Assessment for All initiative(a4a)
The a4a Stock Assessment Modelling Framework

Ernesto Jardim[1], Colin Millar[1,2,3], Finlay Scott[1], Iago Mosqueira[1], and Chato Osio[1]

[1]European Commission, Joint Research Centre, Sustainable resources directorate, Water and Marine Resources unit, 21027 Ispra (VA), Italy
[2]Marine Scotland Freshwater Laboratory, Faskally, Pitlochry, Perthshire PH16 5LB, UK
[3]International Council for the Exploration of the Sea (ICES), H. C. Andersens Boulevard 44-46, 1553 Copenhagen V, Denmark
[*]Corresponding author ernesto.jardim@jrc.ec.europa.eu

December 1, 2016

Contents

1 Installing and loading libraries

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

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

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

2 Background

In the `a4a` assessment model, the model structure is defined by submodels, which are the different parts of a statistical catch at age model that require structural assumptions.

There are 5 submodels in operation: a model for F-at-age, a model for the initial age structure, a model for recruitment, a (list) of model(s) for abundance indices catchability-at-age, and a list of models for the observation variance of catch-at-age and abundance indices. In practice, we fix the variance models and the initial age structure models, but in theory these can be changed.

The submodels form use linear models. This opens the possibility of using the linear modelling tools available in R: see for example the mgcv gam formulas, or factorial design formulas using `lm()`. In R's linear modelling language, a constant model is coded as ~ 1, while a slope over age would simply be $\sim age$. For example, we can write a traditional year/age separable F model like $\sim factor(age) + factor(year)$.

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

There are two basic types of assessments available in `a4a` : the management procedure fit and the full assessment fit. The management procedure fit does not compute estimates of covariances and is therefore quicker to execute, while the full assessment fit returns parameter estimates and their covariances at the expense of longer fitting time.

3 Stock assessment model details

The statistical catch at age model is based on the well known Baranov catch equation

$$e^{\mathrm{E}[\log C_{ay}]} = \frac{\mathbf{F_{ay}}}{\mathbf{F_{ay}} + M_{ay}} \left(1 - e^{-(\mathbf{F_{ay}}+M_{ay})}\right) \mathbf{R_{a=0,y}} e^{-\sum(\mathbf{F_{ay}}+M_{ay})}$$

and the common survey/index catchability

$$e^{\mathrm{E}[\log I_{ay}]} = \mathbf{Q_{ay}} \mathbf{R_{a=0,y}} e^{-\sum(\mathbf{F_{ay}}+M_{ay})}$$

where

$$C_{ay} \sim LogNormal(\mathrm{E}\left[\log C_{ay}\right], \sigma^{\mathbf{2}}_{\mathbf{ay}}) \qquad I \sim LogNormal(\mathrm{E}\left[\log I_{ay}\right], \tau^{\mathbf{2}}_{\mathbf{ay}})$$

The likelihood is defined by

$$\hat{\ell}_C = \sum_{ay} (w^{(c)}_{ay}\ \hat{\ell}_N(log\hat{C}_{ay}, \hat{\sigma}^2_{ay};\ \log C_{ay}))$$

$$\hat{\ell}_I = \sum_{s} \sum_{ay} (w^{(s)}_{ays}\ \hat{\ell}_N(log\hat{I}_{ays}, \hat{\tau}^2_{ays};\ \log I_{ays}))$$

$$\hat{\ell} = \hat{\ell}_C + \hat{\ell}_I$$

If there's a S/R model it's likelihood will be added

$$\hat{\ell}_{SR} = \sum_{y} (\hat{\ell}_N(log\tilde{R}_y, \phi^2_y;\ \log \hat{R}_y))$$

In these equations M is natural mortality, F fishing mortality, R recruitment, Q survey catchability, C catch and $\hat{\ell}$ is the negative log-likelihood of a normal distribution. All these variables are defined by age groups, although in the formula the indices were removed for better readability.

The quantities $\log F$, $\log Q$, $\log R$, $\log observation\,variances$ and $\log initial\,age\,structure$ (in bold in the equations above), need to be given a form, which is done using linear models. Recruitment is a special case. It is modelled as a fixed variance random effect, using the hard coded models Ricker, Beverton Holt, smooth hockeystick or geometric mean, which can use linear models for their parameters $\log a$ or $\log b$, where relevant. As an alternative the $\log R$ submodel can use a linear model like the other submodels

4 Quick and dirty

Here we show a simple example of using the assessment model using plaice in the North Sea. The default settings of the stock assessment model work reasonably well. It's an area of research that will improve with time. Note that because the survey index for plaice has missing values we get a warning saying that we assume these values are missing at random.

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

To inspect the stock assessment summary constituted of trends of fishing mortality (harvest), spawning stock biomass (SSB), catch and recruits, the user may add the `a4aFit` object to the original `FLStock` object using the `+` method and plot the result (Figure 1).

```
stk <- ple4 + fit
plot(stk)
```

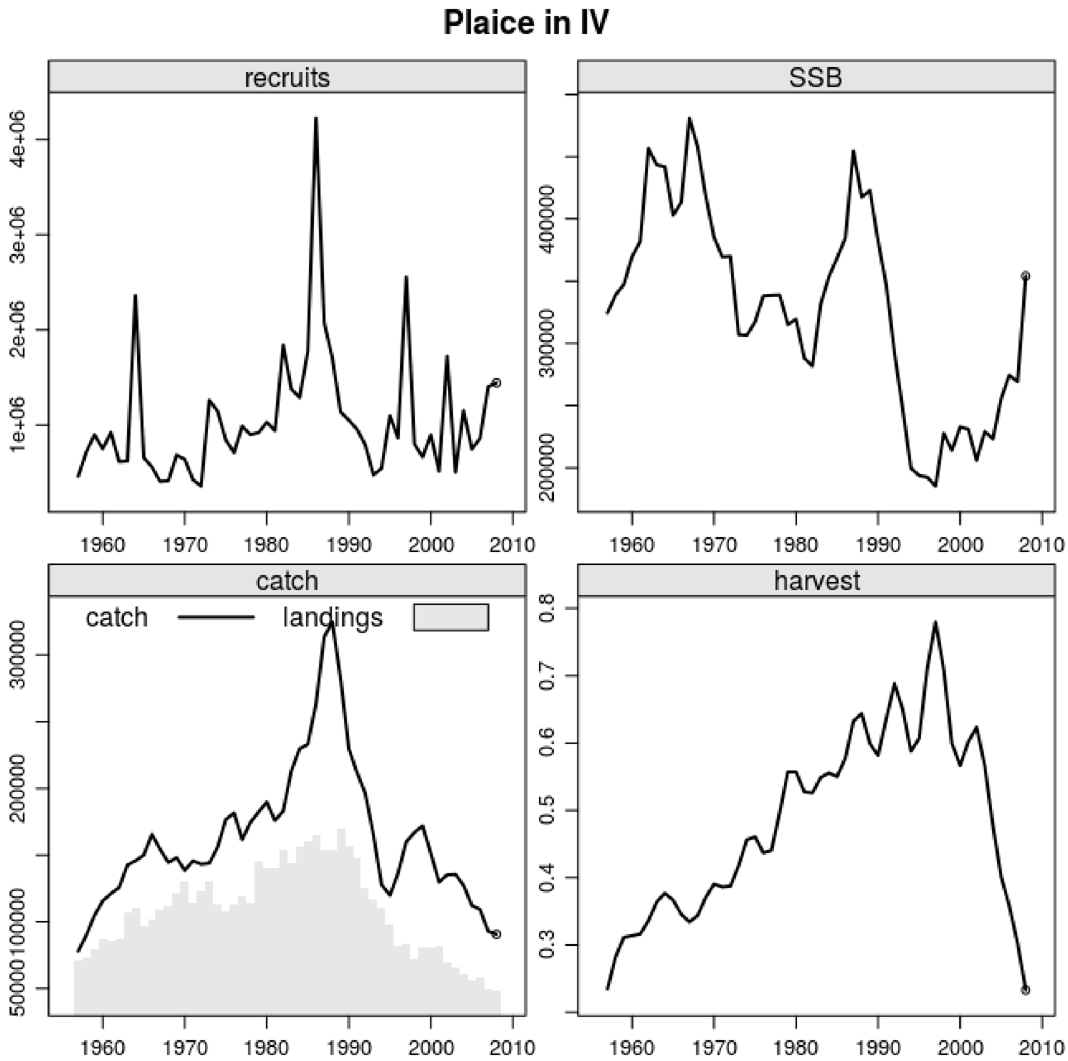


Figure 1: Stock summary for Plaice in ICES area IV, recruits, SSB (Stock Spawning Biomass), catch (catch and landings) and harvest (fishing mortality or F).

In more detail, one can plot a 3D representation of fishing mortality (Figure 2),

```
wireframe(data ~ year + age, data = harvest(fit), zlab = "F", par.settings = list(axis.line = lis
    box.3d = list(col = "transparent")), scales = list(col = 1))
```

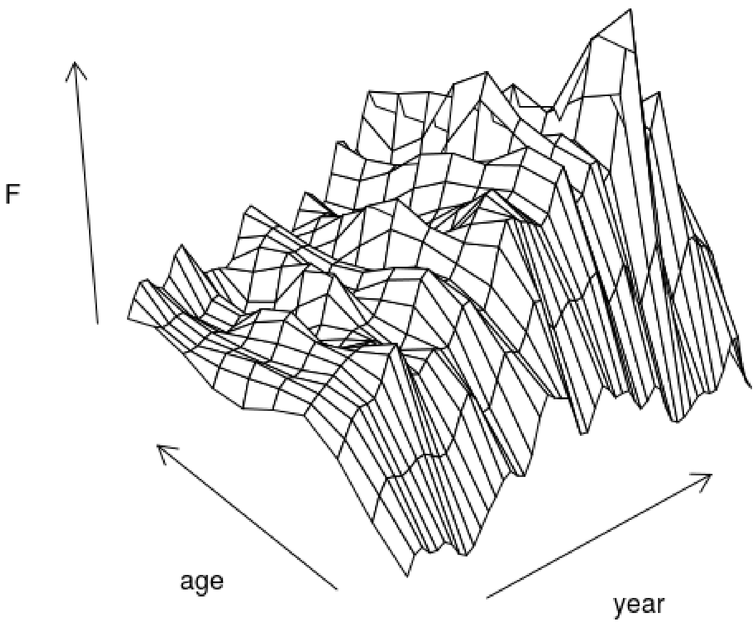


Figure 2: 3D contour plot of estimated fishing mortality at age and year

population abundance (Figure 3) is displaid as a 3D wireframe,

```
wireframe(data ~ year + age, data = stock.n(fit), zlab = "N", par.settings = list(axis.line = lis
    box.3d = list(col = "transparent")), scales = list(col = 1))
```

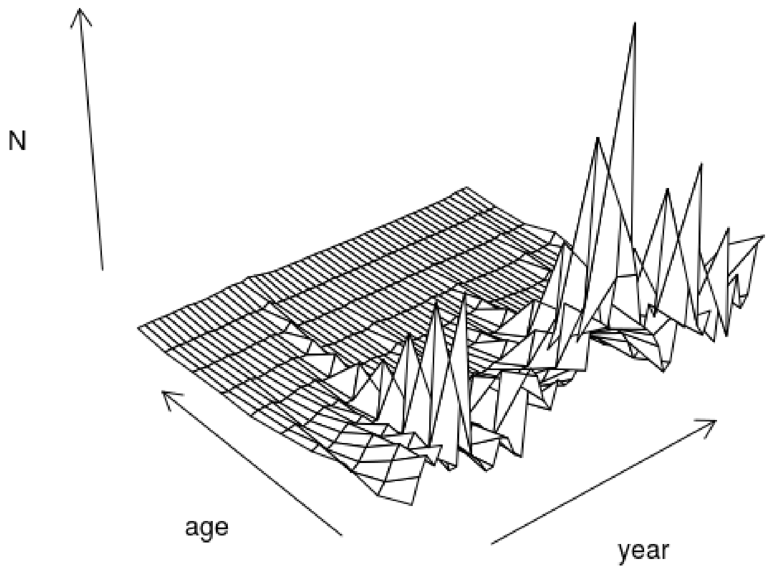


Figure 3: Population abundance by age and year

as well as catch-at-age (Figure 4).

```
wireframe(data ~ year + age, data = catch.n(fit), zlab = "C", par.settings = list(axis.line = lis
    box.3d = list(col = "transparent")), scales = list(col = 1))
```

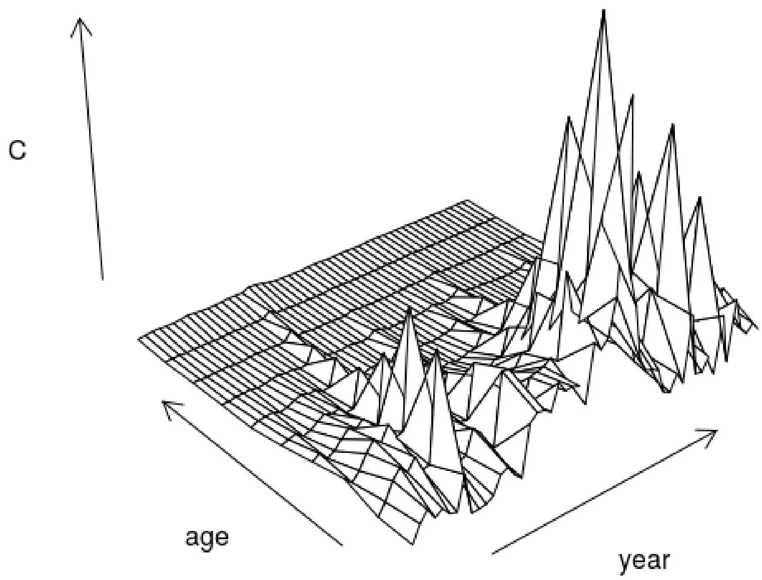


Figure 4: Catches in number of individuals by age and year

5 Diagnostics

A set of plots to inspect the fit quality and assumptions are implemented. The most common is to look at standardized log-residuals to check for biased results or large variances. Note that the standardization should produce residuals with variance 1, which means that most residual values should be between ~ -2 and ~ 2. These residuals also allow the user to check for deviances from the log-normal assumption.

The `residuals()` method will compute standardized residuals which can be plotted using a set of packed methods.

```
res <- residuals(fit, ple4, ple4.indices)
```

Figure 5 shows a scatterplot of residuals by age and survey, with a smoother to guide (or misguide ...) your visual analysis.

```
plot(res)
```

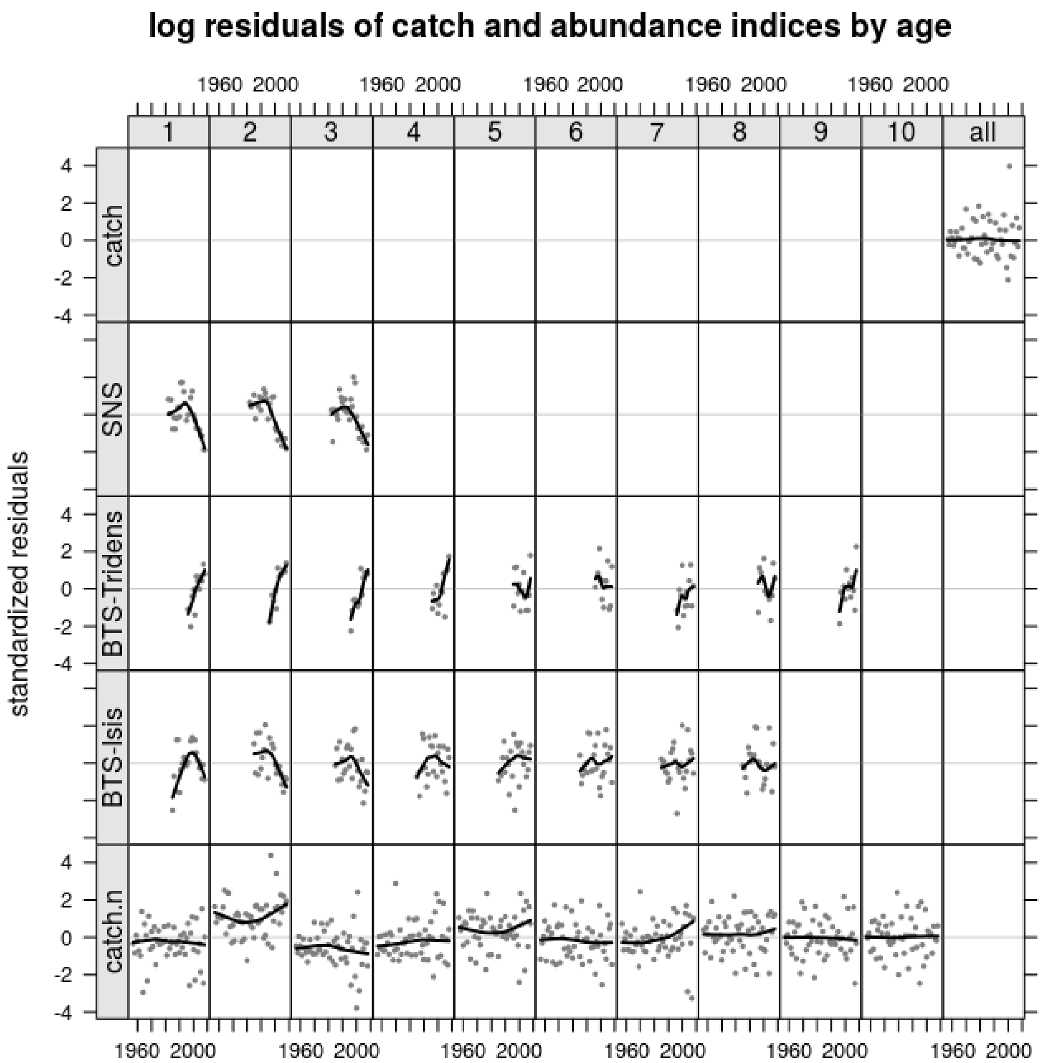


Figure 5: Standardized residuals for abundance indices (SNS, BTS Tridens and BTS Isis) and for catch numbers (catch.n). Each panel is coded by age class, dots represent standardized residuals and lines a simple smoother.

The common bubble plot by year and age for each survey are shown in Figure 6. It shows the same information as Figure 5 but in a multivariate perspective.

```r
bubbles(res)
```

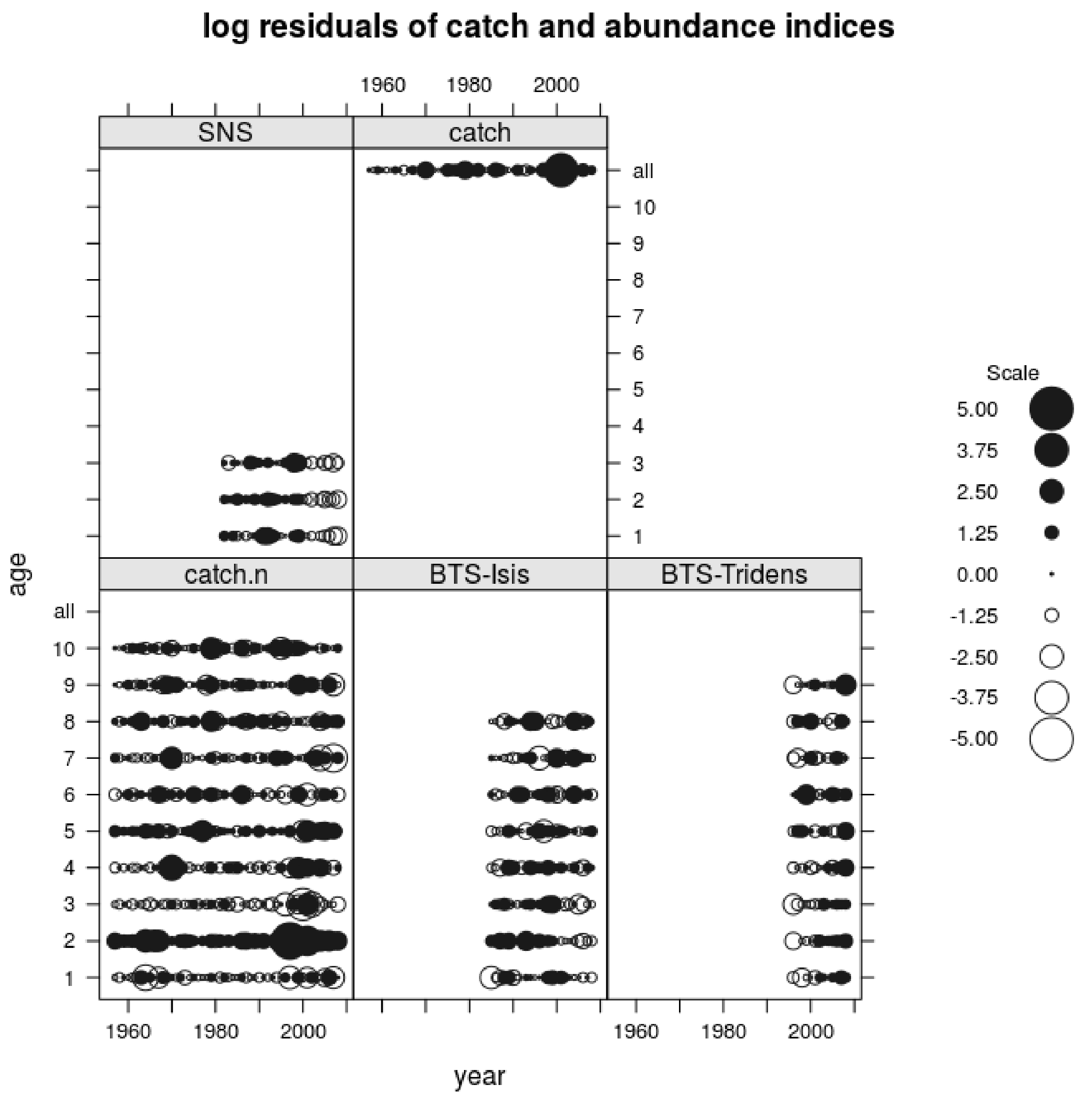


Figure 6: Bubbles plot of standardized residuals for abundance indices (SNS, BTS Tridens and BTS Isis) and for catch numbers (catch.n).

Figure 7 shows a quantile-quantile plot to assess how well do the residuals match the normal distribution.

```
qqmath(res)
```

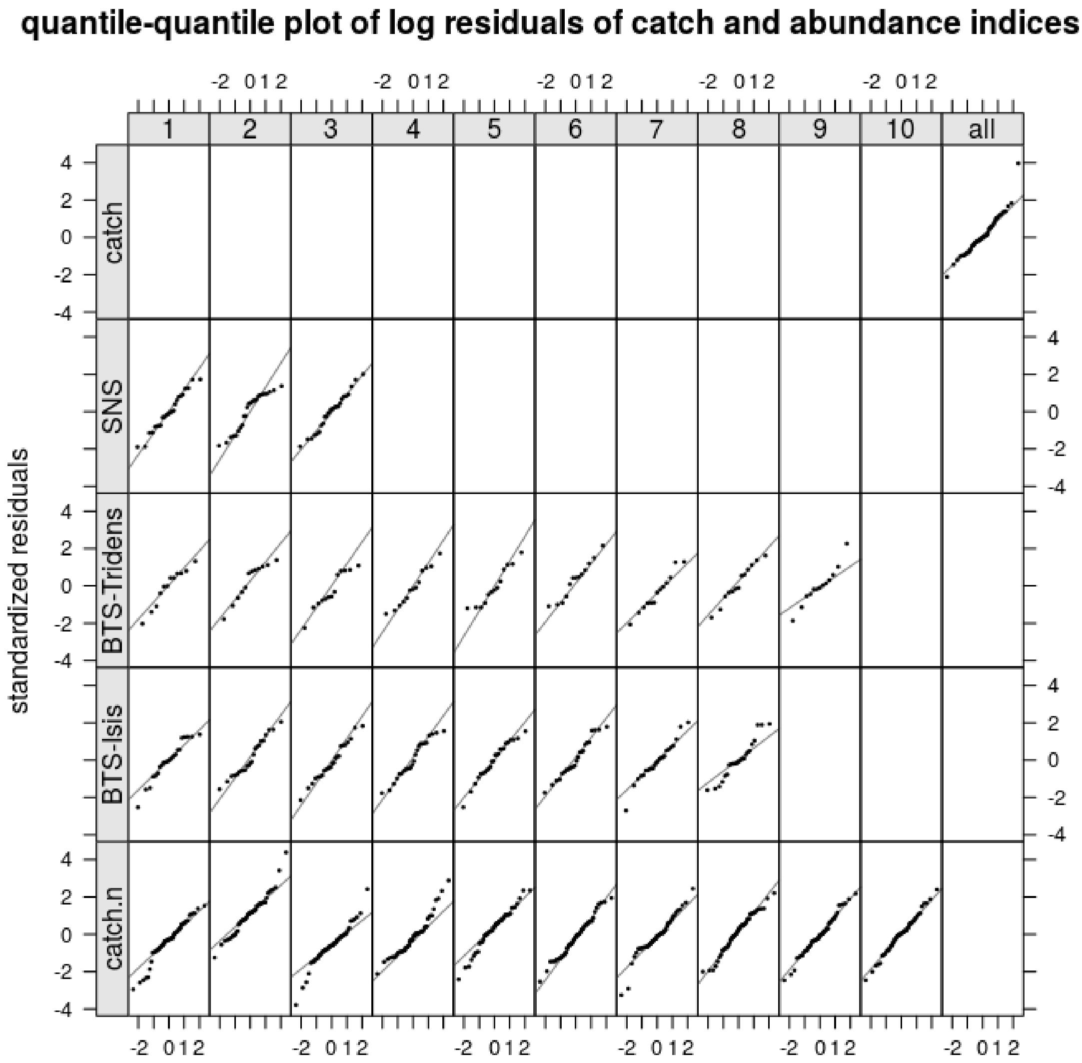


Figure 7: Quantile-quantile plot of standardized residuals for abundance indices (SNS, BTS Tridens and BTS Isis) and for catch numbers (catch.n). Each panel is coded by age class, dots represent standardized residuals and lines the normal distribution quantiles.

To have a look at how well is the model predicting catches and abundance, one can use the `plot()` method with the *a4aFit* object and the *FLStock* (Figure 8) object or the *FLIndex* object (Figure ??).

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

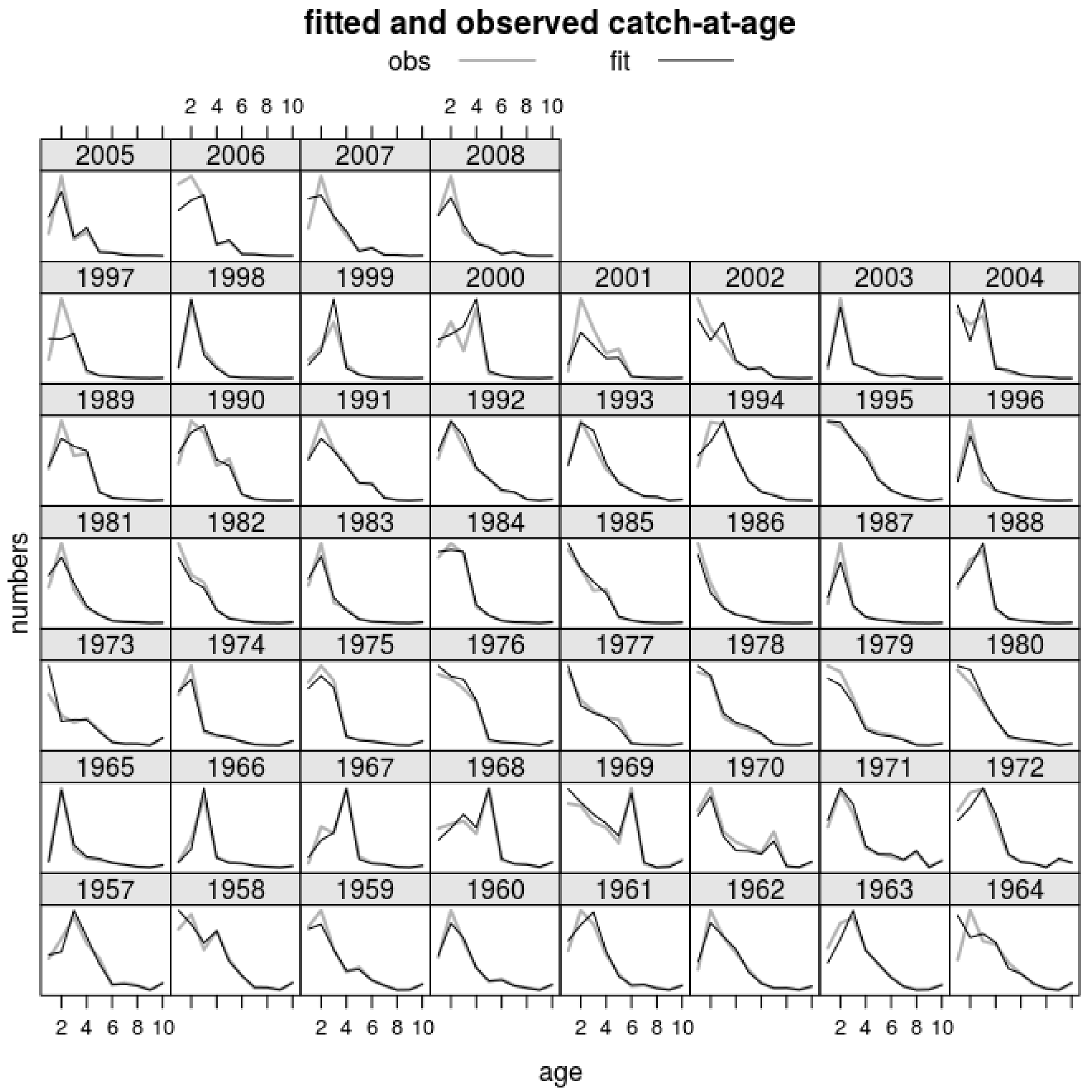


Figure 8: Predict and observed catch-at-age

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

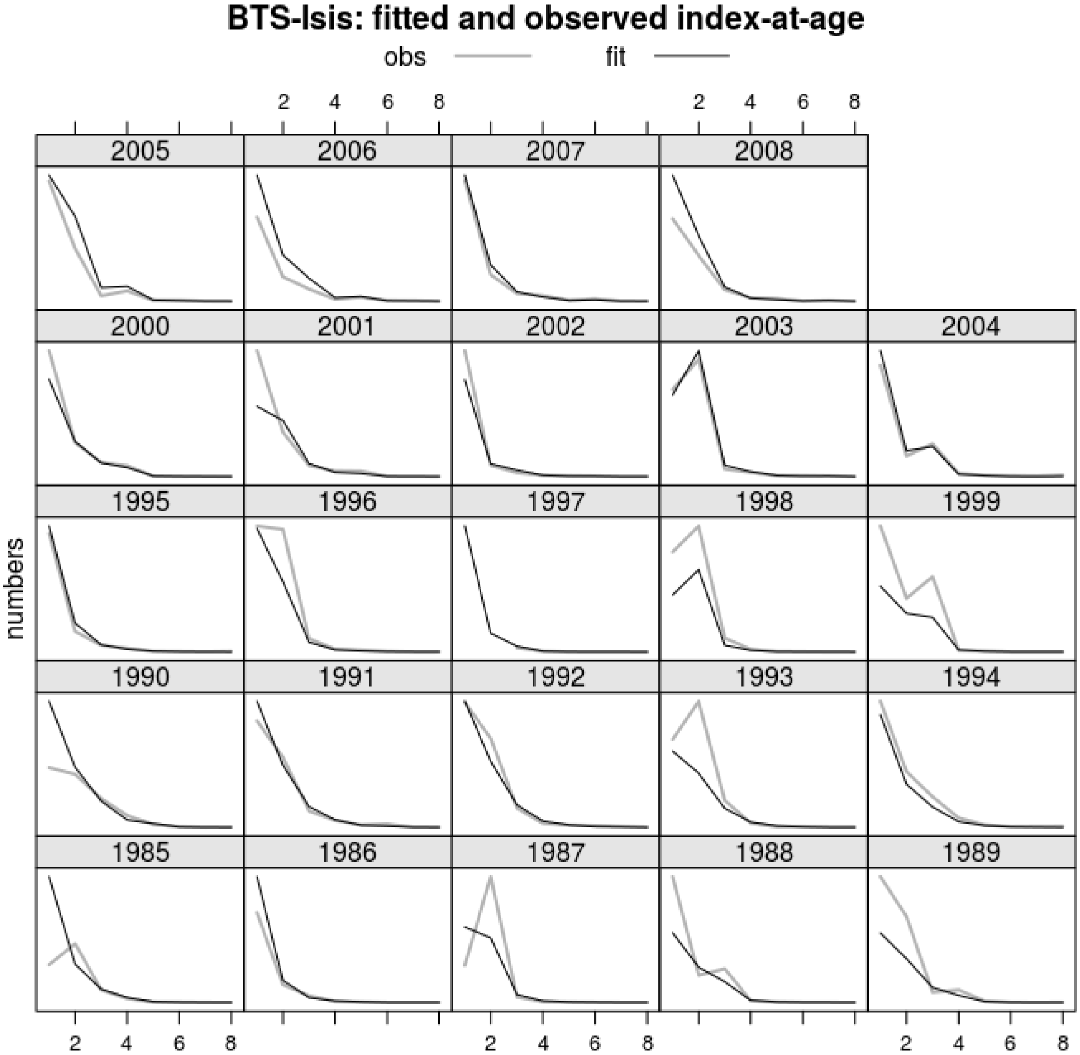


Figure 9: Predict and observed abundance-at-age

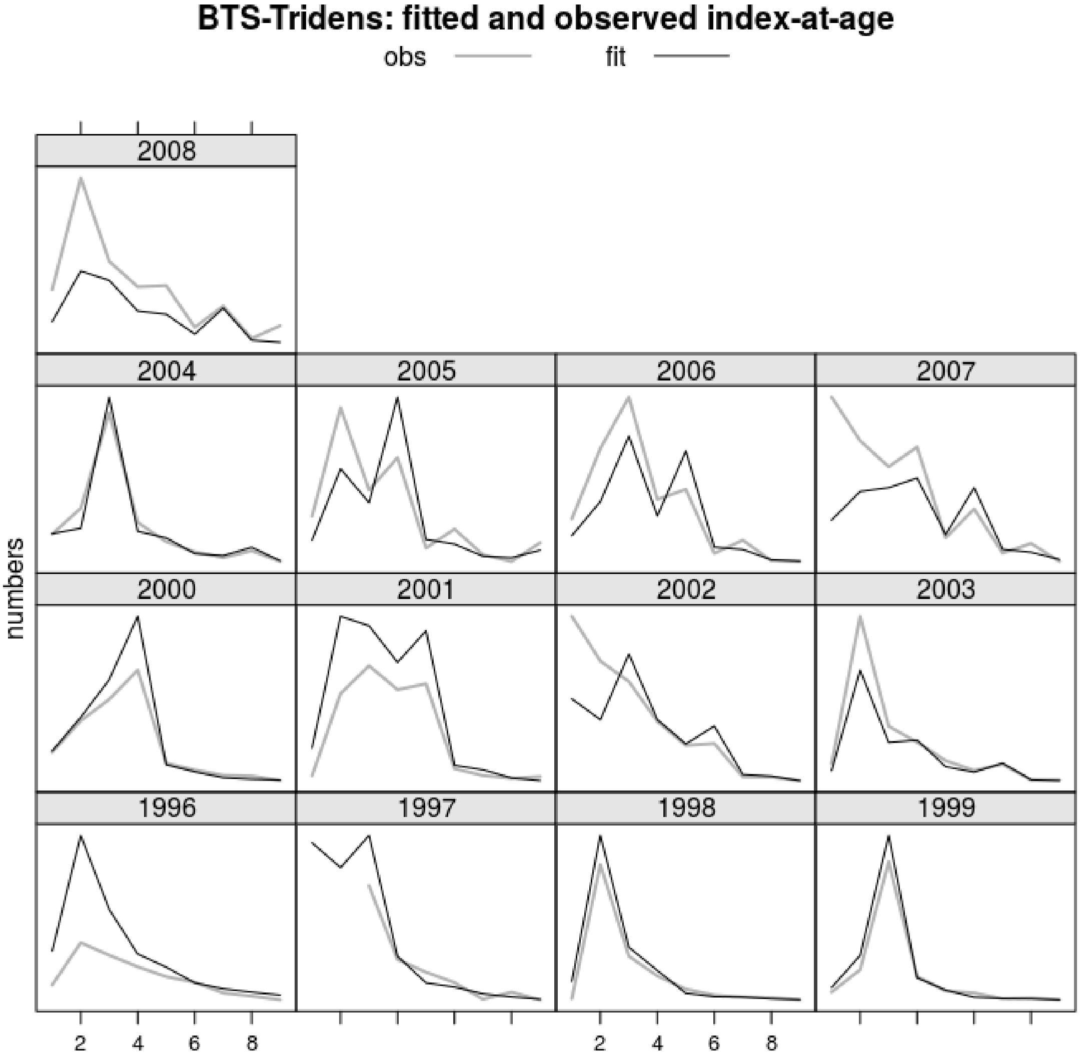


Figure 10: Predict and observed abundance-at-age

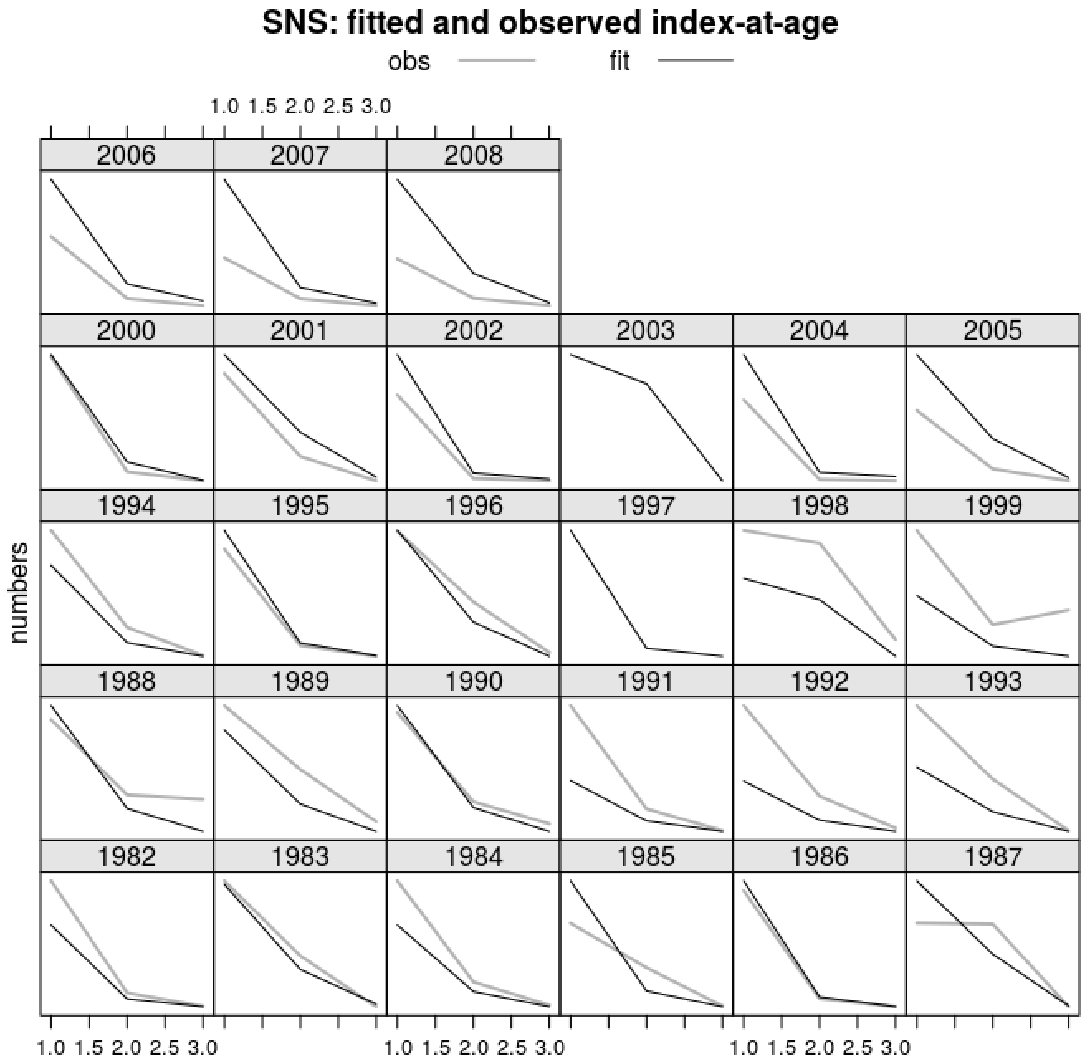


Figure 11: Predict and observed abundance-at-age

To get information about the likelihood fit the method `fitSumm()` will extract information about likelihood, number of parameters, etc, and the methods `AIC()` and `BIC()` will compute the information criteria.

```
fitSumm(fit)

##              iters
##                  1
##   nopar   2.350000e+02
##   nlogl   6.885500e+01
```

```
##    maxgrad      7.254980e-05
##    nobs         9.010000e+02
##    gcv          5.874789e-02
##    convergence            NA
##    accrate                NA
```

```
AIC(fit)
```

```
## [1] 607.71
```

```
BIC(fit)
```

```
## [1] 1736.534
```

6 The statistical catch-at-age stock assessment framework - the sca method

The statistical catch at age (`sca()`) method used in the previous section with the default settings, can be parametrized to control other features of the stock assessment framework. The most interesting ones are the submodels for fishing mortality (F), catchability (Q) and recruitment (R).

An important argument for `sca()` is the type of fit, which controls if a full assessment will be performed or a management procedure type of assessment. The argument is called `fit` and can have the values 'assessment' for a full assessemt or 'MP' for a simpler assessment. By default `sca()` uses `fit='MP'`.

We'll start by looking at the submodel for F, then Q and finally R.

Please note that each of these model *forms* have not been tuned to the data. The degrees of freedom of each model can be better tuned to the data by using model selection procedures such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC), etc.

6.1 Fishing mortality submodel

We will now take a look at some examples for F models and the forms that we can get. We'll fix the Q and R submodels.

Lets start with a separable model in which age and year effects are modelled as dummy variables, using the `factor` coding provided by R (Figure 12).

```
qmod <- list(~factor(age))
fmod <- ~factor(age) + factor(year)
srmod <- ~factor(year)
fit <- sca(stock = ple4, indices = ple4.indices[1], fmodel = fmod, qmodel = qmod,
    srmodel = srmod)
```

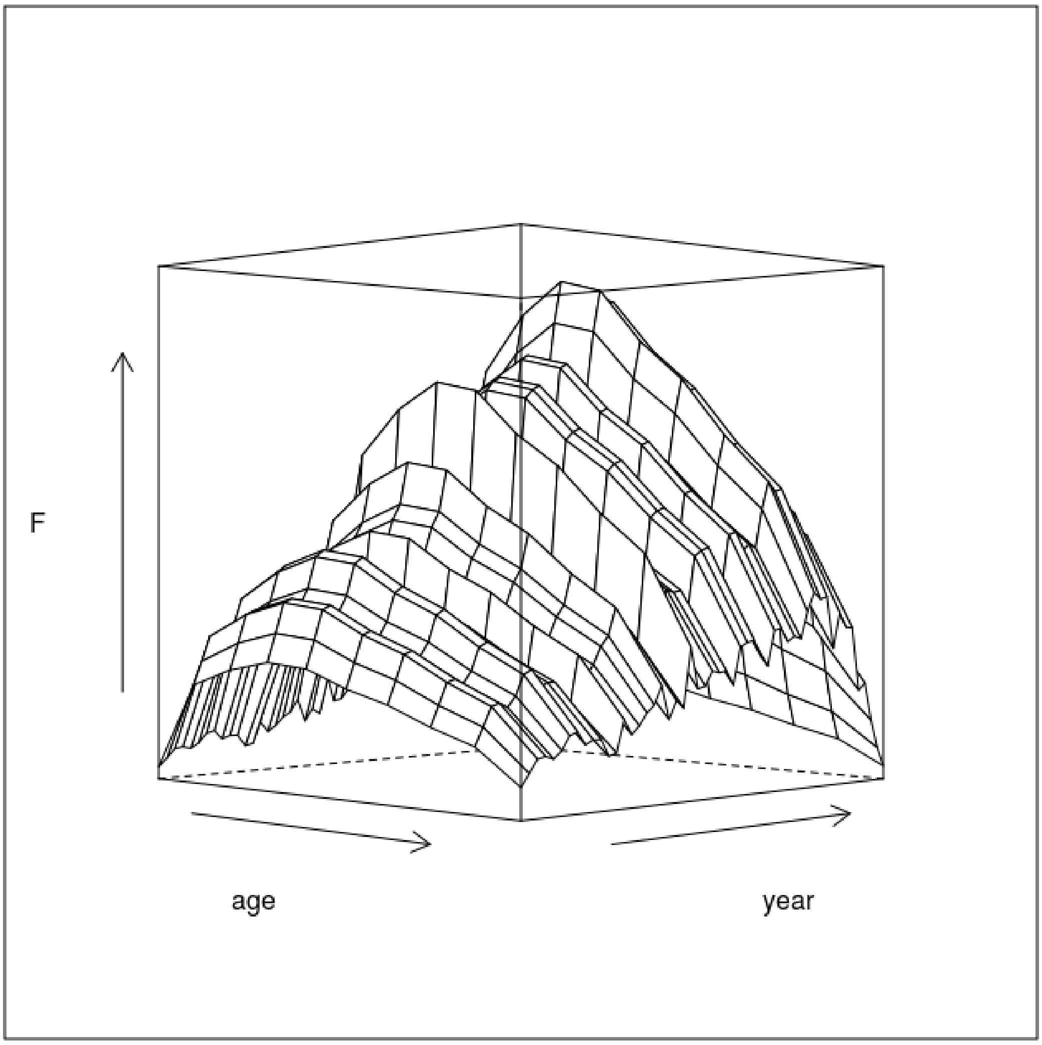


Figure 12: Fishing mortality separable model

Next we may make things a bit more interesting by using an (unpenalised) thin plate spline, where we'll borrow the smoothing splines method (`s()`) provided by package `mgcv`. We're using the North Sea Plaice data again, 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. Note that this is still a separable model, it's a smoothed version of the previous model (Figure 13).

```
fmod <- ~s(age, k = 4) + s(year, k = 20)
# notice that you can specify the submodels without the argument, as an
# example you don't need fmodel=fmod, but the order should be respected...
```

```
fit1 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
```

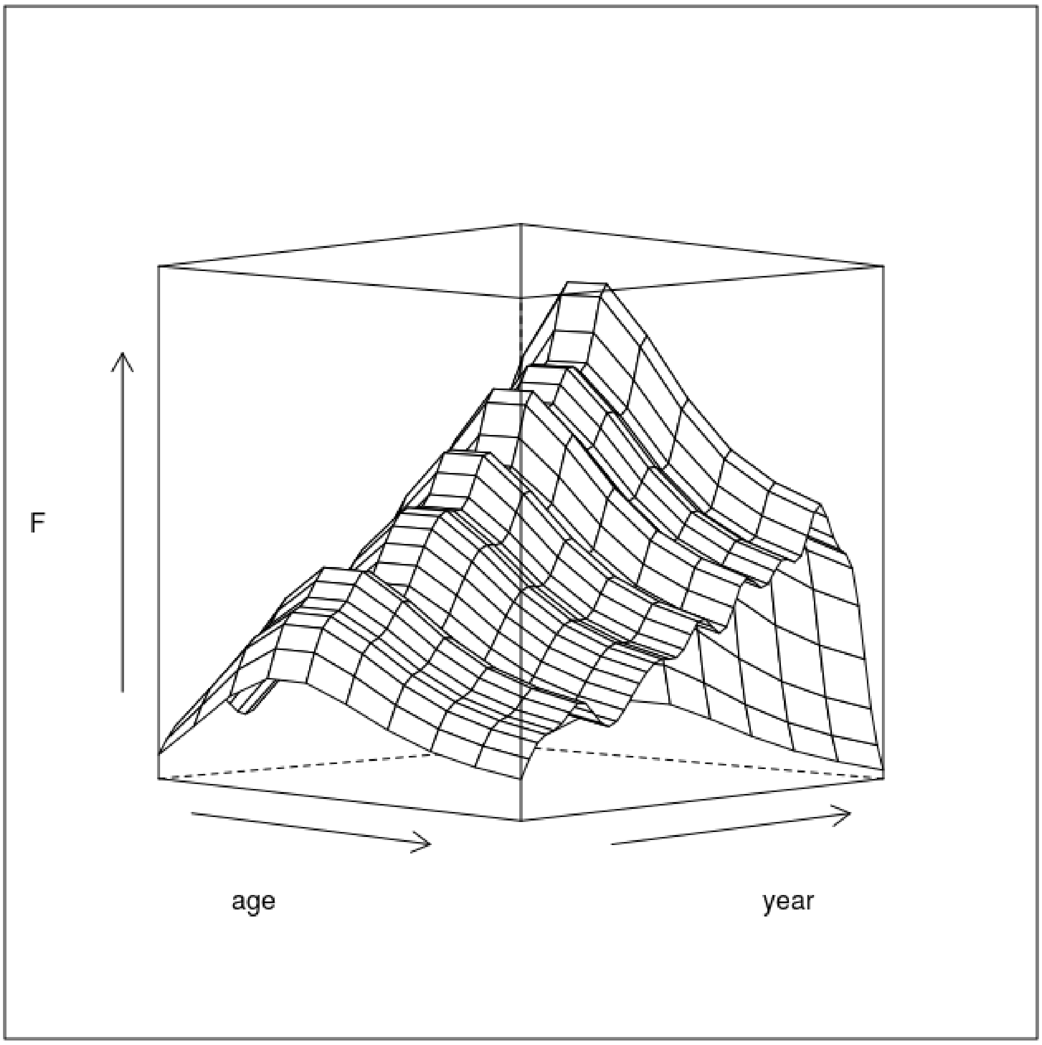


Figure 13: Fishing mortality smoothed separable model

A non-separable model, where we consider age and year to interact can be modeled using a smooth interaction term in the F model using a tensor product of cubic splines with the `te` method (Figure 14), again borrowed from `mgcv`.

```
fmod <- ~te(age, year, k = c(4, 20))
fit3 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
```

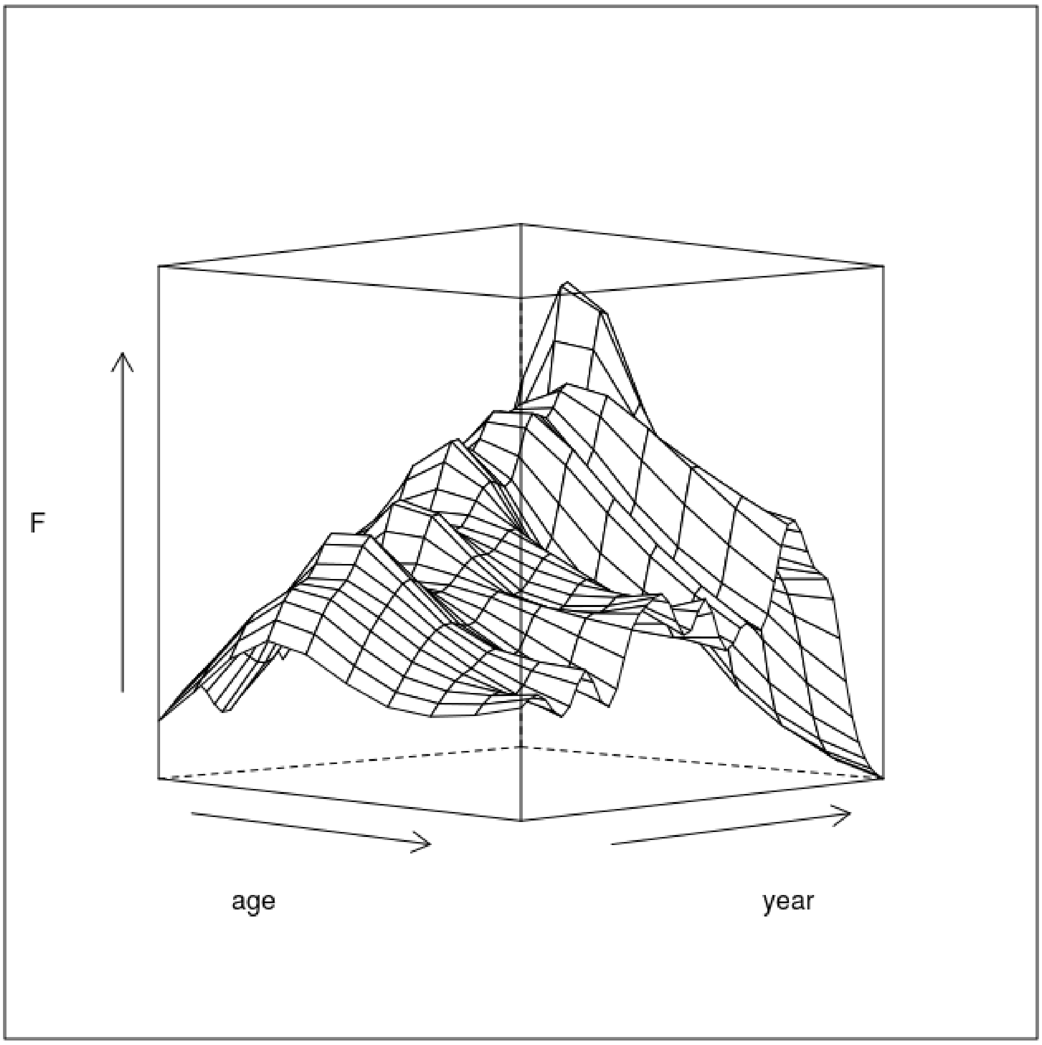


Figure 14: Fishing mortality smoothed non-separable model

In the last examples the fishing mortalities (Fs') are linked across age and time. What if we want to free up a specific age class because in the residuals we see a consistent pattern. This can happen, for example, if the spatial distribution of juveniles is disconnected to the distribution of adults. The fishery focuses on the adult fish, and therefore the the F on young fish is a function of the distribution of the juveniles and could deserve a specific model. This can be achieved by adding a component for the year effect on age 1 (Figure 15).

```
fmod <- ~te(age, year, k = c(4, 20)) + s(year, k = 5, by = as.numeric(age ==
    1))
fit4 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
```

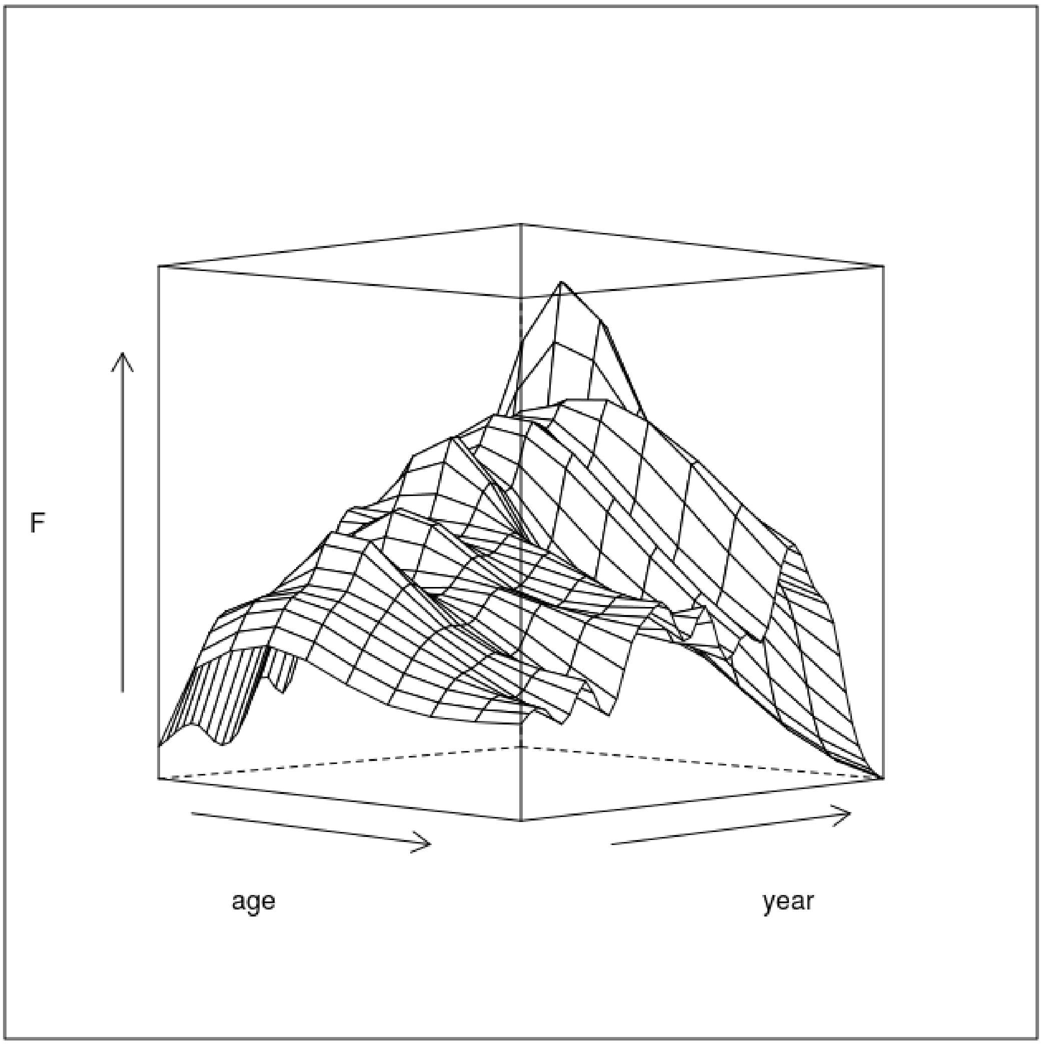


Figure 15: Fishing mortality age-year interaction model with extra age 1 smoother.

6.2 Catchability submodel

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.

We'll start by fixing the F and R models and compute the fraction of the year the index relates to, which will allow us to compute catchability at age and year.

```
fmod <- ~factor(age) + factor(year)
srmod <- ~factor(year)
```

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 16).

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

To compute the catchability estimated for each index we'll need to compute the abundance at the moment the index was carried out and divide the predicted index by the abundance. More precisely we'll compute abundance in the mid of the index period, which is stored in the `FLIndex` object, in the slot `range`, in fractions of the year. Later we'll see that we can use the method `predict()` to get the same result, but we'll need a *a4aFitSA* object to get the fitted parameters.

```
# compute N for the fraction of the year the survey is carried out
sfrac <- mean(range(ple4.indices[[1]])[c("startf", "endf")])
# fraction of total mortality up to that moment
Z <- (m(ple4) + harvest(fit)) * sfrac
lst <- dimnames(fit@index[[1]])
# survivors
lst$x <- stock.n(fit) * exp(-Z)
stkn <- do.call("trim", lst)
qhat <- index(fit)[[1]]/stkn
```

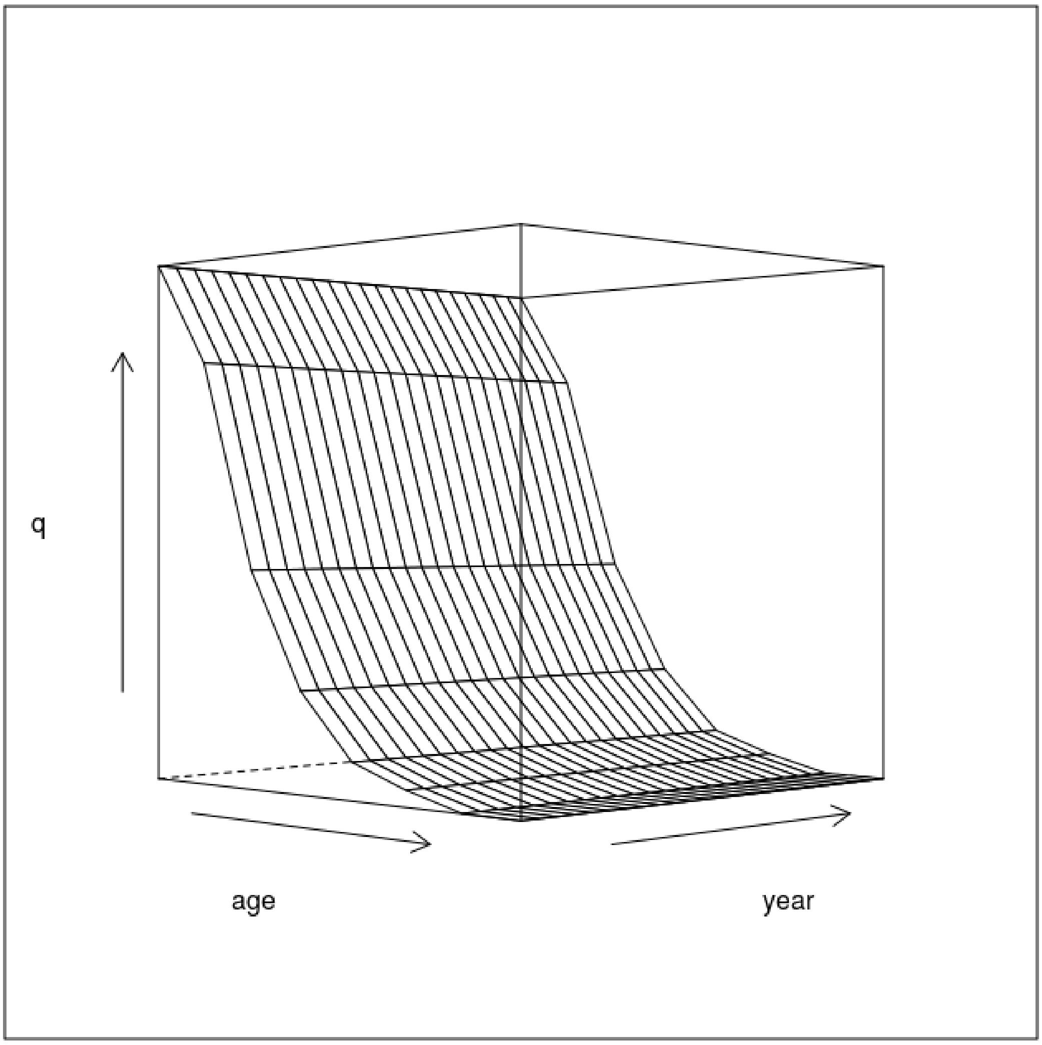


Figure 16: Catchability age independent model

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 17).

```
qmod <- list(~s(age, k = 4))
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

# compute N for the fraction of the year the survey is carried out
Z <- (m(ple4) + harvest(fit)) * sfrac
lst <- dimnames(fit@index[[1]])
lst$x <- stock.n(fit) * exp(-Z)
```

```
stkn <- do.call("trim", lst)
qhat <- index(fit)[[1]]/stkn
```

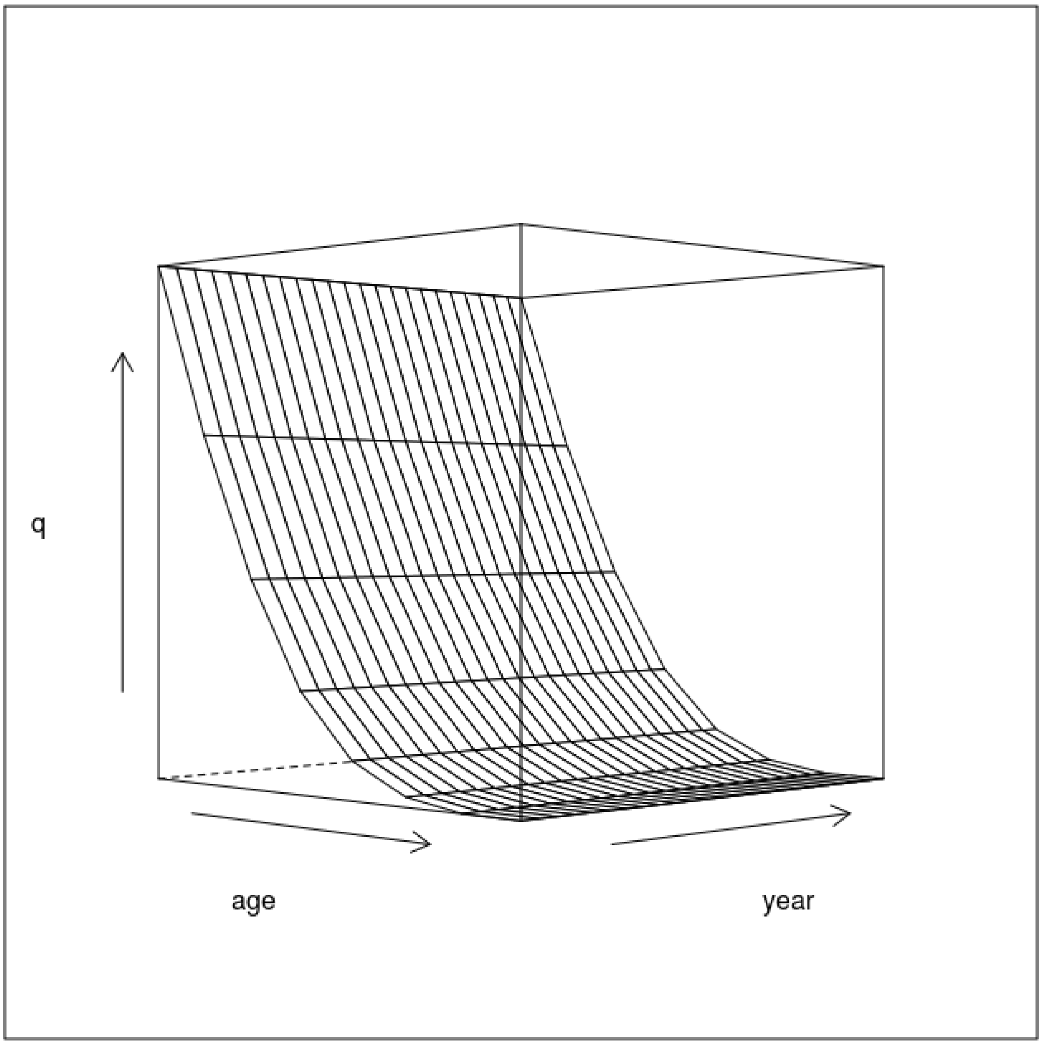


Figure 17: Catchability smoother age model

As in the case of F, one may consider catchability to be a process that evolves with age and year, including an interaction between the two effects. Such model can be modelled using the tensor product of cubic splines, the same way we did for the F model (Figure 18).

```
qmod <- list(~te(age, year, k = c(3, 40)))
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
```

```
# compute N for the fraction of the year the survey is carried out
Z <- (m(ple4) + harvest(fit)) * sfrac
lst <- dimnames(fit@index[[1]])
lst$x <- stock.n(fit) * exp(-Z)
stkn <- do.call("trim", lst)
qhat <- index(fit)[[1]]/stkn
```

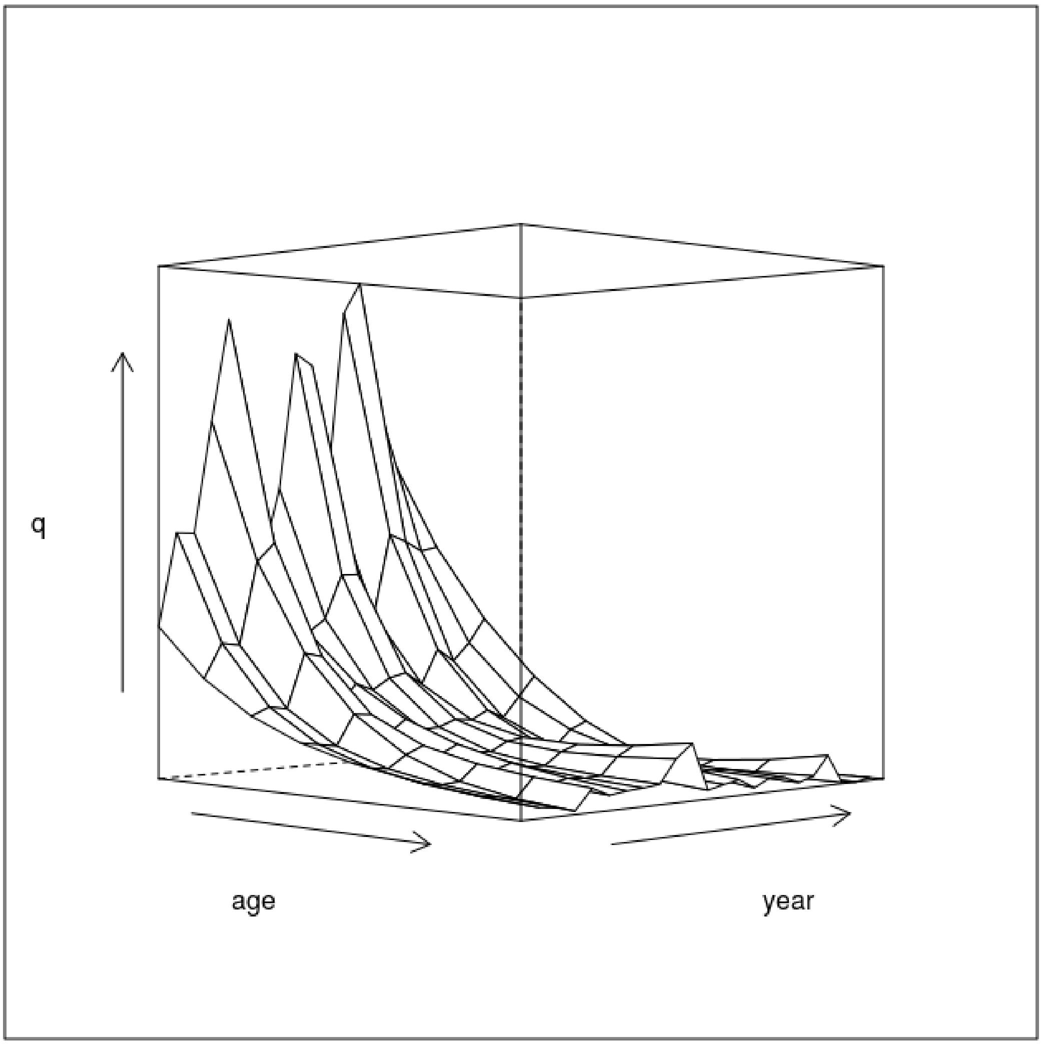


Figure 18: Catchability tensor product of age and year

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 19).

```r
qmod <- list(~s(age, k = 4) + year)
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)

# compute N for the fraction of the year the survey is carried out
Z <- (m(ple4) + harvest(fit)) * sfrac
lst <- dimnames(fit@index[[1]])
lst$x <- stock.n(fit) * exp(-Z)
stkn <- do.call("trim", lst)
qhat <- index(fit)[[1]]/stkn
```

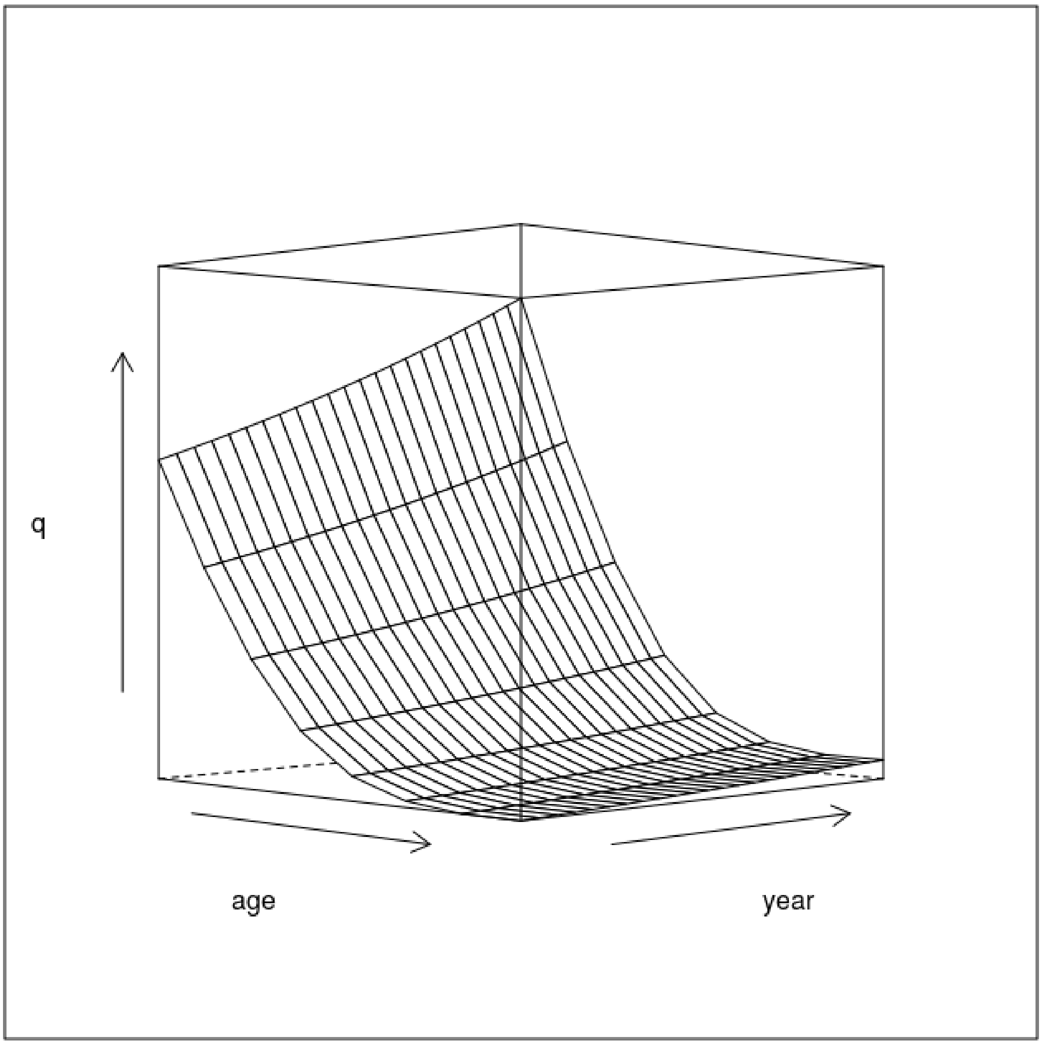


Figure 19: Catchability with a linear trend in year

6.3 Catchability submodel for age aggregated 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 slightly 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.

```
# creating an index (note the name of the first dimension element)
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 364.25 412.67 398.07 399.36 424.01 498.44 416.62 474.29 456.73
##      year
## age   1966   1967   1968   1969   1970   1971   1972   1973   1974
##   all 553.71 496.73 523.47 555.51 372.53 375.92 442.61 396.08 382.04
##      year
## age   1975   1976   1977   1978   1979   1980   1981   1982   1983
##   all 509.44 409.46 496.04 429.51 468.42 429.47 419.90 427.89 487.13
##      year
## age   1984   1985   1986   1987   1988   1989   1990   1991   1992
##   all 525.48 543.94 567.35 738.41 706.19 611.17 576.26 552.09 348.65
##      year
## age   1993   1994   1995   1996   1997   1998   1999   2000   2001
##   all 308.51 248.76 266.53 301.05 482.70 298.98 272.76 295.21 370.06
##      year
## age   2002   2003   2004   2005   2006   2007   2008
##   all 310.51 309.32 291.84 253.44 353.25 258.03 313.75
##
## units:  t

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

The same methods that are applied to age disaggregated indices apply here, see standardized log residuals in Figure 20. It's also possible to mix several indices of both types.

```
plot(residuals(fit, ple4, FLIndices(bioidx)))
```

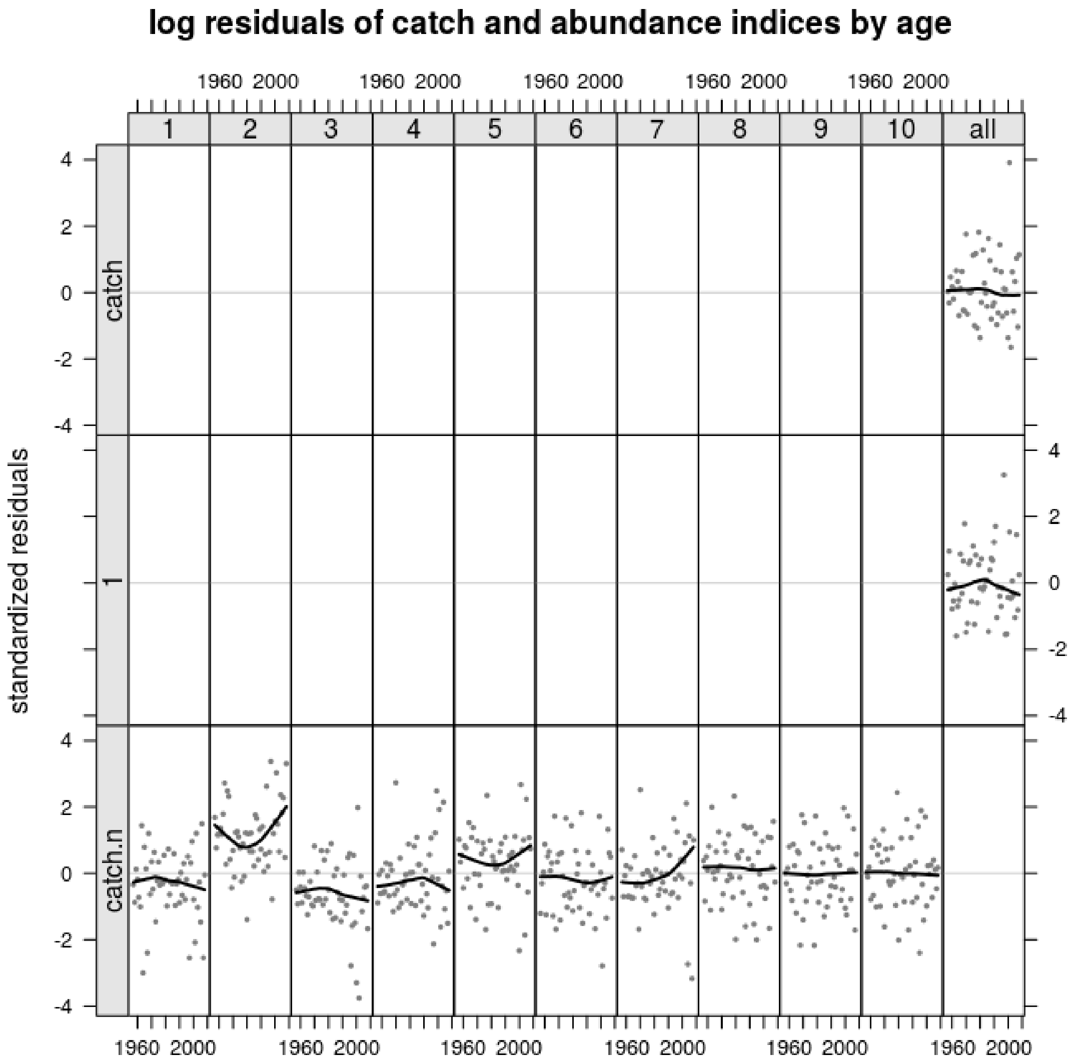


Figure 20: Catchability residuals for a biomass index

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))
```

And once more residuals can be a good dignostics (Figure 21).

```
plot(residuals(fit, ple4, FLIndices(bioidx)))
```

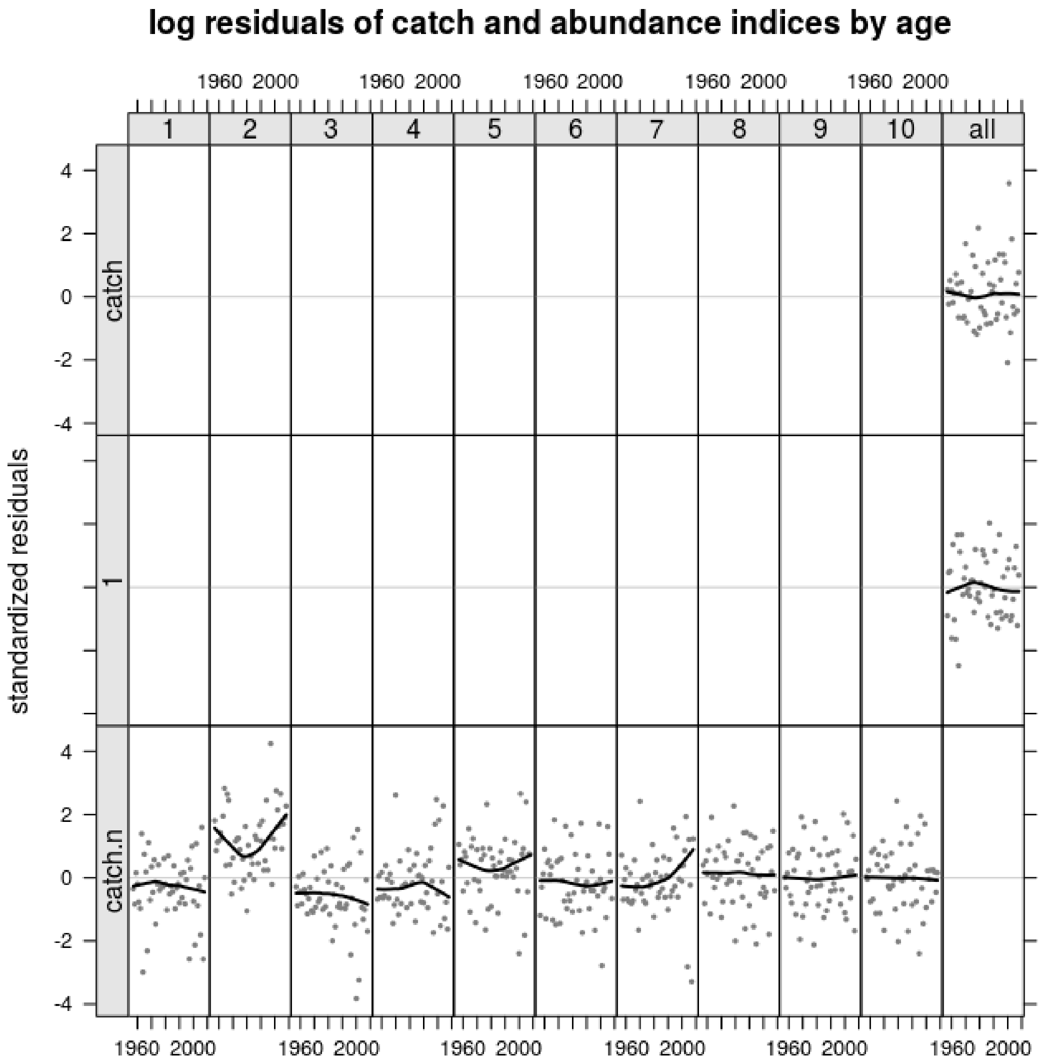


Figure 21: Catchability residuals for a biomass index

6.4 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.

```r
fit <- sca(ple4, FLIndices(ple4.index[1]), qmodel = list(~1))
```

As previously, the same methods apply, see standardized log residuals in Figure 22.

```r
plot(residuals(fit, ple4, FLIndices(ple4.index[1])))
```

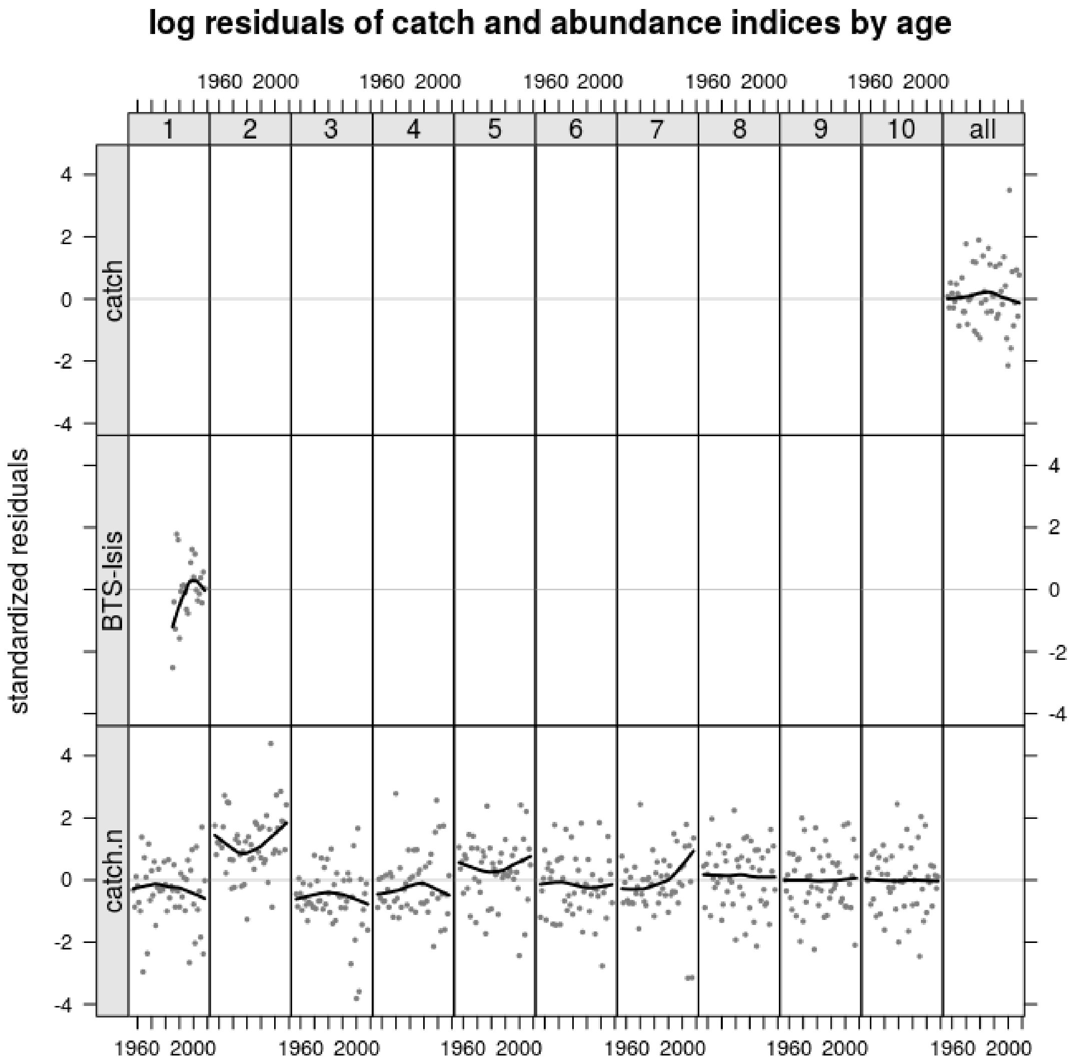


Figure 22: Catchability residuals for a single age index

6.5 Stock-recruitment submodel

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 23 for results. As mentioned before, the 'structural' models have a fixed variance, which must be set by defining the coefficient of variation. We now fix the F and Q submodels before fiddling around with the S/R model.

```
fmod <- ~s(age, k = 4) + s(year, k = 20)
qmod <- list(~s(age, k = 4))
```

```
srmod <- ~factor(year)
fit <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
srmod <- ~s(year, k = 20)
fit1 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
srmod <- ~ricker(CV = 0.1)
fit2 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
srmod <- ~bevholt(CV = 0.1)
fit3 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
srmod <- ~hockey(CV = 0.2)
fit4 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
srmod <- ~geomean(CV = 0.1)
fit5 <- sca(ple4, ple4.indices[1], fmod, qmod, srmod)
flqs <- FLQuants(factor = stock.n(fit)[1], smother = stock.n(fit1)[1], ricker = stock.n(fit2)[1],
    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")
```

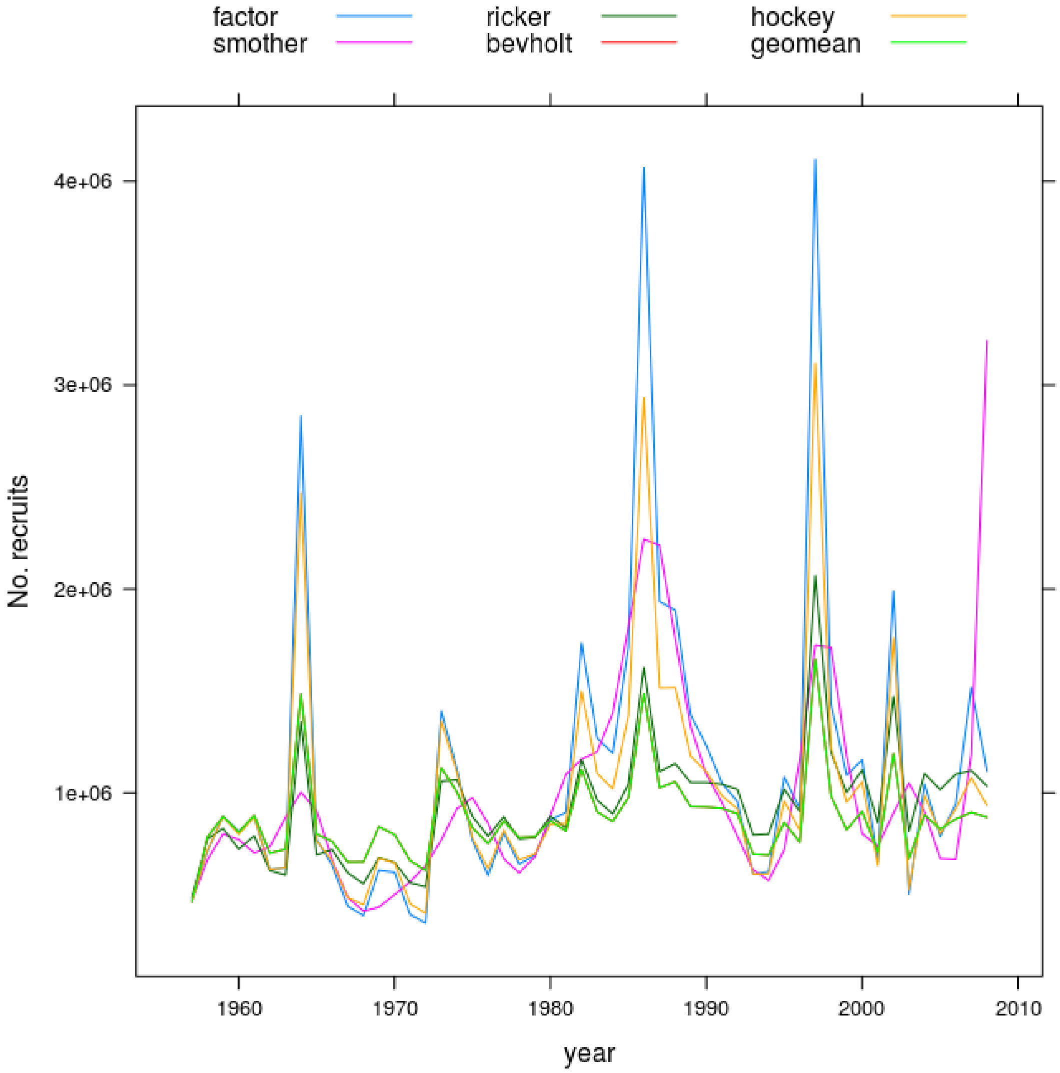


Figure 23: Stock-recruitment models fits

7 The major effects - age, year and cohort

All submodels use the same type of specification process, the R formula interface, wich gives lot's of flexibility to explore models and combination of sub-models. As a reference one can consider three major effects that can be modelled the same way, the age affect, year effect and cohort effect. As examples note the following models, in these cases applied to fishing mortality, and all of them as a factor, which means one coefficient will be estimated for each level of the factor, meaning age, year or cohort repectively.

```
# 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 Figure **??**

```
wireframe(data ~ year * age, data = harvest(fit1), main = "year effect")
```

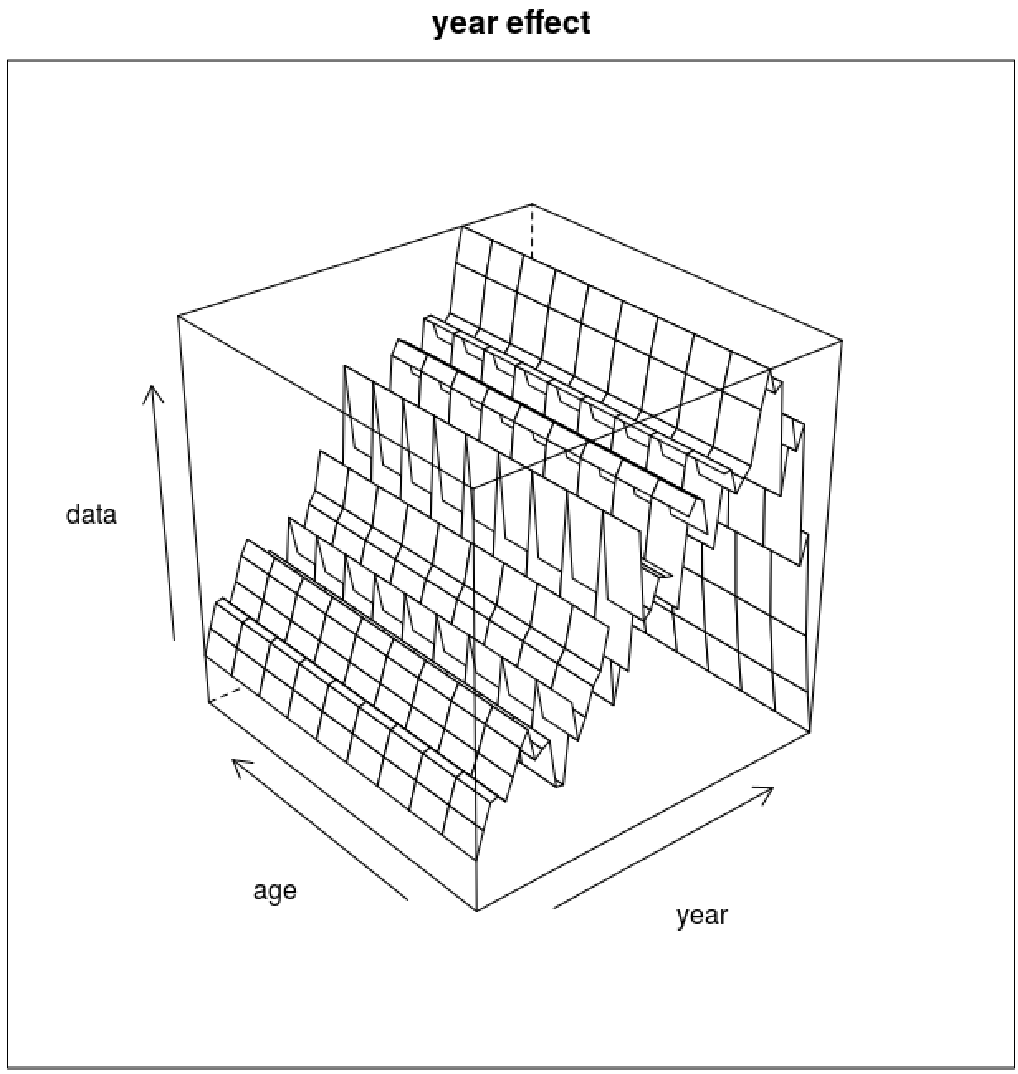


Figure 24: Examples of fishing mortality models for age, year and cohort.

```
wireframe(data ~ year * age, data = harvest(fit2), main = "age effect")
```

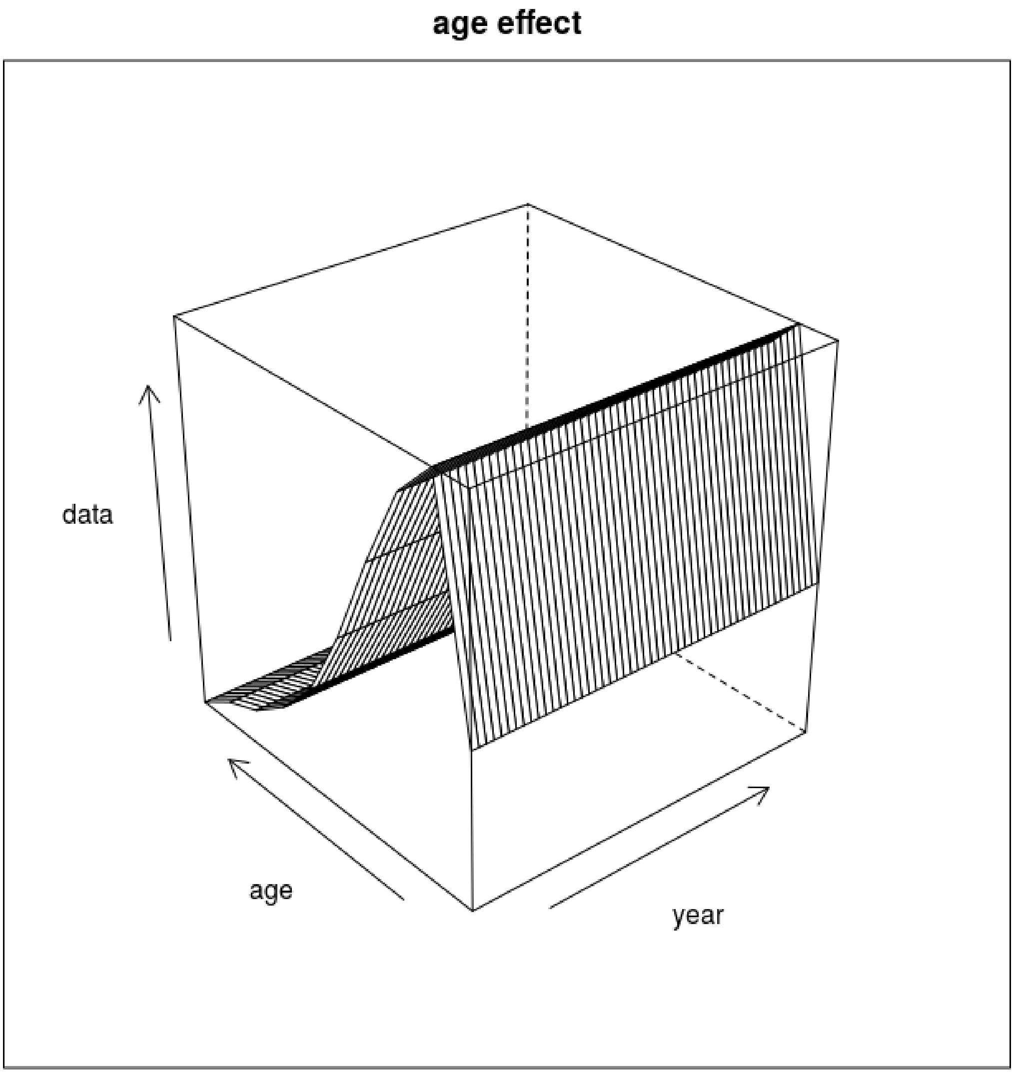


Figure 25: Examples of fishing mortality models for age, year and cohort.

```
wireframe(data ~ year * age, data = harvest(fit3), main = "cohort effect")
```

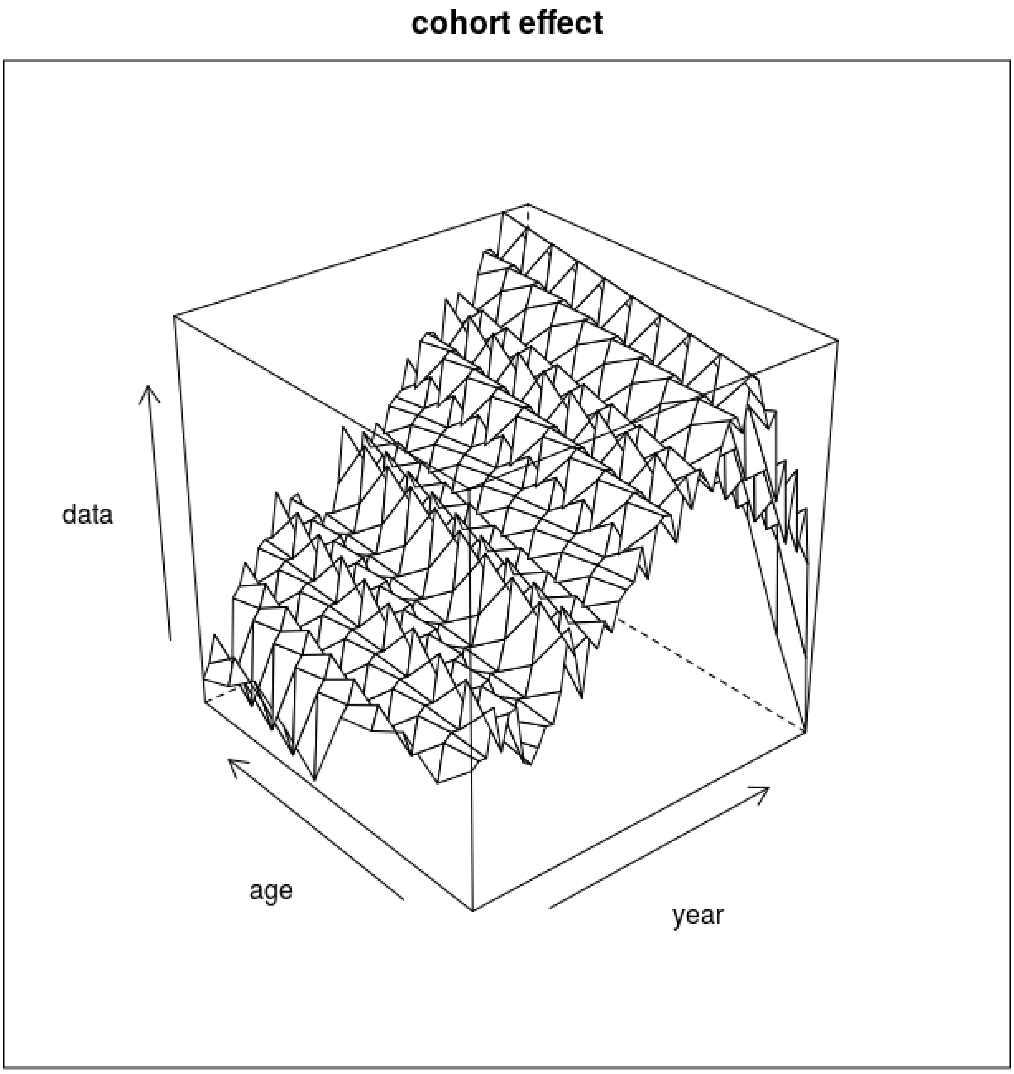


Figure 26: Examples of fishing mortality models for age, year and cohort.

8 The statistical catch-at-age stock assessment framework advanced features - the `a4aSCA` method

A more advanced method for stock assessment can be used through the `a4aSCA()` method. This method gives access to the submodels for $N1$, σ^2_{ay} and I_{ays} as well as arguments to get the ADMB files, etc. Check the manual pages with `?a4aSCA` for more information. This method has 'assessment' as the default value for the `fit` argument, which means that the hessian is going to be computed and all the information about the parameters will be returned by default. Note that the default models of each submodel can be accessed with

```
fit <- a4aSCA(ple4, ple4.indices[1])
submodels(fit)

##   fmodel: ~s(age, k = 3) + factor(year)
##  srmodel: ~factor(year)
##  n1model: ~factor(age)
##   qmodel:
##     BTS-Isis: ~1
##   vmodel:
##     catch:    ~s(age, k = 3)
##     BTS-Isis: ~1
```

8.1 N1 model

The submodel for the stock number at age in the first year of the time series is set up with the usual linear tools (Figure 27), but bare in mind that the year effect does not make sense here.

```
n1mod <- ~s(age, k = 4)
fit1 <- a4aSCA(ple4, ple4.indices[1], n1model = n1mod)
flqs <- FLQuants(smo = stock.n(fit1)[, 1], fac = stock.n(fit)[, 1])
```

```
xyplot(data ~ age, groups = qname, data = flqs, type = "l", auto.key = list(points = FALSE,
    lines = TRUE, columns = 2))
```

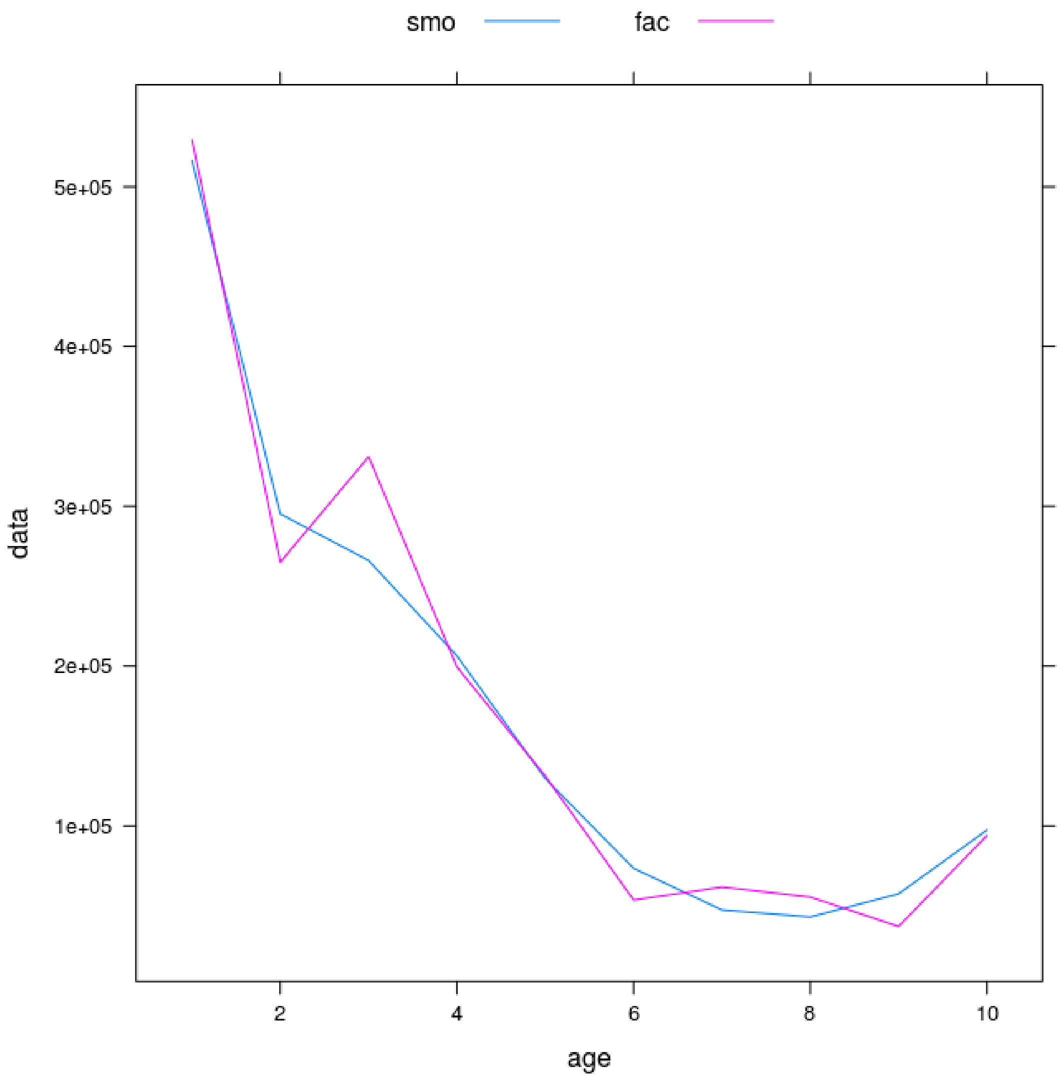


Figure 27: Nay=1 models

8.2 Variance model

The variance model allows the user to set up the shape of the observation variances σ^2_{ay} and I_{ays}. This is an important subject related with fisheries data used for input to stock assessment models. It's quite common to have more precision on the most represented ages and less precision on the less frequent ages. This is due to the fact that the last ages do not appear as often at the auction markets, in the fishing operations or on survey samples.

By default the model assumes constant variance over time and ages (1 model) but it can use other models specified by the user. As with the other submodels, R linear model capabilities are used (Figure 28).

```
vmod <- list(~1, ~1)
fit1 <- a4aSCA(ple4, ple4.indices[1], vmodel = vmod)
vmod <- list(~s(age, k = 4), ~1)
fit2 <- a4aSCA(ple4, ple4.indices[1], vmodel = vmod)
flqs <- FLQuants(cts = catch.n(fit1), smo = catch.n(fit2))
```

```
xyplot(data ~ year | age, groups = qname, data = flqs, type = "l", scales = list(y = list(relatio
    auto.key = list(points = FALSE, lines = TRUE, columns = 2))
```

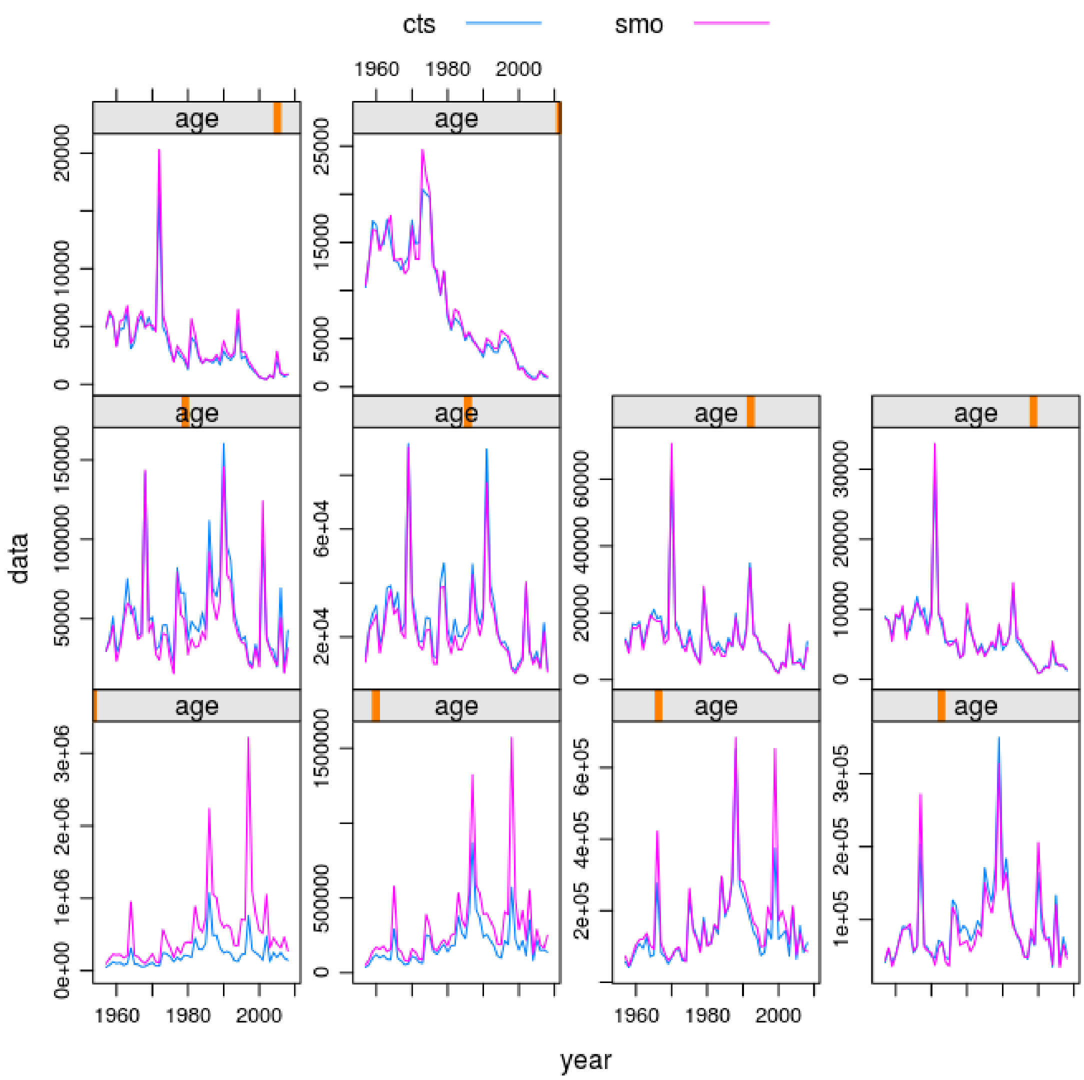


Figure 28: Population estimates using two different variance models

8.3 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 `a4a` framework. The example below uses the North Atlantic Oscillation (NAO) index to model recruitment.

```
nao <- read.table("http://www.cdc.noaa.gov/data/correlation/nao.data", skip = 1,
    nrow = 62, na.strings = "-99.90")
dnms <- list(quant = "nao", year = 1948:2009, 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))
nao <- as.numeric(nao)
```

First by simply assuming that the index drives recruitment (Figure 29).

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

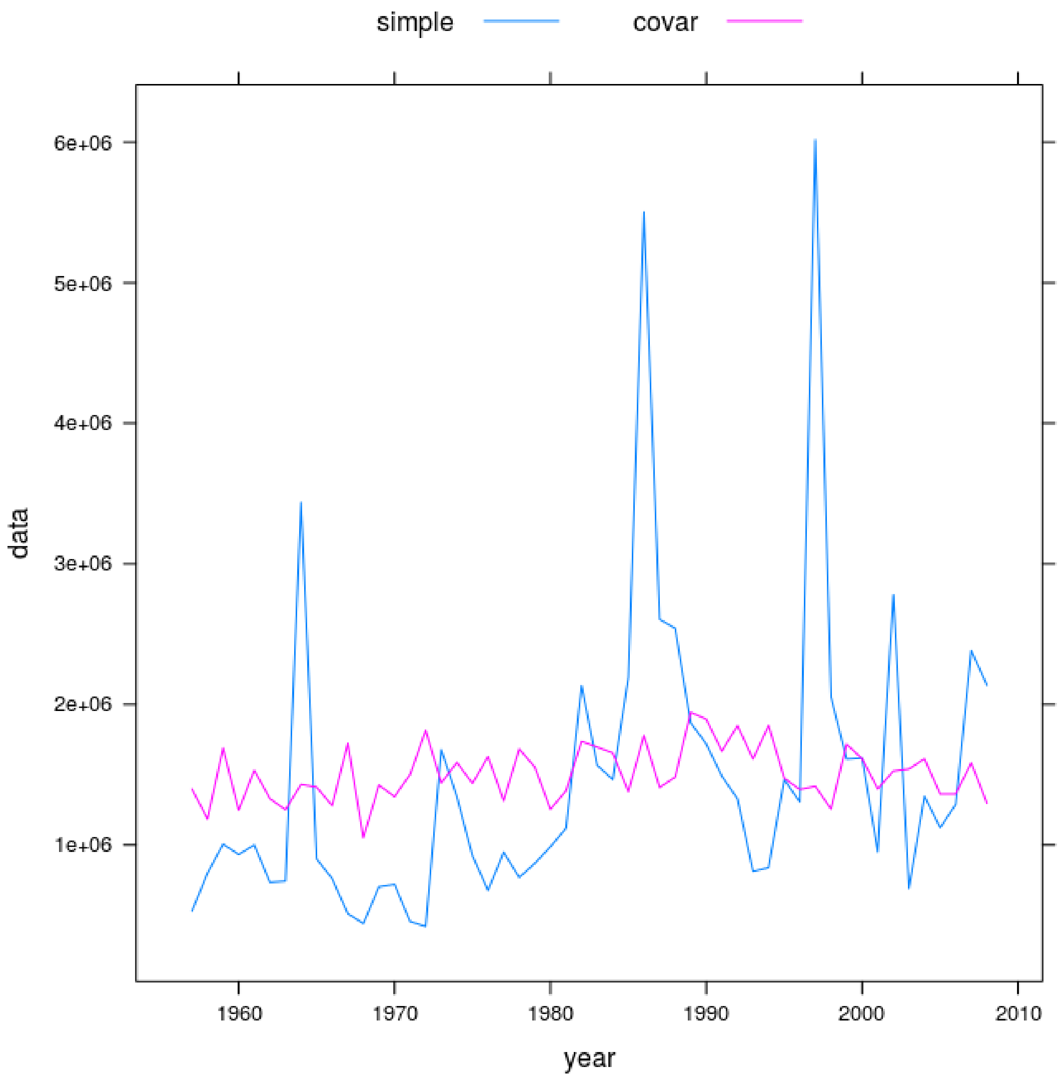


Figure 29: Recruitment model with covariates

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 (Figure 30).

```
srmod <- ~ricker(a = ~nao, CV = 0.1)
fit3 <- sca(ple4, ple4.indices[1], qmodel = list(~s(age, k = 4)), srmodel = srmod)
flqs <- FLQuants(simple = stock.n(fit)[1], covar = stock.n(fit3)[1])
```

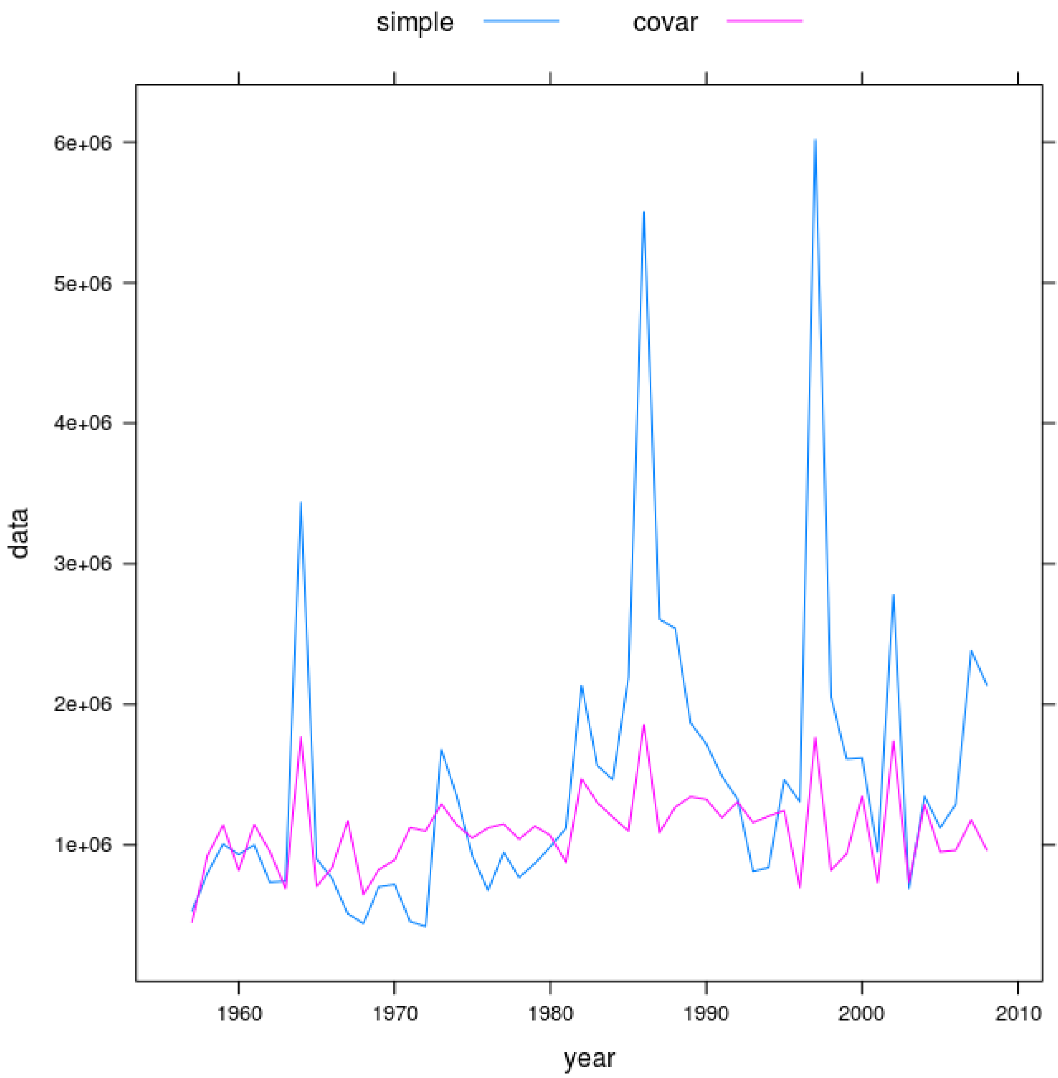


Figure 30: Recruitment model with covariates

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

8.4 Assessing `ADMB` files

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

```
fit1 <- a4aSCA(ple4, ple4.indices, wkdir = "mytest")

## Model and results are stored in working directory [mytest]
```

9 Predict and simulate

To predict and simulate R uses the methods `predict()` and `simulate()`, which were implemented in `FLa4a` in the same fashion.

```
fit <- sca(ple4, ple4.indices[1], fit = "assessment")
```

9.1 Predict

Predict simply computes the quantities of interest using the estimated coefficients and the design matrix of the model.

```
fit.pred <- predict(fit)
lapply(fit.pred, names)

## $stkmodel
## [1] "harvest" "rec"     "ny1"
##
## $qmodel
## [1] "BTS-Isis"
##
## $vmodel
## [1] "catch"    "BTS-Isis"
```

9.2 Simulate

Simulate uses the variance-covariance matrix computed from the Hessian returned by `ADMB` and the fitted parameters, to parametrize a multivariate normal distribution. The simulations are carried out using the method `mvrnorm()` provided by the R package `MASS`. Figure 31 shows a comparison between the estimated values and the medians of the simulation, while Figure 32 presents the stock summary of the simulated and fitted data.

```
fits <- simulate(fit, 100)
flqs <- FLQuants(sim = iterMedians(stock.n(fits)), det = stock.n(fit))
```

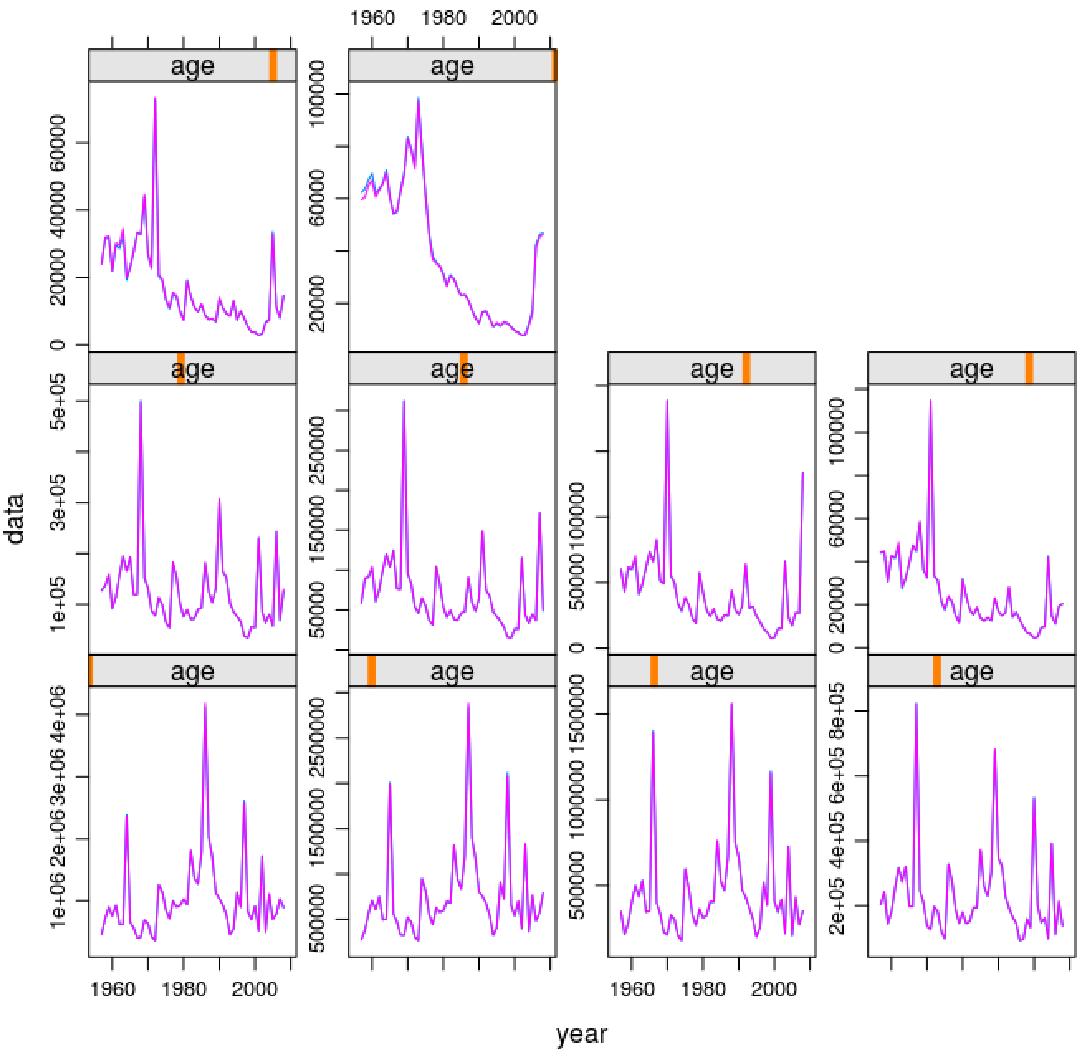


Figure 31: Median simulations VS fit

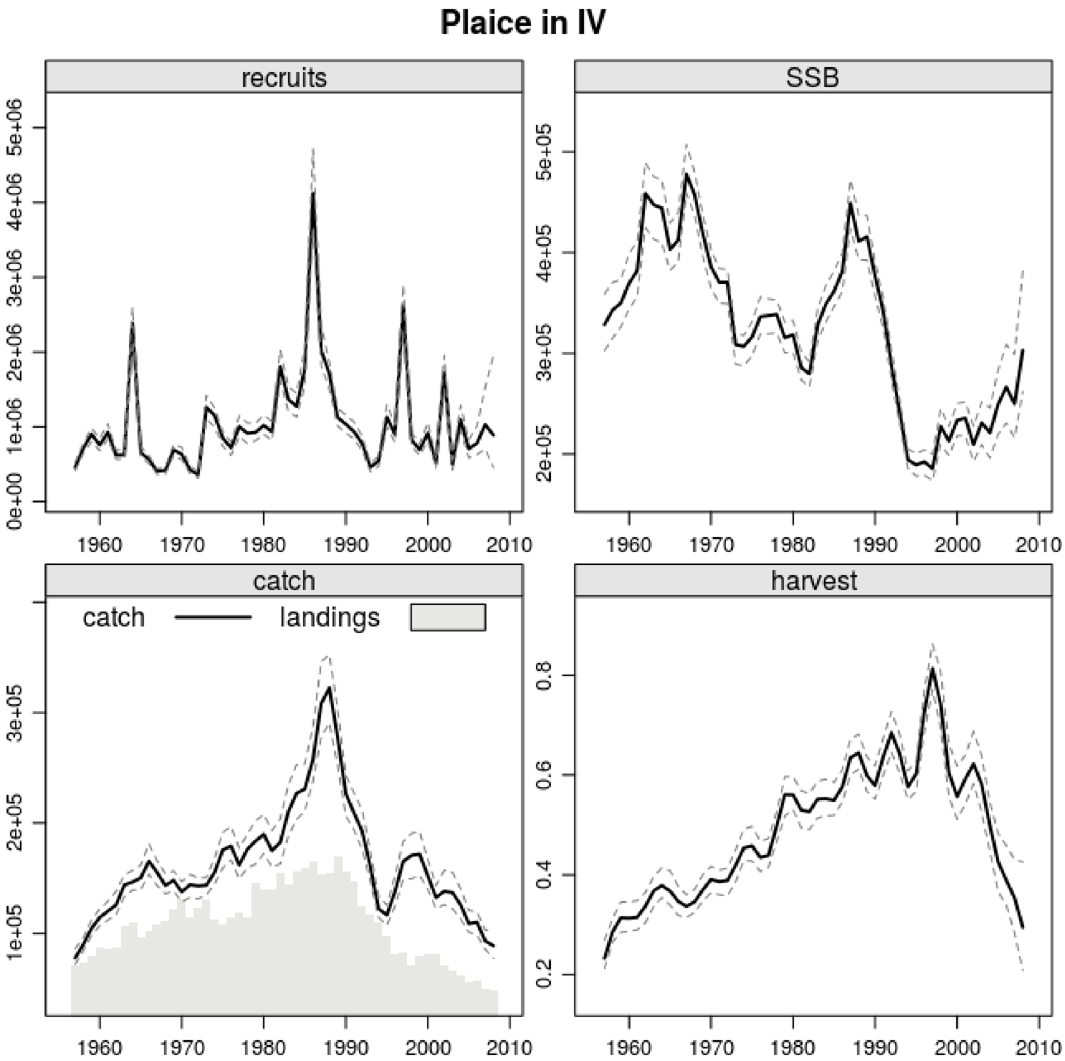


Figure 32: Stock summary of the simulated and fitted data

10 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` and `a4aSCA` methods interface with ADMB to use the MCMC fits.

The likelihood estimate is

```
# ll
fit <- a4aSCA(ple4, ple4.indices)
fit <- simulate(fit, 1000)
```

To run the MCMC estimate, 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.

```
# 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` ius 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.

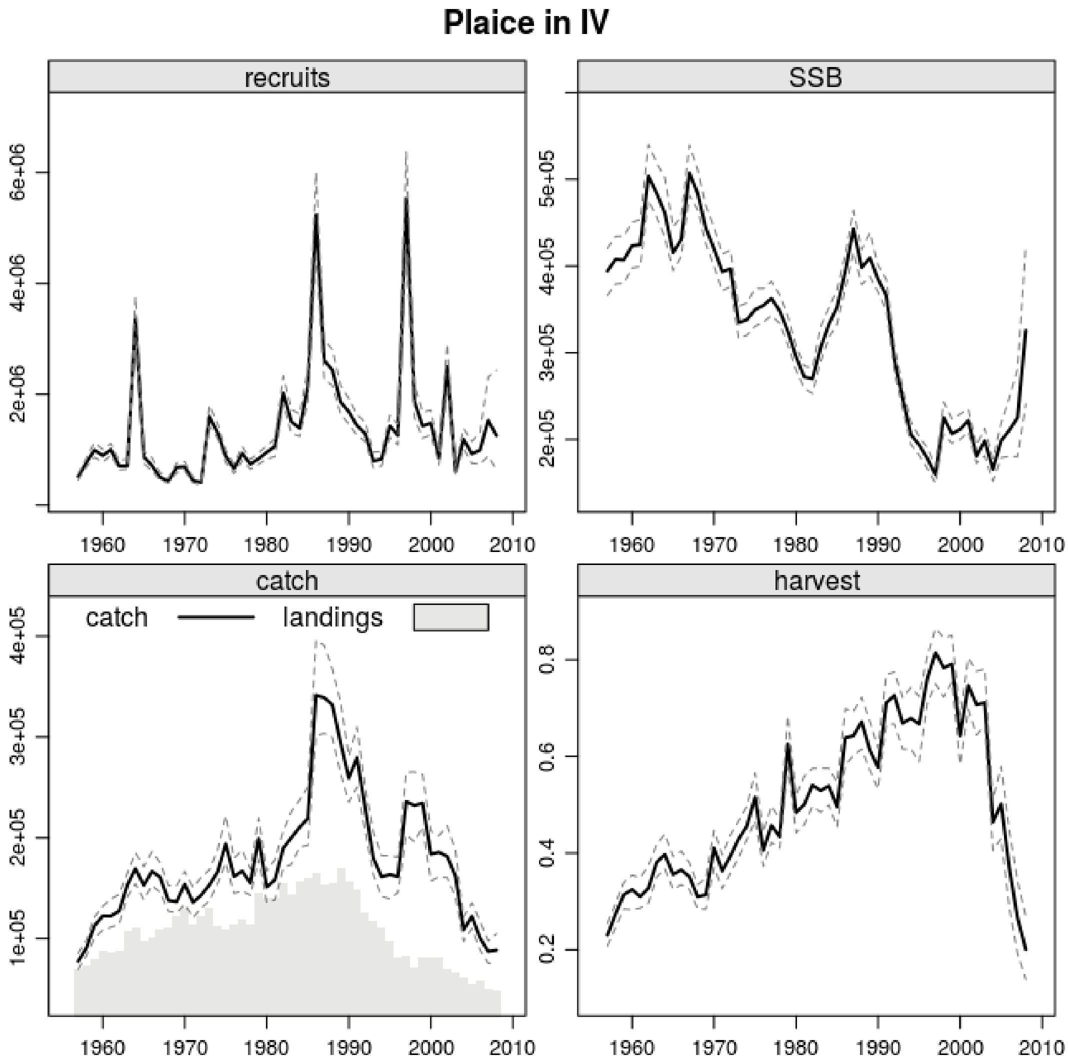
```
# fit the model
fitmc1 <- a4aSCA(ple4, ple4.indices, fit = "MCMC", mcmc = mc)
# check acceptance rate
fitSumm(fitmc1)

##              iters
##                  1
##   nopar   124.0000
##   nlogl         NA
##   maxgrad       NA
##   nobs    901.0000
##   gcv           NA
##   convergence   NA
##   accrate   0.3065
```

We use the package `CODA` to run the diagnostics on MCMC fits. First one can plot chains using coda. Note the mcmc object is a matrix with the parameters (row = iters, cols= pars).

```
fitmc1.mc <- as.mcmc(fitmc1)
plot(fitmc1.mc)
```

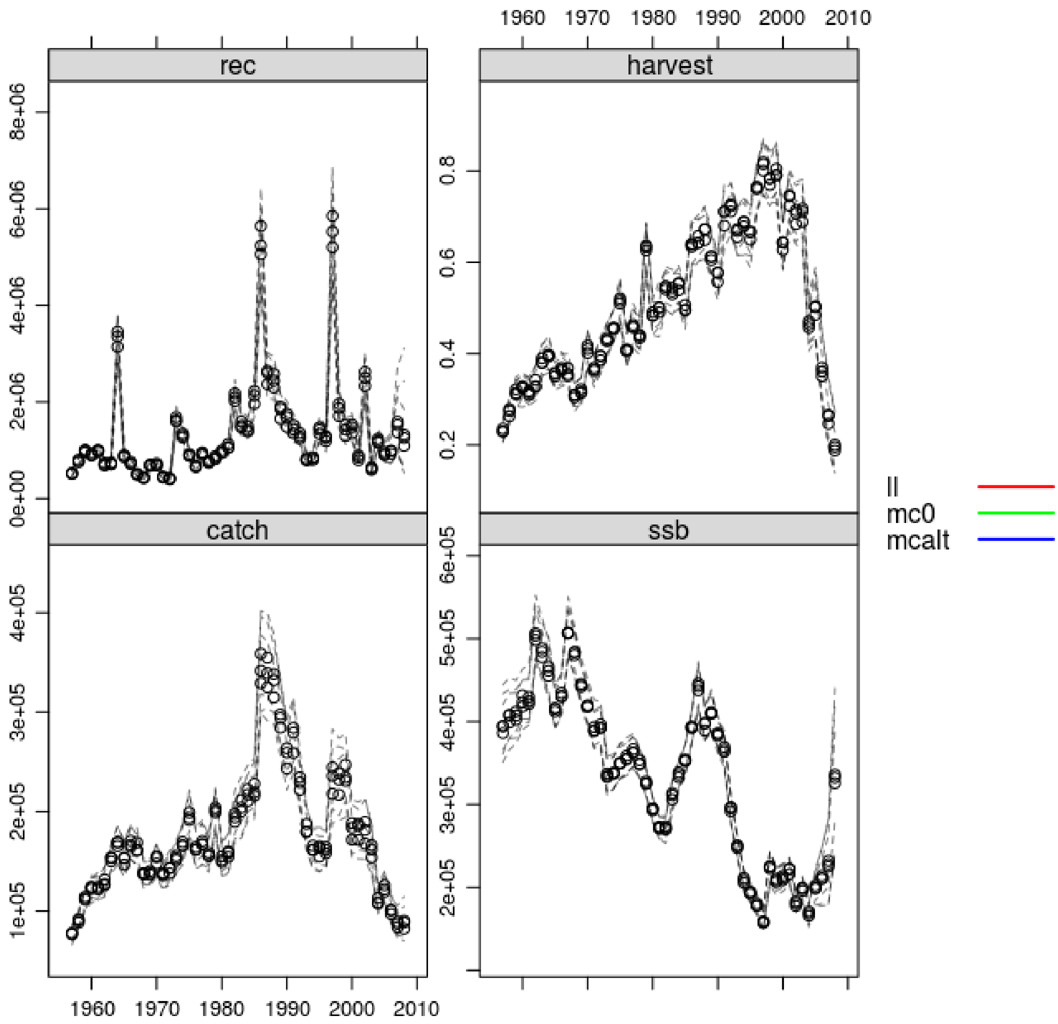
The usual summary plot,



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(mcmc = 10000, mcsave = 200, mcprobe = 0.4)
fitmc2 <- a4aSCA(ple4, ple4.indices, qmodel = list(~s(age, k = 3), ~s(age, k = 3),
    ~s(age, k = 3)), fit = "MCMC", mcmc = mc)
```

All fits together



11 Geeky stuff

A lot more can be done with the a4a framework. The next sections will describe methods that are more technical. What we'd categorize as 'matters for geeks', in the sense that these methods usually will require the users to 'dive' into R a bit more.

```
fit <- sca(ple4, ple4.indices[1], fit = "assessment")
```

11.1 External weigthing of likelihood components

By default the likelihood components are weighted using inverse variance. However, the user may change the weights by setting the variance of the input parameters. This is done by adding a variance matrix to the `catch.n` and `index.n` slots of the stock and index objects. These variances will be used to penalize the data during the likelihood computation. The values should be given as coefficients of variation on the log scale, so that variance is $\log{(CV^2 + 1)}$. Figure 33 shows the results of two fits with distinct likelihood weightings.

```
stk <- ple4
idx <- ple4.indices[1]
# variance of observed catches
varslt <- catch.n(stk)
varslt[] <- 0.4
catch.n(stk) <- FLQuantDistr(catch.n(stk), varslt)
# variance of observed indices
varslt <- index(idx[[1]])
varslt[] <- 0.1
index.var(idx[[1]]) <- varslt
# run
fit1 <- a4aSCA(stk, idx)
flqs <- FLQuants(nowgt = stock.n(fit), extwgt = stock.n(fit1))
```

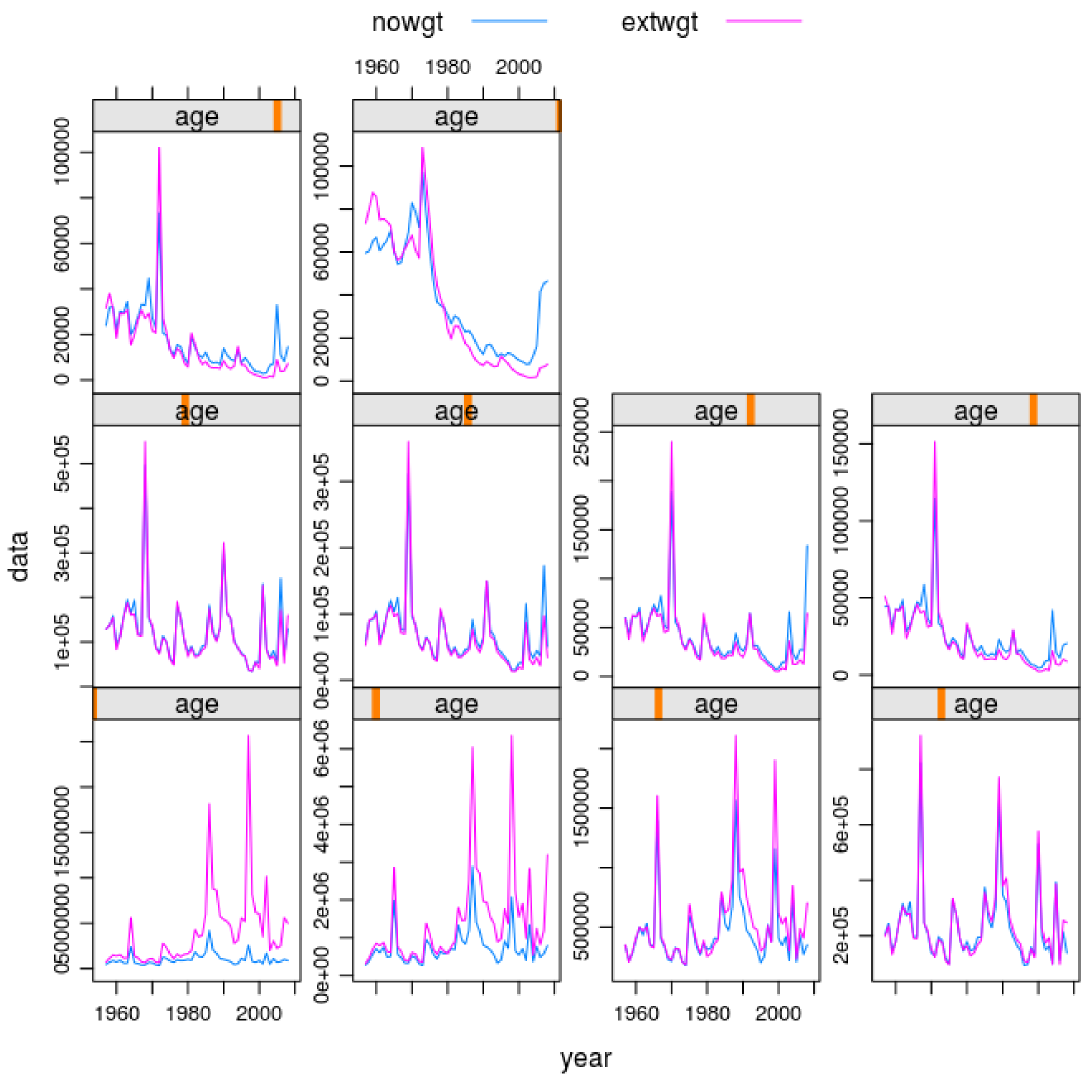


Figure 33: Stock summary of distinct likelihood weightings

11.2 More models

There's a set of methods that allow the user to have more flexibility on applying the models referred before. For example to break the time series in two periods, using the method `breakpts()`, or fixing some parts of the selection pattern by setting F to be the same for a group of ages, using `replace()`.

The example below (Figure 34) replaces all ages above 5 by age 5, which means that a single coefficient is going to be estimated for age 5-10.

```
fmod <- ~s(replace(age, age > 5, 5), k = 4) + s(year, k = 20)
fit <- sca(ple4, ple4.indices, fmod)
```

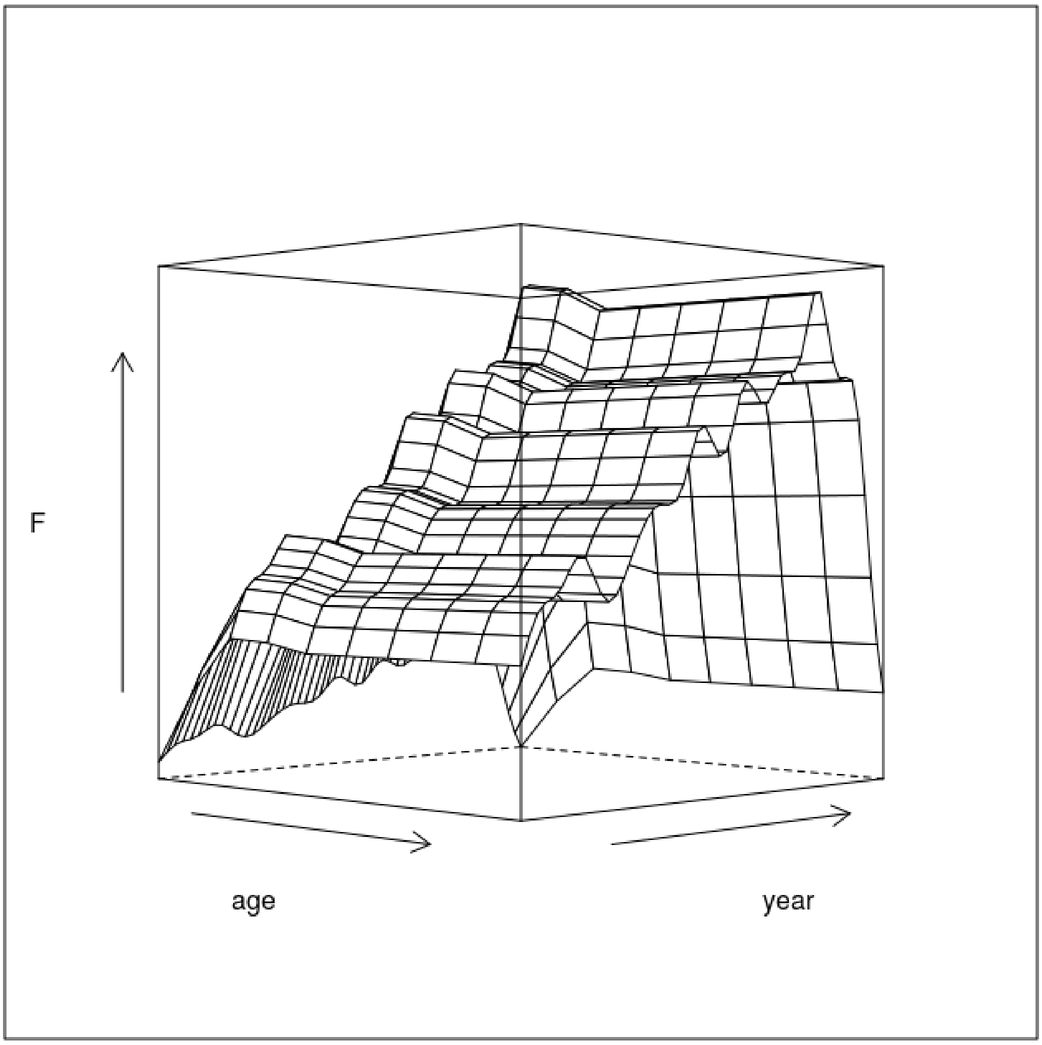


Figure 34: F-at-age fixed above age 5

Or else one can use a closed form fort the selection pattern. The example below uses a logistic form (Figure 35).

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

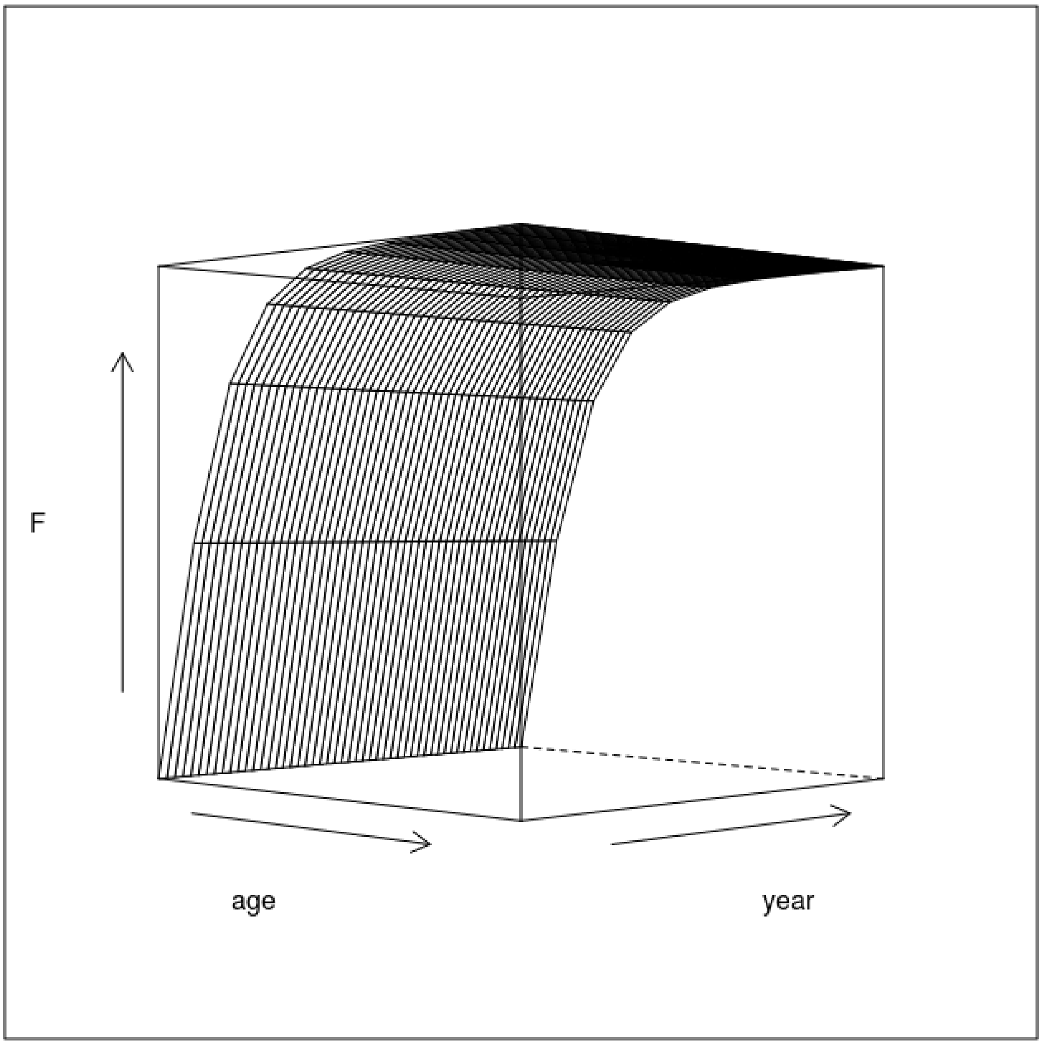


Figure 35: F-at-age logistic

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 36).

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

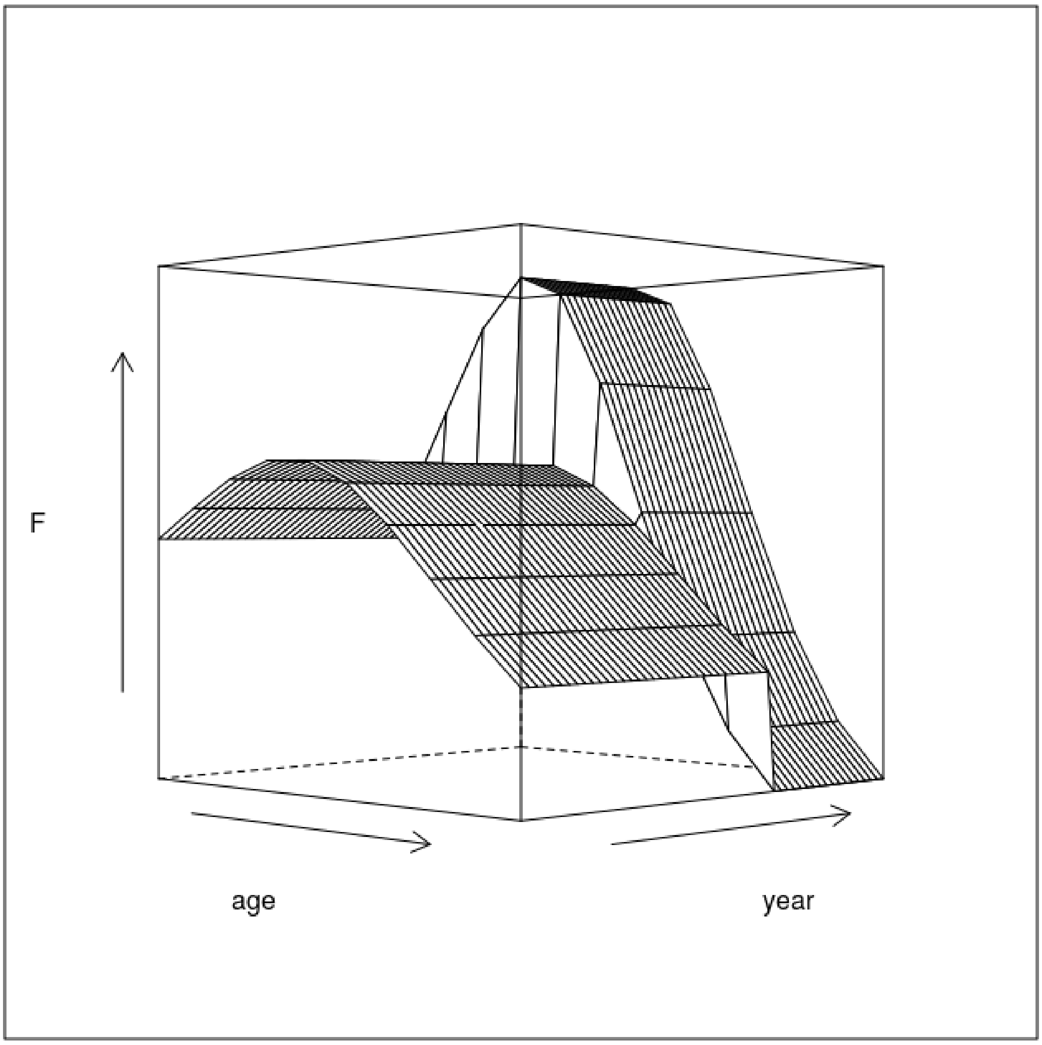


Figure 36: F-at-age in two periods using in both cases a thin plate spline with 3 knots

More complicated models can be built with these tools. For example, Figure 37 shows a model where the age effect is modelled as a smoother (the same thin plate spline) throughout years but independent from each other.

```
fmod <- ~factor(age) + s(year, k = 10, by = breakpts(age, c(2:8)))
fit <- sca(ple4, ple4.indices, fmod)
```

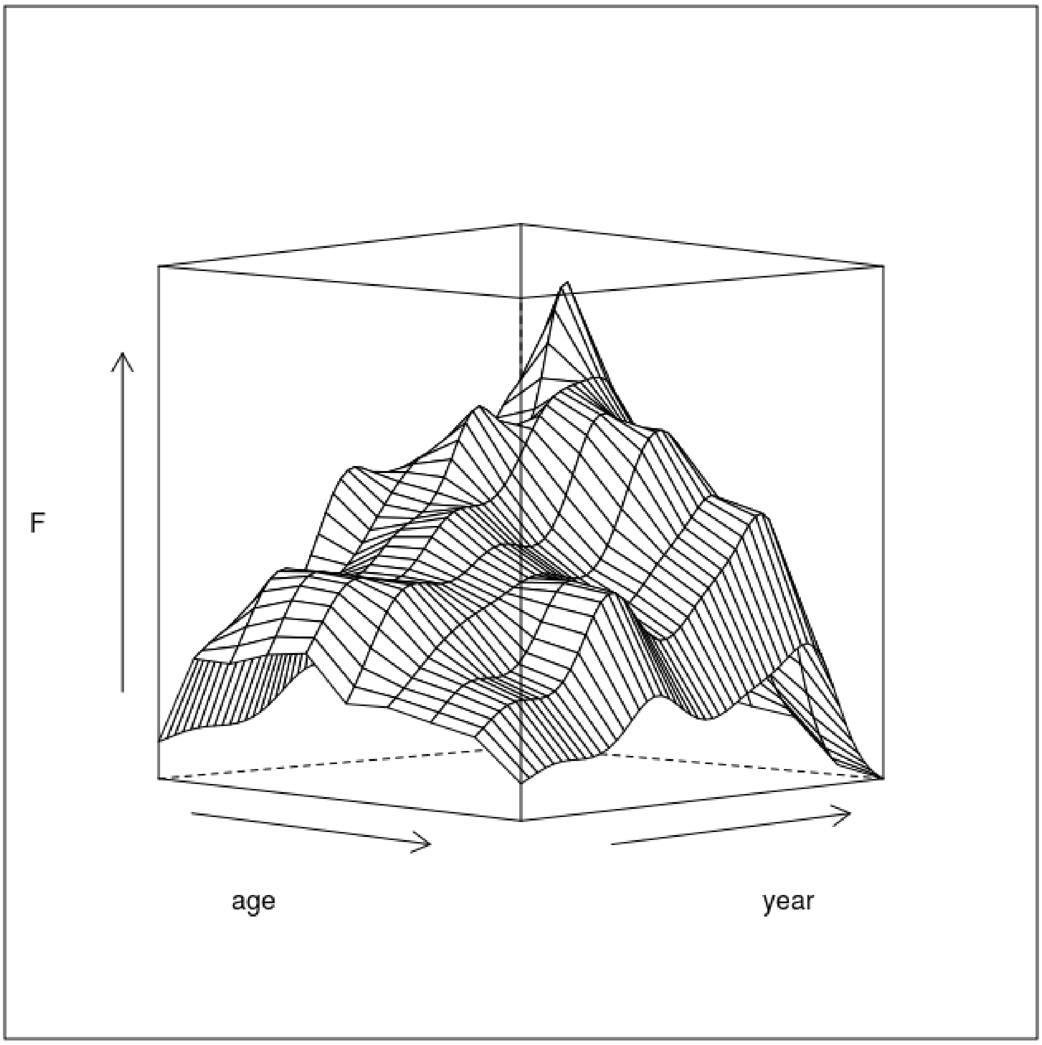


Figure 37: F-at-age as thin plate spline with 3 knots for each age

A quite complex model that implements a cohort effect can be set through the following formula. Figure 38 shows the resulting fishing mortality. Note that in this case we end up with a variable F pattern over time, but rather than using 4 * 10 = 40 parameters, it uses, 4 + 10 + 10 = 24.

```
fmodel <- ~s(age, k = 4) + s(pmax(year - age, 1957), k = 10) + s(year, k = 10)
fit <- sca(ple4, ple4.indices, fmodel = fmodel)
```

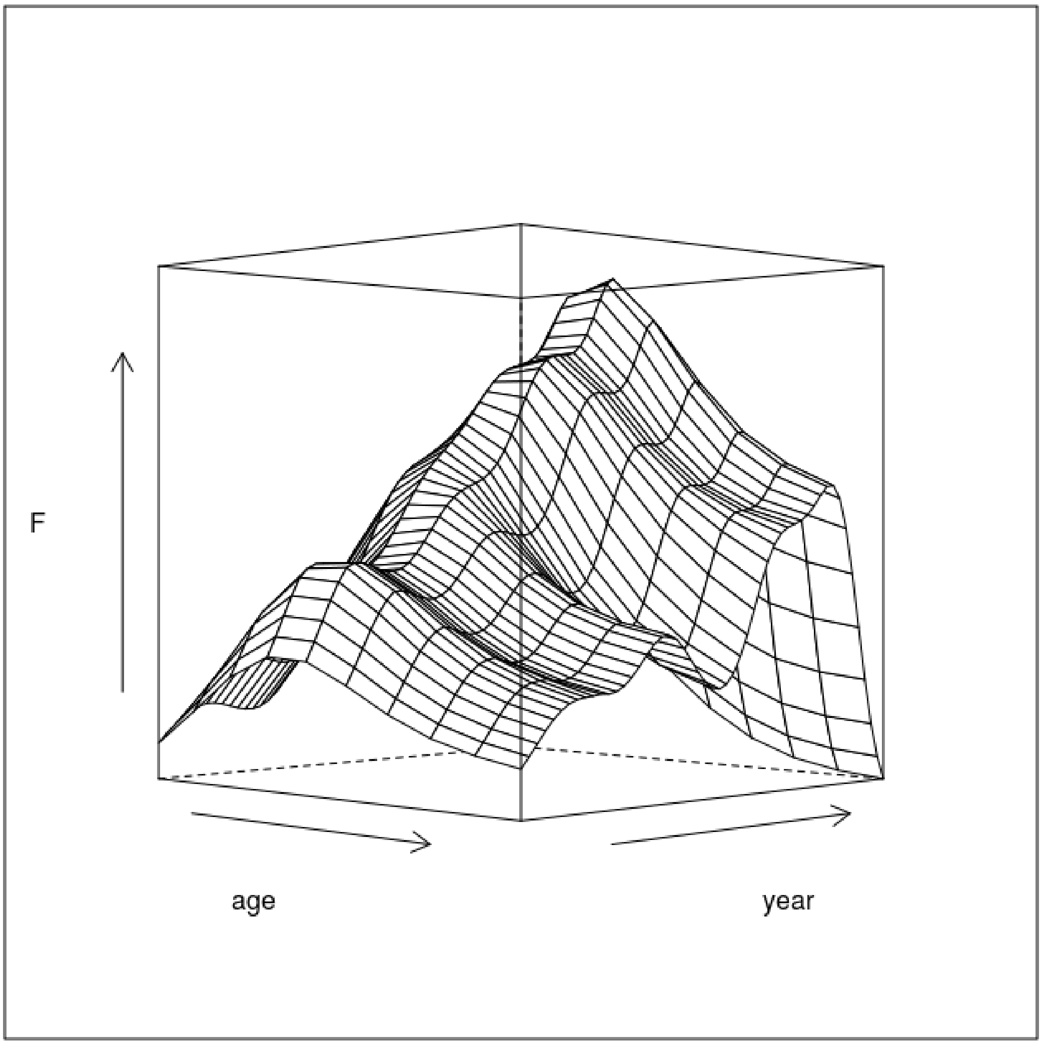


Figure 38: F-at-age with a cohort effect.

11.3 Propagate natural mortality uncertainty

In this section we give an example of how uncertainty in natural mortality, set up using the `m()` method and the class *a4aM* (see Section **??**), is propagated through the stock assessment. We'll start by fitting the default model to the data.

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

Using the `a4a` methods we'll model natural mortality using a negative exponential model by age, Jensen's estimator for the level and a constant trend with time. We include multivariate normal uncertainty using the `mvrnorm()` method and create 25 iterations.

```
nits <- 25

shape <- FLModelSim(model = ~exp(-age - 0.5))
level <- FLModelSim(model = ~k^0.66 * t^0.57, params = FLPar(k = 0.4, t = 10),
    vcov = matrix(c(0.002, 0.01, 0.01, 1), ncol = 2))
trend <- FLModelSim(model = ~b, params = FLPar(b = 0.5), vcov = matrix(0.02))

m4 <- a4aM(shape = shape, level = level, trend = trend)
m4 <- mvrnorm(nits, m4)
range(m4)[] <- range(ple4)[]
range(m4)[c("minmbar", "maxmbar")] <- c(1, 1)
flq <- m(m4)[]
quant(flq) <- "age"
stk <- propagate(ple4, nits)
m(stk) <- flq
```

We fit the same model to the new stock object which has uncertainty in the natural mortality. The assessment is performed for each of the 25 iterations.

```
fit1 <- sca(stk, ple4.indices)
```

And compare the two results (Figure 39). It's quite easy to run these kind of tests and a large part of our effort is to create the tools to do so.

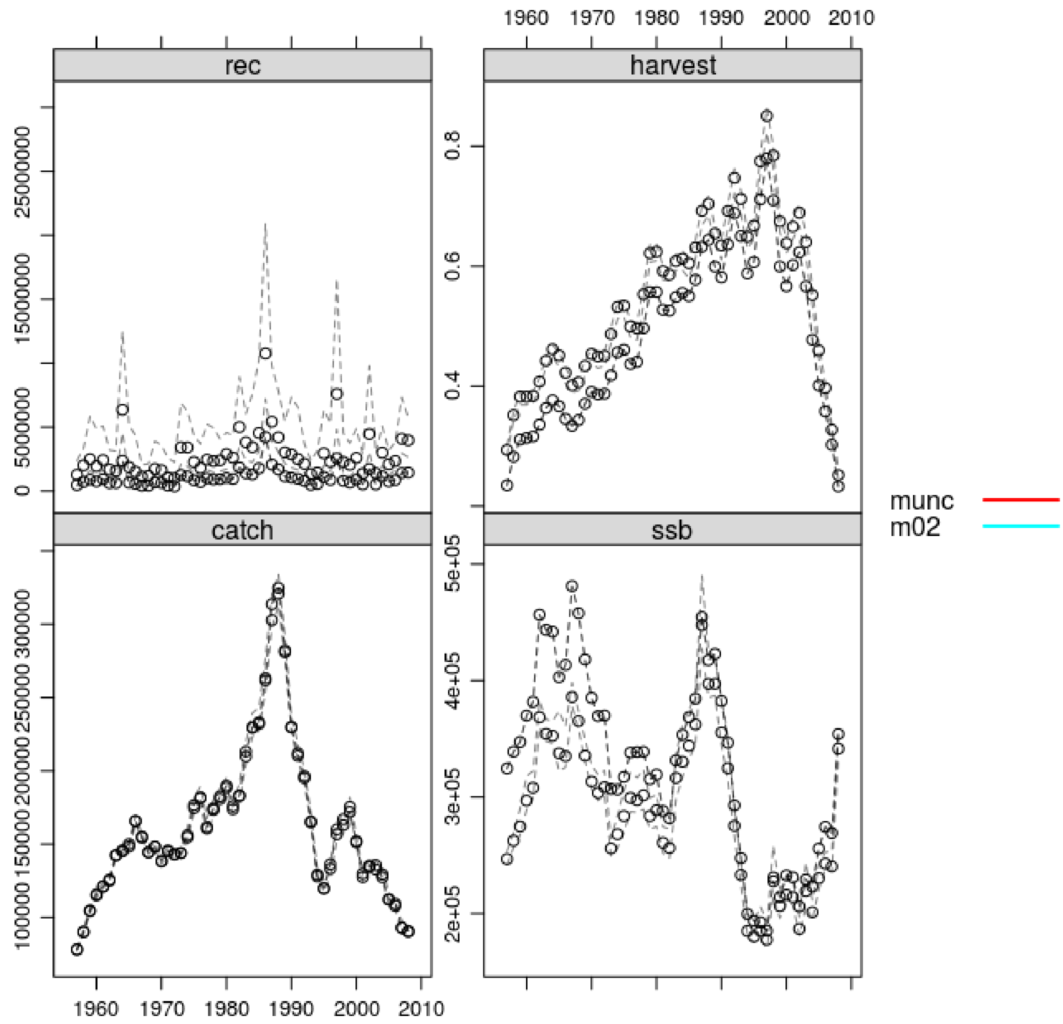


Figure 39: Stock summary for two M models

11.4 WCSAM exercise - replicating itself

The World Conference on Stock Assessment Methods (WCSAM) promoted a workshop where a large simulation study was used to test the performance of distinct stock assessment models. The first criteria used was that the models should be able to reproduce itself. The process involved fitting the model, simulating observation error using the same model, and refitting the model to each iteration. The final results should be similar to the fitted results before observation error was added (see Deroba, et.al, 2014 for details). The following analysis runs this analysis and Figure 40 presents the results.

```
# number of iters
nits <- 25
# fit the model
fit <- a4aSCA(ple4, ple4.indices[1])
# update the stock data
stk <- ple4 + fit
# simulate controlling the random seed
fits <- simulate(fit, nits, 1234)
# update stock and index data, now with iters
stks <- ple4 + fits
idxs <- ple4.indices[1]
index(idxs[[1]]) <- index(fits)[[1]]
# run assessments on each iter
sfit <- a4aSCA(stks, idxs, fit = "MP")
```

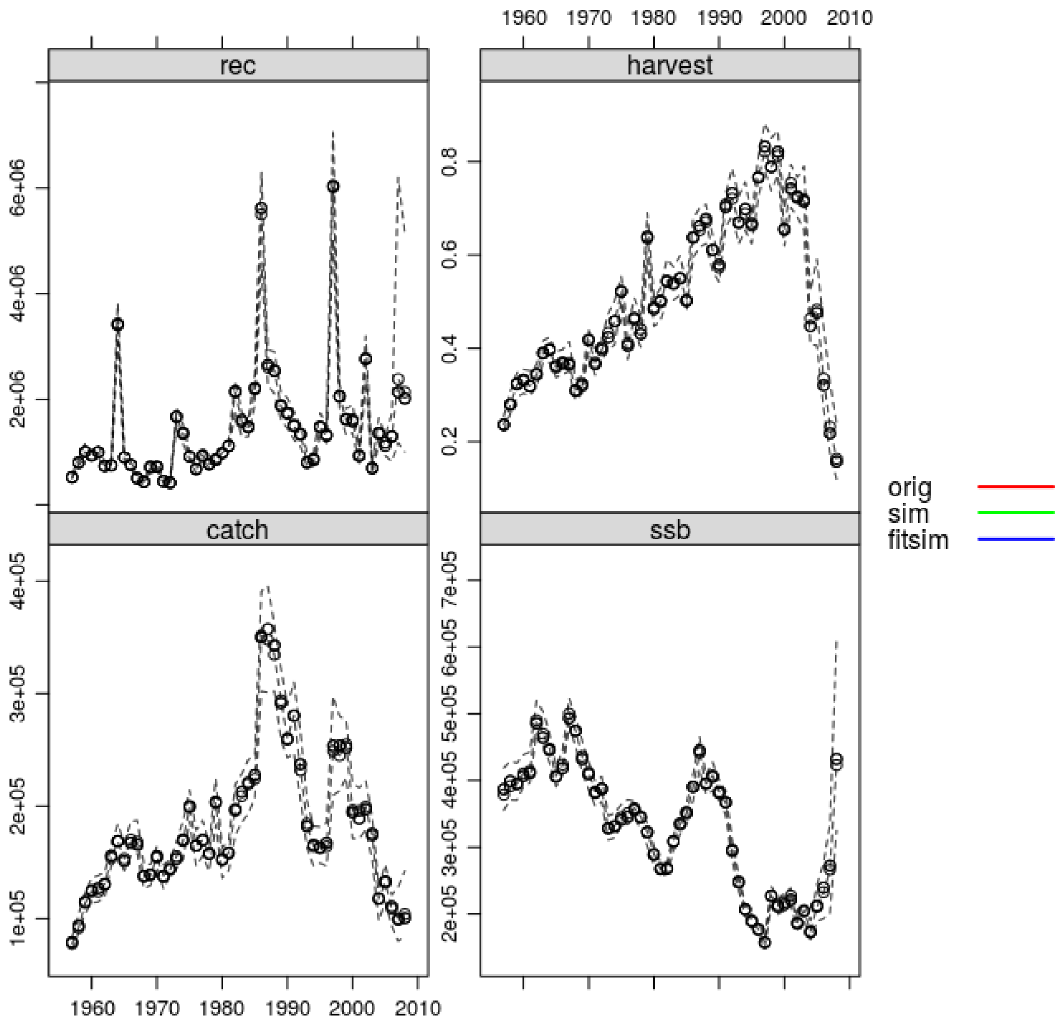


Figure 40: Replicating the stock assessment model (WCSAM approach)

11.5 Parallel computing

This is an example of how to use the `parallel` R package to run assessments. In this example each iteration is a dataset, including surveys, and we'll run one assessment for each iteration. Afterwards the data is pulled back together in an *FLStock* object and plotted (Figure 41). Only 20 iterations are run to avoid taking too long. Also note that we're using 4 cores. This parameter depends on the computer being used. These days almost all computers have at least 2 cores.

Finally, compare this code with the one for replicating WCSAM and note that it's exactly the same, except that we're using `mclapply()` from package `paralell` instead of `lapply()`.

```
data(ple4)
data(ple4.indices)
nits <- 25
fit <- a4aSCA(ple4, ple4.indices[1])
stk <- ple4 + fit
fits <- simulate(fit, nits, 1234)
stks <- ple4 + fits
idxs <- ple4.indices[1]
index(idxs[[1]]) <- index(fits)[[1]]
library(parallel)
lst <- mclapply(split(1:nits, 1:nits), function(x) {
    out <- try(a4aSCA(iter(stks, x), FLIndices(iter(idxs[[1]], x)), fit = "MP"))
    if (is(out, "try-error"))
        NULL else out
})

stks2 <- stks
for (i in 1:nits) {
    iter(catch.n(stks2), i) <- catch.n(lst[[i]])
    iter(stock.n(stks2), i) <- stock.n(lst[[i]])
    iter(harvest(stks2), i) <- harvest(lst[[i]])
}
catch(stks2) <- computeCatch(stks2)
stock(stks2) <- computeStock(stks2)
stks3 <- FLStocks(orig = stk, sim = stks, fitsim = stks2)
```

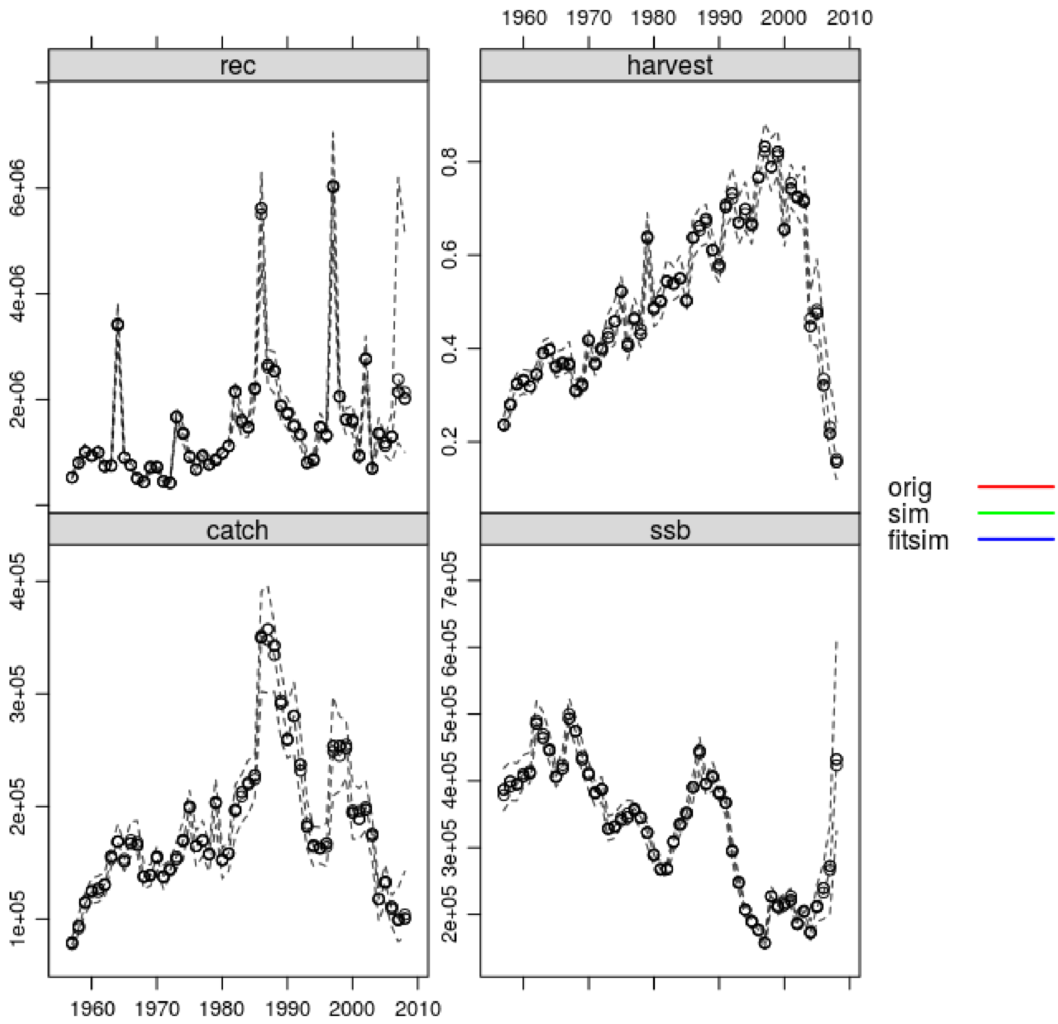


Figure 41: Replicating the stock assessment model (WCSAM approach) using parallel computing

12 Model averaging

To merge results from several fits, using distinct models or datasets, we follow Millar, et.al, 2014. The method `ma()` is the wrapper to the distinct methods, although for now only the AIC averaging is implemented. Figures 42 and 43 show the results.

```
data(ple4)
data(ple4.indices)
```

```
f1 <- sca(ple4, ple4.indices, fmodel = ~factor(age) + s(year, k = 20), qmodel = list(~s(age,
    k = 4), ~s(age, k = 4), ~s(age, k = 3)), fit = "assessment")
f2 <- sca(ple4, ple4.indices, fmodel = ~factor(age) + s(year, k = 20), qmodel = list(~s(age,
    k = 4) + year, ~s(age, k = 4), ~s(age, k = 3)), fit = "assessment")
stock.sim <- ma(a4aFitSAs(list(f1 = f1, f2 = f2)), ple4, AIC, nsim = 100)
stks <- FLStocks(f1 = ple4 + f1, f2 = ple4 + f2, ma = stock.sim)
```

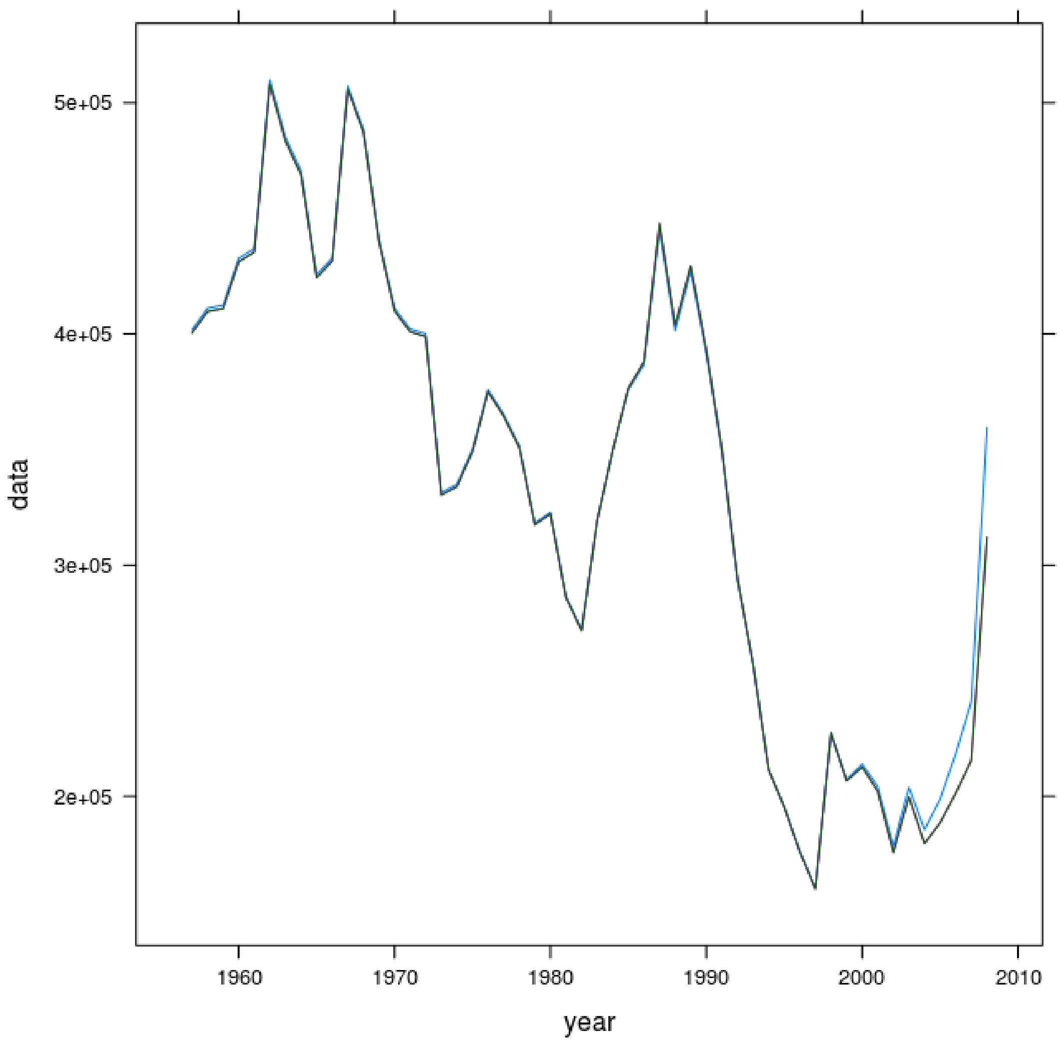


Figure 42: SSB of the two models and their average

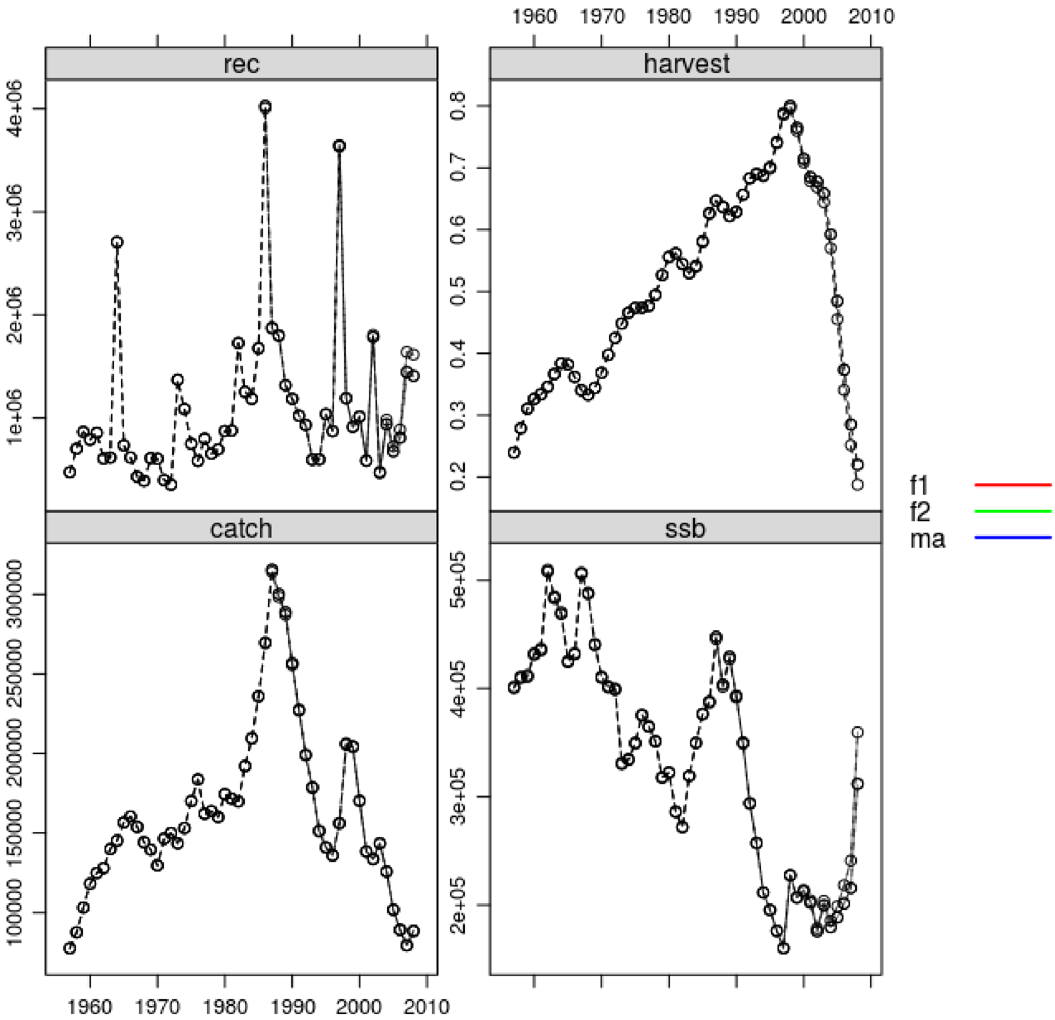


Figure 43: Stock summaries of the two models and their average