Stock assessment and management advice with a4a methods DRAFT

Ernesto Jardim[1], Colin Millar[1], and Finlay Scott[1]

[1]European Commission, Joint Research Centre, IPSC / Maritime Affairs Unit, 21027 Ispra (VA), Italy
*Corresponding author ernesto.jardim@jrc.ec.europa.eu

July 17, 2013

Contents

1 Introduction

- Objectives
- a4a concepts
  - life history considers parameters to have distributions, it's a kind of Bayesian posteriors informed estimates, but if one runs a Bayesian analysis to estimate growth parameters the posteriors can be used
- Workflow diagram

```
#
# ===============================================================================
# libraries and constants
# ===============================================================================

library(FLa4a)

## Loading required package:  FLCore

## Loading required package:  grid

## Loading required package:  lattice

## Loading required package:  MASS

##
## Attaching package:  'FLCore'

## The following object is masked from 'package:base':
##
##    cbind, rbind

## Loading required package:  Matrix

##
## Attaching package:  'Matrix'

## The following object is masked from 'package:FLCore':
##
##    expand

## Loading required package:  mgcv

## This is mgcv 1.7-22.  For overview type 'help("mgcv-package")'.

## Loading required package:  copula

## Loading required package:  triangle

## Warning:  replacing previous import 'barchart' when loading 'lattice'

## Warning:  replacing previous import 'bwplot' when loading 'lattice'

## Warning:  replacing previous import 'densityplot' when loading 'lattice'

## Warning:  replacing previous import 'dotplot' when loading 'lattice'

## Warning:  replacing previous import 'histogram' when loading 'lattice'

## Warning:  replacing previous import 'splom' when loading 'lattice'

## Warning:  replacing previous import 'stripplot' when loading 'lattice'

## Warning:  replacing previous import 'xyplot' when loading 'lattice'

## Warning:  replacing previous import 'expand' when loading 'Matrix'

## This is FLa4a 0.9.3.  For overview type 'help("FLa4a-package")'
```

```
library(XML)
library(reshape2)
data(rfLen)
data(ple4)
data(ple4.indices)
```

```
#
# ==============================================================================
# some functions for later
# ==============================================================================
# quant 2 quant
qt2qt <- function(object, id = 5, split = "-") {
    qt <- object[, id]
    levels(qt) <- unlist(lapply(strsplit(levels(qt), split = split), "[[", 2))
    as.numeric(as.character(qt))
}

# check import and massage
cim <- function(object, n, wt, hrv = "missing") {
    v <- object[sample(1:nrow(object), 1), ]
    c1 <- c(n[as.character(v$V5), as.character(v$V1), 1, as.character(v$V2)] ==
        v$V6)
    c2 <- c(wt[as.character(v$V5), as.character(v$V1), 1, as.character(v$V2)] ==
        v$V7)
    if (missing(hrv)) {
        c1 + c2 == 2
    } else {
        c3 <- c(hrv[as.character(v$V5), as.character(v$V1), 1, as.character(v$V2)] ==
            v$V8)
        c1 + c2 + c3 == 3
    }
}
```

2 Reading files and building FLR objects

For this document we'll use the plaice in ICES area IV dataset, provided by FLR, and a length-based simulated dataset based on red fish, using gadget (http://www.hafro.is/gadget), provided by Daniel Howell (Institute of Marine Research, Norway).

2.1 Red fish length based dataset

```
#
# ==============================================================================
# Read files
# ==============================================================================

# catch
cth.orig <- read.table("data/catch.len", skip = 5)

# stock
stk.orig <- read.table("data/red.len", skip = 4)

# surveys
```

```r
idx.orig <- read.table("data/survey.len", skip = 5)
idxJmp.orig <- read.table("data/jump.survey.len", skip = 5)
idxTrd.orig <- read.table("data/tend.survey.len", skip = 5)

#
# ==============================================================================
# Recode
# ==============================================================================

# catch
cth.orig[, 5] <- qt2qt(cth.orig)

# stock
stk.orig[, 5] <- qt2qt(stk.orig)

# surveys
idx.orig[, 5] <- qt2qt(idx.orig)
idxJmp.orig[, 5] <- qt2qt(idxJmp.orig)
idxTrd.orig[, 5] <- qt2qt(idxTrd.orig)
```

```r
#
# ==============================================================================
# cast
# ==============================================================================

# catch
cth.n <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V6", data = cth.orig)
cth.wt <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V7", data = cth.orig)
hrv <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V8", data = cth.orig)

# stock
stk.n <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V6", data = stk.orig)
stk.wt <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V7", data = stk.orig)

# surveys
idx.n <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V6", data = idx.orig)
idx.wt <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V7", data = idx.orig)
idx.hrv <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V8", data = idx.orig)

idxJmp.n <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V6", data = idxJmp.orig)
idxJmp.wt <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V7", data = idxJmp.orig)
idxJmp.hrv <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V8", data = idxJmp.orig)

idxTrd.n <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V6", data = idxTrd.orig)
idxTrd.wt <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V7", data = idxTrd.orig)
idxTrd.hrv <- acast(V5 ~ V1 ~ 1 ~ V2 ~ 1 ~ 1, value.var = "V8", data = idxTrd.orig)
```

```r
#
# ==============================================================================
# quants
# ==============================================================================

# catch
dnms <- dimnames(cth.n)
names(dnms) <- names(dimnames(FLQuant()))
names(dnms)[1] <- "len"
```

```
cth.n <- FLQuant(cth.n, dimnames = dnms)
cth.wt <- FLQuant(cth.wt, dimnames = dnms)
hrv <- FLQuant(hrv, dimnames = dnms)
units(hrv) <- "f"

# stock
dnms <- dimnames(stk.n)
names(dnms) <- names(dimnames(FLQuant()))
names(dnms)[1] <- "len"
stk.n <- FLQuant(stk.n, dimnames = dnms)
stk.wt <- FLQuant(stk.wt, dimnames = dnms)

# stock
dnms <- dimnames(idx.n)
names(dnms) <- names(dimnames(FLQuant()))
names(dnms)[1] <- "len"
idx.n <- FLQuant(idx.n, dimnames = dnms)
idx.wt <- FLQuant(idx.wt, dimnames = dnms)
idx.hrv <- FLQuant(idx.hrv, dimnames = dnms)

dnms <- dimnames(idxJmp.n)
names(dnms) <- names(dimnames(FLQuant()))
names(dnms)[1] <- "len"
idxJmp.n <- FLQuant(idxJmp.n, dimnames = dnms)
idxJmp.wt <- FLQuant(idxJmp.wt, dimnames = dnms)
idxJmp.hrv <- FLQuant(idxJmp.hrv, dimnames = dnms)

dnms <- dimnames(idxTrd.n)
names(dnms) <- names(dimnames(FLQuant()))
names(dnms)[1] <- "len"
idxTrd.n <- FLQuant(idxTrd.n, dimnames = dnms)
idxTrd.wt <- FLQuant(idxTrd.wt, dimnames = dnms)
idxTrd.hrv <- FLQuant(idxTrd.hrv, dimnames = dnms)
```

```
#
# ==============================================================================
# check
# ==============================================================================

#
# ------------------------------------------------------------------------------
# match original data
# ------------------------------------------------------------------------------

# catch
cim(cth.orig, cth.n, cth.wt, hrv)

## [1] TRUE


# stock
cim(stk.orig, stk.n, stk.wt)

## [1] TRUE


# surveys
cim(idx.orig, idx.n, idx.wt, idx.hrv)
```

```
## [1] TRUE

cim(idxJmp.orig, idxJmp.n, idxJmp.wt, idxJmp.hrv)

## [1] TRUE

cim(idxTrd.orig, idxTrd.n, idxTrd.wt, idxTrd.hrv)

## [1] TRUE
```

```
#
# ==============================================================================
# FLR objects
# ==============================================================================

rfLen.stk <- FLStockLen(stock.n = stk.n, stock.wt = stk.wt, stock = quantSums(stk.wt *
    stk.n), catch.n = cth.n, catch.wt = cth.wt/cth.n, catch = quantSums(cth.wt),
    harvest = hrv)
m(rfLen.stk)[] <- 0.05
mat(rfLen.stk)[] <- m.spwn(rfLen.stk)[] <- harvest.spwn(rfLen.stk)[] <- 0
mat(rfLen.stk)[38:59, , , 3:4] <- 1

rfTrawl.idx <- FLIndex(index = idx.n, catch.n = idx.n, catch.wt = idx.wt, sel.pattern = idx.hrv)
effort(rfTrawl.idx)[] <- 100

rfTrawlJmp.idx <- FLIndex(index = idxJmp.n, catch.n = idxJmp.n, catch.wt = idxJmp.wt,
    sel.pattern = idxJmp.hrv)
effort(rfTrawlJmp.idx)[] <- 100

rfTrawlTrd.idx <- FLIndex(index = idxTrd.n, catch.n = idxTrd.n, catch.wt = idxTrd.wt,
    sel.pattern = idxTrd.hrv)
effort(rfTrawlTrd.idx)[] <- 100

#
# ------------------------------------------------------------------------------
# save
# ------------------------------------------------------------------------------

save(rfLen.stk, rfTrawl.idx, rfTrawlJmp.idx, rfTrawlTrd.idx, file = "rfLen.rdata")
```

3 Converting length data to age

The stock assessment framework is based on age dynamics. To use length information it must be pre-processed before used for assessment. The rationale is that the pre-processing should give the analyst the flexibility to use whatever sources of information, *e.g.* literature or online databases, to grab information about the species growth and the uncertainty about the model parameters.

3.1 a4aGr - The growth class

The convertion of length data to age is performed through the usage of a growth model. The implementation is done through the *a4aGr* class. Check the help file for more information.

```
showClass("a4aGr")

## Class "a4aGr" [package "FLa4a"]
##
## Slots:
##
## Name:      grMod  grInvMod    params      vcov     distr      name
## Class:   formula   formula     FLPar     array character character
##
## Name:       desc     range
## Class: character   numeric
##
## Extends: "FLComp"
```

A simple construction of *a4aGr* objects requires the model and parameters to be provided.

```
#
# ------------------------------------------------------------------------------
# a von Bertalanffy model
# ------------------------------------------------------------------------------

vbObj <- a4aGr(grMod = ~linf * (1 - exp(-k * (t - t0))), grInvMod = ~t0 - 1/k *
    log(1 - len/linf), params = FLPar(linf = 58.5, k = 0.086, t0 = 0.001, units = c("cm",
    "ano-1", "ano")))

# a quick check about the model and it's inverse
lc = 20
predict(vbObj, t = predict(vbObj, len = lc)) == lc

##    iter
##        1
##   1 TRUE
```

The predict method will allow the transformation between age and lengths.

```
#
# ------------------------------------------------------------------------------
# predicting ages from lengths and vice-versa
# ------------------------------------------------------------------------------
predict(vbObj, len = 5:10 + 0.5)

##    iter
##         1
##   1 1.149
##   2 1.371
##   3 1.596
##   4 1.827
##   5 2.062
##   6 2.301

predict(vbObj, t = 5:10 + 0.5)

##    iter
##         1
##   1 22.04
##   2 25.05
```

```
##   3 27.80
##   4 30.33
##   5 32.66
##   6 34.78
```

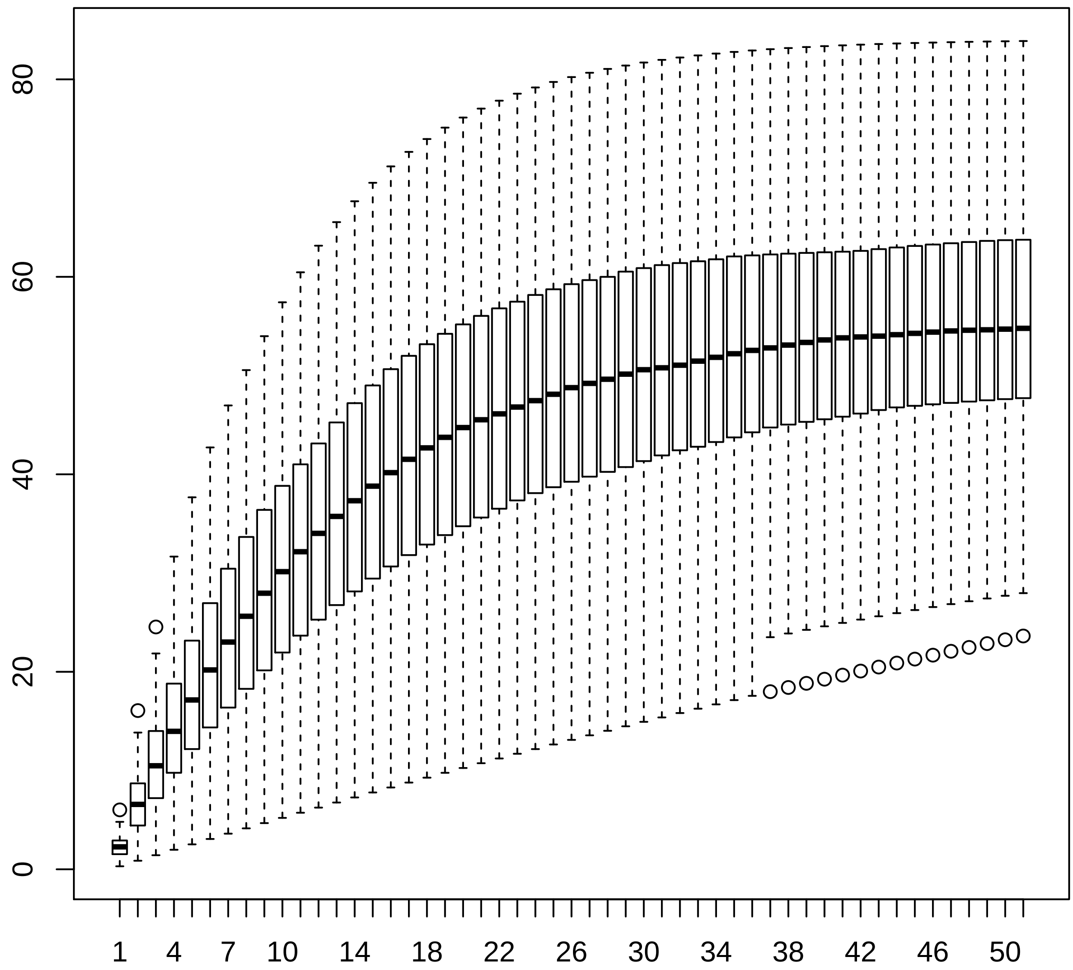
3.2 Adding multivariate normal parameter uncertainty

Uncertainty is introduced through parameter's uncertainty. The most traditional multivariate normal approach can be used. The implementation for *a5aGr* makes use of the *vcov* slot to get the parameter's covariance matrix. If the parameters or the covariance matrix have iterations then the medians accross iterations are computed before simulating. Check help for *mvrnorm* for more information.

```
# vcov matrix
mm <- matrix(NA, ncol = 3, nrow = 3)
diag(mm) <- c(100, 0.001, 0.001)
mm[upper.tri(mm)] <- mm[lower.tri(mm)] <- c(0.1, 0.1, 3e-04)
# object
vbObj <- a4aGr(grMod = ~linf * (1 - exp(-k * (t - t0))), grInvMod = ~t0 - 1/k *
    log(1 - len/linf), params = FLPar(linf = 58.5, k = 0.086, t0 = 0.001, units = c("cm",
    "ano-1", "ano")), vcov = mm, distr = "norm")
# simulate
vbObj <- mvrnorm(100, vbObj)
# predict
predict(vbObj, len = 5:10 + 0.5)

##    iter
##         1     2      3     4     5     6      7      8      9    10    11
##   1 1.656 10.06 0.9038 1.053 0.967 1.234 0.9326  4.937 0.7884 3.210 1.539
##   2 1.982 11.99 1.0662 1.259 1.155 1.480 1.1163  5.918 0.9325 3.833 1.840
##   3 2.316 13.95 1.2322 1.469 1.346 1.732 1.3033  6.925 1.0787 4.469 2.147
##   4 2.660 15.93 1.4021 1.683 1.541 1.991 1.4936  7.958 1.2269 5.117 2.462
##   5 3.014 17.95 1.5760 1.902 1.739 2.256 1.6875  9.018 1.3773 5.780 2.784
##   6 3.377 20.00 1.7542 2.126 1.942 2.528 1.8850 10.108 1.5300 6.456 3.114
##    iter
##        12    13     14    15     16    17    18    19    20    21    22
##   1 2.643 1.247 0.6603 1.049 0.7500 1.006 1.201 1.015 2.089 2.747 1.355
##   2 3.149 1.482 0.7780 1.247 0.8839 1.199 1.446 1.222 2.498 3.283 1.605
##   3 3.664 1.722 0.8975 1.449 1.0199 1.396 1.696 1.434 2.917 3.831 1.858
##   4 4.190 1.967 1.0189 1.655 1.1580 1.597 1.951 1.651 3.344 4.393 2.116
##   5 4.726 2.218 1.1423 1.866 1.2983 1.803 2.213 1.872 3.782 4.968 2.378
##   6 5.273 2.475 1.2676 2.081 1.4408 2.013 2.481 2.098 4.231 5.557 2.645
##    iter
##        23    24     25    26    27    28    29    30    31    32     33
##   1 1.468 1.123 0.9052 1.072 2.030 2.002 1.188 1.788 1.134 1.069  6.321
##   2 1.758 1.318 1.0781 1.275 2.421 2.383 1.424 2.124 1.348 1.278  7.555
##   3 2.053 1.517 1.2540 1.482 2.821 2.772 1.665 2.466 1.566 1.493  8.815
##   4 2.354 1.718 1.4329 1.693 3.229 3.167 1.911 2.815 1.786 1.712 10.102
##   5 2.660 1.922 1.6149 1.909 3.647 3.571 2.163 3.171 2.010 1.938 11.417
##   6 2.972 2.129 1.8003 2.129 4.074 3.983 2.421 3.533 2.238 2.169 12.762
##    iter
##         34    35     36    37    38     39    40     41    42    43    44
##   1 0.7730 1.749 0.9351 1.534 4.546 0.7572 1.301 0.9707 1.834 2.257 2.627
##   2 0.9221 2.103 1.1061 1.832 5.467 0.8931 1.558 1.1555 2.185 2.688 3.141
##   3 1.0738 2.468 1.2795 2.138 6.418 1.0315 1.820 1.3437 2.541 3.126 3.667
##   4 1.2282 2.847 1.4555 2.450 7.402 1.1726 2.087 1.5354 2.902 3.573 4.204
##   5 1.3854 3.240 1.6341 2.771 8.421 1.3163 2.359 1.7308 3.270 4.028 4.754
```

```
##   6 1.5455 3.647 1.8153 3.099 9.477 1.4628 2.636 1.9300 3.644 4.492 5.316
##    iter
##        45    46    47     48    49     50    51     52    53     54     55
##   1 1.543 1.071 1.665 0.9541 0.851 0.4528 1.734 0.9986 1.291  5.837 0.9269
##   2 1.841 1.275 1.992 1.1289 1.003 0.5441 2.068 1.1923 1.551  7.006 1.0918
##   3 2.145 1.483 2.327 1.3067 1.157 0.6369 2.407 1.3897 1.817  8.209 1.2597
##   4 2.452 1.696 2.671 1.4877 1.313 0.7311 2.752 1.5908 2.090  9.450 1.4305
##   5 2.764 1.913 3.024 1.6720 1.471 0.8270 3.104 1.7958 2.370 10.731 1.6045
##   6 3.080 2.135 3.387 1.8596 1.632 0.9244 3.463 2.0048 2.656 12.055 1.7817
##    iter
##         56     57    58    59     60     61     62    63    64    65
##   1 0.9986 0.7824 4.805 2.900 0.8271 0.7683 0.9327 1.408 1.472 3.961
##   2 1.1894 0.9323 5.741 3.499 0.9867 0.9144 1.1047 1.679 1.763 4.752
##   3 1.3839 1.0843 6.696 4.121 1.1496 1.0630 1.2794 1.955 2.062 5.564
##   4 1.5821 1.2387 7.670 4.769 1.3159 1.2141 1.4567 2.235 2.367 6.400
##   5 1.7842 1.3954 8.665 5.443 1.4859 1.3677 1.6367 2.521 2.680 7.260
##   6 1.9903 1.5545 9.681 6.148 1.6596 1.5241 1.8197 2.812 3.001 8.147
##    iter
##         66     67     68     69    70    71    72    73    74    75     76
##   1 0.7976 0.5729  5.918 0.6912 2.498 2.975 1.090 1.474 1.988 1.057 0.9543
##   2 0.9487 0.6787  7.099 0.8286 2.985 3.558 1.301 1.768 2.381 1.267 1.1374
##   3 1.1019 0.7858  8.319 0.9686 3.481 4.153 1.515 2.067 2.783 1.482 1.3247
##   4 1.2572 0.8943  9.580 1.1112 3.986 4.760 1.733 2.372 3.194 1.701 1.5165
##   5 1.4148 1.0043 10.884 1.2567 4.501 5.381 1.955 2.684 3.614 1.926 1.7129
##   6 1.5747 1.1158 12.235 1.4050 5.027 6.014 2.181 3.002 4.043 2.155 1.9143
##    iter
##         77    78    79     80    81    82    83    84     85    86    87
##   1 0.6008 2.364 1.101 0.6648 0.939 1.574 1.663 1.502 0.9415 1.950 1.925
##   2 0.7129 2.834 1.307 0.7930 1.113 1.878 1.998 1.794 1.1167 2.335 2.300
##   3 0.8269 3.317 1.518 0.9231 1.290 2.189 2.341 2.091 1.2954 2.728 2.684
##   4 0.9431 3.812 1.732 1.0552 1.470 2.508 2.694 2.395 1.4777 3.131 3.078
##   5 1.0614 4.322 1.950 1.1893 1.653 2.834 3.056 2.705 1.6639 3.542 3.481
##   6 1.1820 4.845 2.173 1.3256 1.839 3.168 3.430 3.021 1.8540 3.964 3.895
##    iter
##        88     89     90     91    92    93     94    95    96     97    98
##   1 1.668 0.7225 0.6353 0.6552 1.516 2.393 0.8919 1.807 1.596 0.8346 3.864
##   2 1.990 0.8601 0.7518 0.7767 1.819 2.855 1.0598 2.168 1.911 0.9806 4.624
##   3 2.321 1.0003 0.8701 0.9002 2.130 3.326 1.2305 2.537 2.234 1.1288 5.404
##   4 2.660 1.1430 0.9901 1.0257 2.451 3.807 1.4039 2.915 2.563 1.2794 6.204
##   5 3.009 1.2884 1.1120 1.1534 2.781 4.297 1.5801 3.303 2.901 1.4323 7.026
##   6 3.368 1.4366 1.2357 1.2832 3.123 4.797 1.7594 3.701 3.246 1.5877 7.871
##    iter
##        99   100
##   1 1.245 1.575
##   2 1.484 1.881
##   3 1.726 2.195
##   4 1.972 2.517
##   5 2.223 2.848
##   6 2.477 3.186
```

```r
boxplot(t(predict(vbObj, t = 0:50 + 0.5)))
```



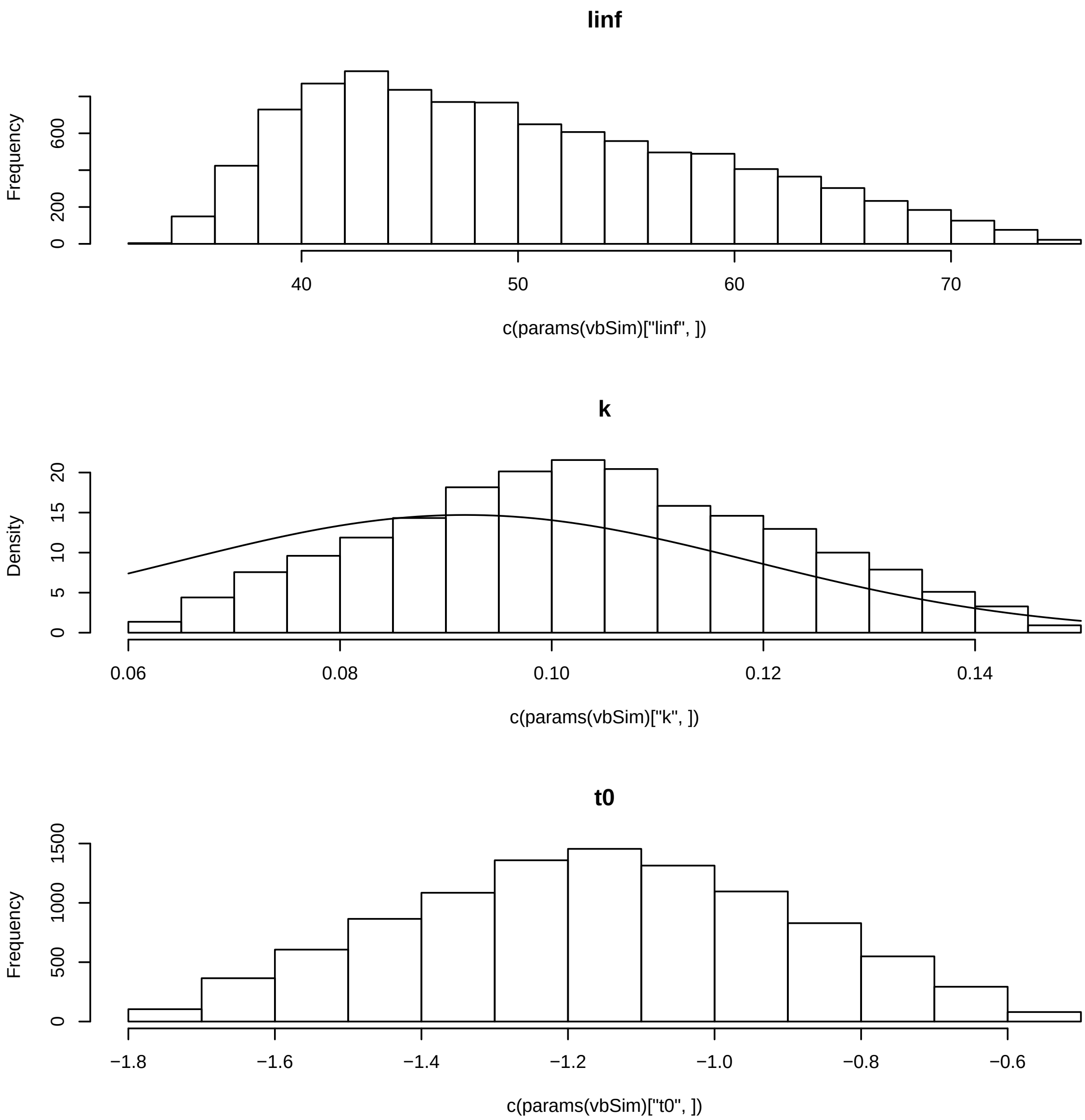
3.3 Adding parameter uncertainty with triangles and elliptic copulas

One alternative that may be interesting if one does not believe in assymptotic theory, is to use triangle distributions (http://en.wikipedia.org/wiki/Triangle_distribution). These distributions are parametrized using min, max and the most frequent value, which make them very interesting if the analyst needs to scrap information from the web or literature and perform some kind of meta-analysis.

```
addr <- "http://www.fishbase.org/PopDyn/PopGrowthList.php?ID=501"
tab <- try(readHTMLTable(addr))
linf <- as.numeric(as.character(tab$dataTable[, 2]))
k <- as.numeric(as.character(tab$dataTable[, 4]))
t0 <- as.numeric(as.character(tab$dataTable[, 5]))
vbObj <- a4aGr(grMod = ~linf * (1 - exp(-k * (t - t0))), grInvMod = ~t0 - 1/k *
    log(1 - len/linf), params = FLPar(linf = 58.5, k = 0.086, t0 = 0.001, units = c("cm",
    "ano-1", "ano")), vcov = mm)
pars <- list(list(a = min(linf), b = max(linf), c = median(linf)), list(a = min(k),
    b = max(k), c = median(k)), list(a = median(t0, na.rm = T) - IQR(t0, na.rm = T)/2,
    b = median(t0, na.rm = T) + IQR(t0, na.rm = T)/2))
vbSim <- mvrtriangle(10000, vbObj, paramMargins = pars)
```

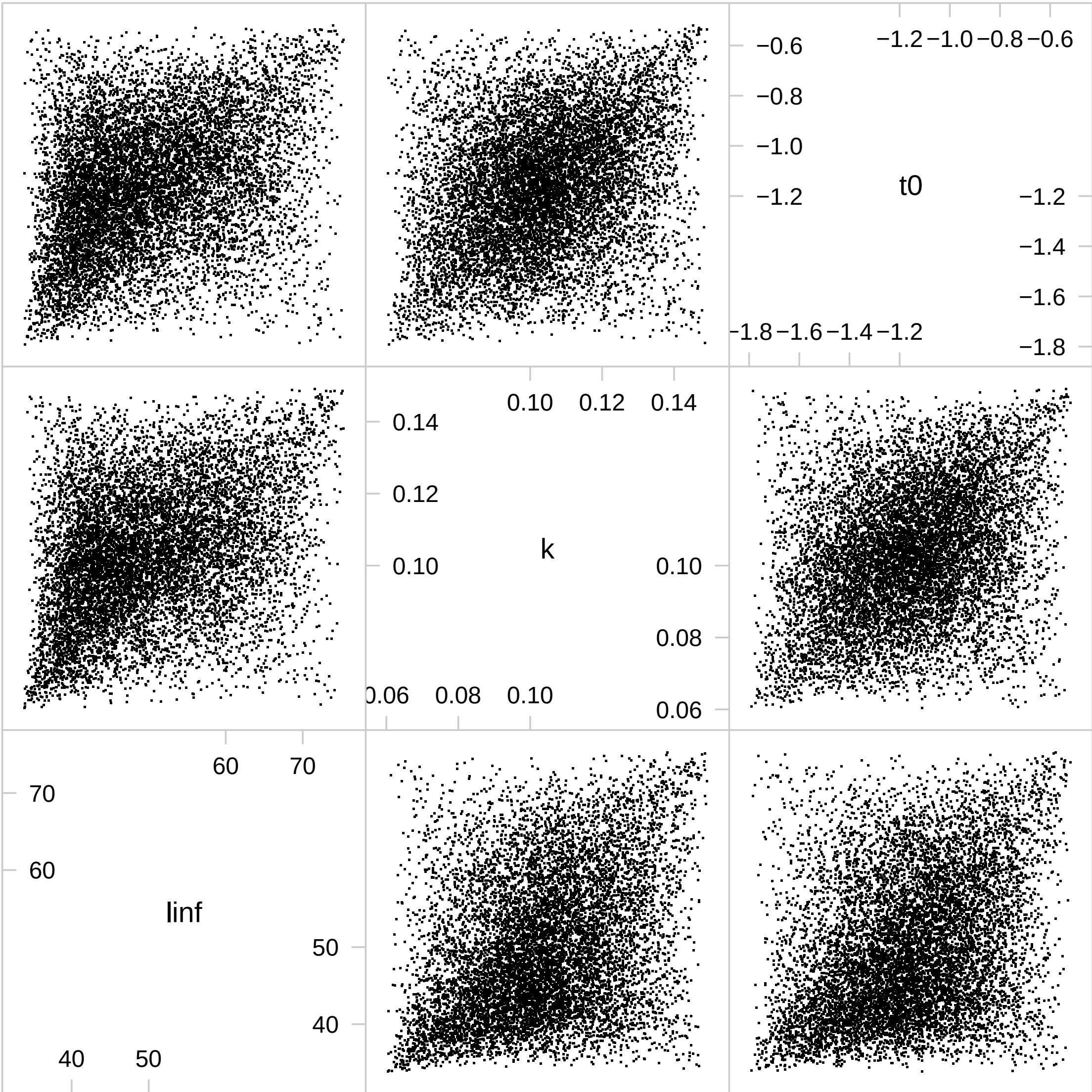
The marginals will reflect the uncertainty on the parameter values that were scrapped from fishbase, but, as we don't really believe the parameters are multivariate normal we addoppted a more relaxed distribution based on a t copula with triangle marginals.

```
par(mfrow = c(3, 1))
hist(c(params(vbSim)["linf", ]), main = "linf")
hist(c(params(vbSim)["k", ]), main = "k", prob = TRUE)
lines(x. <- seq(min(k), max(k), len = 100), dnorm(x., mean(k), sd(k)))
hist(c(params(vbSim)["t0", ]), main = "t0")
```



The shape of the correlation.

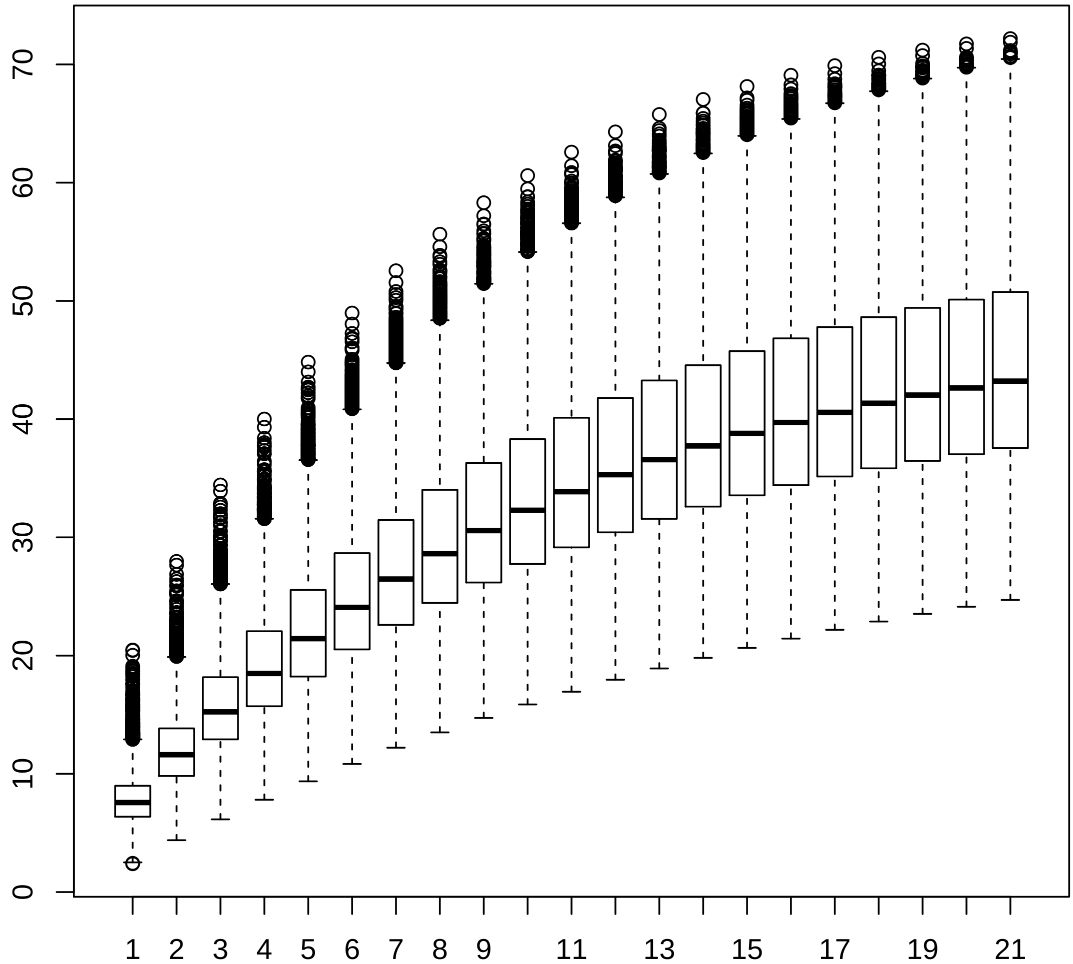
```
splom(data.frame(t(params(vbSim)@.Data)), pch = ".")
```



Scatter Plot Matrix

Off course one can still use predict to get the feeling about the growth model uncertainty.

```
boxplot(t(predict(vbSim, t = 0:20 + 0.5)))
```



If you want to be really geek, you may scrap the entire growth parameters dataset from fishbase and compute the shape of the variance covariance matrix yourself.

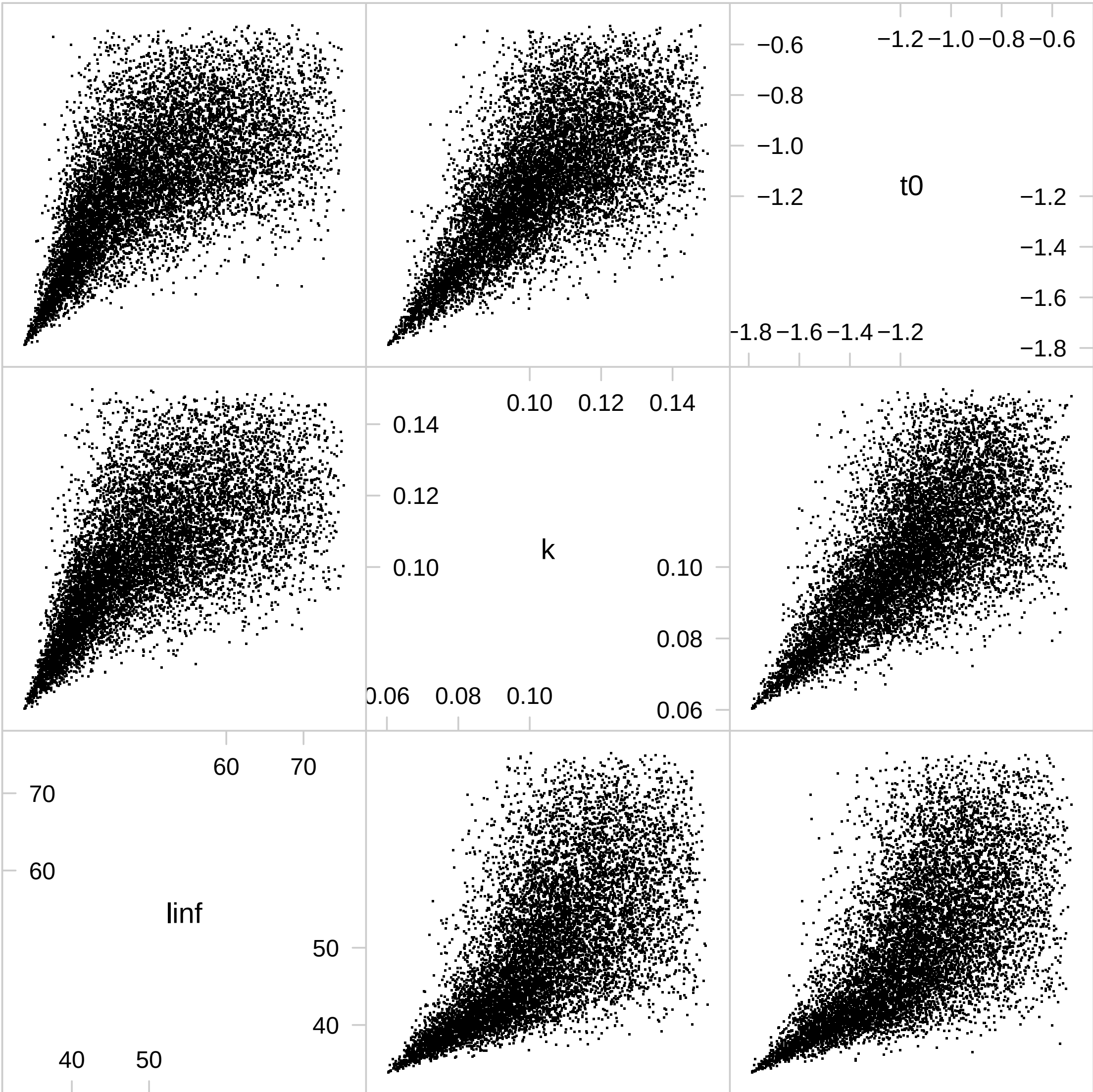
3.4 Adding parameter uncertainty with copulas

A more general approach is to make use of whatever copula and marginal distribution one wants. Which is possible with *mvrcop*. The example keeps the same parameters and changes only the copula type and family but a lot more can be done. Check the package *copula* for more.

```
vbSim <- mvrcop(10000, vbObj, copula = "archmCopula", family = "clayton", param = 2,
    margins = "triangle", paramMargins = pars)
```

The shape of the correlation changes.

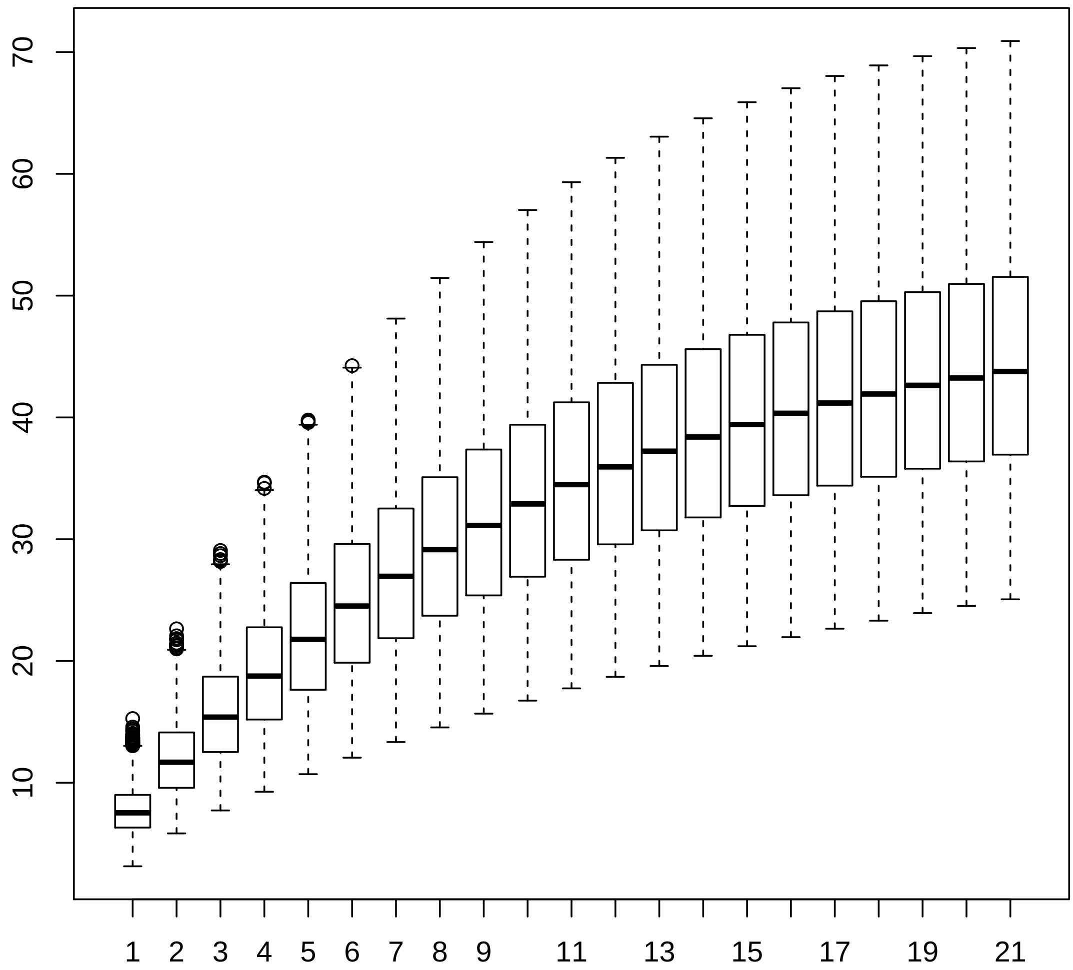
```
splom(data.frame(t(params(vbSim)@.Data)), pch = ".")
```



Scatter Plot Matrix

As well as the predictions.

```
boxplot(t(predict(vbSim, t = 0:20 + 0.5)))
```



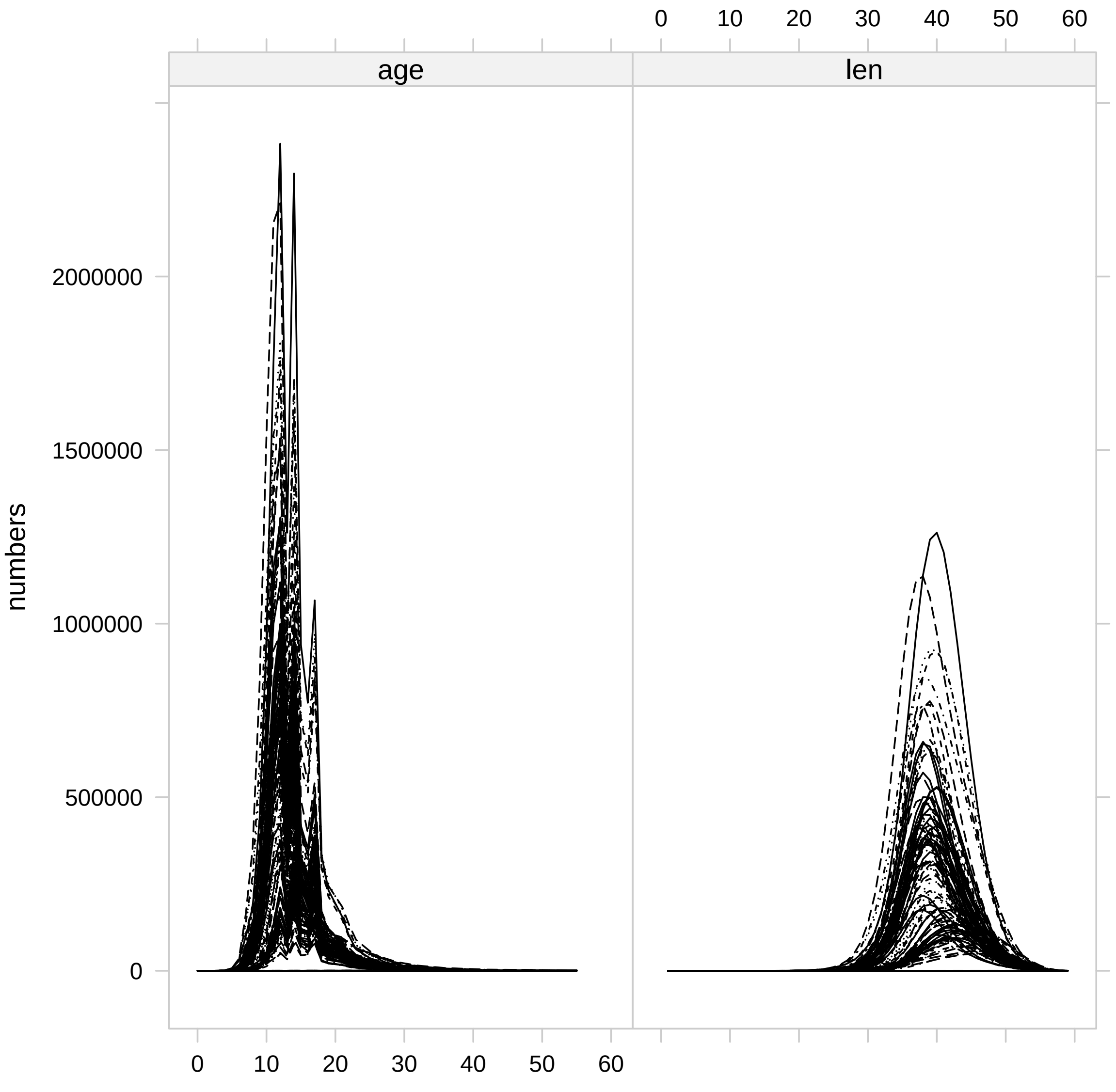
3.5 The "l2a" method

After introducing uncertainty on the growth model it's time to transform the length dataset into an age dataset. The method that deals with this process is *l2a*. The implementation for the *FLQuant* class is the workhorse. There's two other implementations, for *FLStock* and *FLIndex*, which are mainly wrappers that call the *FLQuant* method several times. Note that, for the moment, this method is quite slow. There's a double loop in the code that makes it slow, but we're working on a better solution.

```
# --------------------------------------------------------------------
# growth object
# --------------------------------------------------------------------
vbObj <- a4aGr(grMod = ~linf * (1 - exp(-k * (t - t0))), grInvMod = ~t0 - 1/k *
    log(1 - len/linf), params = FLPar(linf = 58.5, k = 0.086, t0 = 0.001, units = c("cm",
    "ano-1", "ano")), vcov = mm, distr = "norm")
# --------------------------------------------------------------------
# converte catch-at-length to catch-at-age
# --------------------------------------------------------------------
cth.n <- l2a(catch.n(rfLen.stk), vbObj)
```

```
## Converting lengths to ages ...

## Warning:  NaNs produced
```

```
# trick
quant(cth.n) <- "len"
xyplot(data ~ len | qname, groups = year, data = (FLQuants(len = catch.n(rfLen.stk),
    age = cth.n)), type = "l", xlab = "", ylab = "numbers")
```



Or we can convert all the relevant pieces of information in the stock and index dataset.

```
# -------------------------------------------------------------------
# convertion
# -------------------------------------------------------------------
aStk <- l2a(rfLen.stk, vbObj)

## Warning:  Individual weights, M and maturity will be averaged accross lengths, everything
else will be summed.  If this is not what you want, you'll have to deal with these slots
by hand.
```

```
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## Converting lengths to ages ...
## Warning:  NaNs produced
## [1] "maxfbar has been changed to accomodate new plusgroup"
aIdx <- l2a(rfTrawl.idx, vbObj)
```

```
## Warning:  Catch in numbers will be summed accross lenghths, everything else will be averaged.
If this is not what you want, you'll have to deal with these slots by hand.

## Converting lengths to ages ...

## Warning:  NaNs produced

## Converting lengths to ages ...

## Warning:  NaNs produced

## Converting lengths to ages ...

## Warning:  NaNs produced

## Converting lengths to ages ...

## Warning:  NaNs produced

## Converting lengths to ages ...

## Warning:  NaNs produced

## Converting lengths to ages ...

## Warning:  NaNs produced
```

When converting there's a number of defaults that the user must be aware.

All length above Linf are converted to the maximum age. This is not true in most cases, but that's as far as one can go with a age length growth model. There is no information on the model to deal with individuals larger than the maximum length. The variability around Linf is dealt by the randamization of the parameter Linf, and the cappacity to withold all the data depends on how well the analysist matches the variance of the parameter with the variance on the data.

4 Dealing with natural mortality

Natural mortality is dealt as an external parameter to the stock assessment model. The rationale is similar to that of growth. One should be able to grab information from whichever sources are available and use that information in a way that it propagates into stock assessment.

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

The smoother way to insert natural mortality in stock assessment is to use an *a4aM* object and run the method *m* to compute the values. The output is a *FLQuant* that should be directly inserted in the *FLStock* object to be used for assessment.

4.1 a4aM - The M class

Natural mortality is implemented in a class named *a4aM* which has three models of the class *FLModelSim*. Each model represents one effects. An age effect, an year effect and a time trend, named *shape*, *level* and *trend*, respectively. Check the help files for more information.

```
showClass("a4aM")

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

A simple construction of *a4aM* objects requires the models and parameters to be provided. The default method will build each of these models as a constant value of 1. For example the usual "0.2" guessestimate could be set up by

```
mod2 <- FLModelSim(model = ~a, params = FLPar(a = 0.2))
m1 <- a4aM(level = mod2)
```

Off course that would be too much work for the outcome. The interest is in using more knowledge setting M. The following example uses Jensen's second estimator (Kenshington, 2013) $M = 1.5K$ and an exponential decay to set up the level and shape of M.

```
# -------------------------------------------------------------------
# models or shape and level
# -------------------------------------------------------------------
mod1 <- FLModelSim(model = ~exp(-age - 0.5))
mod2 <- FLModelSim(model = ~1.5 * k, params = FLPar(k = 0.4))
# -------------------------------------------------------------------
# constructor
# -------------------------------------------------------------------
m2 <- a4aM(shape = mod1, level = mod2)
```

In alternative, an external factor may have impact on natural mortality which can be added through the *trend* model. Suppose M depends on NAO through some mechanism that results in having lower M when NAO is negative and higher when it's positive. The impact is represented by the NAO value on the quarter before spawning, which occurs in the second quarter.

```
# ------------------------------------------------------------------- get
# NAO -------------------------------------------------------------------
nao.orig <- 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.flq <- FLQuant(unlist(nao.orig[, -1]), dimnames = dnms, units = "nao")
# build covar
nao <- seasonMeans(nao.flq[, , , 1:3])
nao <- nao > 0
# ------------------------------------------------------------------- the
# trend model M increases 50% if NAO is positive on the first quarter
# -------------------------------------------------------------------
mod3 <- FLModelSim(model = ~1 + b * nao, params = FLPar(b = 0.5))
# -------------------------------------------------------------------
# constructor
# -------------------------------------------------------------------
mod1 <- FLModelSim(model = ~exp(-age - 0.5))
mod2 <- FLModelSim(model = ~1.5 * k, params = FLPar(k = 0.4))
m3 <- a4aM(shape = mod1, level = mod2, trend = mod3)
```

4.2 Adding multivariate normal parameter uncertainty

Uncertainty is added through error on parameters. In the case of this class it makes use of the *FLModelSim* "mvr" methods. A wrapper for *mvrnorm* was implemented, but all the other options must be carried out in each sub-model at the time.

```
# -------------------------------------------------------------------- the
# same exponential decay for shape
# --------------------------------------------------------------------
mod1 <- FLModelSim(model = ~exp(-age - 0.5))
# -------------------------------------------------------------------- For
# level we'll use Jensen's third estimator (Kenshington, 2013).
# --------------------------------------------------------------------
mod2 <- FLModelSim(model = ~k^0.66 * t^0.57, params = FLPar(matrix(c(0.4, 10)),
    dimnames = list(params = c("k", "t"), iter = 1)), vcov = array(c(0.002,
    0.01, 0.01, 1), dim = c(2, 2)))
# -------------------------------------------------------------------- and
# a trend from NAO
# --------------------------------------------------------------------
mod3 <- FLModelSim(model = ~1 + b * nao, params = FLPar(b = 0.5), vcov = matrix(0.02))
# --------------------------------------------------------------------
# create object and simulate
# --------------------------------------------------------------------
m4 <- a4aM(shape = mod1, level = mod2, trend = mod3)
m4 <- mvrnorm(100, m4)
```

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

```
m4 <- a4aM(shape = mod1, level = mvrnorm(100, mod2), trend = mvrnorm(100, mod3))
```

Note: How to include ageing error ???

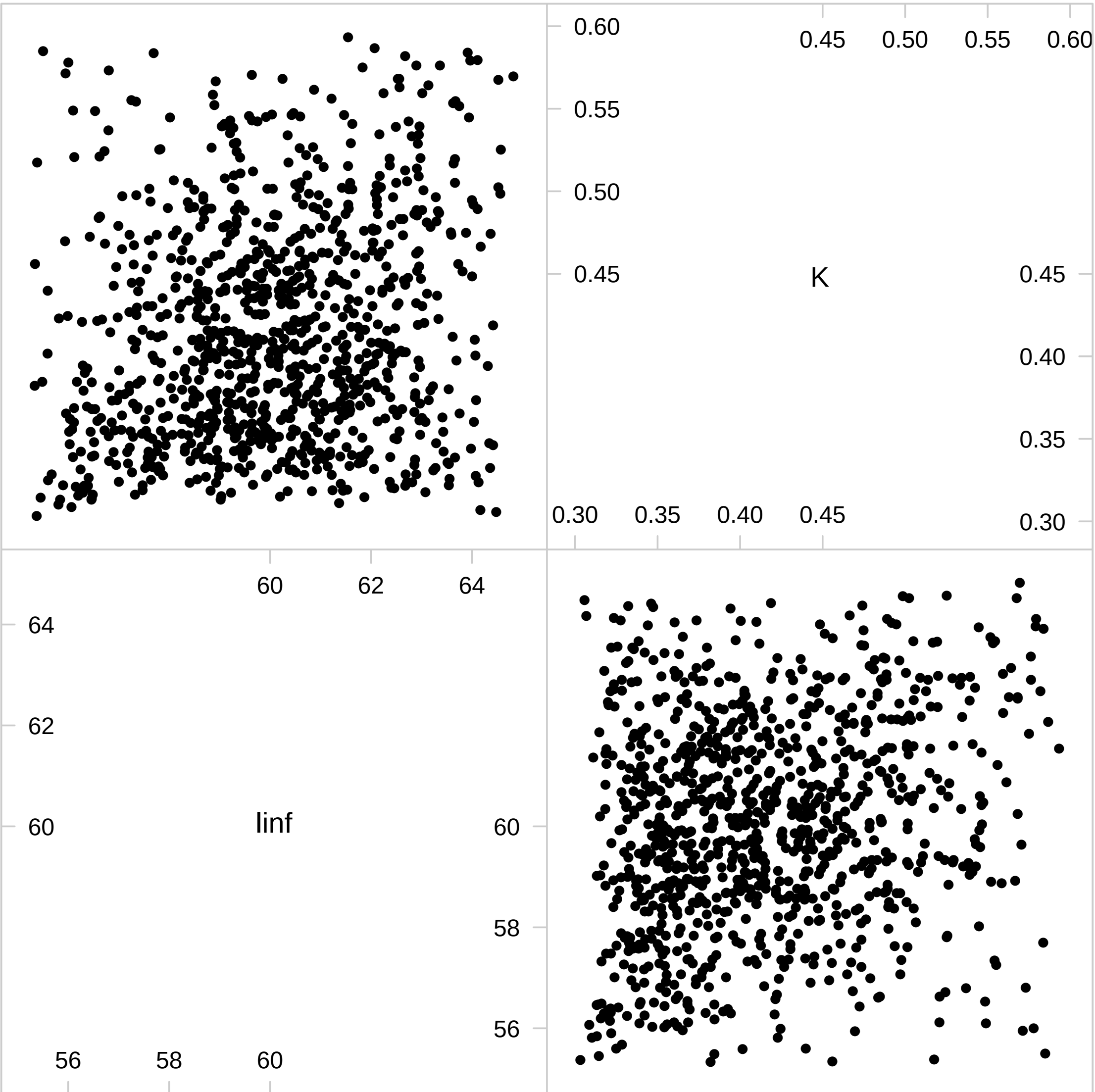
4.3 Adding parameter uncertainty with copulas

As stated above these processes make use of the methods implemented for *FLModelSim*. EXPAND... In the following example we'll use Gislason's second estimator (REF), $M_l = K(\frac{L_inf}{l})^1.5$.

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

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

# create object
mgis2 <- FLModelSim(model = ~K * (linf/len)^1.5, params = FLPar(linf = linf,
```

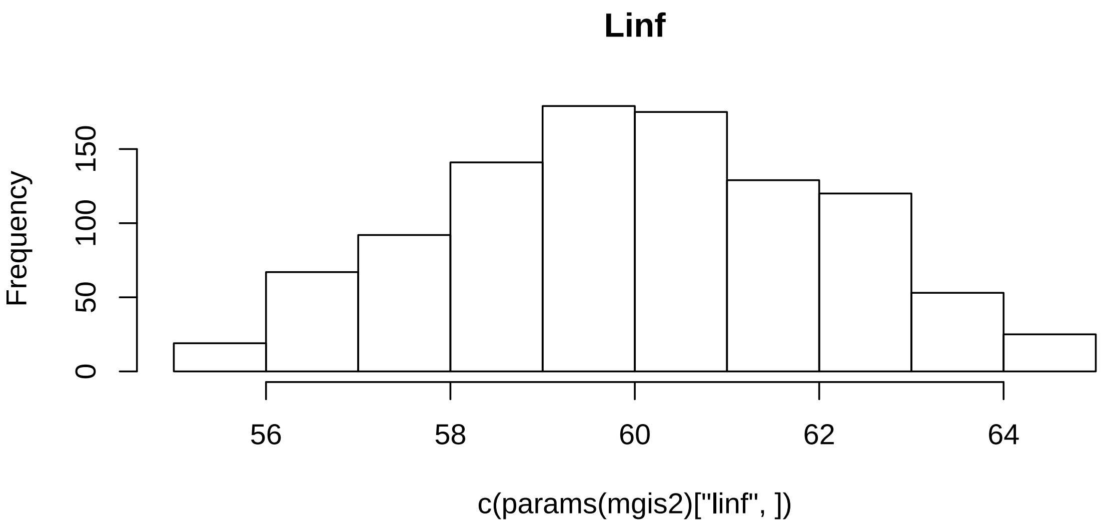
```
    K = k), vcov = mm)

pars <- list(list(55, 65), list(a = 0.3, b = 0.6, c = 0.35))
mgis2 <- mvrtriangle(1000, mgis2, paramMargins = pars)
```

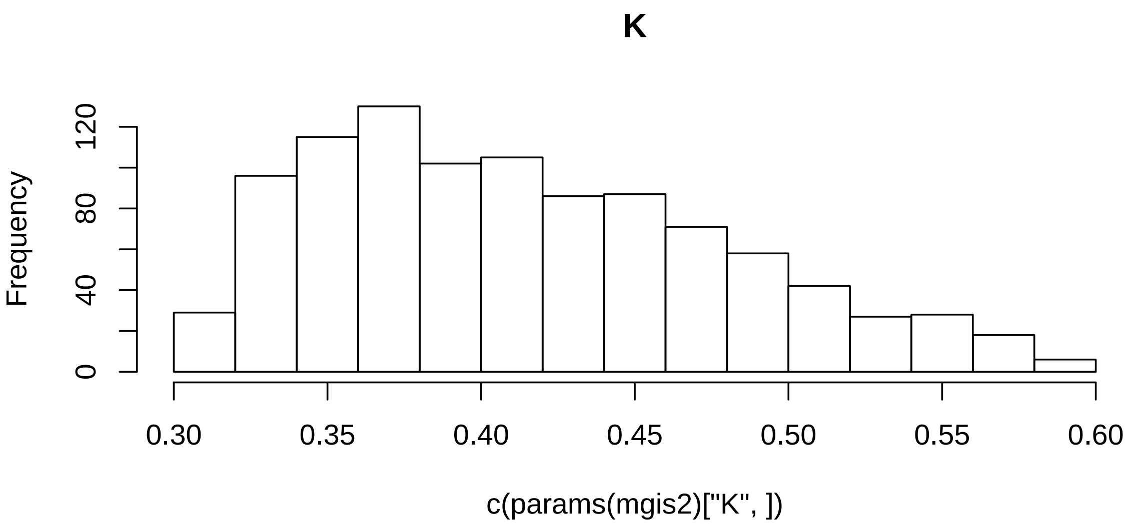
```
splom(t(params(mgis2)@.Data))
```



Scatter Plot Matrix

```
par(mfrow = c(2, 1))
hist(c(params(mgis2)["linf", ]), main = "Linf")
hist(c(params(mgis2)["K", ]), main = "K")
```





Use the constructor or the set method to add the new model. Note that we have a quite complex method now for *M*. A length based *shape* model from Gislason's work, Pauly's based temperature *level* and a time trend depending on NAO.

```
m5 <- a4aM(shape = mgis2, level = mod2, trend = mod3)
# or
m5 <- m4
level(m5) <- mgis2
```

4.4 The "m" method

The *m* method is the workhorse on computing natural mortality. The method returns a *FLQuant* that can be inserted in an *FLStock* for posterior usage by the assessment method. Note that if the models use *age* and/or *year* as terms, the method expects these to be included in the call (will be passed through the ... argument). If they're not, the method will use the range slot to work out the ages and/or years that should be predicted. If *age* and/or *year* are not model terms, the method will use the range slot to define the dimensions of the resulting *M* *FLQuant*.

```
# simple
m(m1)

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

# with ages
rngage(m1) <- c(0, 15)
m(m1)

## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## quant 0
##    0  0.2
##    1  0.2
##    2  0.2
##    3  0.2
##    4  0.2
##    5  0.2
##    6  0.2
##    7  0.2
##    8  0.2
##    9  0.2
##    10 0.2
##    11 0.2
##    12 0.2
##    13 0.2
##    14 0.2
##    15 0.2
##
## units:  NA

# with ages and years
rngyear(m1) <- c(2000, 2010)
m(m1)

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

```
##    8  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##    9  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##    10 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##    11 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##    12 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##    13 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##    14 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##    15 0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
##
## units:  NA
```

The next example as aage based shape. The information on the range of ages can be passed when calling *m*, or else the method will pick it up from the *range* slot. Note that in this case *mbar* becames relevant. It's the range of ages that is used to compute the mean level, which will match the *level* model.

```
# simple
m(m2)

## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## quant 0
##     0 0.6
##
## units:  NA

# with ages
rngage(m2) <- c(0, 15)
m(m2)

## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## quant 0
##    0  6.0000e-01
##    1  2.2073e-01
##    2  8.1201e-02
##    3  2.9872e-02
##    4  1.0989e-02
##    5  4.0428e-03
##    6  1.4873e-03
##    7  5.4713e-04
##    8  2.0128e-04
##    9  7.4046e-05
##    10 2.7240e-05
##    11 1.0021e-05
##    12 3.6865e-06
##    13 1.3562e-06
##    14 4.9892e-07
##    15 1.8354e-07
##
## units:  NA

# with ages and years
rngyear(m2) <- c(2000, 2003)
m(m2)
```

```
## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## quant 2000       2001       2002       2003
##    0  6.0000e-01 6.0000e-01 6.0000e-01 6.0000e-01
##    1  2.2073e-01 2.2073e-01 2.2073e-01 2.2073e-01
##    2  8.1201e-02 8.1201e-02 8.1201e-02 8.1201e-02
##    3  2.9872e-02 2.9872e-02 2.9872e-02 2.9872e-02
##    4  1.0989e-02 1.0989e-02 1.0989e-02 1.0989e-02
##    5  4.0428e-03 4.0428e-03 4.0428e-03 4.0428e-03
##    6  1.4873e-03 1.4873e-03 1.4873e-03 1.4873e-03
##    7  5.4713e-04 5.4713e-04 5.4713e-04 5.4713e-04
##    8  2.0128e-04 2.0128e-04 2.0128e-04 2.0128e-04
##    9  7.4046e-05 7.4046e-05 7.4046e-05 7.4046e-05
##    10 2.7240e-05 2.7240e-05 2.7240e-05 2.7240e-05
##    11 1.0021e-05 1.0021e-05 1.0021e-05 1.0021e-05
##    12 3.6865e-06 3.6865e-06 3.6865e-06 3.6865e-06
##    13 1.3562e-06 1.3562e-06 1.3562e-06 1.3562e-06
##    14 4.9892e-07 4.9892e-07 4.9892e-07 4.9892e-07
##    15 1.8354e-07 1.8354e-07 1.8354e-07 1.8354e-07
##
## units:  NA

# note that
predict(level(m2))

##    iter
##        1
##   1 0.6

# is similar to
m(m2)["0"]

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

# that's because mbar is '0'
rngmbar(m2)

## minmbar maxmbar
##       0       0

# changing ...
rngmbar(m2) <- c(0, 5)
quantMeans(m(m2)[as.character(0:5)])

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

```
##      year
## quant 2000 2001 2002 2003
##   all 0.6  0.6  0.6  0.6
##
## units:  NA
```

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

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

# with ages
rngage(m3) <- c(0, 15)
m(m3, nao = 0)

## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## quant 0
##    0  6.0000e-01
##    1  2.2073e-01
##    2  8.1201e-02
##    3  2.9872e-02
##    4  1.0989e-02
##    5  4.0428e-03
##    6  1.4873e-03
##    7  5.4713e-04
##    8  2.0128e-04
##    9  7.4046e-05
##    10 2.7240e-05
##    11 1.0021e-05
##    12 3.6865e-06
##    13 1.3562e-06
##    14 4.9892e-07
##    15 1.8354e-07
##
## units:  NA

# with ages and years
rngyear(m3) <- c(2000, 2003)
m(m3, nao = as.numeric(nao[, as.character(2000:2003)]))

## An object of class "FLQuant"
## , , unit = unique, season = all, area = unique
##
##      year
## quant 2000       2001       2002       2003
##    0  9.0000e-01 6.0000e-01 9.0000e-01 6.0000e-01
```

```
##    1  3.3109e-01 2.2073e-01 3.3109e-01 2.2073e-01
##    2  1.2180e-01 8.1201e-02 1.2180e-01 8.1201e-02
##    3  4.4808e-02 2.9872e-02 4.4808e-02 2.9872e-02
##    4  1.6484e-02 1.0989e-02 1.6484e-02 1.0989e-02
##    5  6.0642e-03 4.0428e-03 6.0642e-03 4.0428e-03
##    6  2.2309e-03 1.4873e-03 2.2309e-03 1.4873e-03
##    7  8.2069e-04 5.4713e-04 8.2069e-04 5.4713e-04
##    8  3.0192e-04 2.0128e-04 3.0192e-04 2.0128e-04
##    9  1.1107e-04 7.4046e-05 1.1107e-04 7.4046e-05
##    10 4.0860e-05 2.7240e-05 4.0860e-05 2.7240e-05
##    11 1.5032e-05 1.0021e-05 1.5032e-05 1.0021e-05
##    12 5.5298e-06 3.6865e-06 5.5298e-06 3.6865e-06
##    13 2.0343e-06 1.3562e-06 2.0343e-06 1.3562e-06
##    14 7.4838e-07 4.9892e-07 7.4838e-07 4.9892e-07
##    15 2.7531e-07 1.8354e-07 2.7531e-07 1.8354e-07
##
## units:  NA
```

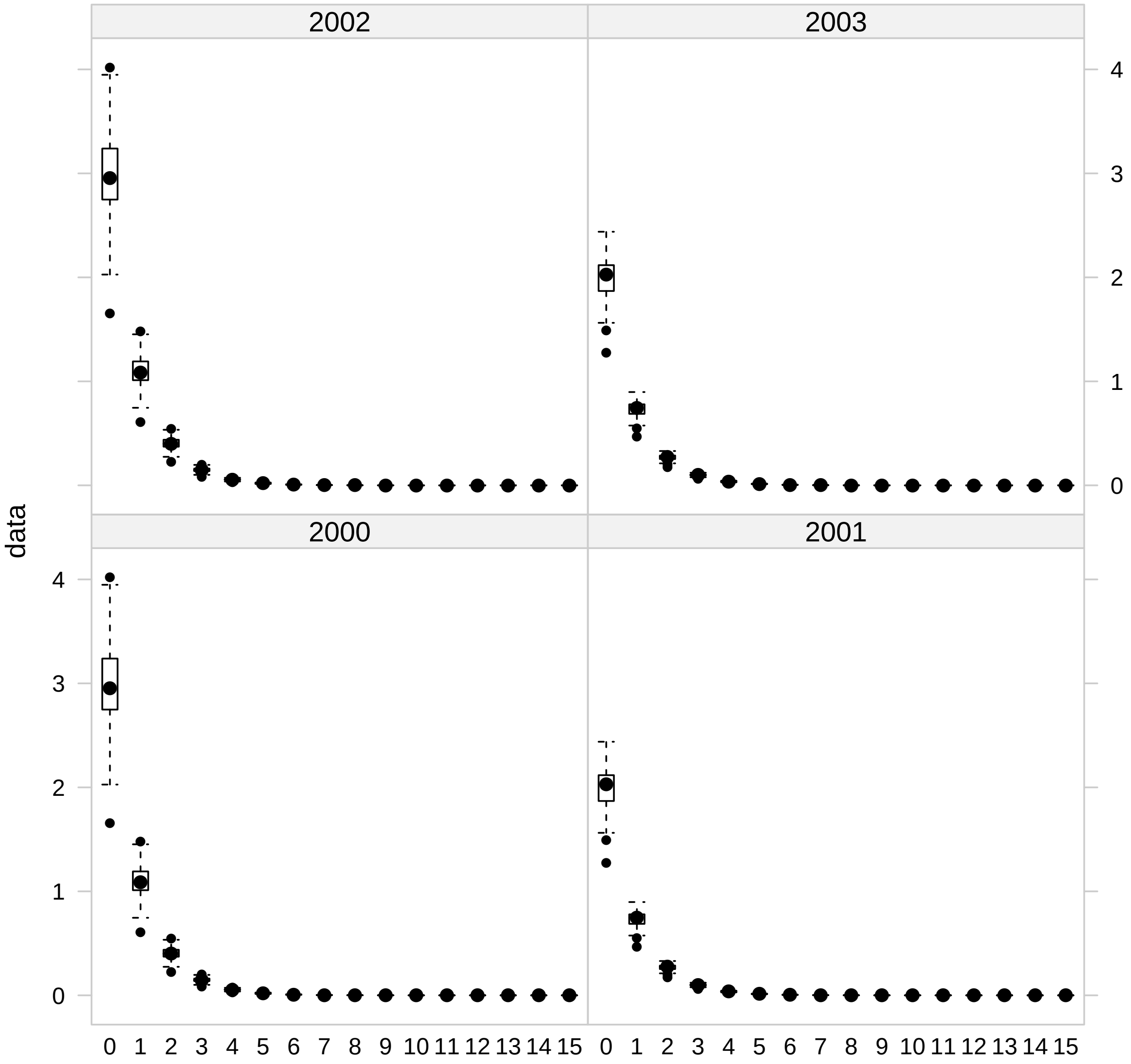
```
# simple
m(m4, nao = 1)

## An object of class "FLQuant"
## iters:  100
##
## , , unit = unique, season = all, area = unique
##
##      year
## quant 0
##     0 2.9551(0.371)
##
## units:  NA

# with ages
rngage(m4) <- c(0, 15)
m(m4, nao = 0)

## An object of class "FLQuant"
## iters:  100
##
## , , unit = unique, season = all, area = unique
##
##      year
## quant 0
##    0  2.0278e+00(1.68e-01)
##    1  7.4599e-01(6.18e-02)
##    2  2.7443e-01(2.27e-02)
##    3  1.0096e-01(8.36e-03)
##    4  3.7141e-02(3.08e-03)
##    5  1.3663e-02(1.13e-03)
##    6  5.0264e-03(4.16e-04)
##    7  1.8491e-03(1.53e-04)
##    8  6.8025e-04(5.63e-05)
##    9  2.5025e-04(2.07e-05)
##    10 9.2062e-05(7.63e-06)
##    11 3.3868e-05(2.81e-06)
##    12 1.2459e-05(1.03e-06)
```

```
##    13 4.5835e-06(3.80e-07)
##    14 1.6862e-06(1.40e-07)
##    15 6.2031e-07(5.14e-08)
##
## units:  NA
```

```
# with ages and years
rngyear(m4) <- c(2000, 2003)
m(m4, nao = as.numeric(nao[, as.character(2000:2003)]))
```

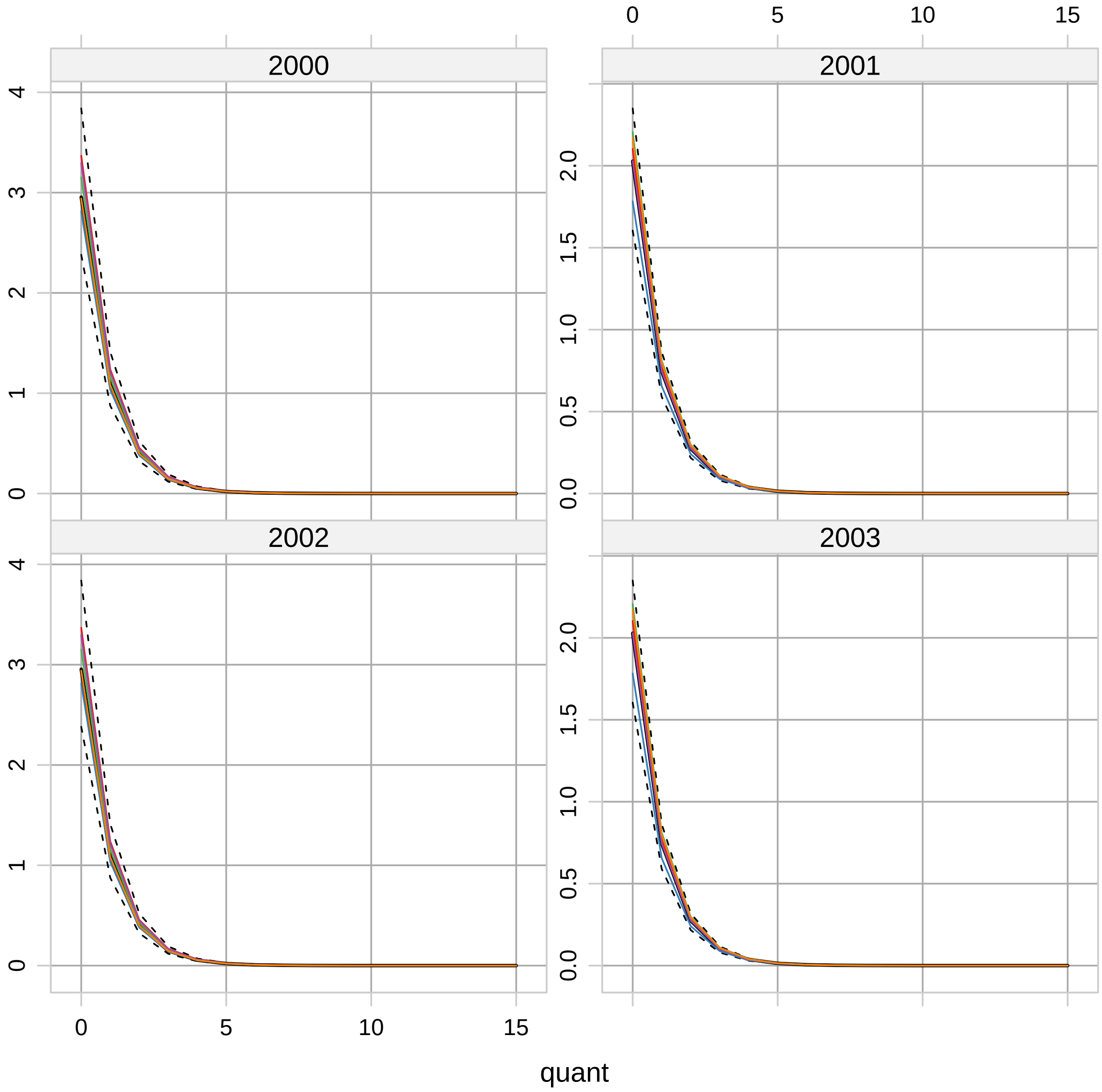
```
## An object of class "FLQuant"
## iters:  100
##
## , , unit = unique, season = all, area = unique
##
##      year
## quant 2000                 2001                 2002
##    0  2.9551e+00(3.71e-01) 2.0278e+00(1.68e-01) 2.9551e+00(3.71e-01)
##    1  1.0871e+00(1.37e-01) 7.4599e-01(6.18e-02) 1.0871e+00(1.37e-01)
##    2  3.9992e-01(5.02e-02) 2.7443e-01(2.27e-02) 3.9992e-01(5.02e-02)
##    3  1.4712e-01(1.85e-02) 1.0096e-01(8.36e-03) 1.4712e-01(1.85e-02)
##    4  5.4124e-02(6.80e-03) 3.7141e-02(3.08e-03) 5.4124e-02(6.80e-03)
##    5  1.9911e-02(2.50e-03) 1.3663e-02(1.13e-03) 1.9911e-02(2.50e-03)
##    6  7.3249e-03(9.20e-04) 5.0264e-03(4.16e-04) 7.3249e-03(9.20e-04)
##    7  2.6947e-03(3.38e-04) 1.8491e-03(1.53e-04) 2.6947e-03(3.38e-04)
##    8  9.9131e-04(1.24e-04) 6.8025e-04(5.63e-05) 9.9131e-04(1.24e-04)
##    9  3.6468e-04(4.58e-05) 2.5025e-04(2.07e-05) 3.6468e-04(4.58e-05)
##    10 1.3416e-04(1.68e-05) 9.2062e-05(7.63e-06) 1.3416e-04(1.68e-05)
##    11 4.9355e-05(6.20e-06) 3.3868e-05(2.81e-06) 4.9355e-05(6.20e-06)
##    12 1.8157e-05(2.28e-06) 1.2459e-05(1.03e-06) 1.8157e-05(2.28e-06)
##    13 6.6794e-06(8.39e-07) 4.5835e-06(3.80e-07) 6.6794e-06(8.39e-07)
##    14 2.4572e-06(3.09e-07) 1.6862e-06(1.40e-07) 2.4572e-06(3.09e-07)
##    15 9.0396e-07(1.14e-07) 6.2031e-07(5.14e-08) 9.0396e-07(1.14e-07)
##      year
## quant 2003
##    0  2.0278e+00(1.68e-01)
##    1  7.4599e-01(6.18e-02)
##    2  2.7443e-01(2.27e-02)
##    3  1.0096e-01(8.36e-03)
##    4  3.7141e-02(3.08e-03)
##    5  1.3663e-02(1.13e-03)
##    6  5.0264e-03(4.16e-04)
##    7  1.8491e-03(1.53e-04)
##    8  6.8025e-04(5.63e-05)
##    9  2.5025e-04(2.07e-05)
##    10 9.2062e-05(7.63e-06)
##    11 3.3868e-05(2.81e-06)
##    12 1.2459e-05(1.03e-06)
##    13 4.5835e-06(3.80e-07)
##    14 1.6862e-06(1.40e-07)
##    15 6.2031e-07(5.14e-08)
##
## units:  NA
```

```
bwplot(data ~ factor(quant) | year, data = m(m4, nao = as.numeric(nao[, as.character(2000:2003)])))
```



or this!

```
plotIters(m(m4, nao = as.numeric(nao[, as.character(2000:2003)])), by = "year")
```



5 Running assessments

There are two basic types of assessments available from using a4a: the management procedure (MP) fit and the full assessment fit. The MP fit does not compute estimates of covariances and is therefore quicker to execute, while the full assessment fit returns parameter estimates and their covariances and hence retains the ability to simulate from the model at the expense of longer fitting time.

5.1 a4aFit* - The fit classes

The basic model output is contained in the a4aFit class. This object contains only the fitted values.

```
showClass("a4aFit")

## Class "a4aFit" [package "FLa4a"]
##
## Slots:
##
```

```
## Name:       call     clock   fitSumm   stock.n   harvest   catch.n
## Class:      call   numeric     array   FLQuant   FLQuant   FLQuant
##
## Name:      index      name      desc     range
## Class:  FLQuants character character   numeric
##
## Extends: "FLComp"
##
## Known Subclasses:
## Class "a4aFitSA", directly
## Class "a4aFitMCMC", directly
## Class "a4aFitExt", by class "a4aFitSA", distance 2
```

Fitted values are stored in the `stock.n`, `harvest`, `catch.n` and `index` slots. It also contains information carried over from the stock object used to fit the model: the name of the stock in `name`, any description provided in `desc` and the age and year range and mean F range in `range`. There is also a wall clock that has a breakdown of the time taken o run the model.

The full assessment fit returns an object of `a4aFitSA` class:

```
showClass("a4aFitSA")

## Class "a4aFitSA" [package "FLa4a"]
##
## Slots:
##
## Name:       pars      call     clock   fitSumm   stock.n   harvest
## Class:   SCAPars      call   numeric     array   FLQuant   FLQuant
##
## Name:    catch.n     index      name      desc     range
## Class:   FLQuant  FLQuants character character   numeric
##
## Extends:
## Class "a4aFit", directly
## Class "FLComp", by class "a4aFit", distance 2
##
## Known Subclasses: "a4aFitExt"
```

The additional slots in the assessment output is the `fitSumm` and `pars` slots which are containers for model summaries and the model parameters. The `pars` slot is a class of type `SCAPars` which is itself composed of sub-classes, designed to contain the information necessary to simulate from the model.

```
showClass("SCAPars")

## Class "SCAPars" [package "FLa4a"]
##
## Slots:
##
## Name:      stkmodel       qmodel       vmodel
## Class: a4aStkParams    submodels    submodels

showClass("a4aStkParams")

## Class "a4aStkParams" [package "FLa4a"]
##
## Slots:
```

```
##
## Name:       fMod     n1Mod     srMod    params      vcov centering
## Class:   formula   formula   formula     FLPar     array   numeric
##
## Name:      distr      name      desc     range
## Class: character character character   numeric
##
## Extends: "FLComp"
```

for example, all the parameters required so simulate a time-series of mean F trends is contained in the `stkmodel` slot, which is a class of type `a4aStkParams`. This class contains the relevant submodels (see later), their parameters `params` and the joint covariance matrix `vcov` for all stock related parameters.

5.2 The submodels

In the `a4a` assessment model, the model structure is defined by submodels. These are models for the different parts of a statistical catch at age model that requires structural assumptions, such as the selectivity of the fishing fleet, or how F-at-age changes over time. It is advantageous to write the model for F-at-age and survey catchability as linear models (by working with log F and log Q) becuase it allows us to use the linear modelling tools available in R: see for example gam formulas, or factorial design formulas using lm. In R's linear modelling lanquage, a constant model is coded as ~ 1, while a slope over age would simply be $\sim$ age. Extending this we can write a traditional year / age seperable F model like $\sim$ factor(age) + factor(year).

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

```
fmodel <- ~factor(age) + factor(year)
qmodel <- list(~factor(age))
```

5.3 Run !!

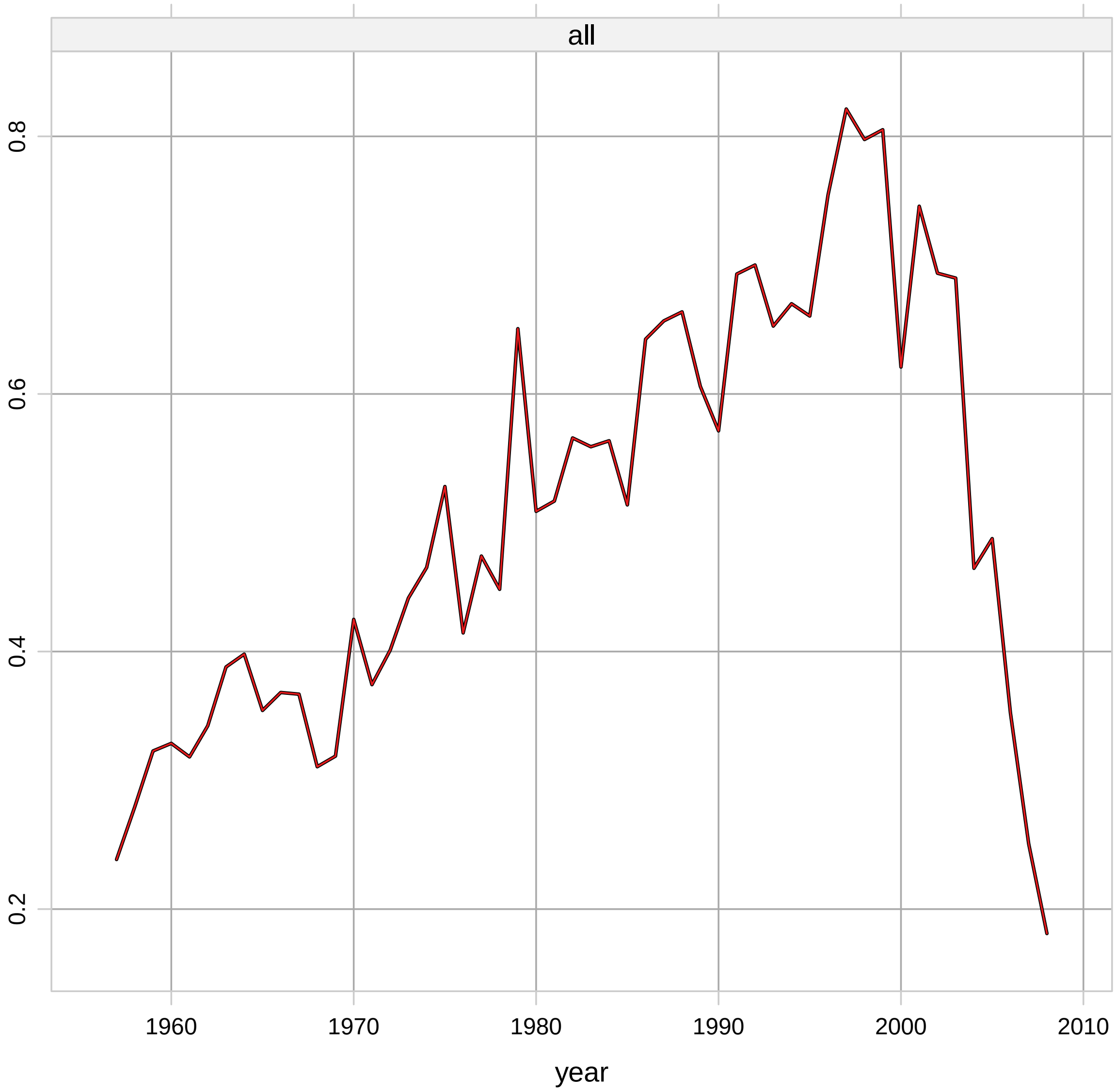
running the model is done by

```
fit <- a4a(fmodel, qmodel, stock = ple4, indices = ple4.indices[1])

## Note:  The following observations are treated as being missing at random:
##     fleet year age
## BTS-Isis 1997  1
## BTS-Isis 1997  2
##      Predictions will be made for missing observations.
```

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, and not because the observations were zero.

We can inspect the summaries from this fit my adding it to the origional stock object, for example to see the fitted fbar we can do

```
fitstk <- ple4 + fit
plotIters(fbar(fitstk))
```

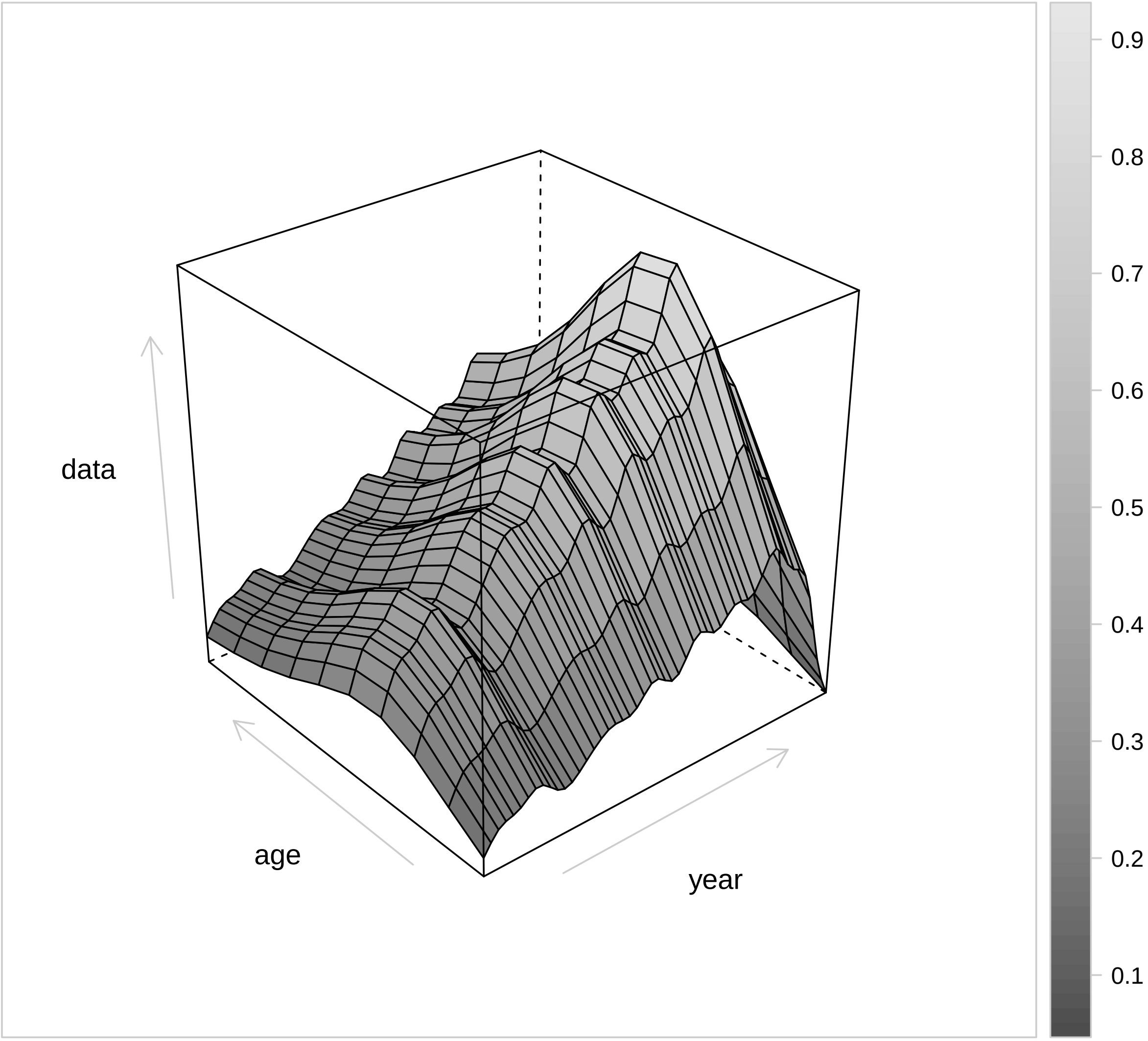


5.4 Some more examples

We will now take a look at some examples for F models and the forms that we can get. Lets start with a separable model in which we model selectivity at age as an (unpenalised) thin plate spline. We will use the North Sea Plaice data again, and since this 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.

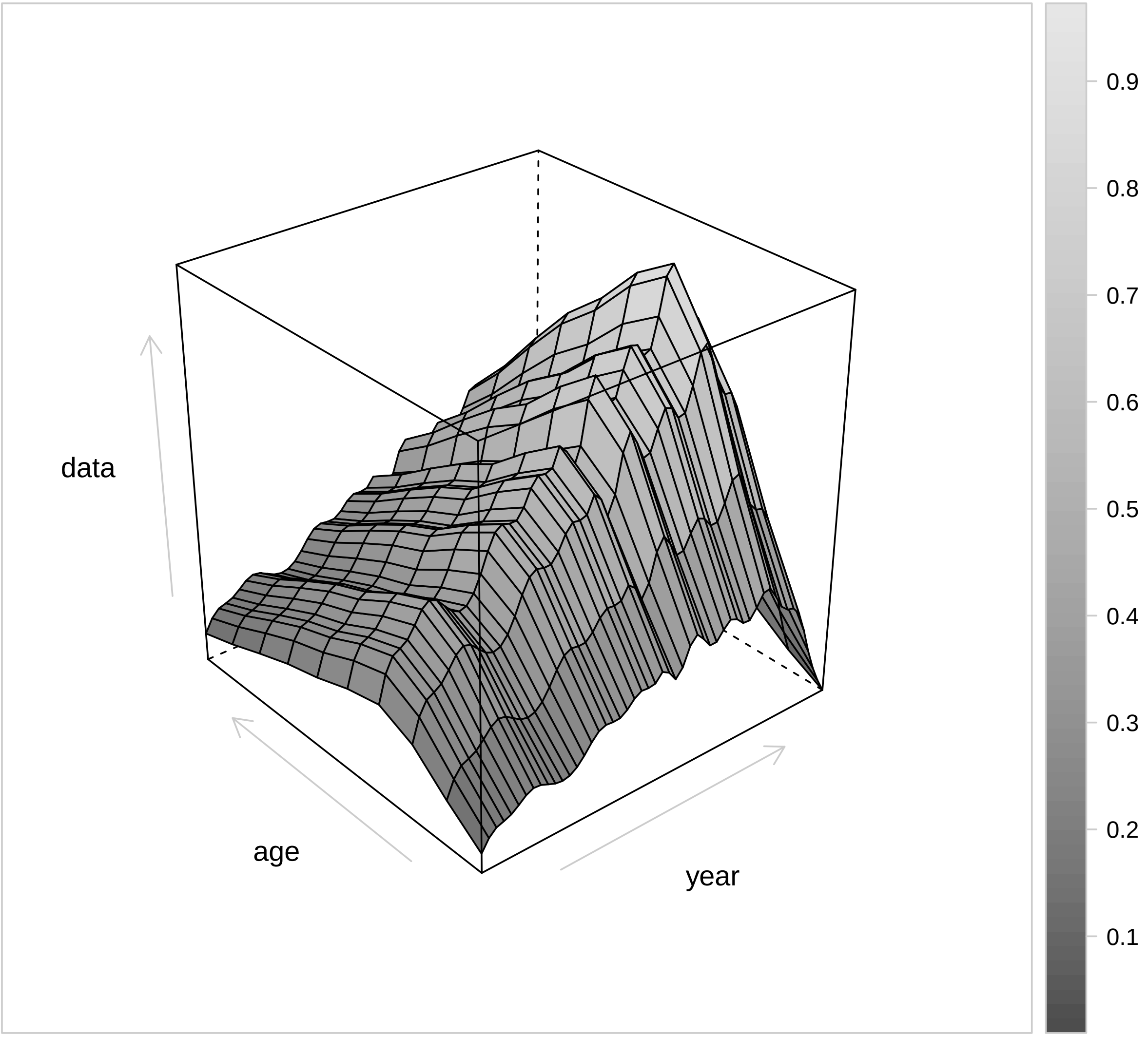
```
fmodel <- ~s(age, k = 4) + s(year, k = 20)
qmodel <- list(~factor(age))
fit1 <- a4a(fmodel, qmodel, stock = ple4, indices = ple4.indices[1])

## Note:  The following observations are treated as being missing at random:
##     fleet year age
## BTS-Isis 1997  1
## BTS-Isis 1997  2
##      Predictions will be made for missing observations.
```

```
wireframe(data ~ year + age, data = as.data.frame(harvest(fit1)), drape = TRUE)
```



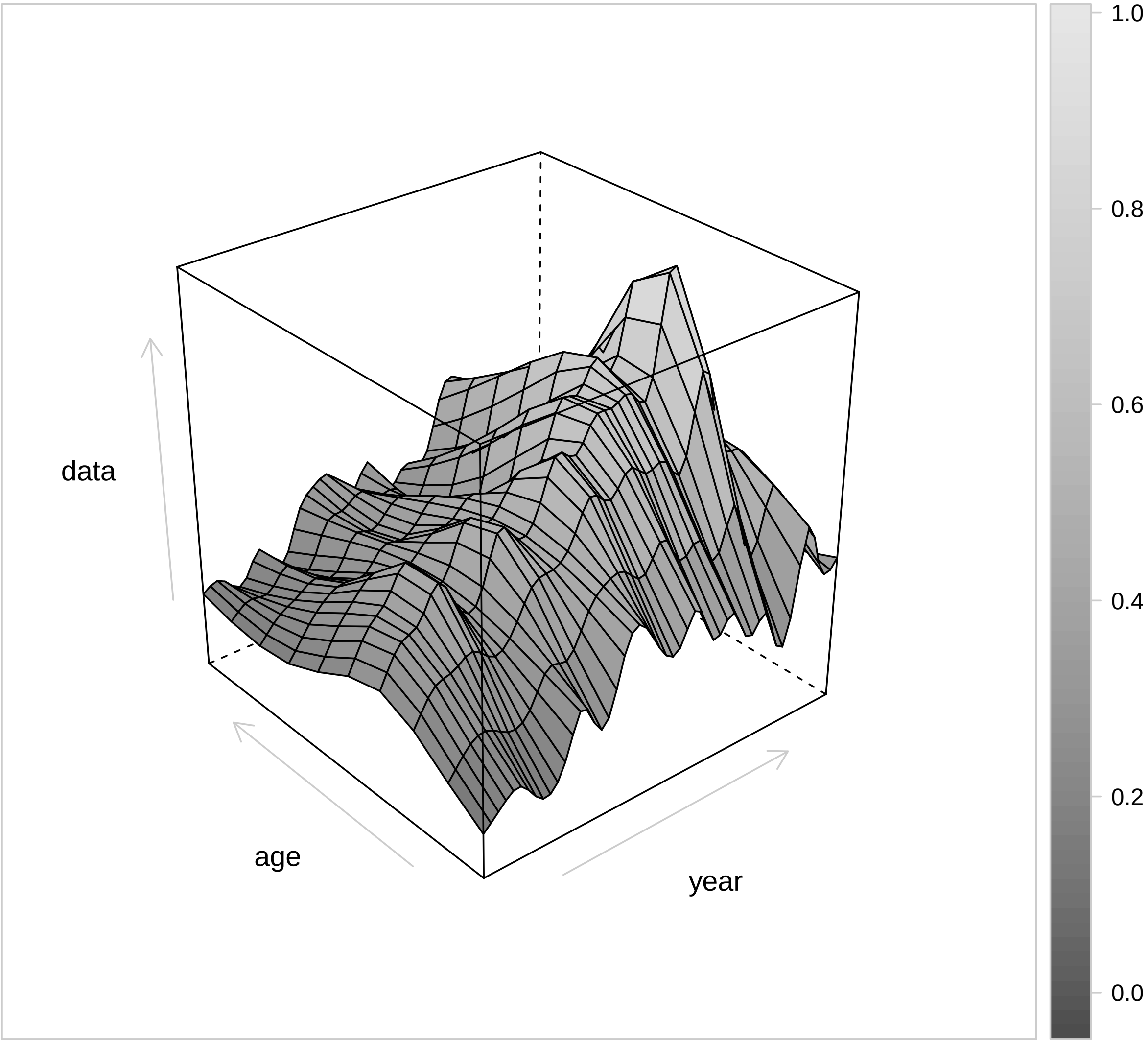
Lets now investigate some variations in the selectivity shape with time, but only a little... we can do this by adding a smooth interaction term in the fmodel

```
fmodel <- ~s(age, k = 4) + s(year, k = 20) + te(age, year, k = c(3, 3))
qmodel <- list(~factor(age))
fit2 <- a4a(fmodel, qmodel, stock = ple4, indices = ple4.indices[1])

## Note:  The following observations are treated as being missing at random:
##      fleet year age
## BTS-Isis 1997  1
## BTS-Isis 1997  2
##       Predictions will be made for missing observations.

wireframe(data ~ year + age, data = as.data.frame(harvest(fit2)), drape = TRUE)
```



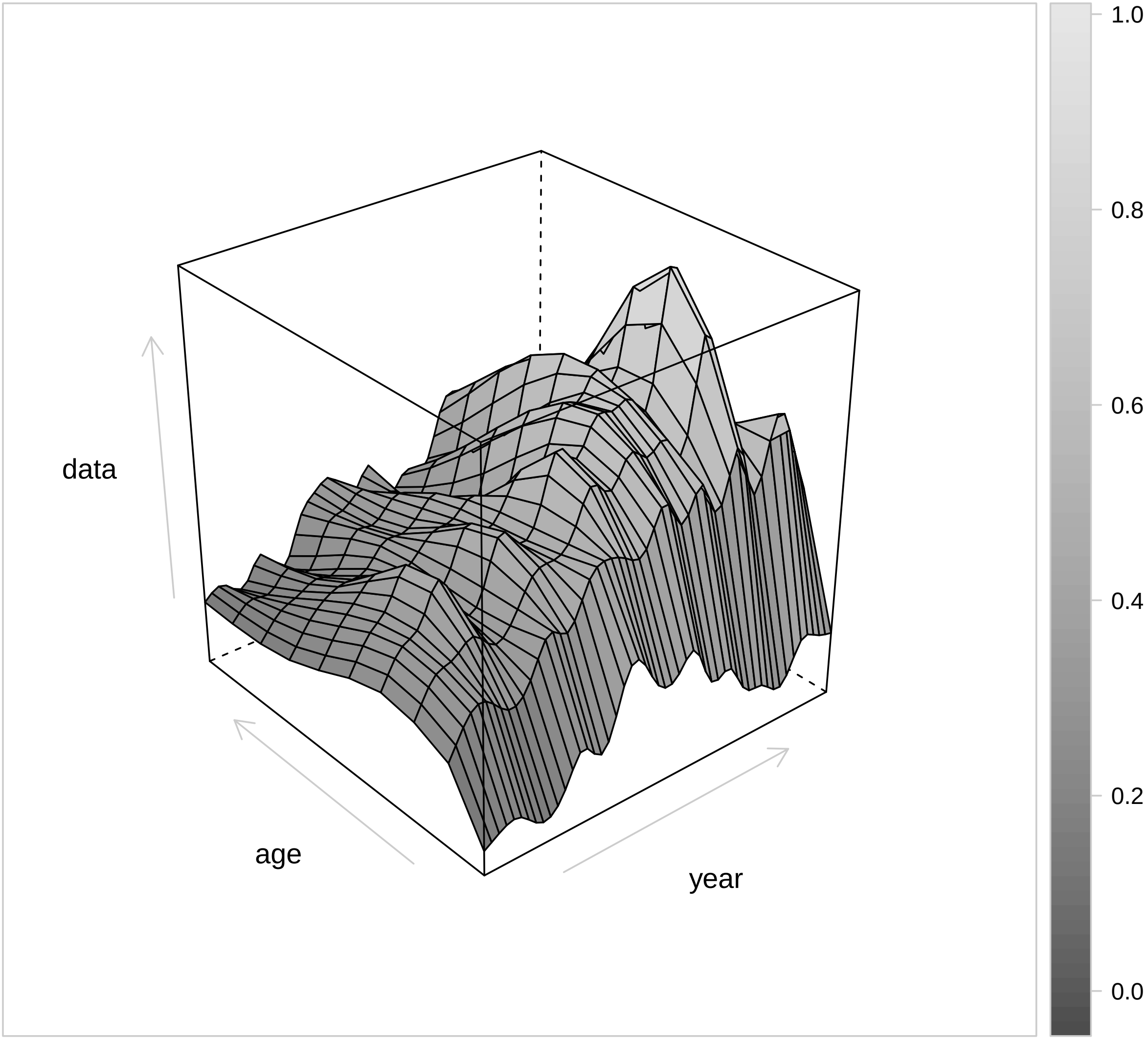
A further move is to free up the Fs to vary more over time

```
fmodel <- ~te(age, year, k = c(4, 20))
qmodel <- list(~factor(age))
fit2 <- a4a(fmodel, qmodel, stock = ple4, indices = ple4.indices[1])

## Note:  The following observations are treated as being missing at random:
##     fleet year age
## BTS-Isis 1997  1
## BTS-Isis 1997  2
##      Predictions will be made for missing observations.

wireframe(data ~ year + age, data = as.data.frame(harvest(fit2)), drape = TRUE)
```



In the last examples the 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 juvenilles 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 seperate model. This can be achieved by

```
fmodel <- ~te(age, year, k = c(4, 20)) + s(year, k = 5, by = as.numeric(age ==
    1))
qmodel <- list(~factor(age))
fit3 <- a4a(fmodel, qmodel, stock = ple4, indices = ple4.indices[1])

## Note:  The following observations are treated as being missing at random:
##      fleet year age
## BTS-Isis 1997  1
## BTS-Isis 1997  2
##       Predictions will be made for missing observations.

wireframe(data ~ year + age, data = as.data.frame(harvest(fit3)), drape = TRUE)
```



Please note that each of these model *structures* lets say, 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 AIC or BIC.