. Using the NAO index as a recruitment index.

In a second model we're using the NAO index not to model recruitment directly but to model one of the parameters of the S/R function (‘r fign('naor2')‘).

```
srmod <- ~ricker(a = ~nao, CV = 0.25)
fit3 <- sca(ple4, ple4.indices[1], qmodel = list(~s(age, k = 4)), srmodel = srmod,
    covar = FLQuants(nao = nao))
flqs <- FLQuants(simple = stock.n(fit)[1], covar = stock.n(fit3)[1])
```

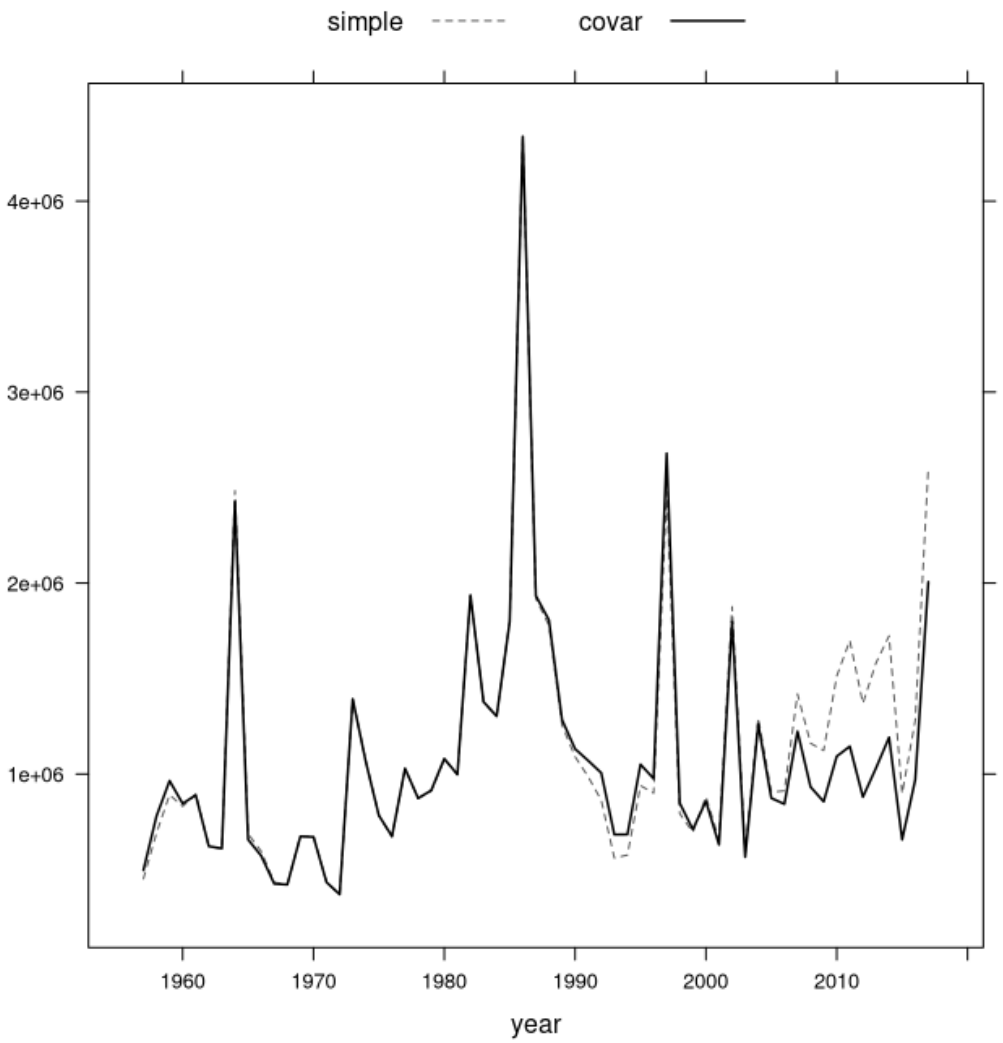


Figure 23: Recruitment model with covariates. Using the NAO index as a covariate for the stock-recruitment model parameters.

Note that covariates can be added to any submodel using the linear model capabilities of R.

5.8 Assessing ADMBfiles

The framework gives access to the files produced to run the ADMBfitting routine through the argument `wkdir`. When set up all the ADMBfiles will be left in the directory. Note that the ADMBtpl file is distributed with the FLa4a. One can get it from your Rlibrary, under the folder `myRlib/FLa4a/admb/`.

```
fit1 <- sca(ple4, ple4.indices, wkdir = "fit1run")
```

5.9 Missing observations in the catch matrix or index

How are the data interpolated etc ...

6 Diagnostics

There's a large number of diagnostics that can be computed for a stock assessment model, the a4aframework implements several analysis of residuals, visualizations and statistics that can be used to evaluate the fit quality and chose across multiple fits.

6.1 Residuals

Residuals are a ubiquos metrics to check quality of a fit. For `sca()` fits there are out-of-the-box methods to compute in the log scale, raw residuals (aka deviances), standardized residuals and pearson residuals. A set of plots to inspect residuals and evaluate fit quality and assumptions are implemented.

Consider x_{ay} to be either a catch-at-age matrix (C_{ay}) or one abundance index (I_{ay}[1]) and d to represent residuals.

Raw residuals are compute by $d_{ay} = \log x_{ay} - \log \tilde{x}_{ay}$ and have distribution $N(0, \upsilon_a^2)$. Standardized residuals will be compute with $d^s_{ay} = \frac{d_{ay}}{\hat{\upsilon}_a^2}$ where $\hat{\upsilon}_a^2 = (n-1)^{-1} \sum_y (d_{ay})^2$. Pearson residuals scale raw residuals by the estimates of σ^2 or τ^2, as such $d^p_{ay} = \frac{d_{ay}}{\tilde{\upsilon}_a^2}$ where $\tilde{\upsilon}_a^2 = \tilde{\sigma}_a^2$ for catches, or $\tilde{\upsilon}_a^2 = \tilde{\tau}_a^2$ for each index of abundance.

The `residuals()` method will compute these residuals and generate a object which can be plotted using a set of packed methods. The argument `type` will allow the user to chose which residuals will be computed. By default the method computes standardized residuals.

```
fit <- sca(ple4, ple4.indices)
d_s <- residuals(fit, ple4, ple4.indices)
```

Figure 24 shows a scatterplot of standardized residuals with a smoother to guide (or mis-guide ...) your visual analysis. Note that the standardization should produce residuals with variance 1, which means that most residual values should be between ~ -2 and ~ 2.

```
plot(d_s)
```

[1]For simplicity of notation we'll avoid the subscript s in I, since we're referring to individual indices

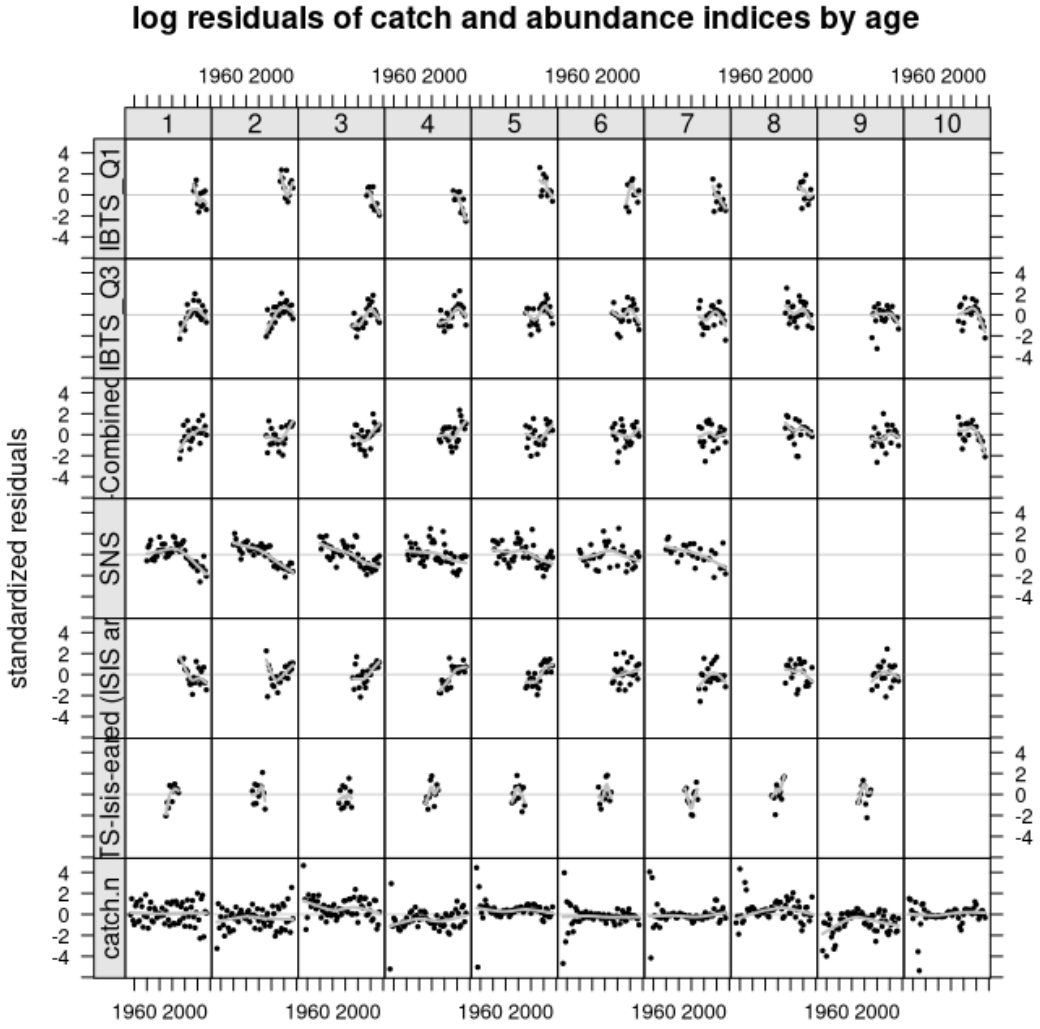


Figure 24: Standardized residuals for abundance indices and catch numbers (catch.n). Each panel is coded by age class, dots represent standardized residuals and lines a simple smoother.

When plotting residuals by default the auxiliar line is a smoother. However it's possible to use other type of lines by setting the argument "auxline" in plot. The argument can take the values used by xyplot, which are (from panel.xyplot help page) one or more of the following: "p", "l", "h", "b", "o", "s", "S", "g", "r", "a", "smooth", and "spline". If type has more than one element, an attempt is made to combine the effect of each of the components. The behaviour if any of the first five are included is similar to the effect of the corresponding type in plot: "p" and "l" stand for points and lines respectively; "b" and "o" (for 'overlay') plot both; "h" draws vertical (or horizontal if horizontal = TRUE) line segments from the points to the origin. Types "s" and "S" are like "l" in the sense that they join consecutive points, but instead of being joined by a straight line, points are connected by a vertical and a horizontal segment forming a 'step', with the vertical segment coming first for "s", and the horizontal segment coming first for "S". "g" adds a reference grid. Type "r" adds a linear regression line, "smooth" adds a loess fit, "spline" adds a cubic smoothing spline fit, and "a" draws line segments joining the average y value for each distinct x value. Figure 25 shows a regression line over the residuals instead of

the loess smooother.

```
plot(d_s, auxline = "r")
```

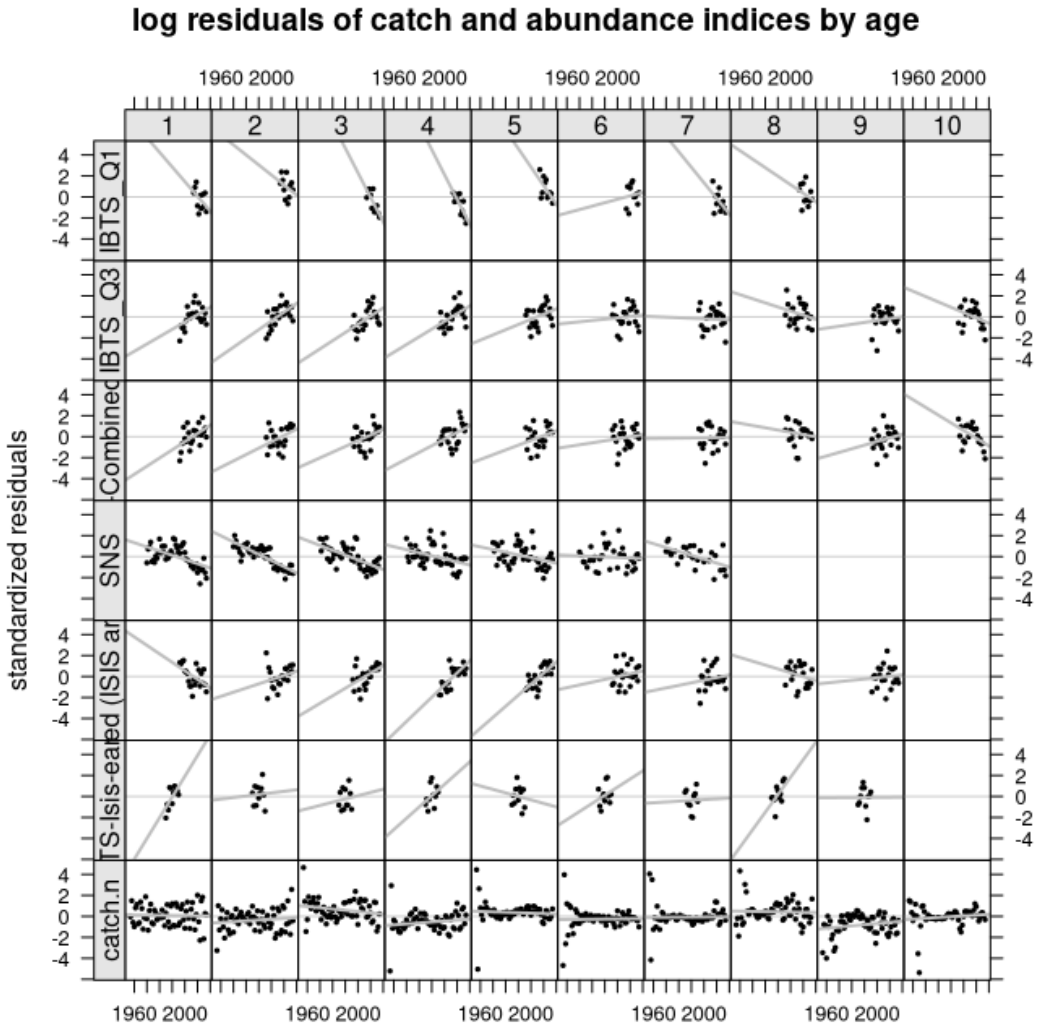


Figure 25: Standardized residuals for abundance indices and catch numbers (catch.n). Each panel is coded by age class, dots represent standardized residuals and lines a simple smoother.

The common bubble plot (`bubble()`) are shown in Figure 26. It shows the same information as Figure 24 but in a multivariate perspective.

```
bubbles(d_s)
```

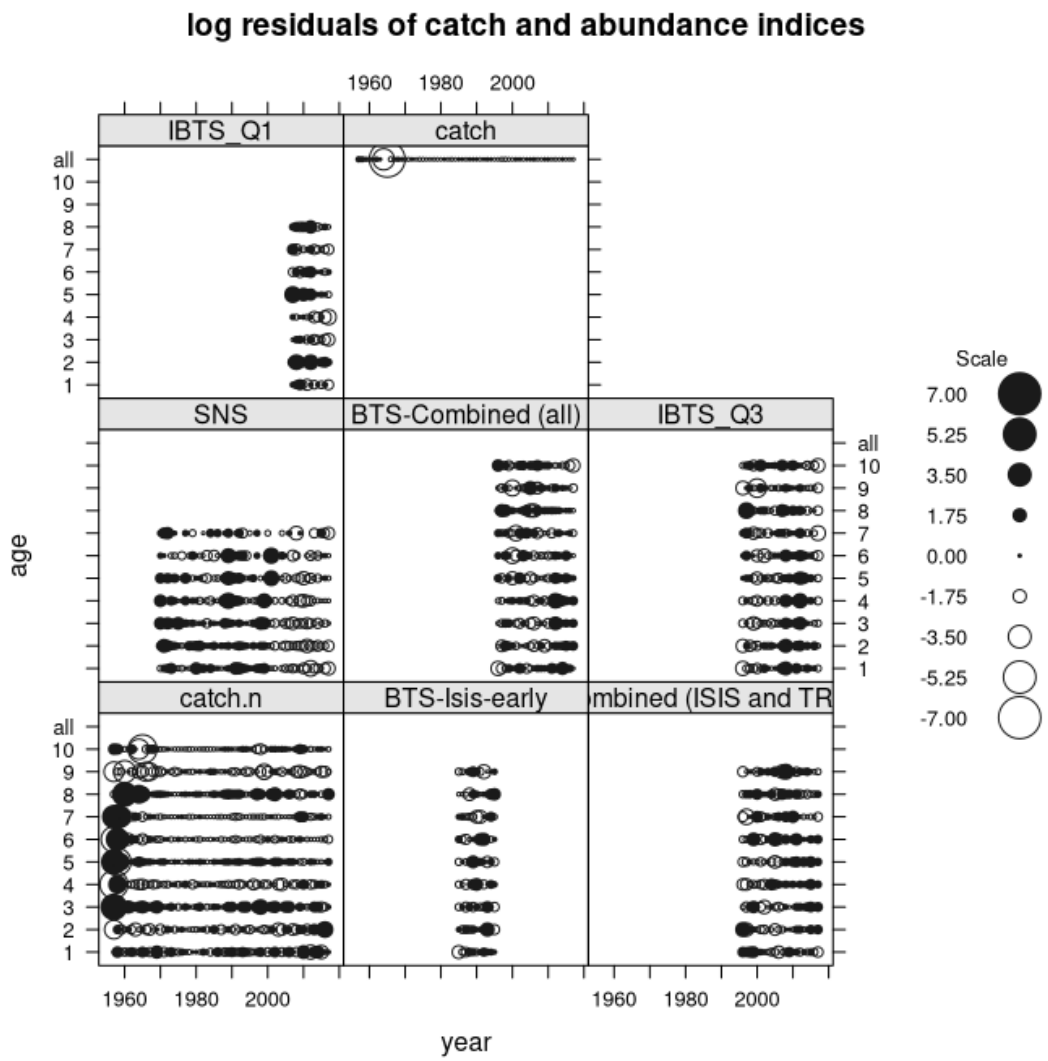


Figure 26: Bubbles plot of standardized residuals for abundance indices and for catch numbers (catch.n).

Figure 27 shows a quantile-quantile plot to assess how well standardized residuals match a normal distribution.

```
qqmath(d_s)
```

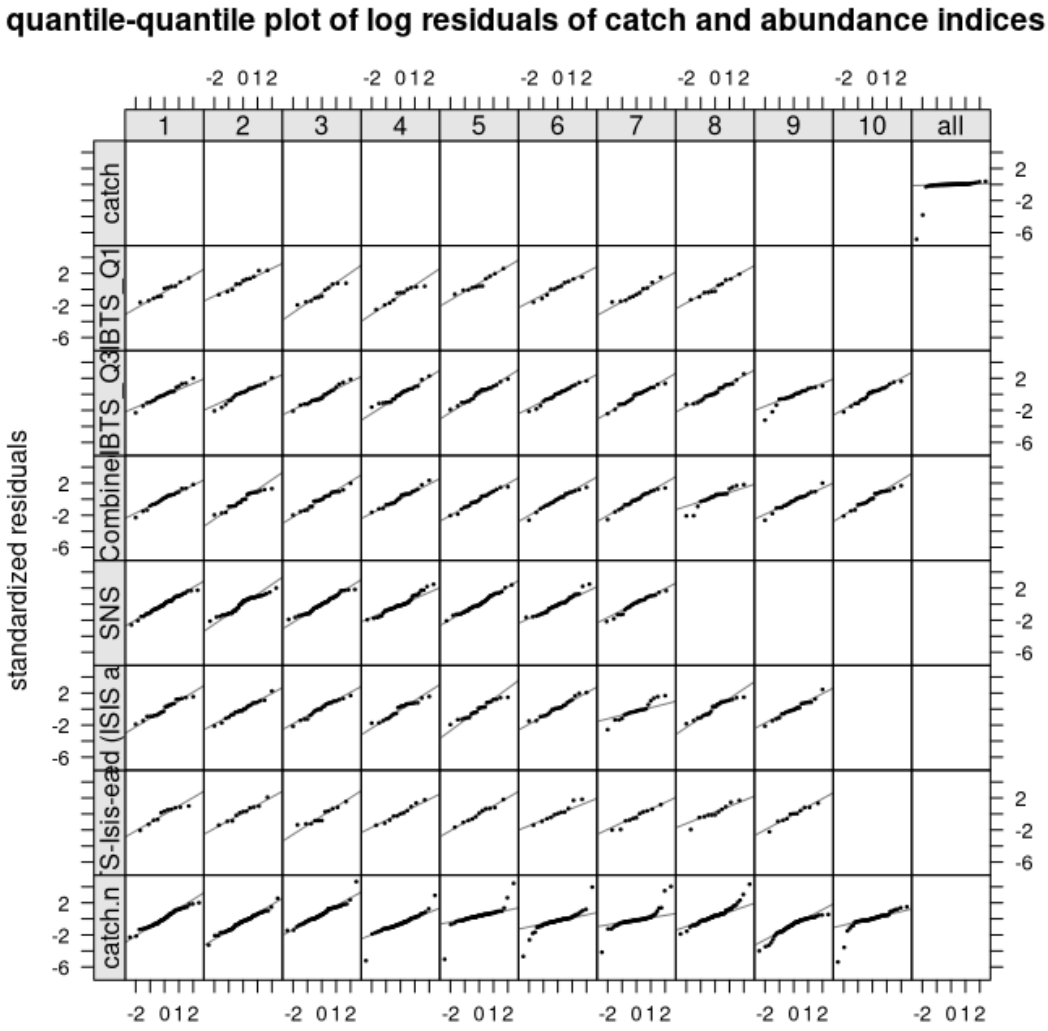


Figure 27: Quantile-quantile plot of standardized residuals for abundance indices and catch numbers (catch.n). Each panel is coded by age class, dots represent standardized residuals and lines the normal distribution quantiles.

Pearson residuals can be computed and plotted the same way as standardized residuals by setting `fit='pearson'` (Figure 28). These residuals are particularly useful to evaluate the `vmodel`, departures from 0 should be fixed by tweaking the `vmodel` equations.

```
d_p <- residuals(fit, ple4, ple4.indices, type = "pearson")
plot(d_p)
```

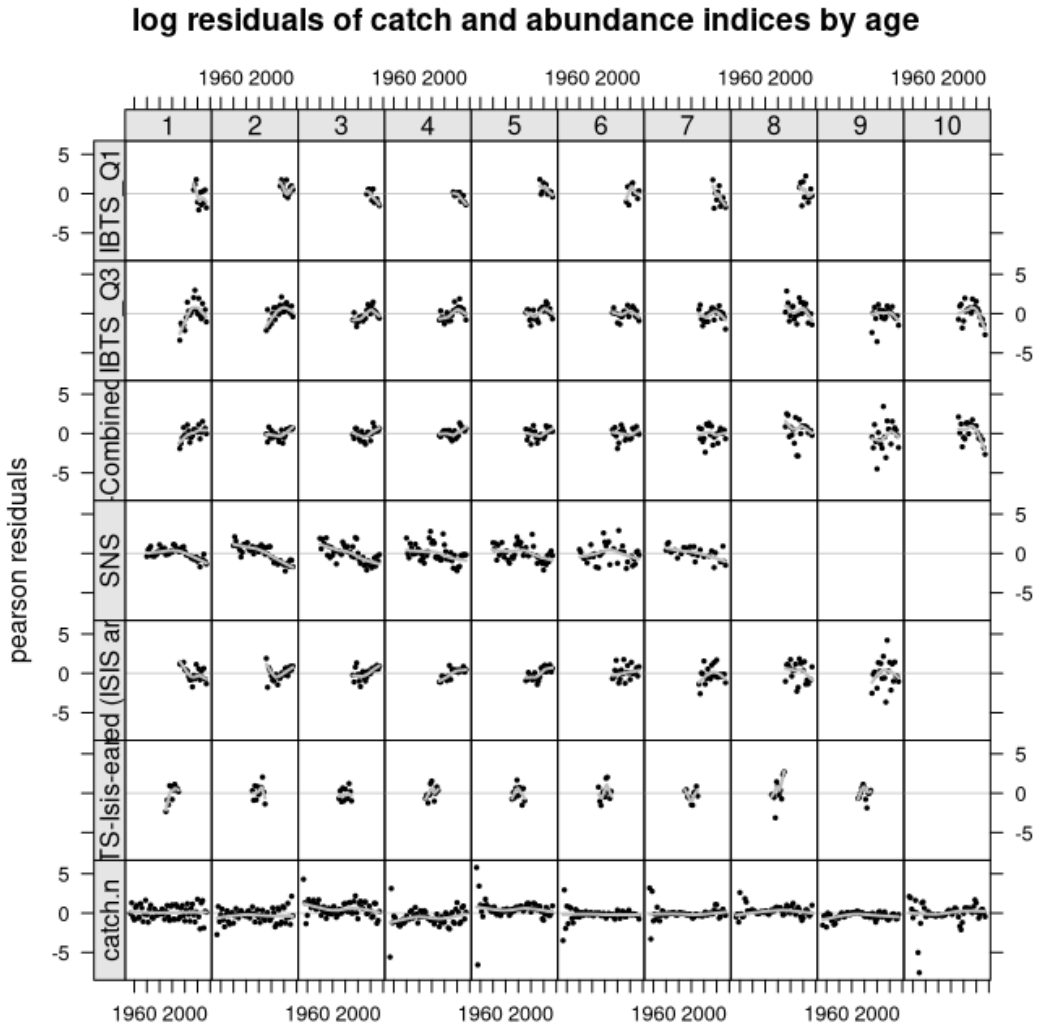


Figure 28: Pearson residuals for abundance indices and catch numbers (catch.n). Each panel is coded by age class, dots represent standardized residuals and lines a simple smoother.

Finally, the raw residuals are computed by setting `fit='deviances'` and plotted the same way as before (Figure 29). These residuals are usefull to identify which data points are not well modelled, showing a large dispersion of the residuals and requiring more attention from the analyst.

```
d_r <- residuals(fit, ple4, ple4.indices, type = "deviances")
plot(d_r)
```

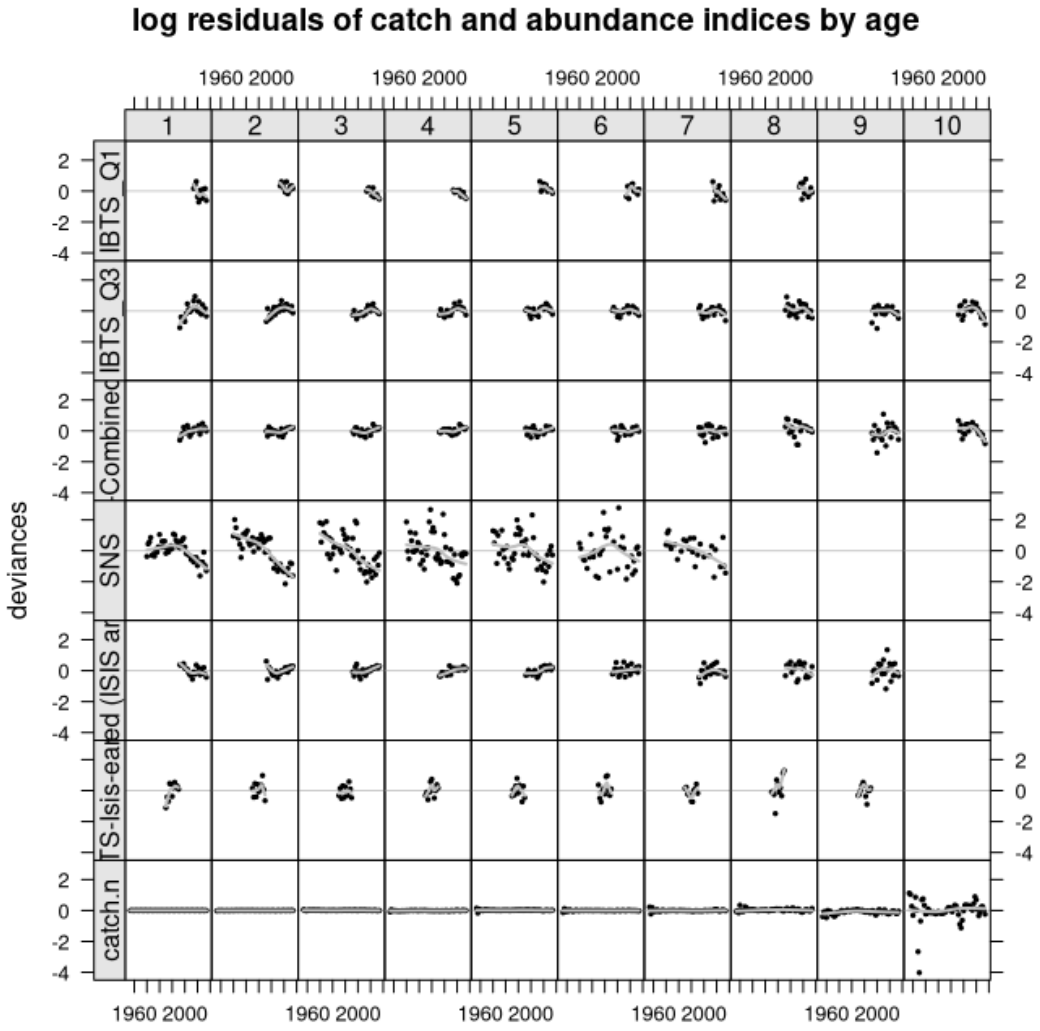


Figure 29: Raw residuals for abundance indices and catch numbers (catch.n). Each panel is coded by age class, dots represent standardized residuals and lines a simple smoother.

6.2 Predictive skill

An important feature of stock assessment model fits is the capacity to predict, since one of the most important analysis done with these fits is forecasting future fishing opportunities under pre-defined conditions. The `a4a`framework implements a visualization of the fit's predictive skill for both catch-at-age and abundance indices. These are generated by the method `plot()` with the fit object and a `FLStock` (Figure 30) or `FLIndices` (Figure **??**) object as arguments.

```
plot(fit, ple4)
```

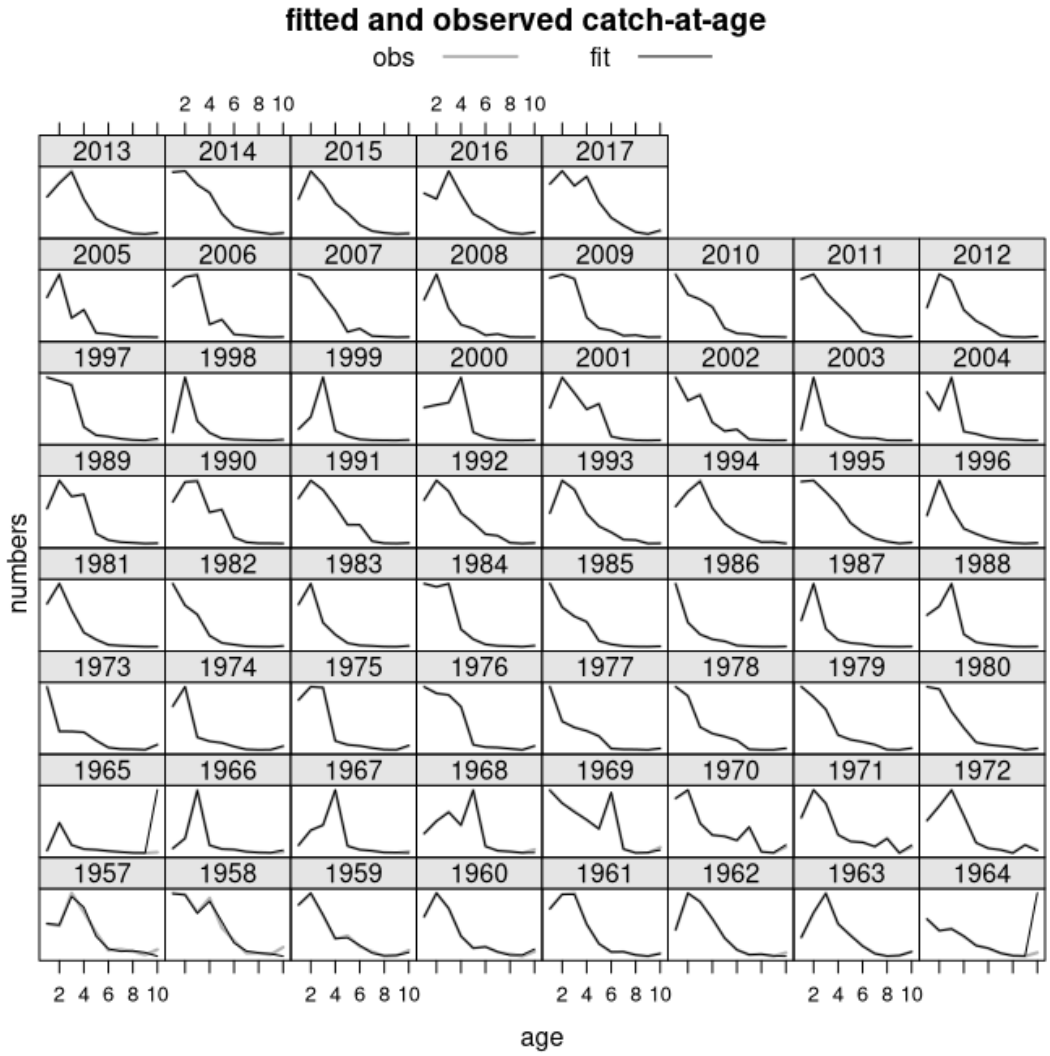


Figure 30: Predict and observed catch-at-age

```
plot(fit, ple4.indices)
```

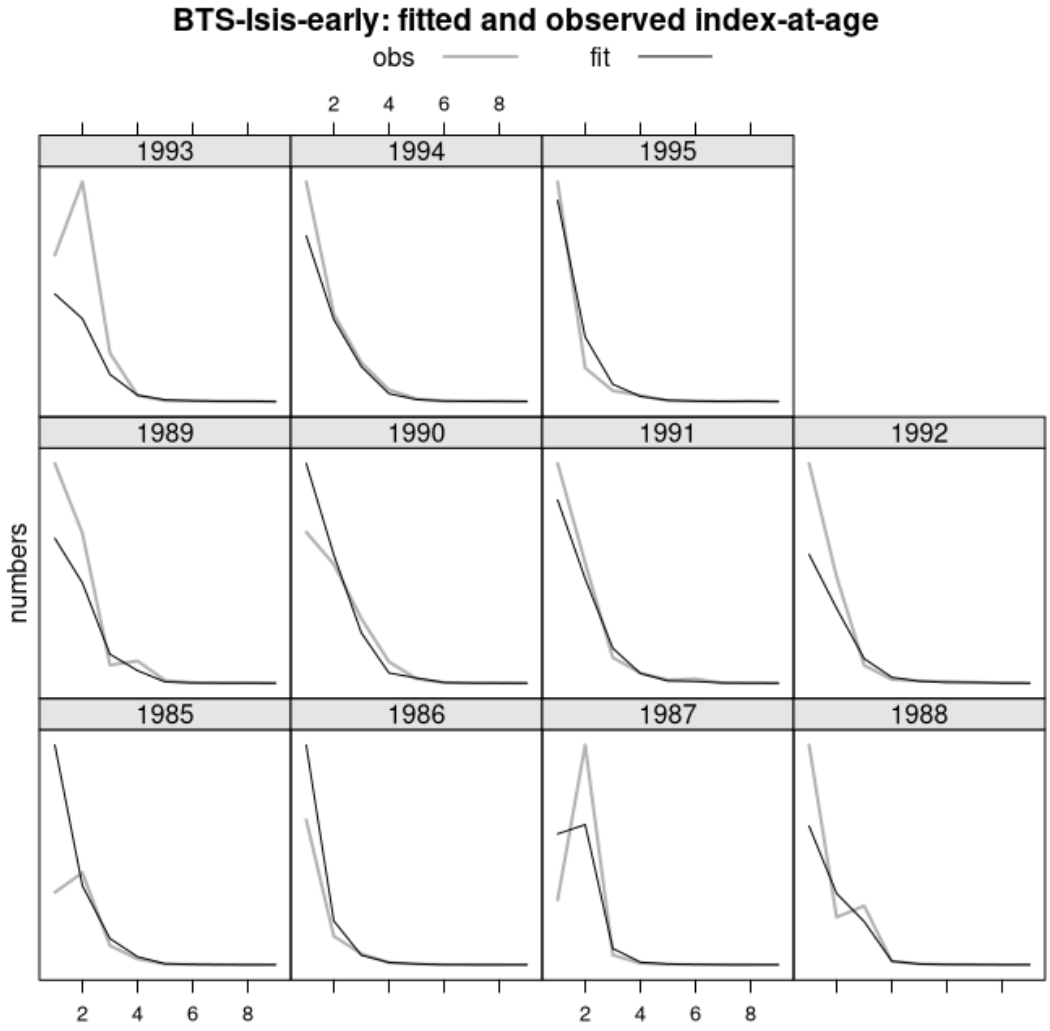


Figure 31: Predict and observed abundance-at-age

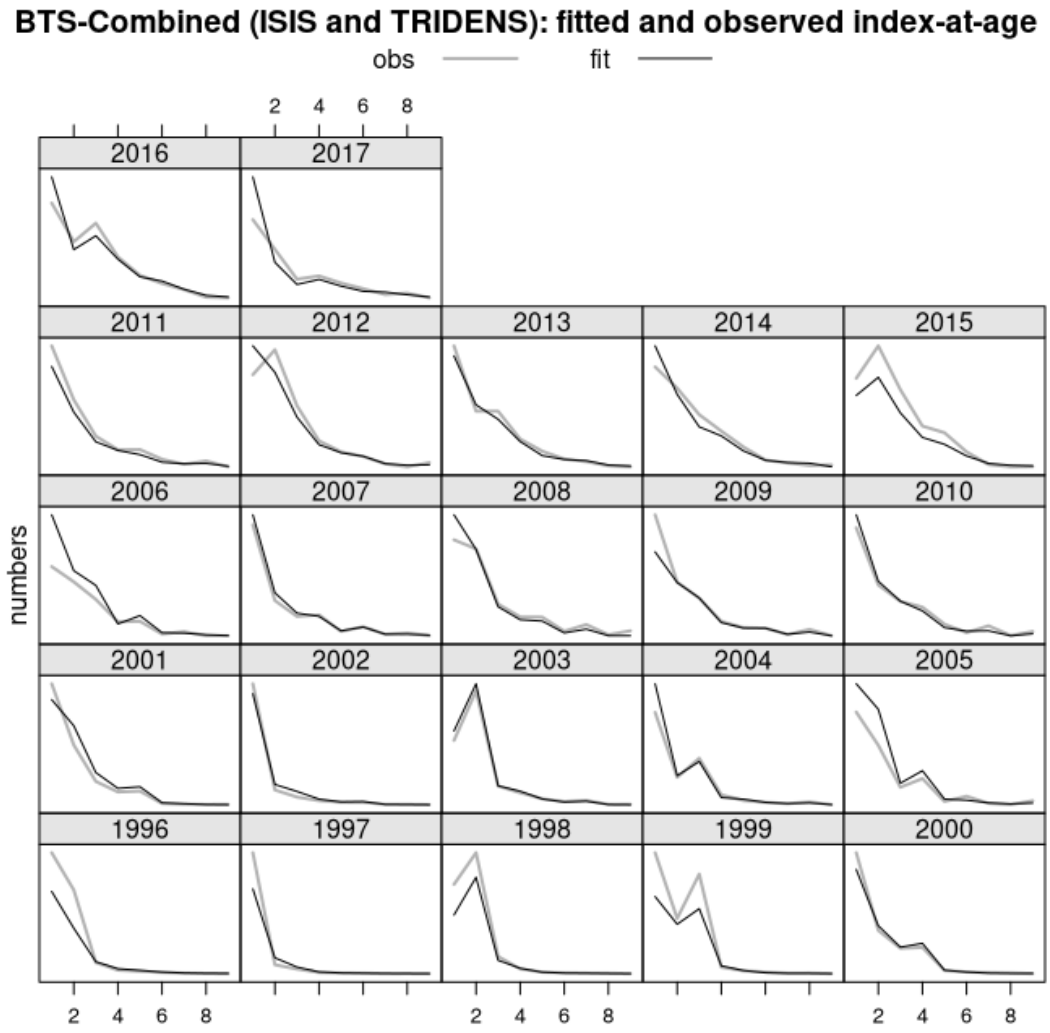


Figure 32: Predict and observed abundance-at-age

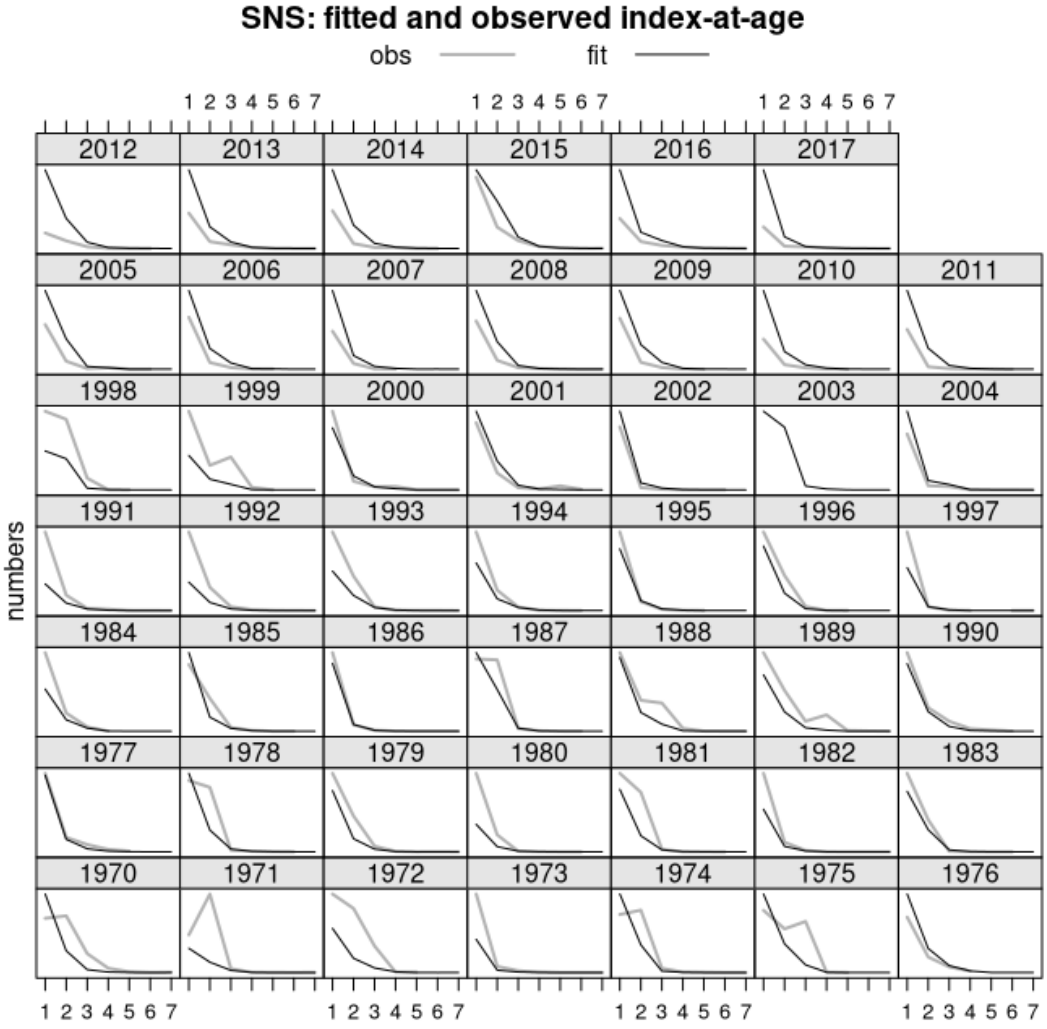


Figure 33: Predict and observed abundance-at-age

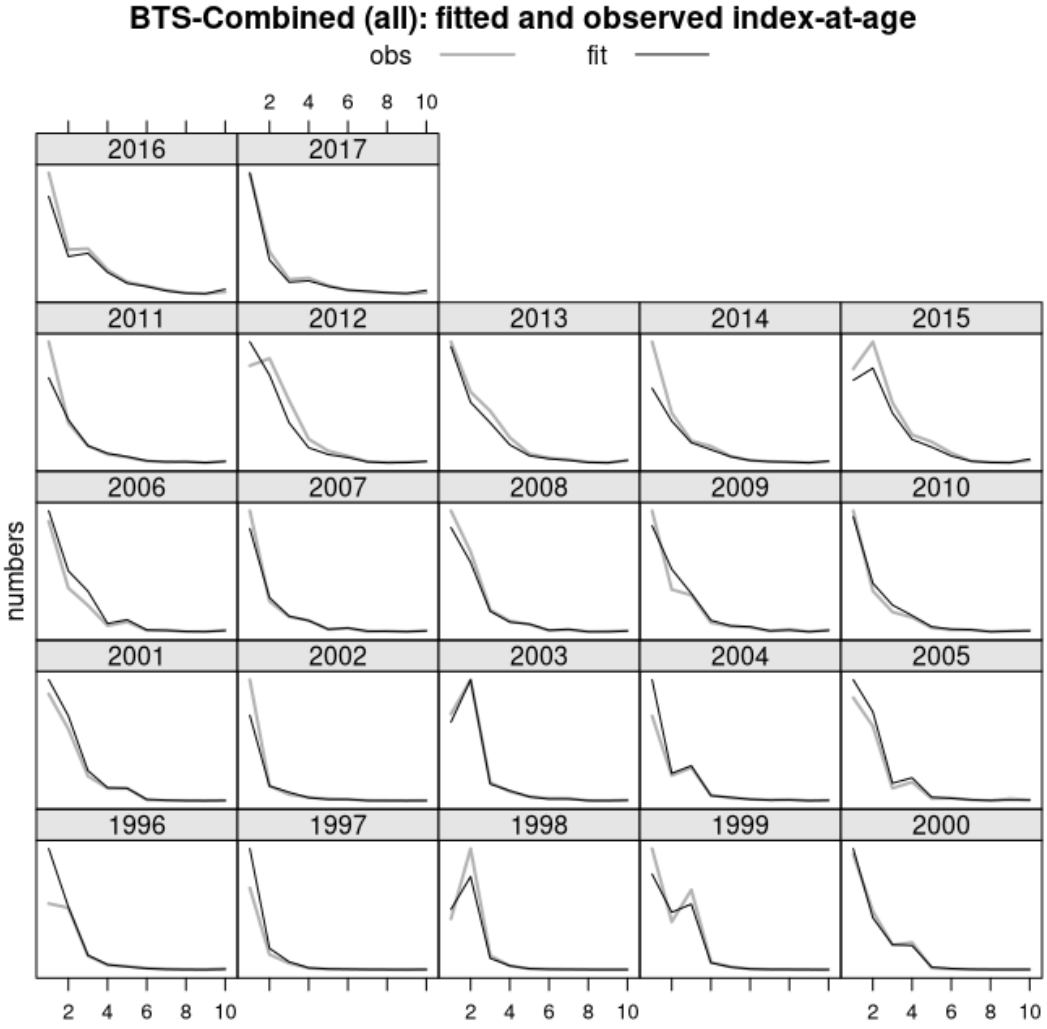


Figure 34: Predict and observed abundance-at-age

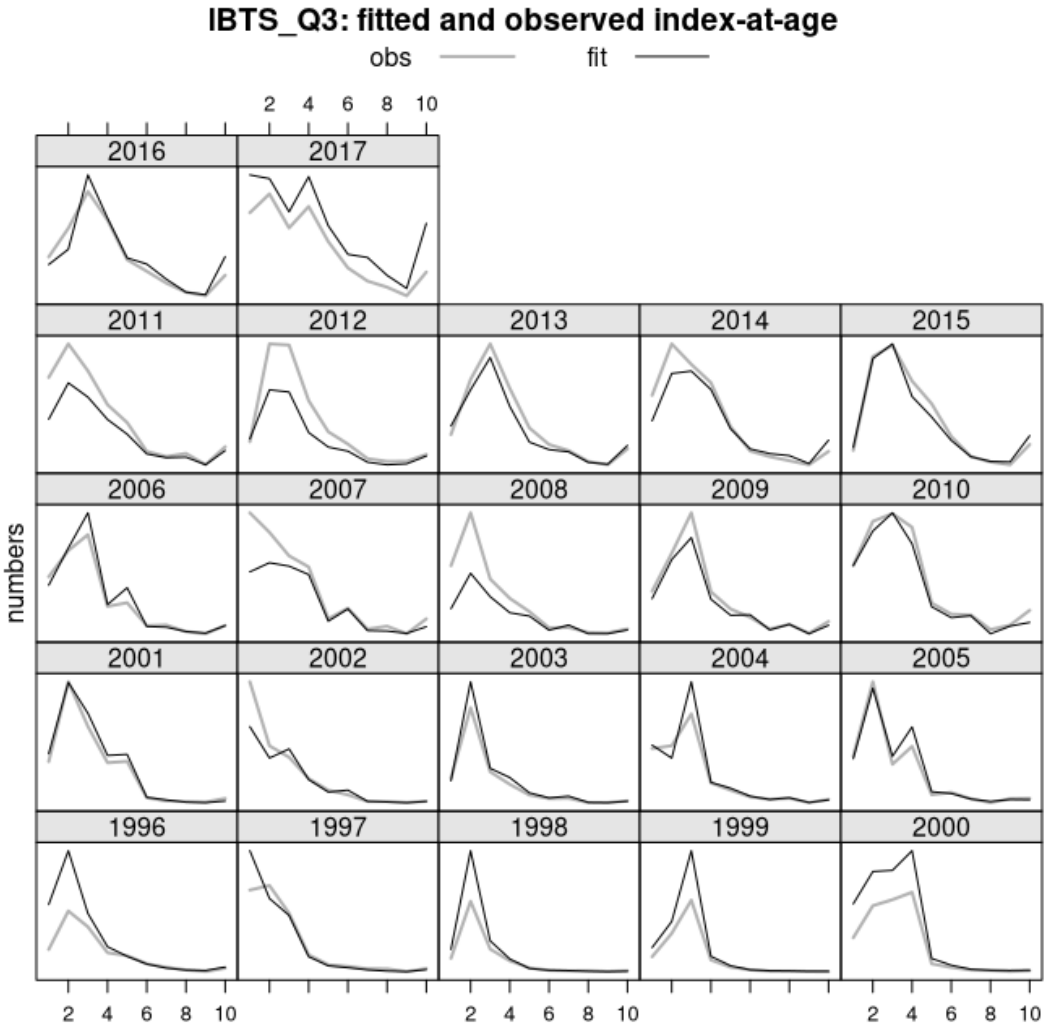


Figure 35: Predict and observed abundance-at-age

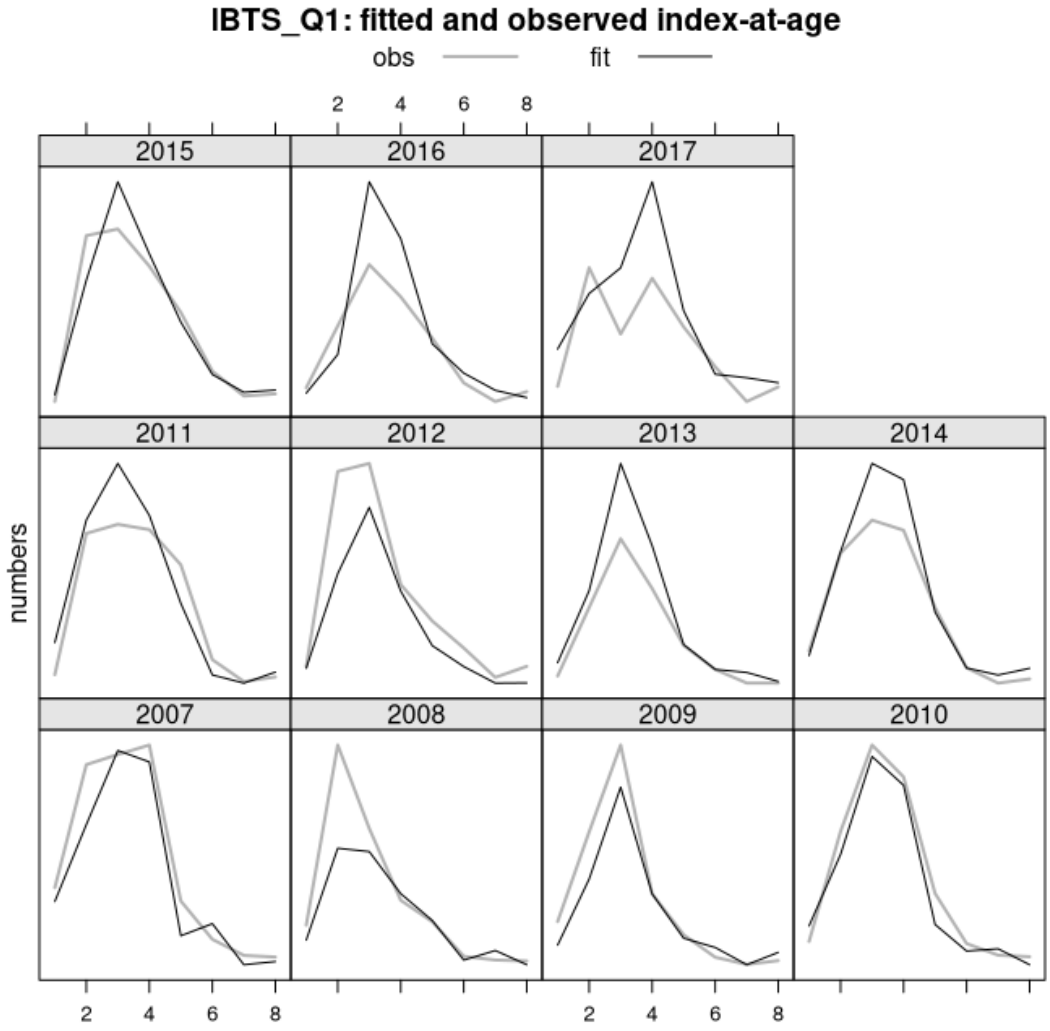


Figure 36: Predict and observed abundance-at-age

6.3 Aggreagted catch in weight

Although a statistical catch-at-age model assumes errors in catch-at-age and, as such, errors in the total catch in weight, there's still interest to evaluate how close the model estimates are of the observed catch in weight[2]. The implementation of this diagnopstics is done through the method `computeCatchDiagnostics()`, which can be visualized with `plot()` (Figure **??**).

```
c_d <- computeCatchDiagnostics(fit, ple4)
plot(c_d)
```

[2]Some analysts believe this is the most important diagnostic since total catch should be trusted. Needless to say we don't agree and consider reported catch in weight one of the less reliable pieces of information available for stock assessment.

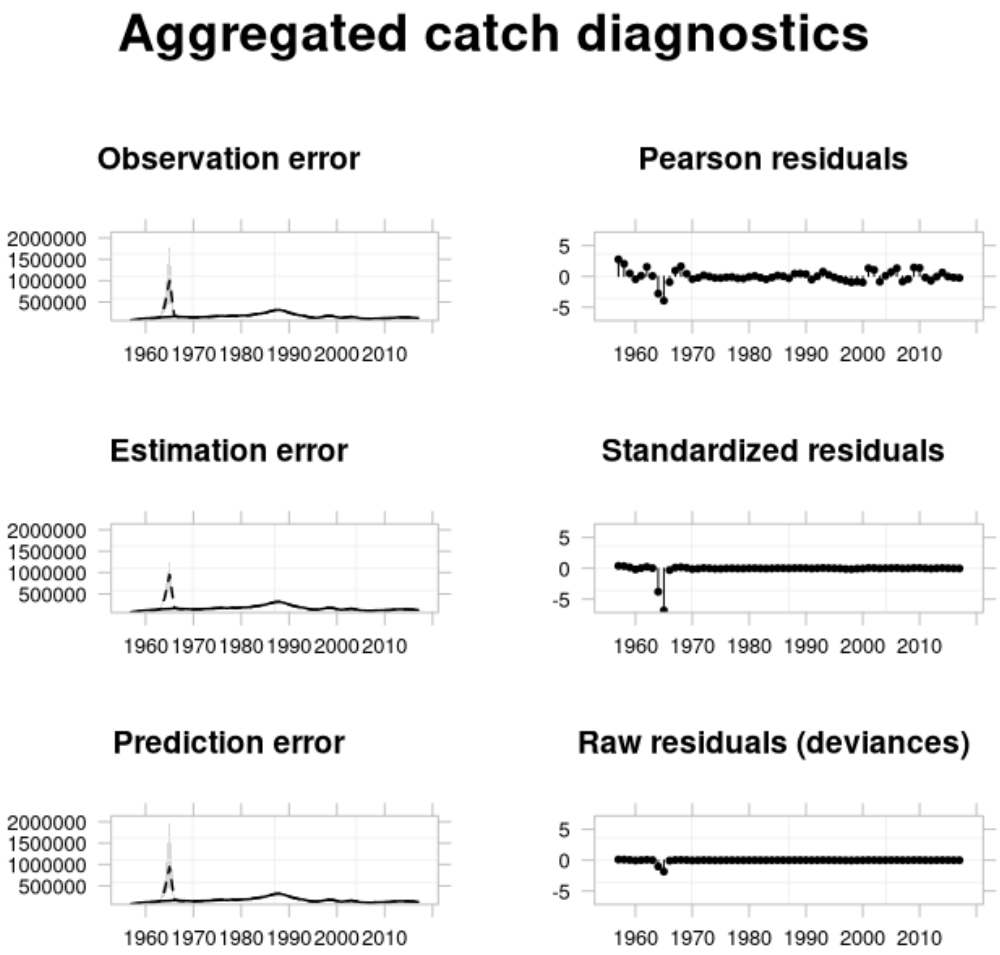


Figure 37: Diagnostics for age aggregated catch in weight

6.4 Fit summary, information and cross-validation metrics

To get information about the likelihood fit the method `fitSumm()` can be used to report number of parameters (`npar`), megative log-likelkihood (`nlogl`), `ADMB`maximum gradient par (`maxgrad`), number of observations (`nobs`), generalized cross validation score (`gcv`), convergence flag (`convergence`) and acceptance rate (`accrate`) relevant for MCMC fits only. The second part refers to the likelihood value for each component.

```
fitSumm(fit)
```

```
##              iters
##                  1
##   nopar   2.870000e+02
##   nlogl  -3.823602e+02
##   maxgrad  2.060663e-04
##   nobs     1.728000e+03
```

```
##   gcv           1.185680e-01
##   convergence   0.000000e+00
##   accrate                NA
##   nlogl_comp1 -1.058450e+03
##   nlogl_comp2  6.695020e+01
##   nlogl_comp3  5.643150e+01
##   nlogl_comp4  4.025550e+02
##   nlogl_comp5  5.836470e+01
##   nlogl_comp6  6.057810e+01
##   nlogl_comp7  3.121530e+01
```

Information criteria based metrics are reported with the methods:

```
AIC(fit)
```

```
## [1] -190.7205
```

```
BIC(fit)
```

```
## [1] 1374.784
```

7 Predict and simulate

8 The statistical catch-at-age stock assessment framework with MCMC

The previous methods were demonstrated using the maximum likelihood estimation method. However, `ADMB` can also use MCMC methods to fit the model. This section shows how the `sca` methods interface with ADMB to use the MCMC fits. For this section we'll use the hake assessment in mediterranean areas (gsa) 1, 5, 6 and 7.

The likelihood estimate is:

```
# ll
data(hke1567)
data(hke1567.idx)
fmod <- ~s(age, k = 4) + s(year, k = 8) + s(year, k = 8, by = as.numeric(age == 0)) +
    s(year, k = 8, by = as.numeric(age == 4))
qmod <- list(~I(1/(1 + exp(-age))))
fit <- sca(hke1567, hke1567.idx, fmodel = fmod, qmodel = qmod)
fit <- simulate(fit, 1000)
```

To run the MCMC method, one needs to configure a set of arguments, which is done by creating a `SCAMCMC` object. For details on the MCMC configuration in `ADMB` visit the ADMB website.

```r
# mcmc
mc <- SCAMCMC()
# check the default pars
mc
```

```
## An object of class "SCAMCMC"
## Slot "mcmc":
## [1] 10000
##
## Slot "mcsave":
## [1] 100
##
## Slot "mcscale":
## [1] NaN
##
## Slot "mcmult":
## [1] NaN
##
## Slot "mcrb":
## [1] NaN
##
## Slot "mcprobe":
## [1] NaN
##
## Slot "mcseed":
## [1] NaN
##
## Slot "mcdiag":
## [1] FALSE
##
## Slot "mcnoscale":
## [1] FALSE
##
## Slot "mcu":
## [1] FALSE
##
## Slot "hybrid":
## [1] FALSE
##
## Slot "hynstep":
## [1] NaN
##
## Slot "hyeps":
```

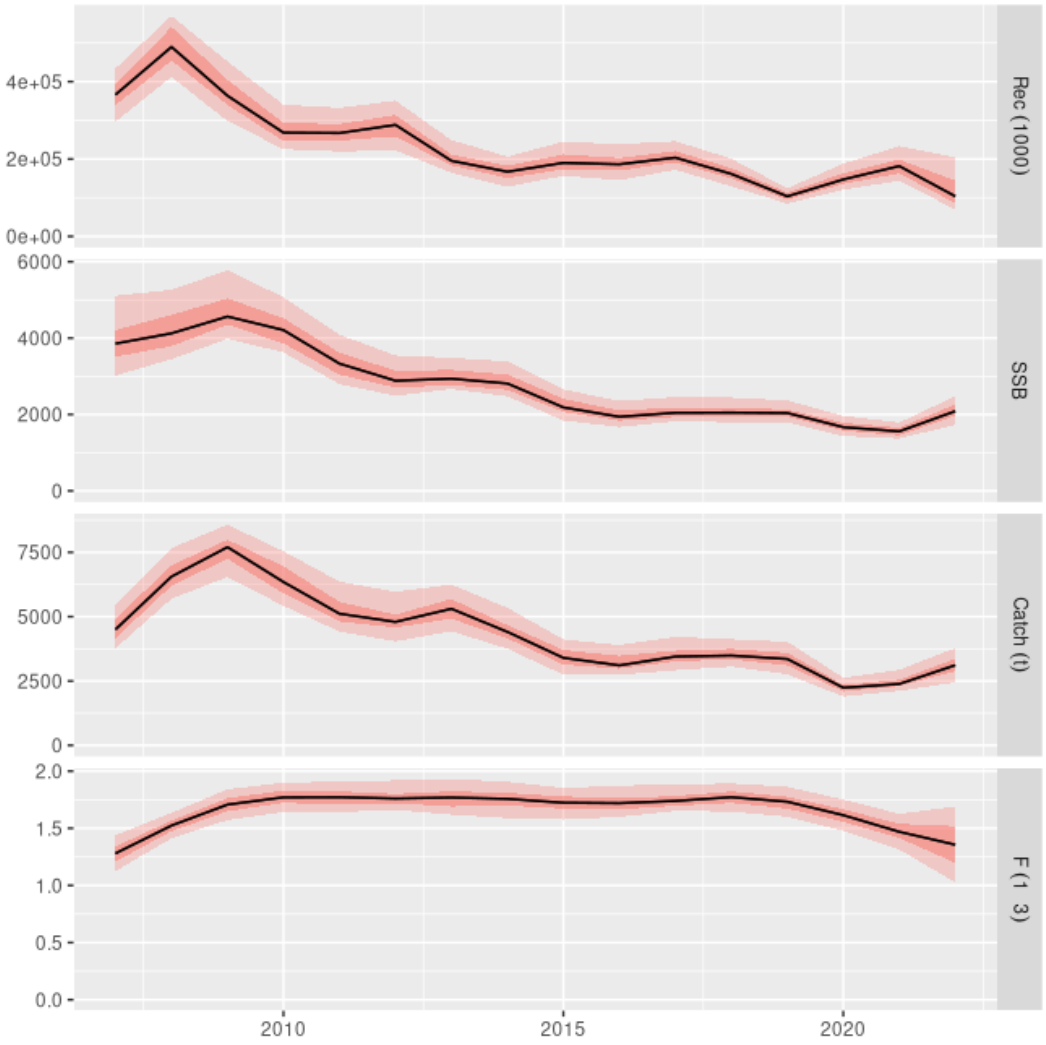
```
## [1] NaN
```

Defaults for now are ok, so lets fit the model. Note that the argument `fit` must be set to `MCMC` and the argument `mcmc` takes the `SCAMCMC` object. A major check when running MCMC is the acceptance rate, which should be around 0.3. This is a rule of thumb, for more information read the (extensive) literature on MCMC. The slot `fitSumm` stores that information.

```
# fit the model
fitmc00 <- sca(hke1567, hke1567.idx, fmodel = fmod, qmodel = qmod, fit = "MCMC",
    mcmc = mc)
# check acceptance rate
fitSumm(fitmc00)
```

```
##              iters
##                  1
##   nopar        52.0000
##   nlogl             NA
##   maxgrad           NA
##   nobs        176.0000
##   gcv               NA
##   convergence       NA
##   accrate       0.3271
```

```
plot(hke1567 + fitmc00)
```

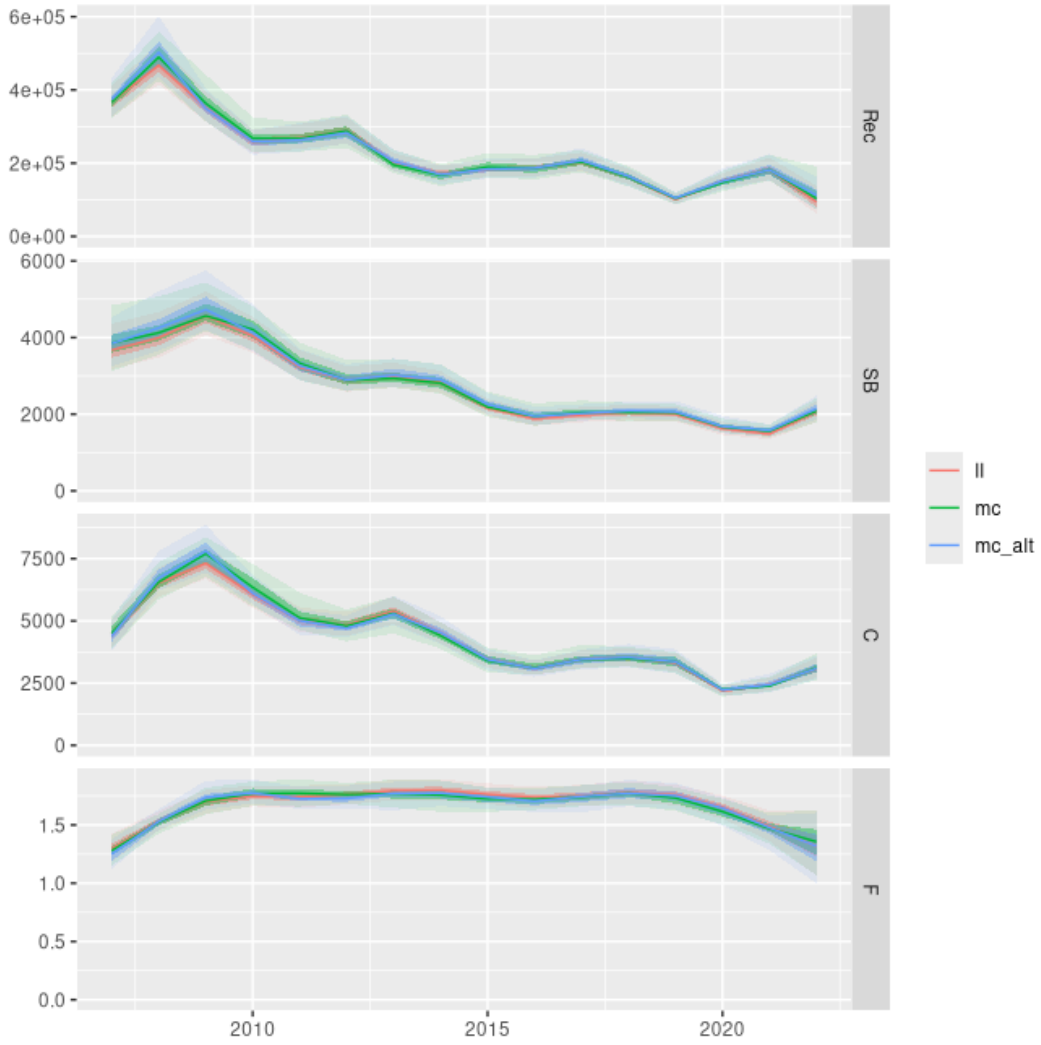


As mentioned above `ADMB` has several options for MCMC. Here we demonstrate one of them, `mcprobe` which sets a fat-tailed proposal distribution, as an example of how to use the `SCAMCMC` objects.

```
mc <- SCAMCMC(mcprobe = 0.45)
fitmc01 <- sca(hke1567, hke1567.idx, fmodel = fmod, qmodel = qmod, fit = "MCMC",
    mcmc = mc)
```

All fits together

```
plot(FLStocks(ll = hke1567 + fit, mc = hke1567 + fitmc00, mc_alt = hke1567 + fitmc01))
```



8.0.1 Diagnostics with CODA

We use the package `CODA` to run the diagnostics on MCMC fits. One needs to convert the `a4a`output into a `mcmc` CODA object over which several diagostics can be ran. The mcmc object is a matrix with the parameters (row = iters, cols= pars).

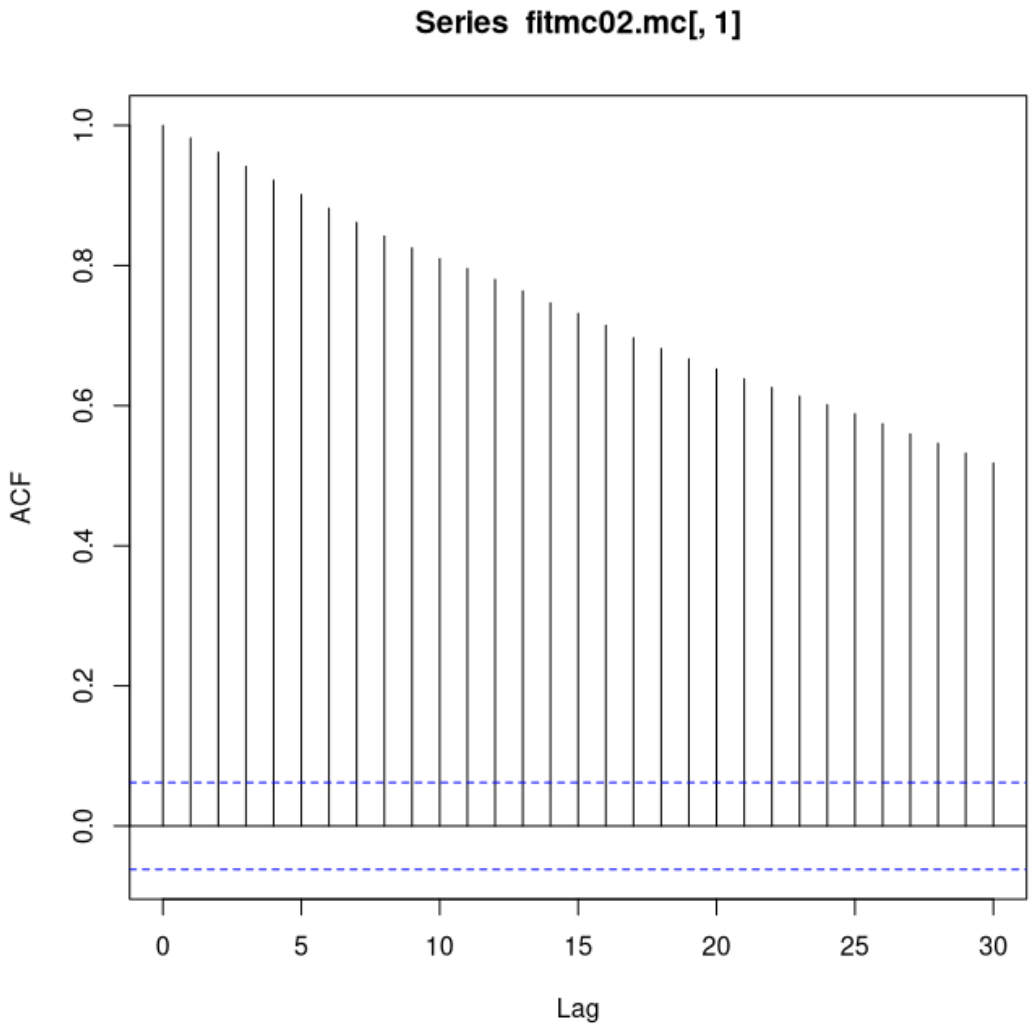
Common diagnostics for MCMC chains is to look at the burn-in period and auto-correlation. The first can be dealt by droping an initial set of iterations, which is done using the function `burnin`. The latter is set by the parameter `mcsave N`, which defines the iteration's saving rate (the inverse of the thinning rate). This is the rate at which samples of the parameters are saved, such that thinning is effectively discarding draws.

Next fit will run 1000 iterations and save every iter (`mcsave=1`).

```
library(coda)
mc <- SCAMCMC(mcmc = 1000, mcsave = 1)
fitmc02 <- sca(hke1567, hke1567.idx, fmodel = fmod, qmodel = qmod, fit = "MCMC",
    mcmc = mc)
fitmc02.mc <- FLa4a::as.mcmc(fitmc02)
```

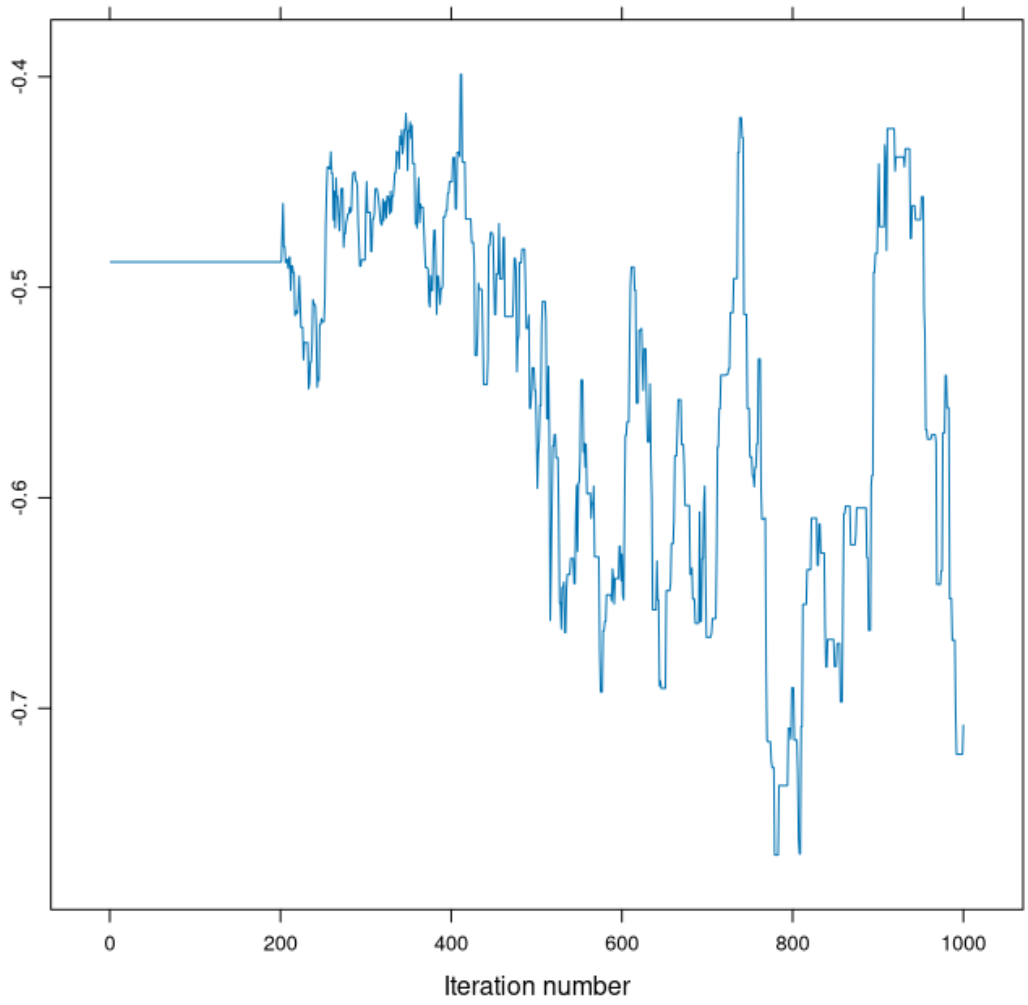
The autocorrelation plots will show the very strong correlation across samples, which we want to avoid. **??** shows autocorrelation for the first parameter.

```
acf(fitmc02.mc[, 1])
```



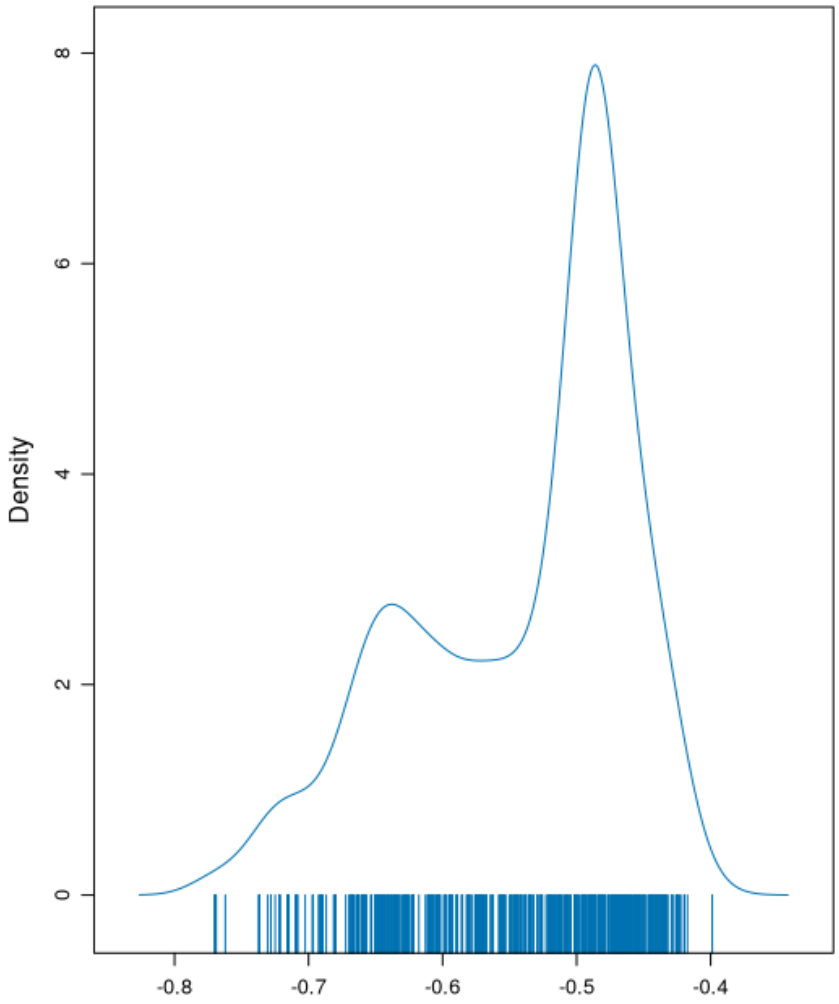
Ploting the chain for the parameter clearly shows the autocorrelation but also the burnin phase, where there's no information about the parameter. These iterations must to be dropped.

```
xyplot(fitmc02.mc[, 1])
```



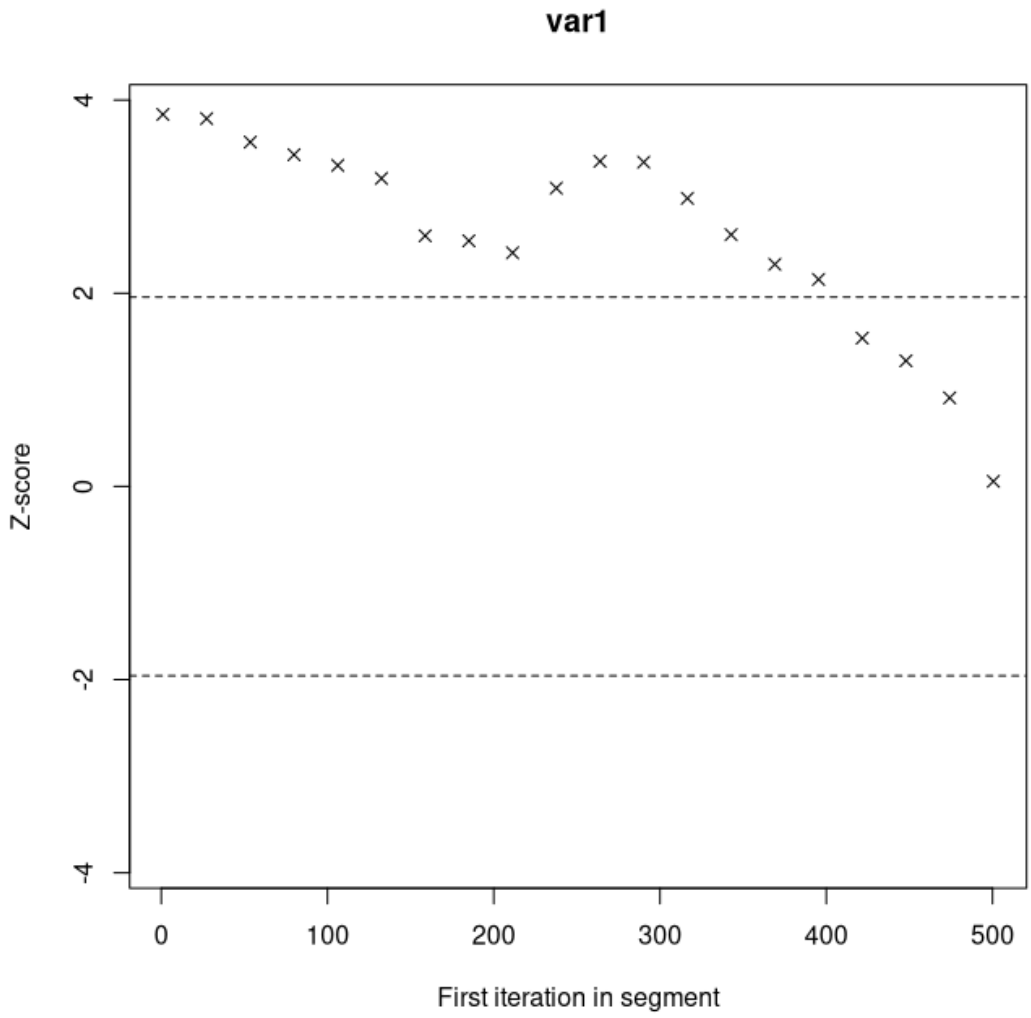
It's also important to check if the distribution of the parameters is normal, which can be done with the `densityplot`:

```
densityplot(fitmc02.mc[, 1])
```



Another interesting diagnostic is the Geweke-Brooks Z-score check. This diagnostic indicates if the first and last part of a sample from a Markov chain may not be drawn from the same distribution. It's useful to decide if the first few iterations should e discarded.

```
geweke.plot(fitmc02.mc[, 1])
```



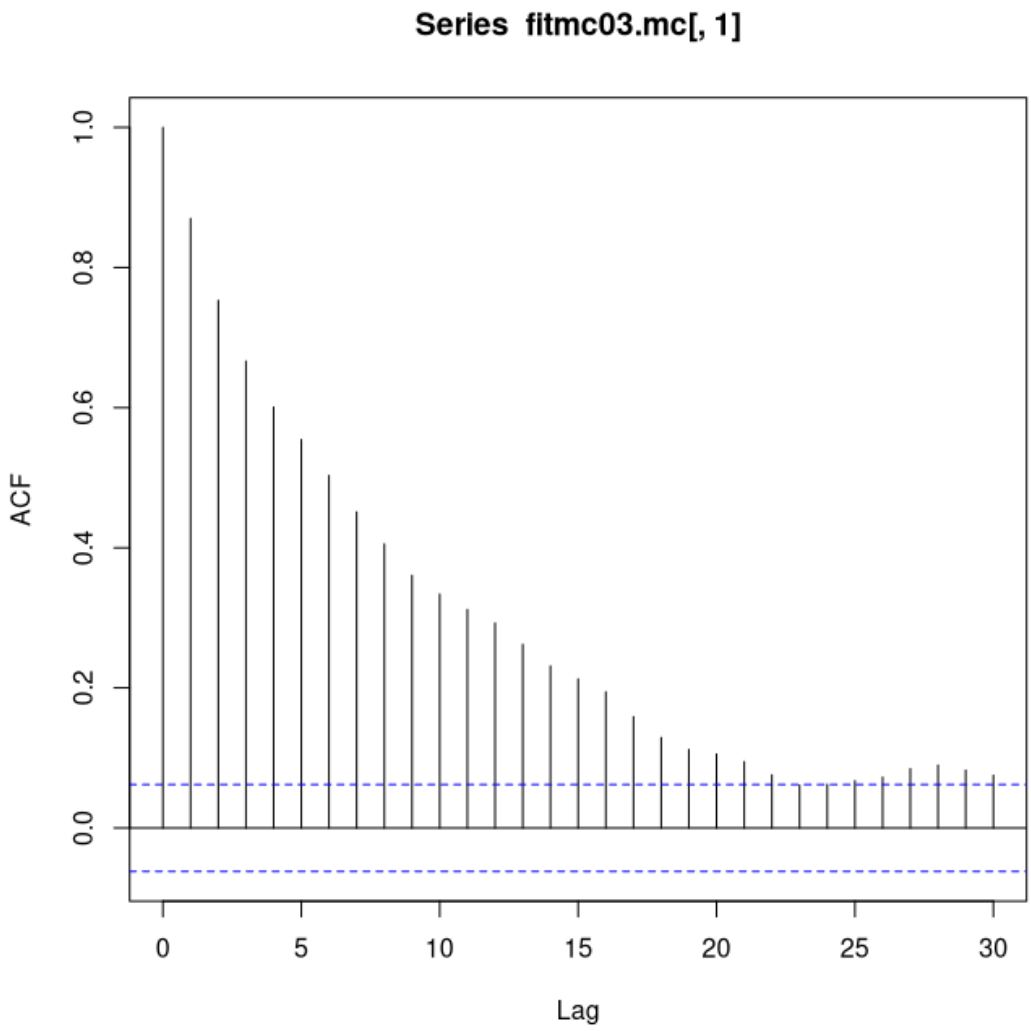
It's clear from the above diagnostics that a burnin phase of about 200 iterations should be considered. With relation to thining one needs to try several values until no autocorrelation exits.

Next fit will run 10000 iterations and save every iter (`mcsave=10`), so that the same 1000 iters are generated by the method.

```
mc <- SCAMCMC(mcmc = 10000, mcsave = 10)
fitmc03 <- sca(hke1567, hke1567.idx, fmodel = fmod, qmodel = qmod, fit = "MCMC",
    mcmc = mc)
fitmc03.mc <- FLa4a::as.mcmc(fitmc03)
```

The autocorrelation plots still shows a strong correlation across samples, although less than in the previous model.

```
acf(fitmc03.mc[, 1])
```



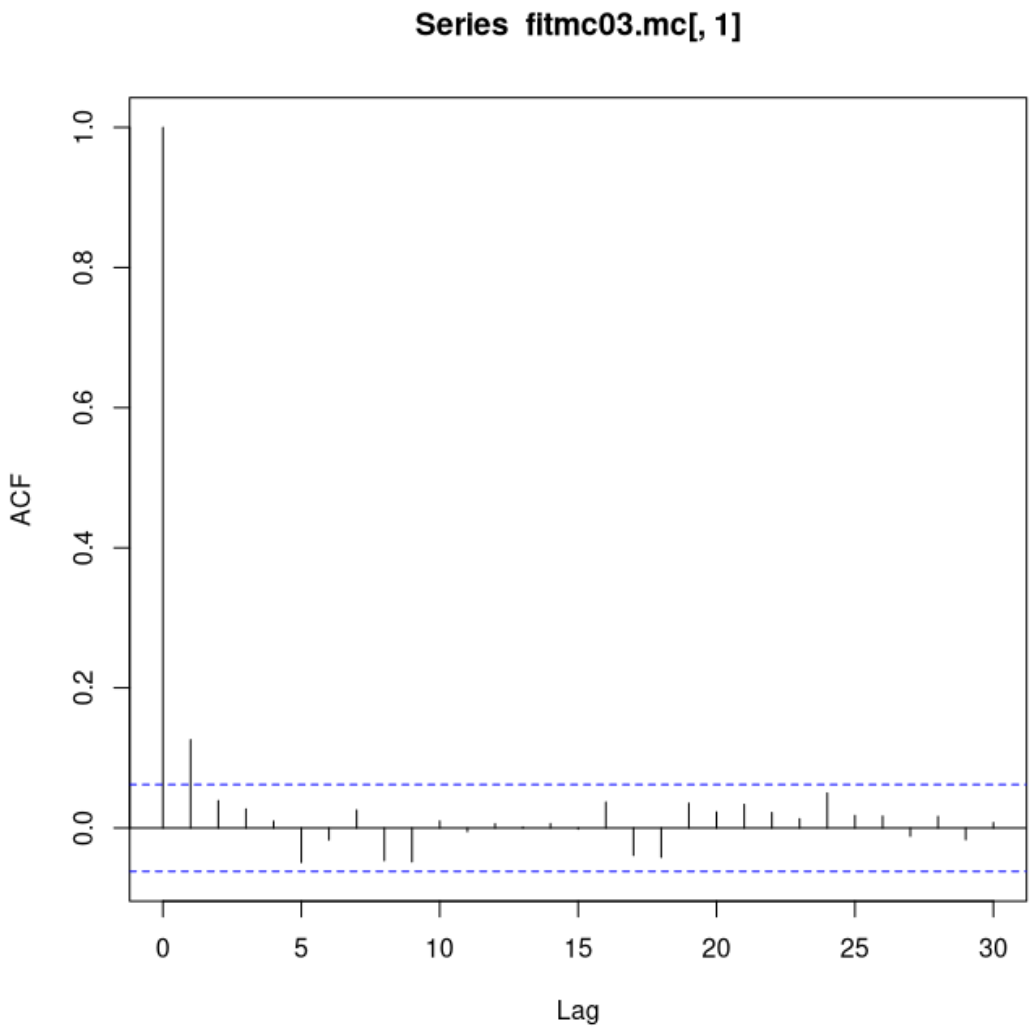
Next fit will run 100000 iterations and save every iter (`mcsave=100`), so that the same 1000 iters are generated by the method. Autocorrelation is much weaker, could still be reduced by increasing `mcsave`.

```
mc <- SCAMCMC(mcmc = 1e+05, mcsave = 100)
fitmc03 <- sca(hke1567, hke1567.idx, fmodel = fmod, qmodel = qmod, fit = "MCMC",
    mcmc = mc)
fitmc03.mc <- FLa4a::as.mcmc(fitmc03)
```

Next fit will run 200000 iterations and save every iter (`mcsave=200`).

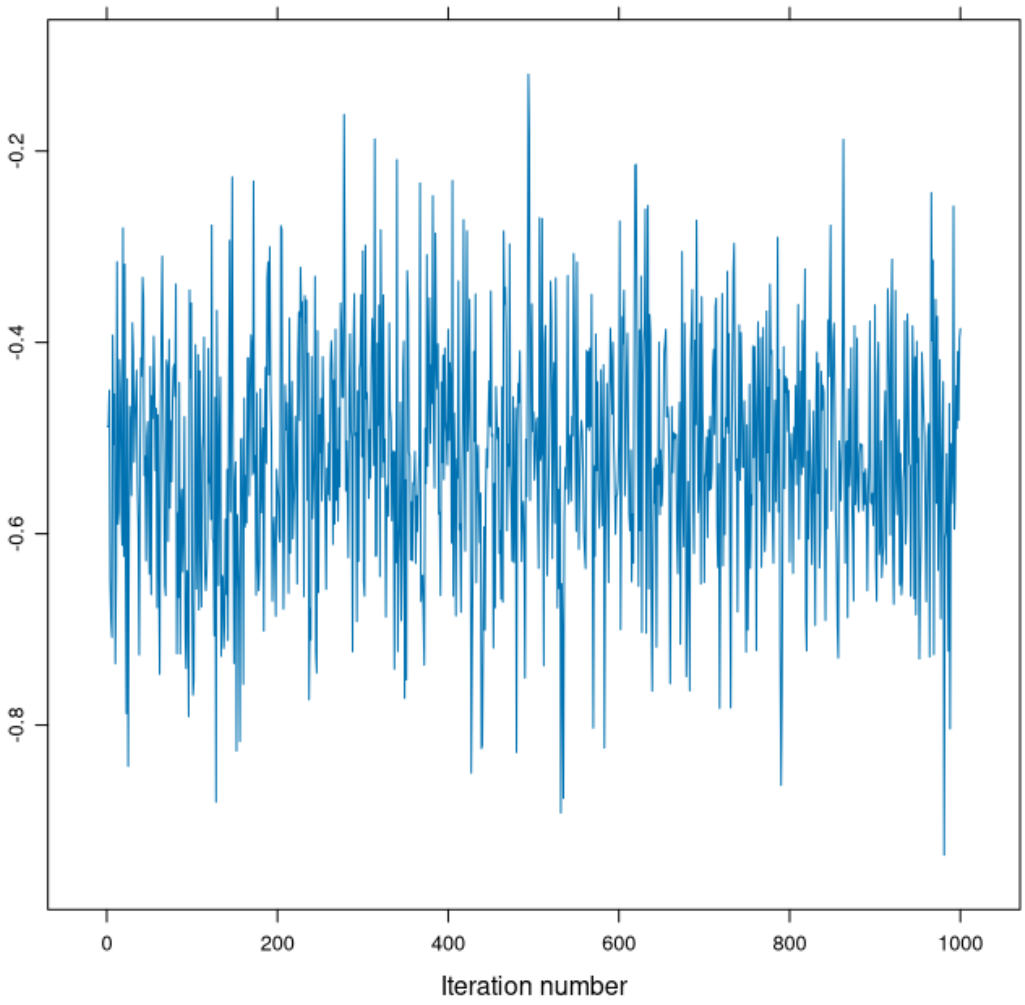
```
mc <- SCAMCMC(mcmc = 2e+05, mcsave = 200)
fitmc03 <- sca(hke1567, hke1567.idx, fmodel = fmod, qmodel = qmod, fit = "MCMC",
    mcmc = mc)
fitmc03.mc <- FLa4a::as.mcmc(fitmc03)
```

```
acf(fitmc03.mc[, 1])
```

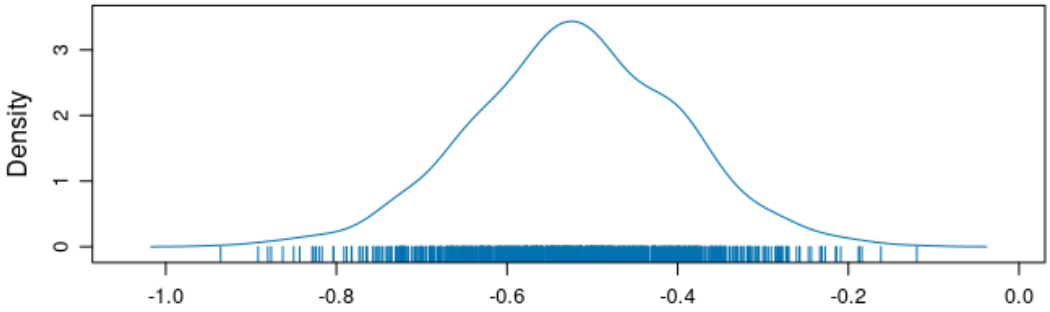


All diagnostics improved with the new thining rate although some other improvements can be done. Note this diagnostics should b echecked for all parameters. For the sake of space the demonstration uses only on the first.

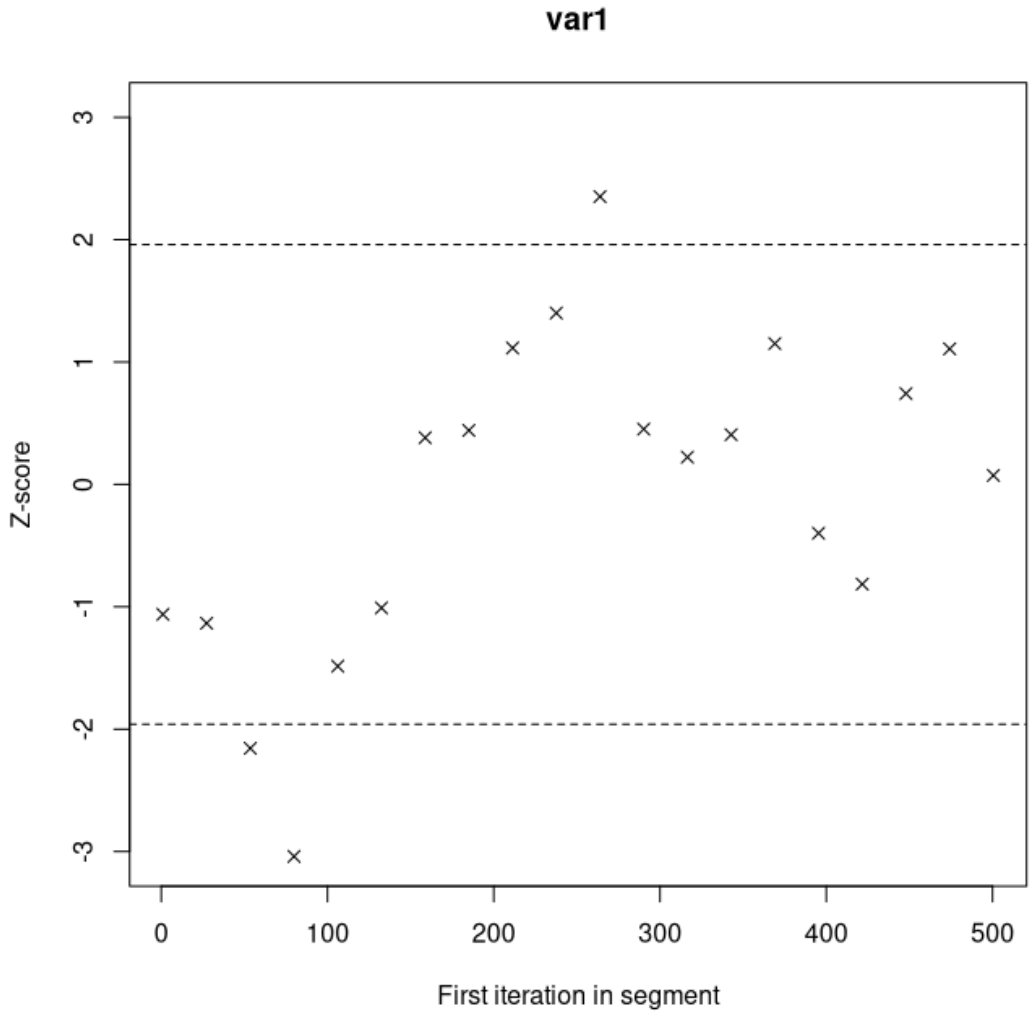
```
xyplot(fitmc03.mc[, 1])
```




```
densityplot(fitmc03.mc[, 1])
```




```
geweke.plot(fitmc03.mc[, 1])
```



The manual "A Guide for Bayesian Analysis in AD Model Builder" by Cole C. Monnahan, Melissa L. Muradian and Peter T. Kuriyam describe and explain a larger group of arguments that can be set when running MCMC with ADMB, which the engine `a4a`uses.

8.0.2 Confidence interval coverage and MCMC setup

8.0.3 Probabilistic assessment (RH code)

9 Advanced features

Not sure this section will stay or subsections will be merged in other sections ...

9.1 Assessing the coverage of confidence intervals

9.2 Propagate natural mortality uncertainty

9.3 Replicating itself (WCSAM) and parallel computing

10 A potential stock assessment workflow